COMPUTER SOFTWARE APPLICATION

TRADE PRACTICAL

NSQF LEVEL - 4

VOLUME - 2

HANDBOOK FOR CRAFTS INSTRUCTOR TRAINING SCHEME

Directorate General of Training

DIRECTORATE GENERAL OF TRAINING
MINISTRY OF SKILL DEVELOPMENT & ENTREPRENEURSHIP
GOVERNMENT OF INDIA

NATIONAL INSTRUCTIONAL MEDIA INSTITUTE, CHENNAI

Post Box No. 3142, CTI Campus, Guindy, Chennai - 600 032

Published by

National Instructional Media Institute
Post.Box.No. 3142,
Guindy, Chennai - 600032
Email : chennai-nimi@nic.in
Website: www.nimi.gov.in

All Rights Reserved

First Edition, 2024

Rs. 360/-

Printed in India at
National Instructional Media Institute
Post. Box. No. 3142,
Guindy,
Chennai - 600032

A Comprehensive Training Program
under Crafts Instructor Training Scheme (CITS)
for Instructors

HANDBOOK ON TECHNICAL INSTRUCTOR TRAINING MODULES

अतुल कुमार तिवारी, I.A.S.
सचिव

ATUL KUMAR TIWARI, I.A.S.
Secretary

सत्यमेव जयते

भारत सरकार
कौशल विकास एवं उद्यमिता मंत्रालय
GOVERNMENT OF INDIA
MINISTRY OF SKILL DEVELOPMENT
AND ENTREPRENEURSHIP

Foreword

In today's rapidly evolving world, the role of skilled craftsmen and women is more crucial than ever. The Craft Instructor Training Scheme (CITS) stands at the forefront of this transformation, shaping the educators who will train the next generation of artisans and technicians. This book aims to provide an in-depth understanding of the subject, exploring its significance, methodologies, and impact on vocational training.

The Craft Instructor Training Scheme was established with the objective of enhancing the quality of instruction in industrial training institutes and other vocational training institutions. By equipping instructors with advanced skills and knowledge, the scheme ensures that they are well-prepared to impart high-quality training to their students. This, in turn, contributes to the creation of a highly skilled workforce capable of meeting the demands of modern industry.

The initial chapters provide the importance of specialized instructor training. Following this, detailed chapters delve into the curriculum covering advanced techniques, safety protocols, and instructional strategies. Each section is designed to offer both theoretical insights and practical applications, ensuring a well-rounded understanding of the subject.

The book offers recommendations for overcoming obstacles and enhancing the effectiveness of the program, with the ultimate goal of producing highly skilled instructors capable of shaping the future workforce.

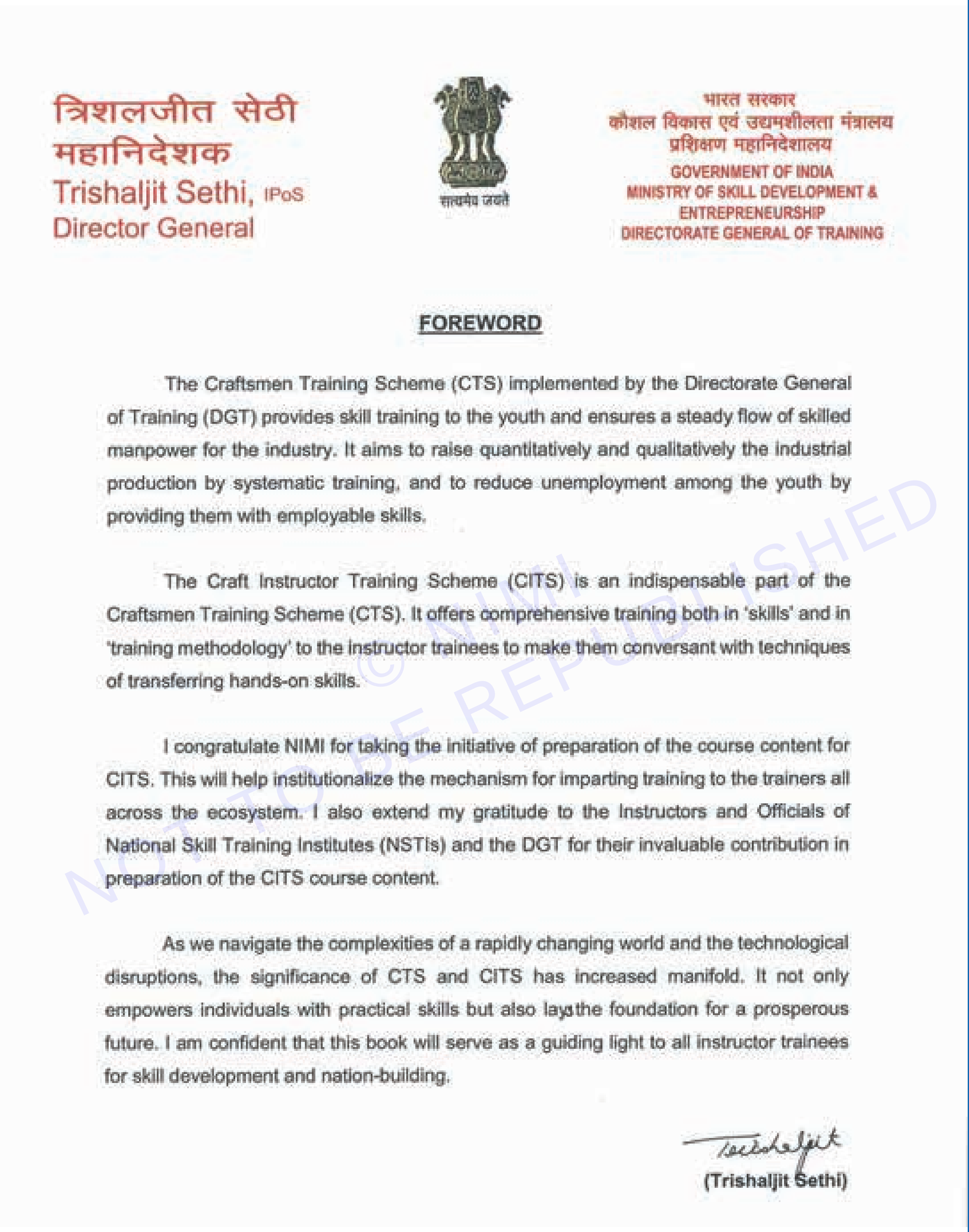
This book is intended for a diverse audience, including current and aspiring instructors, vocational training administrators, policymakers, and industry stakeholders. It serves as a valuable resource for understanding the intricacies of the subject and its pivotal role in vocational education.

I extend my heartfelt gratitude to all contributors who have shared their experiences and expertise, enriching this book with their valuable insights. Special thanks to the contribution of the development team, reviewers and NIMI that have supported this endeavor, providing essential data and resources.

It is my sincere hope that this book will inspire and guide readers in their efforts to enhance vocational training, ultimately contributing to the development of a skilled and competent workforce.

[illegible]

ATUL KUMAR TIWARI, I.A.S.
Secretary, MSDE



त्रिशलजीत सेठी
महानिदेशक
Trishaljit Sethi, IPoS
Director General

सत्यमेव जयते

भारत सरकार
कौशल विकास एवं उद्यमशीलता मंत्रालय
प्रशिक्षण महानिदेशालय
GOVERNMENT OF INDIA
MINISTRY OF SKILL DEVELOPMENT & ENTREPRENEURSHIP
DIRECTORATE GENERAL OF TRAINING

FOREWORD

The Craftsmen Training Scheme (CTS) implemented by the Directorate General of Training (DGT) provides skill training to the youth and ensures a steady flow of skilled manpower for the industry. It aims to raise quantitatively and qualitatively the industrial production by systematic training, and to reduce unemployment among the youth by providing them with employable skills.

The Craft Instructor Training Scheme (CITS) is an indispensable part of the Craftsmen Training Scheme (CTS). It offers comprehensive training both in 'skills' and in 'training methodology' to the instructor trainees to make them conversant with techniques of transferring hands-on skills.

I congratulate NIMI for taking the initiative of preparation of the course content for CITS. This will help institutionalize the mechanism for imparting training to the trainers all across the ecosystem. I also extend my gratitude to the Instructors and Officials of National Skill Training Institutes (NSTIs) and the DGT for their invaluable contribution in preparation of the CITS course content.

As we navigate the complexities of a rapidly changing world and the technological disruptions, the significance of CTS and CITS has increased manifold. It not only empowers individuals with practical skills but also lays the foundation for a prosperous future. I am confident that this book will serve as a guiding light to all instructor trainees for skill development and nation-building.

[illegible]
(Trishaljit Sethi)

PREFACE

The Craft Instructor Training Scheme is an indispensable module of the Craftsmen Training Scheme, which has been an integral part of the Indian skill development industry since its inception. This program aims to equip instructors with the necessary skills and teaching methodology to effectively transfer hands-on skills to trainees and promote a holistic learning experience. The first Craft Instructor Training Institute was established in 1948, followed by six more institutes across India in 1960. Today, these institutes, including the National Skill Training Institute (formerly Central Training Institute for Instructors), offer the CITS course, which is mandated by the Directorate General of Training (DGT).

The Craft Instructor training program is designed to develop skilled manpower for industries. The course aims to offer instructors an opportunity to improve their instructional skills, engage learners effectively, offer impactful mentoring, and make efficient use of resources, leading to a more skilled workforce in various industries. The program emphasizes collaborative and innovative approaches to teaching, resulting in high-quality course delivery. Overall, the Craft Instructor Training Scheme is a pivotal program that helps instructors grow in their careers and make a significant contribution to society. This program is essential for developing skilled manpower and promoting a robust learning environment that benefits both trainees and instructors alike.

ACKNOWLEDGEMENT

National Instructional Media Institute (NIMI) sincerely acknowledges with thanks for the co-operation and contribution extended by the following experts to bring out this Instructional material **(Trade Practical)** for **CITS Computer Software Application (Volume - II of II) (NSQF Level - 4)** under the **IT & ITES** Sector for **Instructors**.

MEDIA DEVELOPMENT COMMITTEE MEMBERS

Smt. P.A. Sulabha	-	Training Officer, NSTI (W), Trivandrum, Kerala.
Shri. M.K. Sunil	-	Training Officer, NSTI, Chennai.

COORDINATORS

Shri. G.C. Ramamurthy	-	Joint Director, CD - Section, DGT.
Shri. T.V. Rajasekar	-	Joint Director, NIMI, Chennai.
Shri. Shiv Kumar	-	Training Officer, CD - Section, DGT.

NIMI records its appreciation of the Data Entry, CAD, DTP Operators for their excellent and devoted services in the process of development of this Instructional Material.

NIMI also acknowledges with thanks, the invaluable efforts rendered by all other staff who have contributed for the development of this Instructional Material.

NIMI is grateful to all others who have directly or indirectly helped in developing this IMP.

ABOUT THE TEXT BOOK

The Vocational Instructor Training Program is a comprehensive initiative designed to equip aspiring students with the necessary skills and knowledge to effectively teach in vocational education settings. This program encompasses a range of pedagogical strategies, instructional techniques, and subject-specific content tailored to the diverse vocational fields. Participants engage in coursework that covers curriculum development, assessment methods, classroom management, and the integration of industry-relevant technologies. Practical experience and hands-on training are emphasized, allowing participants to apply theoretical concepts in real-world teaching environments. Through collaborative learning experiences and mentorship opportunities, aspiring vocational instructors develop the confidence and competence to facilitate engaging and impactful learning experiences for their students. This training program aims to cultivate a new generation of educators who are not only proficient in their respective vocational fields but also adept at fostering the success and employability of their students in today's competitive workforce.

This text book covers communication, self-management, information and communication technology, entrepreneurial and green skills. It has been developed as per the learning outcome-based curriculum.

G C Rama Murthy,
Joint Director,
Curriculum Development, DGT,
MSDE, New Delhi.

CONTENT

Ex. No.	Table of Contents	Page No.
110	Create abstract classes and methods.	163
111	Create interfaces in JAVA.	168
112	Override methods in JAVA.	173
113	Create and implement an interface.	177
114	Extend interfaces in JAVA.	180
115	Create and use a package in JAVA.	184
116	Create a simple container using Frame class and extending another Frame class.	188
117	Create a container with a few controls.	198
118	Create a container with controls with action listeners and event handlers.	204
119	Create a GUI to draw different plane shapes over a predefined area.	209
	Module 7: Programming Language (Python)	
120	Install, set up the environment & run Python.	217
121	Use Command Line and IDE to create and execute a python program.	221
122	Write and test a python program to demonstrate print statement, comments, different types of variables.	226
123	Write and test a python program to perform data and data type operations, string operations, date, input and output, output formatting and operators.	230
124	Determine the sequence of execution based on operator precedence.	236
125	Construct and analyze code segments that use branching statements.	239
126	Construct and analyze code segments that perform iteration.	245
127	Document code segments using comments and documentation strings.	251
128	Write program in python using list, tuples, dictionaries and files.	254
129	Write a python program depicting argument passing and using tuples, dictionaries arguments.	267
130	Write a python program for importing a module.	270
131	Use exception handling in python program.	273
132	Write a python program to use built in functions i.e. chr, cmp, compile, dir, eval, filter, hash, input, len, locals, long, max, pow, range, slice, tuple, Unicode, vars.	277
133	Write a python program to read and write into a file.	286
134	Write a python program depicting argument passing and using tuples, dictionaries as arguments.	290
135	Construct and analyze code segments that include List comprehensions, tuple, set and Dictionary comprehensions.	292
136	Perform basic operations using built-in modules.	297
137	Solve complex computing problems by using built-in modules.	301

Module 6 : Object Oriented Programming and JAVA Language

EXERCISE 78 : Installing JAVA

Objectives

At the end of this exercise you shall be able to

- download JDK Software
- install JDK.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

First ensure that you have Java Development Kit (JDK) installed on your machine. You can download the latest JDK from the official Oracle website or use an open-source distribution like Open JDK. For example,

Installing JDK 21.0.2 on a Windows system:

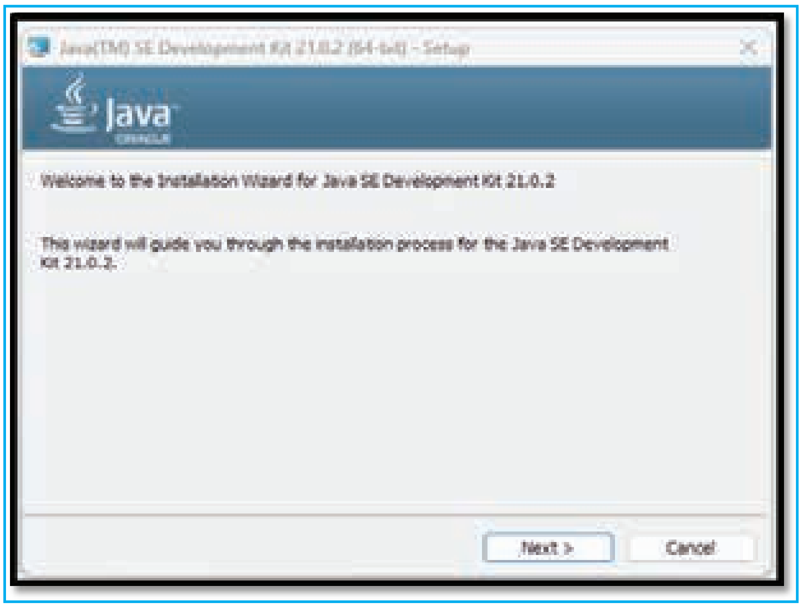
1 **Download JDK 21.0.2**

- Go to the Oracle JDK download page or an official source where JDK 21.0.2 is available for download.
- Accept the license agreement and choose the appropriate JDK package for your Windows system. Make sure to select the correct platform (e.g., Windows x64).

2 **Run the Installer**

- Once the download completes, locate the downloaded JDK installer file (e.g., jdk-21.0.2_windows-x64_bin.exe).
- Double-click on the installer file to run it.

3 **Start JDK Installation**

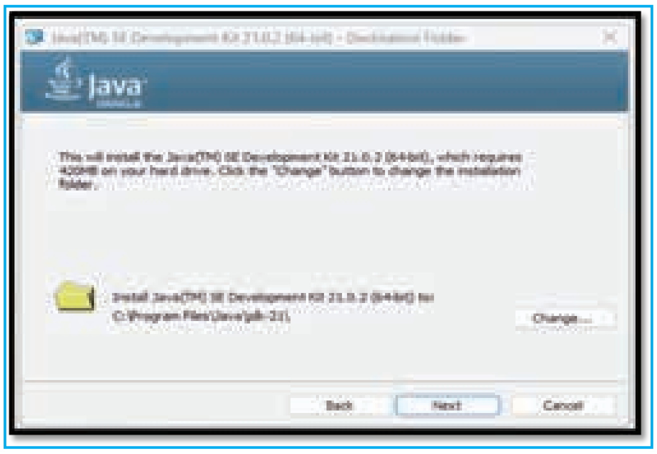

Java(TM) SE Development Kit 21.0.2 (64-bit) - Setup

Welcome to the Installation Wizard for Java SE Development Kit 21.0.2

This wizard will guide you through the installation process for the Java SE Development Kit 21.0.2.

- The installer will launch. Click "Next" to begin the installation process.

4 Choose Installation Location:



Java(TM) SE Development Kit 21.0.2 (64-bit) - Destination Folder

This will install the Java(TM) SE Development Kit 21.0.2 (64-bit), which requires 420MB on your hard drive. Click the "Change" button to change the installation folder.

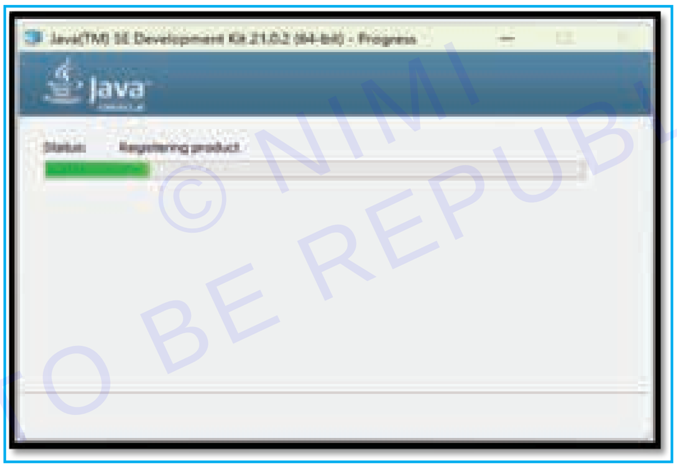
Install Java(TM) SE Development Kit 21.0.2 (64-bit) to:
C:\Program Files\Java\jdk-21\

- Choose the directory where you want to install JDK 21.0.2. The default location is typically C:\Program Files\Java\jdk-21.0.2.

5 Install JDK:

- Click "Next" to proceed with the installation.

6 Installation Progress

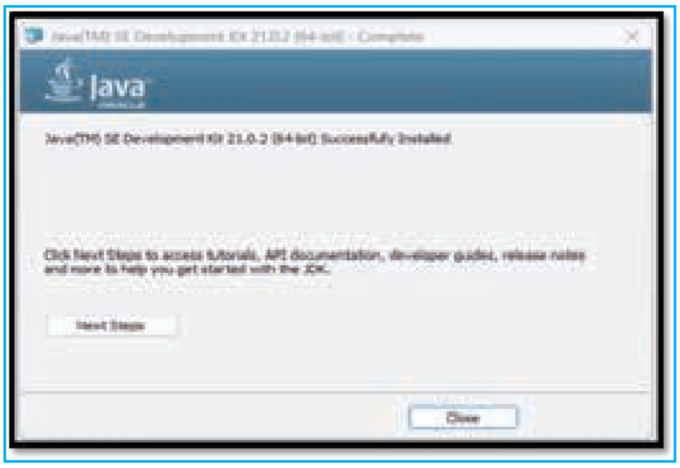


Java(TM) SE Development Kit 21.0.2 (64-bit) - Progress

Status: Registering product

- The installer will start copying files and installing JDK components. This process may take a few minutes depending on your system's speed.

7 Completing the JDK Installation:



Java(TM) SE Development Kit 21.0.2 (64-bit) - Complete

Java(TM) SE Development Kit 21.0.2 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

- Once the installation completes, you will see a confirmation screen. Click "Close" to exit the installer.

EXERCISE 79 : Setting the Class Path

Objectives

At the end of this exercise you shall be able to
- environment Variable/ Path Setting in Java
- system Variable Setting in Java
- verify JDK installation through Command Prompt
- check the version of Java installed in your system.

Requirements

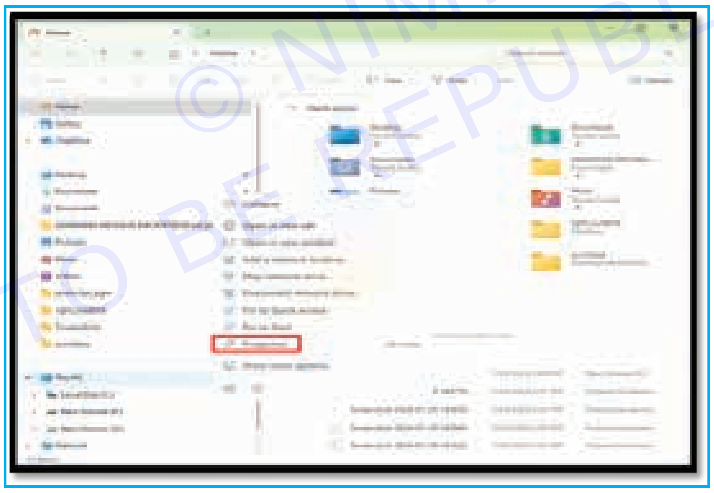
Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad).

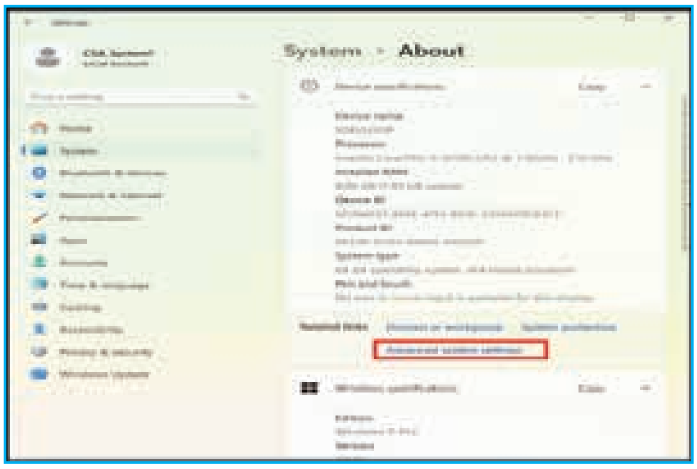
Procedure

Step1: Set JAVA_HOME Environment Variable

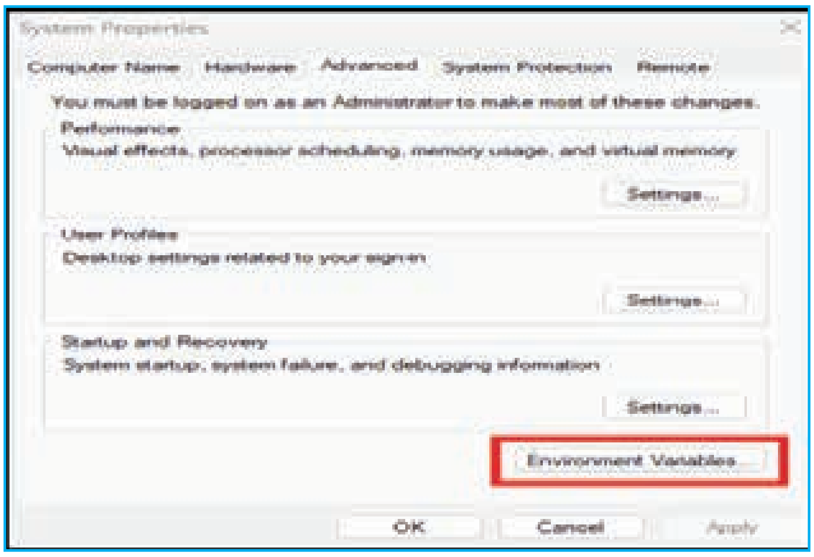
After installing JDK 21.0.2, you may want to set the JAVA_HOME environment variable to point to the JDK installation directory. This step is optional but can be useful for some development tools and applications.



- Right-click on “This PC” or “Computer” and select “Properties.



- Click on “Advanced system settings” on the left.



System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

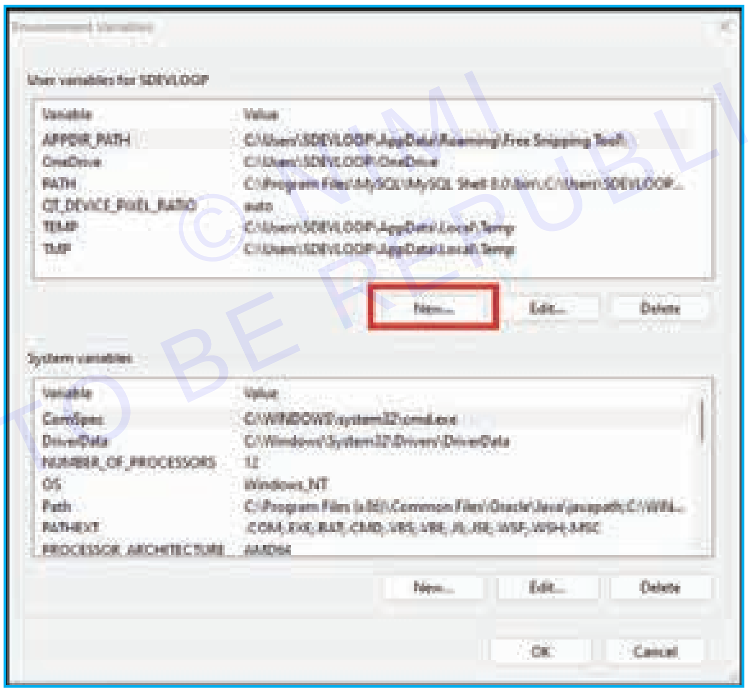
User Profiles
Desktop settings related to your sign-in
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK | Cancel | Apply

- Click the “Environment Variables” button.
- Set User Variable, click the new button.



Environment Variables

User variables for SDEVLOOP

Variable	Value
APPDIR_PATH	C:\Users\SDEVLOOP\AppData\Roaming\Free Snipping Tool\
OneDrive	C:\Users\SDEVLOOP\OneDrive
PATH	C:\Program Files\MySQL\MySQL Shell 8.0\bin\;C:\Users\SDEVLOOP...
QT_DEVICE_PIXEL_RATIO	auto
TEMP	C:\Users\SDEVLOOP\AppData\Local\Temp
TMP	C:\Users\SDEVLOOP\AppData\Local\Temp

New... | Edit... | Delete

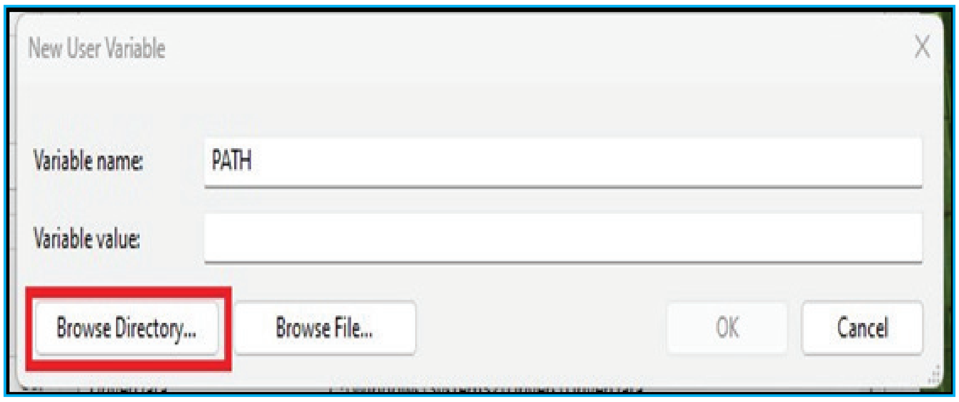
System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	12
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\WIN...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

New... | Edit... | Delete

OK | Cancel

- Set the path as shown below.

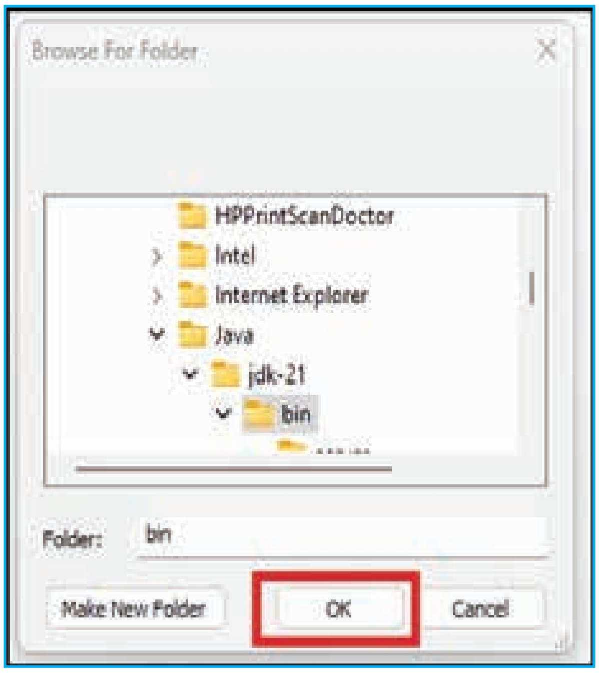


New User Variable

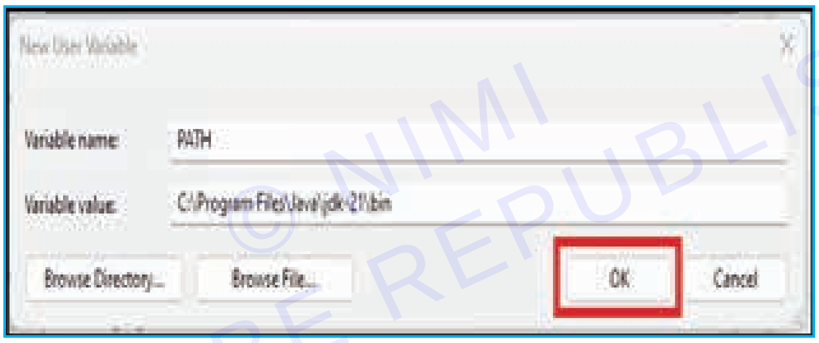
Variable name: PATH

Variable value:

Browse Directory... | Browse File... | OK | Cancel

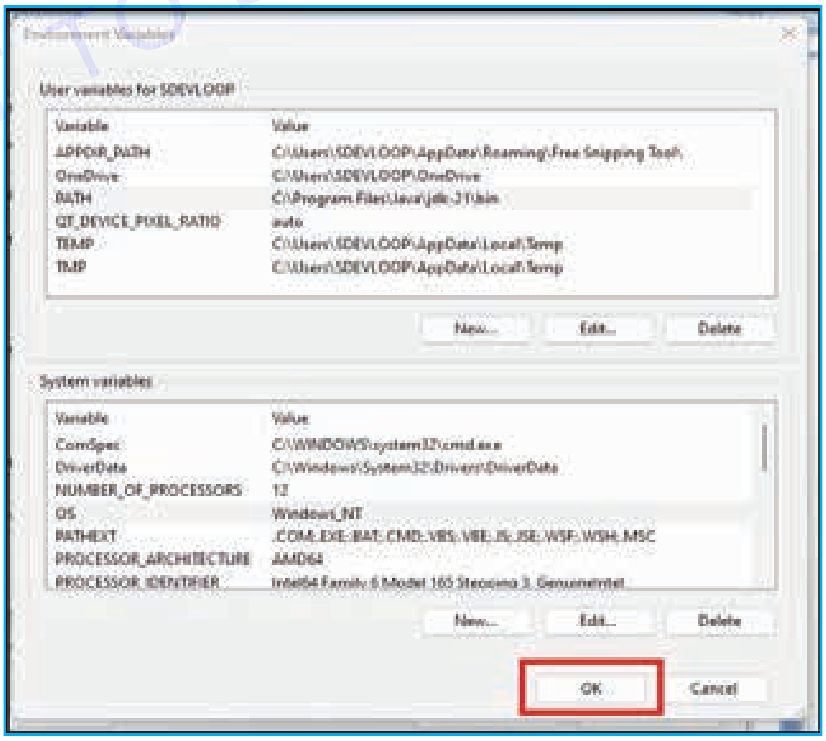


Folder: bin



Variable name: PATH

Variable value: C:\Program Files\Java\jdk-21\bin



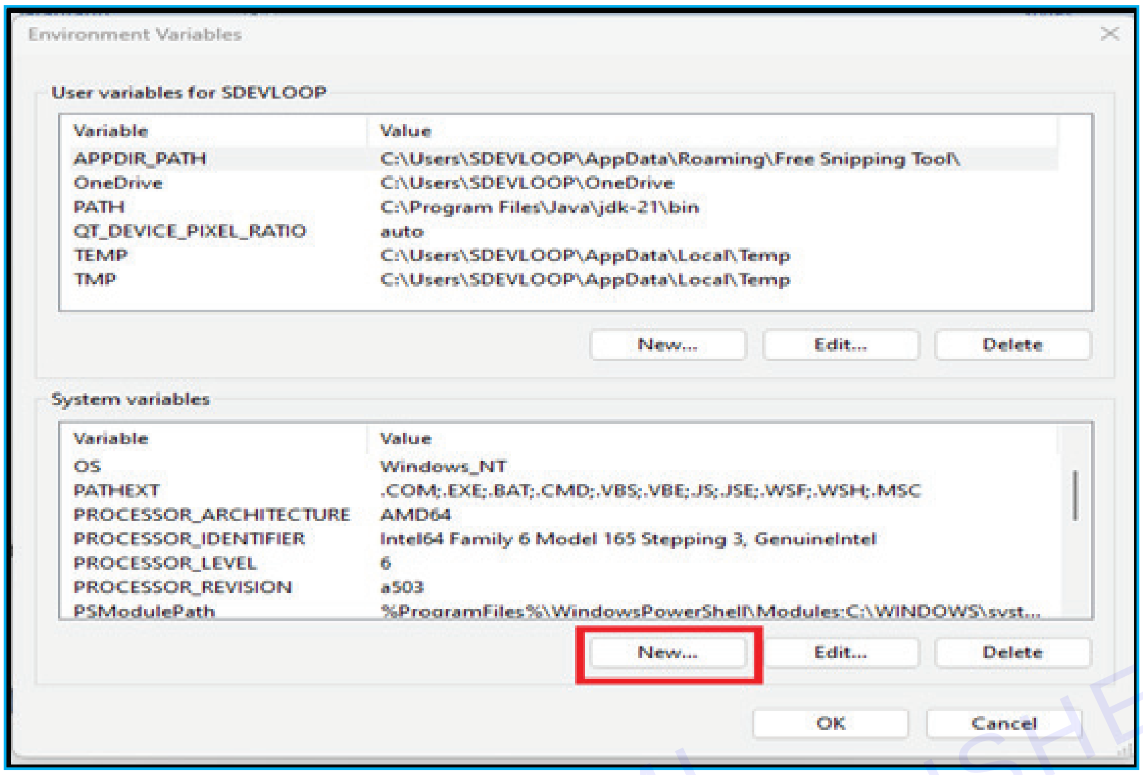
User variables for SDEVLOOP

Variable	Value
APPDIR_PATH	C:\Users\SDEVLOOP\AppData\Roaming\Free Snipping Tool\
OneDrive	C:\Users\SDEVLOOP\OneDrive
PATH	C:\Program Files\Java\jdk-21\bin
QT_DEVICE_PIXEL_RATIO	auto
TEMP	C:\Users\SDEVLOOP\AppData\Local\Temp
TMP	C:\Users\SDEVLOOP\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	12
OS	Windows_NT
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 165 Stepping 3, GenuineIntel

- To Set the system variable click on new button under the 'system variables'.



Environment Variables

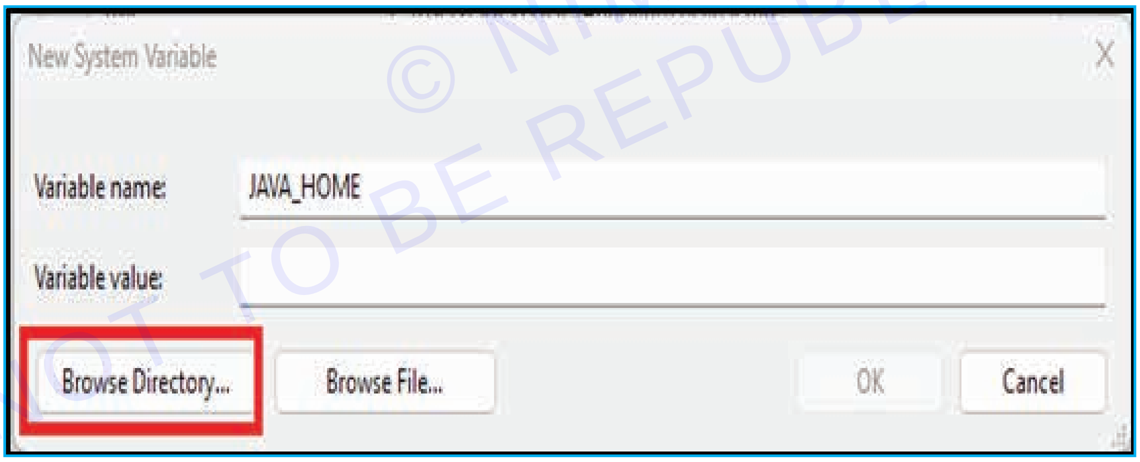
User variables for SDEVLOOP

Variable	Value
APPDIR_PATH	C:\Users\SDEVLOOP\AppData\Roaming\Free Snipping Tool\
OneDrive	C:\Users\SDEVLOOP\OneDrive
PATH	C:\Program Files\Java\jdk-21\bin
QT_DEVICE_PIXEL_RATIO	auto
TEMP	C:\Users\SDEVLOOP\AppData\Local\Temp
TMP	C:\Users\SDEVLOOP\AppData\Local\Temp

System variables

Variable	Value
OS	Windows_NT
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 165 Stepping 3, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	a503
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\syst...

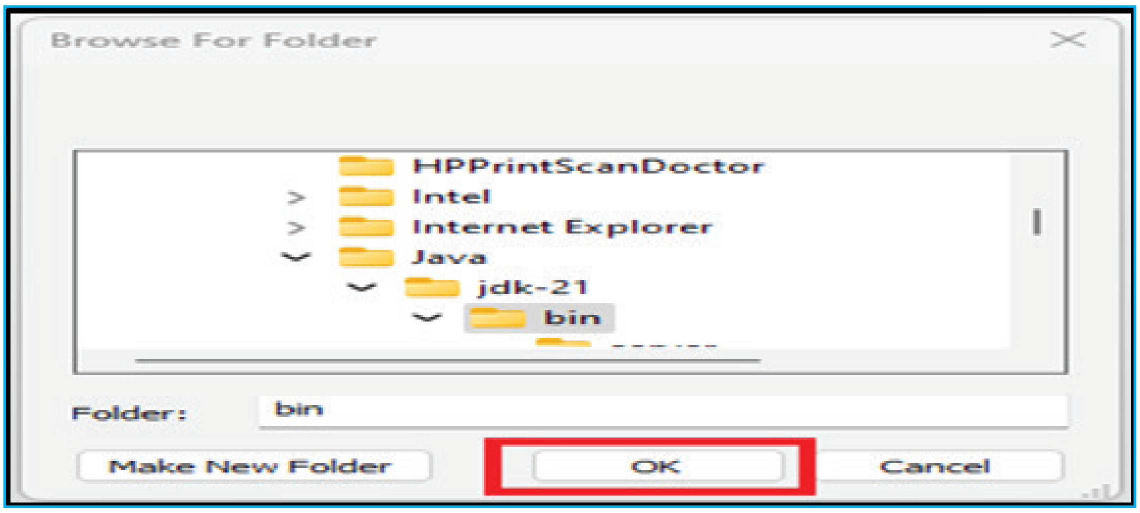
- Type the variable the name and set the path as follows.



New System Variable

Variable name: JAVA_HOME

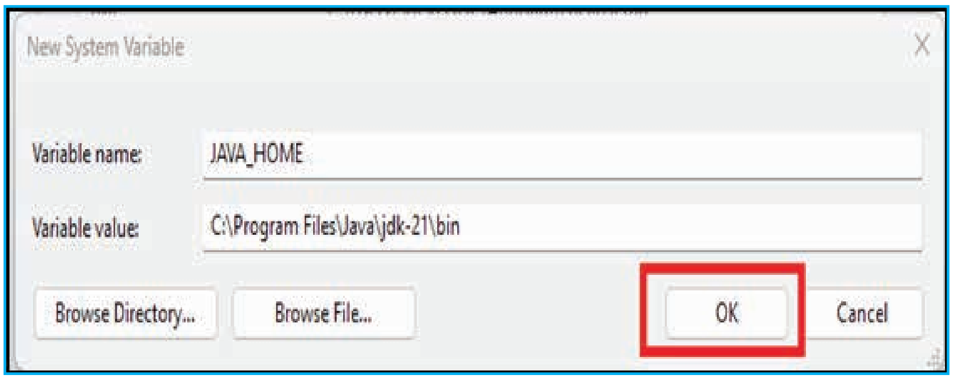
Variable value:



Browse For Folder

Folder: bin

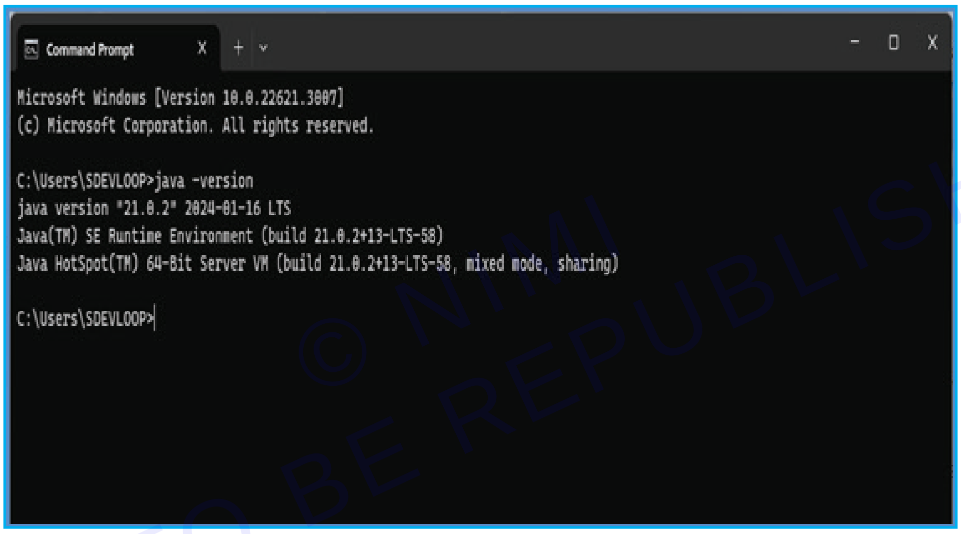
- Under "System Variables," click "New" and add a variable named JAVA_HOME with the value set to the JDK installation directory (e.g., C:\Program Files\Java\jdk-21.0.2\bin).



Variable name: JAVA_HOME
Variable value: C:\Program Files\Java\jdk-21\bin

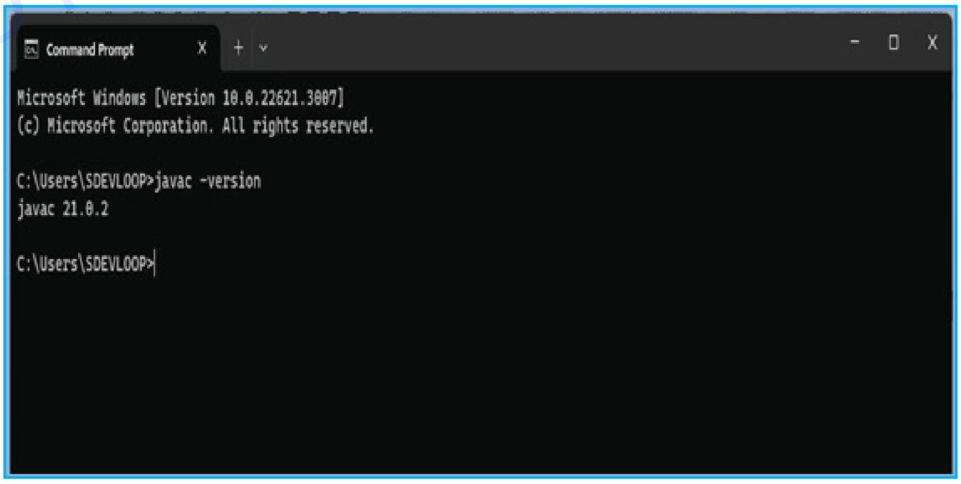
- Click "OK" to save the changes.

1 Verify JDK Installation:



```
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SDEVLOOP>java -version
java version "21.0.2" 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)

C:\Users\SDEVLOOP>
```

- Open a new Command Prompt window and type java -version. This should display the version of Java installed, confirming that JDK 21.0.2 is installed correctly.



```
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SDEVLOOP>javac -version
javac 21.0.2

C:\Users\SDEVLOOP>
```

- Also, type javac -version to verify that the Java compiler is installed and accessible.

That's it! You have successfully installed JDK 21.0.2 on your Windows system. You can now start developing and running Java applications.

EXERCISE 80 : Writing and Executing a simple JAVA Program to display “Hello”

Objectives

At the end of this exercise you shall be able to

- develop a Simple Java Program
- compile and Execute the Java Program.

Requirements

Tools/Materials

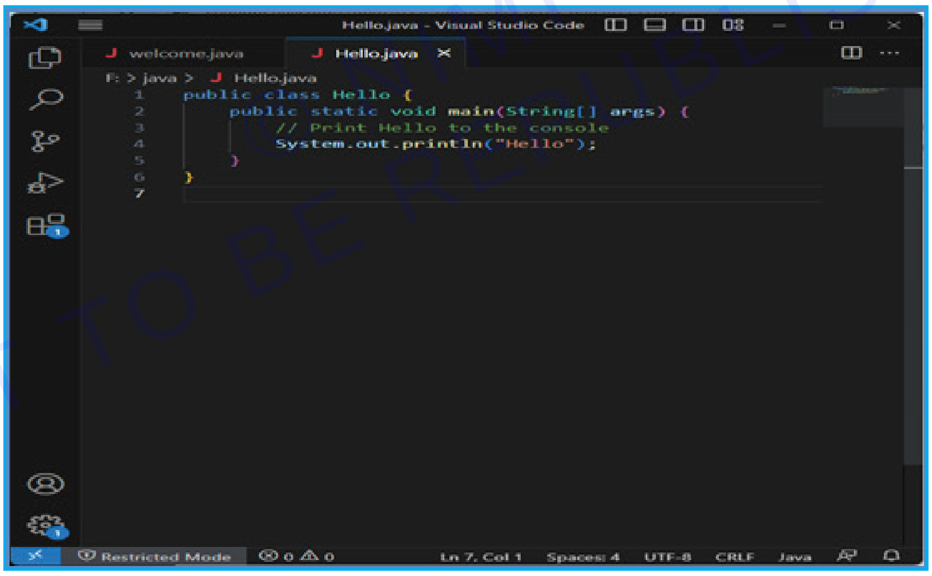
- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

Step 1: Open a Text Editor:

- Open a text editor such as Notepad, Notepad++, Visual Studio Code, or any other text editor of your choice.

Step 2: Write the Java Code:

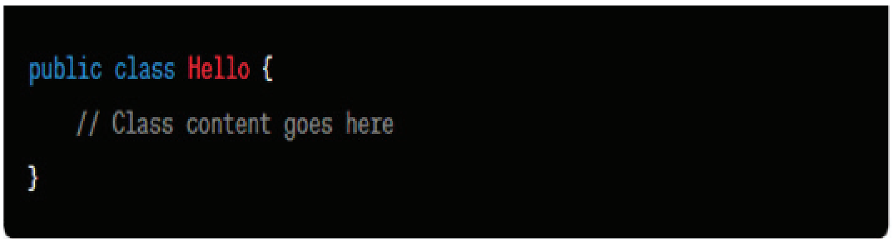


```
public class Hello {
    public static void main(String[] args) {
        // Print Hello to the console
        System.out.println("Hello");
    }
}
```

Step-by-Step Explanation:

Step 2.1: Define the Class and Main Method:

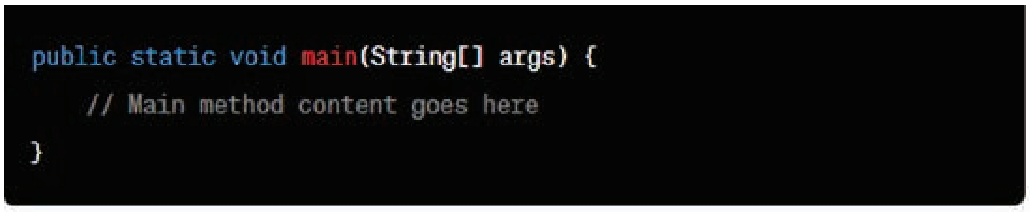
- Start by defining a class named Hello. In Java, every program needs to be within a class.
- The class name must match the filename (Hello.java).



```
public class Hello {
    // Class content goes here
}
```

Step 2.2: Define the Main Method:

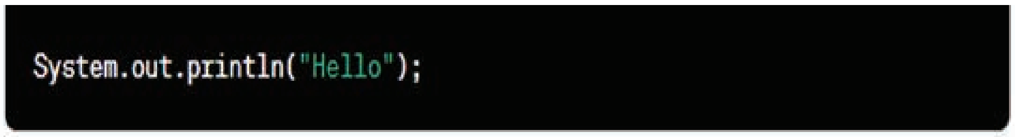
- Inside the class, define the main method. This method is the entry point for Java programs.



```
public static void main(String[] args) {
    // Main method content goes here
}
```

Step 2.3: Print "Hello" to the Console:

- Within the main method, use the System.out.println() statement to print. "Hello" to the console.



```
System.out.println("Hello");
```

Step 3: Save the File: Save the file with the name Hello.java.(Filename should be the class name)

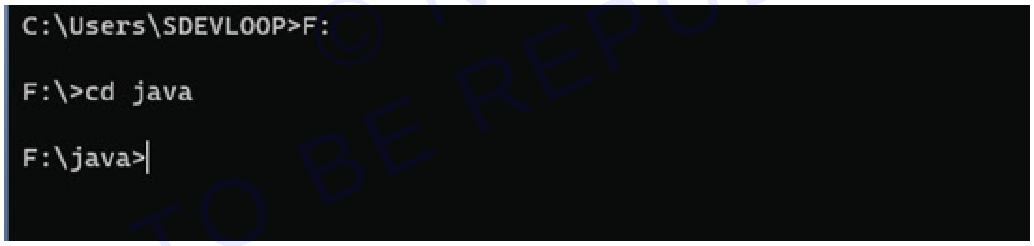
Step 4: Compiling the Java Program

1 **Open Command Prompt (cmd):**

Press Win + R, type cmd, and press Enter to open the Command Prompt.

2 **Navigate to the Directory**:

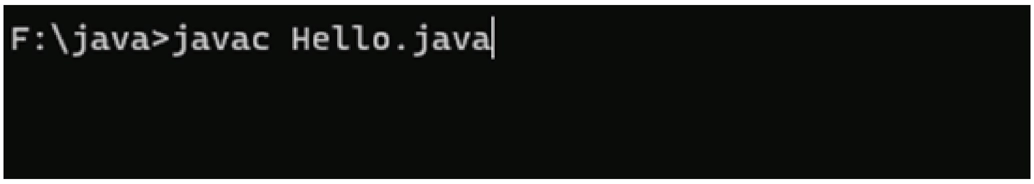
Use the cd command to navigate to the directory where Hello.java is saved. For example:



```
C:\Users\SDEVLOOP>F:

F:\>cd java

F:\java>
```

3 **Compile the Java File:**

In the Command Prompt, type the command   javac filename.java and press the Enter key

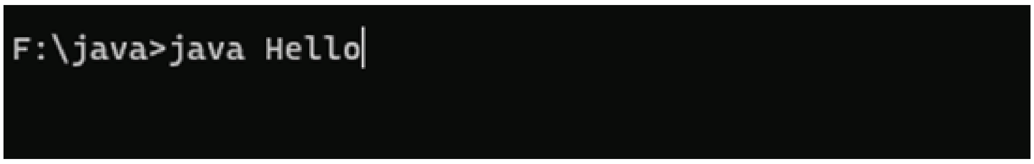


```
F:\java>javac Hello.java
```

If there are no syntax errors in your code, the Java compiler (javac) will create a file named Hello.class in the same directory.

Step 5: Running the Java Program

After successfully compiling the program, run it using the java command:

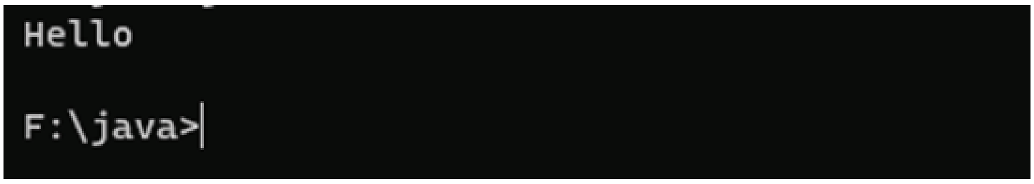


```
F:\java>java Hello
```

In the Command Prompt, type the command java filename and press Enter:

This command will execute the Hello class, and you will see “Hello” printed to the console.

Output:



```
Hello

F:\java>
```

- Related Exercise: Develop a java program to display “ Welcome to Java Programming”.

EXERCISE 81 : Use various data types in JAVA

Objectives

At the end of this exercise you shall be able to

- develop Java Programs with various data types
- compile , Execute and verify the result of the Java Programs.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

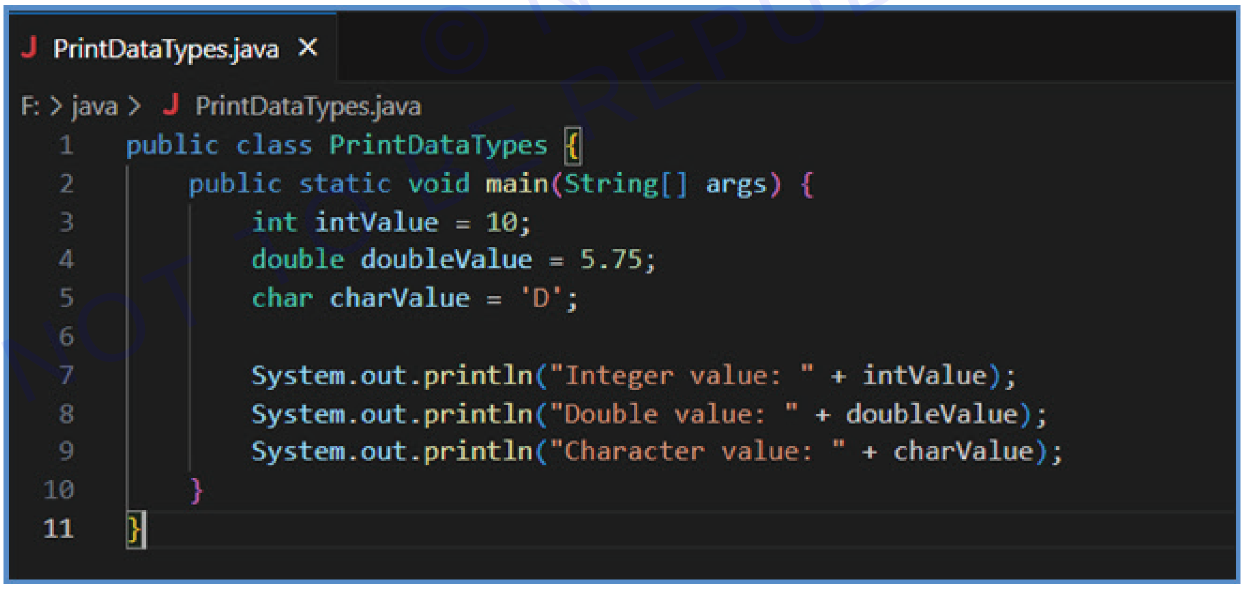
TASK 1: **Write a Java program to declare variables of different primitive data types and print their values**

Step 1: Writing the Java Program to display various data type values.

1 **Open a Text Editor**:

- Open a text editor such as Notepad, Notepad++, Visual Studio Code, or any other text editor of your choice.

2 **Write the Java Code:**



```java
public class PrintDataTypes {
    public static void main(String[] args) {
        int intValue = 10;
        double doubleValue = 5.75;
        char charValue = 'D';

        System.out.println("Integer value: " + intValue);
        System.out.println("Double value: " + doubleValue);
        System.out.println("Character value: " + charValue);
    }
}
```

3 **Save the File:** Save the file with the name PrintDataTypes.java

Step 2: Compiling the Java Program

1 **Open Command Prompt (cmd):**

Press Win + R, type cmd, and press Enter to open the Command Prompt.

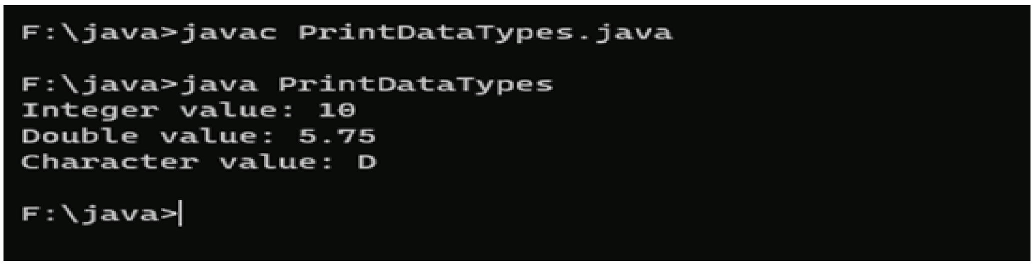
2 **Navigate to the Directory:**

Use the cd command to navigate to the directory where PrintDataTypes.java is saved.

If there are no syntax errors in your code, the Java compiler (javac) will create a file named PrintDataTypes.class in the same directory.

Step 3: Running the Java Program

1 **Run the Java Program:**



```
F:\java>javac PrintDataTypes.java

F:\java>java PrintDataTypes
Integer value: 10
Double value: 5.75
Character value: D

F:\java>
```

In the Command Prompt, type the command java filename (java PrintDataTypes) and press Enter:

This command executes the main method in the PrintDataTypes class. You should see the values of the variables printed in the console.

Summary

This Java program declares variables of different primitive data types and prints their values to the console. Following the steps outlined above will allow you to create and run the program successfully.

— — — — —

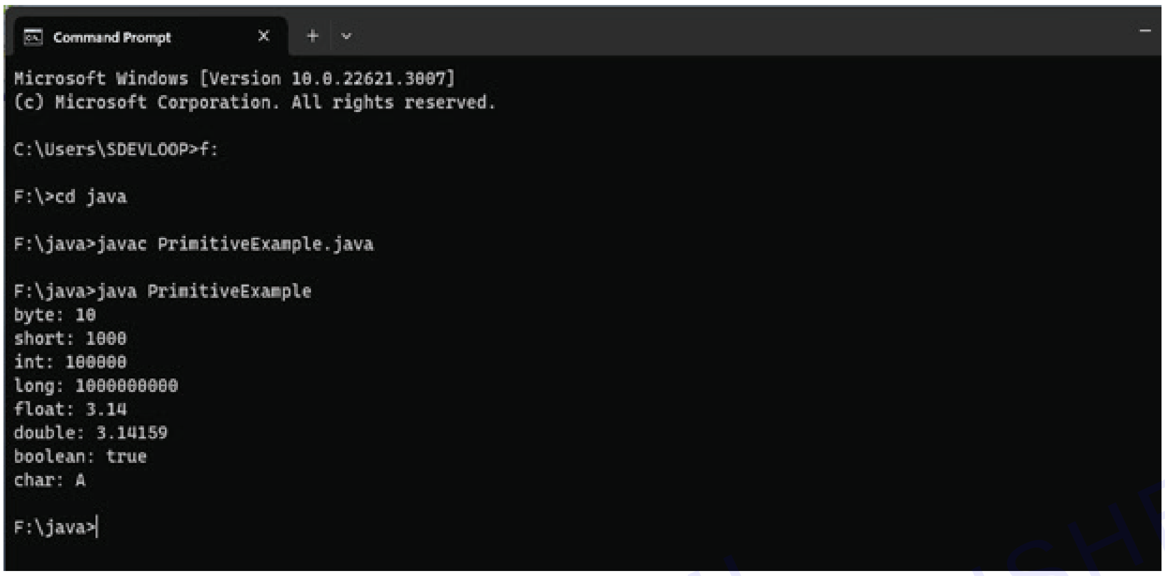
TASK 2 : **Write a Java program to display all primitive data types**

All the steps are same as that of the above program (See the steps of Task 1)

CODE:

```
public class PrimitiveExample {
    public static void main(String[] args) {
        // Declare variables of different primitive data types
        byte myByte = 10;
        short myShort = 1000;
        int myInt = 100000;
        long myLong = 1000000000L;
        float myFloat = 3.14f;
        double myDouble = 3.14159;
        boolean myBoolean = true;
        char myChar = 'A';
        // Print the values of the variables
        System.out.println("byte: " + myByte);
        System.out.println("short: " + myShort);
        System.out.println("int: " + myInt);
        System.out.println("long: " + myLong);
        System.out.println("float: " + myFloat);
        System.out.println("double: " + myDouble);
        System.out.println("boolean: " + myBoolean);
        System.out.println("char: " + myChar);
    }
}
```

Output:



```
Microsoft Windows [Version 10.0.22621.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SDEVLOOP>f:

F:\>cd java

F:\java>javac PrimitiveExample.java

F:\java>java PrimitiveExample
byte: 10
short: 1000
int: 100000
long: 1000000000
float: 3.14
double: 3.14159
boolean: true
char: A

F:\java>
```

EXERCISE 82 : Use various operators in JAVA

Objectives

At the end of this exercise you shall be able to

- know the use of various Operators in Java
- develop Java Programs using different operators
- compile , Execute and verify the result of the Java Programs.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

TASK 1: **Java Unary Operator Example: ++ and –**

CODE:

```
public class OperatorExample1{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Explanation:

1 **Variable Initialization:**

- The program begins by initializing an integer variable x with the value 10.

2 **Post-increment (x++):**

- System.out.println(x++); is a post-increment operation. It prints the current value of x (which is 10) and then increments x by 1.
- The output is 10 because the current value is printed before the increment.

3 **Pre-increment (++x):**

- System.out.println(++x); is a pre-increment operation. It increments the value of x by 1 and then prints the updated value.
- The output is 12 because x was incremented in the previous step.

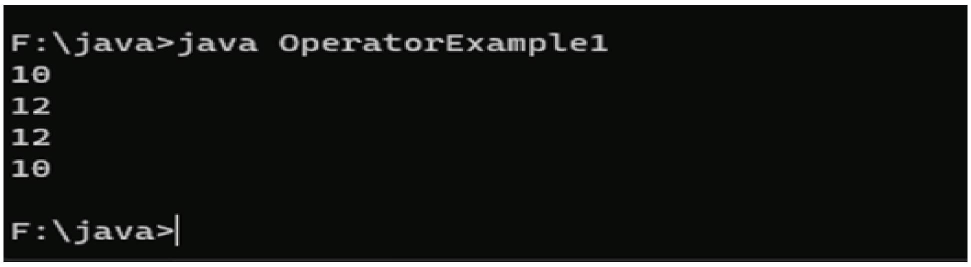
4 **Post-decrement (x--):**

- System.out.println(x--); is a post-decrement operation. It prints the current value of x (which is 12) and then decrements x by 1.
- The output is 12 because the current value is printed before the decrement.

5 Pre-decrement (--x):

- System.out.println(--x); is a pre-decrement operation. It decrements the value of x by 1 and then prints the updated value.
- The output is 10 because x was decremented in the previous step.

Output:



```
F:\java>java OperatorExample1
10
12
12
10

F:\java>
```

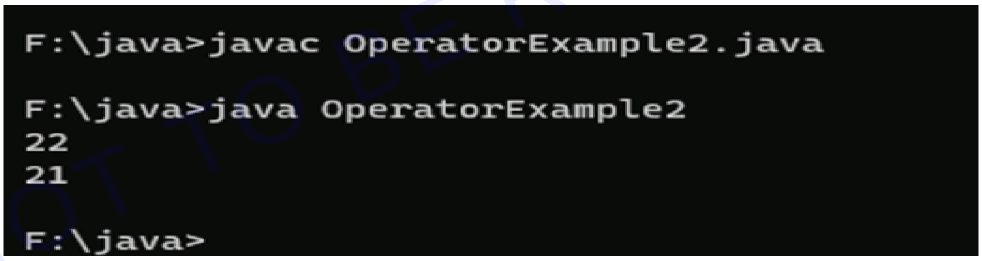
— — — — —

TASK 2: **Java Unary Operator Example 2: ++ and --**

CODE:

```
public class OperatorExample2{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21
}}
```

Output :



```
F:\java>javac OperatorExample2.java

F:\java>java OperatorExample2
22
21

F:\java>
```

Explanation:

- The program illustrates the behavior of post-increment and pre-increment operators in expressions.
- Post-increment (a++ or b++) evaluates to the current value before the increment, while pre-increment (++a) evaluates to the updated value after the increment.
- The output demonstrates the results of the two expressions involving increment operations.

— — — — —

TASK 3: **Java Unary Operator Example: ~ and !**

```
public class OperatorExample3{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//9 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
}}
```

Explanation:

1 **Variable Initialization:**

- The program begins by initializing two integer variables a and b with the values 10 and -10, respectively.
- It also initializes two boolean variables c and d with the values true and false, respectively.

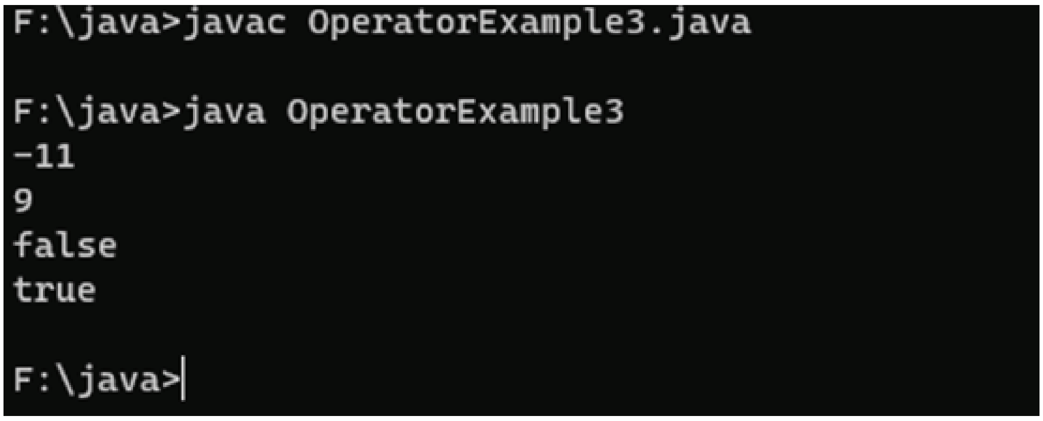
2 **Bitwise Complement (~):**

- System.out.println(~a);
- The bitwise complement (~) operator inverts the bits of the integer a.
- In binary representation, -11 is the two's complement of 10.
- The output is -11, which is the result of inverting the bits of 10.
- System.out.println(~b);
- The bitwise complement (~) operator inverts the bits of the integer b.
- In binary representation, 9 is the two's complement of -10.
- The output is 9, which is the result of inverting the bits of -10.

3 **Logical NOT (!):**

- System.out.println(!c);
- The logical NOT (!) operator negates the boolean value of c.
- The output is false, which is the result of negating the boolean value true.
- System.out.println(!d);
- The logical NOT (!) operator negates the boolean value of d.
- The output is true, which is the result of negating the boolean value false.

Output:

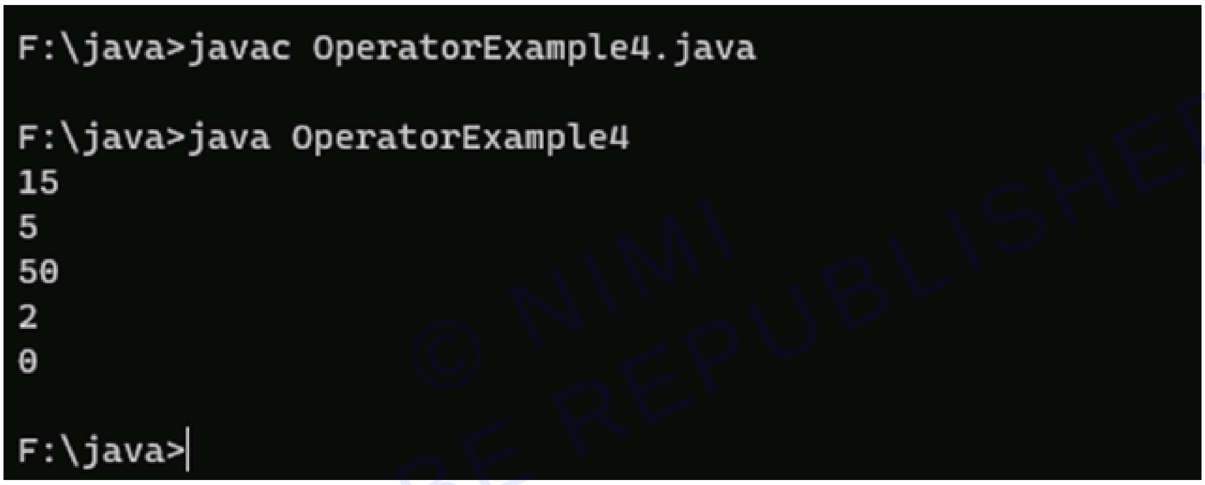


```
F:\java>javac OperatorExample3.java

F:\java>java OperatorExample3
-11
9
false
true

F:\java>
```

TASK 4: **Java Arithmetic Operator Example**

```
public class OperatorExample4{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:



```
F:\java>javac OperatorExample4.java

F:\java>java OperatorExample4
15
5
50
2
0

F:\java>
```

TASK 5: **Java Arithmetic Operator Example: Expression**

```
public class OperatorExample5{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

Explanation:

1 **Arithmetic Expression:**

- The program consists of a single arithmetic expression within the System.out.println statement.

2 **Order of Operations (BODMAS/BIDMAS):**

- The arithmetic expression follows the order of operations (also known as BODMAS or BIDMAS), which stands for:
- Brackets or Parentheses
- Orders (i.e., powers and square roots, etc.)
- Division and Multiplication (from left to right)
- Addition and Subtraction (from left to right)

3 Arithmetic Operations:

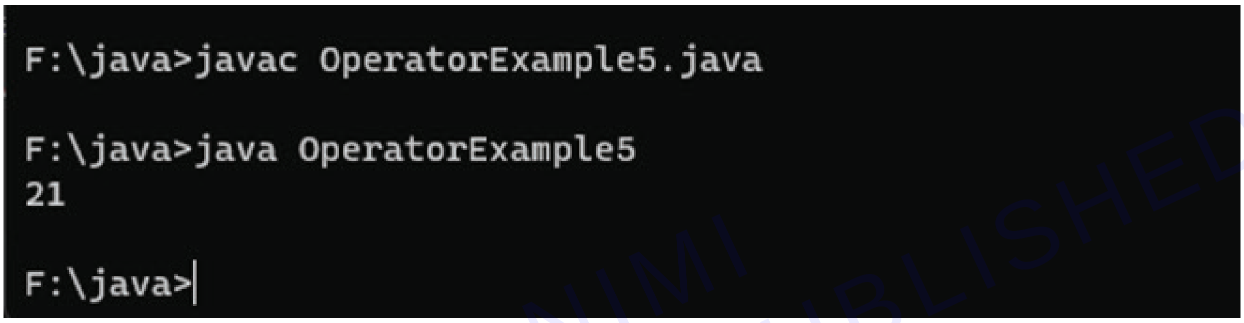
- The expression 10 * 10 / 5 + 3 - 1 * 4 / 2 involves multiplication (*), division (/), addition (+),and subtraction (-) operations.

4 Step-by-Step Evaluation:

10 * 10 / 5 + 3 - 1 * 4 / 2

- Multiply: 100 / 5 + 3 - 1 * 4 / 2
- Divide: 20 + 3 - 1 * 4 / 2
- Multiply: 20 + 3 - 4 / 2
- Divide: 20 + 3 - 2
- Add: 23 - 2
- Subtract: 21

Output:



```
F:\java>javac OperatorExample5.java

F:\java>java OperatorExample5
21

F:\java>
```

TASK 6: **Java Left Shift Operator Example**

```
public class OperatorExample6{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Explanation:

1 Left Shift Operator (<<):

- The left shift operator (<<) shifts the bits of a binary number to the left by a specified number of positions.
- The general form is value << numBits, where value is the number to be shifted, and num Bits is the number of positions to shift.

2 Bitwise Left Shift Operations:

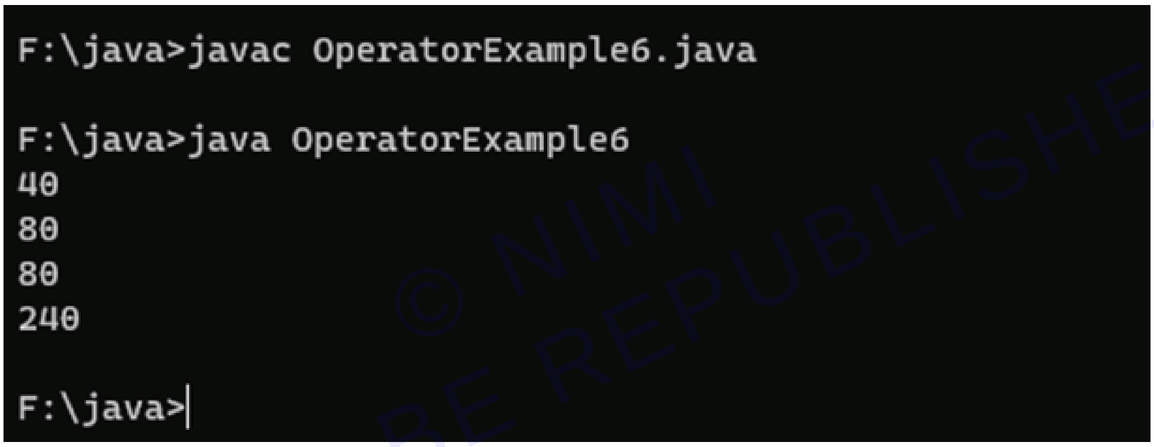
- The program performs left shift operations on various integers.

3 Examples:

- System.out.println(10 << 2);
- Left shift the binary representation of 10 by 2 positions.
- 10 in binary is 1010. After left shifting by 2, it becomes 101000, which is 40 in decimal.
- Output: 40

- System.out.println(10 << 3);
- Left shift the binary representation of 10 by 3 positions.
- After left shifting by 3, 10 becomes 1010000, which is 80 in decimal.
- Output: 80
- System.out.println(20 << 2);
- Left shift the binary representation of 20 by 2 positions.
- After left shifting by 2, 20 becomes 101000, which is 80 in decimal.
- Output: 80
- System.out.println(15 << 4);
- Left shift the binary representation of 15 by 4 positions.
- After left shifting by 4, 15 become 11110000, which is 240 in decimal.
- Output: 240

Output:



```
F:\java>javac OperatorExample6.java

F:\java>java OperatorExample6
40
80
80
240

F:\java>
```

TASK 7: **Java Right Shift Operator Example**

```
public class OperatorExample7{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Explanation:

1 **Right Shift Operator (>>):**

- The right shift operator (>>) shifts the bits of a binary number to the right by a specified number of positions.
- The general form is value >> numBits, where value is the number to be shifted, and numBits is the number of positions to shift.

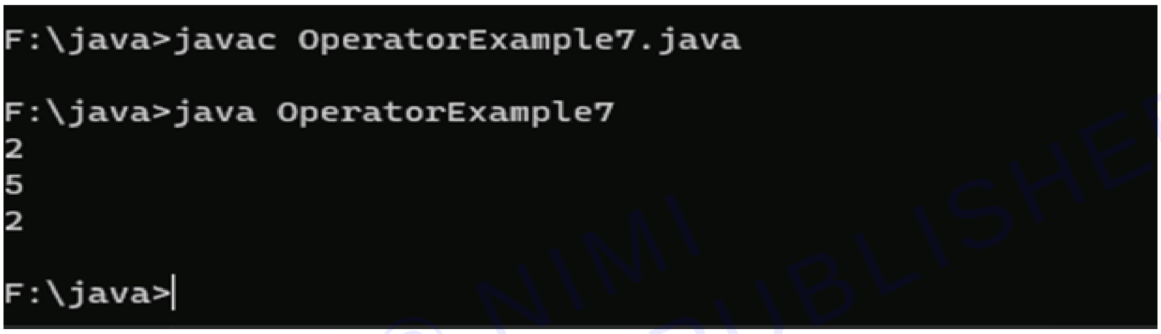
2 **Bitwise Right Shift Operations:**

- The program performs right shift operations on various integers.

3 **Examples:**

- System.out.println(10 >> 2);
- Right shift the binary representation of 10 by 2 positions.
- 10 in binary is 1010. After right shifting by 2, it becomes 10, which is 2 in decimal.
- Output: 2
- System.out.println(20 >> 2);
- Right shift the binary representation of 20 by 2 positions.
- After right shifting by 2, 20 becomes 5 in decimal.
- Output: 5
- System.out.println(20 >> 3);
- Right shift the binary representation of 20 by 3 positions.
- After right shifting by 3, 20 becomes 2 in decimal.
- Output: 2

Output:



```
F:\java>javac OperatorExample7.java

F:\java>java OperatorExample7
2
5
2

F:\java>
```

TASK 8: **Java Shift Operator Example: >>vs>>>**

```
public class OperatorExample8{
public static void main(String args[]){
    //For positive number, >> and >>> works same
    System.out.println(20>>2);
    System.out.println(20>>>2);
    //For negative number, >>> changes parity bit (MSB) to 0
    System.out.println(-20>>2);
    System.out.println(-20>>>2);
}}
```

Explanation:

1 **Right Shift Operator (>>):**

- The right shift operator (>>) shifts the bits of a binary number to the right by a specified number of positions.
- For positive numbers, the vacant leftmost positions are filled with the sign bit (MSB, Most Significant Bit).

2 **Unsigned Right Shift Operator (>>>):**

- The unsigned right shift operator (>>>) also shifts the bits to the right, but it fills the vacant leftmost positions with zeros, irrespective of the sign bit.
- It treats the number as if it were an unsigned quantity.

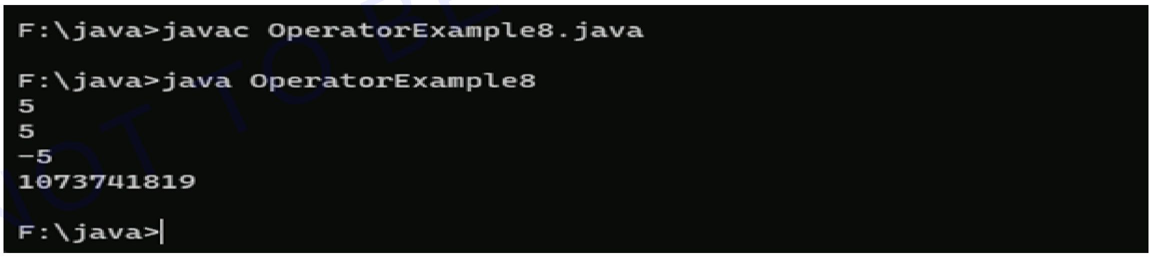
3 Bitwise Right Shift Operations:

- The program performs both right shift (>>) and unsigned right shift (>>>) operations on both positive and negative numbers.

4 Examples:

- System.out.println(20 >> 2);
- Right shift the binary representation of 20 by 2 positions using the >> operator.
- After right shifting by 2, 20 becomes 5 in decimal.
- Output: 5
- System.out.println(20 >>> 2);
- Unsigned right shift the binary representation of 20 by 2 positions using the >>> operator.
- After unsigned right shifting by 2, 20 becomes 5 in decimal.
- Output: 5
- System.out.println(-20 >> 2);
- Right shift the binary representation of -20 by 2 positions using the >> operator.
- For negative numbers, the vacant leftmost positions are filled with the sign bit (1), and -20 becomes -5 in decimal.
- Output: -5
- System.out.println(-20 >>> 2);
- Unsigned right shift the binary representation of -20 by 2 positions using the >>> operator.
- The unsigned right shift fills the vacant leftmost positions with 0, and -20 becomes a positive integer (1073741819 in decimal).
- Output: 1073741819

Output:



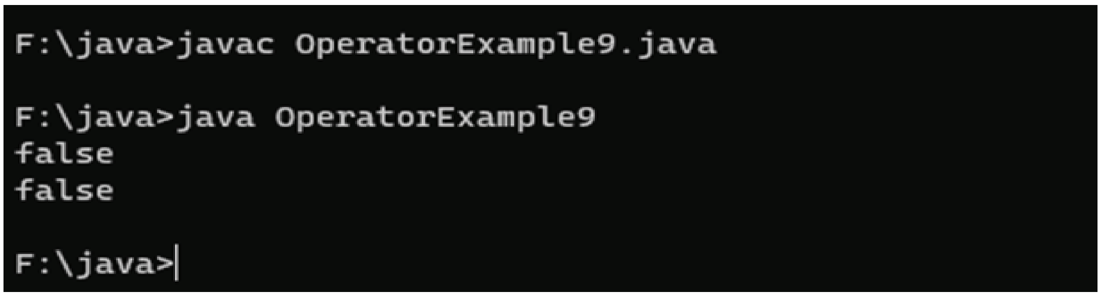
```
F:\java>javac OperatorExample8.java

F:\java>java OperatorExample8
5
5
-5
1073741819

F:\java>
```

— — — — —

TASK 9: **Java AND Operator Example: Logical && and Bitwise &**

```
public class OperatorExample9{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

Output:



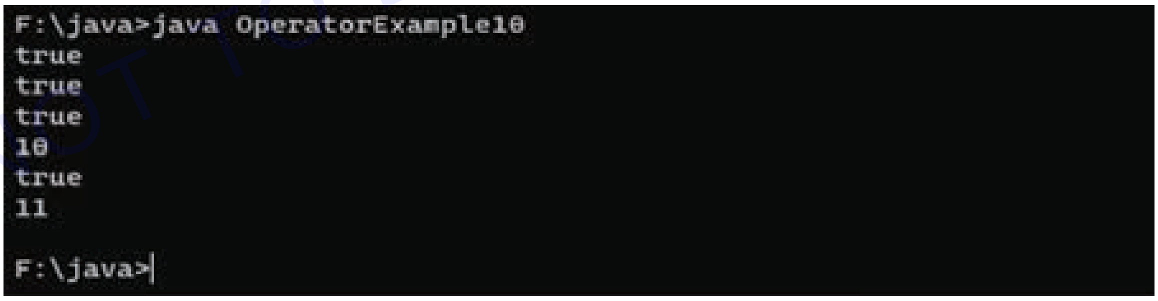
```
F:\java>javac OperatorExample9.java

F:\java>java OperatorExample9
false
false

F:\java>
```

— — — — —

TASK 10: **Java OR Operator Example: Logical || and Bitwise |**

```
public class OperatorExample10{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Output:



```
F:\java>java OperatorExample10
true
true
true
10
true
11

F:\java>
```

— — — — —

TASK 11: **Java Ternary Operator Example**

```
public class OperatorExample11{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Explanation:

1 **Ternary Conditional Operator (? :):**

- The ternary conditional operator is a shorthand way of writing an if-else statement.
- The general form is condition ? expression1 : expression2.
- If the condition is true, it evaluates to expression1; otherwise, it evaluates to expression2.

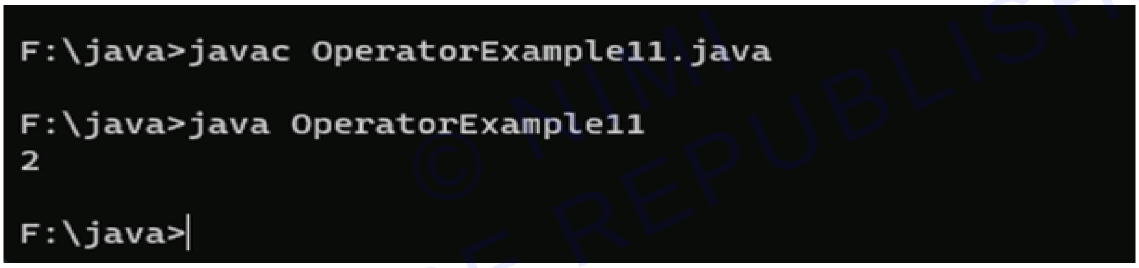
2 **Program Logic:**

- The program defines two integer variables a and b with values 2 and 5, respectively.
- It uses the ternary conditional operator to find the minimum of a and b.
- The condition (a < b) is evaluated. If true, a is assigned to min; otherwise, b is assigned to min.
- The minimum value is then printed to the console.

3 **Example:**

- int min = (a < b) ? a : b;
- The condition (a < b) is true (2 is less than 5).
- Therefore, min is assigned the value of a (2).
- Output: 2

Output:

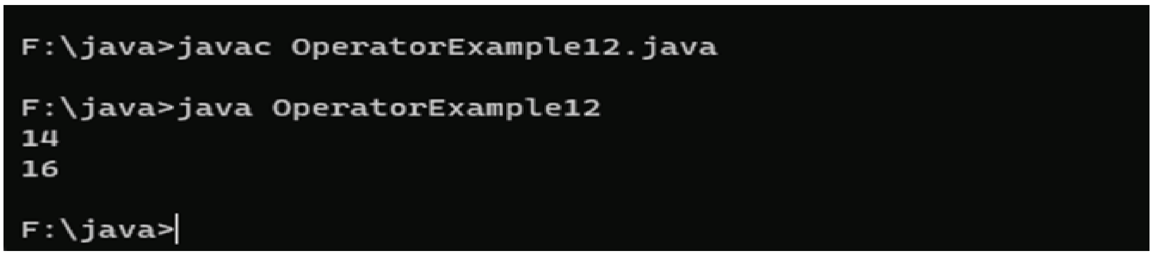


```
F:\java>javac OperatorExample11.java

F:\java>java OperatorExample11
2

F:\java>
```

TASK 12: **Java Assignment Operator Example**

```
public class OperatorExample12{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

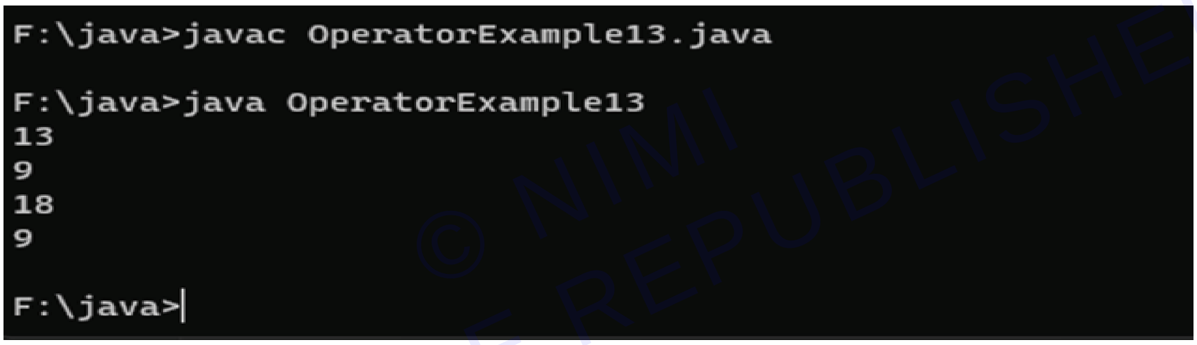


```
F:\java>javac OperatorExample12.java

F:\java>java OperatorExample12
14
16

F:\java>
```

TASK 13: **Java Assignment Operator Example**

```
public class OperatorExample13{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

Output:



```
F:\java>javac OperatorExample13.java

F:\java>java OperatorExample13
13
9
18
9

F:\java>
```

EXERCISE 83 : Create and use Local, Instance and Class variables

Objectives

At the end of this exercise you shall be able to

- know the use of various Local, Instance and Class variables
- develop Java Programs using different Local, Instance and Class variables
- compile , Execute and verify the result of the Java Programs.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software

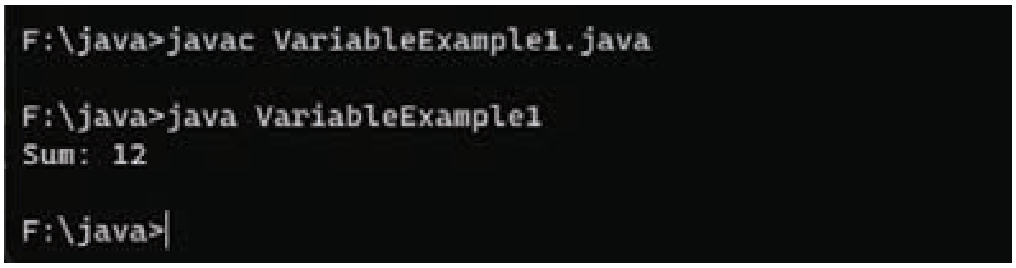
Procedure

Text Editor (Visual Studio/Sublime/Notepad)

TASK 1 : **Local Variables Example**

```
public class VariableExample1 {
 public static void main(String[] args) {
        calculateSum(5, 7);
    }
    static void calculateSum(int a, int b) {
        // Local variable: sum
        int sum = a + b;
        System.out.println(“Sum: “ + sum);
    }
}
```

Output:



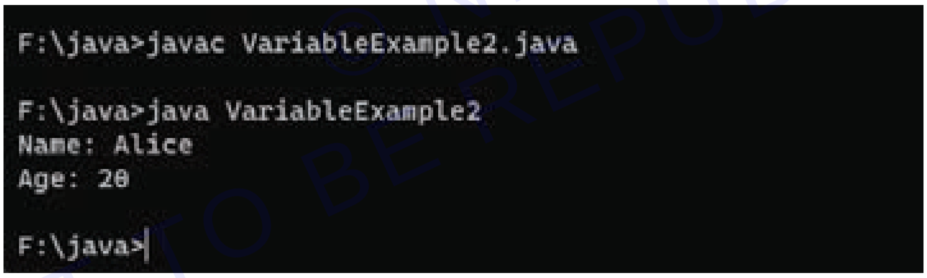
```
F:\java>javac VariableExample1.java

F:\java>java VariableExample1
Sum: 12

F:\java>
```

— — — — —

TASK 2: **Instance Variables Example**

```
public class VariableExample2 {
    public static void main(String[] args) {
        // Create an instance of the Student class
        Student myStudent = new Student();
        // Set values to instance variables
        myStudent.name = "Alice";
        myStudent.age = 20;
        // Display student information
        myStudent.displayStudentInfo();
    }
}
class Student {
    // Instance variables: name and age
    String name;
    int age;
    // Instance method to display student information
    void displayStudentInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Output:



```
F:\java>javac VariableExample2.java

F:\java>java VariableExample2
Name: Alice
Age: 20

F:\java>
```

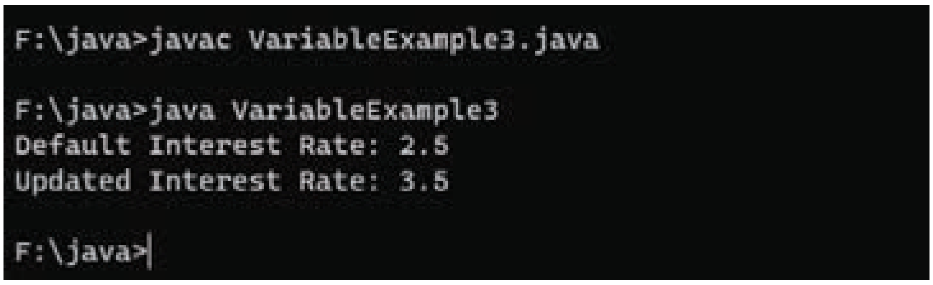
— — — — —

TASK 3: Class Variables Examples

```
public class VariableExample3 {
    public static void main(String[] args) {
        // Access the class variable directly
        System.out.println("Default Interest Rate: " + Bank.defaultInterestRate);
        // Change the class variable
        Bank.defaultInterestRate = 3.5;
  // Access the class variable through an instance
        Bank myBank = new Bank();
        System.out.println("Updated Interest Rate: " + myBank.defaultInterestRate);
    }
}
```

```
class Bank {
    // Class variable: defaultInterestRate
    static double defaultInterestRate = 2.5;
}
```

Output:



```
F:\java>javac VariableExample3.java

F:\java>java VariableExample3
Default Interest Rate: 2.5
Updated Interest Rate: 3.5

F:\java>
```

EXERCISE 84 : Read text from the keyboard using scanner class/ read text from the keyboard using console class

Objectives

At the end of this exercise you shall be able to

- should know the use of scanner class/console class
- develop java programs using scanner class/console clas
- compile, execute and verify the result of the Java Programs.

Requirements

Tools/Materials

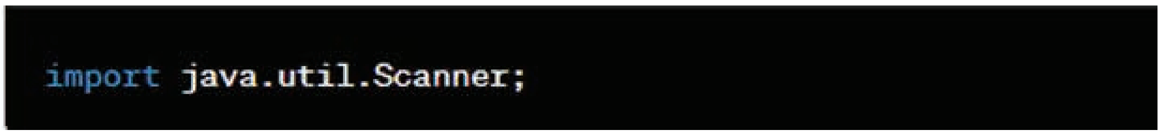
- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

TASK 1: **Reading Text with Scanner Class Example_1**

```
import java.util.Scanner;
public class ScannerExample {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter a name
        System.out.print("Enter your name: ");
        // Read the entered text as a String
        String name = scanner.nextLine();
        // Display a greeting
        System.out.println("Hello, " + name + "!");
        // Close the Scanner
        scanner.close();
    }
}
```

Step 1: Importing the Scanner Class



```
import java.util.Scanner;
```

This line imports the Scanner class from the java.util package. The Scanner class is used for obtaining user input from the keyboard.

Step 2: Declaring the Class

```
public class ScannerExample {
```

This line declares a public class named ScannerExample. All Java applications must have a class with a main method, and this is the starting point of the program.

Step 3: Declaring the main Method

```
public static void main(String[] args) {
```

This line declares the main method, which is the entry point of the Java program. The method takes an array of String arguments (args), though in this case, it is not used.

Step 4: Creating a Scanner Object

```
Scanner scanner = new Scanner(System.in);
```

This line creates a new Scanner object named scanner that reads input from the standard input stream (System.in), which represents the keyboard.

Step 5: Prompting User Input

```
System.out.print("Enter your name: ");
```

This line prints the prompt “Enter your name: “ to the console, prompting the user to enter their name.

Step 6: Reading User Input

```
String name = scanner.nextLine();
```

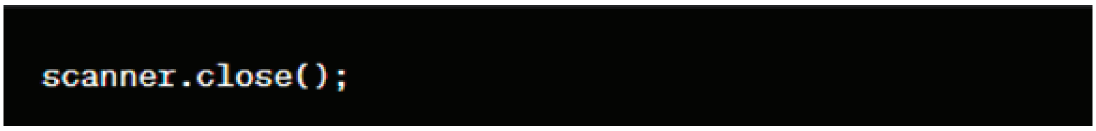
This line uses the nextLine method of the Scanner class to read the entire line of text entered by the user. The entered text is then stored in the String variable name.

Step 7: Displaying Output

```
System.out.println("Hello, " + name + "!");
```

This line prints a greeting message to the console, incorporating the user's entered name. The println method is used to print the message and move to the next line.

Step 8: **Closing the Scanner**



```
scanner.close();
```

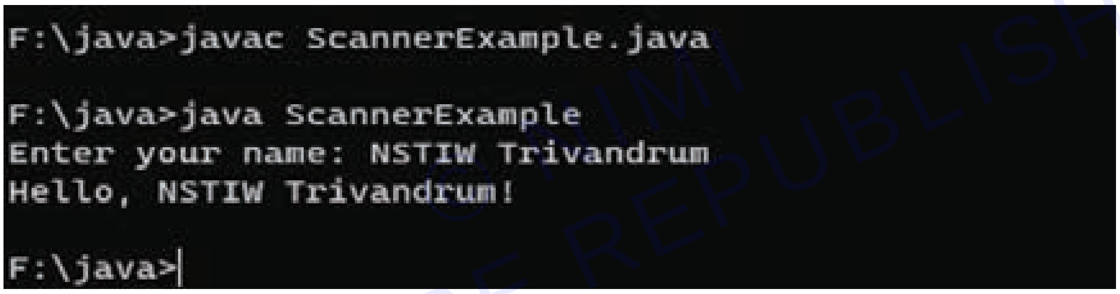
This line closes the Scanner object to release associated resources. It's good practice to close the Scanner when it's no longer needed.

Step 9: End of the Program

```
}
```

This closing brace marks the end of the main method and the end of the ScannerExample class.

Output:



```
F:\java>javac ScannerExample.java

F:\java>java ScannerExample
Enter your name: NSTIW Trivandrum
Hello, NSTIW Trivandrum!

F:\java>
```

Summary:

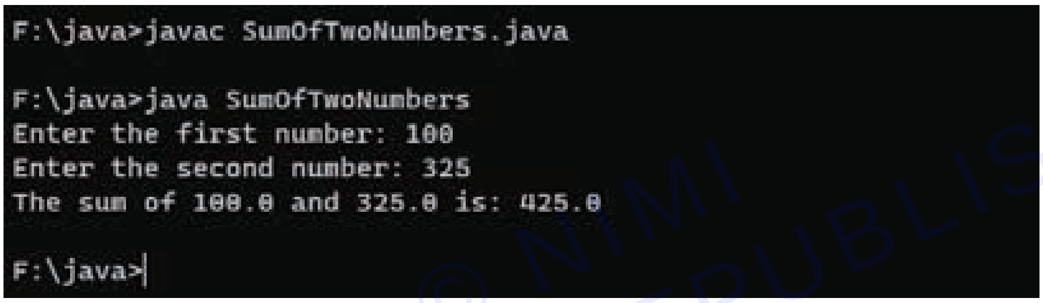
- The program starts by importing the Scanner class.
- It creates a Scanner object to read input from the keyboard.
- It prompts the user to enter their name, reads the input, and stores it in a variable.
- It displays a greeting message incorporating the user's name.
- Finally, it closes the Scanner object.

— — — — —

TASK 2: **Reading Multiple Inputs with Scanner Class Example_2**

```
import java.util.Scanner;
public class SumOfTwoNumbers {
    public static void main(String[] args) {
        // Create a Scanner object to read input
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter the first number
        System.out.print("Enter the first number: ");
        // Read the first number from the user
        double num1 = scanner.nextDouble();
// Prompt the user to enter the second number
        System.out.print("Enter the second number: ");
        // Read the second number from the user
        double num2 = scanner.nextDouble();
        // Close the Scanner to avoid resource leak
        scanner.close();
        // Calculate the sum of the two numbers
        double sum = num1 + num2;
        // Display the result
        System.out.println("The sum of " + num1 + " and " + num2 + " is: " + sum);
    }
}
```

Output:

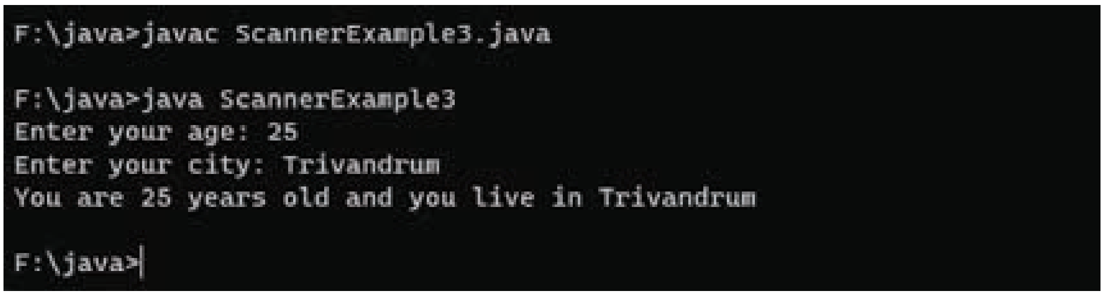


```
F:\java>javac SumOfTwoNumbers.java

F:\java>java SumOfTwoNumbers
Enter the first number: 100
Enter the second number: 325
The sum of 100.0 and 325.0 is: 425.0

F:\java>
```

TASK 3: **Reading Multiple Inputs with Scanner Class Example_3**

```
import java.util.Scanner;
public class ScannerExample3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter age and city
        System.out.print("Enter your age: ");
        int age = scanner.nextInt();
        // Consume the newline character left by nextInt
        scanner.nextLine();
        System.out.print("Enter your city: ");
        String city = scanner.nextLine();
        // Display user information
        System.out.println("You are " + age + " years old and you live in " + city);
        // Close the Scanner
        scanner.close();
    }
}
```

Output:

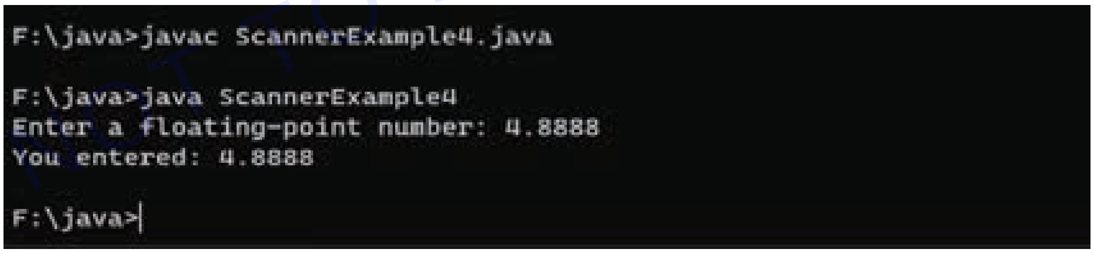

```
F:\java>javac ScannerExample3.java

F:\java>java ScannerExample3
Enter your age: 25
Enter your city: Trivandrum
You are 25 years old and you live in Trivandrum

F:\java>
```

— — — — —

TASK 4: **Reading a floating-point number with Scanner Class Example_4**

```
import java.util.Scanner;
public class ScannerExample4 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
     // Prompt the user to enter a floating-point number
        System.out.print("Enter a floating-point number: ");
        double floatingPointNumber = scanner.nextDouble();
        // Display the entered number
        System.out.println("You entered: " + floatingPointNumber);
        // Close the Scanner
        scanner.close();
    }
}
```

Output:

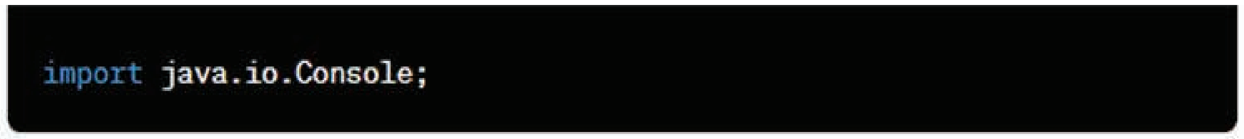

```
F:\java>javac ScannerExample4.java

F:\java>java ScannerExample4
Enter a floating-point number: 4.8888
You entered: 4.8888

F:\java>
```

— — — — —

TASK 5: **Reading Text with Console Class Example_1**

```
import java.io.Console;
public class ConsoleExample1 {
    public static void main(String[] args) {
        // Get the Console object
        Console console = System.console();
        if (console == null) {
            System.out.println("Console not available. Exiting...");
            System.exit(1);
        }
        // Prompt the user to enter a city
        String city = console.readLine("Enter your favorite city: ");
        // Display the entered city
        System.out.println("Your favorite city is: " + city);
    }
}
```

Step 1: Import Console Class

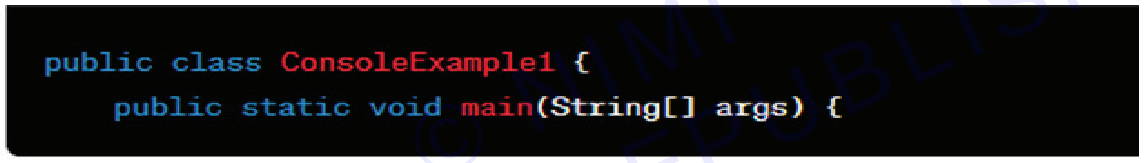


```
import java.io.Console;
```

Explanation:

- The import java.io.Console; statement is used to import the Console class from the java.io package.
- The Console class provides methods for interacting with the console, allowing secure input without echoing characters (useful for reading sensitive information like passwords).

Step 2: Define Class and Main Method

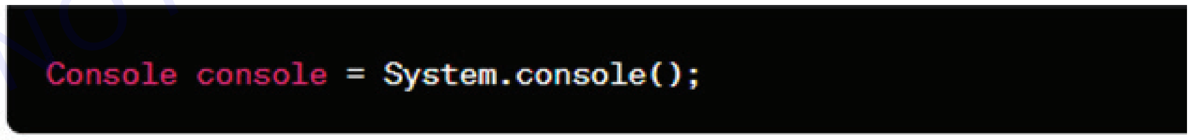


```
public class ConsoleExample1 {
    public static void main(String[] args) {
```

Explanation:

- The code defines a class named ConsoleExample1.
- Inside the class, there is the main method, which serves as the entry point of the program. The main method is the first method that gets executed when the program runs.

Step 3: Get Console Object

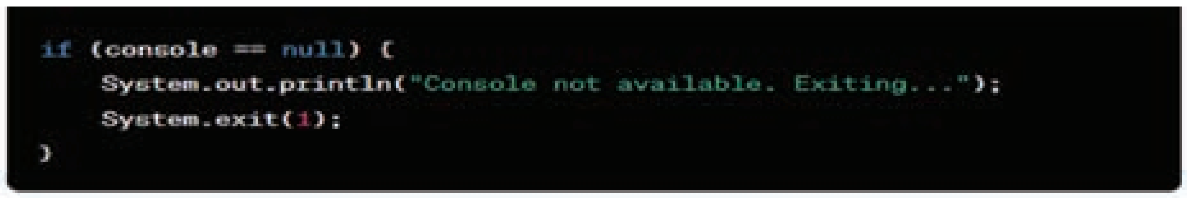


```
Console console = System.console();
```

Explanation:

- The System.console() method is used to obtain a reference to the console.
- If the console is not available (for example, if the program is running in an environment without a console, such as some IDEs), the method returns null.

Step 4: Check for Console Availability

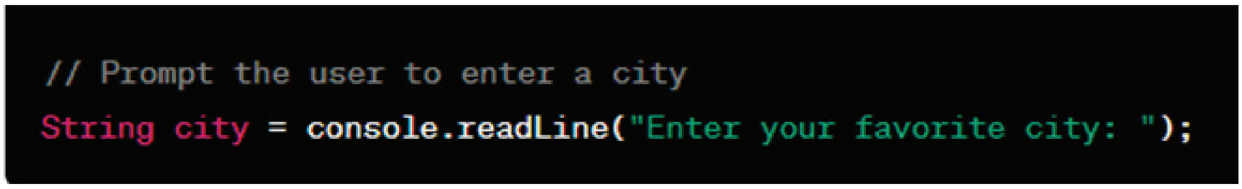


```
if (console == null) {
    System.out.println("Console not available. Exiting...");
    System.exit(1);
}
```

Explanation:

- The program checks whether the console object is null.
- If console is null, it means that the console is not available, and the program prints an error message.
- The System.exit(1); statement is used to exit the program with a status code of 1, indicating an abnormal termination.

Step 5: Prompt the User to Enter a City

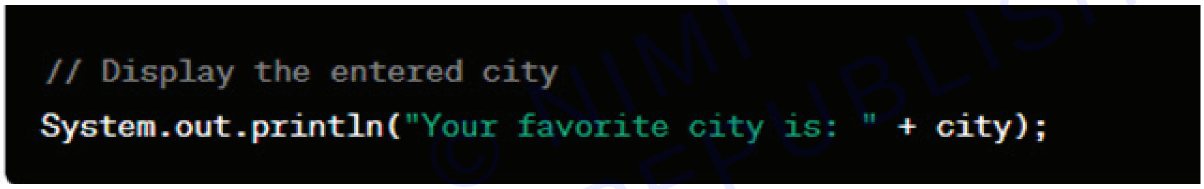


```
// Prompt the user to enter a city
String city = console.readLine("Enter your favorite city: ");
```

Explanation:

- The console.readLine("Enter your favorite city: "); statement prompts the user to enter their favorite city.
- The readLine method of the Console class is used to read a line of text from the console. It displays the specified prompt and waits for the user to enter input.
- The entered text is then assigned to the variable city.

Step 6: Display the Entered City

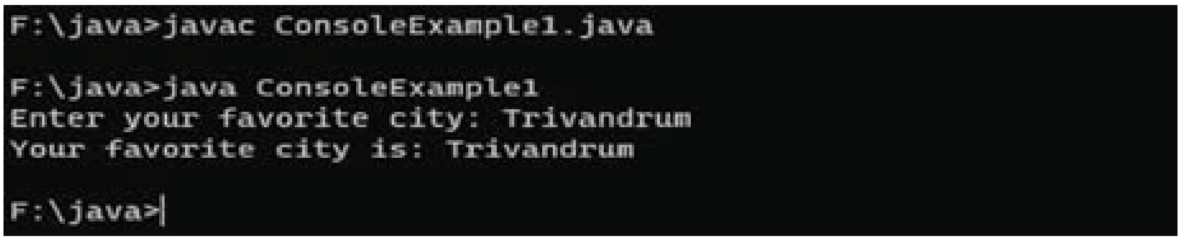


```
// Display the entered city
System.out.println("Your favorite city is: " + city);
```

Explanation:

- After the user enters their favorite city, the program displays the entered city using the System.out.println statement.
- The println method is used to print the message "Your favorite city is: " followed by the value stored in the city variable.
- This line effectively prints the user's favorite city to the console.

Output:



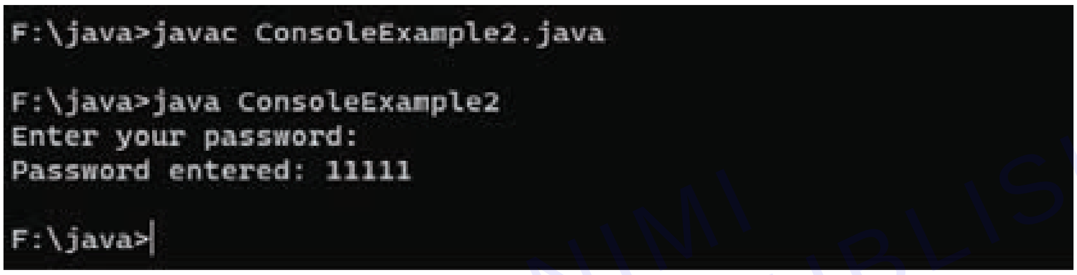
```
F:\java>javac ConsoleExample1.java

F:\java>java ConsoleExample1
Enter your favorite city: Trivandrum
Your favorite city is: Trivandrum

F:\java>
```

— — — — —

TASK 6: **Reading Password with Console Class Example_2**

```
import java.io.Console;
public class ConsoleExample2 {
    public static void main(String[] args) {
        Console console = System.console();
        if (console == null) {
            System.out.println("Console not available. Exiting...");
            System.exit(1);
        }
        // Read a password without displaying it on the console
        char[] passwordChars = console.readPassword("Enter your password: ");
        String password = new String(passwordChars);
        // Display a confirmation message
        System.out.println("Password entered: " + password);
    }
}
```

Output:



```
F:\java>javac ConsoleExample2.java

F:\java>java ConsoleExample2
Enter your password:
Password entered: 11111

F:\java>
```

Related Exercises:

Question1: Create a program that stores information about your favorite book. Use appropriate data types for the title (String), author (String), publication year (int), and price (double). Display the book details on the console.

Question 2: Write a program to calculate the area and perimeter of a rectangle. Prompt the user to enter the length and width using the Scanner class and display the result.

Question 3: Design a class representing a basic bank account. Use instance variables to store the account holder's name, account number, and balance. Implement a method to deposit money into the account and display the updated balance.

Question 4: Develop a program that reads the user's Name, Age, and favorite color using the Scanner class and display it.

Question 5: Develop a program to perform all Simple Arithmetic Operations.

Question 6: Create a program to Swap two numbers.

Question 7: Develop a program to convert the temperature in Celsius to Fahrenheit

Question 8: Create program to calculate area and circumference of a circle.

Question 9: Develop a program to calculate the interest (I=PNR where P=Principle Amount,N=No. of years, R=Rate of interest)

Question 10: Develop a program to display the Product Details (Read Product_Code, Product_Name,Unit_Price and Quantity. Calculate the Total_Price).

EXERCISE 85 : Use the if and if … else statements

Objectives

At the end of this exercise you shall be able to

- know the different syntax and use of if Statements (simple if , if…else, elseif ladder, nested if statements)
- develop java programs using if statement.

Requirements

Tools/Materials

- PC/Laptop with Window OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

Procedure

TASK 1: **Checking if a number is even or odd**

```
import java.util.Scanner;
public class EvenOddChecker {
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number: ");
int number = scanner.nextInt();
if (number % 2 == 0) {
System.out.println(number + " is an even number.");
        } else {
System.out.println(number + " is an odd number.");
        }
scanner.close();
    }
     }
```

Explanation:

This Java program, named EvenOddChecker, is designed to determine whether a given input number is even or odd. Here's how it works:

1 The program starts by importing the Scanner class from the java.util package. The Scanner class allows user input from the console.

2 The EvenOddChecker class contains the main method, which is the entry point of the program.

3 Inside the main method:

- It creates a new Scanner object named scanner to read input from the console.
- It prompts the user to enter a number by displaying the message "Enter a number: ".

- It reads the integer input provided by the user using the nextInt() method of the Scanner class and stores it in the variable number.

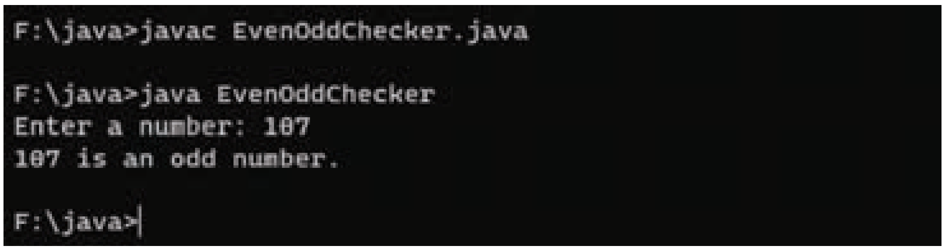
4 The program then checks whether the entered number is even or odd using the modulo operator %.

- If the remainder of number divided by 2 is equal to 0, then the number is even.
- If the remainder is not equal to 0, then the number is odd.

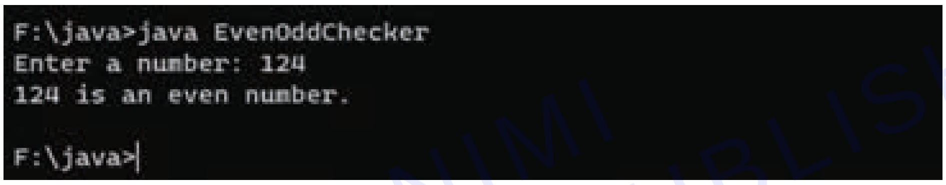
5 Depending on the result of the check, the program prints out a message indicating whether the number is even or odd.

6 Finally, the scanner object is closed to release system resources after it's no longer needed.

Output:



```
F:\java>javac EvenOddChecker.java

F:\java>java EvenOddChecker
Enter a number: 107
107 is an odd number.

F:\java>
```



```
F:\java>java EvenOddChecker
Enter a number: 124
124 is an even number.

F:\java>
```

TASK 2: **Checking if a person is eligible to vote**

```
import java.util.Scanner;
public class VotingEligibilityCheck {
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
System.out.print("Enter your age: ");
int age = scanner.nextInt();
if (age >= 18) {
System.out.println("You are eligible to vote.");
    } else {
System.out.println("You are not eligible to vote yet.");
    }
scanner.close}}
```

Explanation:

This Java program, named VotingEligibilityCheck, determines whether a person is eligible to vote based on their age. Here's a brief explanation of how it works:

1 The program starts by importing the Scanner class from the java.util package, which allows user input from the console.

2 The VotingEligibilityCheck class contains the main method, which serves as the entry point of the program.

3 Inside the main method:

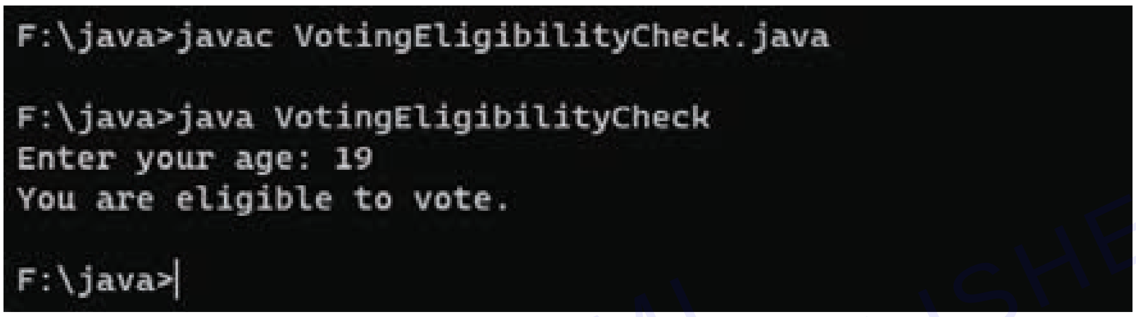
- It creates a new Scanner object named scanner to read input from the console.
- It prompts the user to enter their age by displaying the message "Enter your age: ".
- It reads the integer input provided by the user using the nextInt() method of the Scanner class and stores it in the variable age.

4. The program then checks whether the entered age is greater than or equal to 18, the legal voting age in many countries.

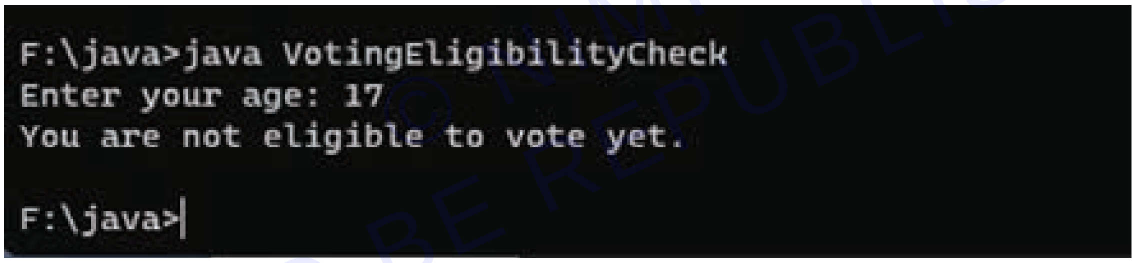
- If the age is 18 or older, it prints "You are eligible to vote."
- If the age is less than 18, it prints "You are not eligible to vote yet."

5. Finally, the scanner object is closed to release system resources after it's no longer needed.

Output:



```
F:\java>javac VotingEligibilityCheck.java

F:\java>java VotingEligibilityCheck
Enter your age: 19
You are eligible to vote.

F:\java>
```



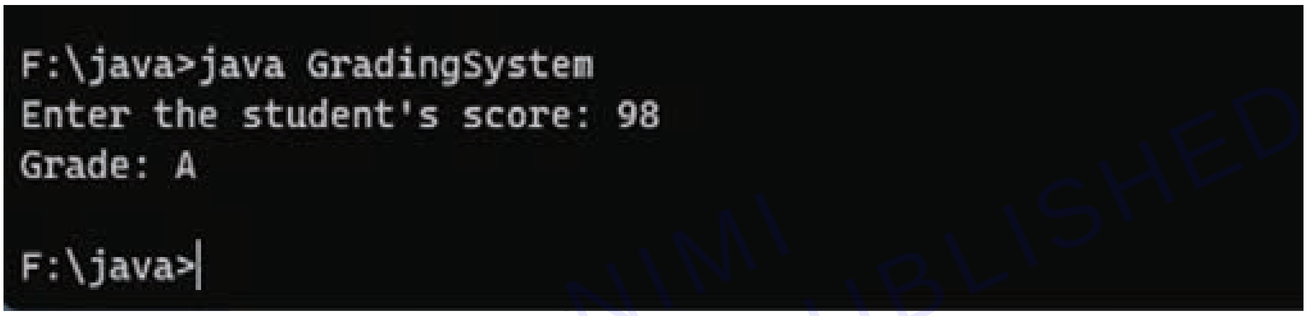
```
F:\java>java VotingEligibilityCheck
Enter your age: 17
You are not eligible to vote yet.

F:\java>
```

— — — — —

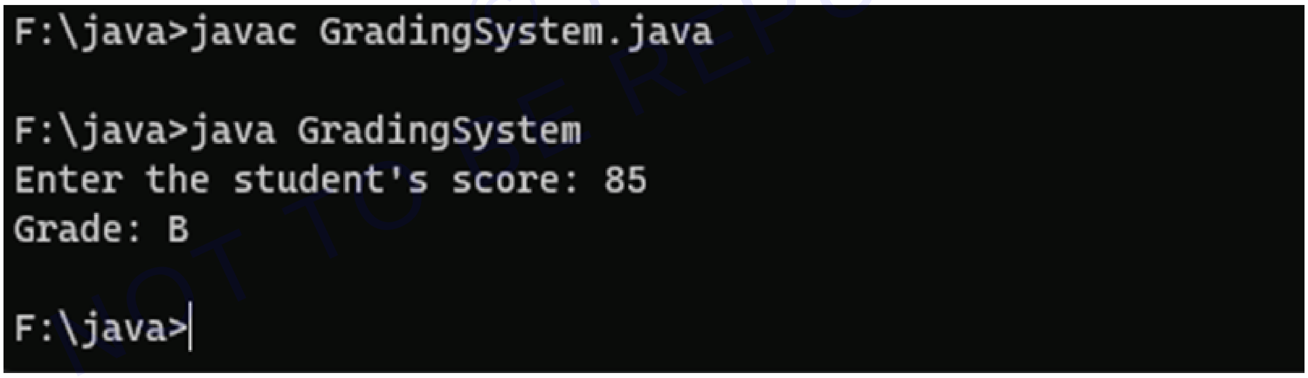
TASK 3: **Grading system**

```
// Grading System
import java.util.Scanner;
public class GradingSystem {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the console
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter the student's score
        System.out.print("Enter the student's score: ");
        // Read the integer input provided by the user
        int score = scanner.nextInt();
        // Determine the grade based on the score and print the result
        if (score >= 90) {
            System.out.println("Grade: A");
        } else if (score >= 80) {
            System.out.println("Grade: B");
        } else if (score >= 70) {
            System.out.println("Grade: C");
        } else if (score >= 60) {
            System.out.println("Grade: D");
        } else {
            System.out.println("Grade: F");
        }
        // Close the scanner object to release system resources
        scanner.close();
    }
}
```

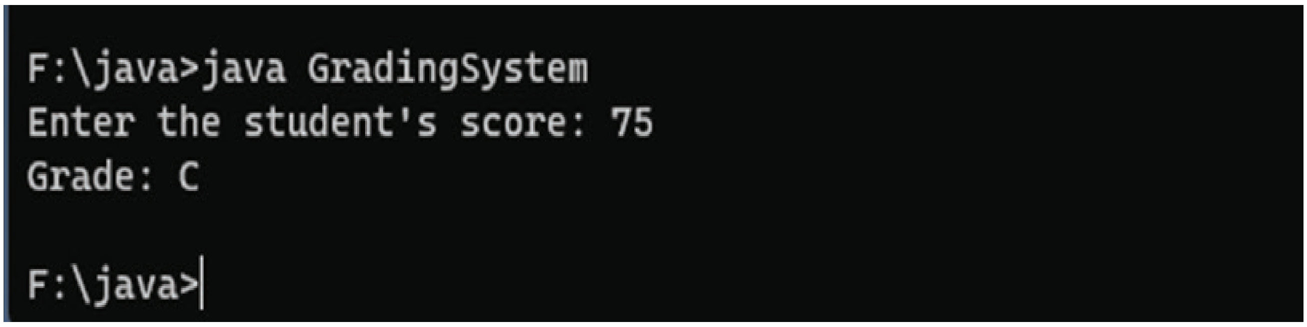
Output:



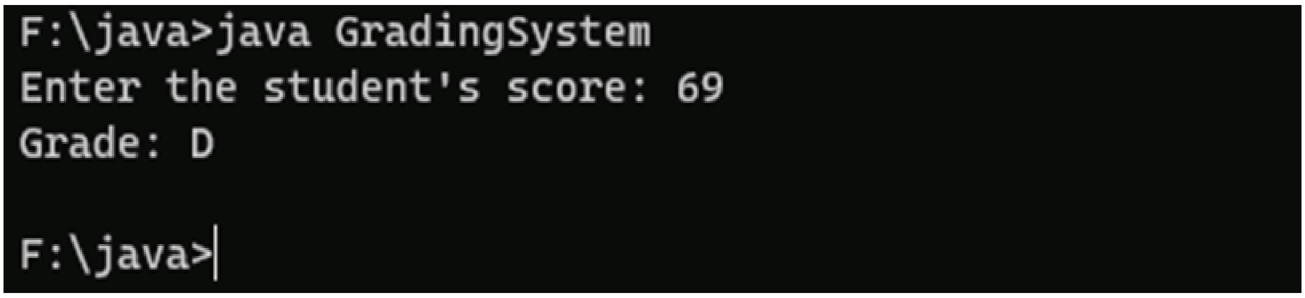
```
F:\java>java GradingSystem
Enter the student's score: 98
Grade: A

F:\java>
```



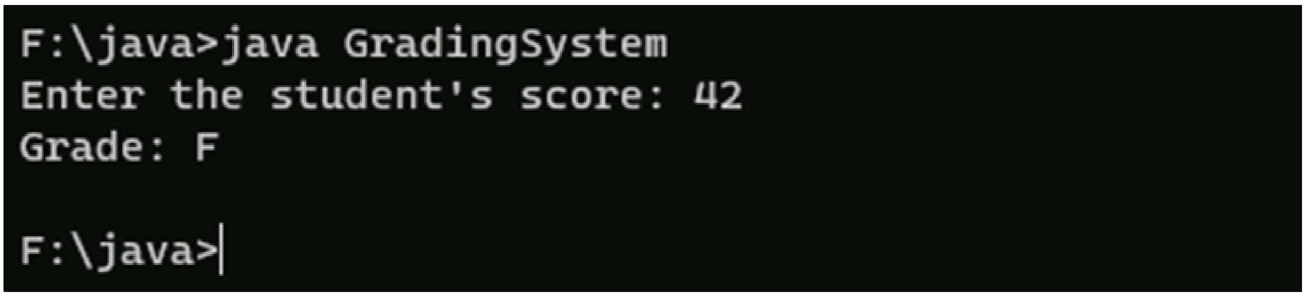
```
F:\java>javac GradingSystem.java

F:\java>java GradingSystem
Enter the student's score: 85
Grade: B

F:\java>
```



```
F:\java>java GradingSystem
Enter the student's score: 75
Grade: C

F:\java>
```



```
F:\java>java GradingSystem
Enter the student's score: 69
Grade: D

F:\java>
```



```
F:\java>java GradingSystem
Enter the student's score: 42
Grade: F

F:\java>
```

— — — — —

TASK 4: **Check if a year is a century year**

```
import java.util.Scanner;
public class CenturyYearCheck {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the console
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter a year
        System.out.print("Enter a year: ");
        // Read the integer input provided by the user
        int year = scanner.nextInt();
        // Check if the year is divisible by 100 and print the result
        if (year % 100 == 0) {
            System.out.println(year + " is a century year.");
        } else {
            System.out.println(year + " is not a century year.");
        }
        // Close the scanner object to release system resources
        scanner.close();
    }
}
```

Explanation:

1 The program begins by importing the Scanner class from the java.util package to allow user input from the console.

2 The CenturyYearCheck class contains the main method, which serves as the entry point of the program.

3 Inside the main method:

- It creates a new Scanner object named scanner to read input from the console.

- It prompts the user to enter a year by displaying the message “Enter a year: “.
- It reads the integer input provided by the user using the nextInt() method of the Scanner class and stores it in the variable year.

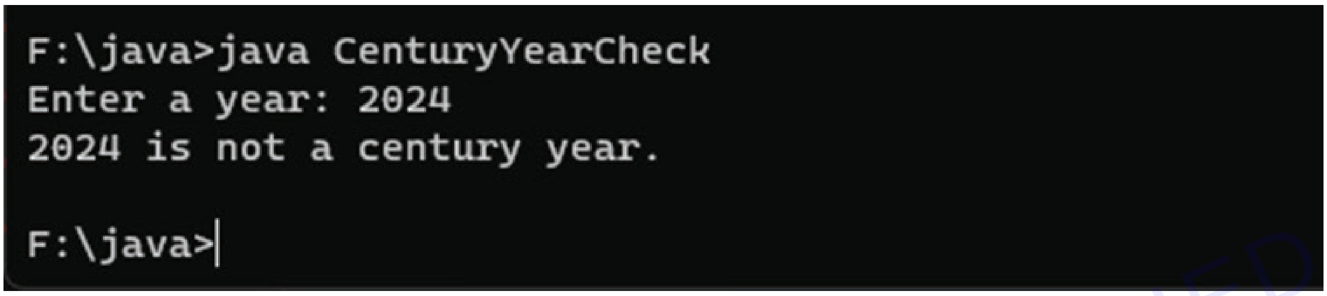
4 The program then checks whether the entered year is divisible by 100 without leaving a remainder:

- If the remainder of year divided by 100 is equal to 0, then the year is a century year.
- If the remainder is not equal to 0, then the year is not a century year.

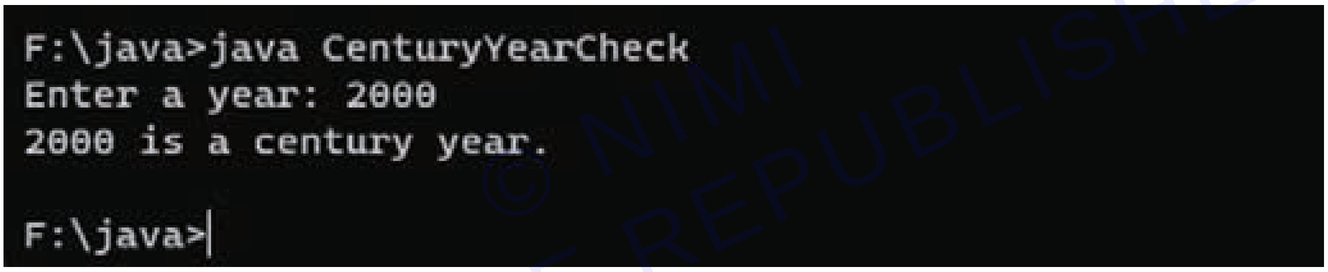
5 Depending on the result of the check, the program prints out a message indicating whether the year is a century year or not.

6 Finally, the scanner object is closed to release system resources after it’s no longer needed.

Output:



```
F:\java>java CenturyYearCheck
Enter a year: 2024
2024 is not a century year.

F:\java>
```



```
F:\java>java CenturyYearCheck
Enter a year: 2000
2000 is a century year.

F:\java>
```

EXERCISE 86 : Use the switch statement

Objectives

At the end of this exercise you shall be able to

- know the syntax of switch statement and its use
- develop Java programs using switch statement.

Requirements

Tools/Materials

- PC/Laptop with Window OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

Procedure

A switch statement in Java provides an alternative way to express a multi-branch decision based on the value of an expression. It's often used when you have a single variable or expression whose value needs to be tested against multiple conditions.

```
 switch (expression) {
    case value1:
        // Code block executed if expression equals value1
        break;
    case value2:
        // Code block executed if expression equals value2
        break;
    // Additional cases as needed
    default:
        // Code block executed if expression doesn't match any case
        break;
}
```

TASK 1: **Select an option using a switch statement**

```
import java.util.Scanner;
public class SimpleMenu {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.println("Select an option:");
System.out.println("1. Print Hello");
System.out.println("2. Print World");
System.out.println("3. Exit");
int choice = scanner.nextInt();
switch (choice) {
case 1:
System.out.println("Hello");
break;
case 2:
System.out.println("World");
break;
case 3:
System.out.println("Exiting the program.");
break;
default:
System.out.println("Invalid choice");
        }
scanner.close();
    }
}
```

Explanation :

This Java program, named SimpleMenu, presents a basic menu system that allows users to select from a list of options and performs actions based on their selection. Here's how it works:

1 The program starts by importing the Scanner class from the java.util package to allow user input from the console.

2 The SimpleMenu class contains the main method, which serves as the entry point of the program.

3 Inside the main method:

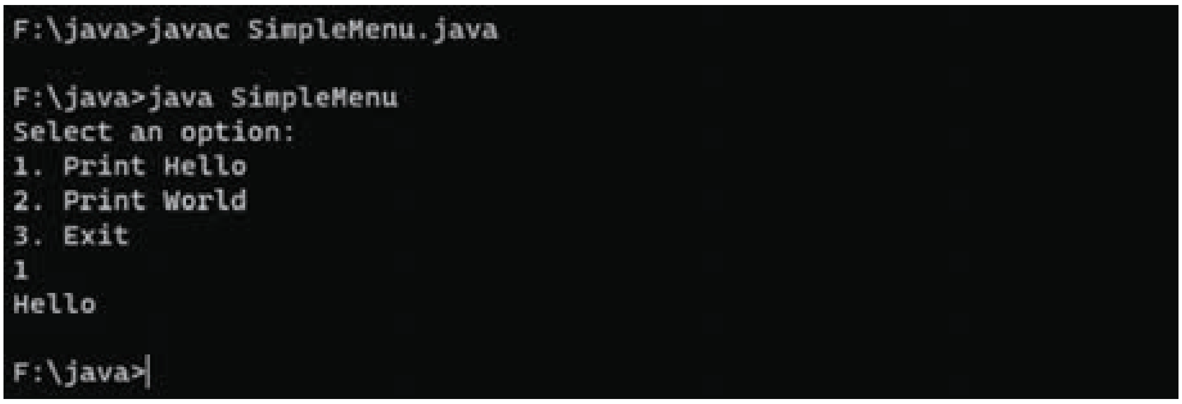
- It creates a new Scanner object named scanner to read input from the console.
- It displays a menu to the user, prompting them to select an option:
- Option 1: Print "Hello"
- Option 2: Print "World"
- Option 3: Exit the program
- It reads the integer input provided by the user using the nextInt() method of the Scanner class and stores it in the variable choice.

4 The program then uses a switch statement to perform different actions based on the value of choice:

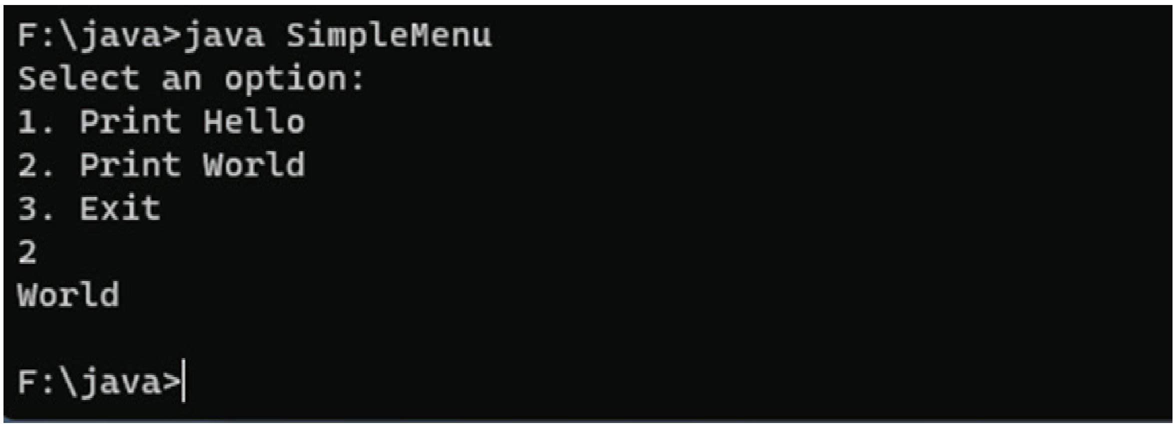
- If the user selects 1, it prints "Hello" to the console.
- If the user selects 2, it prints "World" to the console.
- If the user selects 3, it prints "Exiting the program." and terminates.
- If the user selects any other value, it prints "Invalid choice" to the console.

5 After executing the appropriate action, the program closes the scanner object to release system resources.

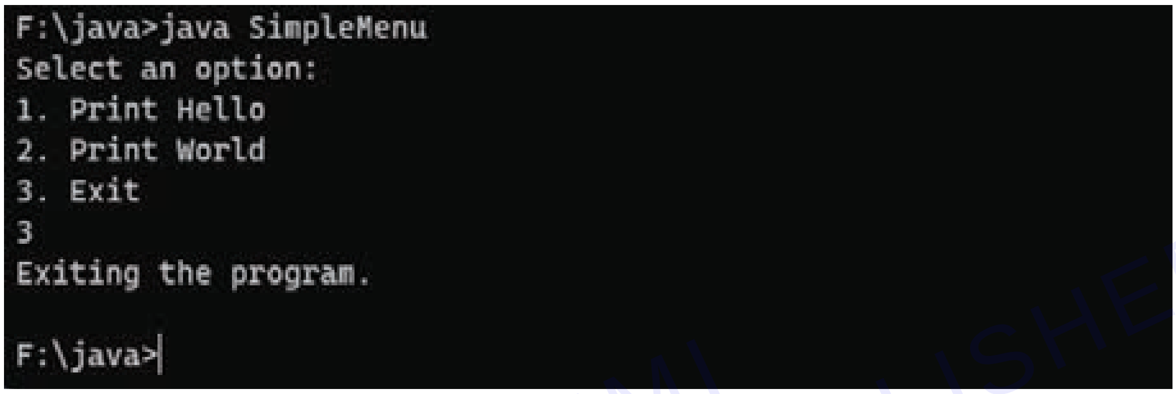
Output:



```
F:\java>javac SimpleMenu.java

F:\java>java SimpleMenu
Select an option:
1. Print Hello
2. Print World
3. Exit
1
Hello

F:\java>
```



```
F:\java>java SimpleMenu
Select an option:
1. Print Hello
2. Print World
3. Exit
2
World

F:\java>
```



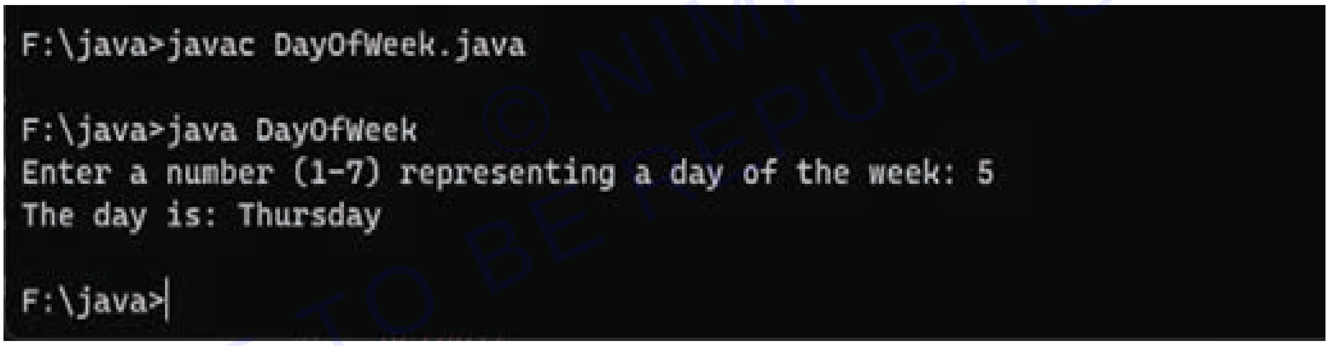
```
F:\java>java SimpleMenu
Select an option:
1. Print Hello
2. Print World
3. Exit
3
Exiting the program.

F:\java>
```

— — — — —

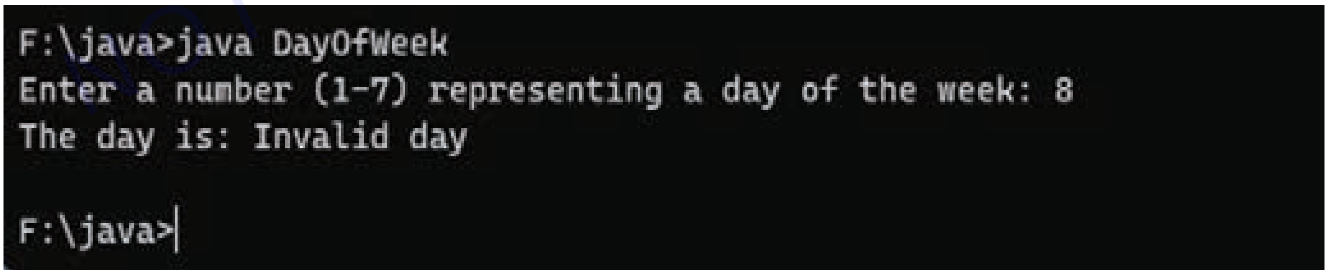
TASK 2 : **Day of the week using a switch statement**

```
import java.util.Scanner;
public class DayOfWeek {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number (1-7) representing a day of the week: ");
int day = scanner.nextInt();
String dayName;
switch (day) {
case 1:
dayName = "Sunday";
break;
case 2:
dayName = "Monday";
break;
case 3:
dayName = "Tuesday";
break;
case 4:
dayName = "Wednesday";
break;
case 5:
dayName = “Thursday”;
break;
case 6:
dayName = “Friday”;
break;
case 7:
dayName = “Saturday”;
break;
default:
dayName = “Invalid day”;
        }
System.out.println(“The day is: “ + dayName);
scanner.close();
    }
}
```

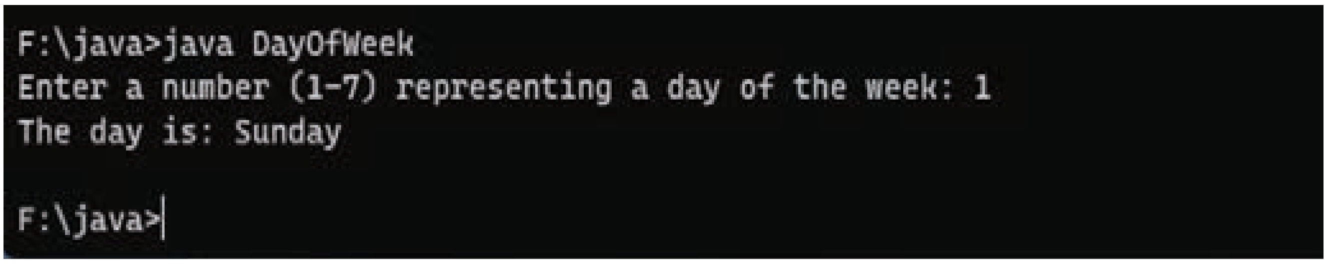
Output:



```
F:\java>javac DayOfWeek.java

F:\java>java DayOfWeek
Enter a number (1-7) representing a day of the week: 5
The day is: Thursday

F:\java>
```



```
F:\java>java DayOfWeek
Enter a number (1-7) representing a day of the week: 8
The day is: Invalid day

F:\java>
```



```
F:\java>java DayOfWeek
Enter a number (1-7) representing a day of the week: 1
The day is: Sunday

F:\java>
```

EXERCISE 87 : Use the Do … While and While – do loops

Objectives

At the end of this exercise you shall be able to

- know the syntax of while and do…while loops and its use
- develop Java programs using while and do…while loops.

Requirements

Tools/Materials

- PC/Laptop with Window OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

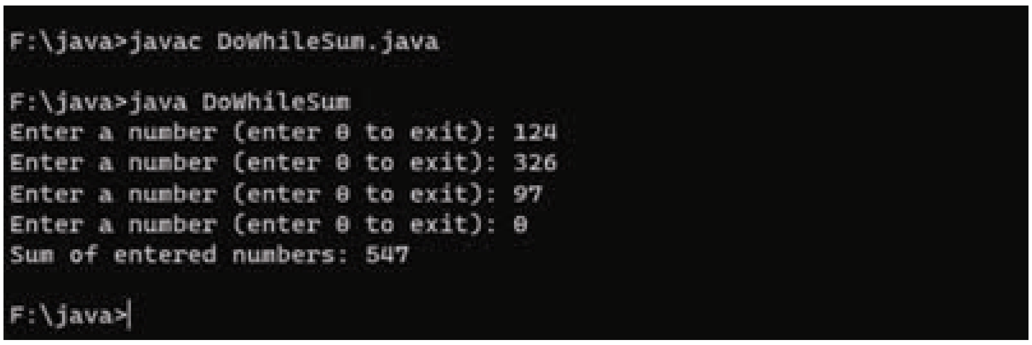
Procedure

Both do-while and while loops are control flow structures in Java used for repetitive execution of a block of code. They differ primarily in when the loop condition is evaluated.

TASK 1: **Sum of numbers using a do-while loop**

```
import java.util.Scanner;
public class DoWhileSum {
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
int sum = 0;
int number;
do {
System.out.print("Enter a number (enter 0 to exit): ");
number = scanner.nextInt();
sum += number;
   } while (number != 0);
System.out.println("Sum of entered numbers: " + sum);
scanner.close();
    }
}
```

Output:



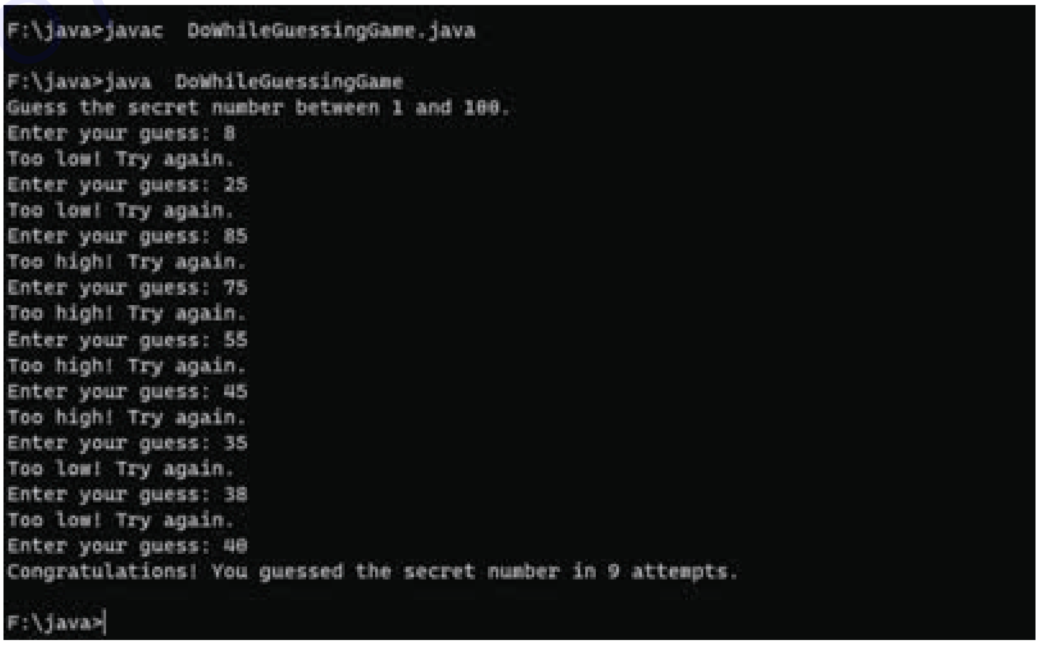
```
F:\java>javac DoWhileSum.java

F:\java>java DoWhileSum
Enter a number (enter 0 to exit): 124
Enter a number (enter 0 to exit): 326
Enter a number (enter 0 to exit): 97
Enter a number (enter 0 to exit): 0
Sum of entered numbers: 547

F:\java>
```

This program uses a do-while loop to repeatedly prompt the user to enter numbers until they enter 0. It calculates and prints the sum of all entered numbers

TASK 2: **Guessing Game using do-while loop**

```
import java.util.Random;
importjava.util.Scanner;
public class DoWhileGuessingGame {
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
  Random random = new Random();
intsecretNumber = random.nextInt(100) + 1;
int guess;
int attempts = 0;
System.out.println("Guess the secret number between 1 and 100.");
do {
System.out.print("Enter your guess: ");
guess = scanner.nextInt();
attempts++;
if (guess <secretNumber) {
System.out.println("Too low! Try again.");
       } else if (guess >secretNumber) {
System.out.println("Too high! Try again.");
       } else {
System.out.println("Congratulations! You guessed the secret number in " + attempts + " attempts.");
       }
   } while (guess != secretNumber);
scanner.close();
}
}
```

Output:

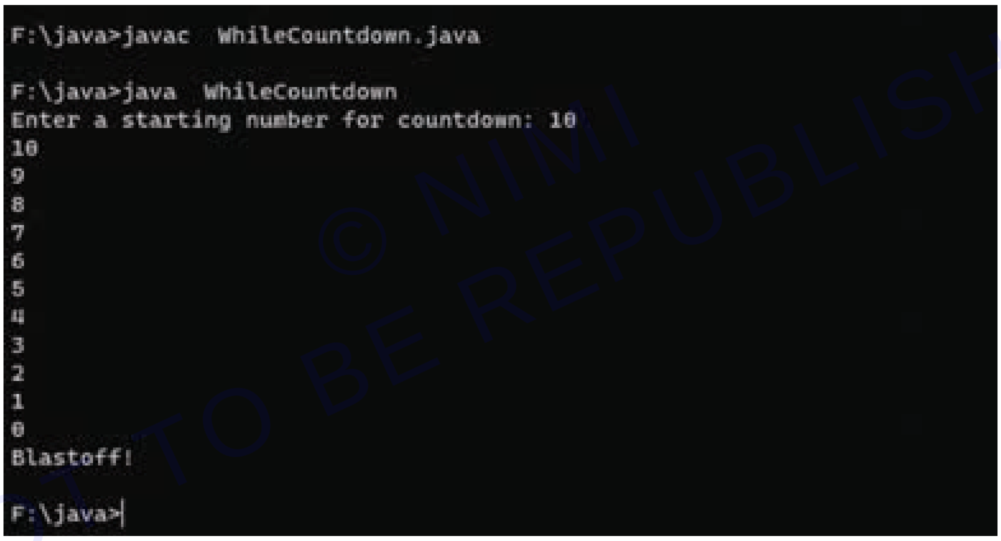


```
F:\java>javac  DoWhileGuessingGame.java

F:\java>java  DoWhileGuessingGame
Guess the secret number between 1 and 100.
Enter your guess: 8
Too low! Try again.
Enter your guess: 25
Too low! Try again.
Enter your guess: 85
Too high! Try again.
Enter your guess: 75
Too high! Try again.
Enter your guess: 55
Too high! Try again.
Enter your guess: 45
Too high! Try again.
Enter your guess: 35
Too low! Try again.
Enter your guess: 38
Too low! Try again.
Enter your guess: 40
Congratulations! You guessed the secret number in 9 attempts.

F:\java>
```

TASK 3: **Countdown using a while loop**

```
import java.util.Scanner;
public class WhileCountdown {
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
System.out.print("Enter a starting number for countdown: ");
int count = scanner.nextInt();
while (count >= 0) {
System.out.println(count);
count--;
        }
System.out.println("Blastoff!");
scanner.close();
    }
}
```

Output:



```
F:\java>javac  WhileCountdown.java

F:\java>java  WhileCountdown
Enter a starting number for countdown: 10
10
9
8
7
6
5
4
3
2
1
0
Blastoff!

F:\java>
```

This program uses a while loop to create a countdown starting from a user-specified number. It prints the countdown and then displays "Blastoff!" when the countdown reaches 0.

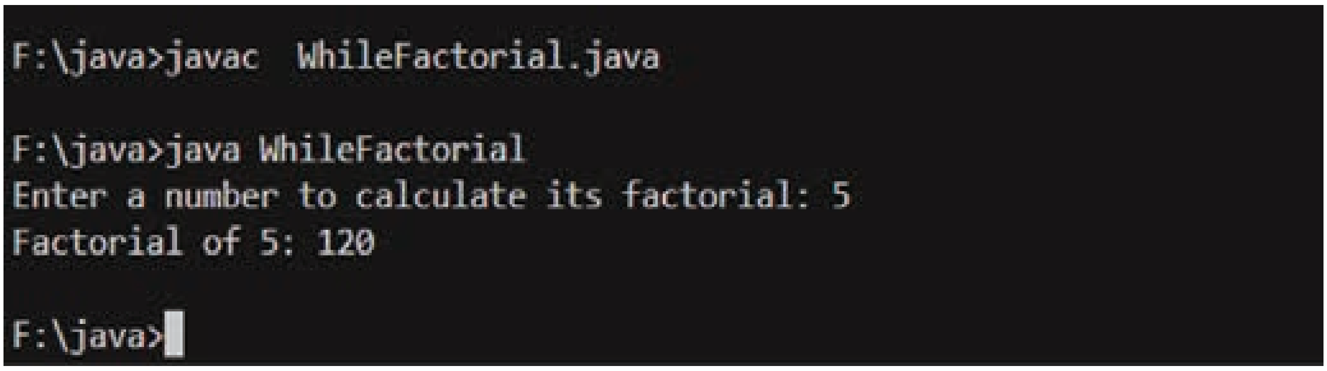
— — — — —

TASK 4: **Factorial Calculation using while loop**

```
import java.util.Scanner;
public class WhileFactorial {
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number to calculate its factorial: ");
int number = scanner.nextInt();
int factorial = 1;
int i = 1;
while (i<= number) {
factorial *= i;
i++;
```

```
        }
System.out.println("Factorial of " + number + ": " + factorial);
scanner.close();
    }
}
```

Output:



```
F:\java>javac  WhileFactorial.java

F:\java>java WhileFactorial
Enter a number to calculate its factorial: 5
Factorial of 5: 120

F:\java>
```

EXERCISE 88 : Use the For Loop

Objectives

At the end of this exercise you shall be able to

- know the syntax of for loop and its use
- develop Java programs using for loop.

Requirements

Tools/Materials

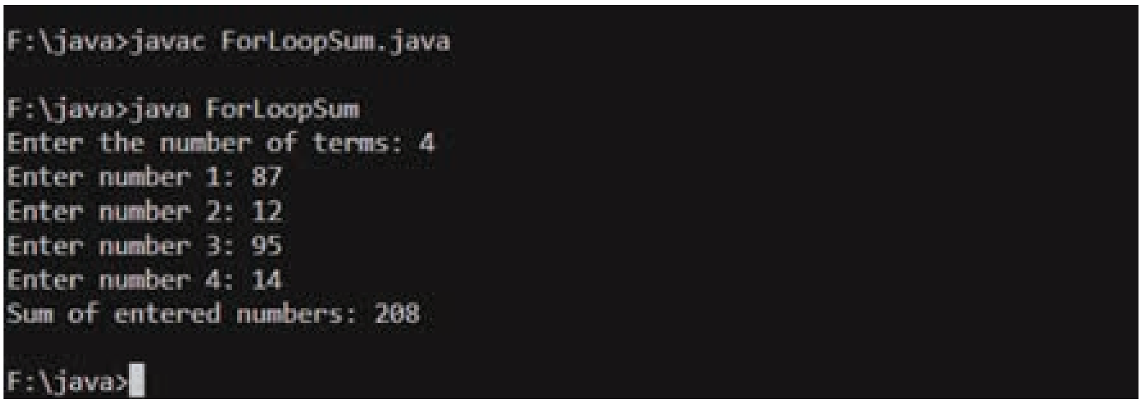
- PC/Laptop with Window OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

Procedure

TASK 1: **Sum of 'n' Numbers using for loop**

```
import java.util.Scanner;
public class ForLoopSum {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of terms: ");
int n = scanner.nextInt();
int sum = 0;
for (int i = 1; i<= n; i++) {
System.out.print("Enter number " + i + ": ");
int number = scanner.nextInt();
sum += number;
    }
System.out.println("Sum of entered numbers: " + sum);
scanner.close();
  }
}
```

Output:



```
F:\java>javac ForLoopSum.java

F:\java>java ForLoopSum
Enter the number of terms: 4
Enter number 1: 87
Enter number 2: 12
Enter number 3: 95
Enter number 4: 14
Sum of entered numbers: 208

F:\java>
```

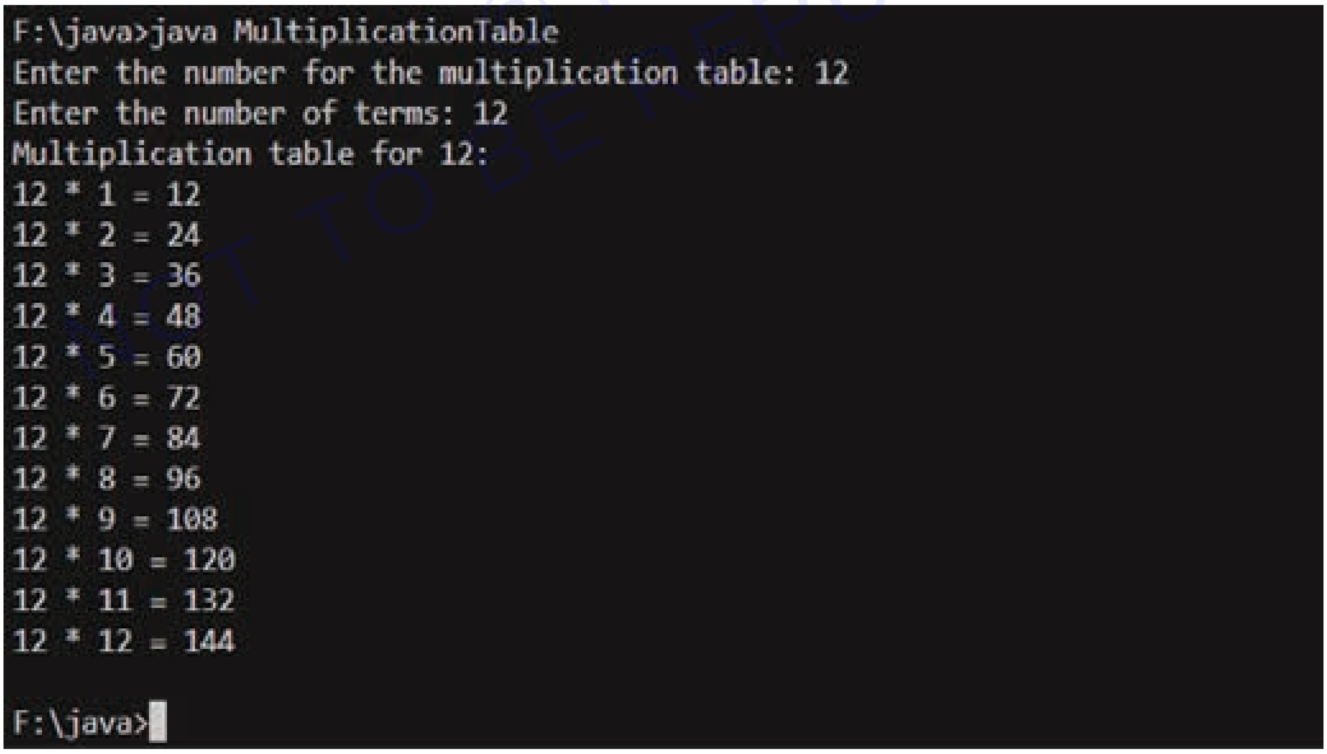
This program uses a for loop to calculate the sum of a specified number of terms. It prompts the user to enter numbers for each term and then calculates the sum of those numbers.

— — — — —

TASK 2: **Multiplication Table using for loop**

```
import java.util.Scanner;
public class MultiplicationTable {
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number for the multiplication table: ");
int number = scanner.nextInt();
System.out.print("Enter the number of terms: ");
int n = scanner.nextInt();
System.out.println("Multiplication table for " + number + ":");
for (int i = 1; i<= n; i++) {
System.out.println(number + " * " + i + " = " + (number * i));
    }
scanner.close();
  }
}
```

Output:



```
F:\java>java MultiplicationTable
Enter the number for the multiplication table: 12
Enter the number of terms: 12
Multiplication table for 12:
12 * 1 = 12
12 * 2 = 24
12 * 3 = 36
12 * 4 = 48
12 * 5 = 60
12 * 6 = 72
12 * 7 = 84
12 * 8 = 96
12 * 9 = 108
12 * 10 = 120
12 * 11 = 132
12 * 12 = 144

F:\java>
```

EXERCISE 89 : Use the Break and Continue Keywords

Objectives

At the end of this exercise you shall be able to

- know the syntax break and continue statements and its use
- develop Java programs using break and continue statements.

Requirements

Tools/Materials

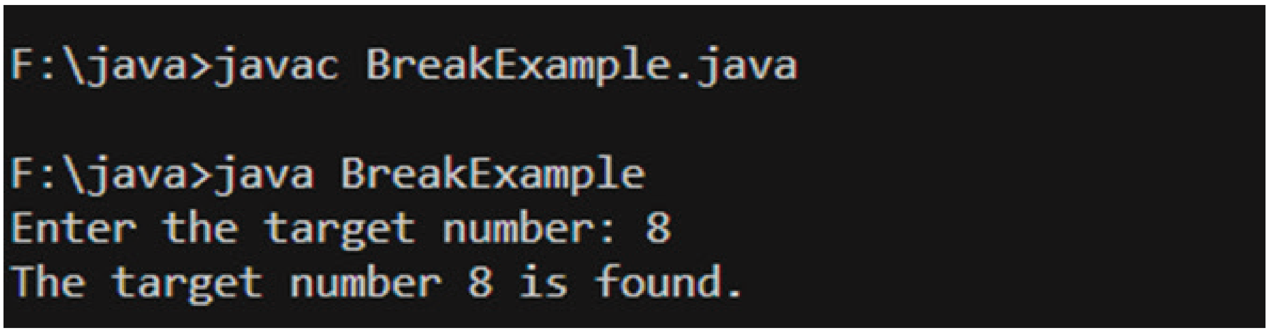
- PC/Laptop with Window OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

Procedure

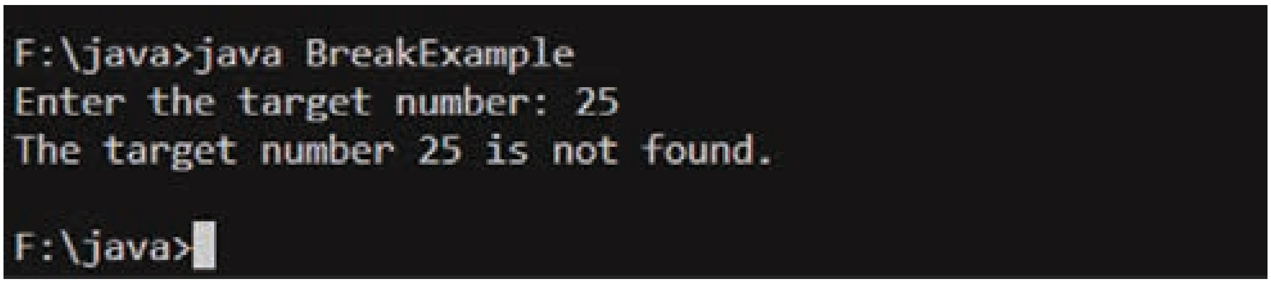
TASK 1: **Using break in a Loop**

```
import java.util.Scanner;
public class BreakExample {
 public static void main(String[] args) {
        // This program searches for a specific number in a loop
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the target number: ");
        int target = scanner.nextInt();
        boolean found = false;
        // Search for the target number in a loop
        for (int i = 1; i <= 10; i++) {
            if (i == target) {
                found = true;
                break;  // Exit the loop when the target number is found
            }
        }
        if (found) {
            System.out.println("The target number " + target + " is found.");
        } else {
            System.out.println("The target number " + target + " is not found.");
        }
        scanner.close();
    }
}
```

Output:



```
F:\java>javac BreakExample.java

F:\java>java BreakExample
Enter the target number: 8
The target number 8 is found.
```



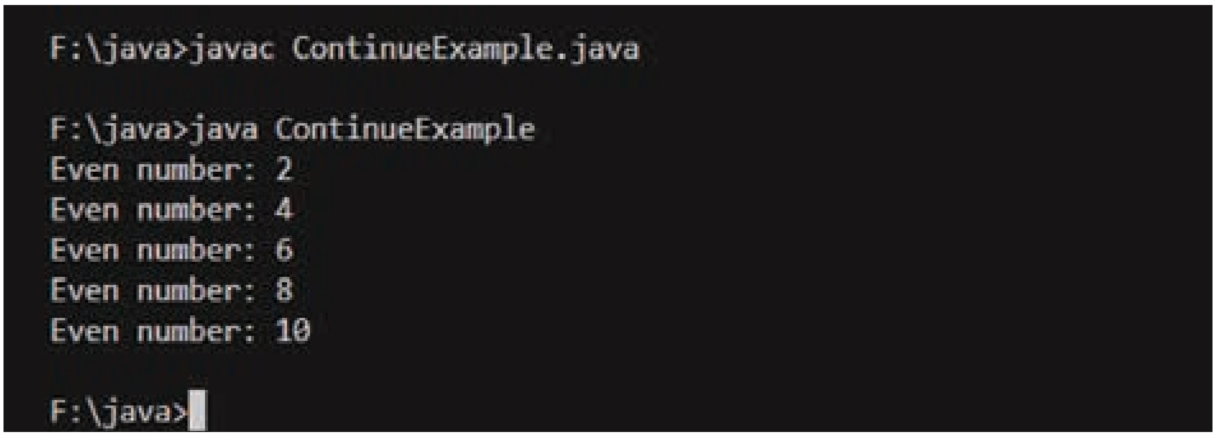
```
F:\java>java BreakExample
Enter the target number: 25
The target number 25 is not found.

F:\java>
```

— — — — —

TASK 2: **Using continue in a Loop**

```
public class ContinueExample {
    public static void main(String[] args) {
        // This program prints even numbers in a loop, skipping odd numbers

        // Print even numbers in a loop
        for (int i = 1; i <= 10; i++) {
            if (i % 2 != 0) {
                // Skip odd numbers and continue to the next iteration
                continue;
            }
            System.out.println("Even number: " + i);
        }
    }
}
```

Output:



```
F:\java>javac ContinueExample.java

F:\java>java ContinueExample
Even number: 2
Even number: 4
Even number: 6
Even number: 8
Even number: 10

F:\java>
```

EXERCISE 90 : Use the JAVA numbers class methods

Objectives

At the end of this exercise you shall be able to

- know more about the use of number class methods in Java
- develop Java programs using number class methods.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

Procedure

In Java, the Number class is an abstract class that serves as the superclass for all the numeric wrapper classes. The most commonly used numeric wrapper classes are:

1 Byte

2 Short

3 Integer

4 Long

5 Float

6 Double

All these classes extend the Number class and provide methods to convert the primitive types (byte, short, int, long, float, double) into their respective wrapper objects and vice versa. They also inherit methods from the Number class.

1 Byte

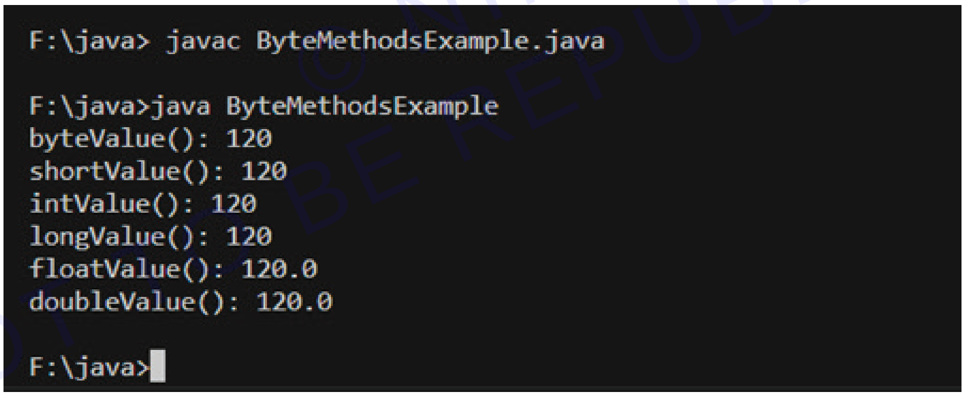
- Represents an 8-bit signed integer.
- Methods include: byteValue(), shortValue(), intValue(), longValue(), floatValue(), doubleValue().

Task_Byte : Here's an example program demonstrating the use of methods inherited from the Number class for the Byte class:

```
public class ByteMethodsExample {
   public static void main(String[] args) {
       // Example using Byte class
       Byte byteValue = 120;
      // byteValue()
       byte byteResult = byteValue.byteValue();
       System.out.println("byteValue(): " + byteResult);
       // shortValue()
       short shortResult = byteValue.shortValue();
       System.out.println("shortValue(): " + shortResult);
```

```
        // intValue()
        int intResult = byteValue.intValue();
        System.out.println("intValue(): " + intResult);

        // longValue()
        long longResult = byteValue.longValue();
        System.out.println("longValue(): " + longResult);

        // floatValue()
        float floatWResult = byteValue.floatValue();
        System.out.println("floatValue(): " + floatResult);

        // doubleValue()
        double doubleResult = byteValue.doubleValue();
        System.out.println("doubleValue(): " + doubleResult);
    }
}
```

Output :



```
F:\java> javac ByteMethodsExample.java

F:\java>java ByteMethodsExample
byteValue(): 120
shortValue(): 120
intValue(): 120
longValue(): 120
floatValue(): 120.0
doubleValue(): 120.0

F:\java>
```

2 Short

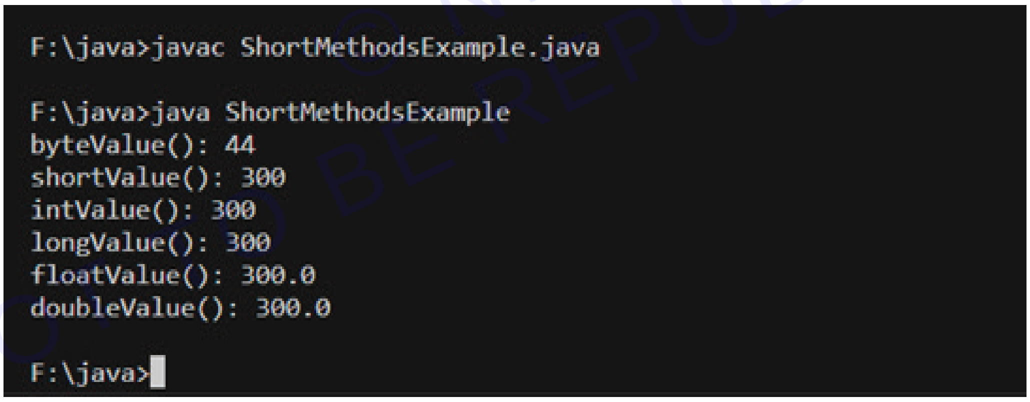
- **Represents** a 16-bit signed integer.
- Methods include: shortValue(), intValue(), longValue(), floatValue(), doubleValue().

Task_Short : Here's an example program demonstrating the use of methods inherited from the Number class for the Short class:

```
public class ShortMethodsExample {
    public static void main(String[] args) {
        // Example using Short class
        Short shortValue = 300;

        // byteValue()
        byte byteResult = shortValue.byteValue();
        System.out.println("byteValue(): " + byteResult);
```

```
            // shortValue()
            short shortResult = shortValue.shortValue();
            System.out.println("shortValue(): " + shortResult);

            // intValue()
            int intResult = shortValue.intValue();
            System.out.println("intValue(): " + intResult);

            // longValue()
            long longResult = shortValue.longValue();
            System.out.println("longValue(): " + longResult);
           // floatValue()
            float floatResult = shortValue.floatValue();
            System.out.println("floatValue(): " + floatResult);

            // doubleValue()
            double doubleResult = shortValue.doubleValue();
            System.out.println("doubleValue(): " + doubleResult);
        }
    }
```

Output:



```
F:\java>javac ShortMethodsExample.java

F:\java>java ShortMethodsExample
byteValue(): 44
shortValue(): 300
intValue(): 300
longValue(): 300
floatValue(): 300.0
doubleValue(): 300.0

F:\java>
```

3 **Integer**

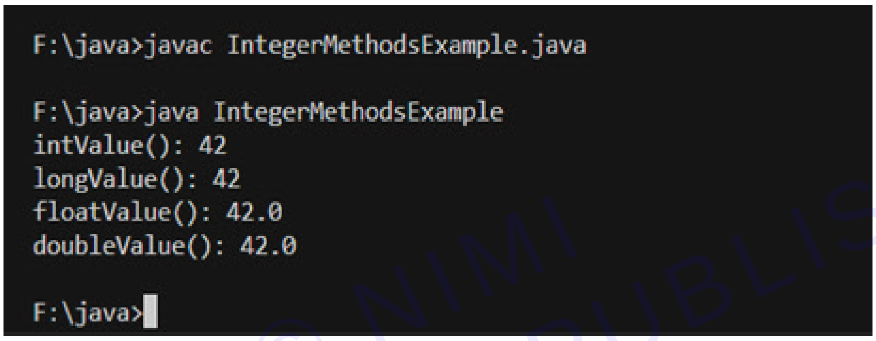
- Represents a 32-bit signed integer.
- Methods include: intValue(), longValue(), floatValue(), doubleValue().

Task_Integer: Here's an example program demonstrating the use of methods inherited from the Number class for the Integer class:

```
public class IntegerMethodsExample {
 public static void main(String[] args) {
   // Example using Integer class
  Integer intValue = 42;

   // intValue()
   int intResult = intValue.intValue();
   System.out.println("intValue(): " + intResult);
```

```
        // longValue()
        long longResult = intValue.longValue();
        System.out.println("longValue(): " + longResult);

        // floatValue()
        float floatResult = intValue.floatValue();
        System.out.println("floatValue(): " + floatResult);

        // doubleValue()
        double doubleResult = intValue.doubleValue();
        System.out.println("doubleValue(): " + doubleResult);
    }
}
```

Output :



```
F:\java>javac IntegerMethodsExample.java

F:\java>java IntegerMethodsExample
intValue(): 42
longValue(): 42
floatValue(): 42.0
doubleValue(): 42.0

F:\java>
```

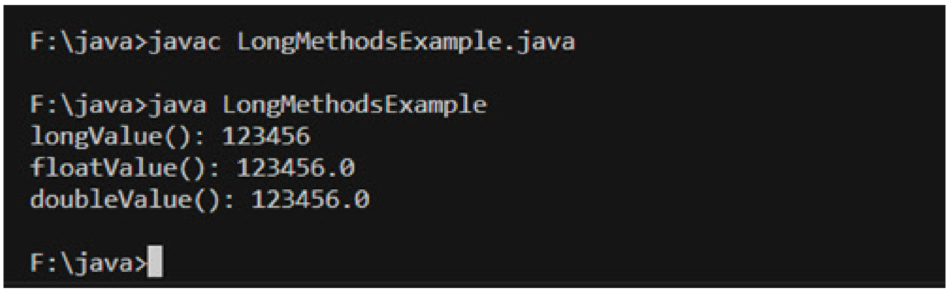
4 Long

- Represents a 64-bit signed integer.
- Methods include: longValue(), floatValue(), doubleValue().

Task_Long: Here's an example program demonstrating the use of methods inherited from the Number class for the Long class:

```
public class LongMethodsExample {
public static void main(String[] args) {
    // Example using Long class
    Long longValue = 123456L;

    // longValue()
    long longResult = longValue.longValue();
    System.out.println("longValue(): " + longResult);

    // floatValue()
    float floatResult = longValue.floatValue();
    System.out.println("floatValue(): " + floatResult);

    // doubleValue()
    double doubleResult = longValue.doubleValue();
    System.out.println("doubleValue(): " + doubleResult);
}
}
```

Output:



```
F:\java>javac LongMethodsExample.java

F:\java>java LongMethodsExample
longValue(): 123456
floatValue(): 123456.0
doubleValue(): 123456.0

F:\java>
```

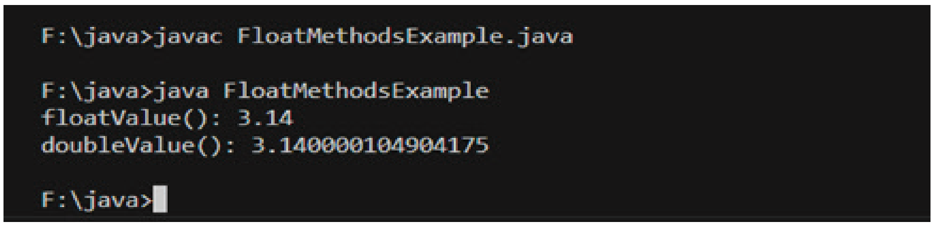
5 Float

- Represents a 32-bit IEEE 754 floating-point.
- Methods include: floatValue(), doubleValue().

Task_Float: Here's an example program demonstrating the use of methods inherited from the Number class for the Float class:

```
public class FloatMethodsExample {
 public static void main(String[] args) {
     // Example using Float class
     Float floatValue = 3.14f;

     // floatValue()
     float floatResult = floatValue.floatValue();
     System.out.println("floatValue(): " + floatResult);

     // doubleValue()
     double doubleResult = floatValue.doubleValue();
     System.out.println("doubleValue(): " + doubleResult);
 }
}
```

Output:



```
F:\java>javac FloatMethodsExample.java

F:\java>java FloatMethodsExample
floatValue(): 3.14
doubleValue(): 3.140000104904175

F:\java>
```

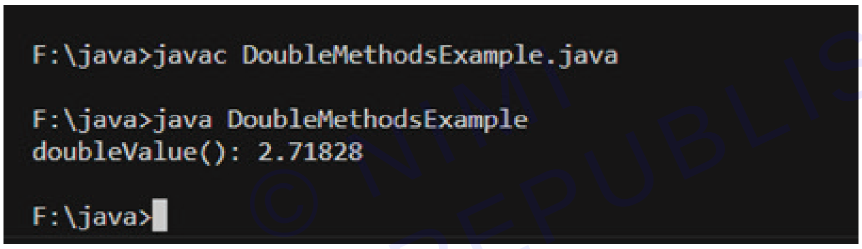
6 Double

- Represents a 64-bit IEEE 754 floating-point.
- Methods include: doubleValue().

Task_Double: Here's an example program demonstrating the use of the doubleValue() method for the Double class:

```
 public class DoubleMethodsExample {
public static void main(String[] args) {
    // Example using Double class
    Double doubleValue = 2.71828;

    // doubleValue()
    double doubleResult = doubleValue.doubleValue();
    System.out.println("doubleValue(): " + doubleResult);
  }
}
```

Output:



```
F:\java>javac DoubleMethodsExample.java

F:\java>java DoubleMethodsExample
doubleValue(): 2.71828

F:\java>
```

EXERCISE 91 : Use the JAVA character class methods

Objectives

At the end of this exercise you shall be able to

- know more about the use of character class methods in Java
- develop Java programs using character class methods.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

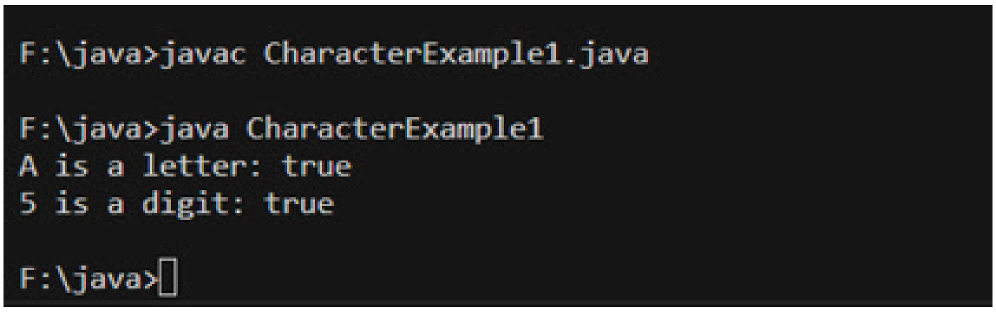
Procedure

Here are some examples demonstrating the use of various methods in the Character class in Java:

TASK 1: **Check if a Character is a Letter or Digit**

```
   public class CharacterExample1 {
    public static void main(String[] args) {
        char ch1 = 'A';
        char ch2 = '5';

        System.out.println(ch1 + " is a letter: " + Character.isLetter(ch1));
        System.out.println(ch2 + " is a digit: " + Character.isDigit(ch2));
    }
}
```

Output;



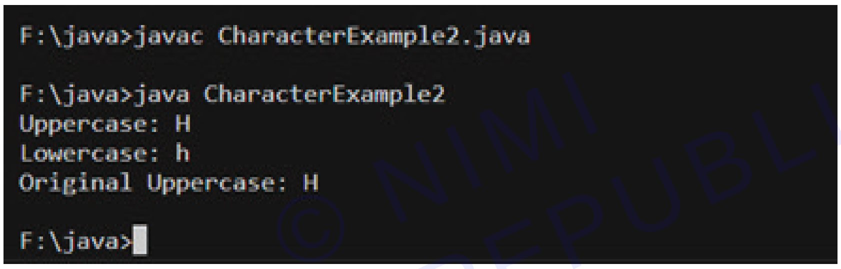
```
F:\java>javac CharacterExample1.java

F:\java>java CharacterExample1
A is a letter: true
5 is a digit: true

F:\java>
```

— — — — —

TASK 2: **Convert Uppercase to Lowercase and Vice Versa**

```
 public class CharacterExample2 {
   public static void main(String[] args) {
       char uppercaseChar = ‘H’;
       // Convert to lowercase
       char lowercaseChar = Character.toLowerCase(uppercaseChar);
       System.out.println(“Uppercase: “ + uppercaseChar);
       System.out.println(“Lowercase: “ + lowercaseChar);

       // Convert back to uppercase
       char originalUppercaseChar = Character.toUpperCase(lowercaseChar);
       System.out.println(“Original Uppercase: “ + originalUppercaseChar);
   }
}
```

Output:



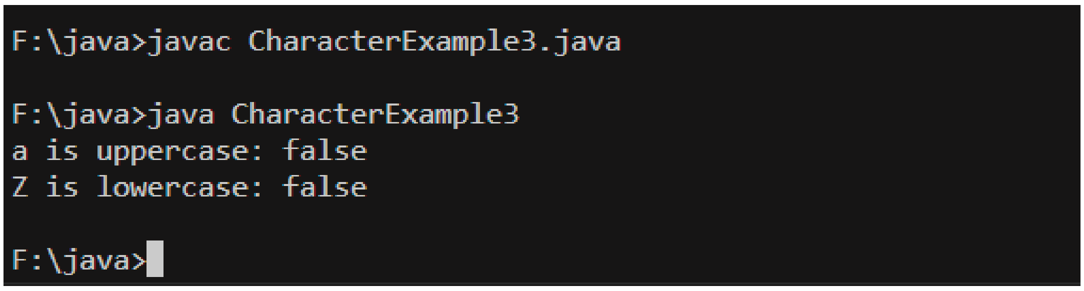
```
F:\java>javac CharacterExample2.java

F:\java>java CharacterExample2
Uppercase: H
Lowercase: h
Original Uppercase: H

F:\java>
```

— — — — —

TASK 3: **Check if a Character is Uppercase or Lowercase**

```
public class CharacterExample3 {
   public static void main(String[] args) {
       char ch1 = ‘a’;
       char ch2 = ‘Z’;

       System.out.println(ch1 + “ is uppercase: “ + Character.isUpperCase(ch1));
       System.out.println(ch2 + “ is lowercase: “ + Character.isLowerCase(ch2));
   }
}
```

Output:

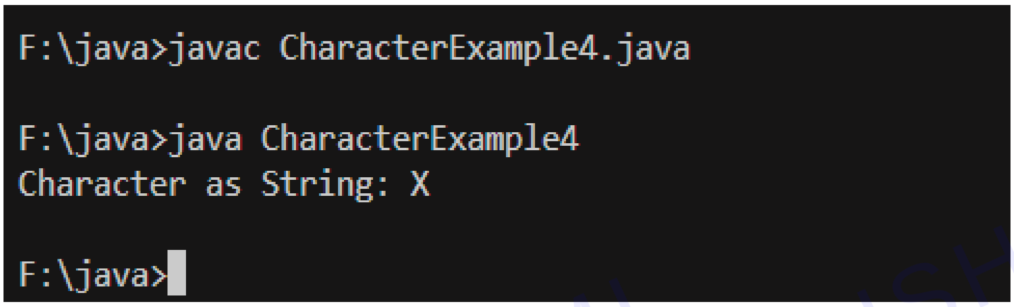


```
F:\java>javac CharacterExample3.java

F:\java>java CharacterExample3
a is uppercase: false
Z is lowercase: false

F:\java>
```

TASK 4: **Convert a Character to String**

```
public class CharacterExample4 {
    public static void main(String[] args) {
        char ch = 'X';

        String charString = Character.toString(ch);
        System.out.println("Character as String: " + charString);
    }
}
```

Output:

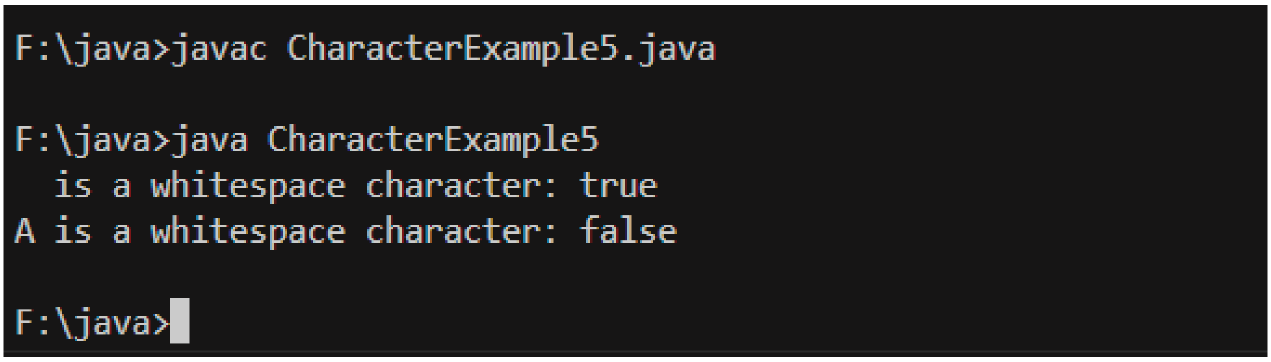


```
F:\java>javac CharacterExample4.java

F:\java>java CharacterExample4
Character as String: X

F:\java>
```

TASK 5: **Check if a character is a Whitespace Character**

```
public class CharacterExample5 {
    public static void main(String[] args) {
        char ch1 = ' ';
        char ch2 = 'A';

        System.out.println(ch1 + " is a whitespace character: " + Character.isWhitespace(ch1));
        System.out.println(ch2 + " is a whitespace character: " + Character.isWhitespace(ch2));
    }
}
```

Output:



```
F:\java>javac CharacterExample5.java

F:\java>java CharacterExample5
  is a whitespace character: true
A is a whitespace character: false

F:\java>
```

EXERCISE 92 : Use the JAVA string class methods

Objectives

At the end of this exercise you shall be able to

- know more about the use of string class methods in Java
- develop Java programs using string class methods.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

Procedure

The String class in Java provides numerous methods for working with strings. Here are a few examples demonstrating the use of some common methods of the String class:

TASK 1: **Concatenate Strings**

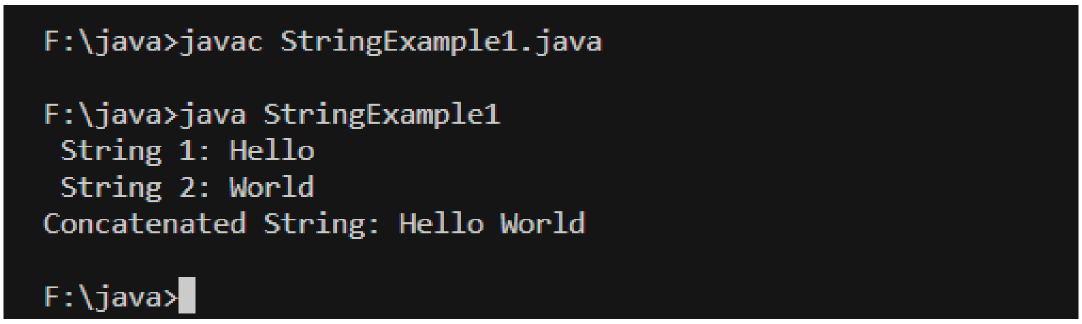
```
//Concatenate Strings
 public class StringExample1 {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "World";
        System.out.println(" String 1: " + str1);
        System.out.println(" String 2: " + str2);
        // Concatenation using the concat() method
String result = str1.concat(" " + str2);
        System.out.println("Concatenated String: " + result);
    }
}
```

Explanation:

- In Java, string concatenation is the process of combining two or more strings into a single string.
- The concat() method is one way to concatenate strings in Java. It appends one string to the end of another string.
- In the example, str1.concat(" " + str2) concatenates str1, a space, and str2.
- The result is a new string "Hello World", which is stored in the variable result.
- String concatenation using the concat() method creates a new string object containing the concatenated result. The original strings remain unchanged.
- The + operator can also be used for string concatenation in Java, as demonstrated in the println() statements. It works similarly to the concat() method.

- String concatenation is a common operation in Java when dealing with text processing, output formatting, and building dynamic strings for display or storage.

Output:



```
F:\java>javac StringExample1.java

F:\java>java StringExample1
 String 1: Hello
 String 2: World
Concatenated String: Hello World

F:\java>
```

— — — — —

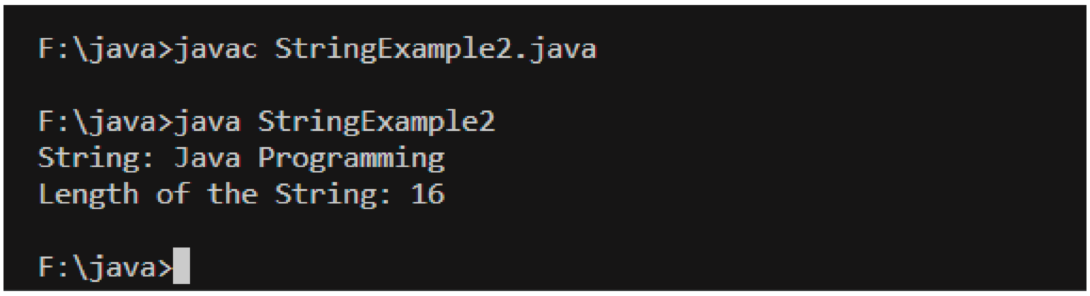
TASK 2: **String Length**

```
public class StringExample2 {
    public static void main(String[] args) {
        String str = "Java Programming";
        System.out.println("String: " + str);
        // Get the length of the string
        int length = str.length();
      System.out.println("Length of the String: " + length);
    }
}
```

Explanation:

- The length() method is a built-in method provided by the String class in Java.
- It returns an integer representing the number of characters in the string.
- The length includes all characters in the string, including whitespace characters such as spaces.
- In the example, the string "Java Programming" consists of 16 characters, including the space between "Java" and "Programming".
- The length() method is commonly used in string manipulation tasks, input validation, and text processing to determine the size of the string dynamically at runtime.
- It's important to note that the length of a string does not include the null character (\0) at the end of the string, as Java strings are null-terminated.

Output:

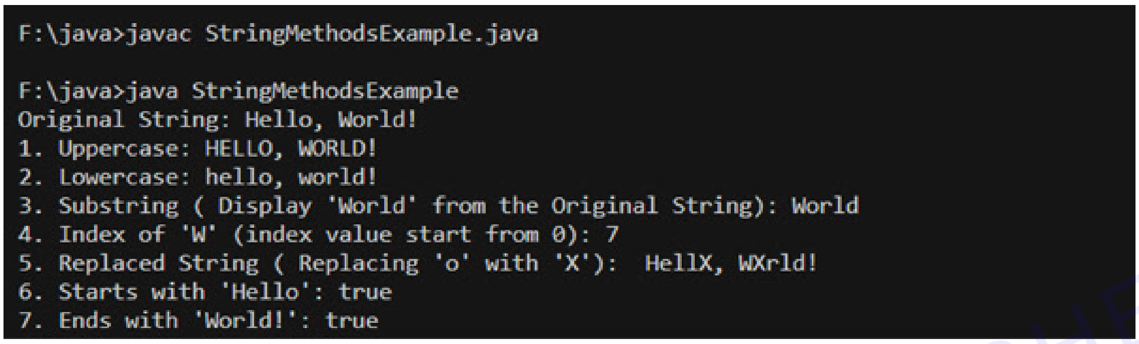


```
F:\java>javac StringExample2.java

F:\java>java StringExample2
String: Java Programming
Length of the String: 16

F:\java>
```

TASK 3: **Java program to perform various string methods in single program**

```
 public class StringMethodsExample {
   public static void main(String[] args) {
        // Example String
        String originalString = "Hello, World!";
         System.out.println("Original String: " + originalString);



        // 1. Convert to Uppercase
        String uppercaseString = originalString.toUpperCase();
        System.out.println("1. Uppercase: " + uppercaseString);

        // 2. Convert to Lowercase
        String lowercaseString = originalString.toLowerCase();
        System.out.println("2. Lowercase: " + lowercaseString);

        // 3. Substring
        String substring = originalString.substring(7, 12);
        System.out.println("3. Substring ( Display 'World' from the Original String): " + substring);

        // 4. Index of a Character
        int indexOfW = originalString.indexOf('W');
        System.out.println("4. Index of 'W' (index value start from 0): " + indexOfW);

        // 5. Replace characters
        String replacedString = originalString.replace('o', 'X');
        System.out.println("5. Replaced String ( Replacing 'o' with 'X'):  " + replacedString);

        // 6. Check if starts with "Hello"
        boolean startsWithHello = originalString.startsWith("Hello");
        System.out.println("6. Starts with 'Hello': " + startsWithHello);

        // 7. Check if ends with "World!"
        boolean endsWithWorld = originalString.endsWith("World!");
        System.out.println("7. Ends with 'World!': " + endsWithWorld);

        // 8. Trim leading and trailing whitespaces
        String stringWithWhitespaces = "   Trim Me   ";
        System.out.println("8. string With Whitespaces  : " + stringWithWhitespaces);
      String trimmedString = stringWithWhitespaces.trim();
       System.out.println("   Trimmed String (in the begining and end): '" + trimmedString + "'");
    }
}
```

Output:



```
F:\java>javac StringMethodsExample.java

F:\java>java StringMethodsExample
Original String: Hello, World!
1. Uppercase: HELLO, WORLD!
2. Lowercase: hello, world!
3. Substring ( Display 'World' from the Original String): World
4. Index of 'W' (index value start from 0): 7
5. Replaced String ( Replacing 'o' with 'X'):  HellX, WXrld!
6. Starts with 'Hello': true
7. Ends with 'World!': true
```

Related Exercise:

Q1. Develop a java program to input a string through the keyboard and perform various string class methods based on User's Choice.

EXERCISE 93 : Create and use arrays

Objectives

At the end of this exercise you shall be able to

- know about single dimensional and two dimensional arrays and its use in Java
- develop Java programs using single dimensional and two dimensional arrays.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text editor (Visual studio / Sublime / Note pad)

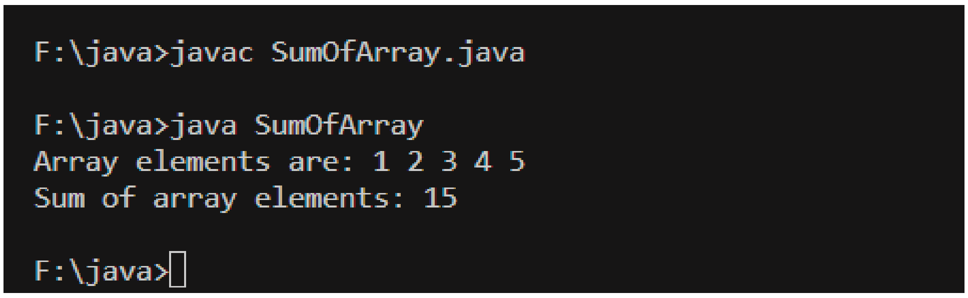
Procedure

1 Single Dimensional Arrays

TASK 1: **Method1: Sum of Array Elements**

```
// Sum of Array elements
public class SumOfArray {
    public static void main(String[] args) {
        // Declare and initialize an array
        int[] numbers = {1, 2, 3, 4, 5};
        System.out.print("Array elements are: " );
        // Initialize sum variable
        int sum = 0;

        // Iterate through the array and add each element to sum
        for (int i = 0; i < numbers.length; i++) {
            System.out.print(numbers[i]+" ");
            sum += numbers[i];
        }
        // Go to the next line
        System.out.println();
       // Print the sum
        System.out.println("Sum of array elements: " + sum);
    }
}
```

Output:



```
F:\java>javac SumOfArray.java

F:\java>java SumOfArray
Array elements are: 1 2 3 4 5
Sum of array elements: 15

F:\java>
```

Explanation:

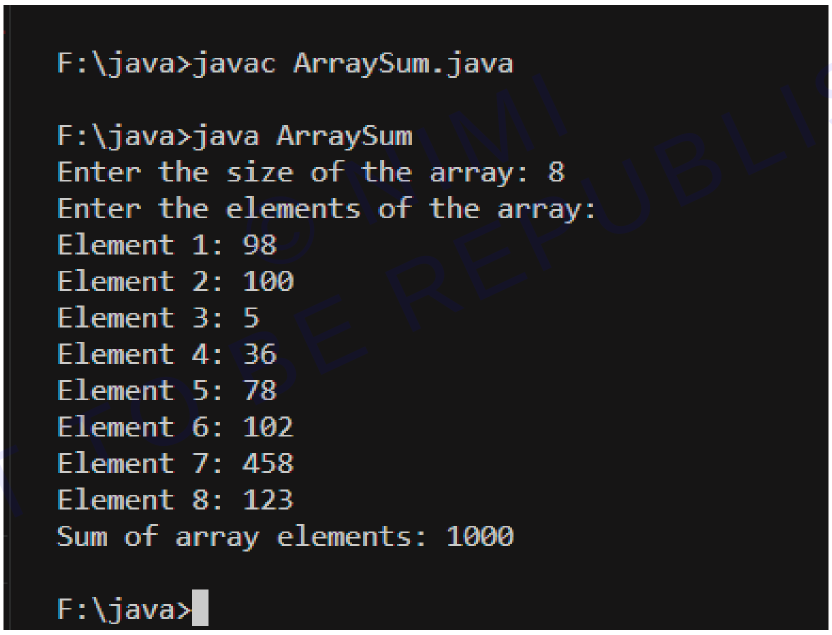
- This program calculates the sum of elements in an array of integers.
- It initializes an array numbers with values {1, 2, 3, 4, 5}.
- It iterates through each element of the array using a for loop and adds each element to the variable sum.
- Finally, it prints the sum of the array elements.

TASK 1_Method2: Java program that allows the user to input array elements through the keyboard and displays the sum of those elements

Here's a Java program that allows the user to input array elements through the keyboard and displays the sum of those elements:

```
//Sum of array elements
import java.util.Scanner;
public class ArraySum {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

       // Create an array of the specified size
        int[] numbers = new int[size];

        // Prompt the user to enter array elements
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            System.out.print("Element " + (i + 1) + ": ");
            numbers[i] = scanner.nextInt();
        }
        // Calculate the sum of array elements
        int sum = 0;
        for (int i = 0; i < size; i++) {
            sum += numbers[i];
        }

        // Display the sum of array elements
        System.out.println("Sum of array elements: " + sum);

        scanner.close(); // Close the scanner to avoid resource leak
    }
}
```

Output:



```
F:\java>javac ArraySum.java

F:\java>java ArraySum
Enter the size of the array: 8
Enter the elements of the array:
Element 1: 98
Element 2: 100
Element 3: 5
Element 4: 36
Element 5: 78
Element 6: 102
Element 7: 458
Element 8: 123
Sum of array elements: 1000

F:\java>
```

Explanation:

- The program starts by importing the Scanner class from the java.util package, which is used to read input from the keyboard.
- It creates a Scanner object named scanner to read input.
- The program prompts the user to enter the size of the array and reads the input using scanner.nextInt().
- It then creates an integer array numbers of the specified size.
- Using a for loop, the program prompts the user to enter each element of the array and reads the input for each element.
- After reading all the elements, another for loop calculates the sum of all the elements of the array.

- Finally, it prints out the sum of the array elements.
- The scanner.close() statement closes the scanner to release system resources.

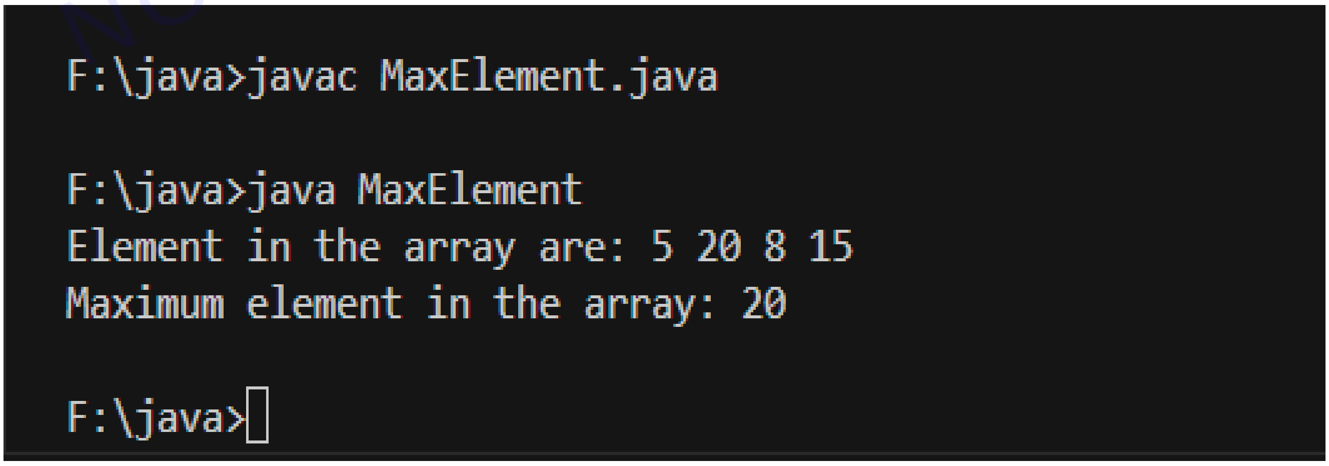
This program allows the user to dynamically input array elements and calculates the sum accordingly.

— — — — —

TASK 2: **Finding Maximum Element in an Array**

```
public class MaxElement {
    public static void main(String[] args) {
        // Declare and initialize an array
        int[] numbers = {10, 5, 20, 8, 15};
         System.out.print("Element in the array are: ");
       // Assume the first element is the maximum
        int max = numbers[0];

        // Iterate through the array to find the maximum element
        for (int i = 1; i < numbers.length; i++) {
            System.out.print(numbers[i]+" ") ;
            if (numbers[i] > max) {
                max = numbers[i];
            }
        }
        System.out.println();
        // Print the maximum element
        System.out.println("Maximum element in the array: " + max);
    }
}
```

Output:



```
F:\java>javac MaxElement.java

F:\java>java MaxElement
Element in the array are: 5 20 8 15
Maximum element in the array: 20

F:\java>
```

Explanation:

- This program finds the maximum element in an array of integers.
- It initializes an array numbers with values {10, 5, 20, 8, 15}.
- It assumes the first element of the array as the maximum.
- Then, it iterates through the array starting from the second element, comparing each element with the current maximum and updating max if a larger element is found.
- Finally, it prints the maximum element.

These examples demonstrate some basic operations with arrays in Java, such as summing up elements and finding the maximum element. They showcase the use of loops for iteration and conditional statements for making comparisons.

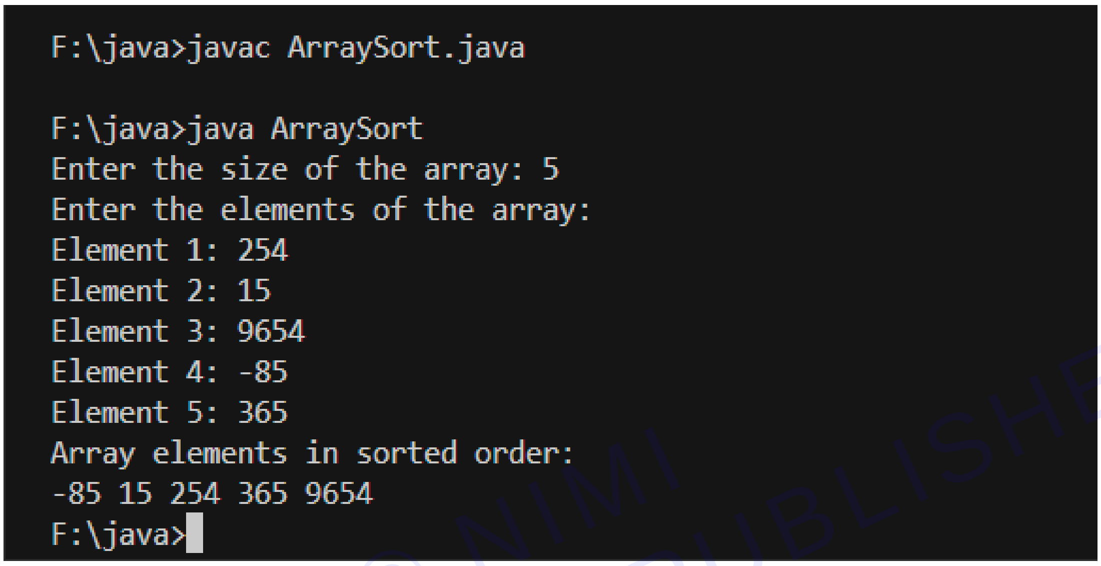
— — — — —

TASK 3: **Sorting Array elements**

Method 1: Java program that allows the user to input array elements through the keyboard and displays them in sorted order using Quick Sort - Arrays.sort() method:

```
// using Quick Sort - Arrays.sort() method
 import java.util.Arrays;
import java.util.Scanner;
public class ArraySort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        // Create an array of the specified size
        int[] numbers = new int[size];

        // Prompt the user to enter array elements
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            System.out.print("Element " + (i + 1) + ": ");
            numbers[i] = scanner.nextInt();
        }

        // Sort the array
        Arrays.sort(numbers);

        // Display the sorted array
        System.out.println("Array elements in sorted order:");
        for (int i = 0; i < size; i++) {
            System.out.println(numbers[i]);
        }
        scanner.close(); // Close the scanner to avoid resource leak
    }
}
```

Output:



```
F:\java>javac ArraySort.java

F:\java>java ArraySort
Enter the size of the array: 5
Enter the elements of the array:
Element 1: 254
Element 2: 15
Element 3: 9654
Element 4: -85
Element 5: 365
Array elements in sorted order:
-85 15 254 365 9654
F:\java>
```

Explanation:

- This program is quite similar to the previous one but with an additional step to sort the array elements before displaying them.
- It imports the Arrays class from the java.util package, which provides a method sort() to sort arrays.
- After reading all the elements, it uses Arrays.sort(numbers) to sort the array in ascending order.
- Then, it displays the sorted array elements using a loop.
- Finally, it closes the Scanner object to release system resources.

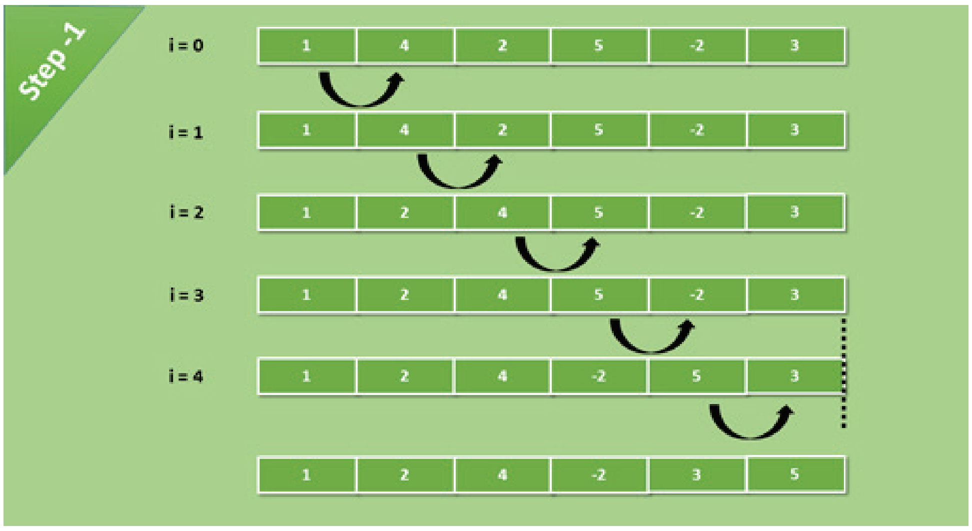
This program allows the user to input array elements dynamically and displays them in sorted order.

Method 2: Java program that sorts n elements in descending order using the bubble sort method:

Let's see with an example. Here, each step is briefly illustrated:

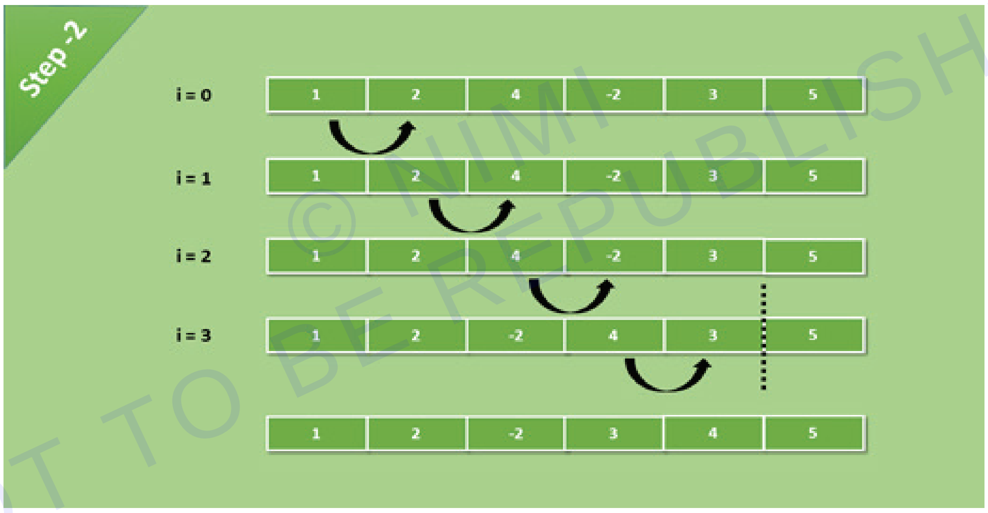
Bubble sort algorithm is an algorithm that sorts an array by comparing two adjacent elements and swapping them if they are not in the intended order. Here order can be anything like increasing or decreasing.

Let's see with an example. Here, each step is briefly illustrated:

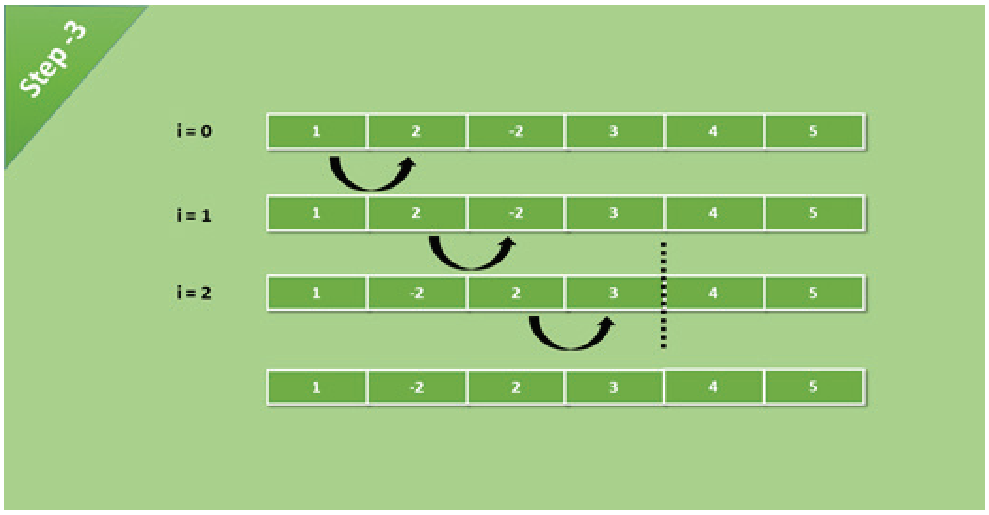


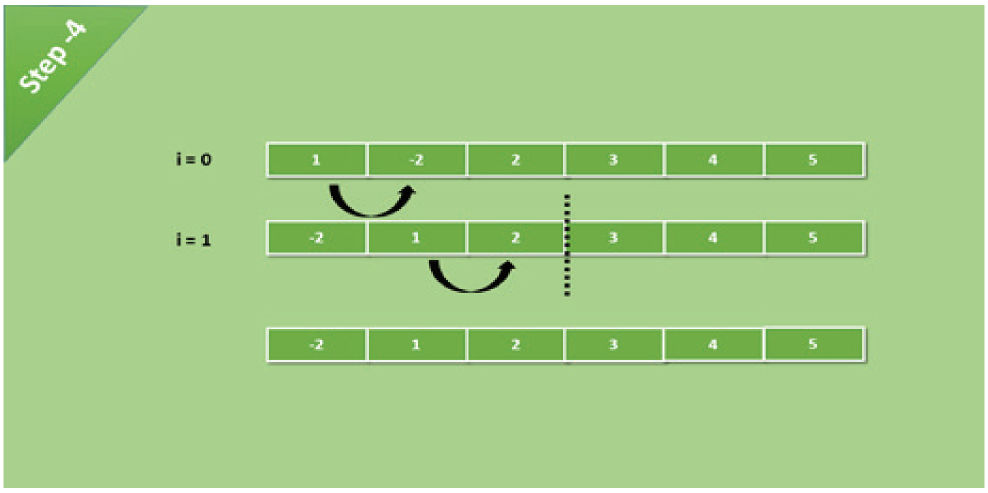
Comparisons happen till the last element 1

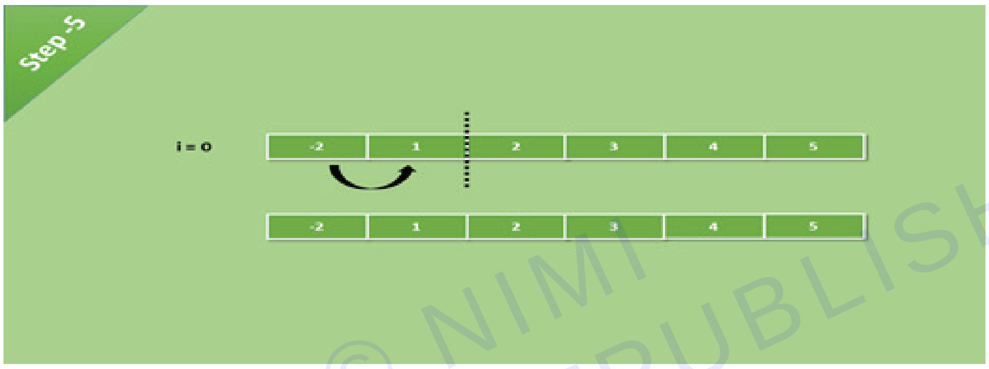
After each iteration, the greatest value of the array becomes the last index value of the array. In each iteration, the comparison happens till the last unsorted element.



Now comparison reduced one step because the biggest element is at its right place







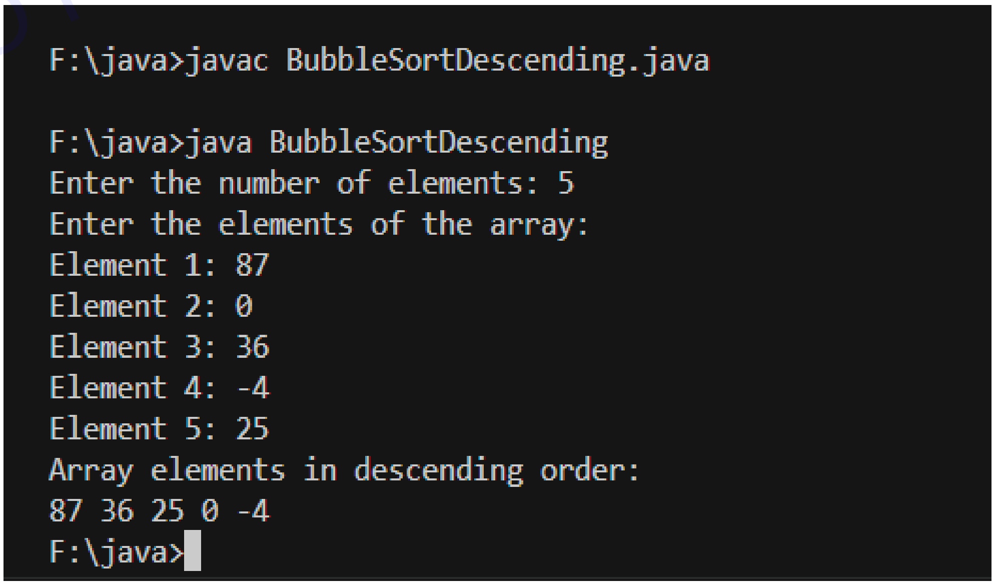
After all the iteration and comparisons of elements, we get a sorted array.

— — — — —

TASK 4: **Sort array in descending order**

```
// Sort array in descending order
import java.util.Scanner;
public class BubbleSortDescending {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the number of elements
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        // Create an array of the specified size
        int[] arr = new int[n];

        // Prompt the user to enter array elements
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
                System.out.print("Element " + (i + 1) + ": ");
                arr[i] = scanner.nextInt();
            }

            // Bubble sort in descending order
            for (int i = 0; i < n - 1; i++) {
                for (int j = 0; j < n - i - 1; j++) {
                    if (arr[j] < arr[j + 1]) {
                        // Swap arr[j] and arr[j+1]
                        int temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                    }
                }
            }

            // Display the sorted array in descending order
            System.out.println("Array elements in descending order:");
            for (int i = 0; i < n; i++) {
           System.out.println(arr[i]);
            }

            scanner.close(); // Close the scanner to avoid resource leak
        }
    }
```

Output:



```
F:\java>javac BubbleSortDescending.java

F:\java>java BubbleSortDescending
Enter the number of elements: 5
Enter the elements of the array:
Element 1: 87
Element 2: 0
Element 3: 36
Element 4: -4
Element 5: 25
Array elements in descending order:
87 36 25 0 -4
F:\java>
```

Explanation:

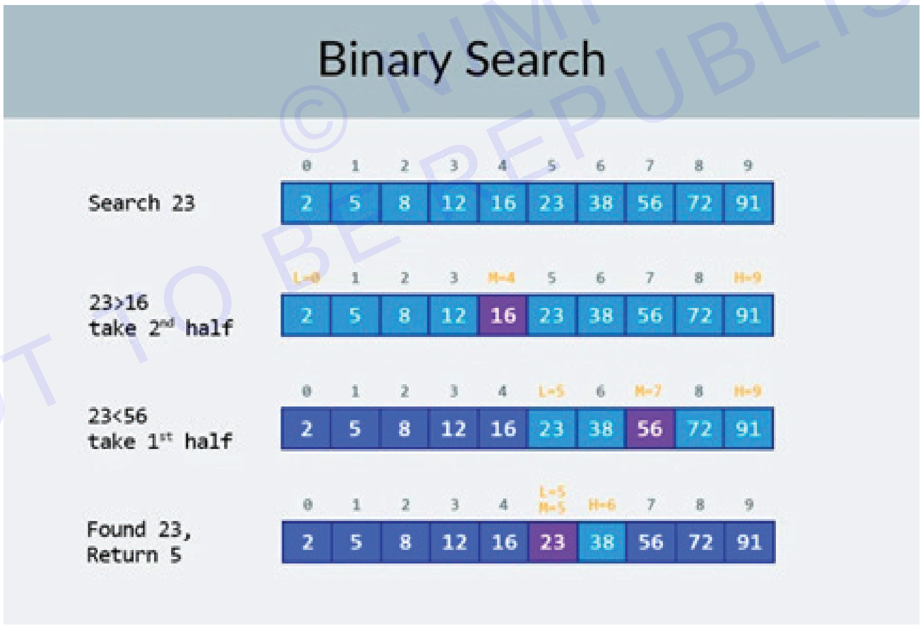
- This program sorts n elements in descending order using the bubble sort algorithm.
- It first prompts the user to enter the number of elements and reads the input.
- Then, it creates an array of the specified size.
- The user is prompted to enter each element of the array.
- The program then performs the bubble sort algorithm to sort the array in descending order.
- In the bubble sort algorithm, we iterate through the array multiple times, comparing adjacent elements and swapping them if they are in the wrong order.
- After sorting, it displays the sorted array elements in descending order.
- Finally, it closes the Scanner object to release system resources.

Bubble sort is easy to understand and implement. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until the list is sorted.

— — — — —

TASK 5 : **Binary Search**

Binary search is one of the searching techniques applied when the input is sorted here we are focusing on finding the middle element that acts as a reference frame whether to go left or right to it as the elements are already sorted. This searching helps in optimizing the search technique with every iteration is referred to as binary search and readers do stress over it as it is indirectly applied in solving questions.



Binary Search Algorithm in Java

Below is the Algorithm designed for Binary Search:

1 Start

2 Take input array and Target

3 Sort the array if it is not in sorted order .

4 Initialise start = 0 and end = (array size -1)

5 Initialise mid variable

```
6   mid = (start+end)/2
7   if array[ mid ] == target then return mid
8   if array[ mid ] < target then start = mid+1
9   if array[ mid ] > target then end = mid-1
10  if start<=end then goto step 6
11  return -1 as Not element found
12  Exit
```

Here’s a Java program that allows the user to input array elements through the keyboard and performs binary search:

```
import java.util.Scanner;
public class BinarySearch {
    public static int binarySearch(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target)
                return mid;
            else if (arr[mid] < target)
                left = mid + 1;
            else
                right = mid - 1;
        }

        return -1;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the size of the array
        System.out.print(“Enter the size of the array: “);
        int size = scanner.nextInt();

        // Create an array of the specified size
        int[] arr = new int[size];

        // Prompt the user to enter array elements
        System.out.println(“Enter the elements of the array :”);
        for (int i = 0; i < size; i++) {
            System.out.print("Element " + (i + 1) + ": ");
           arr[i] = scanner.nextInt();
        }
        // Bubble sort in ascending order
        for (int i = 0; i < size - 1; i++) {
            for (int j = 0; j < size - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Swap arr[j] and arr[j+1]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
         // Display the sorted array in ascending order
        System.out.print("Array elements in ascending order:");
        for (int i = 0; i < size; i++) {
            System.out.print(arr[i]+" ");
        }
        // Prompt the user to enter the target element to be searched
        System.out.println();
        System.out.print("Enter the target element to search: ");
        int target = scanner.nextInt();

        // Perform binary search
        int index = binarySearch(arr, target);

        // Print the result of the search
        if (index != -1)
            System.out.println("Element " + target + " found at index " + (index=index+1));
        else
            System.out.println("Element " + target + " not found in the array.");

        scanner.close(); // Close the scanner
    }
}
```

Explanation:

1 **Binary Search Method:**

- The binarySearch method performs a binary search on a sorted array to find the index of the target element.
- It takes two parameters: the array (arr) and the target element (target).
- The method uses a while loop with left and right pointers to narrow down the search space until the target element is found or the search space is empty.
- If the target element is found, the method returns the index; otherwise, it returns -1.

2 **Main Method:**

- The main method is the entry point of the program.
- It uses a Scanner to take user input for the size of the array, array elements, and the target element to be searched**.**

3 **Array Input and Bubble Sort:**

- The program prompts the user to enter the size of the array and then input the array elements.
- After taking the array elements, a simple bubble sort algorithm is applied to sort the array in ascending order.

4 **Sorted Array Display:**

- The sorted array is displayed in ascending order after the bubble sort.

5 **Target Element Input:**

- The user is prompted to enter the target element that needs to be searched in the array.

6 **Binary Search Execution:**

- The binarySearch method is called with the sorted array and the target element as parameters to perform the binary search.

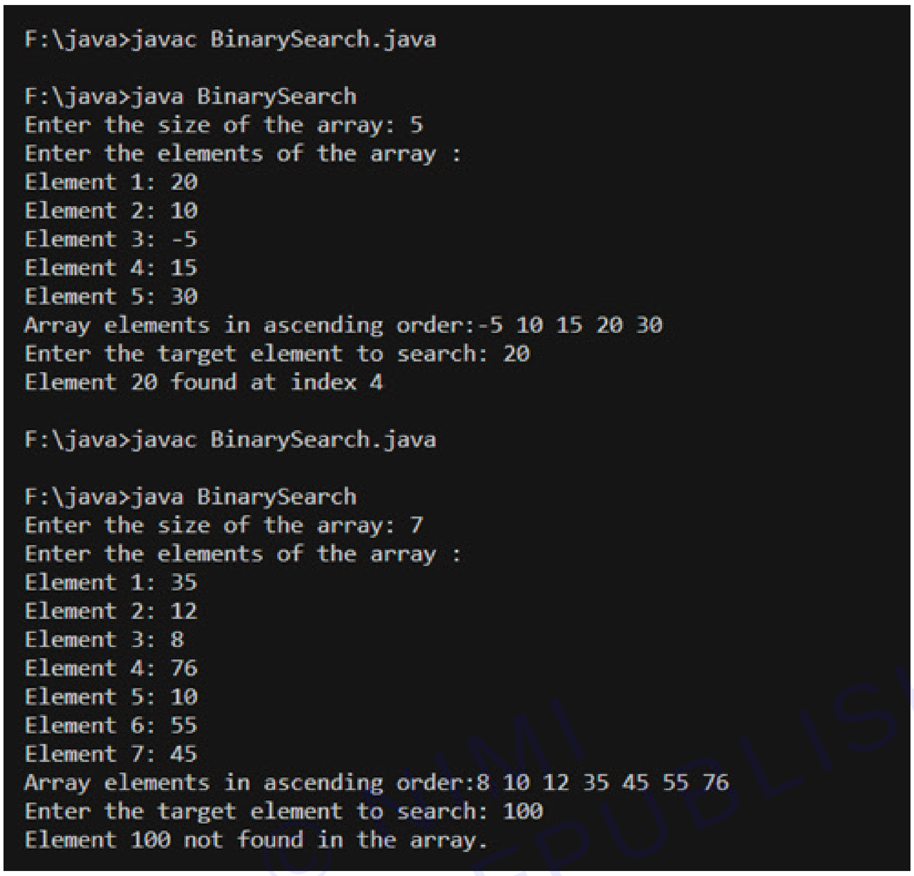
7 **Search Result Display:**

- The result of the binary search is displayed:
- If the target element is found, the program prints its index.
- If the target element is not found, a corresponding message is printed**.**

8 **Scanner Closing:**

- The Scanner is closed to release system resources.

Output:



```
F:\java>javac BinarySearch.java

F:\java>java BinarySearch
Enter the size of the array: 5
Enter the elements of the array :
Element 1: 20
Element 2: 10
Element 3: -5
Element 4: 15
Element 5: 30
Array elements in ascending order:-5 10 15 20 30
Enter the target element to search: 20
Element 20 found at index 4

F:\java>javac BinarySearch.java

F:\java>java BinarySearch
Enter the size of the array: 7
Enter the elements of the array :
Element 1: 35
Element 2: 12
Element 3: 8
Element 4: 76
Element 5: 10
Element 6: 55
Element 7: 45
Array elements in ascending order:8 10 12 35 45 55 76
Enter the target element to search: 100
Element 100 not found in the array.
```

Related Tasks:

Develop the following java programs:

1 To display the sum of odd and even numbers in an array

2 Read 'n' elements of an array and display the count of Zeros, Positive, Negative numbers in the array.

3 To search the given element is present in the array or not using Linear Search method (If present display the position also)

4 Read 'n'elements and store the even and odd numbers in two different arrays and display it.

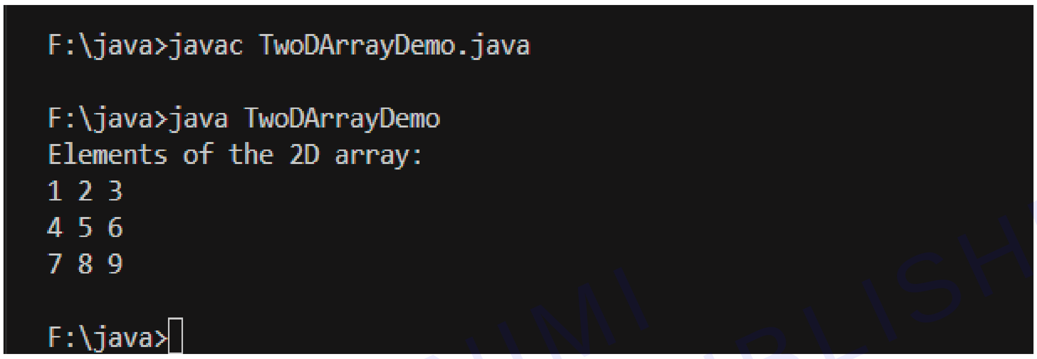
5 Develop a program to sort 'n' elements in Ascending order using Bubble sort method.

2 Two Dimensional Arrays in Java

TASK 1: **Here's a Java program that demonstrates the usage of two-dimensional arrays along with an explanation:**

```
public class TwoDArrayDemo {
    public static void main(String[] args) {
        // Declaration and initialization of a 2D array
        int[][] matrix = { {1, 2, 3},
                          {4, 5, 6},
                          {7, 8, 9} };
 // Displaying the elements of the 2D array
        System.out.println("Elements of the 2D array:");
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println(); // Move to the next line after printing each row
        }
    }
}
```

Output:



```
F:\java>javac TwoDArrayDemo.java

F:\java>java TwoDArrayDemo
Elements of the 2D array:
1 2 3
4 5 6
7 8 9

F:\java>
```

Explanation:

- A two-dimensional array in Java is an array of arrays. It is a matrix-like structure with rows and columns.
- In the above program, a 2D array named matrix is declared and initialized with integer values.
- The declaration int[][] matrix indicates that matrix is a two-dimensional array of integers.
- The array initializer { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} } initializes the 2D array with three rows and three columns.
- The outer loop for (int i = 0; i < matrix.length; i++) iterates over the rows of the array, and the inner loop for (int j = 0; j < matrix[i].length; j++) iterates over the columns of each row.
- Within the nested loops, matrix[i][j] is used to access each element of the 2D array.
- The program prints each element of the 2D array row-wise, with each row printed on a separate line.

Two-dimensional arrays are commonly used to represent tabular data, matrices, grids, and other structured data in Java programs. They offer a convenient way to organize and manipulate data in rows and columns. In this example, the 2D array matrix represents a 3x3 matrix with integer values. The nested loops are used to iterate over each element of the array and perform operations as needed.

— — — — —

TASK 2: **Java program that allows the user to input two matrices through the keyboard and displays their sum:**

```
import java.util.Scanner;
public class MatrixSum {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Prompt the user to enter the dimensions of the matrices
        System.out.print("Enter the number of rows: ");
        int rows = scanner.nextInt();
        System.out.print("Enter the number of columns: ");
        int cols = scanner.nextInt();

        // Create arrays to store the matrices
        int[][] matrix1 = new int[rows][cols];
        int[][] matrix2 = new int[rows][cols];
        int[][] sumMatrix = new int[rows][cols];

        // Prompt the user to input elements for the first matrix
        System.out.println("Enter elements for the first matrix:");
        inputMatrix(scanner, matrix1);

        // Prompt the user to input elements for the second matrix
        System.out.println("Enter elements for the second matrix:");
        inputMatrix(scanner, matrix2);

        // Calculate the sum of the matrices
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                sumMatrix[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }

        // Display the sum matrix
        System.out.println("Sum of the matrices:");
        displayMatrix(sumMatrix);

        scanner.close(); // Close the scanner
    }

    // Method to input elements into a matrix
    public static void inputMatrix(Scanner scanner, int[][] matrix) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                System.out.print("Enter element [" + (i+1) + "][" + (j+1) + "]: ");
                matrix[i][j] = scanner.nextInt();
            }
        }
    }

    // Method to display a matrix
    public static void displayMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int element : row) {
                System.out.print(element + " ");
            }
            System.out.println();
        }
    }
}
```

Explanation:

- This program allows the user to input two matrices of the same dimensions (number of rows and columns) through the keyboard and then displays their sum.
- It first prompts the user to enter the dimensions (number of rows and columns) of the matrices.
- Two 2D arrays, matrix1 and matrix2, are created to store the input matrices, and another array, sumMatrix, is created to store the sum of the matrices.
- The inputMatrix() method is used to prompt the user to input elements for each matrix.
- The displayMatrix() method is used to display the elements of a matrix.
- After inputting the elements for both matrices, the program calculates the sum of the corresponding elements from the two matrices and stores the result in the sumMatrix array.
- Finally, it displays the sum matrix to the user.

This program demonstrates the use of 2D arrays in Java to represent matrices and perform basic matrix operations, such as addition. It illustrates the inputting of matrix elements through the keyboard, calculating the sum of two matrices, and displaying the resulting sum matrix.

Related Tasks:

Develop the following java programs:

1 Display the upper diagonal elements and its sum of a matrix

2 Display the upper triangular elements of a matrix

3 Display the transpose of a matrix

4 Display the product of two matrices

5 Check the given matrix is Symmetric or not (A symmetric matrix is a square matrix where A [i][j]=A[j][i]. ie., A= AT)

EXERCISE 94 : Create and use simple classes, objects and methods in JAVA

Objectives

At the end of this exercise you shall be able to

- know about Create and use simple classes, objects and methods in JAVA
- develop Java programs using classes, objects and Methods.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

Here's a brief overview of creating and using simple classes, objects, and methods in Java:

Classes:

- In Java, a class is a blueprint for creating objects. It defines the properties and behaviors of objects.
- A class is declared using the class keyword followed by the class name.
- Inside a class, you define fields (variables) and methods to represent the attributes and behaviors of objects of that class.

Objects:

- An object is an instance of a class. It represents a real-world entity and has its own state and behavior.
- To create an object in Java, you use the new keyword followed by the constructor of the class.
- The constructor initializes the object and allocates memory for it.

Methods:

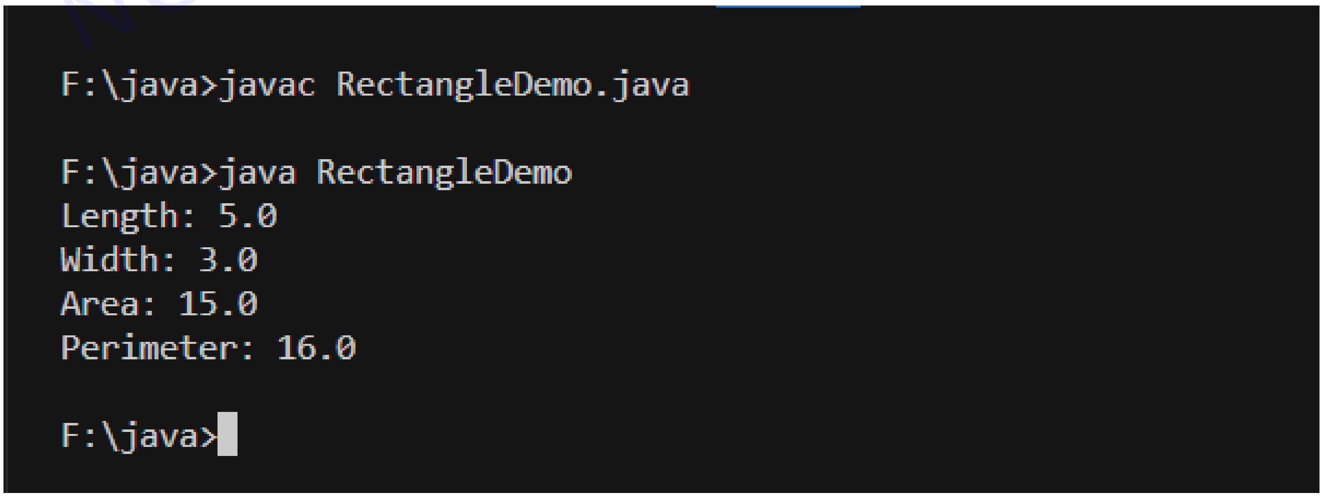
- Methods are functions defined within a class that perform certain actions or calculations.
- Methods encapsulate behavior and can be called to perform specific tasks on objects of the class.
- Methods can have parameters (inputs) and return values (outputs).

Below is a simple Java program that demonstrates the creation and usage of classes, objects, and methods:

TASK 1: **Calculate the area and perimeter of a rectangle using rectangle class demo**

```
    // Define a class named Rectangle
class Rectangle {
    // Instance variables
    double length;
    double width;
    // Constructor to initialize the rectangle with length and width
    Rectangle(double l, double w) {
        length = l;
        width = w;
    }
    // Method to calculate area of the rectangle
    double calculateArea() {
        return length * width;
    }
    // Method to calculate perimeter of the rectangle
    double calculatePerimeter() {
        return 2 * (length + width);
    }
}
// Main class to demonstrate the usage of Rectangle class
public class RectangleDemo {
    public static void main(String[] args) {
        // Create an object of Rectangle class
        Rectangle rect1 = new Rectangle(5.0, 3.0);
        // Accessing object properties and methods
        System.out.println("Length: " + rect1.length);
        System.out.println("Width: " + rect1.width);
        System.out.println("Area: " + rect1.calculateArea());
        System.out.println("Perimeter: " + rect1.calculatePerimeter());
    }
}
```

Output:



```
F:\java>javac RectangleDemo.java

F:\java>java RectangleDemo
Length: 5.0
Width: 3.0
Area: 15.0
Perimeter: 16.0

F:\java>
```

Explanation

- **Rectangle class**: It represents a simple geometric rectangle with two properties - length and width. It also contains methods to calculate the area and perimeter of the rectangle.
- **length and width** are instance variables that define the state of the rectangle.
- **The Rectangle** constructor initializes the length and width of the rectangle when an object is created.
- **calculateArea()** and calculatePerimeter() are methods that calculate the area and perimeter of the rectangle, respectively.
- **RectangleDemo class**: It serves as the main class to demonstrate the usage of the Rectangle class.
- In the main method, an object rect1 of the Rectangle class is created using the constructor.
- We access the properties of the rect1 object (length and width) and call its methods (calculateArea() and calculatePerimeter()).
- The calculated area and perimeter are then printed to the console.

  This program illustrates the concept of classes and objects in Java. The Rectangle class encapsulates related data and behavior, allowing for easier management and manipulation. By creating objects of the Rectangle class, we can instantiate multiple rectangles with different dimensions and perform operations specific to each object.

Here are a few more Java programs demonstrating the creation and usage of classes, objects, and methods:

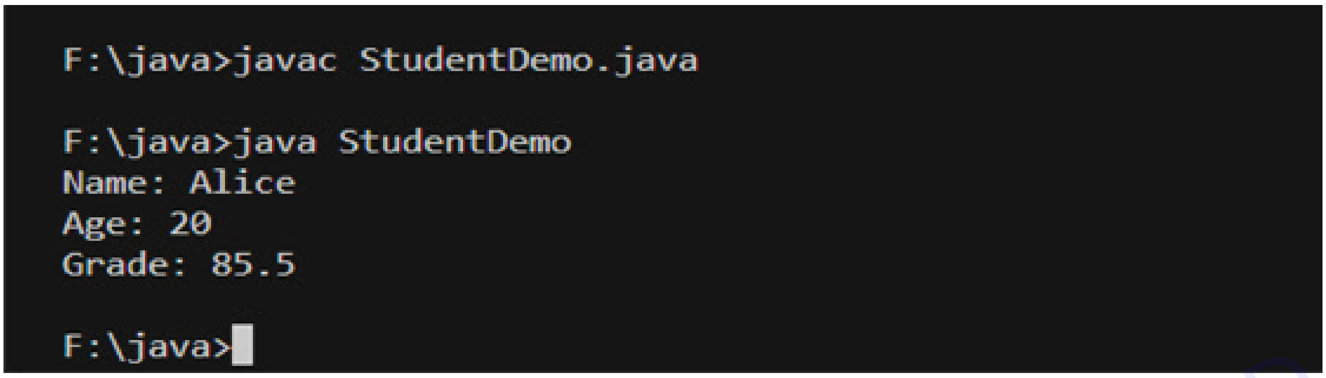
— — — — —

TASK 2**: Student Class**

```
class Student {
    // Instance variables
    String name;
    int age;
    double grade;
    // Constructor
    Student(String n, int a, double g) {
        name = n;
        age = a;
        grade = g;
    }
    // Method to display student information
    void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Grade: " + grade);
    }
}
public class StudentDemo {
    public static void main(String[] args) {
        // Create an object of Student class
```

```
            Student student1 = new Student("Alice", 20, 85.5);
            // Accessing object properties and methods
            student1.displayInfo();
        }
    }
```

Output:



```
F:\java>javac StudentDemo.java

F:\java>java StudentDemo
Name: Alice
Age: 20
Grade: 85.5

F:\java>
```

Explanation:

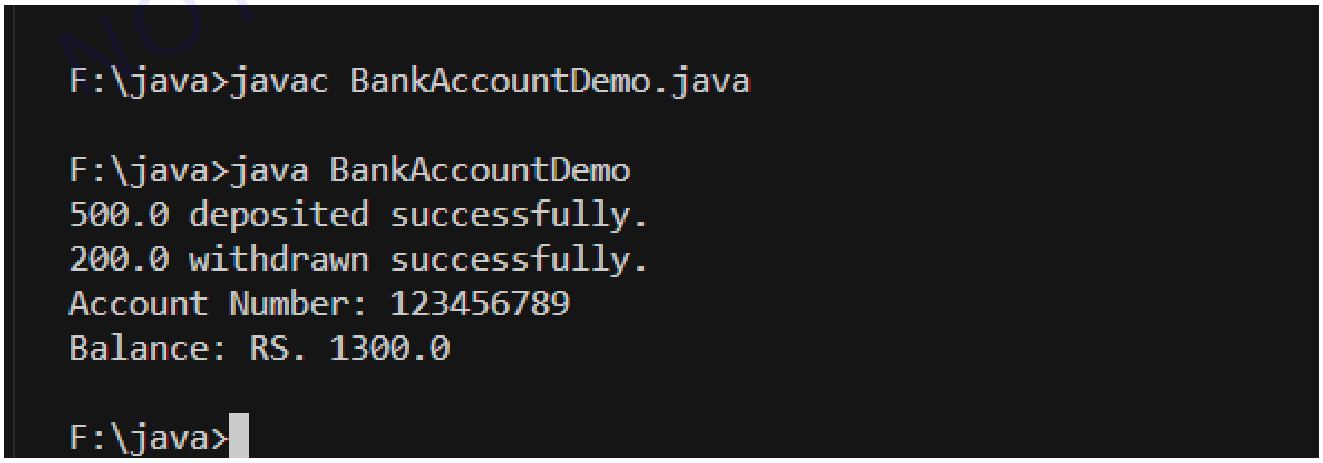
- This program defines a Student class with properties like name, age, and grade.
- It has a constructor to initialize the instance variables.
- The displayInfo() method prints out the student's information.
- In the main() method, an object student1 of the Student class is created and its properties are accessed using the displayInfo() method.

— — — — —

TASK 3**: Bank Account Class**

```
  class BankAccount {
      // Instance variables
      String accountNumber;
      double balance;
      // Constructor
      BankAccount(String accNum, double initialBalance) {
          accountNumber = accNum;
          balance = initialBalance;
      }
      // Method to deposit money
      void deposit(double amount) {
          balance += amount;
          System.out.println(amount + " deposited successfully.");
      }
      // Method to withdraw money
      void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            System.out.println(amount + " withdrawn successfully.");
        } else {
            System.out.println("Insufficient funds.");
        }
    }
    // Method to display account information
    void displayAccountInfo() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Balance: $" + balance);
    }
}
public class BankAccountDemo {
    public static void main(String[] args) {
        // Create an object of BankAccount class
        BankAccount account1 = new BankAccount("123456789", 1000.0);
        // Deposit and withdraw operations
        account1.deposit(500.0);
        account1.withdraw(200.0);
        // Display account information
        account1.displayAccountInfo();
    }
}
```

Output:



```
F:\java>javac BankAccountDemo.java

F:\java>java BankAccountDemo
500.0 deposited successfully.
200.0 withdrawn successfully.
Account Number: 123456789
Balance: RS. 1300.0

F:\java>
```

Explanation:

- This program defines a BankAccount class with properties like account number and balance.
- It has a constructor to initialize the instance variables.
- The deposit() method deposits money into the account, withdraw() method withdraws money, and displayAccountInfo() method displays the account information.
- In the main() method, an object account1 of the BankAccount class is created and operations like deposit, withdraw, and display account information are performed.

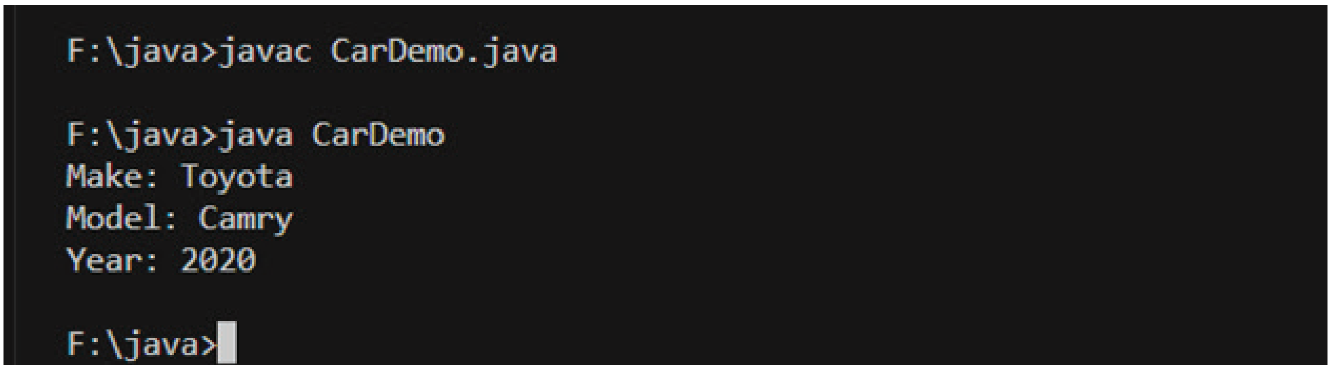
These examples demonstrate how classes, objects, and methods are used in Java to model real-world entities and perform operations on them. They encapsulate data and behavior within classes, promoting code reusability and maintainability.

— — — — —

TASK 4**: Car Class**

```
  class Car {
   // Instance variables
   String make;
   String model;
   int year;
   // Constructor
   Car(String make, String model, int year) {
       this.make = make;
       this.model = model;
       this.year = year;
   }
   // Method to display car information
   void displayInfo() {
       System.out.println("Make: " + make);
       System.out.println("Model: " + model);
       System.out.println("Year: " + year);
   }
}
public class CarDemo {
   public static void main(String[] args) {
       // Create an object of Car class
       Car myCar = new Car("Toyota", "Camry", 2020);
       // Accessing object properties and methods
       myCar.displayInfo();
   }
}
```

Output :



```
F:\java>javac CarDemo.java

F:\java>java CarDemo
Make: Toyota
Model: Camry
Year: 2020

F:\java>
```

Explanation:

- This program defines a Car class with properties like make, model, and year.
- It has a constructor to initialize the instance variables.
- The displayInfo() method prints out the car's information.
- In the main() method, an object myCar of the Car class is created and its properties are accessed using the displayInfo() method.

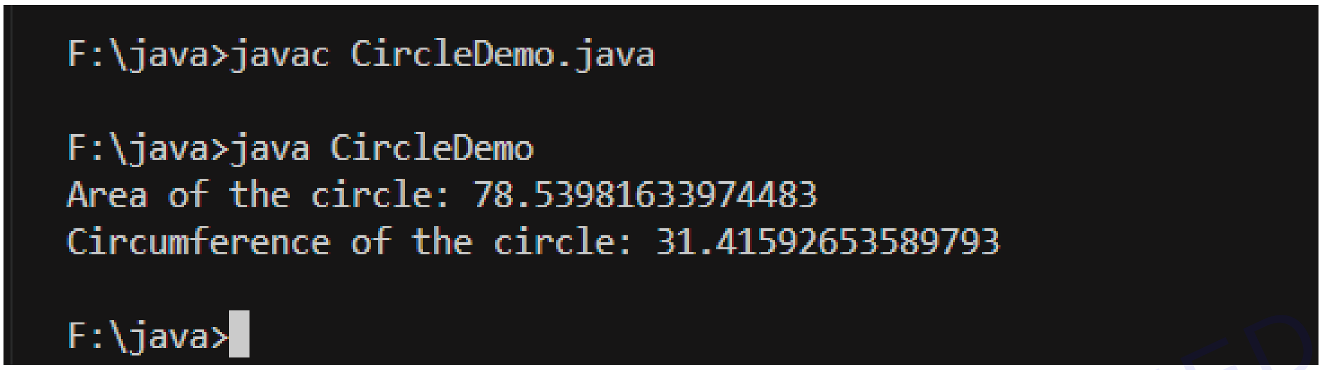
— — — — —

TASK 5**: Circle Class**

```
class Circle {
    // Instance variable
    double radius;
    // Constructor
    Circle(double r) {
        radius = r;
    }
    // Method to calculate area
    double calculateArea() {
        return Math.PI * radius * radius;
    }
    // Method to calculate circumference
    double calculateCircumference() {
        return 2 * Math.PI * radius;
    }
}
public class CircleDemo {
    public static void main(String[] args) {
        // Create an object of Circle class
```

```
            Circle myCircle = new Circle(5.0);
            // Accessing object properties and methods
            System.out.println("Area of the circle: " + myCircle.calculateArea());
            System.out.println("Circumference of the circle: " + myCircle.calculateCircumference());
        }
    }
```

Output:



```
F:\java>javac CircleDemo.java

F:\java>java CircleDemo
Area of the circle: 78.53981633974483
Circumference of the circle: 31.41592653589793

F:\java>
```

Explanation:

- This program defines a Circle class with a property radius.
- It has a constructor to initialize the radius.
- The calculateArea() method calculates the area of the circle, and the calculateCircumference() method calculates the circumference.
- In the main() method, an object myCircle of the Circle class is created, and its methods are called to calculate the area and circumference of the circle.

These examples demonstrate how classes, objects, and methods can be used to model various entities and perform operations on them in Java. They encapsulate data and behavior within classes, promoting code organization, reusability, and maintainability.

EXERCISE 95 : Pass data and Objects to Methods

Objectives

At the end of this exercise you shall be able to

- know about how to pass data and objects to methods in JAVA
- develop Java programs using data and object passing to methods.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

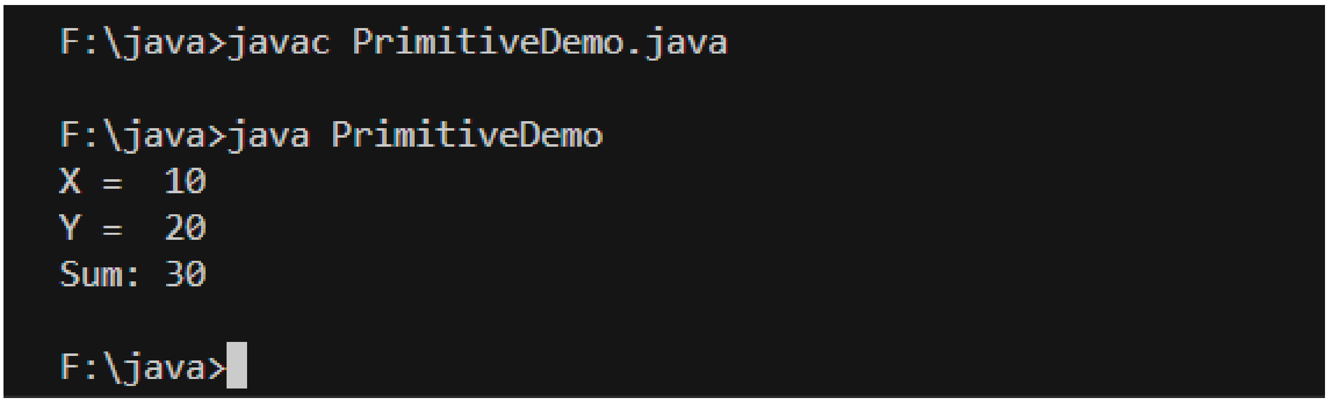
In Java, you can pass data and objects to methods in various ways, such as passing primitive data types, passing objects, and passing arrays. Below are examples of Java programs demonstrating these concepts:

1 **Passing Primitive Data Types to Methods**

TASK 1: **Finding sum of two numbers**

```
    public class PrimitiveDemo {
     public static void main(String[] args) {
          int x = 10;
          i nt y = 20;
          // Passing primitive data types to a method
          int sum = add(x, y);
          System.ouwt.println("Sum: " + sum);
      }
      // Method to add two integers
      public static int add(int a, int b) {
          return a + b;
      }
  }
```

Output :



```
F:\java>javac PrimitiveDemo.java

F:\java>java PrimitiveDemo
X =  10
Y =  20
Sum: 30

F:\java>
```

Explanation:

- In this program, we define a method add that takes two integers as parameters and returns their sum.
- We declare variables x and y in the main method and assign values to them.
- We call the add method and pass x and y as arguments.
- The method performs addition on the provided integers and returns the result, which is then printed in the main method.

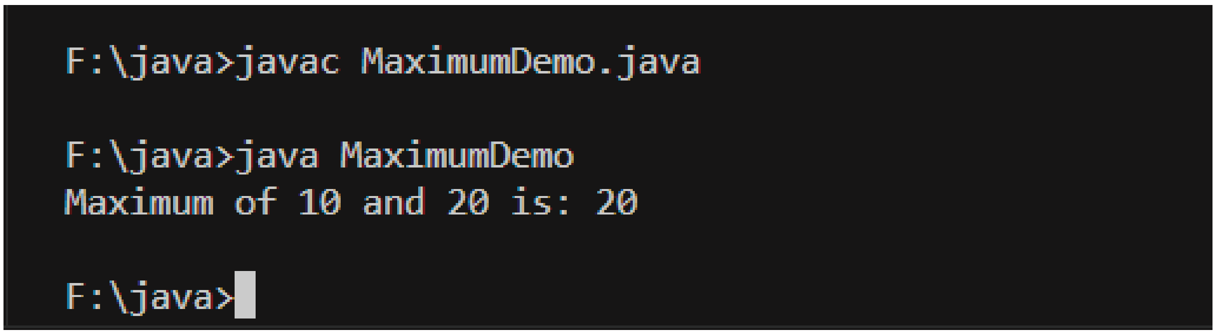
Here's another example demonstrating the concept of passing primitive data types to methods:

— — — — —

TASK 2: **Finding the Maximum of Two Integers**

```
public class MaximumDemo {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        // Calling the method to find the maximum of two integers
        int max = findMaximum(a, b);
        System.out.println("Maximum of " + a + " and " + b + " is: " + max);
    }
    // Method to find the maximum of two integers
    public static int findMaximum(int x, int y) {
        return (x > y) ? x : y;
    }
}
```

Output:



```
F:\java>javac MaximumDemo.java

F:\java>java MaximumDemo
Maximum of 10 and 20 is: 20

F:\java>
```

Explanation

- In this program, we define a method findMaximum that takes two integers as parameters and returns the maximum of the two.
- We declare variables a and b in the main method and assign values to them.
- We call the findMaximum method and pass a and b as arguments.
- Inside the findMaximum method, we use the ternary operator to determine the maximum of the two integers.
- The maximum value is returned and stored in the max variable in the main method.
- Finally, we print the maximum value to the console.

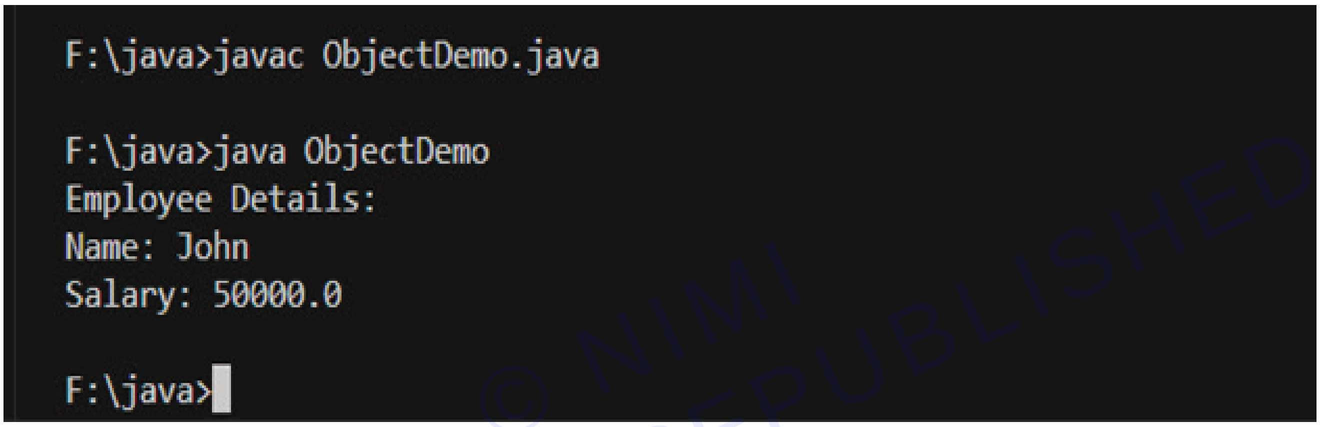
This example illustrates how to pass primitive data types, such as integers, to methods in Java. The findMaximum method encapsulates the logic for finding the maximum of two integers, which promotes code reusability and improves readability. When passing primitive data types to methods, the values of the variables are copied, and changes made to the parameters inside the method do not affect the original variables in the calling method.

2 **Passing Objects to Methods**

TASK 1: **Employee Salary**

```
class Employee {
    String name;
    double salary;
    // Constructor to initialize Employee object
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
}
public class ObjectDemo {
    public static void main(String[] args) {
        // Creating an Employee object
        Employee employee = new Employee("John", 50000);
     // Passing the employee object to a method for display
        displayEmployeeDetails(employee);
        }

        // Method to display employee details
        public static void displayEmployeeDetails(Employee employee) {
            System.out.println("Employee Details:");
            System.out.println("Name: " + employee.name);
            System.out.println("Salary: " + employee.salary);
        }
    }
```

Output:



```
F:\java>javac ObjectDemo.java

F:\java>java ObjectDemo
Employee Details:
Name: John
Salary: 50000.0

F:\java>
```

Explanation:

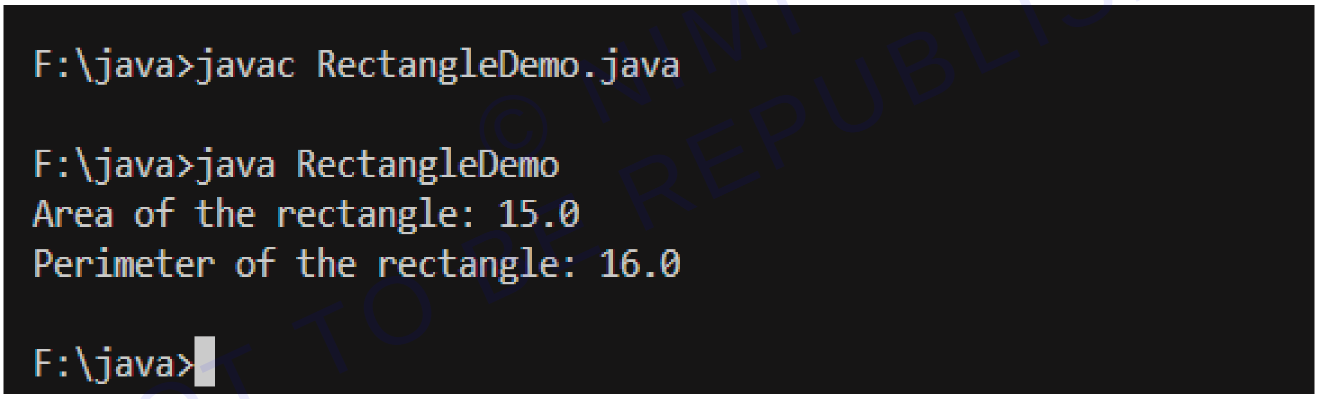
- This program defines a class Employee with name and salary attributes.
- We create an object employee of the Employee class and initialize it with name and salary values.
- We define a method displayEmployeeDetails that takes an Employee object as a parameter and displays its details.
- In the main method, we call the displayEmployeeDetails method and pass the employee object as an argument.

— — — — —

TASK 2: **Calculating Area and Perimeter of a Rectangle**

```
    class Rectangle {
        double length;
        double width;
        Rectangle(double length, double width) {
            this.length = length;
            this.width = width;
        }
    // Method to calculate the area of the rectangle
    double calculateArea() {
        return length * width;
    }
    // Method to calculate the perimeter of the rectangle
    double calculatePerimeter() {
        return 2 * (length + width);
    }
}
public class RectangleDemo {
    public static void main(String[] args) {
        // Create a Rectangle object
        Rectangle rectangle = new Rectangle(5.0, 3.0);
        // Display the calculated area and perimeter
        System.out.println("Area of the rectangle: " + rectangle.calculateArea());
        System.out.println("Perimeter of the rectangle: " + rectangle.calculatePerimeter());
    }
}
```

Output:



```
F:\java>javac RectangleDemo.java

F:\java>java RectangleDemo
Area of the rectangle: 15.0
Perimeter of the rectangle: 16.0

F:\java>
```

Explanation:

- In this program, we define a class Rectangle representing a rectangle with attributes length and width.
- The Rectangle class has methods calculateArea() and calculatePerimeter() to compute the area and perimeter of the rectangle, respectively.
- We create an instance of the Rectangle class named rectangle with a length of 5.0 units and a width of 3.0 units.
- We call the calculateArea() and calculatePerimeter() methods on the rectangle object to compute and display its area and perimeter.
- Inside the methods, we access the length and width attributes of the object using the this keyword to perform the calculations.

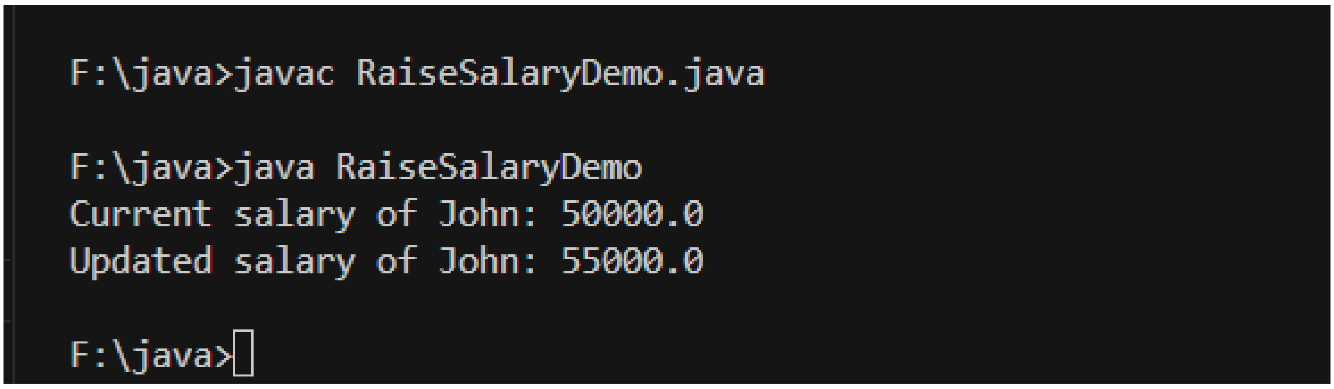
This example demonstrates how objects can encapsulate data and behavior within them, allowing us to perform operations on them effectively. By passing objects to methods, we can invoke their behavior and access their state to perform various operations and computations, making the code more organized and modular.

— — — — —

TASK 3: **Updating Employee Salary**

```
class Employee {
    String name;
    double salary;
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    // Method to raise the salary of an employee by a given percentage
    void raiseSalary(double percentage) {
        double raiseAmount = salary * (percentage / 100);
        salary += raiseAmount;
    }
}
public class RaiseSalaryDemo {
    public static void main(String[] args) {
        // Create an Employee object
        Employee emp = new Employee("John", 50000);

        // Print current salary
        System.out.println("Current salary of " + emp.name + ": " + emp.salary);

        // Call the method to raise the salary by 10%
        emp.raiseSalary(10);

        // Print updated salary
        System.out.println("Updated salary of " + emp.name + ": " + emp.salary);
    }
}
```

Output:



```
F:\java>javac RaiseSalaryDemo.java

F:\java>java RaiseSalaryDemo
Current salary of John: 50000.0
Updated salary of John: 55000.0

F:\java>
```

Explanation:

1 **Employee Class Definition**

- The program begins by defining a class named Employee.
- This class encapsulates attributes such as name and salary.
- It also includes a method named raiseSalary designed to increase an employee's salary by a specified percentage.

2 **Create Employee Object and Print Current Salary**

- Within the main method, an instance of the Employee class is created and named emp.
- The program prints the current salary of the employee (emp) to the console.

3 **Method to Raise Employee Salary**

- A method named raiseEmployeeSalary is introduced.
- This method takes an Employee object and a percentage as parameters.
- Internally, it calls the raiseSalary method of the Employee object, leading to an increase in the salary.

4 **Raise Employee Salary in Main Method**

- The raiseEmployeeSalary method is invoked within the main method, aiming to raise the salary of the employee (emp) by 10%.

5 **Print Updated Salary**

- Subsequently, the program prints the updated salary of the employee (emp) to reflect the changes made by the raiseEmployeeSalary method.

6 **Conclusion**

- This program serves as a practical example of how to pass objects, specifically instances of the Employee class, to methods in Java.
- The design of the Employee class, along with the use of methods, illustrates the principles of object-oriented programming (OOP), encapsulation, and the effective modification of object states.

  This explanation aims to provide a conceptual understanding of the program without delving into the specific code details. If you have further questions or need clarification on any specific aspect, feel free to ask!

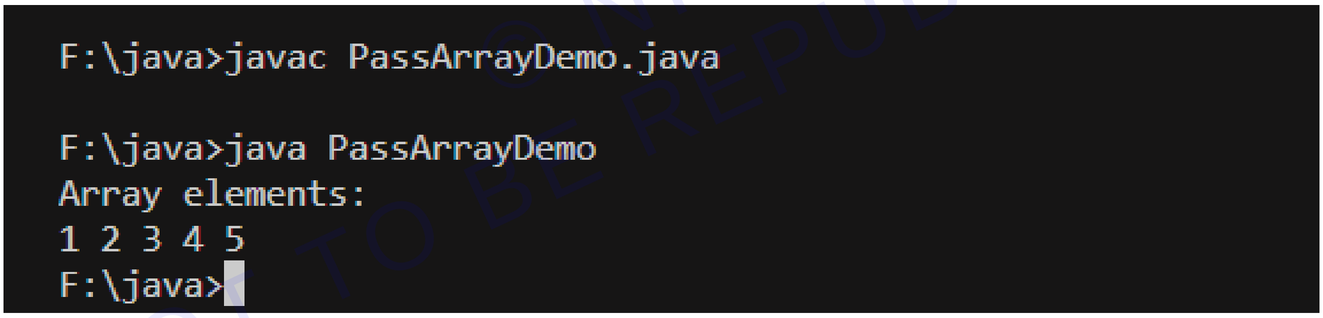
— — — — —

3 Passing Arrays to Methods

TASK 1: **to pass data and objects to methods in Java programs**

```
public class ArrayDemo {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};

        // Passing an array to a method
        printArray(numbers);
    }
    // Method to print array elements
    public static void printArray(int[] arr) {
        System.out.println("Array elements:");
        for (int num : arr) {
            System.out.print(num + " ");
        }
    }
}
```

Output:



```
F:\java>javac PassArrayDemo.java

F:\java>java PassArrayDemo
Array elements:
1 2 3 4 5
F:\java>
```

Explanation:

- In this program, we define an array numbers containing integers.
- We define a method printArray that takes an array of integers as a parameter and prints its elements.
- In the main method, we call the printArray method and pass the numbers array as an argument.

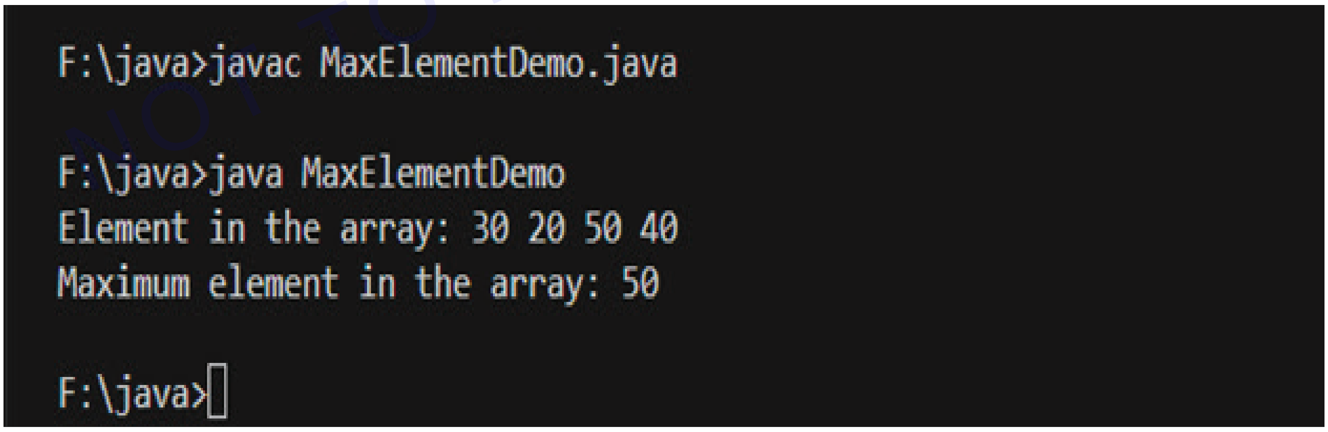
These examples demonstrate how to pass data and objects to methods in Java programs. It's important to note that Java is pass-by-value, meaning that when you pass a parameter to a method, a copy of the parameter's value is passed, not the actual object itself. For objects, the value passed is the reference to the object in memory. Thus, changes made to parameters inside the method are reflected in the original objects or variables if they are mutable.

— — — — —

TASK 2: **Finding the Maximum Element in an Array**

```
    public class MaxElementDemo {
      public static void main(String[] args) {
          int[] numbers = {10, 30, 20, 50, 40};
           System.out.print("Element in the array: " );
          // Calling the method to find the maximum element
          int max = findMaxElement(numbers);
          System.out.println();
          System.out.println("Maximum element in the array: " + max);
      }
         // Method to find the maximum element in an array
      public static int findMaxElement(int[] arr) {
          int max = arr[0];
          for (int i = 1; i < arr.length; i++) {
               System.out.print(arr[i] +" ");
              if (arr[i] > max) {
                  max = arr[i];
              }
          }
          return max;
      }
    }
```

Output:



```
F:\java>javac MaxElementDemo.java

F:\java>java MaxElementDemo
Element in the array: 30 20 50 40
Maximum element in the array: 50

F:\java>
```

Explanation:

- In this program, we define a method findMaxElement that takes an array of integers as a parameter and returns the maximum element in the array.
- We declare an array numbers containing integers in the main method.
- We call the findMaxElement method and pass the numbers array as an argument.

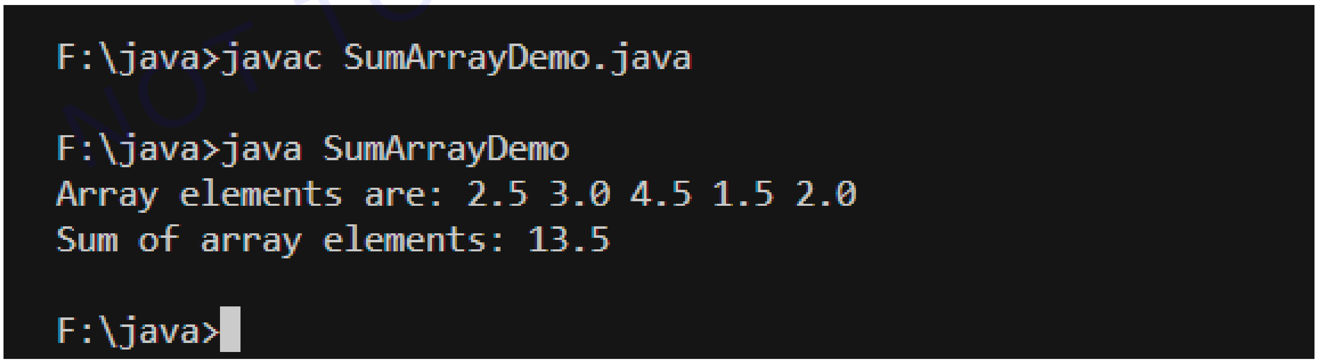
- Inside the findMaxElement method, we iterate through the array elements and update the max variable if we encounter a larger element.
- The maximum element is returned and stored in the max variable in the main method.
- Finally, we print the maximum element to the console.

— — — — —

TASK 3: **Calculating the Sum of Array Elements**

```
public class SumArrayDemo {
   public static void main(String[] args) {
      double[] values = {2.5, 3.0, 4.5, 1.5, 2.0};
      // Calling the method to calculate the sum of array elements
      double sum = calculateSum(values);
      System.out.println("Sum of array elements: " + sum);
   }
   // Method to calculate the sum of array elements
   public static double calculateSum(double[] arr) {
      double sum = 0;
      for (double value : arr) {
         sum += value;
      }
      return sum;
   }
}
```

Output:



```
F:\java>javac SumArrayDemo.java

F:\java>java SumArrayDemo
Array elements are: 2.5 3.0 4.5 1.5 2.0
Sum of array elements: 13.5

F:\java>
```

Explanation:

- In this program, we define a method calculateSum that takes an array of doubles as a parameter and returns the sum of the array elements.
- We declare an array values containing double values in the main method.
- We call the calculateSum method and pass the values array as an argument.
- Inside the calculateSum method, we iterate through the array elements and accumulate their sum in the sum variable.

- The sum of array elements is returned and stored in the sum variable in the main method.
- Finally, we print the sum of array elements to the console.

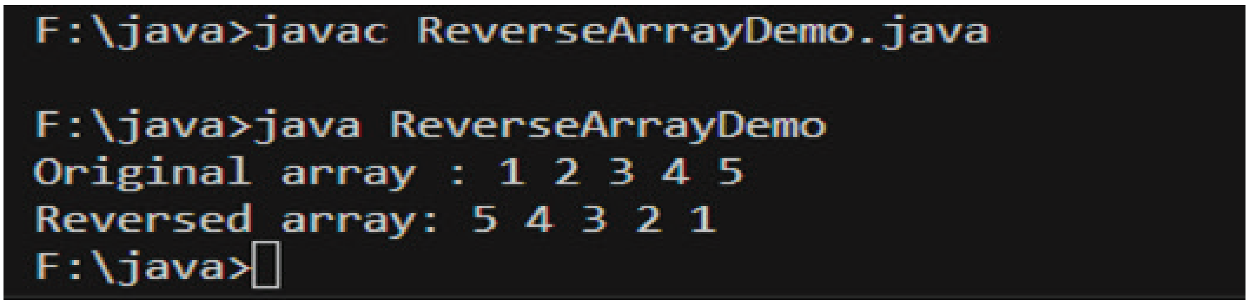
These examples illustrate how to pass arrays to methods in Java. By passing arrays to methods, we can perform operations on arrays effectively and encapsulate array-related logic within methods, promoting code reusability and readability.

— — — — —

TASK 4: **Reversing an Array**

```
    // Reverse Array Demo
 public class ReverseArrayDemo {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        // Displaying the original array
        System.out.print("Original array : ");
        for (int num : numbers) {
            System.out.print(num + " ");
        }
        // Calling the method to reverse the array
        reverseArray(numbers);
        // Displaying the reversed array
        System.out.println();
        System.out.print("Reversed array: ");
        for (int num : numbers) {
            System.out.print(num + " ");
        }
    }
    // Method to reverse an array
    public static void reverseArray(int[] numbers) {
        int left = 0;
        int right = numbers.length - 1;
        while (left < right) {
            // Swap elements at left and right indices
            int temp = numbers[left];
            numbers[left] = numbers[right];
            numbers[right] = temp;
            // Move left index to the right and right index to the left
            left++;
            right--;
        }
    }
 }
```

Output:



```
F:\java>javac ReverseArrayDemo.java

F:\java>java ReverseArrayDemo
Original array : 1 2 3 4 5
Reversed array: 5 4 3 2 1
F:\java>
```

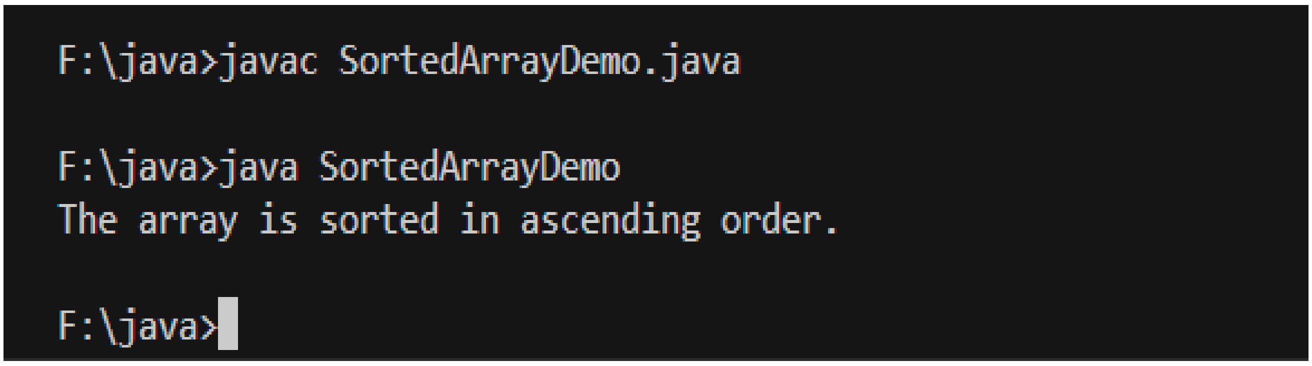
Explanation:

- In this program, we define a method reverseArray that takes an array of integers as a parameter and reverses the elements of the array in place.
- We declare an array numbers containing integers in the main method.
- We call the reverseArray method and pass the numbers array as an argument.
- Inside the reverseArray method, we use two pointers (left and right) to traverse the array from both ends and swap the elements until they meet in the middle.
- After reversing the array, we display the elements of the reversed array in the main method.

TASK 5: **Checking if an Array is Sorted in Ascending Order**

```
public class SortedArrayDemo {
  public static void main(String[] args) {
    int[] numbers = {1, 3, 5, 7, 9};
    // Calling the method to check if the array is sorted
    boolean sorted = isSorted(numbers);
    if (sorted) {
      System.out.println("The array is sorted in ascending order.");
    } else {
      System.out.println("The array is not sorted in ascending order.");
    }
  }
  // Method to check if an array is sorted in ascending order
  public static boolean isSorted(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
      if (arr[i] > arr[i + 1]) {
        return false;
      }
    }
    return true;
  }
}
```

Output:



```
F:\java>javac SortedArrayDemo.java

F:\java>java SortedArrayDemo
The array is sorted in ascending order.

F:\java>
```

Explanation:

- In this program, we define a method isSorted that takes an array of integers as a parameter and checks if the elements of the array are sorted in ascending order.
- We declare an array numbers containing integers in the main method.
- We call the isSorted method and pass the numbers array as an argument.
- Inside the isSorted method, we iterate through the array and compare adjacent elements to check if they are in ascending order.
- If any element is greater than the next element, we return false, indicating that the array is not sorted.
- If all elements are in ascending order, we return true, indicating that the array is sorted.

These examples demonstrate various operations that can be performed on arrays by passing them to methods in Java. Passing arrays to methods allows us to encapsulate array-related logic and promote code reusability and readability.

EXERCISE 96 : Return data and Objects from Methods

Objectives

At the end of this exercise you shall be able to

- know about how to return data and objects from methods in JAVA
- develop Java programs to return data and object from methods.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

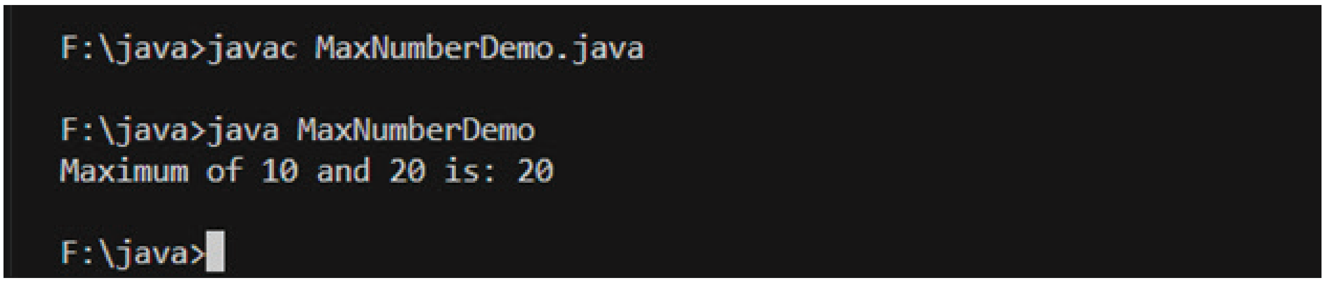
Procedure

Below are examples of Java programs demonstrating returning data and objects from methods:

TASK 1: **Returning the Maximum of Two Numbers**

```
public class MaxNumberDemo {
  public static void main(String[] args) {
    int a = 10;
    int b = 20;
    // Calling the method to find the maximum of two numbers
    int max = findMax(a, b);
    System.out.println("Maximum of " + a + " and " + b + " is: " + max);
  }
  // Method to find the maximum of two numbers
  public static int findMax(int x, int y) {
    return (x > y) ? x : y;
  }
}
```

Output:



```
F:\java>javac MaxNumberDemo.java

F:\java>java MaxNumberDemo
Maximum of 10 and 20 is: 20

F:\java>
```

Explanation:

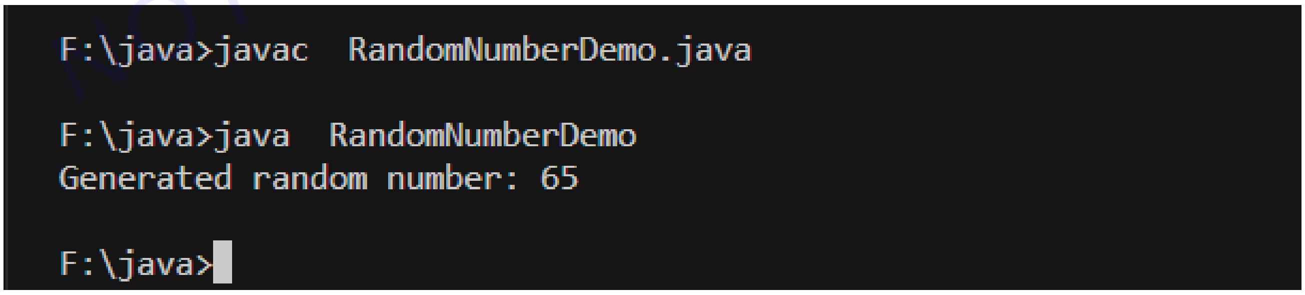
- In this program, we define a method findMax that takes two integers as parameters and returns the maximum of the two.
- We declare variables a and b in the main method and assign values to them.
- We call the findMax method and pass a and b as arguments.
- Inside the findMax method, we use the ternary operator to determine the maximum of the two integers.
- The maximum value is returned and stored in the max variable in the main method.
- Finally, we print the maximum value to the console.

— — — — —

TASK 2: **Generating a Random Number**

```
import java.util.Random;
public class RandomNumberDemo {
    public static void main(String[] args) {
        // Calling the method to generate a random number
        int randomNumber = generateRandomNumber();
        System.out.println("Generated random number: " + randomNumber);
    }
    // Method to generate a random number
    public static int generateRandomNumber() {
        Random random = new Random();
        return random.nextInt(100); // Generates a random number between 0 and 99
    }
}
```

Output:



```
F:\java>javac  RandomNumberDemo.java

F:\java>java  RandomNumberDemo
Generated random number: 65

F:\java>
```

Explanation:

- In this program, we import the Random class from the java.util package to generate random numbers.
- We define a method generateRandomNumber that returns a randomly generated integer.
- Inside the generateRandomNumber method, we create an instance of the Random class and use its nextInt method to generate a random integer between 0 and 99.
- The generated random number is returned to the main method, where it is stored in the randomNumber variable.
- Finally, we print the generated random number to the console.

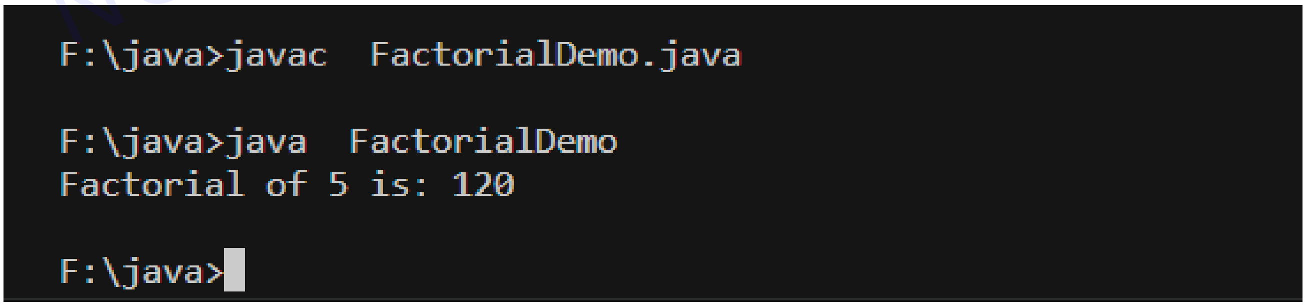
These examples demonstrate how to return data from methods in Java. The findMax method returns the maximum of two numbers, while the generateRandomNumber method returns a randomly generated integer. By returning data from methods, we can encapsulate logic and computations, making our code more modular and reusable.

— — — — —

TASK 3: **Calculating the Factorial of a Number**

```
public class FactorialDemo {
    public static void main(String[] args) {
        int n = 5;
        // Calling the method to calculate the factorial of a number
        long factorial = calculateFactorial(n);
        System.out.println("Factorial of " + n + " is: " + factorial);
    }
    // Method to calculate the factorial of a number
    public static long calculateFactorial(int n) {
        if (n == 0) {
            return 1;
        } else {
            long factorial = 1;
            for (int i = 1; i <= n; i++) {
                factorial *= i;
            }
            return factorial;
        }
    }
}
```

Output:



```
F:\java>javac  FactorialDemo.java

F:\java>java  FactorialDemo
Factorial of 5 is: 120

F:\java>
```

Explanation:

- In this program, we define a method calculateFactorial that takes an integer n as a parameter and returns the factorial of n.
- We declare a variable n in the main method and assign a value to it.
- We call the calculateFactorial method and pass n as an argument.
- Inside the calculateFactorial method, we use a for loop to calculate the factorial of n.

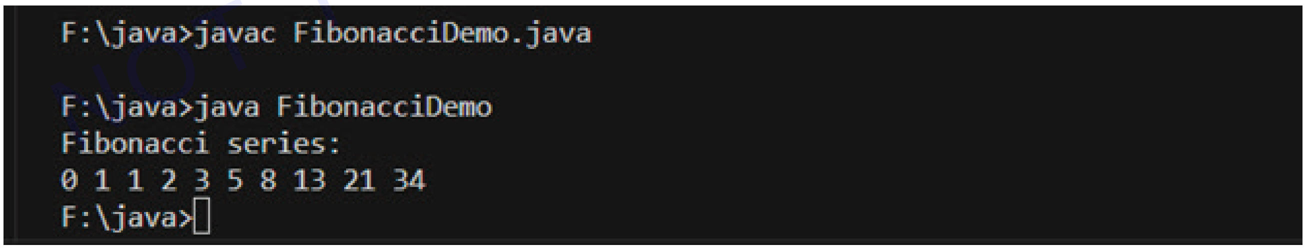
- The calculated factorial is returned to the main method, where it is stored in the factorial variable.
- Finally, we print the factorial of n to the console.

— — — — —

TASK 4: **Generating Fibonacci Series**

```
public class FibonacciDemo {
    public static void main(String[] args) {
        int n = 10;
        // Calling the method to generate Fibonacci series
        System.out.println("Fibonacci series:");
        for (int i = 0; i < n; i++) {
            System.out.print(fibonacci(i) + " ");
        }
    }
    // Method to generate the nth Fibonacci number
    public static int fibonacci(int n) {
        if (n <= 1) {
            return n;
        } else {
            return fibonacci(n - 1) + fibonacci(n - 2);
        }
    }
}
```

Output:



```
F:\java>javac FibonacciDemo.java

F:\java>java FibonacciDemo
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
F:\java>
```

Explanation:

- In this program, we define a method fibonacci that takes an integer n as a parameter and returns the n-th Fibonacci number.
- We declare a variable n in the main method and assign a value to it.
- We iterate from 0 to n in the main method and print the Fibonacci series using the fibonacci method.
- Inside the fibonacci method, we use recursion to calculate the Fibonacci number for each value of n.
- The calculated Fibonacci number is returned to the caller.

These examples illustrate how to return data from methods in Java. The calculateFactorial method returns the factorial of a number, while the fibonacci method returns the nth Fibonacci number. Returning data from methods allows us to perform calculations and computations, making our code more modular and reusable.

EXERCISE 97 : Use constructors in JAVA

Objectives

At the end of this exercise you shall be able to

- know about constructors and its function in JAVA
- develop Java programs using different constructors such as default and parameterised constructors
- develop Java programs using Constructor chaining.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

Constructors in Java can have various forms, including default constructors, parameterized constructors, and constructor chaining. Here are examples demonstrating the use of various constructors in Java:

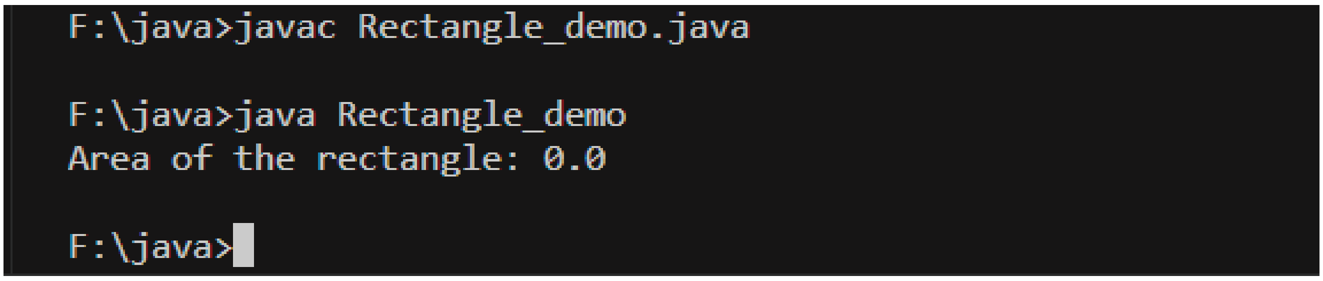
1 Default Constructor:

TASK 1: **Area of Rectangle**

```
// Area of Rectangle using  default constructor
 class Rectangle {
   double length;
   double width;
   // Default constructor
   Rectangle() {
       length = 0;
       width = 0;
   }
   // Method to calculate the area of the rectangle
   double calculateArea() {
       return length * width;
   }
}
public class Rectangle_Demo {
   public static void main(String[] args) {
       // Creating an instance of Rectangle using the default constructor
       Rectangle rectangle = new Rectangle();
       // Displaying the area of the rectangle
```

```
        System.out.println("Area of the rectangle: " + rectangle.calculateArea());
    }
}
```

Output:



```
F:\java>javac Rectangle_demo.java

F:\java>java Rectangle_demo
Area of the rectangle: 0.0

F:\java>
```

Explanation:

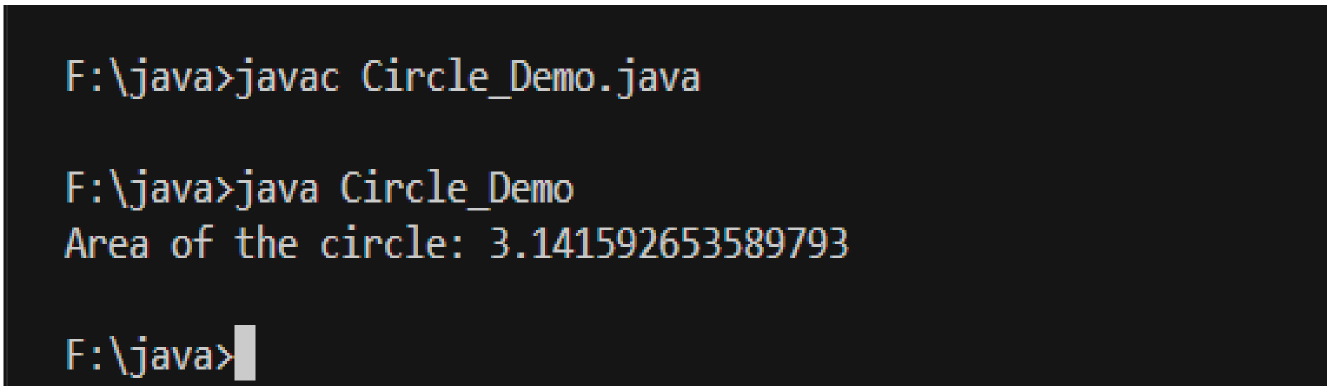
- In this program, we define a class Rectangle with attributes length and width.
- We provide a default constructor for the Rectangle class, which initializes the length and width attributes to zero.
- In the main method, we create an instance of the Rectangle class using the default constructor.
- We call the calculateArea method of the Rectangle class to calculate and display the area of the rectangle.

— — — — —

TASK 2: **Default Constructor with Initialization: Area of a Circle**

```
//Area of a Circle using Default Constructor with Initialization
 class Circle {
    double radius;
    // Default constructor
    Circle() {
        radius = 1.0; // Initialize radius to a default value
    }
    // Method to calculate the area of the circle
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
public class Circle_Demo {
    public static void main(String[] args) {
        // Creating an instance of Circle using the default constructor
        Circle circle = new Circle();
        // Displaying the area of the circle
        System.out.println("Area of the circle: " + circle.calculateArea());
        }
    }
```

Output:



```
F:\java>javac Circle_Demo.java

F:\java>java Circle_Demo
Area of the circle: 3.141592653589793

F:\java>
```

Explanation:

- In this program, we define a class Circle with an attribute radius.
- We provide a default constructor for the Circle class, which initializes the radius attribute to a default value (1.0 in this case).
- In the main method, we create an instance of the Circle class using the default constructor.
- We call the calculateArea method of the Circle class to calculate and display the area of the circle.

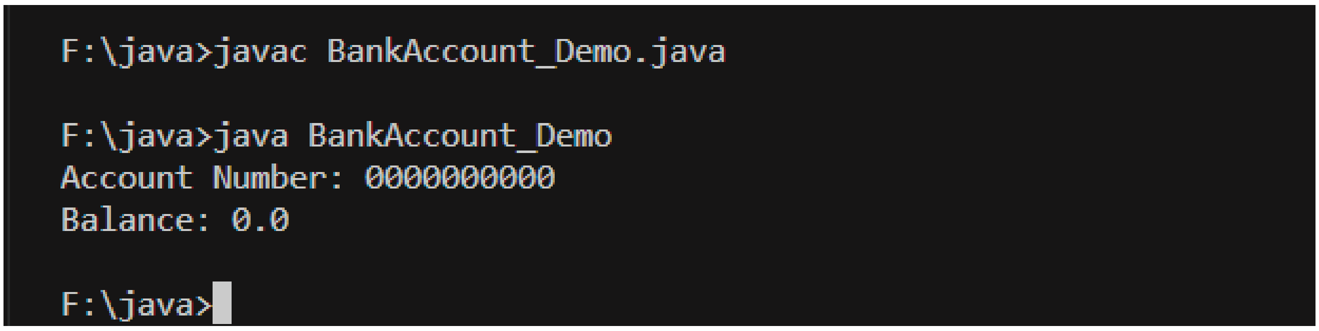
— — — — —

TASK 3: **Default Constructor in a Bank Account Class**

```
// Bank Account using default constructors
class BankAccount {
    String accountNumber;
    double balance;
    // Default constructor
    BankAccount() {
        accountNumber = "0000000000"; // Default account number
        balance = 0.0; // Initialize balance to zero
    }
    // Method to display account details
    void displayAccountDetails() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Balance: " + balance);
    }
}
public class BankAccount_Demo {
    public static void main(String[] args) {
        // Creating an instance of BankAccount using the default constructor
        BankAccount account = new BankAccount();
```

```
            // Displaying account details
            account.displayAccountDetails();
        }
    }
```

Output:



```
F:\java>javac BankAccount_Demo.java

F:\java>java BankAccount_Demo
Account Number: 0000000000
Balance: 0.0

F:\java>
```

Explanation:

- In this program, we define a class BankAccount with attributes accountNumber and balance.
- We provide a default constructor for the BankAccount class, which initializes the accountNumber attribute to a default value ("0000000000") and the balance attribute to zero.
- In the main method, we create an instance of the BankAccount class using the default constructor.
- We call the displayAccountDetails method of the BankAccount class to display the account details.

Default constructors are invoked automatically when an object is created if no other constructor is explicitly defined. They initialize the object's attributes to default values, ensuring that the object is in a consistent state upon creation. Default constructors are particularly useful when no specific initialization is required.

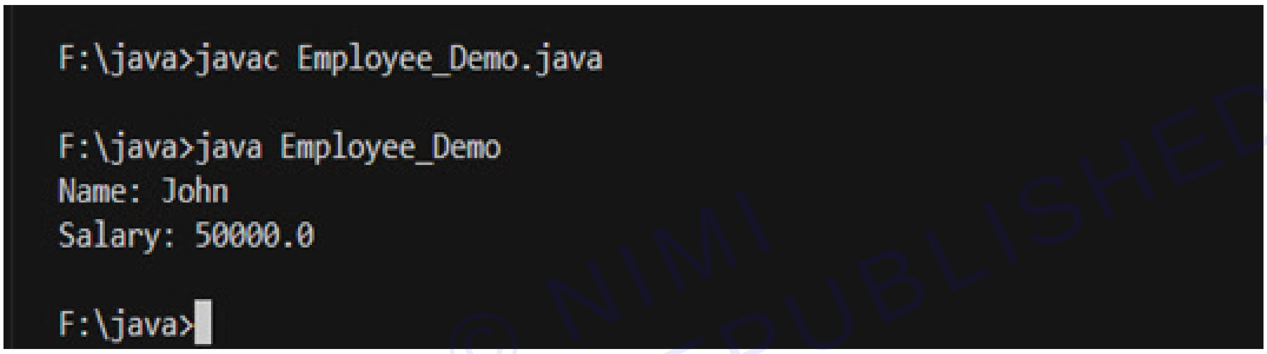
— — — — —

2 Parameterized Constructor

TASK 1: **Salary details of an employee**

```
//Salary details of an employee
class Employee {
    String name;
    double salary;
    // Parameterized constructor
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    // Method to display employee details
    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
    }
}
public class Employee_Demo {
    public static void main(String[] args) {
        // Creating an instance of Employee using the parameterized constructor
        Employee employee = new Employee("John", 50000);

        // Displaying employee details
        employee.displayDetails();
    }
}
```

Output:



```
F:\java>javac Employee_Demo.java

F:\java>java Employee_Demo
Name: John
Salary: 50000.0

F:\java>
```

Explanation:

- In this program, we define a class Employee with attributes name and salary.
- We provide a parameterized constructor for the Employee class, which initializes the name and salary attributes with the values passed as arguments.
- In the main method, we create an instance of the Employee class using the parameterized constructor, passing values for the name and salary.
- We call the displayDetails method of the Employee class to display the employee details.

These examples demonstrate the use of different types of constructors in Java. Constructors are used to initialize the state of objects during their creation. Default constructors initialize the attributes to default values, while parameterized constructors allow us to initialize the attributes with specific values. Understanding and using constructors effectively is essential for object-oriented programming in Java.

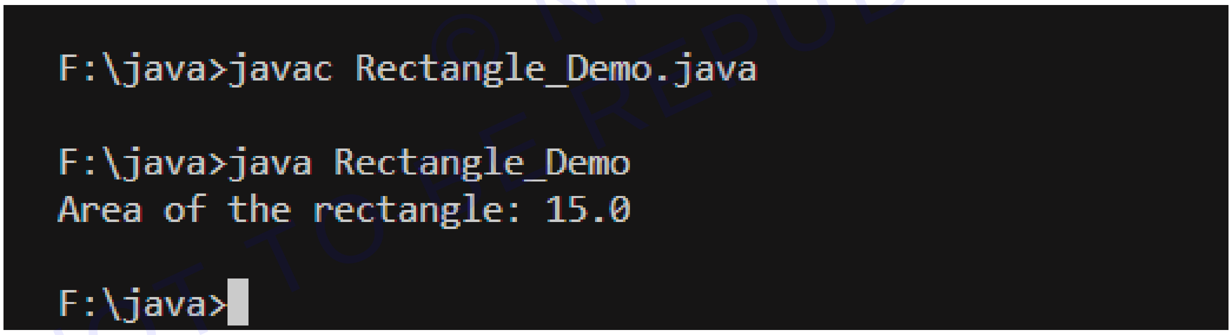
— — — — —

TASK 2: **Parameterized Constructor in a Rectangle Class**

```
// Area of a rectangle using Parameterized Constructor
class Rectangle {
    double length;
    double width;
    // Parameterized constructor
    Rectangle(double length, double width) {
        this.length = length;
```

```
            this.width = width;
        }
        // Method to calculate the area of the rectangle
        double calculateArea() {
            return length * width;
        }
    }
    public class RectangleDemo {
        public static void main(String[] args) {
            // Creating an instance of Rectangle using the parameterized constructor
            Rectangle rectangle = new Rectangle(5.0, 3.0);

            // Displaying the area of the rectangle
            System.out.println("Area of the rectangle: " + rectangle.calculateArea());
        }
    }
```

Output:



```
F:\java>javac Rectangle_Demo.java

F:\java>java Rectangle_Demo
Area of the rectangle: 15.0

F:\java>
```

Explanation:

- In this program, we define a class Rectangle with attributes length and width.
- We provide a parameterized constructor for the Rectangle class, which initializes the length and width attributes with the values passed as parameters.
- In the main method, we create an instance of the Rectangle class using the parameterized constructor, passing specific values for the length and width.
- We call the calculateArea method of the Rectangle class to calculate and display the area of the rectangle.

— — — — —

TASK 3: **Parameterized Constructor in a Student Class**

```
    // Student Class  using Parameterized Constructor
    class Student {
        String name;
        int age;
    // Parameterized constructor
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    // Method to display student details
    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
public class StudentDemo {
    public static void main(String[] args) {
        // Creating an instance of Student using the parameterized constructor
        Student student = new Student("John", 20);
        // Displaying student details
        student.displayDetails();
    }
}
```

Explanation:

- In this program, we define a class Student with attributes name and age.
- We provide a parameterized constructor for the Student class, which initializes the name and age attributes with the values passed as parameters.
- In the main method, we create an instance of the Student class using the parameterized constructor, passing specific values for the name and age.
- We call the displayDetails method of the Student class to display the student details.

Parameterized constructors allow you to initialize object attributes with specific values at the time of object creation. They provide flexibility and convenience in setting initial state for objects, making them ready for use.

— — — — —

3 Constructor chaining in Java

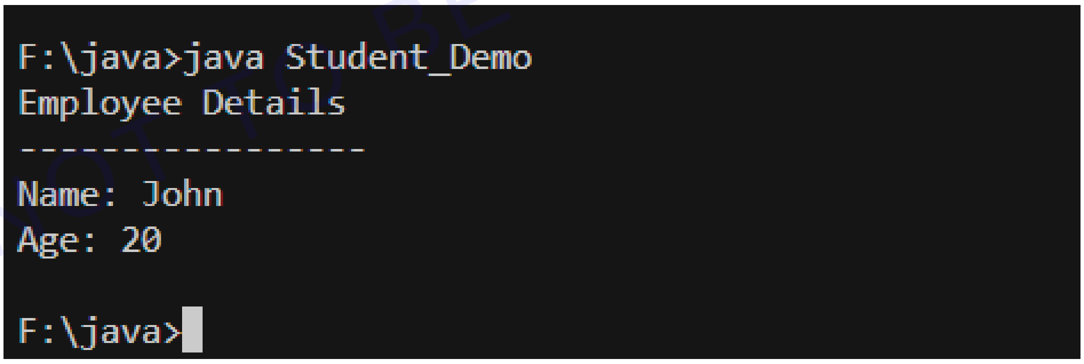
Constructor chaining in Java refers to the process of calling one constructor from another constructor within the same class or from the constructor of the superclass. Here are examples of constructor chaining with explanations:

TASK 1: **Constructor Chaining within the Same Class**

```
class Student {
    String name;
    int age;
    // Parameterized constructor
```

```
        Student(String name, int age) {
            this.name = name;
            this.age = age;
        }
        // Method to display student details
        void displayDetails() {
            System.out.println("Employee Details " );
            System.out.println("-----------------" );
            System.out.println("Name: " + name);
            System.out.println("Age: " + age);
        }
    }
    public class Student_Demo {
        public static void main(String[] args) {
            // Creating an instance of Student using the parameterized constructor
            Student student = new Student("John", 20);
            // Displaying student details
            student.displayDetails();
        }
    }
```

Output:



```
F:\java>java Student_Demo
Employee Details
-----------------
Name: John
Age: 20

F:\java>
```

Explanation:

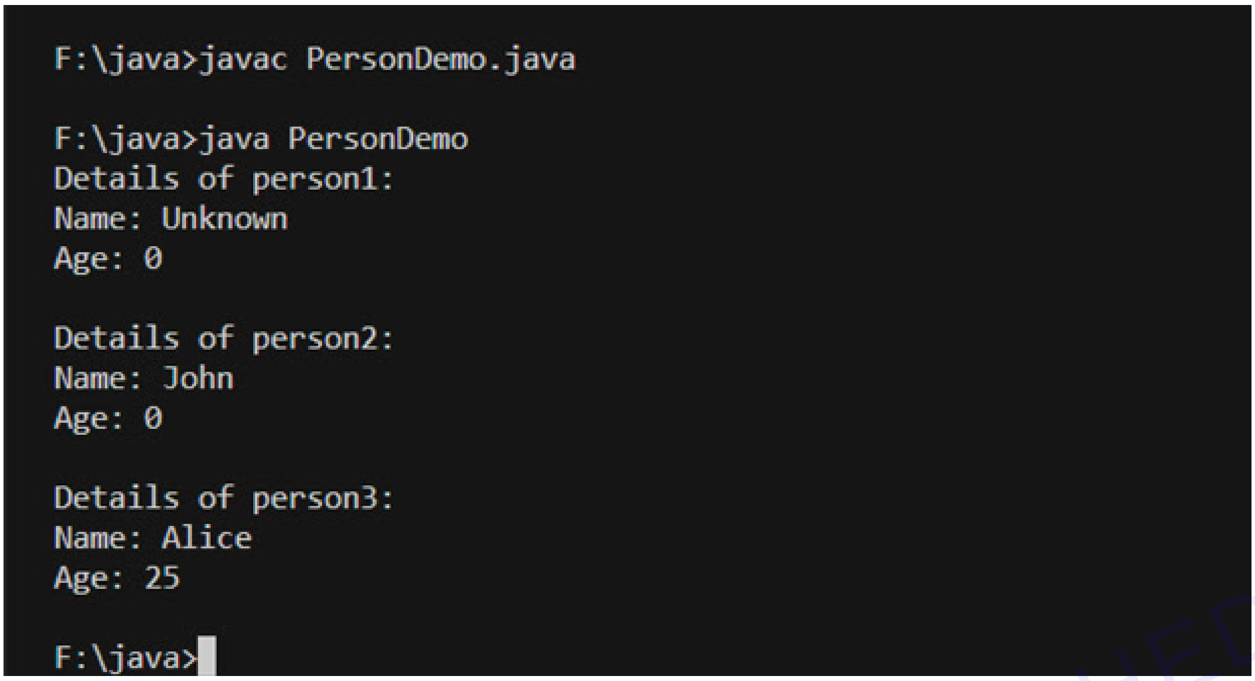
- In this example, we have a class MyClass with three constructors.
- The first constructor calls the second constructor using this(0, 0), which initializes x and y to 0.
- The second constructor calls the third constructor using this(x, y, "Default"), which initializes x, y, and name.
- The third constructor initializes all the attributes x, y, and name based on the provided arguments.
- In the main method, we create an instance of MyClass using the first constructor, which in turn triggers the chain of constructors.
- The display method is called to print the object's attributes.

— — — — —

TASK 2: **Details of Persons**

```
class Person {
    String name;
    int age;
    // Default constructor
    Person() {
        this("Unknown", 0); // Call parameterized constructor with default values
    }
    // Parameterized constructor with name
    Person(String name) {
        this(name, 0); // Call parameterized constructor with default age
    }
    // Parameterized constructor with name and age
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    void display() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
public class PersonDemo {
    public static void main(String[] args) {
        // Creating instances of Person with different constructors
        Person person1 = new Person();
        Person person2 = new Person("John");
        Person person3 = new Person("Alice", 25);

        // Displaying details of persons
        System.out.println("Details of person1:");
        person1.display();
        System.out.println("\nDetails of person2:");
        person2.display();
        System.out.println("\nDetails of person3:");
        person3.display();
    }
}
```

Output:



```
F:\java>javac PersonDemo.java

F:\java>java PersonDemo
Details of person1:
Name: Unknown
Age: 0

Details of person2:
Name: John
Age: 0

Details of person3:
Name: Alice
Age: 25

F:\java>
```

Explanation:

- In this program, we have a class Person with three constructors.
- The default constructor initializes the person with default values by calling the parameterized constructor with default values.
- The parameterized constructor with only the name initializes the person with the provided name and default age by calling the parameterized constructor with default age.
- The parameterized constructor with both name and age initializes the person with the provided name and age.
- The display method prints the details of the person, including name and age.
- In the main method, we create instances of Person using different constructors to demonstrate constructor chaining.
- Finally, we call the display method to print the details of each person.

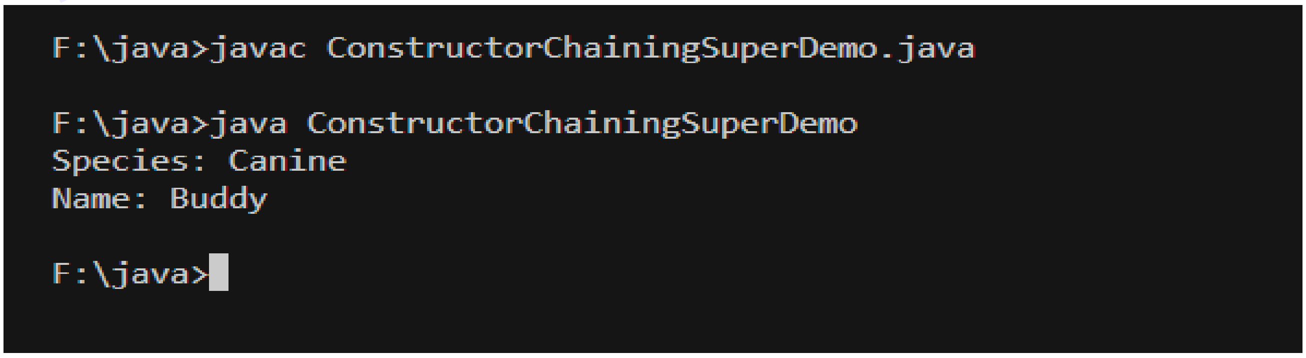
This program illustrates how constructor chaining within the same class can be used to provide multiple ways of object initialization while avoiding code duplication. It allows for cleaner and more concise code by reusing constructor logic.

— — — — —

TASK 3: **Constructor Chaining with Superclass**

```
class Animal {
    String species;
    // Constructor of superclass
    Animal(String species) {
        this.species = species;
    }
    void displaySpecies() {
            System.out.println("Species: " + species);
        }
    }
    class Dog extends Animal {
        String name;
        // Constructor of subclass
        Dog(String species, String name) {
            super(species); // Call superclass constructor
            this.name = name;
        }
        void displayName() {
            System.out.println("Name: " + name);
        }
    }
    public class ConstructorChainingSuperDemo {
        public static void main(String[] args) {
            // Creating an instance of Dog
            Dog dog = new Dog("Canine", "Buddy");
            // Displaying species and name of the dog
            dog.displaySpecies();
            dog.displayName();
        }
    }
```

Output:



```
F:\java>javac ConstructorChainingSuperDemo.java

F:\java>java ConstructorChainingSuperDemo
Species: Canine
Name: Buddy

F:\java>
```

Explanation:

- In this example, we have a superclass Animal and a subclass Dog.
- The superclass Animal has a parameterized constructor that initializes the species attribute.
- The subclass Dog has a parameterized constructor that initializes the name attribute and calls the superclass constructor using super(species).
- In the main method, we create an instance of Dog with a species of “Canine” and a name of “Buddy”.
- We call methods to display the species and name of the dog.

Constructor chaining allows us to reuse code and avoid redundancy by calling one constructor from another. In the first example, the constructor chaining within the same class helps in providing multiple ways to initialize object attributes. In the second example, constructor chaining between superclass and subclass helps in initializing attributes of both classes efficiently.

EXERCISE 98 : Create and use Overloaded methods in JAVA

Objectives

At the end of this exercise you shall be able to

- know overloaded methods in JAVA
- develop Java programs using overloaded methods.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Overloaded methods in Java are methods that have the same name but different parameter lists. Here are a few examples demonstrating the creation and usage of overloaded methods:

Procedure

TASK 1: **Calculating the Area of Shapes**

```
// Area of Shapes using overloaded methods
import java.io.*;
public class GeometricalShapes {
// Method to calculate the area of a square
public static double calculateArea(int side) {
return side * side;
}
// Method to calculate the area of a rectangle
public static double calculateArea(long length, long  width) {
return length * width;
}
// Method to calculate the area of a circle
public static double calculateArea(double radius) {
return Math.PI * radius * radius;
}
// Method to calculate the area of a triangle
public static double calculateArea(double base, double height) {
return 0.5 * base * height;
}
public static void main(String[] args) {
int side = 5;
long length =6L;
long width = 4L;
double radius = 3.0;
double base = 8.0;
double height = 5.0;
// Calculate areas using overloaded methods
System.out.println("Area of square: " + calculateArea(side));
System.out.println("Area of rectangle: " + calculateArea(length, width));
System.out.println("Area of circle: " + calculateArea(radius));
System.out.println("Area of triangle: " + calculateArea(base, height));
}
}
```

Explanation:

1 **Method Overloading:**

- The program uses method overloading to define multiple versions of the calculateArea method with different parameter types or numbers.
- This allows the same method name to be used for calculating the area of various geometrical shapes.

2 **Calculate Area of a Square:**

- The first overloaded method, calculateArea(int side), is designed to calculate the area of a square.
- It takes an integer parameter (side) representing the side length and returns the calculated area using the formula side * side.

3 **Calculate Area of a Rectangle:**

- The second overloaded method, calculateArea(long length, long width), is tailored for calculating the area of a rectangle.
- It takes two long integer parameters (length and width) representing the length and width, and returns the calculated area using the formula length * width.

4 **Calculate Area of a Circle:**

- The third overloaded method, calculateArea(double radius), is specialized for calculating the area of a circle.
- It takes a double parameter (radius) representing the radius and returns the calculated area using the formula Math.PI * radius * radius.

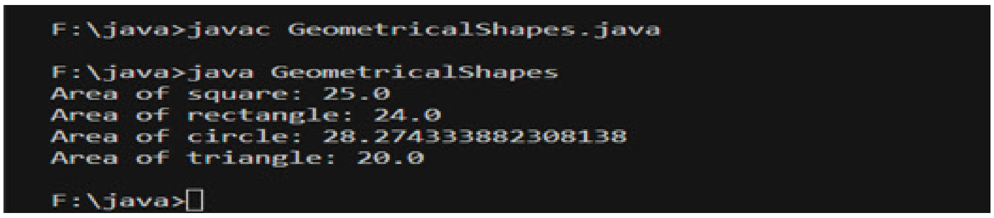
5 **Calculate Area of a Triangle:**

- The fourth overloaded method, calculateArea(double base, double height), is designed for calculating the area of a triangle.
- It takes two double parameters (base and height) representing the base and height, and returns the calculated area using the formula 0.5 * base * height.

6 **Main Method:**

- In the main method, sample values for the dimensions of a square, rectangle, circle, and triangle are declared.
- The overloaded methods are then called with these values to calculate and print the areas of the respective shapes.

Output:



```
F:\java>javac GeometricalShapes.java

F:\java>java GeometricalShapes
Area of square: 25.0
Area of rectangle: 24.0
Area of circle: 28.274333882308138
Area of triangle: 20.0

F:\java>
```

The output displays the calculated areas for each shape.

Conclusion:

- The program effectively demonstrates the concept of method overloading to provide a single, consistent interface (calculateArea) for calculating areas of different geometrical shapes.
- The choice of appropriate data types (int, long, double) for different parameters adds flexibility to the program.

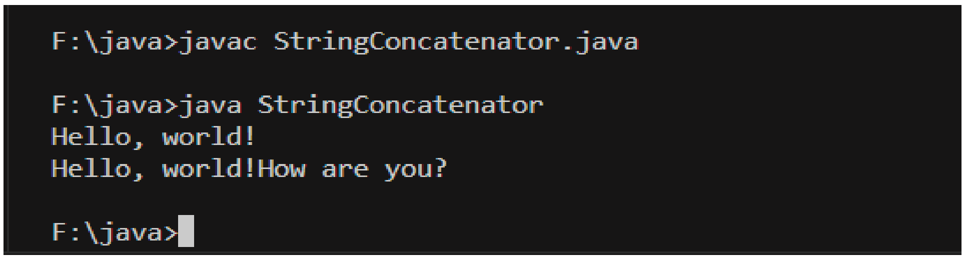
Overall, the program showcases a clean and modular approach to handle various geometrical calculations in a concise manner.

— — — — —

TASK 2: **Overloaded Methods for String Concatenation**

```
// String Concatenation using Overloaded Methods
import java.io.*;
public class StringConcatenator {
// Method to concatenate two strings
public static String concatenate(String str1, String str2) {
return str1 + str2;
}
// Method to concatenate three strings
public static String concatenate(String str1, String str2, String str3) {
return str1 + str2 + str3;
}
public static void main(String[] args) {
String str1 = "Hello, ";
String str2 = "world!";
String str3 = "How are you?";
// Concatenate strings using overloaded methods
System.out.println(concatenate(str1, str2));
System.out.println(concatenate(str1, str2, str3));
}
}
```

Output:



```
F:\java>javac StringConcatenator.java

F:\java>java StringConcatenator
Hello, world!
Hello, world!How are you?

F:\java>
```

Explanation:

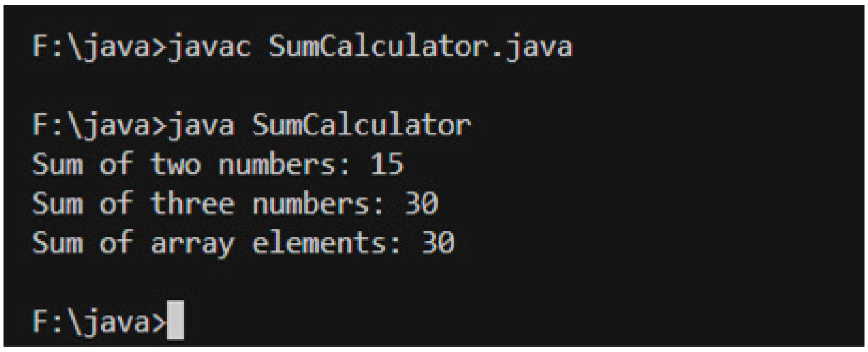
- In this program, we have a class StringConcatenator with two overloaded methods named concatenate.
- The first method concatenates two strings.
- The second method concatenates three strings.
- In the main method, we call each of the overloaded methods to concatenate strings with different numbers of parameters.

These examples illustrate how overloaded methods allow us to create methods with the same name but different behaviors based on the parameters passed to them. It enhances code readability and reusability by providing multiple ways to accomplish a task.

— — — — —

TASK 3: **Summing Numbers**

```
// Summing Numbers  using Overloaded Methods
public class SumCalculator {
// Method to sum two integers
public static int sum(int a, int b) {
return a + b;
}
// Method to sum three integers
public static int sum(int a, int b, int c) {
return a + b + c;
}
// Method to sum an array of integers
public static int sum(int[] numbers) {
int total = 0;
for (int num : numbers) {
total += num;
}
return total;
}
public static void main(String[] args) {
int num1 = 5;
int num2 = 10;
int num3 = 15;
int[] array = {3, 6, 9, 12};
// Calculate sums using overloaded methods
System.out.println("Sum of two numbers: " + sum(num1, num2));
System.out.println("Sum of three numbers: " + sum(num1, num2, num3));
System.out.println("Sum of array elements: " + sum(array));
}
}
```



```
F:\java>javac SumCalculator.java

F:\java>java SumCalculator
Sum of two numbers: 15
Sum of three numbers: 30
Sum of array elements: 30

F:\java>
```

Explanation:

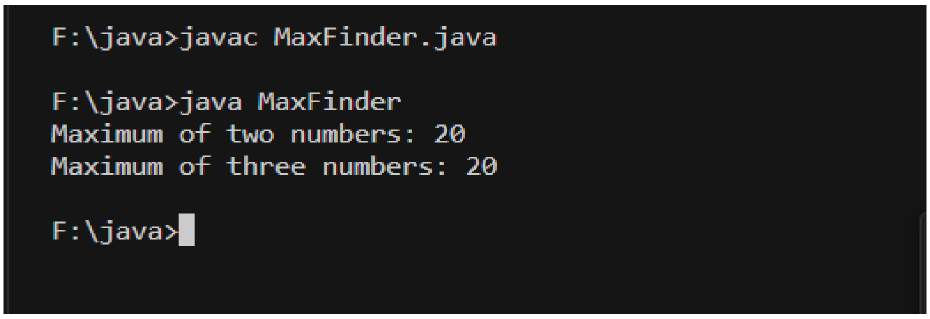
- In this program, we have a class **SumCalculator** with three overloaded methods named **sum**.
- The first method sums two integers.
- The second method sums three integers.
- The third method sums an array of integers.
- In the main method, we call each of the overloaded methods to calculate the sum of two numbers, three numbers, and an array of numbers.

— — — — —

TASK 4: **Finding Maximum**

```
// To find the Maximum using Overloaded Methods
public class MaxFinder {
// Method to find the maximum of two integers
public static int max(int a, int b) {
return (a > b) ? a : b;
}
// Method to find the maximum of three integers
public static int max(int a, int b, int c) {
return max(max(a, b), c); // Using recursion to find the maximum
}
public static void main(String[] args) {
int num1 = 15;
int num2 = 20;
int num3 = 10;
// Find maximum using overloaded methods
System.out.println("Maximum of two numbers: " + max(num1, num2));
System.out.println("Maximum of three numbers: " + max(num1, num2, num3));
}
}
```

Output:



```
F:\java>javac MaxFinder.java

F:\java>java MaxFinder
Maximum of two numbers: 20
Maximum of three numbers: 20

F:\java>
```

Explanation:

- In this program, we have a class **MaxFinder** with two overloaded methods named **max**.
- The first method finds the maximum of two integers.
- The second method finds the maximum of three integers by using recursion to call the first method.
- In the **main** method, we call each of the overloaded methods to find the maximum of two numbers and three numbers.

These examples demonstrate how overloaded methods in Java allow us to define methods with the same name but different parameter lists, providing flexibility and code reuse in our programs. Overloaded methods simplify method naming and enhance code readability.

EXERCISE 99 : Override methods in JAVA

Objectives

At the end of this exercise you shall be able to

- know override methods in JAVA
- develop Java programs using override methods.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Method overriding in Java allows a subclass to provide a specific implementation of a method that is already provided by its superclass. Here's an example demonstrating method overriding along with its explanation:

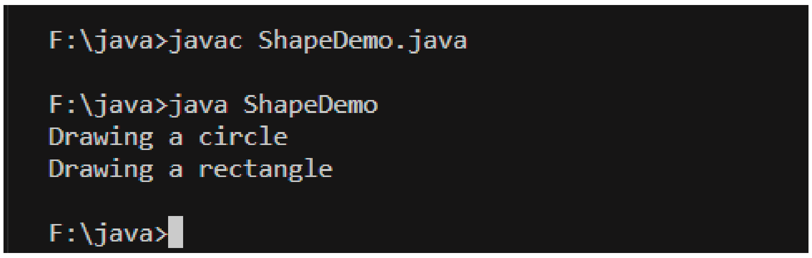
Procedure

TASK 1: **Shape and its Subclasses**

```
// Shape and its Subclasses using override method
class Shape {
void draw() {
System.out.println("Drawing a shape");
}
}
class Circle extends Shape {
// Overriding the draw method of the superclass
@Override
void draw() {
System.out.println("Drawing a circle");
}
}
class Rectangle extends Shape {
// Overriding the draw method of the superclass
@Override
void draw() {
System.out.println("Drawing a rectangle");
}
}
```

```
public class ShapeDemo {
public static void main(String[] args) {
Shape shape1 = new Circle();
shape1.draw(); // Output: Drawing a circle
Shape shape2 = new Rectangle();
shape2.draw(); // Output: Drawing a rectangle
}
}
```

Output:



```
F:\java>javac ShapeDemo.java

F:\java>java ShapeDemo
Drawing a circle
Drawing a rectangle

F:\java>
```

Explanation:

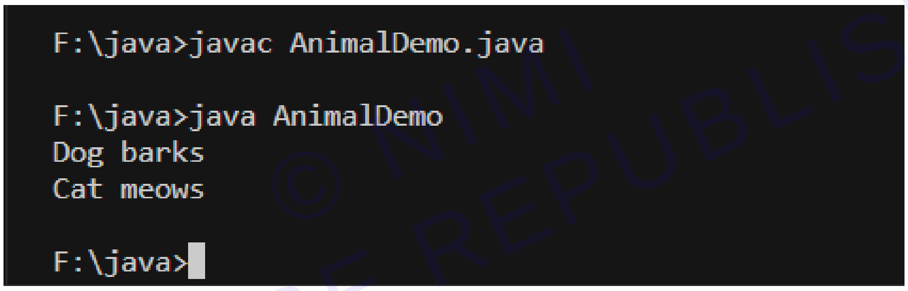
- In this program, we have a superclass **Shape** and two subclasses **Circle** and **Rectangle**.
- The **Shape** class has a method named **draw()** that prints "Drawing a shape".
- Both **Circle** and **Rectangle** classes extend the **Shape** class and override the **draw()** method with their own specific implementations.
- In the **ShapeDemo** class, we create instances of **Circle** and **Rectangle** and assign them to references of type **Shape**.
- When we call the **draw()** method on each object, the overridden version of the method is invoked based on the actual object type, demonstrating dynamic dispatch.
- Method overriding allows us to provide specialized implementations of methods in subclasses, promoting code reusability and polymorphic behavior.

TASK 2: **Animal and its Subclasses**

```
// Method overriding
class Animal {
void sound() {
System.out.println("Animal makes a sound");
}
}
class Dog extends Animal {
// Overriding the sound method of the superclass
@Override
void sound() {
System.out.println("Dog barks");
}
}
```

```
class Cat extends Animal {
// Overriding the sound method of the superclass
@Override
void sound() {
System.out.println("Cat meows");
}
}
public class AnimalDemo {
public static void main(String[] args) {
Animal animal1 = new Dog();
animal1.sound(); // Output: Dog barks
Animal animal2 = new Cat();
animal2.sound(); // Output: Cat meows
}
}
```

Output:



```
F:\java>javac AnimalDemo.java

F:\java>java AnimalDemo
Dog barks
Cat meows

F:\java>
```

Explanation:

- In this program, we have a superclass **Animal** and two subclasses **Dog** and **Cat**.
- The **Animal** class has a method named **sound()** that prints "Animal makes a sound".
- Both **Dog** and **Cat** classes extend the Animal class and override the **sound()** method with their own specific sound implementations.
- In the **AnimalDemo** class, we create instances of Dog and Cat and assign them to references of type Animal.
- When we call the sound() method on each object, the overridden version of the method is invoked based on the actual object type, demonstrating polymorphism and method overriding.

  These examples illustrate how method overriding allows subclasses to provide specialized implementations of methods inherited from their superclass, enabling polymorphic behavior and code flexibility in Java.

— — — — —

TASK 3: **Vehicle and its Subclasses**

```
// Method Overriding
class Vehicle {
void accelerate() {
System.out.println("Vehicle is accelerating");
}
}
```

```
class Car extends Vehicle {
// Overriding the accelerate method of the superclass
@Override
void accelerate() {
System.out.println("Car is accelerating");
}
}
class Truck extends Vehicle {
// Overriding the accelerate method of the superclass
@Override
void accelerate() {
System.out.println("Truck is accelerating");
}
}
public class VehicleDemo {
public static void main(String[] args) {
Vehicle vehicle1 = new Car();
vehicle1.accelerate(); // Output: Car is accelerating
Vehicle vehicle2 = new Truck();
vehicle2.accelerate(); // Output: Truck is accelerating
}
}
```

Explanation:

- In this program, we have a superclass **Vehicle** and two subclasses **Car** and **Truck**.
- The **Vehicle** class has a method named **accelerate()** that prints "Vehicle is accelerating".
- Both **Car** and **Truck** classes extend the **Vehicle** class and override the **accelerate()** method with their own specific implementations.
- In the **VehicleDemo** class, we create instances of **Car** and **Truck** and assign them to references of type **Vehicle**.
- When we call the **accelerate()** method on each object, the overridden version of the method is invoked based on the actual object type, demonstrating polymorphism and method overriding.

— — — — —

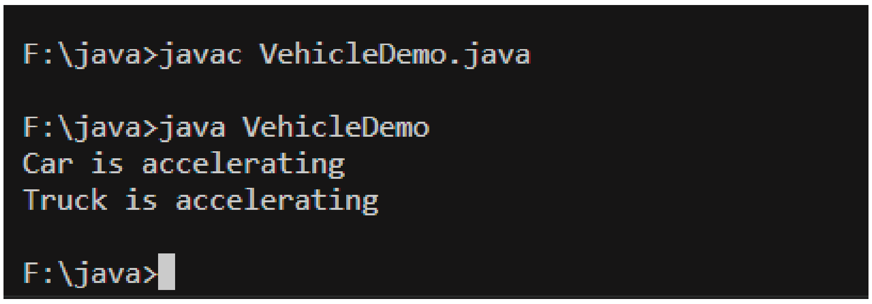
TASK 4: **Bank Account and its Subclasses**

```
// Bank Account and its Subclasses using method overriding
class BankAccount {
double balance;
void deposit(double amount) {
balance += amount;
}
void withdraw(double amount) {
```

```
balance -= amount;
}
void displayBalance() {
System.out.println("Balance: " + balance);
}
}
class SavingsAccount extends BankAccount {
// Overriding the withdraw method of the superclass
@Override
void withdraw(double amount) {
if (balance - amount >= 1000) {
balance -= amount;
} else {
System.out.println("Insufficient balance");
}
}
}
class CurrentAccount extends BankAccount {
// Overriding the withdraw method of the superclass
@Override
void withdraw(double amount) {
if (balance - amount >= 0) {
balance -= amount;
} else {
System.out.println("Insufficient balance");
}
}
}
public class BankAccountDemo {
public static void main(String[] args) {
BankAccount account1 = new SavingsAccount();
account1.deposit(5000);
account1.withdraw(3000);
account1.displayBalance(); // Output: Balance: 2000
BankAccount account2 = new CurrentAccount();
account2.deposit(3000);
account2.withdraw(5000);
account2.displayBalance(); // Output: Insufficient balance
```

```
}
}
```

Output:



```
F:\java>javac VehicleDemo.java

F:\java>java VehicleDemo
Car is accelerating
Truck is accelerating

F:\java>
```

Explanation:

- In this program, we have a superclass **BankAccount** and two subclasses **SavingsAccount** and **CurrentAccount**.
- The **BankAccount** class has methods for depositing, withdrawing, and displaying balance.
- Both **SavingsAccount** and **CurrentAccount** classes extend the **BankAccount** class and override the withdraw() method with their own specific implementations to handle withdrawal rules.
- In the **BankAccountDemo** class, we create instances of **SavingsAccount** and **CurrentAccount**.
- When we call the **withdraw()** method on each object, the overridden version of the method is invoked based on the actual object type, demonstrating polymorphism and method overriding.

  These examples illustrate how method overriding enables subclasses to provide their own specific implementations of methods inherited from their superclass, allowing for code reuse and flexibility in Java programs.

EXERCISE 100 : Create and use Super class, Sub class in JAVA

Objectives

At the end of this exercise you shall be able to

- know the use of Super class and Sub class in JAVA
- develop Java programs using Super class and Sub class.

Requirements

.Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

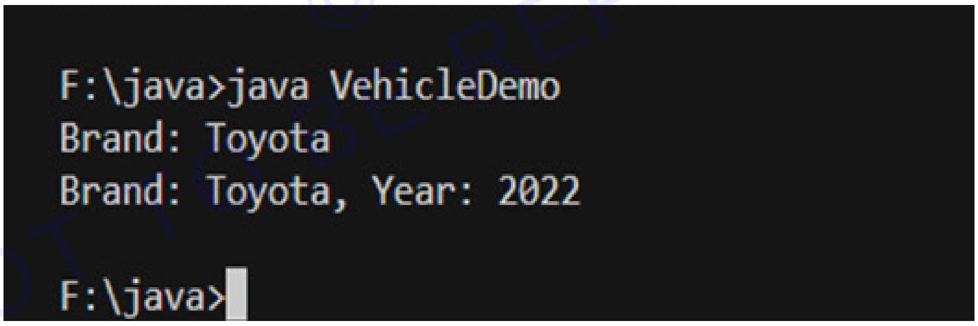
Procedure

- In object-oriented programming, the concept of inheritance allows a subclass to inherit properties and behaviors from a superclass. Here's a simple Java program demonstrating the usage of a superclass and its subclass along with explanations

TASK 1. **Superclass and Subclass: Vehicle and Car Classes**

```
// Superclass
class Vehicle {
    String brand;

    // Constructor
    Vehicle(String brand) {
        this.brand = brand;
    }

    void displayBrand() {
        System.out.println("Brand: " + brand);
    }
}

// Subclass
class Car extends Vehicle {
    int year;

    // Constructor
    Car(String brand, int year) {
        super(brand); // Call superclass constructor
        this.year = year;
    }

    void displayDetails() {
        System.out.println("Brand: " + brand + ", Year: " + year);
    }
}

// Main class
public class VehicleDemo {
    public static void main(String[] args) {
        // Creating an instance of the Car class
        Car car = new Car("Toyota", 2022);

        // Calling methods from both Vehicle and Car classes
        car.displayBrand(); // Output: Brand: Toyota
        car.displayDetails(); // Output: Brand: Toyota, Year: 2022
    }
}
```

Output:



```
F:\java>java VehicleDemo
Brand: Toyota
Brand: Toyota, Year: 2022

F:\java>
```

Explanation:

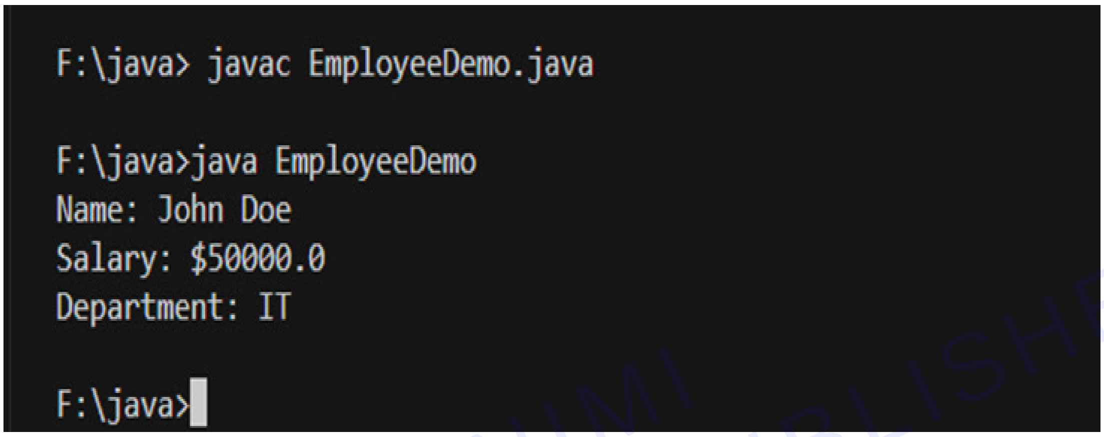
- In this program, we have a superclass Vehicle and a subclass Car.
- The Vehicle class has a brand attribute and a displayBrand() method to display the brand.
- The Car class extends the Vehicle class and adds its own attribute year and a displayDetails() method to display the brand and year.
- In the Vehicle constructor, super(brand) is used to call the superclass constructor and initialize the brand attribute.
- In the main method, we create an instance of the Car class with the brand "Toyota" and year 2022.
- We call methods from both the Vehicle and Car classes to demonstrate inheritance and method overriding.
- The superclass constructor is invoked using the super keyword within the subclass constructor to initialize inherited attributes.

This program illustrates how superclasses and subclasses are used in Java to achieve code reuse and inheritance. The subclass inherits attributes and methods from its superclass and can also provide its own unique attributes and behaviors. This hierarchical relationship promotes code organization and reusability.

TASK 2: **Employee and Manager Classes:**

```
    // Superclass
  class Employee {
    String name;
    double salary;

    // Constructor
    Employee(String name, double salary) {
      this.name = name;
      this.salary = salary;
    }

    void displayDetails() {
      System.out.println("Name: " + name);
      System.out.println("Salary: $" + salary);
    }
  }

  // Subclass
  class Manager extends Employee {
    String department;

    // Constructor
    Manager(String name, double salary, String department) {
      super(name, salary); // Call superclass constructor
      this.department = department;
    }

    @Override
    void displayDetails() {
      super.displayDetails(); // Call superclass method
      System.out.println("Department: " + department);
  }
}

// Main class
public class EmployeeDemo {
  public static void main(String[] args) {
        // Create an instance of the Manager class
        Manager manager = new Manager("John Doe", 50000, "IT");

        // Call the displayDetails method of the Manager class
        manager.displayDetails();
    }
}
```

Output:



```
F:\java> javac EmployeeDemo.java

F:\java>java EmployeeDemo
Name: John Doe
Salary: $50000.0
Department: IT

F:\java>
```

Explanation:

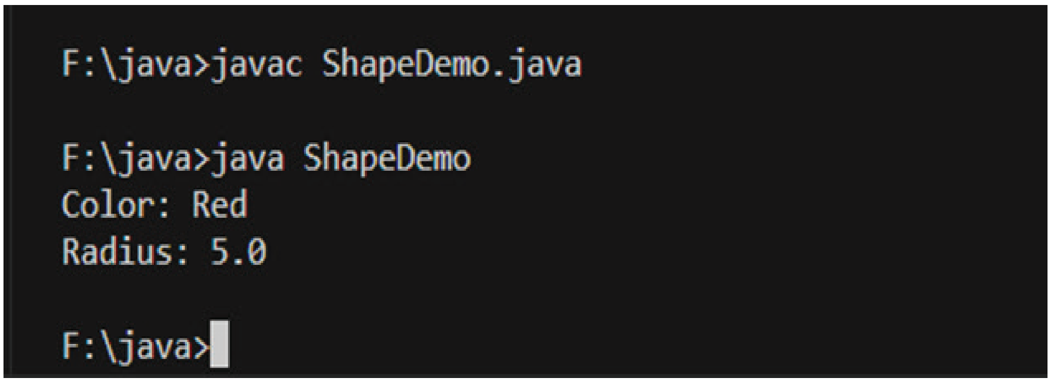
- In this program, we have a superclass **Employee** and a subclass **Manager.**
- The **Employee** class has attributes **name** and **salary,** and a method **displayDetails()** to display employee details.
- The **Manager** class extends the **Employee** class and adds its own attribute **department**.
- The **Manager** class overrides the **displayDetails()** method to display manager-specific details along with employee details.
- In the **Manager** constructor, **super(name, salary)** is used to call the superclass constructor and initialize inherited attributes.
- In the **main** method, we create an instance of the **Manager** class with a name, salary, and department, and call the **displayDetails()** method.

— — — — —

TASK 3**: Shape and its Subclasses**

```
// Superclass
class Shape {
    String color;
    // Constructor
    Shape(String color) {
        this.color = color;
    }
    void displayColor() {
        System.out.println("Color: " + color);
    }
}
// Subclass
class Circle extends Shape {
    double radius;
    // Constructor
    Circle(String color, double radius) {
        super(color); // Call superclass constructor
        this.radius = radius;
    }
    void displayRadius() {
        System.out.println("Radius: " + radius);
    }
}

// Main class
public class ShapeDemo {
    public static void main(String[] args) {
        // Create an instance of the Circle class
        Circle circle = new Circle("Red", 5.0);

        // Call the displayColor and displayRadius methods
        circle.displayColor();
        circle.displayRadius();
    }
}
```

Output:



```
F:\java>javac ShapeDemo.java

F:\java>java ShapeDemo
Color: Red
Radius: 5.0

F:\java>
```

Explanation:

- In this program, we have a superclass **Shape** and a subclass **Circle**.
- The **Shape** class has an attribute **color**, and a method **displayColor()** to display the color.
- The **Circle** class extends the **Shape** class and adds its own attribute **radius** and a method **displayRadius()** to display the radius.
- In the **Circle** constructor, **super(color)** is used to call the superclass constructor and initialize the inherited attribute.
- In the **main** method, we create an instance of the **Circle** class with a color and radius, and call the **displayColor()** and **displayRadius()** methods.

These examples demonstrate how superclasses and subclasses are used in Java to create hierarchical relationships, promote code reusability, and achieve inheritance. The subclass inherits attributes and methods from its superclass and can also add its own unique attributes and behaviors. This allows for code organization and simplifies the maintenance and extension of software systems.

Demonstrate writing JAVA programs to :

EXERCISE 101 : Create and run a thread

Objectives

At the end of this exercise you shall be able to

- know the creation and execution of threads in Java
- develop Java programs using Super class and Sub class.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

In Java, a thread represents a single flow of execution within a program. Threads allow programs to perform multiple tasks concurrently, making it possible to execute multiple pieces of code simultaneously.

TASK 1: **Creating and activating a thread**

```
public class MyThread1
{
// Main method
public static void main(String argvs[])
{
// creating an object of the Thread class using the constructor Thread(String name)
Thread t= new Thread("My first thread");

// the start() method moves the thread to the active state
t.start();
// getting the thread name by invoking the getName() method
String str = t.getName();
System.out.println(str);
}
}
```

Explanation:

Step 1: Develop the above code

- This code defines a class named MyThread1.
- The main method is the entry point of the program.

Step 2: Create a Thread Object

```
Thread t = new Thread("My first thread");
```

- A new instance of the Thread class is created.
- The constructor Thread(String name) is used to create the thread with the specified name ("My first thread" in this case).

Step 3: Start the Thread

```
t.start();
```

- The start() method is called on the thread object.
- This method initiates the execution of the thread and moves it to the active state.
- The thread will execute its run method in a separate thread of control.

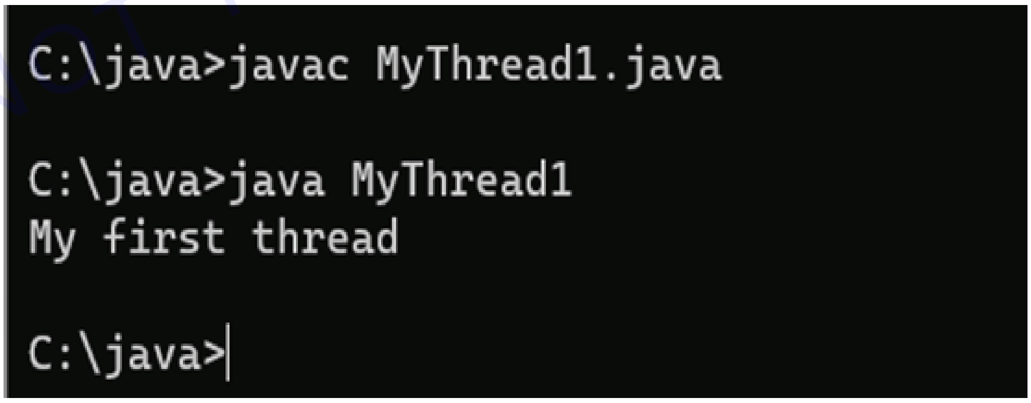
Step 4: Get and Print Thread Name

```
String str = t.getName();
System.out.println(str);
```

- The getName() method is called on the thread object (t) to retrieve its name.
- The retrieved name is stored in the variable str.
- The name is then printed to the console using System.out.println().

Step 5: Execution and Output

- When you run this program, you will see the output as the name of the thread:



```
C:\java>javac MyThread1.java

C:\java>java MyThread1
My first thread

C:\java>
```

This program demonstrates creating a thread, starting it, and retrieving its name using the Thread class in Java.

EXERCISE 102 : Create a thread by extending Thread class

Objectives

At the end of this exercise you shall be able to

- know the extension of threads in Java
- develop Java programs using thread extension.

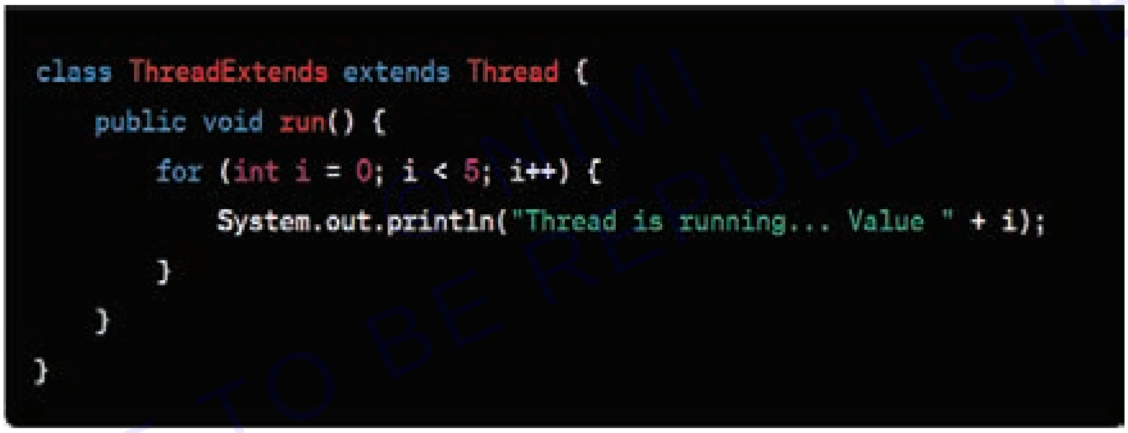
Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

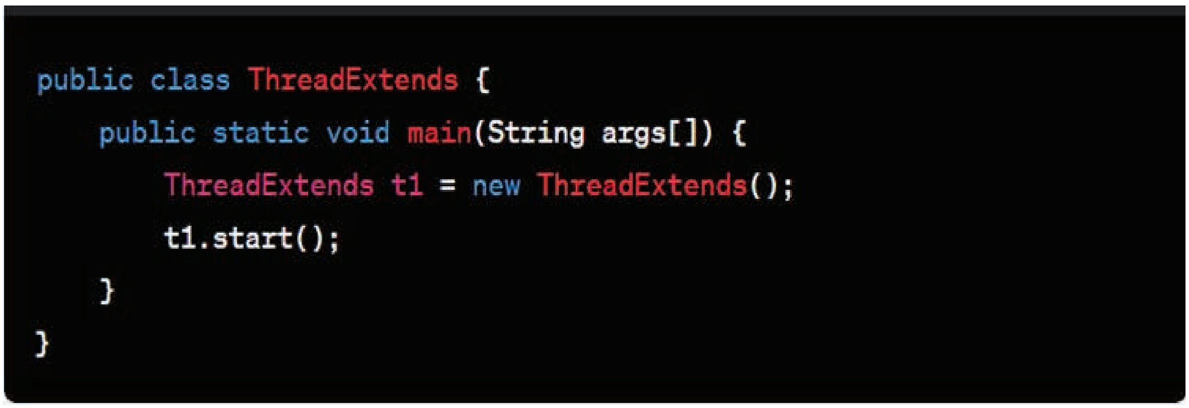
Step 1: Define a class extending Thread



```
class ThreadExtends extends Thread {
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("Thread is running... Value " + i);
        }
    }
}
```

- A new class named ThreadExtends is defined, and it extends the Thread class.
- The run method is overridden to define the behavior of the thread.
- Inside the run method, there is a for loop that prints a message with the current loop variable.

Step 2: Create an instance and Start the Thread

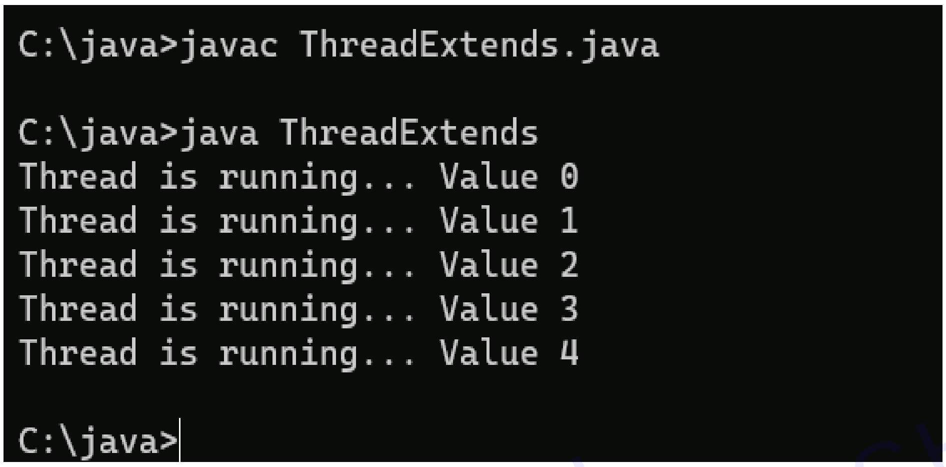


```
public class ThreadExtends {
    public static void main(String args[]) {
        ThreadExtends t1 = new ThreadExtends();
        t1.start();
    }
}
```

- A new class named ThreadExtends is defined (this class contains the main method).
- Inside the main method:
- An instance of ThreadExtends is created and assigned to the variable t1.
- The start method is called on t1. This initiates the execution of the thread, and the run method of ThreadExtends will be executed in a separate thread.

Step 3: Execution

When you run this program, it will output messages similar to the following:



```
C:\java>javac ThreadExtends.java

C:\java>java ThreadExtends
Thread is running... Value 0
Thread is running... Value 1
Thread is running... Value 2
Thread is running... Value 3
Thread is running... Value 4

C:\java>
```

EXERCISE 103 : Create thread by implementing Runnable interface

Objectives

At the end of this exercise you shall be able to

- know the creation and execution of threads in Java
- develop Java programs using Super class and Sub class.

Requirements

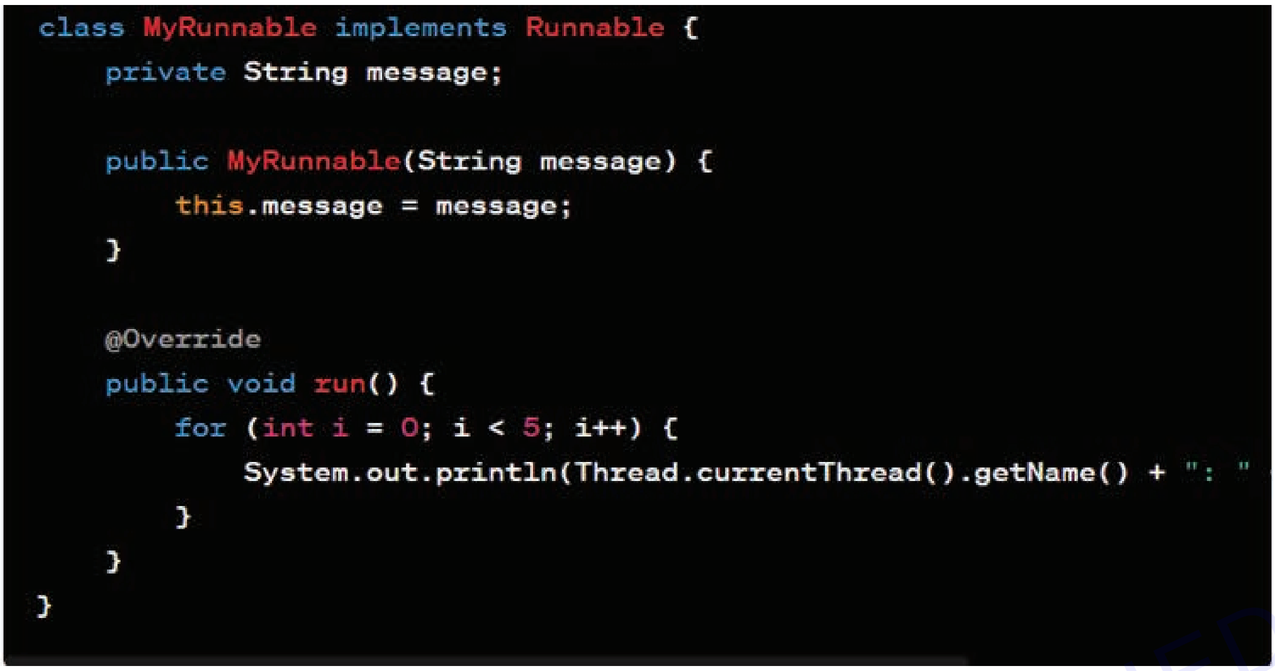
Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

```
class MyRunnable implements Runnable {
private String message;
public MyRunnable(String message) {
       this.message = message;
    }
    @Override
    public void run() {
   for (int i = 0; i < 5; i++) {
  System.out.println(Thread.currentThread().getName() + ": " + message + " " + i);
        }
    }
}
public class ThreadExample {
    public static void main(String args[]) {
        // Create instances of MyRunnable with different messages
        MyRunnable myRunnable1 = new MyRunnable("Thread 1");
        MyRunnable myRunnable2 = new MyRunnable("Thread 2");
        // Create Thread objects using MyRunnable instances
        Thread t1 = new Thread(myRunnable1, "Thread A");
        Thread t2 = new Thread(myRunnable2, "Thread B");
        // Start the threads
        t1.start();
        t2.start();
    }
}
```

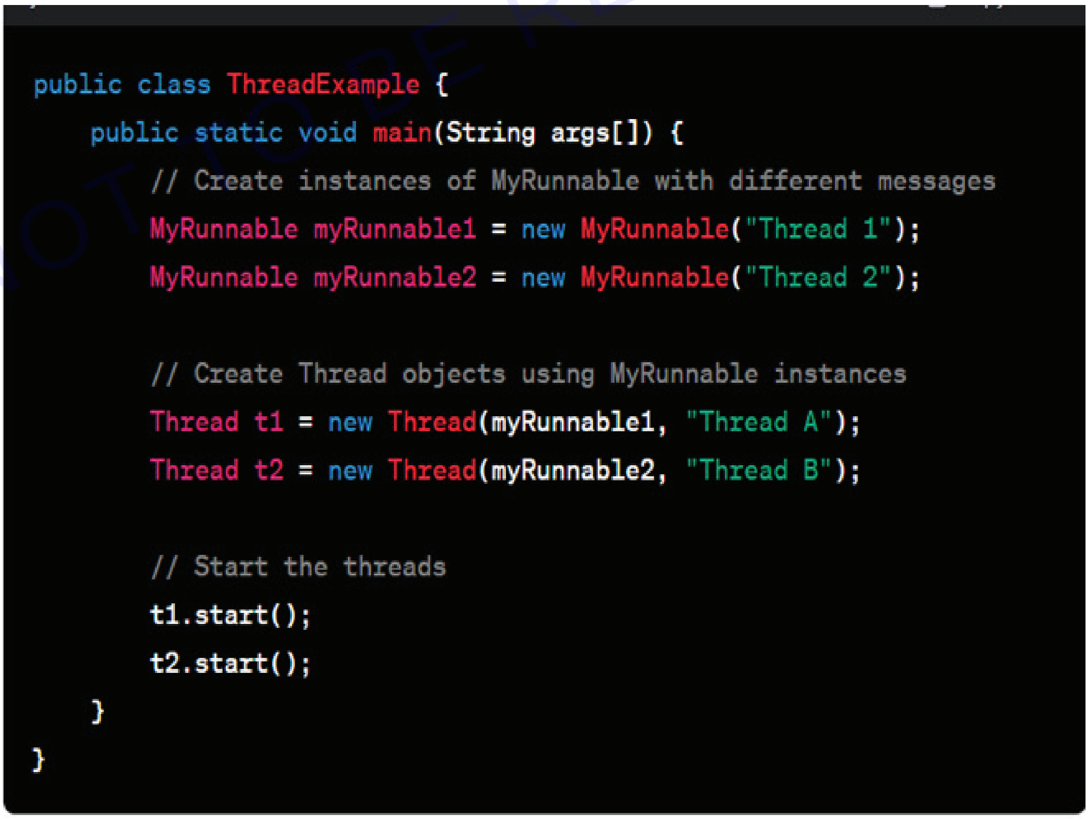
Step 1: Create a Thread class



```
class MyRunnable implements Runnable {
    private String message;

    public MyRunnable(String message) {
        this.message = message;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + ": "
        }
    }
}
```

- A new class named MyRunnable is defined, and it implements the Runnable interface.
- The class includes a private field message and a constructor to set its value.
- The run method is overridden from the Runnable interface. It contains a loop that prints the thread name, the message, and a loop variable.

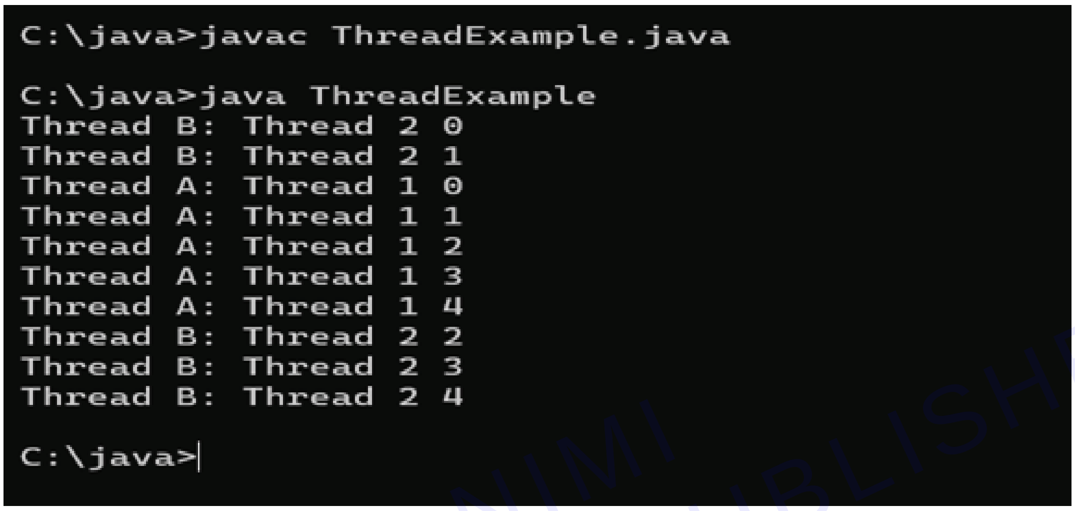
Step 2: Create the main class (ThreadExample)



```
public class ThreadExample {
    public static void main(String args[]) {
        // Create instances of MyRunnable with different messages
        MyRunnable myRunnable1 = new MyRunnable("Thread 1");
        MyRunnable myRunnable2 = new MyRunnable("Thread 2");

        // Create Thread objects using MyRunnable instances
        Thread t1 = new Thread(myRunnable1, "Thread A");
        Thread t2 = new Thread(myRunnable2, "Thread B");

        // Start the threads
        t1.start();
        t2.start();
    }
}
```

- A new class named ThreadExample is defined with a main method.
- Inside the main method:
- Two instances of MyRunnable are created with different messages.
- Two Thread instances (t1 and t2) are created, each initialized with a different MyRunnable instance and a thread name.
- The start method is called on each Thread instance, initiating the execution of the run method in a separate thread.

Step 3: Execution



```
C:\java>javac ThreadExample.java

C:\java>java ThreadExample
Thread B: Thread 2 0
Thread B: Thread 2 1
Thread A: Thread 1 0
Thread A: Thread 1 1
Thread A: Thread 1 2
Thread A: Thread 1 3
Thread A: Thread 1 4
Thread B: Thread 2 2
Thread B: Thread 2 3
Thread B: Thread 2 4

C:\java>
```

- When the program runs, two threads (t1 and t2) are created, and each executes its run method independently.
- The run method contains a loop that prints the thread name, the specified message, and a loop variable from 0 to 4.
- As a result, you'll see interleaved output from both threads, showing the concurrent execution.

EXERCISE 104 : Use major thread methods

Objectives

At the end of this exercise you shall be able to

- know the Major Thread Methods in Java
- develop Java programs using Thread Methods.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

1 sleep(long millis) Method:

- **Purpose:** Pauses the execution of the current thread for the specified number of milliseconds.
- **Syntax:Thread.sleep(long millis)**
- **Parameters:**

millis: The duration, in milliseconds, for which the thread should sleep.

- Use Case: Used to introduce delays or pauses in the execution of a thread.

```
public class SleepExample extends Thread {
    public void run() {
        System.out.println("Thread is running...");
        // Simulate some work
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getId() + " Value " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted");
                // Handle the exception or propagate it further if necessary
            }
        }
    }
    public static void main(String args[]) {
        SleepExample sleepThread = new SleepExample();
        sleepThread.start();
    }
}
```

Explanation:

1 **SleepExample Class:**

- This class extends the Thread class, indicating that instances of this class can be executed as separate threads.

2 **run() Method:**

- The run() method overrides the run() method of the Thread class, defining the behavior of the thread when it starts.
- Inside the run() method:
- It prints “Thread is running...” to indicate that the thread has started its execution.
- It enters a loop to simulate some work by printing the thread ID along with a sequence number (Value) from 0 to 4.
- After printing each value, the thread calls Thread.sleep(500) to pause its execution for 500 milliseconds (0.5 seconds).
- The sleep() method may throw an InterruptedException, so it’s enclosed within a try-catch block to handle this exception gracefully.

3 **main() Method:**

- This method serves as the entry point of the program.
- Inside main():
- An instance of the SleepExample class named sleepThread is created.
- The start() method is invoked on sleepThread to begin the execution of the thread.

4 **Output:**

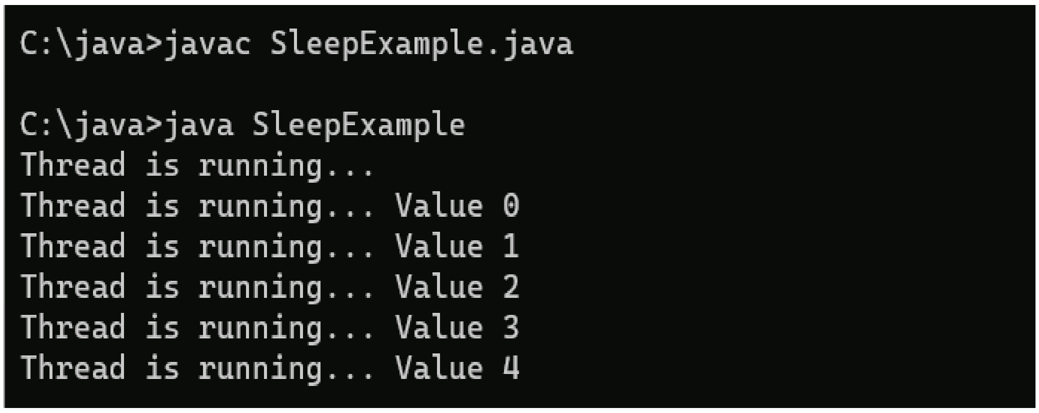
- When the program is executed:
- The thread starts its execution and prints “Thread is running...”.
- Then, in a loop, it prints the thread ID along with values from 0 to 4, with a 500-millisecond delay between each value.
- After printing all values, the thread completes its execution.

5 **Exception Handling:**

- The sleep() method can throw an InterruptedException if the thread is interrupted while sleeping.
- In the catch block, the program prints “Thread interrupted” to indicate that the thread’s sleep was interrupted.
- Depending on the application’s requirements, the exception can be handled or propagated further for handling at another level.

This program demonstrates how to use the sleep() method to introduce delays in the execution of threads, which is useful for scenarios where you need to control timing or simulate processing delays.

Output:



```
C:\java>javac SleepExample.java

C:\java>java SleepExample
Thread is running...
Thread is running... Value 0
Thread is running... Value 1
Thread is running... Value 2
Thread is running... Value 3
Thread is running... Value 4
```

2 **join() Method:**

- **Purpose:** Waits for the thread on which it's called to die.
- **Syntax:join()**
- **Use Case:**
- Ensures that the current thread waits for the completion of the thread on which join() is called before proceeding.

```
class JoinExample extends Thread {
public void run() {
System.out.println("Thread is running...");
}
public static void main(String args[]) throws InterruptedException {
JoinExample joinThread = new JoinExample();
joinThread.start();
joinThread.join();
System.out.println("Thread has finished");
}
}
```

Explanation:

1 **JoinExample Class:**

- This class extends the Thread class, indicating that instances of this class can be executed as separate threads.

2 **run() Method:**

- The run() method overrides the run() method of the Thread class, defining the behavior of the thread when it starts.
- Inside the run() method, it prints "Thread is running..." to indicate that the thread has started its execution.

3 **main() Method:**

- This method serves as the entry point of the program.
- Inside main():
- An instance of the JoinExample class named joinThread is created.
- The start() method is invoked on joinThread to begin the execution of the thread.
- The join() method is called on joinThread. This causes the main thread to wait until joinThread finishes its execution before continuing further.
- After joinThread finishes executing, "Thread has finished" is printed to indicate that the join operation has completed.

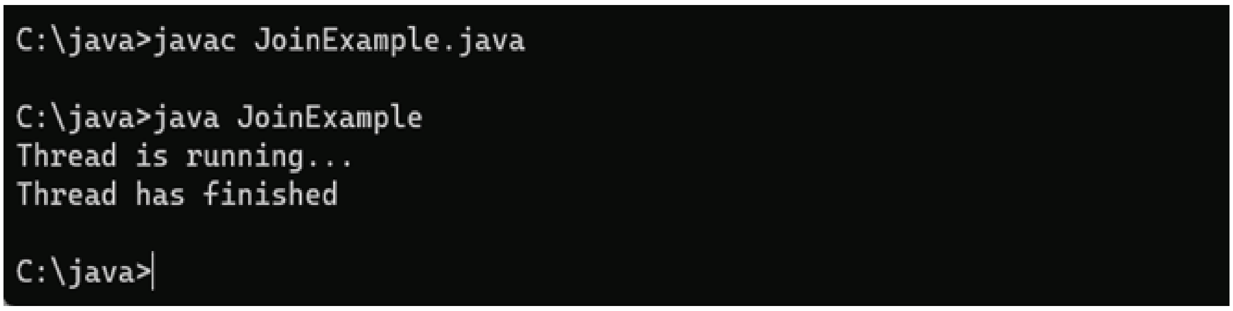
4 **Output:**

- When the program is executed:
- The thread starts its execution and prints "Thread is running...".
- Meanwhile, the main thread waits for joinThread to finish its execution using the join() method.
- Once joinThread finishes executing, "Thread has finished" is printed by the main thread.

5 **Exception Handling:**

- The join() method can throw an InterruptedException if the thread is interrupted while waiting for another thread to finish.
- In this example, the main() method declares that it throws InterruptedException. However, no explicit handling of the exception is performed within the method.

Output:



```
C:\java>javac JoinExample.java

C:\java>java JoinExample
Thread is running...
Thread has finished

C:\java>
```

3 **isAlive() Method:**

- **Purpose: Tests if a thread is alive.**
- **Syntax:isAlive()**
- **Return Type:boolean**
- **Use Case:**
- Allows you to check if a thread has been started and has not yet completed its execution.
- Useful when you want to perform actions in the main thread after ensuring that another thread has finished.

```
class IsAliveExample extends Thread {
    public void run() {
        System.out.println("Thread is running...");
    }
    public static void main(String args[]) throws InterruptedException {
        IsAliveExample isAliveThread = new IsAliveExample();
        isAliveThread.start();
        System.out.println("Is thread alive? " + isAliveThread.isAlive());
    }
}
```

Explanation:

1 **IsAliveExample Class:**

- This class extends the Thread class, indicating that instances of this class can be executed as separate threads.

2 **run() Method:**

- The run() method overrides the run() method of the Thread class, defining the behavior of the thread when it starts.
- Inside the run() method, it prints "Thread is running..." to indicate that the thread has started its execution.

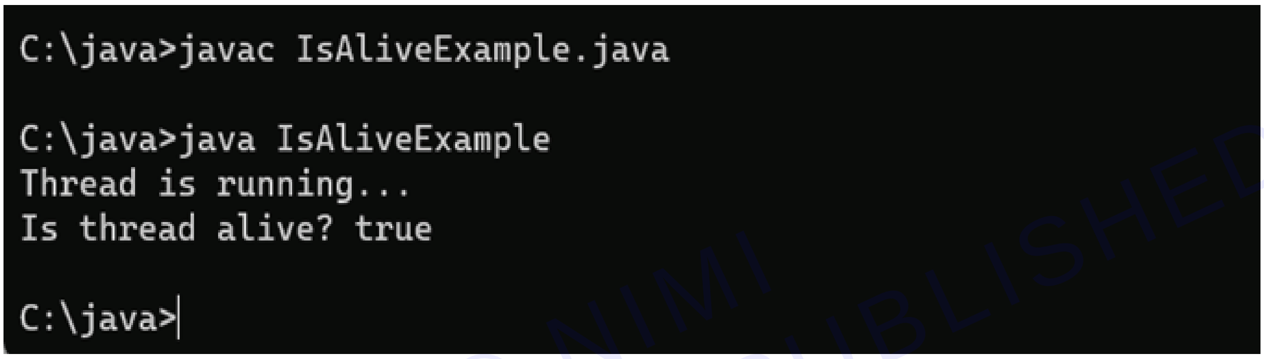
3. **main() Method:**

- This method serves as the entry point of the program.
- Inside main():
- An instance of the IsAliveExample class named isAliveThread is created.
- The start() method is invoked on isAliveThread to begin the execution of the thread.
- Immediately after starting the thread, isAlive() method is called on isAliveThread to check if the thread is alive.
- The result of isAlive() method (which returns true if the thread is alive and false otherwise) is printed.

4. **Output:**
   - When the program is executed:
   - The thread starts its execution and prints "Thread is running...".
   - Meanwhile, in the main thread, isAlive() method is called on isAliveThread to check if the thread is alive. Since the thread is just started, it is alive.
   - Therefore, "Is thread alive? true" is printed by the main thread.
5. **Exception Handling:**
   - No explicit exception handling is performed in this program.

This program demonstrates how to use the isAlive() method to determine if a thread is currently alive or has completed its execution. It's often used in scenarios where you need to check the status of threads in a multithreaded application.

Output:



```
C:\java>javac IsAliveExample.java

C:\java>java IsAliveExample
Thread is running...
Is thread alive? true

C:\java>
```

4 **setName(String name) Method:**
   - Purpose: Changes the name of the thread.
   - Syntax:setName(String name)
   - Parameters:
   - name: The new name to be assigned to the thread.
   - Use Case:
   - Provides a way to give a meaningful and recognizable name to threads for better identification.
   - Useful for debugging and logging purposes.

```
class SetNameExample extends Thread {
   public void run() {
      System.out.println("Thread is running...");
   }
   public static void main(String args[]) {
      SetNameExample setNameThread = new SetNameExample();
      setNameThread.setName("CustomThreadName");
      setNameThread.start();
      System.out.println("Thread Name: " + setNameThread.getName());
   }
}
```

Explanation:

1 **SetNameExample Class:**

- This class extends the Thread class, indicating that instances of this class can be executed as separate threads.

2 **run() Method:**

- The run() method overrides the run() method of the Thread class, defining the behavior of the thread when it starts.
- Inside the run() method, it prints “Thread is running...” to indicate that the thread has started its execution.

3 **main() Method:**

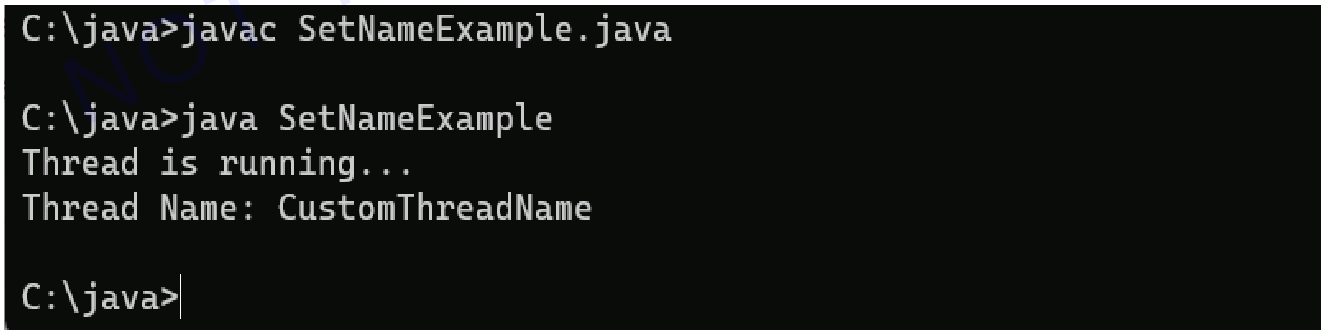
- This method serves as the entry point of the program.
- Inside main():
- An instance of the SetNameExample class named setNameThread is created.
- The setName() method is invoked on setNameThread to set the name of the thread to “CustomThreadName”.
- The start() method is invoked on setNameThread to begin the execution of the thread.
- Immediately after starting the thread, getName() method is called on setNameThread to retrieve the name of the thread.
- The name of the thread is then printed.

4 **Output:**

- When the program is executed:
- The thread starts its execution and prints “Thread is running...”.
- Meanwhile, in the main thread, the name of the thread (CustomThreadName) is retrieved using the getName() method and printed.
- No explicit exception handling is performed in this program.

This program demonstrates how to set and get the name of a thread in Java using the setName() and getName() methods. Naming threads can be helpful for identification and debugging purposes in multithreaded applications.

Output:



```
C:\java>javac SetNameExample.java

C:\java>java SetNameExample
Thread is running...
Thread Name: CustomThreadName

C:\java>
```

EXERCISE 105 : Test multithreading with and without synchronization

Objectives

At the end of this exercise you shall be able to

- know the multithreading with and without synchronization
- develop Java programs using multithreading with and without synchronization.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

```
//multithreading with and without synchronization
class CounterWithSync {
    private int count = 0;

    public synchronized void increment() {
        for (int i = 0; i < 5; i++) {
            int currentValue = count;
            System.out.println(Thread.currentThread().getName() + " - Before Increment: " + currentValue);
            count = currentValue + 1;
            System.out.println(Thread.currentThread().getName() + " - After Increment: " + count);
        }
    }
}
class IncrementThreadWithSync extends Thread {
    private CounterWithSync counter;
    public IncrementThreadWithSync(CounterWithSync counter) {
        this.counter = counter;
    }
    public void run() {
        counter.increment();
    }
}
public class MultithreadingWithSyncExample {
public static void main(String[] args) {
        CounterWithSync counter = new CounterWithSync();
        IncrementThreadWithSync thread1 = new IncrementThreadWithSync(counter);
        IncrementThreadWithSync thread2 = new IncrementThreadWithSync(counter);

        thread1.start();
        thread2.start();
    }
}
```

Explanation:

1 CounterWithSync Class:

- This class represents a counter with a private count variable.
- The increment() method is synchronized, which means only one thread can execute this method at a time.
- Inside the increment() method, a loop iterates five times, each time incrementing the count by one.

2 IncrementThreadWithSync Class:

- This class extends the Thread class and represents a thread that increments the counter.
- It has a reference to the CounterWithSync object.

3 MultithreadingWithSyncExample Class:

- This class contains the main() method, which serves as the entry point of the program.
- Inside main():
- An instance of CounterWithSync named counter is created.
- Two instances of IncrementThreadWithSync, thread1 and thread2, are created with a reference to the counter object.
- Both threads are started concurrently using the start() method.

4 Output:

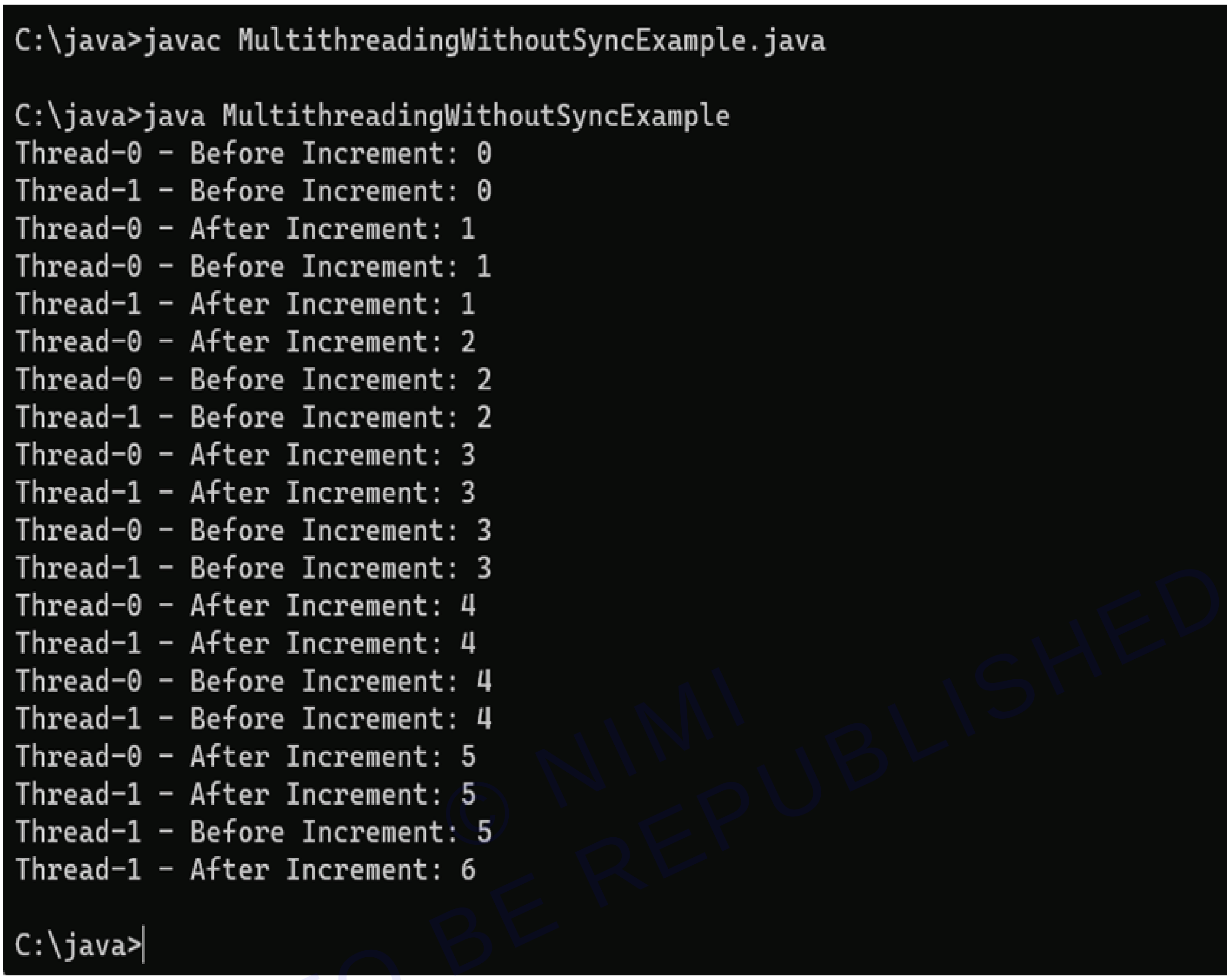
- Since the increment() method is synchronized, only one thread can execute it at a time.
- Thus, the output will show the interleaved execution of the two threads incrementing the counter.
- Each thread displays the count before and after incrementing it, ensuring that the counter is incremented in a thread-safe manner.

5 Thread Safety:

- Synchronization ensures that only one thread can execute the critical section of code (in this case, the increment() method) at a time.
- This prevents race conditions and ensures that the shared resource (count variable) is accessed in a thread-safe manner.

This program demonstrates how to use synchronization in Java to ensure thread safety when multiple threads access shared resources concurrently. It's essential for preventing data corruption and maintaining the integrity of shared data in multithreaded environments.

Output:



```
C:\java>javac MultithreadingWithoutSyncExample.java

C:\java>java MultithreadingWithoutSyncExample
Thread-0 - Before Increment: 0
Thread-1 - Before Increment: 0
Thread-0 - After Increment: 1
Thread-0 - Before Increment: 1
Thread-1 - After Increment: 1
Thread-0 - After Increment: 2
Thread-0 - Before Increment: 2
Thread-1 - Before Increment: 2
Thread-0 - After Increment: 3
Thread-1 - After Increment: 3
Thread-0 - Before Increment: 3
Thread-1 - Before Increment: 3
Thread-0 - After Increment: 4
Thread-1 - After Increment: 4
Thread-0 - Before Increment: 4
Thread-1 - Before Increment: 4
Thread-0 - After Increment: 5
Thread-1 - After Increment: 5
Thread-1 - Before Increment: 5
Thread-1 - After Increment: 6

C:\java>
```

EXERCISE 106 : Handle common exceptions

Objectives

At the end of this exercise you shall be able to

- develop Java programs using various exceptions.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

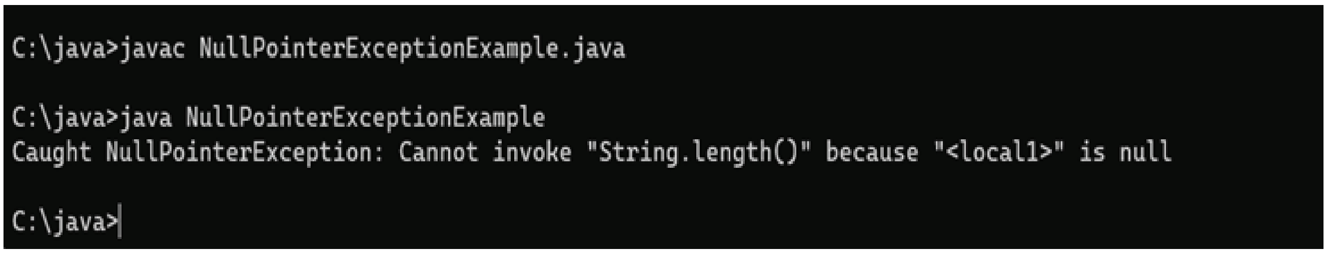
Procedure

1 NullPointerException:

- Description: Occurs when attempting to access members (methods or fields) of an object that is null.
- Handling Approach: Check if the object is null before accessing its members.

```
 public class NullPointerExceptionExample {
public static void main(String[] args) {
    try {
        // Step 1: Declare a String variable and initialize it to null
        String str = null;
        // Step 2: Attempt to access the length() method on a null reference
        int length = str.length(); // This will throw a NullPointerException
        // Step 3: Display the length of the string (this won't be reached due to the exception)
        System.out.println("Length of the string: " + length);
    } catch (NullPointerException e) {
        // Step 4: Catch the NullPointerException and handle it
        System.out.println("Caught NullPointerException: " + e.getMessage());
    }
  }
}
```

Output:



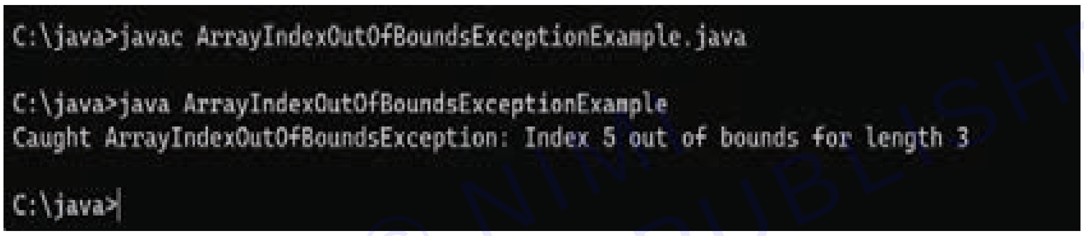
```
C:\java>javac NullPointerExceptionExample.java

C:\java>java NullPointerExceptionExample
Caught NullPointerException: Cannot invoke "String.length()" because "<local1>" is null

C:\java>
```

2 **ArrayIndexOutOfBoundsException:**

- Description: Occurs when attempting to access an array element with an index outside the array's bounds.
- Handling Approach: Ensure the index is within the array's bounds.

```
public class ArrayIndexOutOfBoundsExceptionExample {
    public static void main(String[] args) {
        try {
            int[] arr = {1, 2, 3};
            int element = arr[5]; // This will throw an ArrayIndexOutOfBoundsException
            System.out.println("Element at index 5: " + element);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());
        }
    }
}
```

Output:



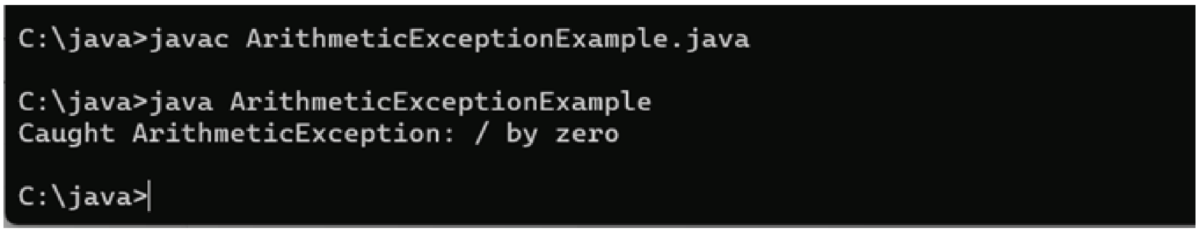
```
C:\java>javac ArrayIndexOutOfBoundsExceptionExample.java

C:\java>java ArrayIndexOutOfBoundsExceptionExample
Caught ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3

C:\java>
```

3 **ArithmeticException:**

- Description: Occurs when an arithmetic operation results in an undefined mathematical result (e.g., division by zero).
- Handling Approach: Check for conditions that might lead to undefined results.

```
public class ArithmeticExceptionExample {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // This will throw an ArithmeticException
            System.out.println("Result of division: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Caught ArithmeticException: " + e.getMessage());
        }
    }
}
```

Output:



```
C:\java>javac ArithmeticExceptionExample.java

C:\java>java ArithmeticExceptionExample
Caught ArithmeticException: / by zero

C:\java>
```

EXERCISE 107 : Use multiple try – catch blocks

Objectives

At the end of this exercise you shall be able to

- develop Java programs using multiple try – catch blocks.

Requirements

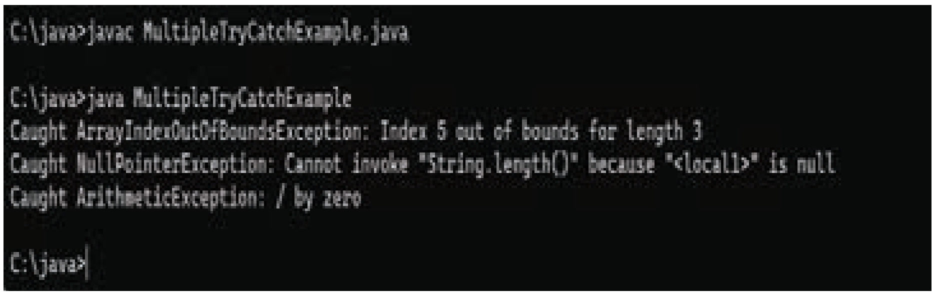
Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

```
// programs using  multiple try – catch blocks
public class MultipleTryCatchExample {
    public static void main(String[] args) {
        try {
            // First try block
            int[] numbers = {1, 2, 3};
          System.out.println(“Element at index 5: “ + numbers[5]); // This will throw ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(“Caught ArrayIndexOutOfBoundsException: “ + e.getMessage());
        }

        try {
            // Second try block
            String str = null;
            System.out.println(“Length of the string: “ + str.length()); // This will throw NullPointerException
        } catch (NullPointerException e) {
            System.out.println(“Caught NullPointerException: “ + e.getMessage());
        }

        try {
            // Third try block
            int result = 10 / 0; // This will throw ArithmeticException
            System.out.println(“Result of division: “ + result);
        } catch (ArithmeticException e) {
            System.out.println(“Caught ArithmeticException: “ + e.getMessage());
        }
    }
}
```

Output:



```
C:\java>javac MultipleTryCatchExample.java

C:\java>java MultipleTryCatchExample
Caught ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
Caught NullPointerException: Cannot invoke "String.length()" because "<local1>" is null
Caught ArithmeticException: / by zero

C:\java>
```

EXERCISE 108 : Use the “throw” and “finally” keywords handle user defined exceptions

Objectives

At the end of this exercise you shall be able to

- develop Java programs using “throw” and “finally” keywords handle user defined exceptions.

Requirements

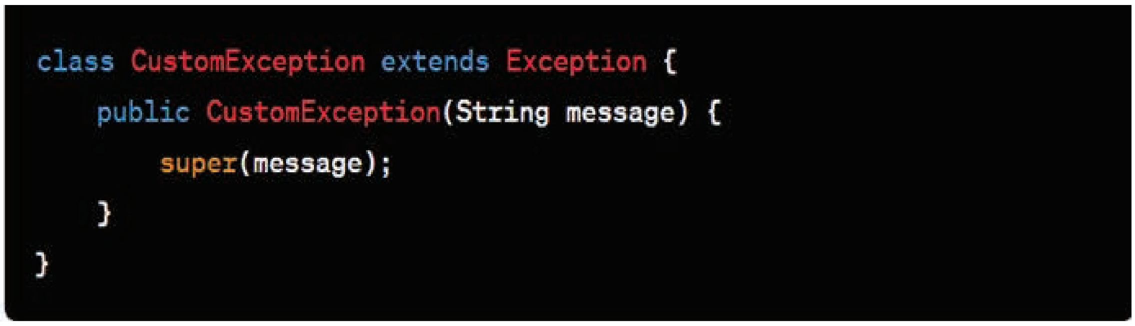
Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

```
// Custom exception class
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
public class ThrowFinallyExample {
    public static void main(String[] args) {
        try {
            // Simulating a condition where a custom exception is thrown
            throwCustomException();
        } catch (CustomException e) {
            System.out.println(“Caught CustomException: “ + e.getMessage());
        } finally {
            // Code in the finally block will always be executed
            System.out.println(“Finally block: This will execute whether an exception is thrown or not.”);
        }
    }

    // Method that throws a custom exception
    private static void throwCustomException() throws CustomException {
        // Using the ‘throw’ keyword to throw a custom exception
        throw new CustomException(“This is a custom exception.”);
    }
}
```

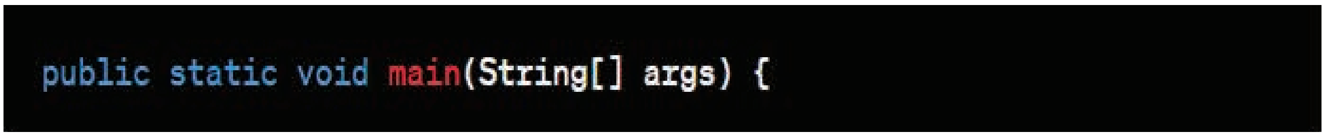
1 Custom Exception Class:



```java
class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
```

Define a custom exception class CustomException that extends the Exception class.

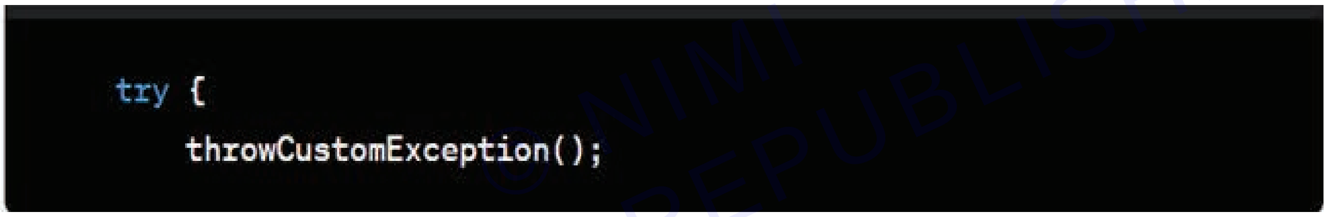
2 Main Method:



```java
public static void main(String[] args) {
```

- Declare the main method.

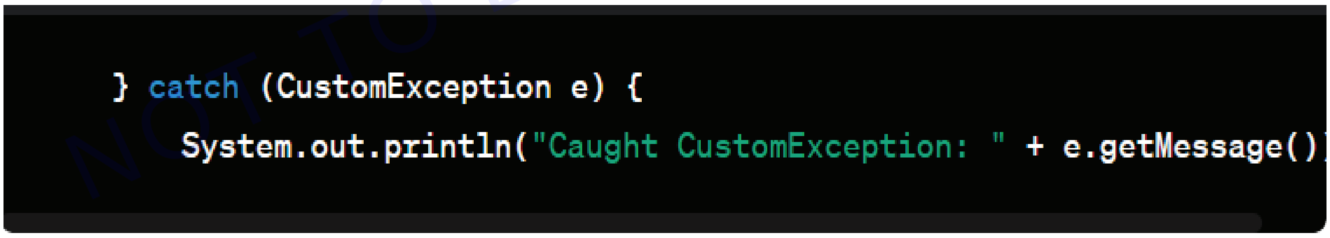
3. Try Block:



```java
    try {
        throwCustomException();
```

- Invoke the method throwCustomException() that throws a custom exception using the throw keyword.

4 Catch Block (CustomException):



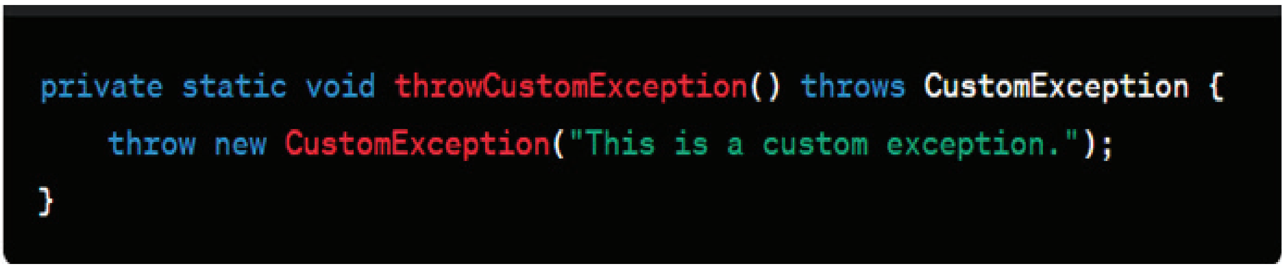
```java
    } catch (CustomException e) {
        System.out.println("Caught CustomException: " + e.getMessage()
```

- Catch the custom exception and print a message.

5 Finally Block:

- The finally block contains code that will always be executed, whether an exception is thrown or not.

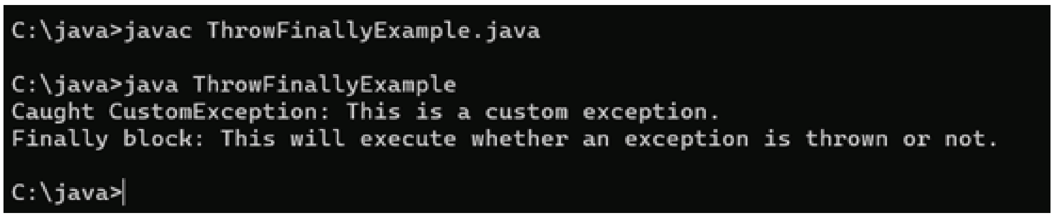
6 Throw Custom Exception Method:



```java
private static void throwCustomException() throws CustomException {
    throw new CustomException("This is a custom exception.");
}
```

- Define a method throwCustomException() that throws a custom exception.

When you run this program, it will output:



```
C:\java>javac ThrowFinallyExample.java

C:\java>java ThrowFinallyExample
Caught CustomException: This is a custom exception.
Finally block: This will execute whether an exception is thrown or not.

C:\java>
```

Related Exercises:

1 Write a program that prints "Good morning" and "Welcome" continuously on the screen in Java using threads.

2 Add a step method in the welcome thread of question 1 to delay its execution for 200ms.

3 Demonstrate gerPriority() and setPriority() methods in Java threads.

4 How do you get the state of a given thread in Java?

5 How do you get the reference to the current thread in Java?

6 Write a program that performs a file transfer using multiple threads.

7 Write a program that performs a database operation using multiple threads.

8 Write a program that handles the following exceptions:

- FileNotFoundException
- IOException
- NullPointerException
- ArithmeticException

9 Write a program that uses the finally block to ensure that resources are always closed, even if an exception occurs.

EXERCISE 109 : Create and use virtual methods

Objectives

At the end of this exercise you shall be able to

- develop Java programs to Create and use virtual methods.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

In Java, the term "virtual methods" is often associated with polymorphism, specifically dynamic method dispatch, which is a key feature of object-oriented programming (OOP). In Java, all non-static methods are inherently virtual.

Let's create a simple Java program that demonstrates the use of virtual methods and provide an explanation:

```
// Base class
class Shape {
    // Virtual method
    public void draw() {
        System.out.println("Drawing a generic shape");
    }
}

// Derived class 1
class Circle extends Shape {
    // Overrides the virtual method
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}

// Derived class 2
class Square extends Shape {
    // Overrides the virtual method
    @Override
    public void draw() {
        System.out.println("Drawing a square");
    }
}
// Main class
public class VirtualMethodExample {
    public static void main(String[] args) {
        // Create instances of the base class and derived classes
        Shape genericShape = new Shape();
        Circle myCircle = new Circle();
        Square mySquare = new Square();
        // Demonstrate virtual method calls
        genericShape.draw();  // Calls Shape's draw method
        myCircle.draw();      // Calls Circle's overridden draw method
        mySquare.draw();      // Calls Square's overridden draw method
        // Demonstrate polymorphism
        Shape polymorphicShape;
        polymorphicShape = myCircle;  // Circle assigned to Shape reference
        polymorphicShape.draw();      // Calls Circle's overridden draw method
        polymorphicShape = mySquare;  // Square assigned to Shape reference
        polymorphicShape.draw();      // Calls Square's overridden draw method
    }
}
```

Explanation:

1 **Base Class (Shape):**

- Defines a virtual method draw that serves as a generic drawing method.

2 **Derived Classes (Circle and Square):**

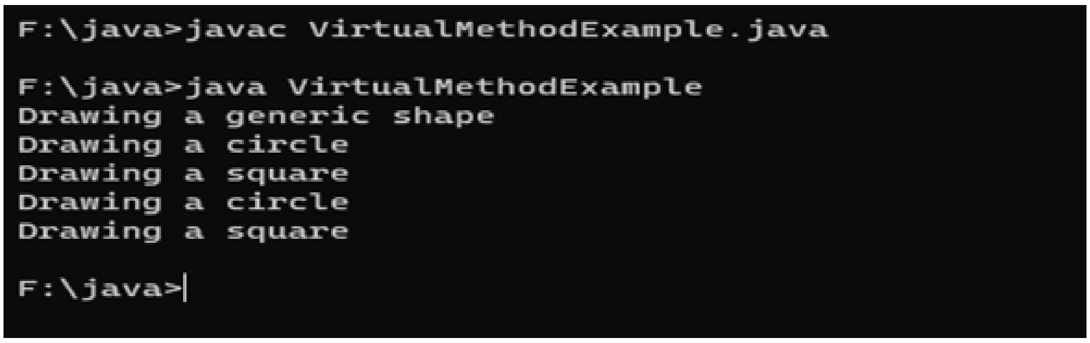
- Extend the Shape class.
- Override the virtual method draw with specific implementations for drawing a circle and a square.

3 **Main Class (VirtualMethodExample):**

- Creates instances of the base class and derived classes.
- Demonstrates virtual method calls for each instance.
- Illustrates polymorphism by assigning instances of derived classes to a Shape reference and invoking overridden methods.

In this example, the virtual method draw is overridden in the derived classes (Circle and Square). When an object is assigned to a reference of the base class (Shape), the appropriate overridden method is called at runtime based on the actual object type. This showcases the concept of virtual methods and polymorphism in Java.

Output:



```
F:\java>javac VirtualMethodExample.java

F:\java>java VirtualMethodExample
Drawing a generic shape
Drawing a circle
Drawing a square
Drawing a circle
Drawing a square

F:\java>
```

EXERCISE 110 : Create abstract classes and methods

Objectives

At the end of this exercise you shall be able to

- develop Java programs to Create abstract classes and methods.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

TASK 1: **Create abstract classes and methods example**

```
// Abstract class
abstract class Shape {
    // Abstract method
    public abstract double calculateArea();
    // Concrete method
    public void displayArea() {
        System.out.println("Area: " + calculateArea());
    }
}
// Concrete class 1
class Circle extends Shape {
    private double radius;
    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }
    // Implementation of abstract method
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}
// Concrete class 2
```

```
class Rectangle extends Shape {
    private double length;
    private double width;
    // Constructor
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    // Implementation of abstract method
    @Override
    public double calculateArea() {
        return length * width;
    }
}
// Main class
public class AbstractClassExample {
    public static void main(String[] args) {
        // Create instances of concrete classes
        Circle circle = new Circle(5.0);
        Rectangle rectangle = new Rectangle(4.0, 6.0);

        // Call abstract and concrete methods
        circle.displayArea();    // Calls abstract method implementation in Circle
        rectangle.displayArea(); // Calls abstract method implementation in Rectangle
    }
}
```

Explanation:

1 **Abstract Class (Shape):**

- Declares an abstract method calculateArea() without implementation.
- Defines a concrete method displayArea() that calls the abstract method.

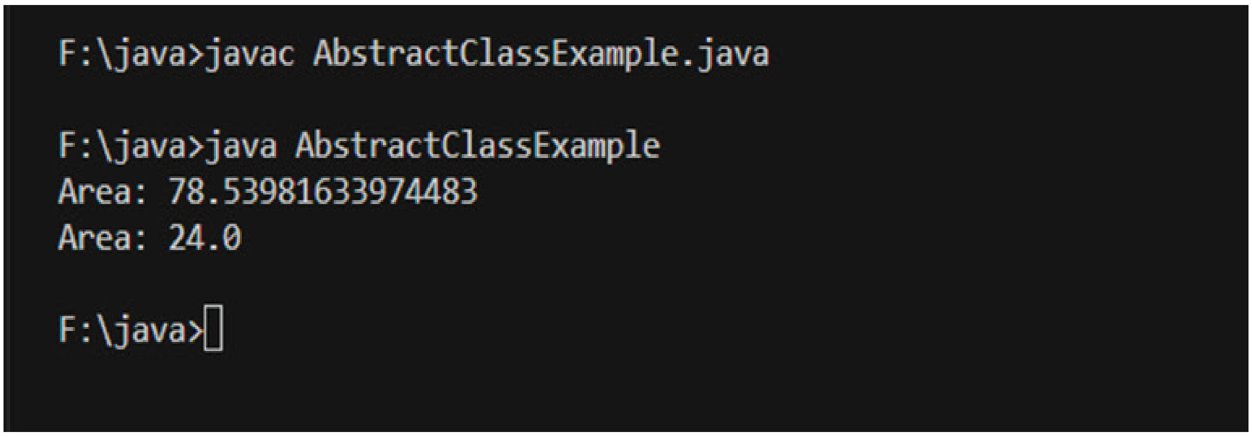
2 **Concrete Classes (Circle and Rectangle):**

- Extend the abstract class Shape.
- Provide implementations for the abstract method calculateArea().

3 **Main Class (AbstractClassExample):**

- Creates instances of concrete classes.
- Demonstrates calling abstract and concrete methods.

4 Output:



```
F:\java>javac AbstractClassExample.java

F:\java>java AbstractClassExample
Area: 78.53981633974483
Area: 24.0

F:\java>
```

In this example, Shape is an abstract class with an abstract method calculateArea(). Concrete classes (Circle and Rectangle) extend Shape and provide their implementations for the calculateArea() method. The AbstractClassExample class demonstrates creating instances of concrete classes and calling both abstract and concrete methods

— — — — —

TASK 2: **Abstract Class with Multiple Abstract Methods**

```
// Abstract class
abstract class Animal {
    // Abstract methods
    public abstract void makeSound();
    public abstract void eat();
    // Concrete method
    public void sleep() {
        System.out.println("Zzz... (Animal is sleeping)");
    }
}
// Concrete class 1
class Dog extends Animal {
    // Implementing abstract methods
    @Override
    public void makeSound() {
        System.out.println("Woof! Woof!");
    }
    @Override
    public void eat() {
        System.out.println("Dog is eating");
    }
}
```

```
// Concrete class 2
class Cat extends Animal {
    // Implementing abstract methods
    @Override
    public void makeSound() {
        System.out.println("Meow!");
    }

    @Override
    public void eat() {
        System.out.println("Cat is eating");
    }
}

// Main class
public class AnimalExample {
    public static void main(String[] args) {
        // Create instances of concrete classes
        Dog myDog = new Dog();
        Cat myCat = new Cat();

        // Call abstract and concrete methods
        myDog.makeSound();
        myDog.eat();
        myDog.sleep();

        myCat.makeSound();
        myCat.eat();
        myCat.sleep();
    }
}
```

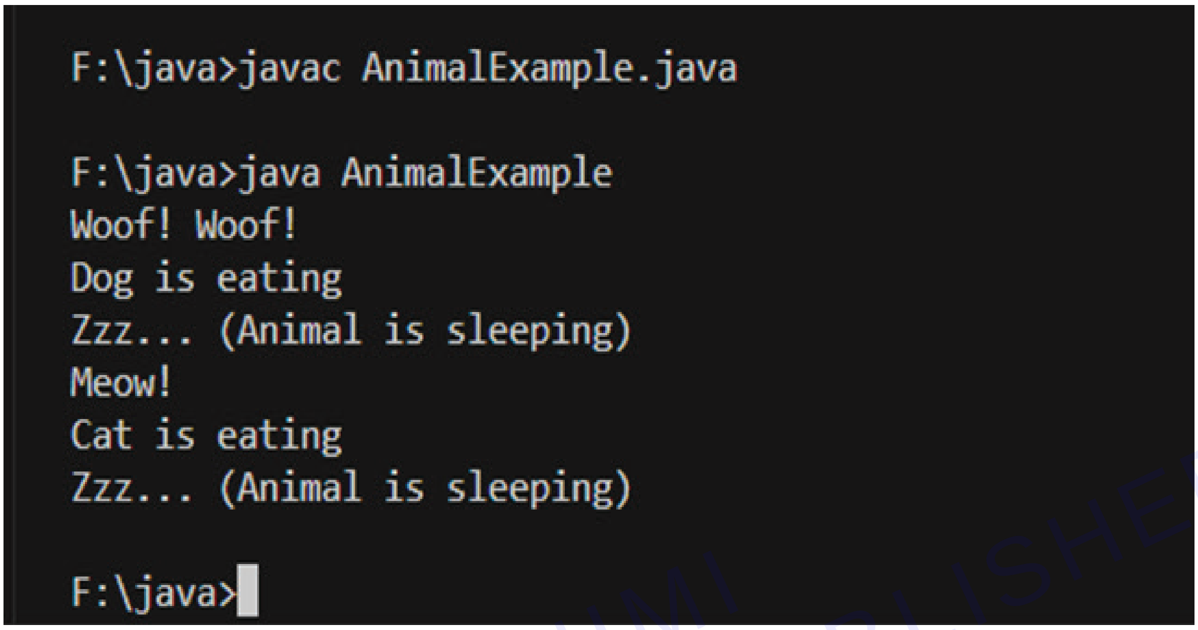
Explanation:

1. Abstract Class (Animal):
   - Declares two abstract methods makeSound() and eat().
   - Defines a concrete method sleep() with a default implementation.
2. Concrete Classes (Dog and Cat):
   - Extend the abstract class Animal.
   - Provide implementations for the abstract methods makeSound() and eat().

3. Main Class (AnimalExample):

- Creates instances of concrete classes (Dog and Cat).
- Calls both abstract and concrete methods.

4. Output:



```
F:\java>javac AnimalExample.java

F:\java>java AnimalExample
Woof! Woof!
Dog is eating
Zzz... (Animal is sleeping)
Meow!
Cat is eating
Zzz... (Animal is sleeping)

F:\java>
```

In this example, Animal is an abstract class representing common characteristics of animals. It declares two abstract methods (makeSound() and eat()) that must be implemented by its concrete subclasses. The sleep() method is a concrete method with a default implementation.

- **Concrete Class Dog:**
  - Implements the makeSound() and eat() methods specific to a dog.
  - Inherits the sleep() method from the Animal class.
- **Concrete Class Cat:**
  - Implements the makeSound() and eat() methods specific to a cat.
  - Inherits the sleep() method from the Animal class.
- **Main Class (AnimalExample):**
  - Creates instances of Dog and Cat.
  - Demonstrates calling methods: makeSound(), eat(), and sleep().

This example showcases the concept of abstraction where the common behavior is defined in the abstract class (Animal), and specific details are implemented in the concrete subclasses (Dog and Cat). The main class demonstrates polymorphism, as instances of different subclasses can be treated as instances of the common abstract class (Animal).

EXERCISE 111 : Create interfaces in JAVA

Objectives

At the end of this exercise you shall be able to

- develop Java programs to Create interface in Java.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

TASK 1: **Here's a basic example of using an interface in Java:**

```
// Define an interface named Printable
interface Printable {
    void print(); // Abstract method without implementation
}
// Implement the Printable interface in a class
class Printer implements Printable {
    // Provide implementation for the print method
    @Override
    public void print() {
        System.out.println("Printing a document...");
    }
}
// Main class to demonstrate the interface usage
public class InterfaceBasicExample {
    public static void main(String[] args) {
        // Create an instance of the Printer class
        Printer myPrinter = new Printer();

        // Call the print method through the Printable interface
        myPrinter.print();
    }
}
```

Explanation:

1 Interface (Printable):

- Declares a single abstract method named print().

2. Class Implementing the Interface (Printer):

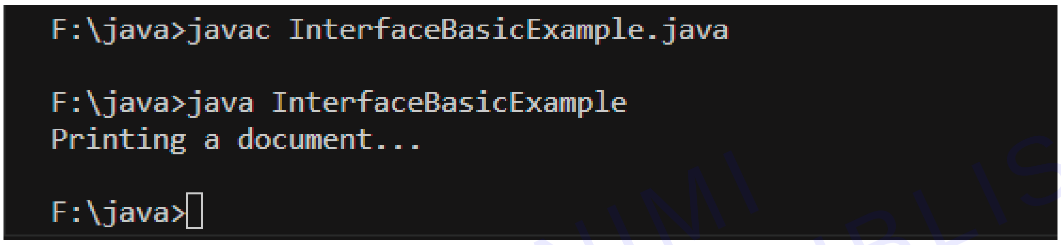
- Implements the Printable interface.
- Provides a concrete implementation for the print method.

3. Main Class (InterfaceBasicExample):

- Creates an instance of the Printer class.
- Calls the print method through the Printable interface.

This basic example demonstrates how an interface defines a contract (in this case, the print method), and a class implementing the interface must provide a concrete implementation for that method. The main class then utilizes the interface to call the implemented method. The use of interfaces allows for a level of abstraction and helps achieve better code organization and maintainability.

Output:



```
F:\java>javac InterfaceBasicExample.java

F:\java>java InterfaceBasicExample
Printing a document...

F:\java>
```

TASK 2 : **In Java, an interface is a collection of abstract methods. It provides a way to achieve abstraction and multiple inheritance. Here is an example of creating interfaces in Java**

```
// Example 1: Basic Interface
interface Printable {
    void print(); // Abstract method (no method body)
}

// Example 2: Interface with Constant
interface Shape {
    double PI = 3.14; // Constant (implicitly public, static, and final)

    double calculateArea(); // Abstract method
}

// Example 3: Interface with Default Method
interface Greeting {
    void greet(); // Abstract method

    default void farewell() {
        System.out.println("Goodbye!"); // Default method with implementation
        }
     }
// Example 4: Interface with Static Method
interface Utility {
    static void showInfo() {
    System.out.println("This is a utility interface."); // Static method with implementation
    }
    void performTask(); // Abstract method
}
// Example 5: Interface Inheritance
interface Flyable {
    void fly();
}
interface Swimmable {
    void swim();
}
// Combined interface inheriting from Flyable and Swimmable
interface FlyingSwimmingCreature extends Flyable, Swimmable {
    // No additional methods needed
}
// Main class
public class InterfaceExample implements Printable, Shape, Greeting, Utility, FlyingSwimmingCreature {
    // Implementation of Printable interface method
    @Override
    public void print() {
        System.out.println("Printing...");
    }
    // Implementation of Shape interface method
    @Override
    public double calculateArea() {
        return PI * 2 * 2; // Area of a circle with radius 2
    }
    // Implementation of Greeting interface method
    @Override
    public void greet() {
        System.out.println("Hello!");
    }
    // Implementation of Utility interface method
    @Override
    public void performTask() {
        System.out.println("Performing a task...");
    }
    // Implementation of Flyable interface method
    @Override
    public void fly() {
        System.out.println("Flying...");
    }
    // Implementation of Swimmable interface method
    @Override
    public void swim() {
        System.out.println("Swimming...");
    }
    // Main method
    public static void main(String[] args) {
        InterfaceExample example = new InterfaceExample();
        // Calling methods from implemented interfaces
        example.print();
        System.out.println("Area: " + example.calculateArea());
        example.greet();
        example.farewell(); // Calling default method
        example.performTask();
        example.fly();
        example.swim();
    }
}
```

Explanation:

1 Basic Interface (Printable):
   - Declares a single abstract method print().

2 Interface with Constant (Shape):
   - Declares a constant PI and an abstract method calculateArea().

3 Interface with Default Method (Greeting):
   - Declares an abstract method greet() and a default method farewell() with an implementation.

4 Interface with Static Method (Utility):
   - Declares an abstract method performTask() and a static method showInfo() with an implementation.

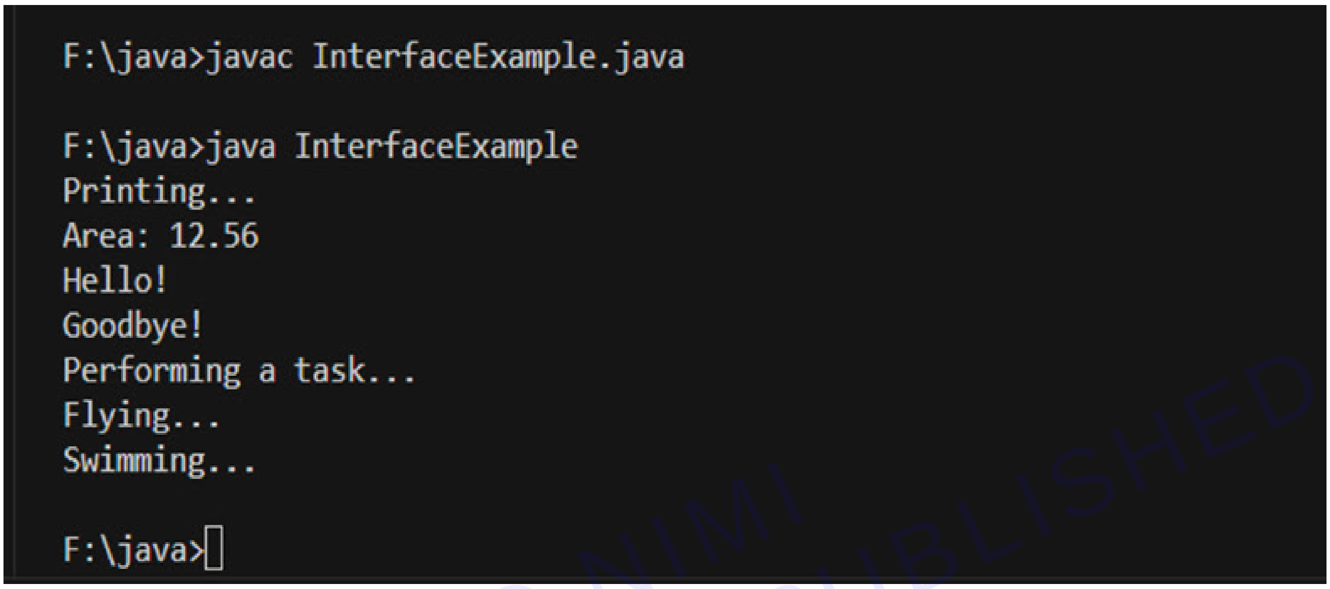
5 Interface Inheritance (Flyable and Swimmable):
   - Two interfaces with different methods.
   - A third interface (FlyingSwimmingCreature) inherits from both Flyable and Swimmable.

6. Main Class (InterfaceExample):

- Implements multiple interfaces (Printable, Shape, Greeting, Utility, FlyingSwimmingCreature).
- Provides implementations for all abstract methods from the interfaces.
- Demonstrates calling methods from the implemented interfaces.

This example illustrates the use of interfaces with abstract methods, constants, default methods, static methods, and interface inheritance in Java. Instances of the InterfaceExample class can be treated as instances of each implemented interface.

Output:



```
F:\java>javac InterfaceExample.java

F:\java>java InterfaceExample
Printing...
Area: 12.56
Hello!
Goodbye!
Performing a task...
Flying...
Swimming...

F:\java>
```

EXERCISE 112 : Override methods in JAVA

Objectives

At the end of this exercise you shall be able to
- develop Java programs to Override methods in JAVA.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

In Java, method overriding is a mechanism by which a subclass provides a specific implementation of a method that is already provided by its superclass. The overridden method in the subclass should have the same signature (name, return type, and parameters) as the method in the superclass.

TASK 1: **Here's a basic example of method overriding in Java:**

```
// Base class (Superclass)
class Animal {
   // Method to make a sound
   public void makeSound() {
      System.out.println("Some generic animal sound");
   }
}
// Derived class (Subclass) extending Animal
class Cat extends Animal {
   // Override the makeSound method from the Animal class
   @Override
   public void makeSound() {
      System.out.println("Meow!");
   }

   // Additional method specific to Cat
   public void purr() {
      System.out.println("Purring...");
   }
}
// Main class to demonstrate method overriding
public class MethodOverrideExample {
   public static void main(String[] args) {
```

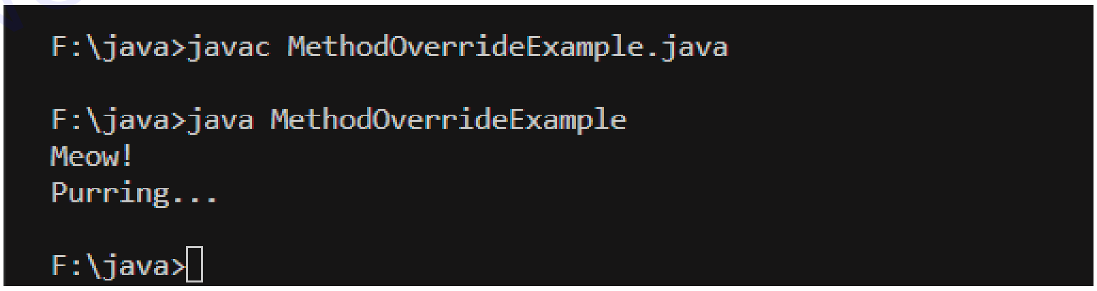
```
            // Create an instance of the Cat class
            Cat myCat = new Cat();

            // Call the overridden method
            myCat.makeSound();  // Outputs: Meow!

            // Call the additional method specific to Cat
            myCat.purr();  // Outputs: Purring
        }
    }
```

Explanation:

1. Base Class (Animal):
   - Contains a method named makeSound.
2. Derived Class (Cat):
   - Extends the Animal class.
   - Overrides the makeSound method to provide a specific implementation for the Cat class.
   - Adds an additional method purr specific to the Cat class.
3. Main Class (MethodOverrideExample):
   - Creates an instance of the Cat class.
   - Calls the overridden method makeSound, which outputs the specific sound for a cat.
   - Calls the additional method purr, which is specific to the Cat class.

In this example, the makeSound method is overridden in the Cat class to provide a more specific implementation for a cat's sound. Method overriding is a key feature in achieving polymorphism in object-oriented programming. Instances of the subclass can be treated as instances of the superclass, allowing for flexibility and extensibility in code.

Output:



```
F:\java>javac MethodOverrideExample.java

F:\java>java MethodOverrideExample
Meow!
Purring...

F:\java>
```

— — — — —

TASK 2 : **Here's a basic example of method overriding in Java**

```
// Base class (Superclass)
class Vehicle {
    // Method to display information about the vehicle
    public void displayInfo() {
        System.out.println("This is a generic vehicle.");
    }
}

// Derived class (Subclass) extending Vehicle
class Car extends Vehicle {
    // Override the displayInfo method from the Vehicle class
    @Override
    public void displayInfo() {
        System.out.println("This is a car.");
    }
    // Additional method specific to Car
    public void startEngine() {
        System.out.println("Car engine started.");
    }
}

// Main class to demonstrate method overriding
public class MethodOverrideBasicExample {
    public static void main(String[] args) {
        // Create an instance of the Car class
        Car myCar = new Car();

        // Call the overridden method
        myCar.displayInfo();  // Outputs: This is a car.

        // Call the additional method specific to Car
        myCar.startEngine();  // Outputs: Car engine started.
    }
}
```

Explanation:

1 Base Class (Vehicle):
   - Contains a method named displayInfo to display generic information about the vehicle.

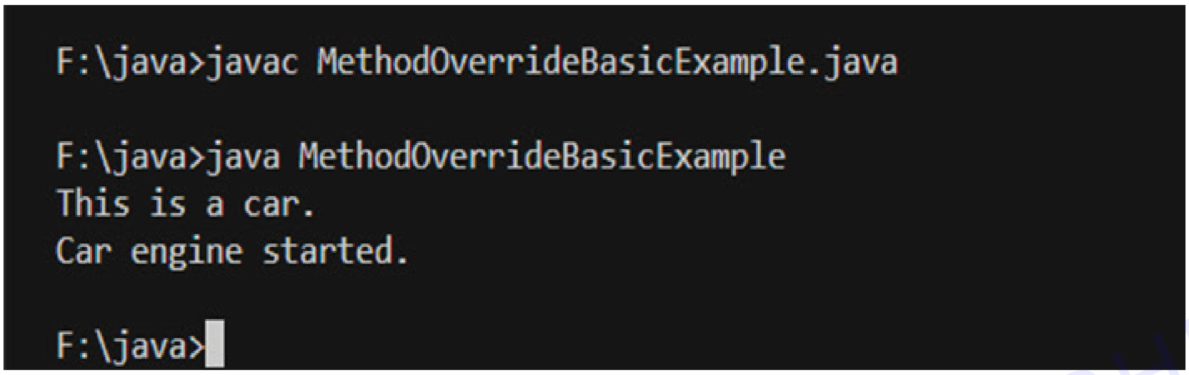
2 Derived Class (Car):
   - Extends the Vehicle class.
   - Overrides the displayInfo method to provide a specific implementation for a car.
   - Adds an additional method startEngine specific to the Car class.

3 Main Class (MethodOverrideBasicExample):

- Creates an instance of the Car class.
- Calls the overridden method displayInfo, which outputs specific information for a car.
- Calls the additional method startEngine, which is specific to the Car class.

In this basic example, the displayInfo method is overridden in the Car class to provide specific information for a car. Method overriding allows the subclass to provide its own implementation of a method defined in the superclass. Instances of the subclass can be treated as instances of the superclass, providing flexibility in coding and enhancing code reusability.

Output:



```
F:\java>javac MethodOverrideBasicExample.java

F:\java>java MethodOverrideBasicExample
This is a car.
Car engine started.

F:\java>
```

EXERCISE 113 : Create and implement an interface

Objectives

At the end of this exercise you shall be able to

- develop Java programs to Create and implement an interface.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

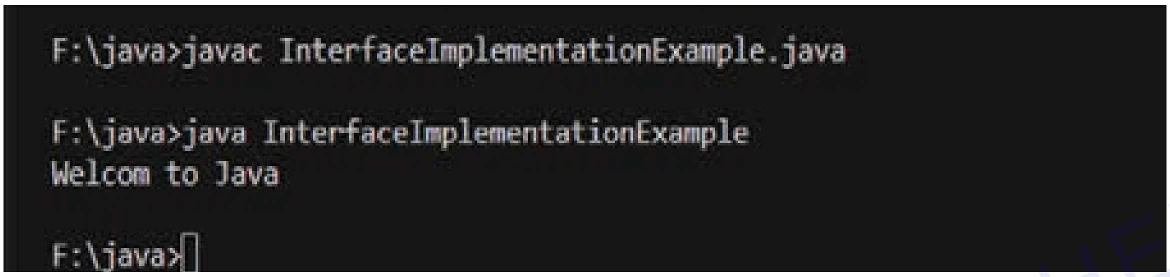
TASK 1: **Interface and Implementation**

```
    // Define an interface named Printable
    interface Printable {
        void print(); // Abstract method without implementation
    }

    // Implement the Printable interface in a class
    class Printer implements Printable {
        // Provide implementation for the print method
        @Override
        public void print() {
            System.out.println("Welcom to Java");
        }
    }

    // Main class to demonstrate interface implementation
    public class InterfaceImplementationExample {
        public static void main(String[] args) {
            // Create an instance of the Printer class
            Printer myPrinter = new Printer();

            // Call the print method through the Printable interface
            myPrinter.print();
        }
}
```

Explanation:

1. Interface (Printable):
   - Declares a single abstract method named print.
2. Class Implementing the Interface (Printer):
   - Implements the Printable interface.
   - Provides a concrete implementation for the print method.
3. Main Class (InterfaceImplementationExample):
   - Creates an instance of the Printer class.
   - Calls the print method through the Printable interface.

Output:



```
F:\java>javac InterfaceImplementationExample.java

F:\java>java InterfaceImplementationExample
Welcom to Java

F:\java>
```

In this example, the Printable interface declares an abstract method print. The Printer class implements this interface and provides a concrete implementation for the print method. The main class demonstrates how to create an instance of the implementing class and call the method through the interface.

Interfaces in Java are used to achieve abstraction, define a contract, and support multiple inheritances. Classes can implement multiple interfaces, allowing for more flexibility in the design of Java programs.

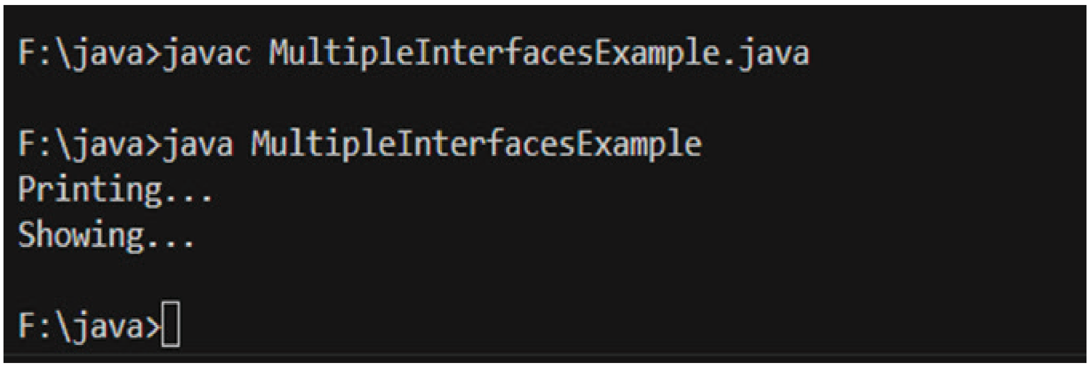
TASK 2: **Multiple Interfaces**

```
// Interface 1
interface Printable {
    void print();
}
// Interface 2
interface Showable {
    void show();
}
// Class implementing multiple interfaces
class Display implements Printable, Showable {
    @Override
    public void print() {
        System.out.println("Printing...");
    }

    @Override
    public void show() {
        System.out.println("Showing...");
    }
}

// Main class
public class MultipleInterfacesExample {
    public static void main(String[] args) {
        Display display = new Display();
        display.print();
        display.show();
    }
}
```

Explanation:

1. Interface Definitions (Printable and Showable):
    - The code defines two interfaces: Printable and Showable.
    - Each interface declares a single abstract method (print in Printable and show in Showable).
2. Class Implementation (Display):
    - The Display class implements both the Printable and Showable interfaces.
    - It provides concrete implementations for the print and show methods.
3. Method Implementations (print and show):
    - The print method prints "Printing..." to the console.
    - The show method prints "Showing..." to the console.
4. Main Class (MultipleInterfacesExample):
    - The main method creates an instance of the Display class named display.
    - It demonstrates the use of the print and show methods through the display object.

Output:



```
F:\java>javac MultipleInterfacesExample.java

F:\java>java MultipleInterfacesExample
Printing...
Showing...

F:\java>
```

In summary, this example illustrates the implementation of multiple interfaces (Printable and Showable) in a class (Display). The class provides concrete implementations for the methods defined in each interface. The main method demonstrates the usage of these methods through an instance of the implementing class, showcasing how a class can incorporate multiple interfaces in Java.

EXERCISE 114 : Extend interfaces in JAVA

Objectives

At the end of this exercise you shall be able to

- develop Java programs to extend interface in Java.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

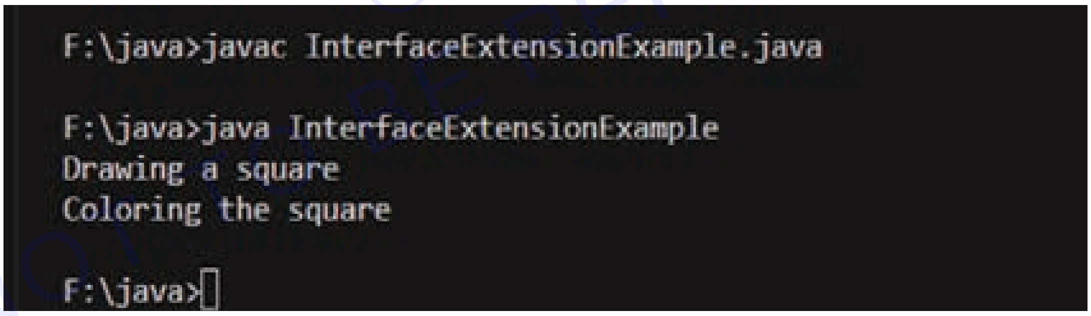
TASK 1: **Extending Interfaces**

```
// Base interface
interface Shape {
    void draw(); // Abstract method
}

// Extended interface inheriting from Shape
interface Colorable extends Shape {
    void color(); // Additional abstract method
}

// Class implementing the extended interface
class Square implements Colorable {
    @Override
    public void draw() {
        System.out.println("Drawing a square");
    }

    @Override
    public void color() {
        System.out.println("Coloring the square");
    }
}

// Main class
public class InterfaceExtensionExample {
        public static void main(String[] args) {
            Square square = new Square();
            square.draw();
            square.color();
        }
    }
```

Explanation:

1. Base Interface (Shape):
   - Defines a base interface named Shape with a single abstract method draw.
2. Extended Interface (Colorable):
   - Extends the Shape interface using the extends keyword.
   - Introduces an additional abstract method color.
3. Class Implementation (Square):
   - Implements the extended interface Colorable.
   - Provides concrete implementations for both draw and color methods.
4. Main Class (InterfaceExtensionExample):
   - Creates an instance of the Square class.
   - Demonstrates the usage of both draw and color methods through the square object.

Output:



```
F:\java>javac InterfaceExtensionExample.java

F:\java>java InterfaceExtensionExample
Drawing a square
Coloring the square

F:\java>
```

This example illustrates how an interface (Colorable) can extend another interface (Shape) to inherit its methods while adding new methods. The class Square implements the extended interface, providing implementations for all abstract methods. The main method demonstrates the usage of methods from both the base and extended interfaces through an instance of the implementing class.

— — — — —

TASK 2: **Let's explore another example that extends interfaces in Java:**

```
// Base interface
interface Animal {
    void eat(); // Abstract method
}

// Extended interface inheriting from Animal
interface Mammal extends Animal {
    void giveBirth(); // Additional abstract method
}

// Class implementing the extended interface
class Dog implements Mammal {
    @Override
    public void eat() {
        System.out.println("Dog is eating");
    }

    @Override
    public void giveBirth() {
        System.out.println("Dog gives birth to puppies");
    }
}

// Main class
public class InterfaceExtensionExample2 {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();
        myDog.giveBirth();
    }
}
```

Explanation:

1 **Base Interface (Animal):**

- Defines a base interface named Animal with a single abstract method eat.

2 **Extended Interface (Mammal):**

- Extends the Animal interface using the extends keyword.
- Introduces an additional abstract method giveBirth.

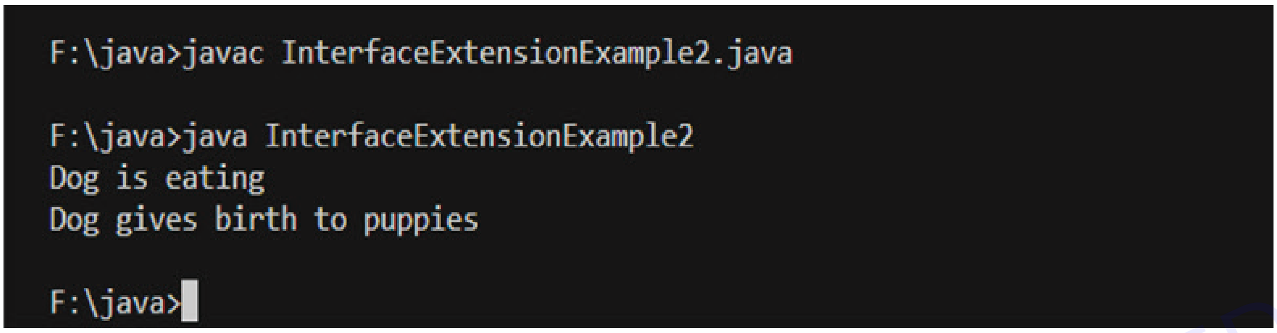
3 **Class Implementation (Dog):**

- Implements the extended interface Mammal.
- Provides concrete implementations for both eat and giveBirth methods.

4 **Main Class (InterfaceExtensionExample2):**

- Creates an instance of the Dog class.
- Demonstrates the usage of both eat and giveBirth methods through the myDog object.

5 **Output:**



```
F:\java>javac InterfaceExtensionExample2.java

F:\java>java InterfaceExtensionExample2
Dog is eating
Dog gives birth to puppies

F:\java>
```

In this example, the Mammal interface extends the Animal interface, and the Dog class implements the Mammal interface. It showcases how interfaces can be extended to inherit methods from a base interface while introducing new methods specific to the extended interface. The main method demonstrates the usage of methods from both the base and extended interfaces through an instance of the implementing class.

EXERCISE 115 : Create and use a package in JAVA

Objectives

At the end of this exercise you shall be able to

- develop Java programs to Create and use a package in JAVA.

Requirements

Tools/Materials

- PC / laptop with windows OS
- SDK software
- Test editor (Visual studio/ subline/ notepad)

Procedure

TASK 1: **Create and use a package**

```
package p2;
import java.util.Scanner;
public class Sub {
    int d;
    public void diff() {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter the first number: ");
        int x = scan.nextInt();

        System.out.print("Enter the second number: ");
        int y = scan.nextInt();

        d = x - y;
        System.out.println("Difference = " + d);

        // Close the Scanner to release resources
        scan.close();
    }
    // Main method for testing
    public static void main(String[] args) {
        Sub sub = new Sub();
        sub.diff();
    }
}
```

Explanation:

1 **Package Declaration (package p2;):** Specifies that the Sub class belongs to the p2 package.

2 **Import Statement (import java.util.Scanner;):** Imports the Scanner class from the java.util package, which is used for reading user input.

3 **Class Definition (public class Sub { ... }):** Defines the Sub class.

4 **Instance Variable (int d;):** Declares an instance variable d to store the difference between two numbers.

5 **Method Definition (public void diff() { ... }):** Defines a method named diff that calculates and displays the difference between two numbers.

6 **Scanner Initialization (Scanner scan = new Scanner(System.in);):** Creates a Scanner object to read input from the console.

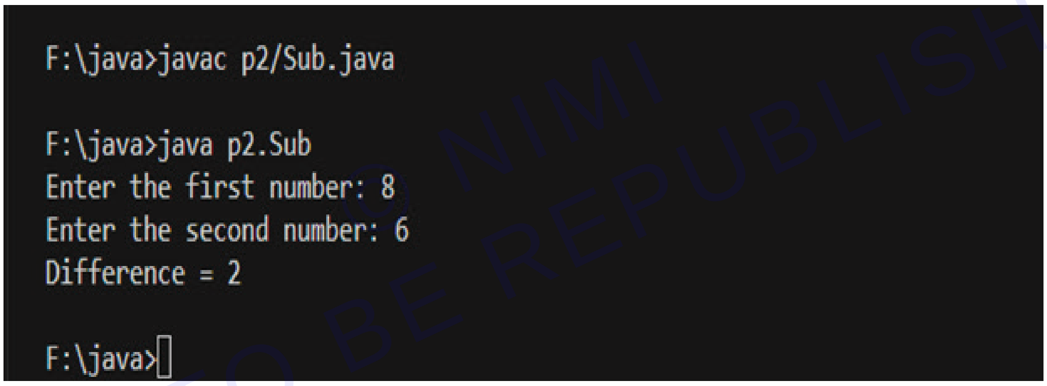
7 **User Input and Calculation (int x = scan.nextInt();**, int y = scan.nextInt();, d = x - y;): Prompts the user to enter two numbers, reads the input, and calculates the difference.

8 **Output Display (System.out.println("Difference = " + d);):** Displays the calculated difference.

9 **Resource Cleanup (scan.close();):** Closes the Scanner object to release system resources.

10 Main Method (public static void main(String[] args) { ... }): The entry point of the program. Creates an instance of the Sub class and calls the diff method for testing.

Output:



```
F:\java>javac p2/Sub.java

F:\java>java p2.Sub
Enter the first number: 8
Enter the second number: 6
Difference = 2

F:\java>
```

This program demonstrates basic input/output operations and the use of the Scanner class to interact with the user.

— — — — —

TASK 2: **Example: Using a Package (Another Example)**

Step 1: Create a Package

Create a folder named myPackage and save two Java files inside it.

Step 2: Create Classes inside the Package

1. MyMath.java

```
// Inside the 'myPackage' folder
package myPackage;
public class MyMath {
    public static int add(int a, int b) {
        return a + b;
    }
    public static int subtract(int a, int b) {
        return a - b;
    }
}
```

2 Calculator.java

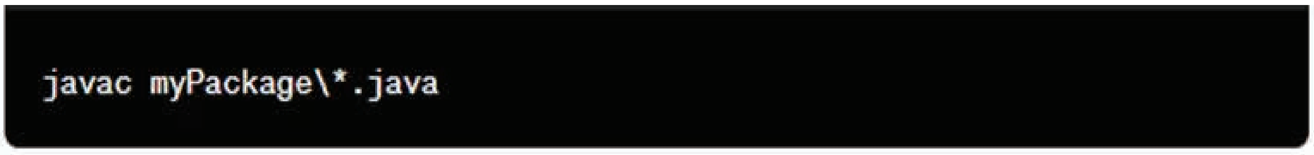
```
// Inside the 'myPackage' folder
package myPackage;
import java.util.Scanner;
public class Calculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the first number: ");
        int num1 = scanner.nextInt();

        System.out.print("Enter the second number: ");
        int num2 = scanner.nextInt();

        int sum = MyMath.add(num1, num2);
        int difference = MyMath.subtract(num1, num2);

        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);

        scanner.close();
    }
}
```

Explanation:

- The MyMath class provides simple mathematical operations (addition and subtraction).
- The Calculator class takes user input, calls methods from MyMath, and displays the results.

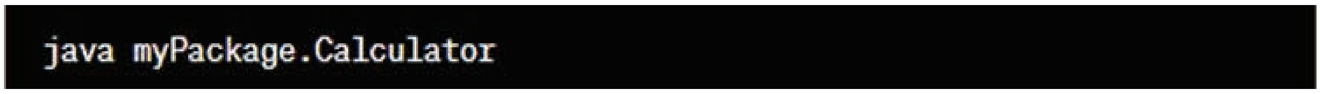
Execution:

- The program prompts the user to enter two numbers, performs addition and subtraction using methods from MyMath, and displays the results.
- **Compile the code:**



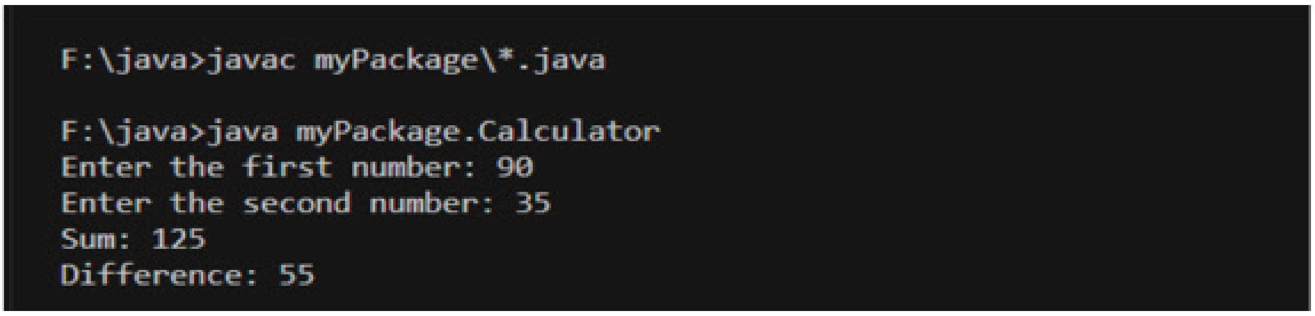
```
javac myPackage\*.java
```

- **Run the program:**



```
java myPackage.Calculator
```

Output:



```
F:\java>javac myPackage\*.java

F:\java>java myPackage.Calculator
Enter the first number: 90
Enter the second number: 35
Sum: 125
Difference: 55
```

This example demonstrates how to use multiple classes in a package, keeping related functionality organized.

Related Exersises

1 Create and use virtual methods.

- **Question:** Create a base class Shape with a virtual method calculateArea(). Create derived classes Circle and Rectangle implementing this method to calculate the area of a circle and rectangle, respectively.

2 Create abstract classes and methods.

- **Question:** Define an abstract class Bank with an abstract method calculateInterest(). Create subclasses SavingsAccount and FixedDeposit to implement this method for calculating interest.

3 Create interfaces in JAVA.

- **Question:** Define an interface Drawable with a method draw(). Create classes Circle and Square implementing this interface to provide their own implementations of drawing.

4 Override methods in JAVA.

- **Question:** Create a base class Animal with a method makeSound(). Create subclasses Dog and Cat that override this method to make different sounds.

5 Create and implement an interface.

- **Question:** Define an interface Resizable with a method resize(int percentage). Implement this interface in a class ResizableRectangle that adjusts its dimensions based on the percentage.

6 Extend interfaces in JAVA.

- **Question:** Create an interface AdvancedDrawable extending the Drawable interface with a method rotate(). Implement this interface in a class RotatableSquare.

7 Create and use a package in JAVA.

- **Question:** Create a package named utilities and include a class Calculator with methods for addition, subtraction, multiplication, and division. Use this package in a Main class to perform basic arithmetic operations.

EXERCISE 116 : Create a simple container using Frame class and extending another Frame class

Objectives

At the end of this exercise you shall be able to

- develop Java programs to Create a simple container using Frame class and extending another Frame class

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

TASK 1: **Simple Frame : a simple graphical window (frame) with a label**

```
import java.awt.Frame;
import java.awt.Label;

public class SimpleContainer extends Frame {

    public SimpleContainer(String title) {
        // Call the constructor of the superclass (Frame) with the specified title
        super(title);

        // Create a label to add to the frame
        Label label = new Label("Hello, I'm a simple container!");

        // Add the label to the frame
        add(label);

        // Set the size of the frame
        setSize(300, 200);

        // Make the frame visible
        setVisible(true);
    }
public static void main(String[] args) {
// Create an instance of SimpleContainer, passing the desired title
        SimpleContainer simpleContainer = new SimpleContainer("Simple Container Example");
    }
}
```

1 Import necessary classes:

```
import java.awt.Frame;
import java.awt.Label;
```

These lines import the Frame and Label classes from the java.awt package, which are used for creating graphical user interface components.

2 Define the class SimpleContainer:

```
public class SimpleContainer extends Frame {
```

This line declares a class named SimpleContainer that extends the Frame class. This means that SimpleContainer is a subclass of Frame and inherits its properties and methods.

3 Constructor of SimpleContainer class:

```
public SimpleContainer(String title) {
```

This line begins the constructor of the SimpleContainer class. The constructor is a special method that is called when an instance of the class is created. It takes a String parameter title to set the title of the frame.

4 Call the superclass constructor:

```
super(title);
```

This line calls the constructor of the superclass (Frame) with the provided title. It sets the title of the frame.

5 Create a label:

```
Label label = new Label("Hello, I'm a simple container!");
```

This line creates a new Label component with the specified text.

6 Add the label to the frame:

```
add(label);
```

This line adds the created label to the frame. It places the label inside the frame.

7 Set the size of the frame:

```
setSize(300, 200);
```

This line sets the size of the frame to 300 pixels in width and 200 pixels in height.

8 Make the frame visible:

```
setVisible(true);
```

This line makes the frame visible. Without this, the frame would be created but not displayed on the screen.

9 Close the constructor:

```
}
```

This line closes the constructor of the SimpleContainer class.

10 Main method:

```
public static void main(String[] args) {
```

This line starts the main method, which serves as the entry point of the program.

11 Create an instance of SimpleContainer:

This line creates an instance of the SimpleContainer class, passing the string "Simple Container Example" as the title.

12 Close the main method and the class:

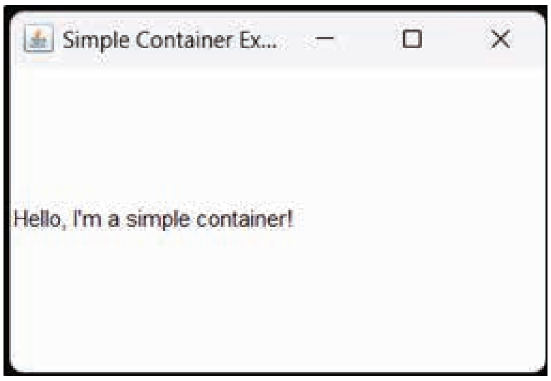
```
}
```

These lines close the main method and the SimpleContainer class.

This program creates a simple graphical window (frame) with a label saying "Hello, I'm a simple container!" and displays it on the screen when executed.

Output:

```
C:\java>javac SimpleContainer.java

C:\java>java SimpleContainer
```



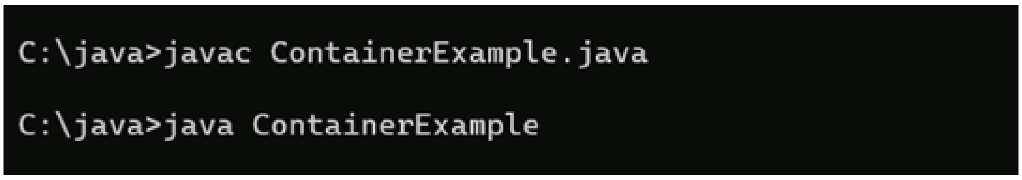
Simple Container Ex...

Hello, I'm a simple container!

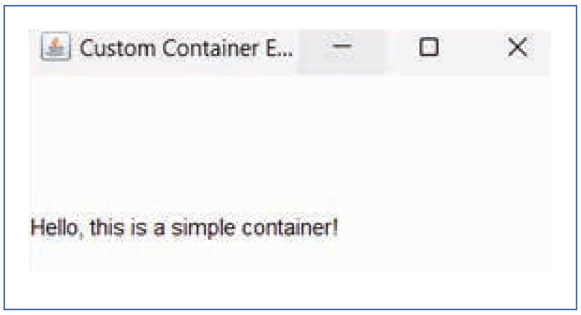
TASK 2: **A simple graphical window (frame) with a label and Window Event Handling**

```
import java.awt.Frame;
import java.awt.Label;

// CustomFrame class extending Frame
class CustomFrame extends Frame {
   public CustomFrame(String title) {
      super(title);
   }
}

// Main class demonstrating the usage
public class ContainerExample {
   public static void main(String[] args) {
      // Creating an instance of CustomFrame
      CustomFrame customFrame = new CustomFrame("Custom Container Example");

      // Adding a label to the custom frame
      Label label = new Label("Hello, this is a simple container!");
      customFrame.add(label);

      // Setting size and visibility
      customFrame.setSize(300, 200);
      customFrame.setVisible(true);

      // Adding window close event handling
      customFrame.addWindowListener(new java.awt.event.WindowAdapter() {
         public void windowClosing(java.awt.event.WindowEvent windowEvent) {
            System.exit(0);
         }
      });
}}
```

Output:



```
C:\java>javac ContainerExample.java

C:\java>java ContainerExample
```



Hello, this is a simple container!

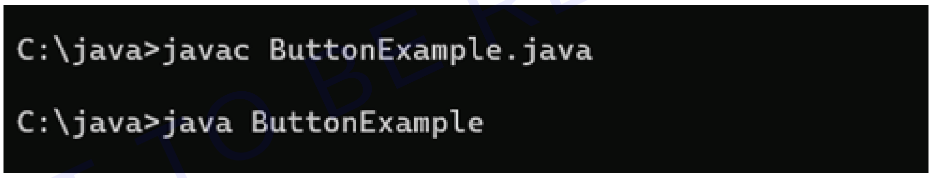
In this example, we create a custom class CustomFrame that extends the Frame class. We then create an instance of this custom class, add a Label component to it, and set the size and visibility. Additionally, a window close event listener is added to handle the closing of the frame.

TASK 3: **Button Example**

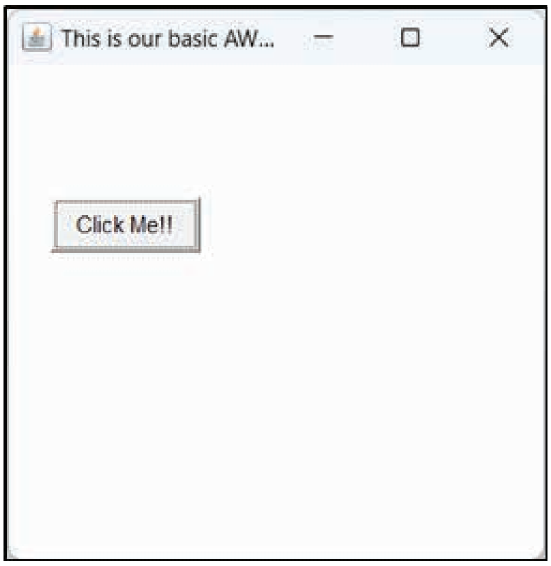
```
// importing Java AWT class
import java.awt.*;

// extending Frame class to our class ButtonExample
public class ButtonExample extends Frame {

    // initializing using constructor
    ButtonExample() {

        // creating a button
        Button b = new Button("Click Me!!");

        // setting button position on screen
       b.setBounds(30, 100, 80, 30);

        // adding button into frame
        add(b);

        // frame size 300 width and 300 height
        setSize(300, 300);

        // setting the title of Frame
        setTitle("This is our basic AWT example");

        // no layout manager
        setLayout(null);
```

```
            // now frame will be visible, by default it is not visible
            setVisible(true);

            // Adding window close event handling
            addWindowListener(new java.awt.event.WindowAdapter() {
                public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                    System.exit(0);
                }
            });
        }

        // main method
        public static void main(String args[]) {

            // creating instance of Frame class
            ButtonExample f = new ButtonExample();
        }
    }
```

Output:



```
C:\java>javac ButtonExample.java

C:\java>java ButtonExample
```

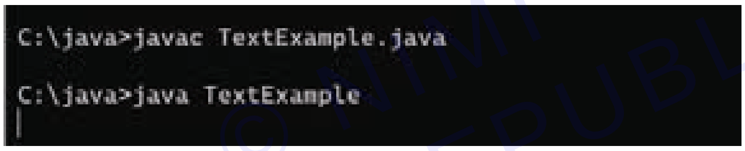


— — — — —

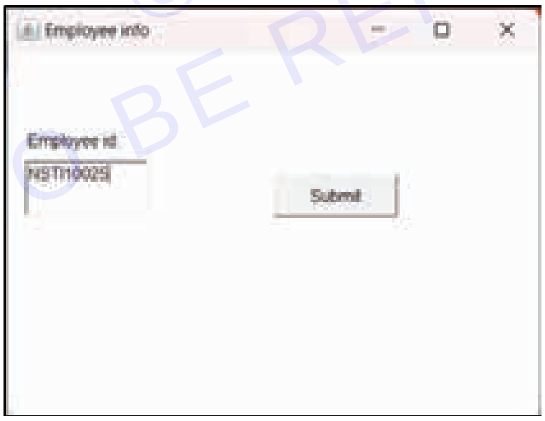
TASK 4: **Creating a TextField, Label and Button component on the Frame**

```
 import java.awt.*;

class TextExample {

    // initializing using constructor
    TextExample() {

        // creating a Frame
        Frame f = new Frame();

        // creating a Label
        Label l = new Label("Employee id:");

        // creating a Button
        Button b = new Button("Submit");
      // creating a TextField
        TextField t = new TextField();

        // setting position of above components in the frame
        l.setBounds(20, 80, 80, 30);
        t.setBounds(20, 110, 90, 40);
        b.setBounds(200, 120, 90, 30);

    // adding components into frame
        f.add(b);
        f.add(l);
        f.add(t);

        // frame size 400 width and 300 height
        f.setSize(400, 300);

        // setting the title of frame
        f.setTitle("Employee info");

        // no layout
        f.setLayout(null);

        // setting visibility of frame
        f.setVisible(true);
        // Adding window close event handling
        f.addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                System.exit(0);
            }
        });
    }

    // main method
    public static void main(String args[]) {

        // creating instance of TextExample class
        TextExample awt_obj = new TextExample();
    }
}
```

Output:



```
C:\java>javac TextExample.java

C:\java>java TextExample
```



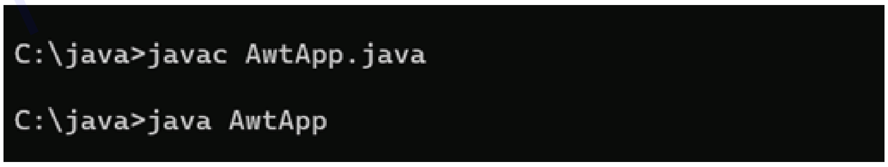
— — — — —

TASK 5: **creating simple Application Form**

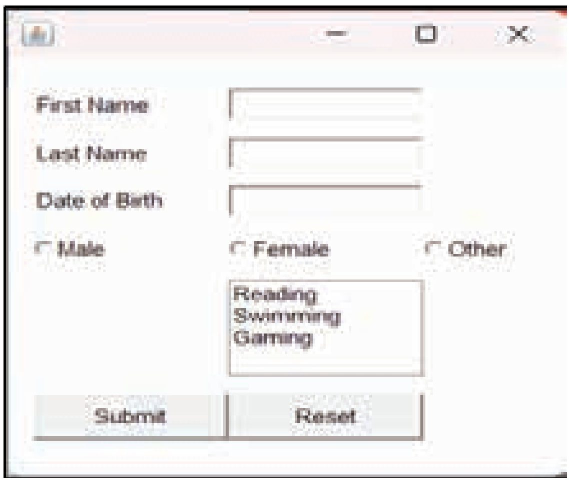
```
import java.awt.*;
public class AwtApp extends Frame {
AwtApp() {
        Label firstName = new Label("First Name");
        firstName.setBounds(20, 50, 80, 20);

        Label lastName = new Label("Last Name");
        lastName.setBounds(20, 80, 80, 20);
        Label dob = new Label("Date of Birth");
    dob.setBounds(20, 110, 80, 20);

    TextField firstNameTF = new TextField();
    firstNameTF.setBounds(120, 50, 100, 20);

    TextField lastNameTF = new TextField();
    lastNameTF.setBounds(120, 80, 100, 20);

    TextField dobTF = new TextField();
    dobTF.setBounds(120, 110, 100, 20);

    // Adding CheckboxGroup for grouping gender checkboxes
    CheckboxGroup genderGroup = new CheckboxGroup();
    Checkbox maleCheckbox = new Checkbox("Male", genderGroup, false);
    maleCheckbox.setBounds(20, 140, 80, 20);
    Checkbox femaleCheckbox = new Checkbox("Female", genderGroup, false);
    femaleCheckbox.setBounds(120, 140, 80, 20);
    Checkbox otherCheckbox = new Checkbox("Other", genderGroup, false);
    otherCheckbox.setBounds(220, 140, 80, 20);

    // Adding a List with heading "Hobbies"
    List hobbiesList = new List();
    hobbiesList.add("Reading");
    hobbiesList.add("Swimming");
    hobbiesList.add("Gaming");
    hobbiesList.setBounds(120, 170, 100, 60);
  Button sbmt = new Button("Submit");
    sbmt.setBounds(20, 240, 100, 30);

    Button reset = new Button("Reset");
    reset.setBounds(120, 240, 100, 30);

    add(firstName);
    add(lastName);
    add(dob);
    add(firstNameTF);
```

```
        add(lastNameTF);
        add(dobTF);
        add(maleCheckbox);
        add(femaleCheckbox);
        add(otherCheckbox);
        add(new Label("Gender"));
        add(hobbiesList);
        add(sbmt);
        add(reset);

        // Adding window close event handling
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                System.exit(0);
            }
        });

        setSize(300, 300);
        setLayout(null);
        setVisible(true);
    }

    public static void main(String[] args) {
     AwtApp awt = new AwtApp();
    }
}
```

Output:



```
C:\java>javac AwtApp.java

C:\java>java AwtApp
```



EXERCISE 117 : Create a container with a few controls

Objectives

At the end of this exercise you shall be able to

- develop Java programs to Create a container with a few controls

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

TASK 1: **Create a JCheckBox for Multiple Selection**

Code:

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class CheckBoxControlExample {
    public static void main(String[] args) {
        // Create a JFrame (window) with the title “CheckBox Example”
        JFrame frame = new JFrame(“CheckBox Example”);

        // Create two JCheckBox components with labels “Option 1” and “Option 2”
        JCheckBox checkBox1 = new JCheckBox(“Option 1”);
        JCheckBox checkBox2 = new JCheckBox(“Option 2”);

        // Create a JButton with the label “Submit”
        JButton submitButton = new JButton(“Submit”);

        // Add an ActionListener to the submitButton
        submitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Create a StringBuilder to build the message
                StringBuilder selectedOptions = new StringBuilder(“Selected Options: “);

                // Check if checkBox1 is selected and append the corresponding text
                if (checkBox1.isSelected()) {
```

```
                selectedOptions.append("Option 1 ");
            }

            // Check if checkBox2 is selected and append the corresponding text
            if (checkBox2.isSelected()) {
                selectedOptions.append("Option 2 ");
            }

            // Display a JOptionPane with the selected options
            JOptionPane.showMessageDialog(frame, selectedOptions.toString());
        }
    });

    // Set the layout manager for the frame to BoxLayout along the Y-axis
    frame.setLayout(new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));

    // Add the checkboxes and submit button to the frame
    frame.getContentPane().add(checkBox1);
    frame.getContentPane().add(checkBox2);
    frame.getContentPane().add(submitButton);

    // Set the size, default close operation, and make the frame visible
    frame.setSize(300, 200);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```

Explanation:

1 Imports: Import necessary classes from the javax.swing and java.awt.event packages for GUI components and event handling.

2 Class Declaration: Declare a class named CheckBoxControlExample.

3 Main Method: The main method is the entry point for the application.

4 JFrame Initialization: Create a JFrame object named frame with the title "CheckBox Example."

5 Checkboxes and Button Initialization: Create two JCheckBox components (checkBox1 and checkBox2) and a JButton (submitButton).

6 ActionListener: Add an ActionListener to the submitButton. Inside the actionPerformed method, check the selected state of each checkbox and build a message accordingly.

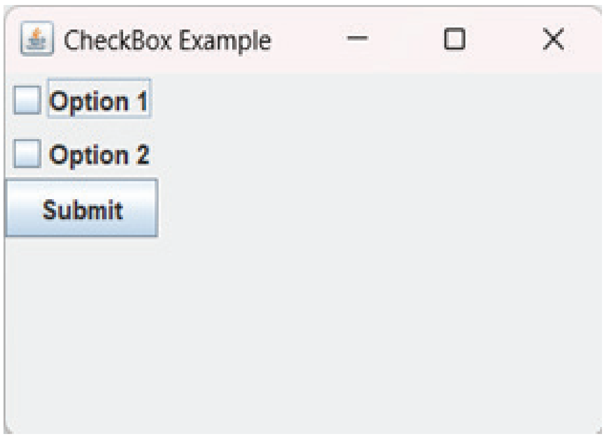
7 Layout Manager: Set the layout manager for the frame to BoxLayout along the Y-axis.

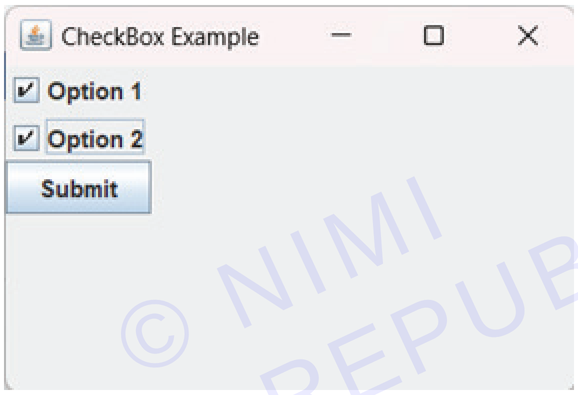
8 Component Addition: Add the checkboxes and the submit button to the frame's content pane.

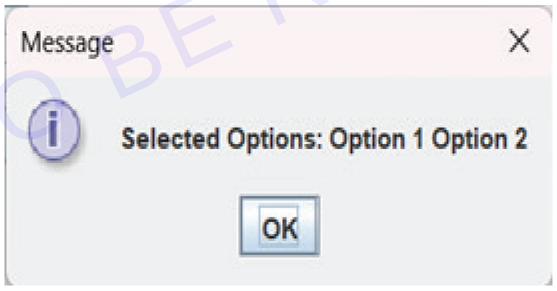
9 Frame Configuration: Set the frame's size, default close operation, and make it visible.

When you run this program, a window will appear with two checkboxes and a submit button. Upon clicking the submit button, a message dialog will show the selected options based on the checkboxes.

Output:







Selected Options: Option 1 Option 2

— — — — —

TASK 2: **Create a JCheckBox for Multiple Selection**

Code:

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class RadioButtonControlExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame(“RadioButton Example”);
        JRadioButton radioButton1 = new JRadioButton(“Male”);
        JRadioButton radioButton2 = new JRadioButton("Female");
        JButton submitButton = new JButton("Submit");

        ButtonGroup group = new ButtonGroup();
        group.add(radioButton1);
        group.add(radioButton2);

        submitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String selectedOption = radioButton1.isSelected() ? "Male" : "Female";
                JOptionPane.showMessageDialog(frame, "Selected Option: " + selectedOption);
            }
        });

        frame.setLayout(new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));
        frame.getContentPane().add(radioButton1);
        frame.getContentPane().add(radioButton2);
        frame.getContentPane().add(submitButton);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

1 **Import Statements:**

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

Import necessary packages for Swing components and event handling.

2 **Class Definition:**

```
public class RadioButtonControlExample {
```

Define a class named RadioButtonControlExample.

3 **Main Method:**

```
public static void main(String[] args) {
```

Start the main method.

4 **Frame Initialization:**

JFrame frame = new JFrame("RadioButton Example");

Create a JFrame (window) with the title "RadioButton Example".

5 **RadioButton Creation:**

JRadioButton radioButton1 = new JRadioButton("Male"); JRadioButton radioButton2 = new JRadioButton("Female");

Create two JRadioButton instances with labels "Male" and "Female".

6 **ButtonGroup Setup:**

ButtonGroup group = new ButtonGroup();

group.add(radioButton1);

group.add(radioButton2);

Create a ButtonGroup and add the radio buttons to it. This ensures that only one radio button in the group can be selected at a time.

7 **Submit Button Creation:**

JButton submitButton = new JButton("Submit");

Create a JButton with the label "Submit".

8 **ActionListener for Submit Button:**

submitButton.addActionListener(new ActionListener()

{ @Override public void actionPerformed(ActionEvent e)

{StringselectedOption=radioButton1.isSelected()?"Male":"Female";JOptionPane.showMessageDialog(frame, "Selected Option: " + selectedOption); } });

Attach an ActionListener to the submit button. When the button is clicked, it checks which radio button is selected and displays a message dialog accordingly.

9 **Frame Layout and Components Addition:**

frame.setLayout(new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS)); frame.getContentPane().add(radioButton1); frame.getContentPane().add(radioButton2); frame.getContentPane().add(submitButton);

Set the layout of the frame to a vertical box layout and add the radio buttons and the submit button to the content pane of the frame.

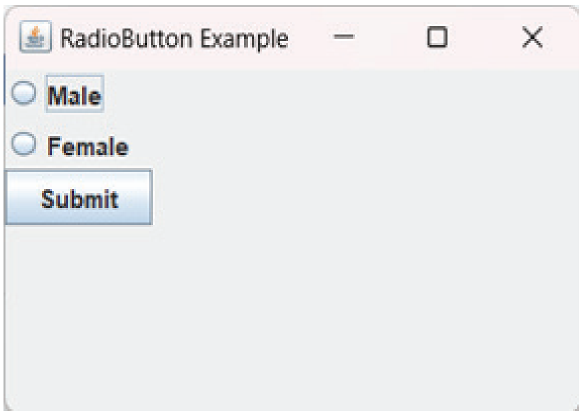
10 **Frame Configuration and Display:**

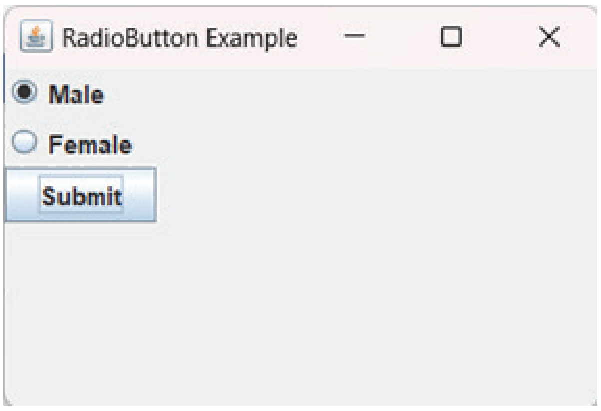
frame.setSize(300, 200); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setVisible(true);

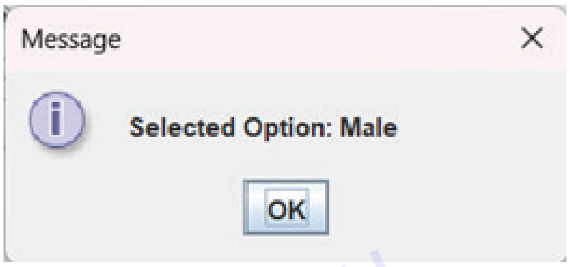
Set the size, default close operation, and make the frame visible.

This program creates a simple GUI with two radio buttons and a submit button. Upon clicking the submit button, a message dialog is shown, indicating the selected gender. The use of ButtonGroup ensures exclusive selection.

Output:







Selected Option: Male

EXERCISE 118 : Create a container with controls with action listeners and event handlers

Objectives

At the end of this exercise you shall be able to

- Develop Java programs to Create a container with controls with action listeners and event handlers.

Requirements

Tools/Materials

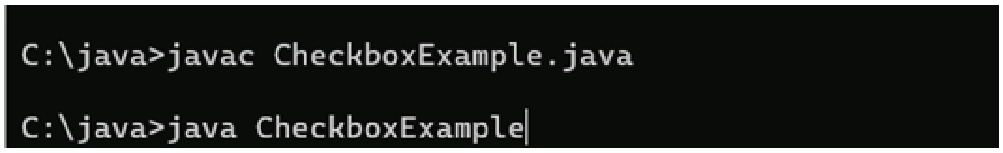
- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

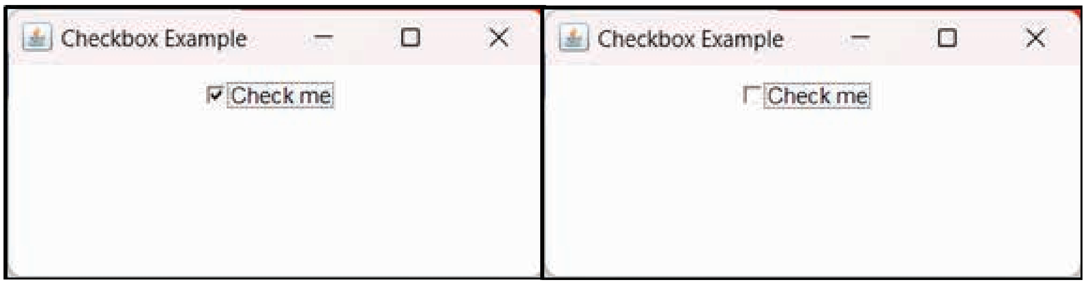
TASK 1: **Creating a container with controls (CheckBox) using AWT in Java, along with action listeners and event handlers**

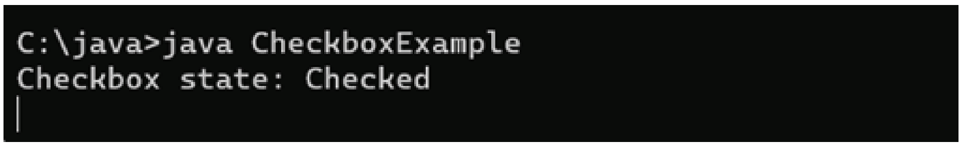
```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class CheckboxExample {


  public static void main(String[] args) {
    Frame frame = new Frame("Checkbox Example");
    Checkbox checkbox = new Checkbox("Check me");


    checkbox.addItemListener(e -> {
      System.out.println("Checkbox state: " + (checkbox.getState() ? "Checked" : "Unchecked"));
    });


    frame.add(checkbox);
    frame.setSize(300, 150);
    frame.setLayout(new FlowLayout());
    frame.setVisible(true);
frame.addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent windowEvent) {
        System.exit(0);
      }
    });
  }
}
```

Output:

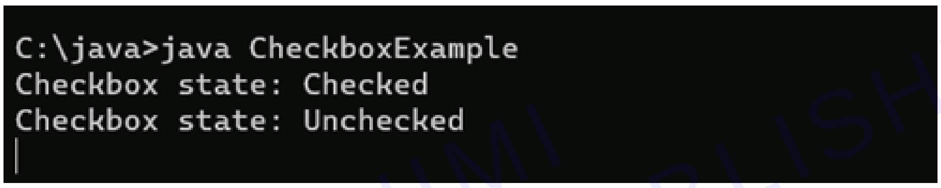


```
C:\java>javac CheckboxExample.java

C:\java>java CheckboxExample
```





```
C:\java>java CheckboxExample
Checkbox state: Checked
```



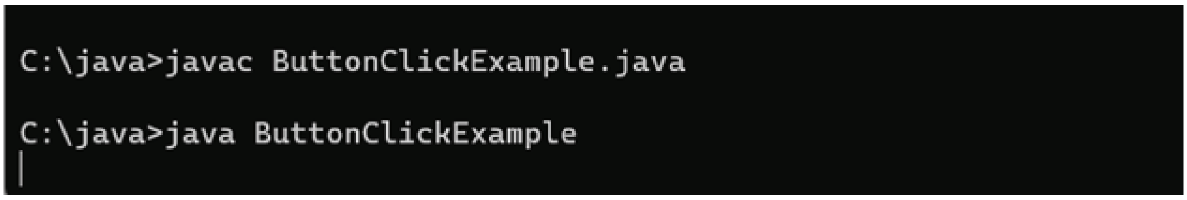
```
C:\java>java CheckboxExample
Checkbox state: Checked
Checkbox state: Unchecked
```

TASK 2: **Button Selection with Action Listener**

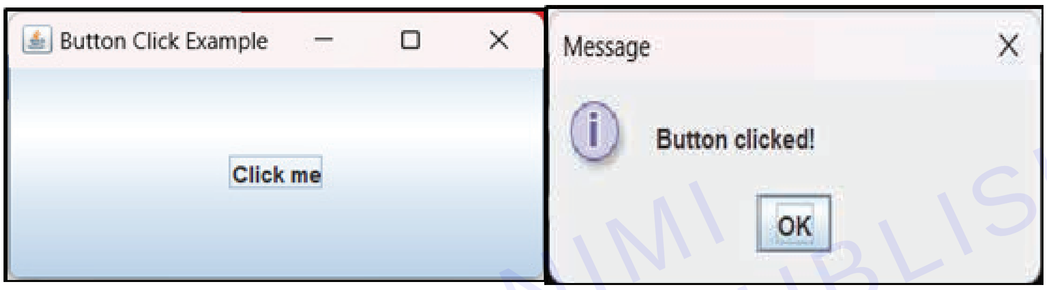
```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ButtonClickExample {

  public static void main(String[] args) {
    JFrame frame = new JFrame("Button Click Example");
    JButton button = new JButton("Click me");

    button.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Button clicked!");
      }
    });
```

```
        frame.getContentPane().add(button);
        frame.setSize(300, 150);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Output:



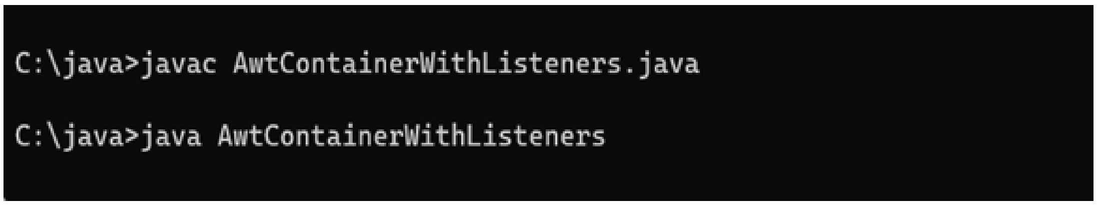
```
C:\java>javac ButtonClickExample.java

C:\java>java ButtonClickExample
```



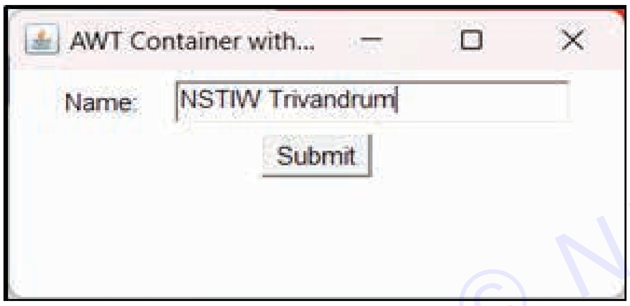
TASK 3: **Creating a container with controls (TextField, Button) using AWT in Java, along with action listeners and event handlers**

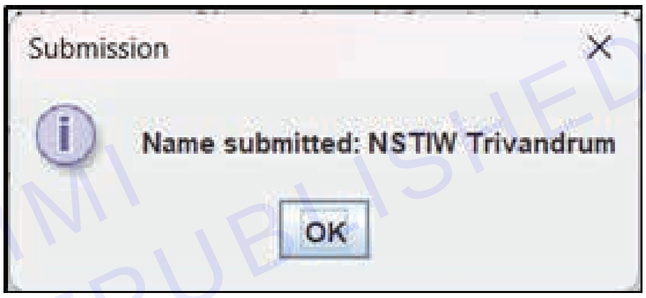
```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;

public class AwtContainerWithListeners {

    public static void main(String[] args) {
        // Creating the main frame
        Frame frame = new Frame("AWT Container with Listeners");
        frame.setSize(300, 150);

        // Creating components
        Label nameLabel = new Label("Name:");
        TextField nameTextField = new TextField(20);
        Button submitButton = new Button("Submit");
        // Setting layout to FlowLayout
        frame.setLayout(new FlowLayout());

        // Adding components to the frame
       frame.add(nameLabel);
        frame.add(nameTextField);
        frame.add(submitButton);



// Adding action listener to the button
        submitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String name = nameTextField.getText();
                System.out.println("Name submitted: " + name);
                showConfirmationDialog("Submission", "Name submitted: " + name);
            }
        });

        // Adding event handler to the text field
        nameTextField.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
           String name = nameTextField.getText();
               System.out.println("Enter key pressed in the text field. Name: " + name);
        showConfirmationDialog("Text Field Event", "Enter key pressed. Name: " + name);
            }
        });

        // Setting frame visibility
        frame.setVisible(true);

        // Adding window close event handling
        frame.addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                System.exit(0);
            }
        });
    }
    private static void showConfirmationDialog(String title, String message) {
        // Display a confirmation dialog
        JOptionPane.showMessageDialog(null, message, title, JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Output:



```
C:\java>javac AwtContainerWithListeners.java

C:\java>java AwtContainerWithListeners
```





Name submitted: NSTIW Trivandrum

EXERCISE 119 : Create a GUI to draw different plane shapes over a predefined area

Objectives

At the end of this exercise you shall be able to

- Develop Java programs to Create a GUI to draw different plane shapes over a predefined area.

Requirements

Tools/Materials

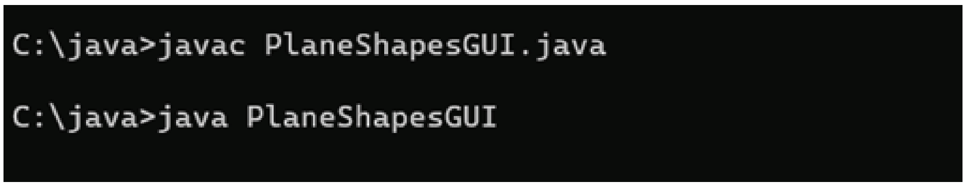
- PC/Laptop with Windows OS
- JDK Software
- Text Editor (Visual Studio/Sublime/Notepad)

Procedure

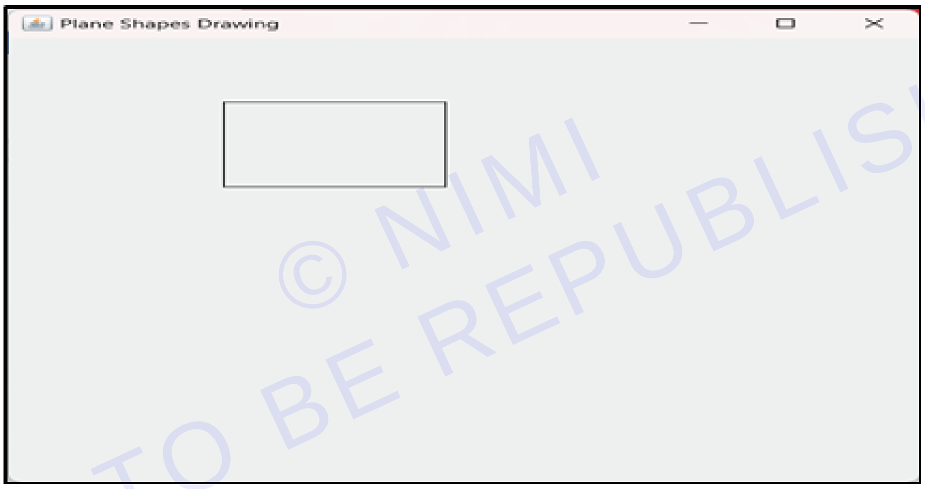
TASK 1: **Example program that allows the user to draw different plane shapes (e.g., circles, rectangles) on a JPanel using mouse interactions**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;

class PlaneShapesGUI extends JFrame {

    private ArrayList<Shape> shapes = new ArrayList<>();
    private Shape currentShape;
    private int startX, startY;

    public PlaneShapesGUI() {
        setTitle("Plane Shapes Drawing");
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel drawingPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                drawShapes(g);
            }
        };
        drawingPanel.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                startX = e.getX();
                startY = e.getY();
            }

            @Override
           public void mouseReleased(MouseEvent e) {
                int endX = e.getX();
                int endY = e.getY();
                createShape(startX, startY, endX, endY);
                repaint();
            }
        });

        add(drawingPanel);
    }

    private void drawShapes(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setColor(Color.BLACK);

        for (Shape shape : shapes) {
            g2d.draw(shape);
        }
    }

    private void createShape(int startX, int startY, int endX, int endY) {
        int width = Math.abs(endX - startX);
        int height = Math.abs(endY - startY);

        if (width > 0 && height > 0) {
            currentShape = new Rectangle(startX, startY, width, height);
            shapes.add(currentShape);
        }
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            PlaneShapesGUI gui = new PlaneShapesGUI();
            gui.setVisible(true);
        });
    }
}
```

Output:



```
C:\java>javac PlaneShapesGUI.java

C:\java>java PlaneShapesGUI
```

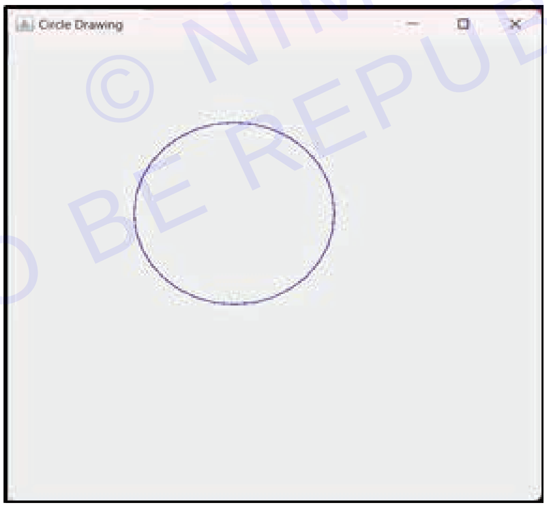


TASK 2: **Drawing Circles**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.awt.geom.Ellipse2D;
public class CircleDrawingGUI extends JFrame {

    private ArrayList<Shape> circles = new ArrayList<>();
    private Shape currentCircle;
    private int centerX, centerY;
  public CircleDrawingGUI() {
        setTitle(“Circle Drawing”);
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel drawingPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                drawCircles(g);
            }
        };

        drawingPanel.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                centerX = e.getX();
                centerY = e.getY();
            }

            @Override
            public void mouseReleased(MouseEvent e) {
                int radius = (int) Math.sqrt(Math.pow(e.getX() - centerX, 2) + Math.pow(e.getY() - centerY, 2));
                createCircle(centerX, centerY, radius);
                repaint();
            }
        });

        add(drawingPanel);
    }

    private void drawCircles(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setColor(Color.BLUE);
       for (Shape circle : circles) {
            g2d.draw(circle);
        }
    }
    private void createCircle(int centerX, int centerY, int radius) {
        currentCircle = new Ellipse2D.Double(centerX - radius, centerY - radius, 2 * radius, 2 * radius);
        circles.add(currentCircle);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            CircleDrawingGUI gui = new CircleDrawingGUI();
            gui.setVisible(true);
        });
    }
}
```

Output:

```
C:\java>javac CircleDrawingGUI.java

C:\java>java CircleDrawingGUI
```

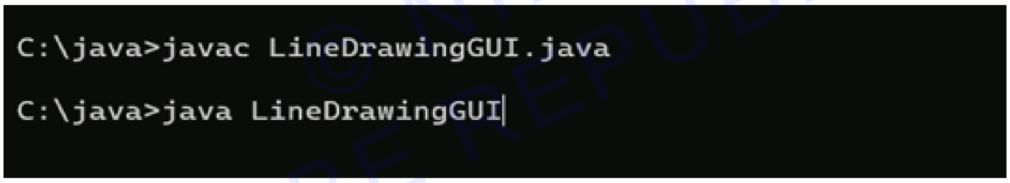


— — — — —

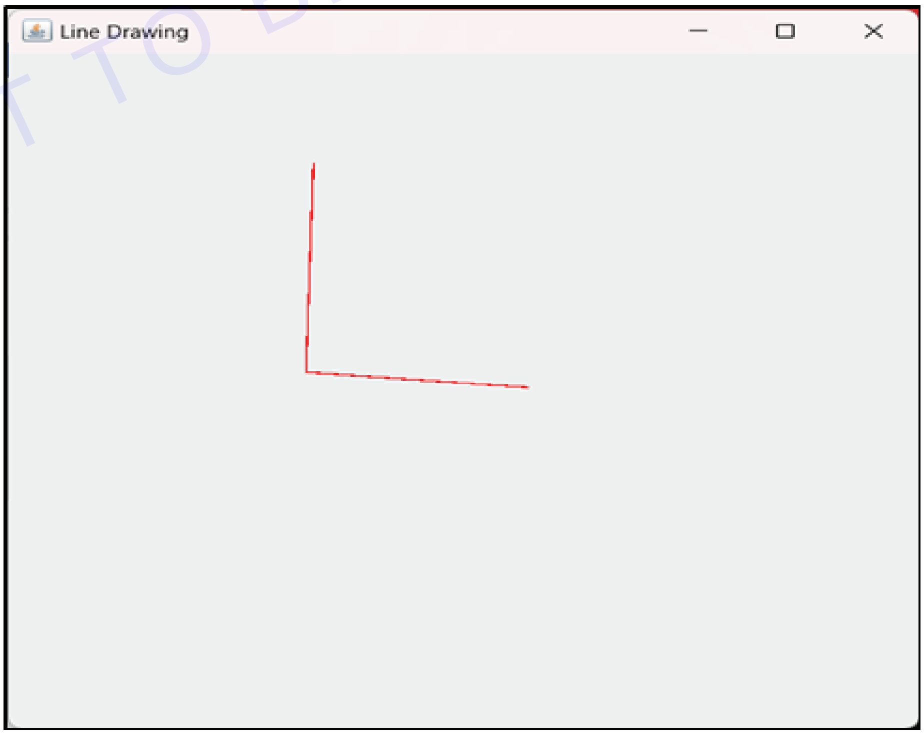
TASK 3: **Drawing Lines**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.awt.geom.Line2D;
class LineDrawingGUI extends JFrame {
    private ArrayList<Shape> lines = new ArrayList<>();
    private Shape currentLine;
    private int startX, startY;

    public LineDrawingGUI() {
        setTitle("Line Drawing");
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel drawingPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                drawLines(g);
            }
        };

        drawingPanel.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                startX = e.getX();
                startY = e.getY();
            }

            @Override
  public void mouseReleased(MouseEvent e) {
                createLine(startX, startY, e.getX(), e.getY());
                repaint();
            }
        });

        add(drawingPanel);
    }

    private void drawLines(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setColor(Color.RED);
        for (Shape line : lines) {
            g2d.draw(line);
        }
    }

    private void createLine(int startX, int startY, int endX, int endY) {
        currentLine = new Line2D.Double(startX, startY, endX, endY);
        lines.add(currentLine);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            LineDrawingGUI gui = new LineDrawingGUI();
            gui.setVisible(true);
        });
    }
}
```

Output:



```
C:\java>javac LineDrawingGUI.java

C:\java>java LineDrawingGUI
```



Related Exercises

1 **Create a Login Form using AWT:**

Design a simple login form with two TextFields (for username and password), a Button (for login), and display a message using a Label based on the login attempt.

2 **Calculator Application using AWT:**

Build a basic calculator application with buttons for digits, arithmetic operations, and a TextField to display the result.

3 **Temperature Converter:**

Create a temperature converter application using AWT. Include TextFields for input and output, Buttons for conversion between Celsius and Fahrenheit, and a Label to display the result.

4 **Student Registration Form:**

Design a student registration form with TextFields for entering student details such as name, roll number, and a Choice component for selecting the course. Use a Button to submit the form and display the entered information in a TextArea.

5 **Simple Drawing App:**

Develop a simple drawing application using AWT. Include buttons for selecting different shapes (e.g., line, circle, rectangle) and a Canvas to draw these shapes. Implement event handling to draw the selected shape when the user clicks on the Canvas.

6 **Password Strength Checker:**

Create a password strength checker using AWT. Include a TextField for entering the password, a Button to check the strength, and a Label to display the result (e.g., Weak, Medium, Strong).

7 **Image Viewer:**

Build a basic image viewer using AWT. Include buttons for opening an image, zooming in/out, and scrolling through multiple images. Display the selected image in a Canvas.

8 **Chat Application:**

Design a simple chat application with TextAreas for displaying chat history and entering messages, a Button to send messages, and a TextField for entering the username.

9 **File Explorer:**

Create a basic file explorer using AWT components. Include a List or TextArea to display the list of files in a directory and buttons for navigating through directories.

10 **Color Picker:**

Develop a color picker application using AWT. Include sliders for adjusting RGB values, a Canvas to display the selected color, and a TextField to show the hexadecimal color code

Module 7 : Programming Language (Python)

EXERCISE 120 : Install, set up the environment & run Python

Objectives

At the end of this exercise you shall be able to

- download python software
- install python.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

Python is a high-level, interpreted, and general-purpose programming language known for its readability and simplicity. Python has gained widespread popularity due to its ease of learning, versatility, and extensive community support. This module will guide you Python programming with hands-on examples and real-world applications.

In order to become Python developer, the first step is to learn how to install or update Python on a local machine or computer. In this tutorial, we will discuss the installation of Python on various operating systems.

Procedure

Installation on Windows

Visit the link https://www.python.org to download the latest release of Python. In this process, we will install Python 3.12.2 on our Windows operating system. When we click on the above link, it will bring us the following page.

Step 1: Download Python:

- Visit the official Python website: https://www.python.org/.
- Navigate to the "Downloads" section.
- Download the latest version suitable for your operating system (Windows, macOS, or Linux)

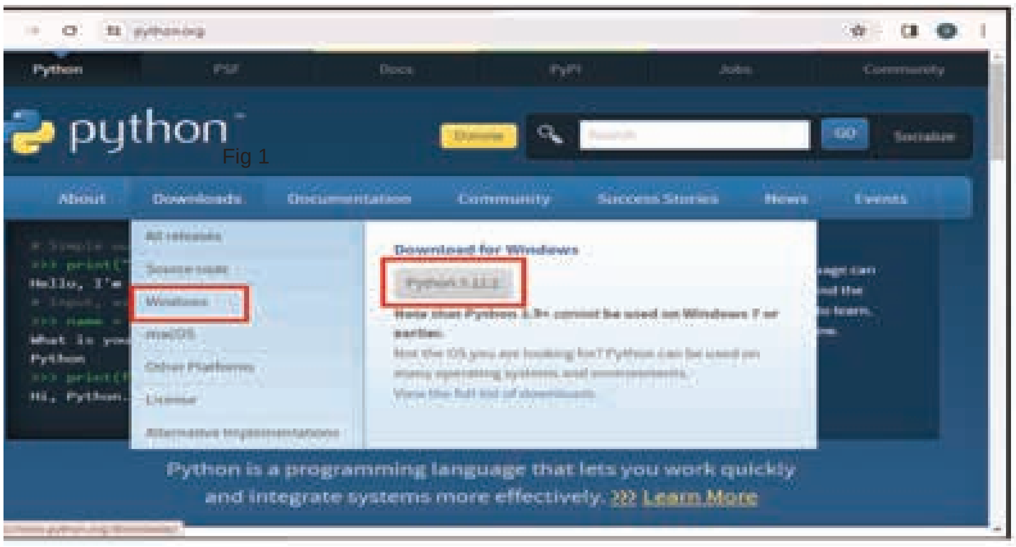
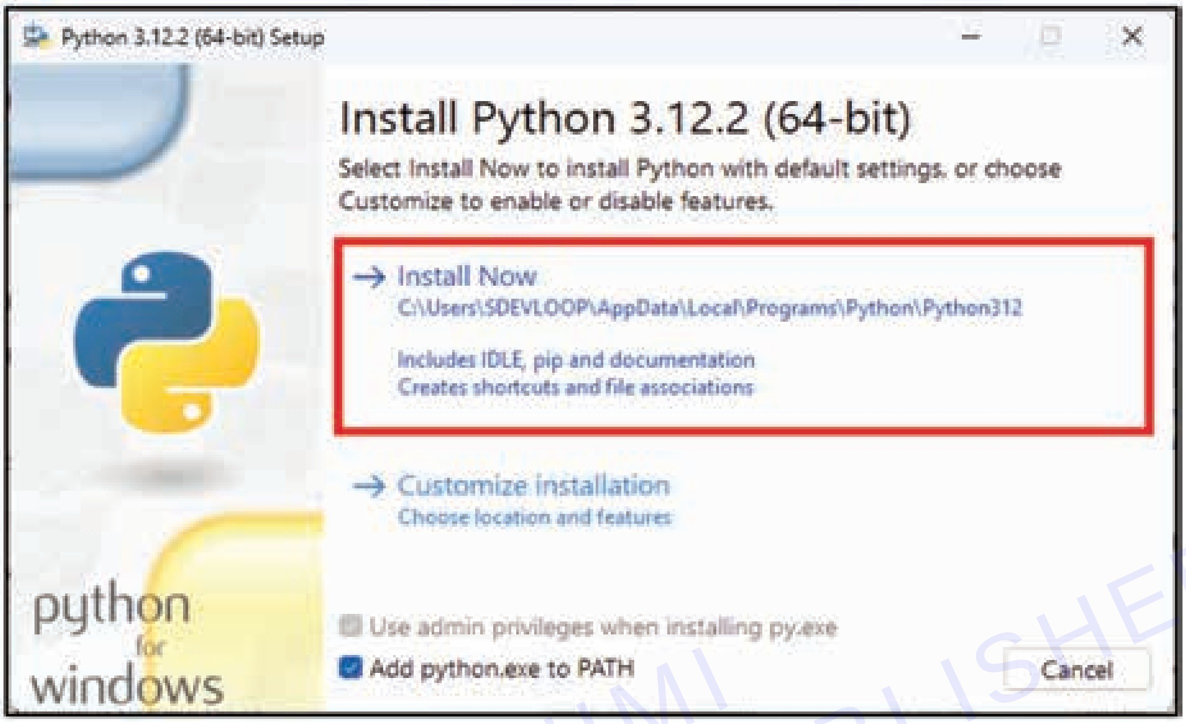


Fig 1

Step 2: Install Python:

- Run the installer that you downloaded.



Python 3.12.2 (64-bit) Setup

Install Python 3.12.2 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

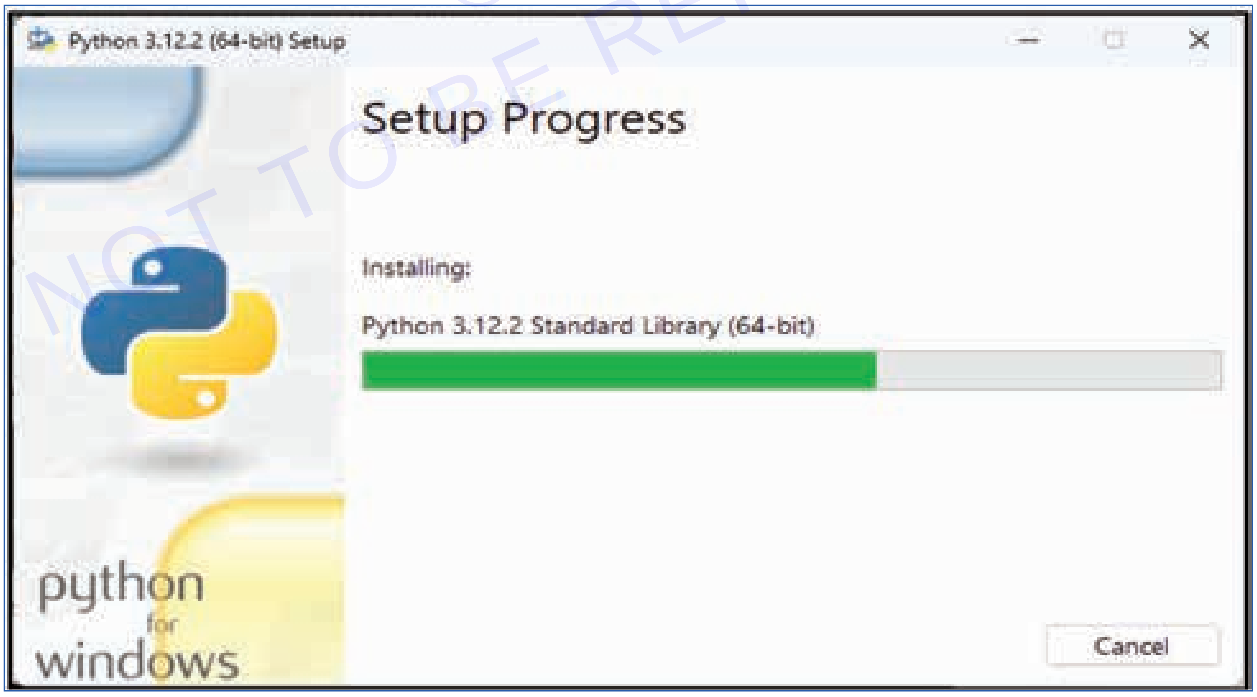
→ Install Now
C:\Users\SDEVLOOP\AppData\Local\Programs\Python\Python312

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☐ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

- During the installation, make sure to check the option that says “Add Python to PATH” to make Python accessible from the command line.



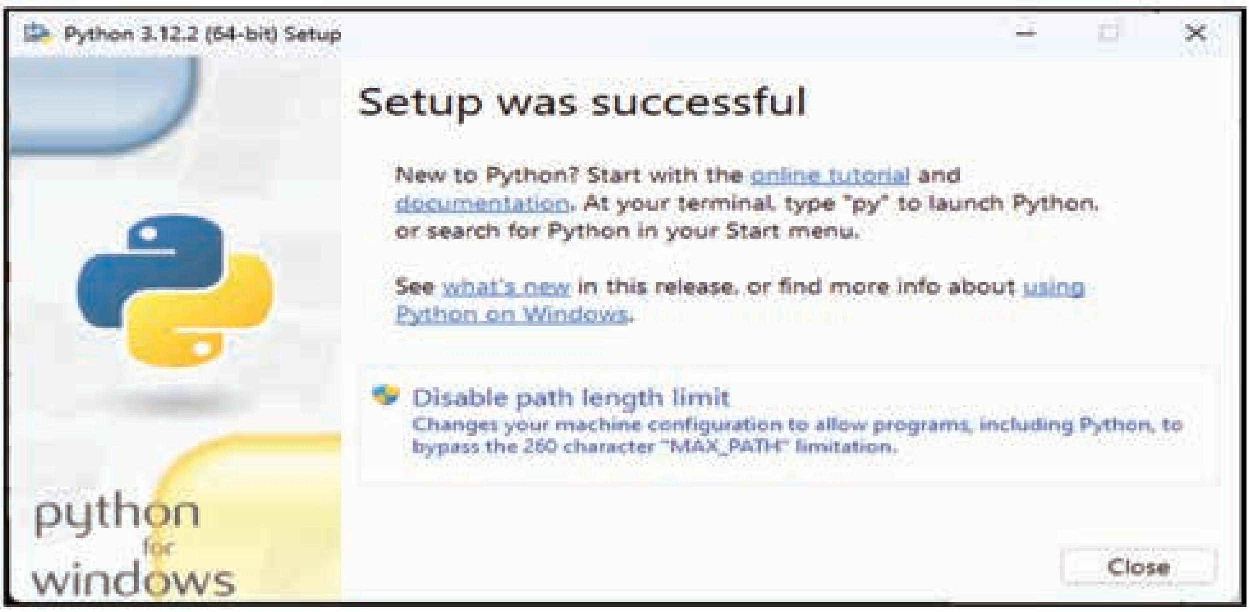
Python 3.12.2 (64-bit) Setup

Setup Progress

Installing:

Python 3.12.2 Standard Library (64-bit)

- Complete the installation by following the prompts.



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

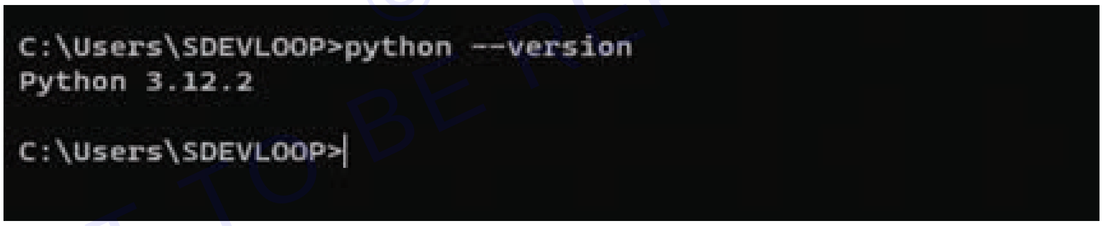
Verify Python Installation:

1 Open Command Prompt or Terminal:

- On Windows, you can press Win + R, type cmd, and press Enter.
- On macOS or Linux, you can open Terminal.

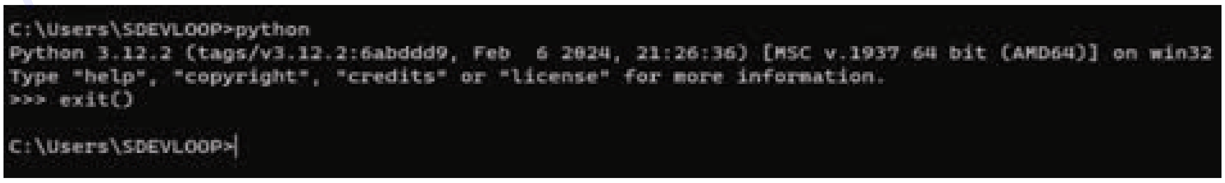
2 Check Python Version:

- Type the following command and press Enter:



```
C:\Users\SDEVLOOP>python --version
Python 3.12.2

C:\Users\SDEVLOOP>
```

- This should display the installed Python version.
- You can also check the Python interpreter by typing 'python' :



```
C:\Users\SDEVLOOP>python
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\SDEVLOOP>
```

This will open the Python interpreter, and you can exit it by typing exit().

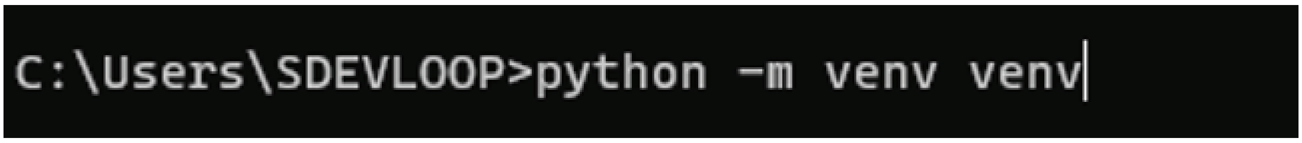
Set Up Python Environment:

1 **Install a Code Editor (Optional):**

- You may choose a code editor like Visual Studio Code, PyCharm, or Jupyter Notebook for a more convenient development environment.

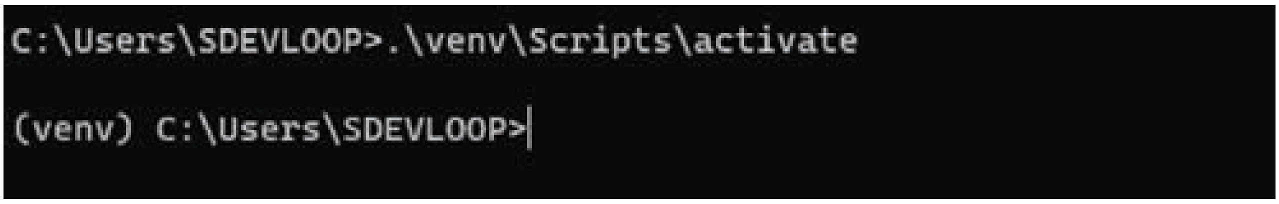
2 **Create a Virtual Environment (Optional but recommended):**

- Open the command prompt or terminal and navigate to your project directory.
- Run the following commands to create and activate a virtual environment:



```
C:\Users\SDEVLOOP>python -m venv venv
```

- On Windows, activate the virtual environment with:



```
C:\Users\SDEVLOOP>.\venv\Scripts\activate

(venv) C:\Users\SDEVLOOP>
```

- By using virtual environments, you can maintain a clean and organized development environment for each of your Python projects.

EXERCISE 121 : Use Command Line and IDE to create and execute a python program

Objectives

At the end of this exercise you shall be able to

- develop simple python programs and execute it in Commandline/IDE.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

Procedure

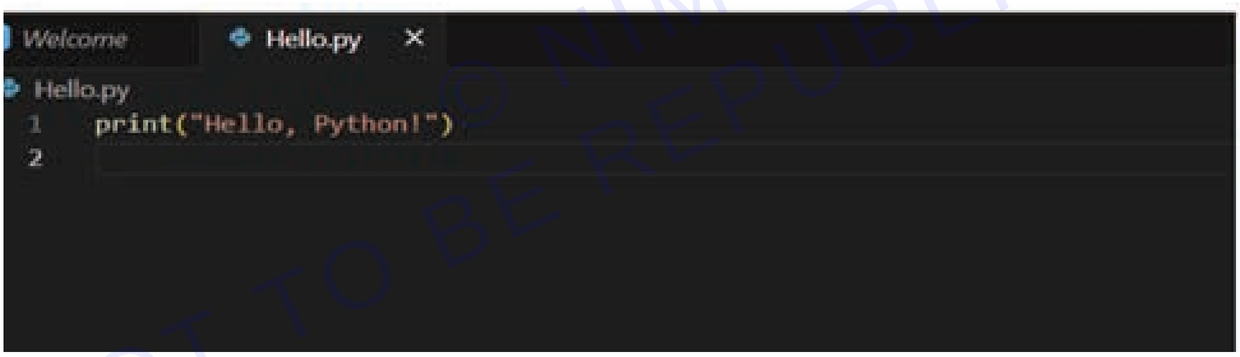
Write and Run a Simple Python Program

1 **Create a Python File:**

- Use a text editor or your chosen code editor to create a file with a .py extension (e.g., hello.py).

2 **Write a Simple Python Program:**

- Open the file and write a simple Python program, for example:

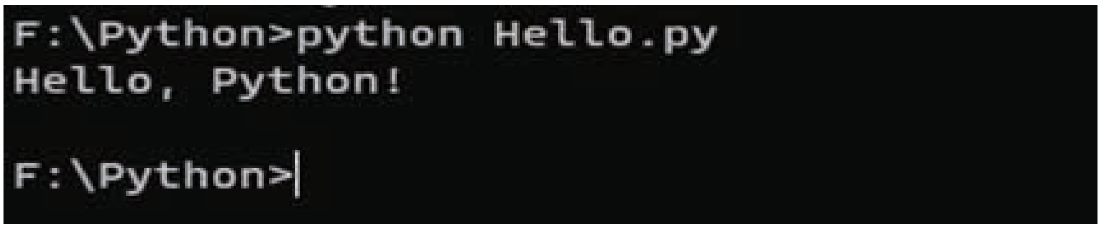


```
print("Hello, Python!")
```

Run the Program:

Method 1: Usingcommand prompt or terminal

- Save the file and return to the command prompt or terminal.
- Navigate to the directory where your Python file is located.
- Run the Python script with:



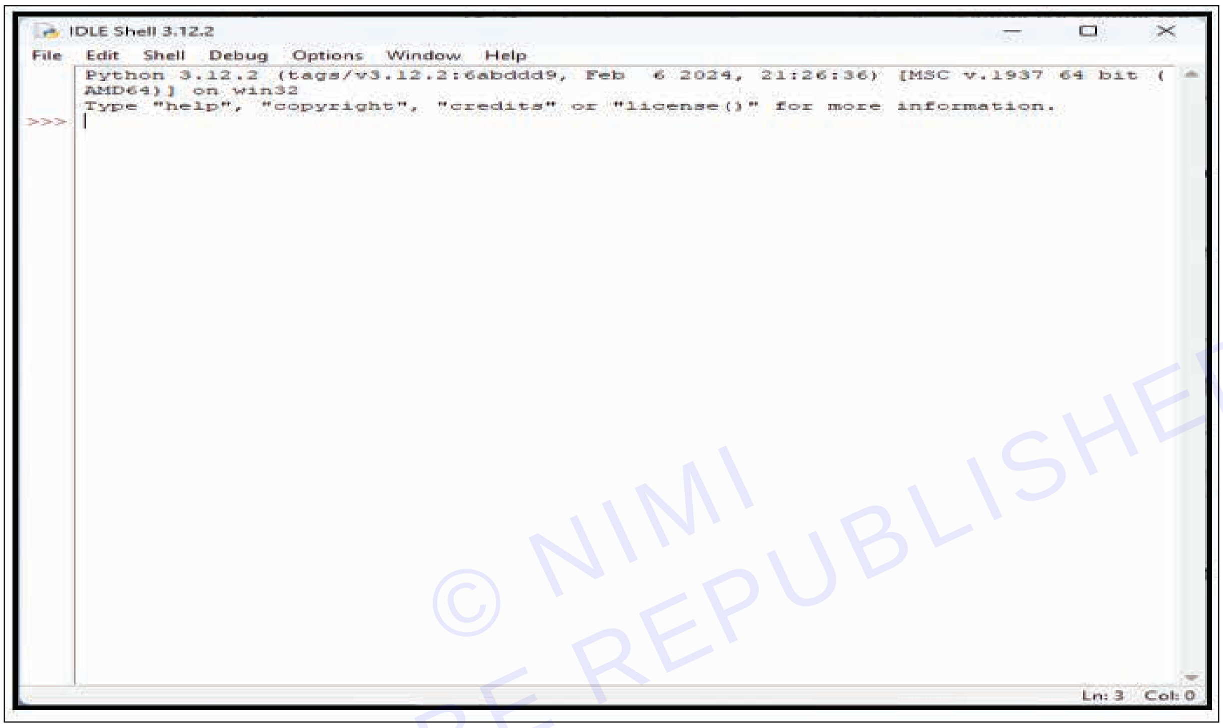
```
F:\Python>python Hello.py
Hello, Python!

F:\Python>
```

Method 2: UsingPython IDLE

Python IDLE (Integrated Development and Learning Environment) is an integrated development environment for Python. It is a simple and lightweight IDE that comes bundled with the Python programming language. IDLE provides a convenient environment for writing, testing, and debugging Python code.

1 **Open IDLE:**

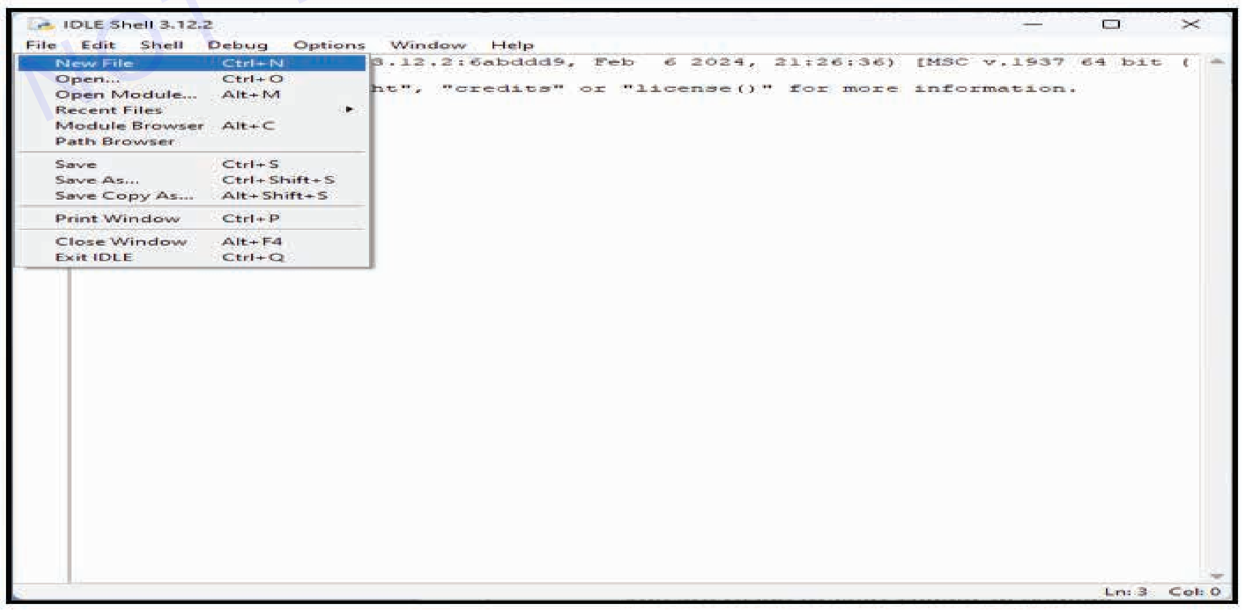
- Launch IDLE by searching for it in your computer's applications or programs menu.
- On Windows, you can also open it by searching for "IDLE" in the Start menu.



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> |
```

2 **Create a New File:**

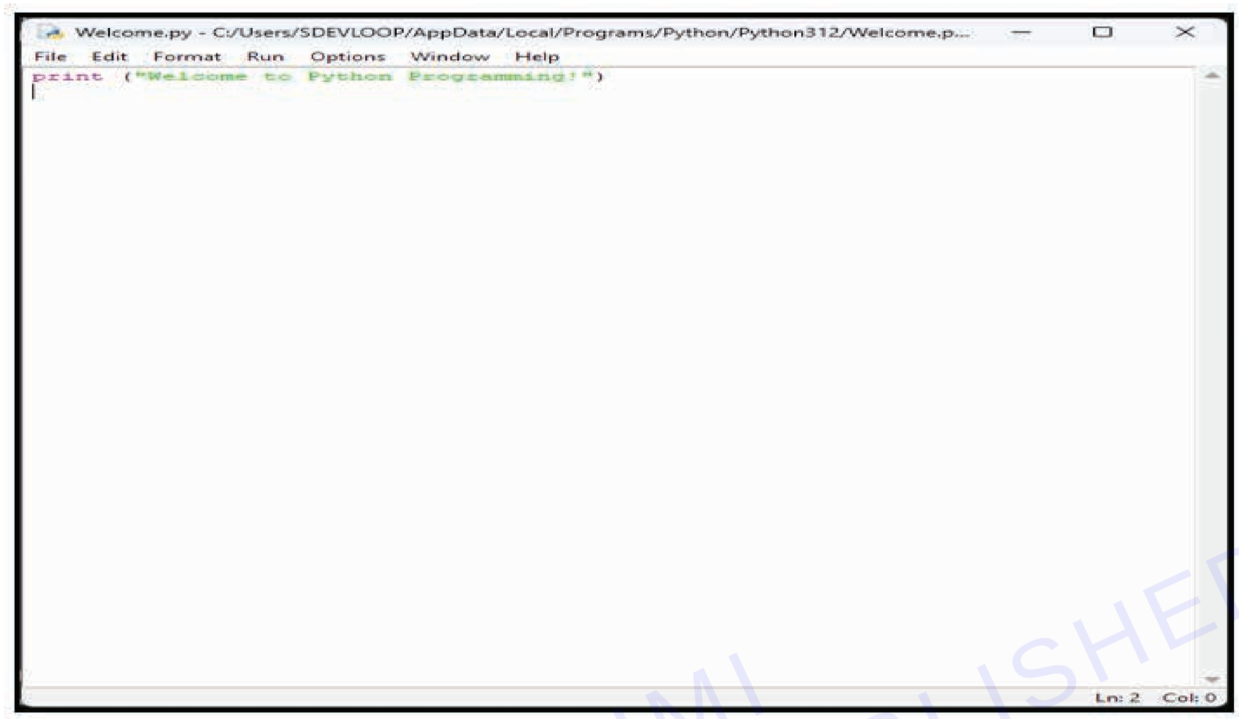
- In IDLE, go to the "File" menu and choose "New File" to open a new script editor window.



```
3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
ht", "credits" or "license()" for more information.
```

3 **Write Your Python Code:**

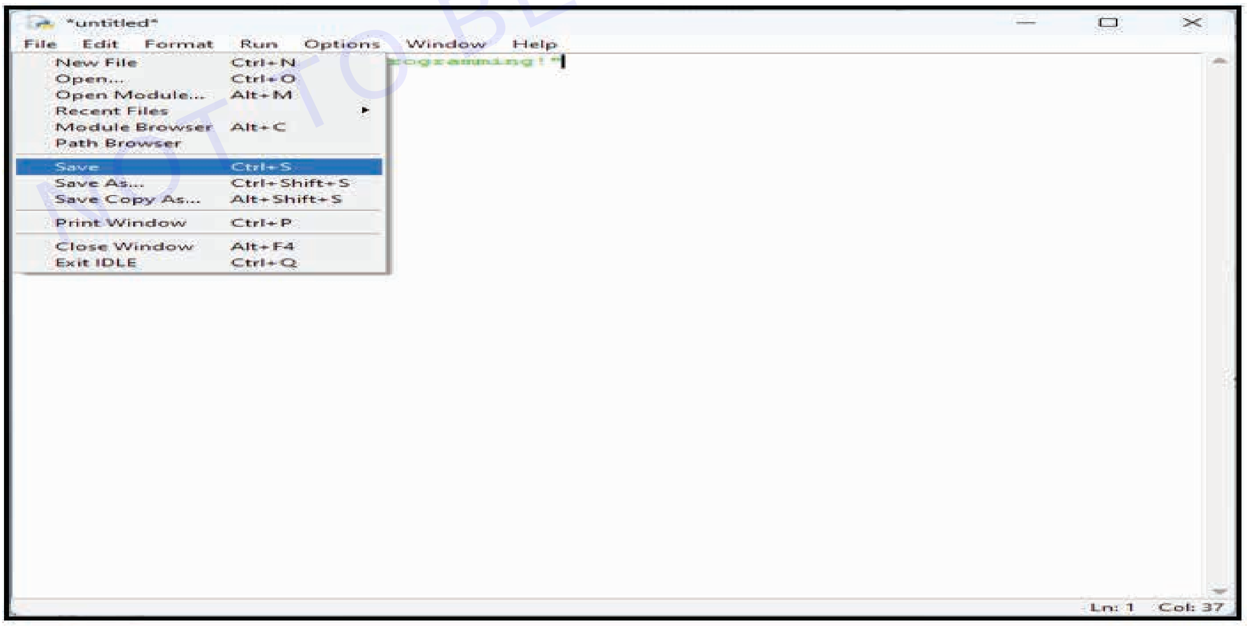
- Write or paste your Python code into the script editor.


Welcome.py - C:/Users/SDEVLOOP/AppData/Local/Programs/Python/Python312/Welcome.p...

```
print ("Welcome to Python Programming!")
```

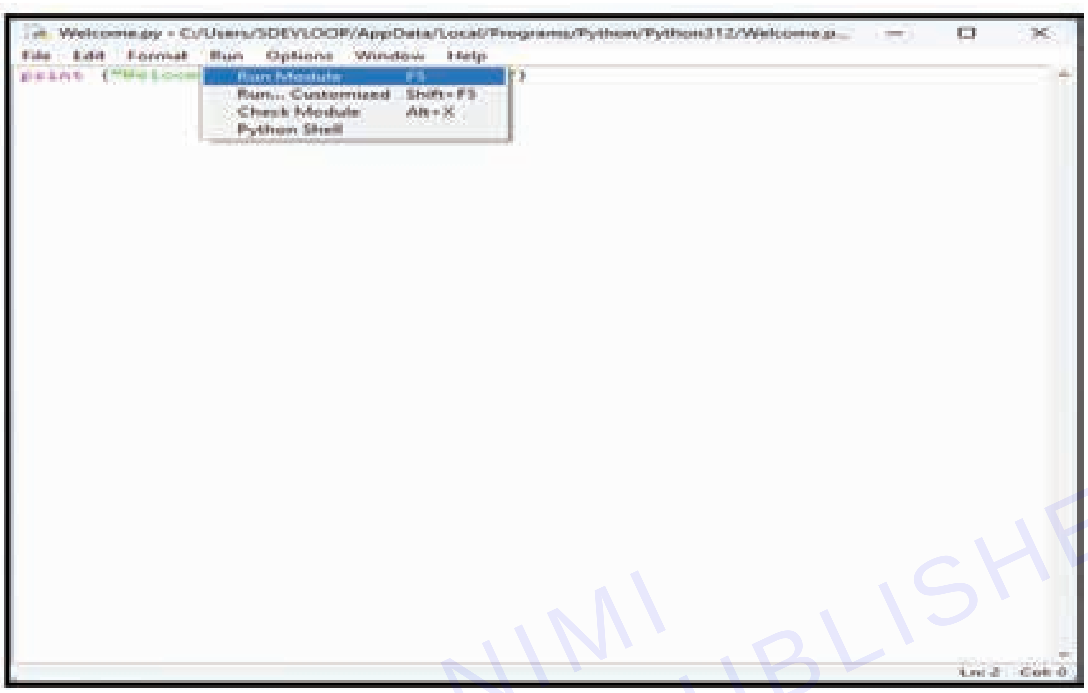
4 **Save Your Python Script:**

- Save your Python script by going to the "File" menu and selecting "Save" or "Save As."
- Choose a file name and location on your computer to save the script with a .py extension (e.g., myscript.py).



5 **Run Your Python Script:**

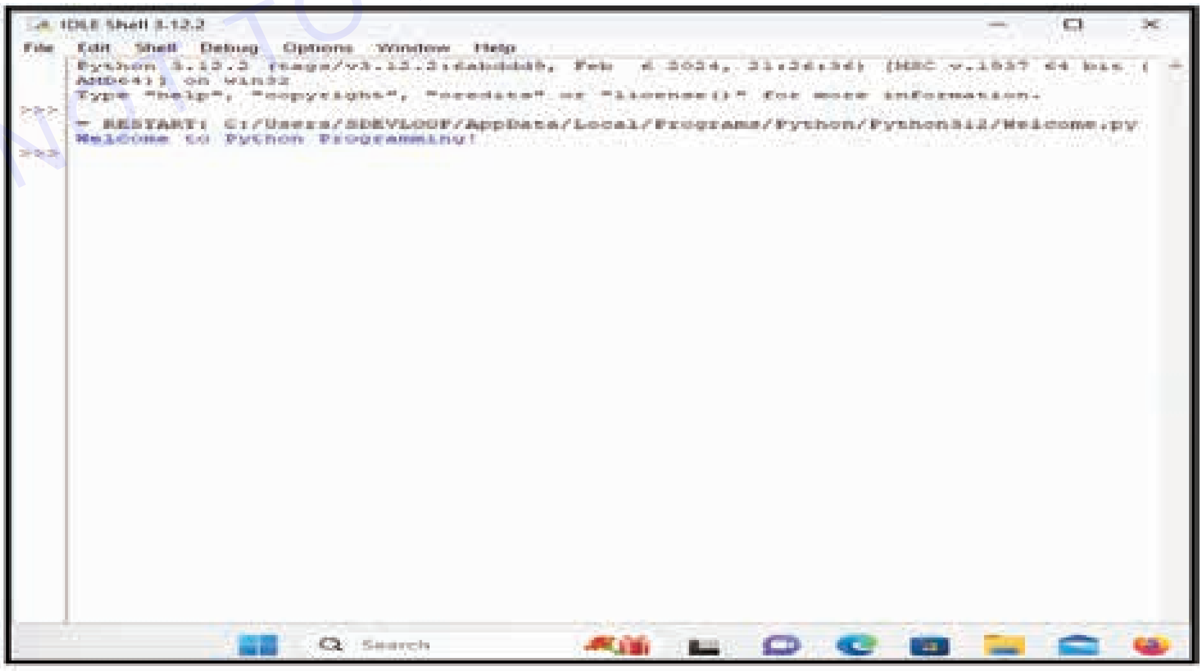
- After saving your script, you can run it by selecting the "Run" menu and choosing "Run Module" or using the keyboard shortcut F5.
- Alternatively, you can use the command Run -> Run Module from the menu.


Welcome.py - C:/Users/SDEVLOOP/AppData/Local/Programs/Python/Python312/Welcome.p...
File
Edit
Format
Run
Options
Window
Help
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell
Ln: 2 Col: 0

```
print ("Welcom
```

6 **View Output in Shell:**

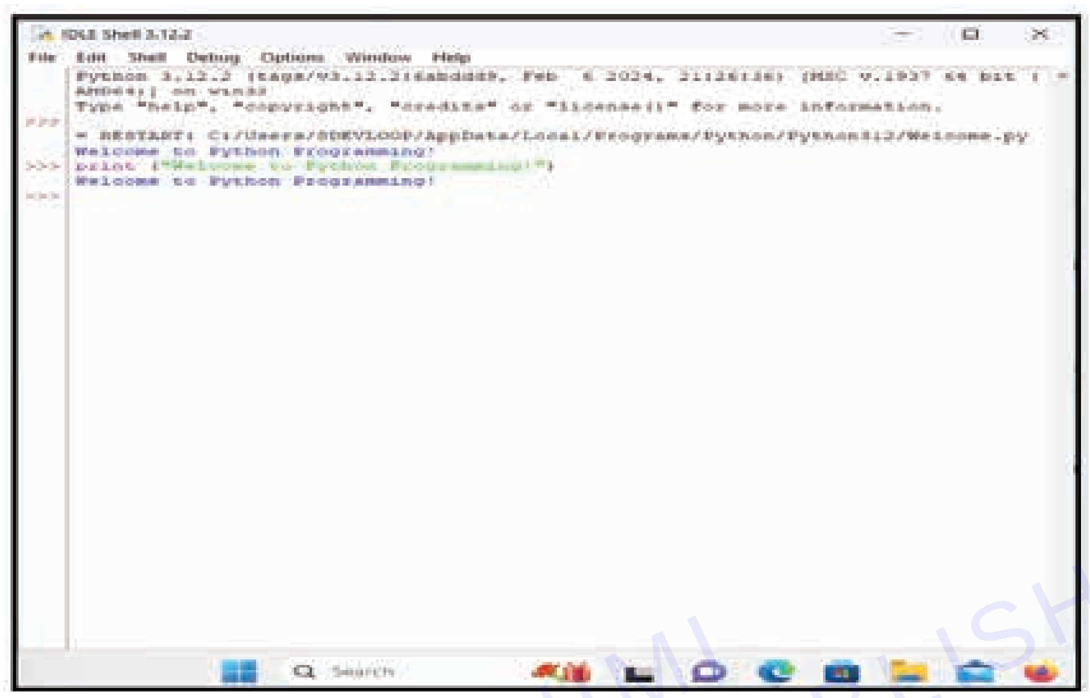
- If your script includes print statements or generates output, you can view the results in the Python Shell window, which typically opens automatically when you run the script.
- If the Shell window is not open, you can open it by selecting "Shell" from the "View" menu.



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/SDEVLOOP/AppData/Local/Programs/Python/Python312/Welcome.py
Welcome to Python Programming!
>>>
```

7 Interactive Mode:

- You can also run Python code interactively in the IDLE shell. Just type or paste the code directly into the shell, press Enter, and see the results immediately.



```
    Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    = RESTART: C:/Users/SDEVLOOP/AppData/Local/Programs/Python/Python312/Welcome.py
    Welcome to Python Programming!
>>> print ("Welcome to Python Programming!")
    Welcome to Python Programming!
>>>
```

8 Debugging (Optional):

- IDLE provides a basic debugger that can help you identify and fix errors in your code. You can set breakpoints and step through your code to investigate issues.

9 Close IDLE:

- Once you have finished working with IDLE, you can close it like any other application

  Now, you have successfully installed Python, set up your environment, and run a simple Python program. You are ready to start coding in Python!

EXERCISE 122 : Write and test a python program to demonstrate print statement, comments, and different types of variables

Objectives

At the end of this exercise you shall be able to

- develop python program to demonstrate print statement, comments, and different types of variables.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

Procedure

TASK 1: **String Variables**

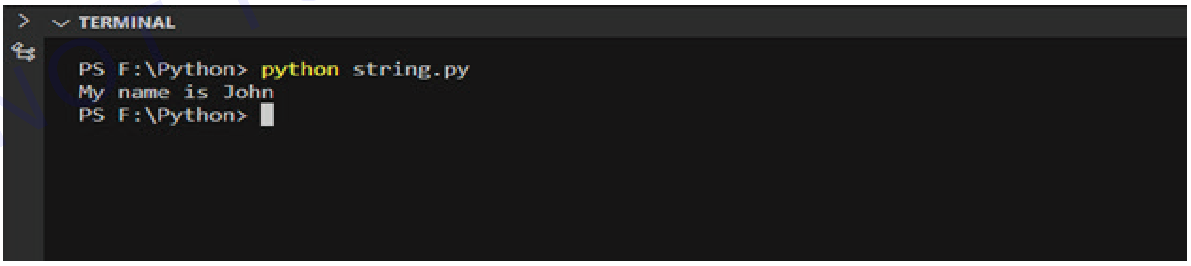
Code:

```
# Example 1: String Variables with Comment
name = "John"
print("My name is", name)
```

Explanation:

- name = "John": This line declares a variable named name and assigns the string value "John" to it. In Python, you can create a string by enclosing characters in single or double quotes.
- print("My name is", name): The print statement is used to display output. In this case, it prints the string "My name is" followed by the value stored in the name variable. Multiple values can be printed in a single print statement, and they are separated by commas.

Output:



```
PS F:\Python> python string.py
My name is John
PS F:\Python> 
```

— — — — —

TASK 2: **Numeric Variables**

Code:

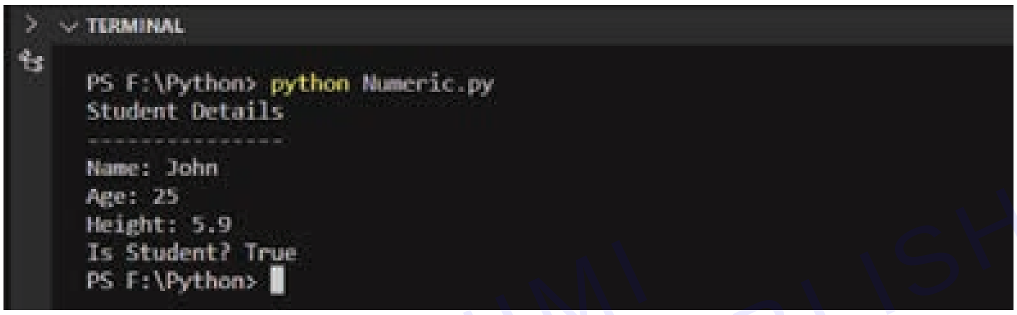
```
# Example 2: Variables and Data Types
# Declare variables with different data types
name = "John"    # String
age = 25         # Integer
height = 5.9     # Float
is_student = True  # Boolean
```

```
# Print variables
print("Student Details")
print("---------------")
print("Name:", name)
print("Age:", age)
print("Height:", height)
print("Is Student?", is_student)
```

Explanation:

- Variables (name, age, height, is_student) are declared with different data types (String, Integer, Float, Boolean).
- The print statements display the values of these variables.

Output:



```
TERMINAL
PS F:\Python> python Numeric.py
Student Details
---------------
Name: John
Age: 25
Height: 5.9
Is Student? True
PS F:\Python>
```

— — — — —

TASK 3: **Multiple Assignments**

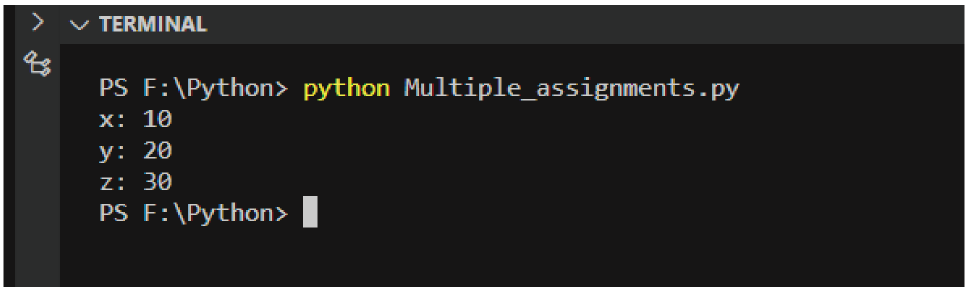
Code:

```
# Example 3: Multiple Assignments
# Multiple assignments in a single line
x, y, z = 10, 20, 30
# Print the variables
print("x:", x)
print("y:", y)
print("z:", z)
```

Explanation:

- Multiple variables (x, y, z) are assigned values in a single line.
- The print statements display the values of these variables.

Output:



```
TERMINAL
PS F:\Python> python Multiple_assignments.py
x: 10
y: 20
z: 30
PS F:\Python>
```

TASK 4: **String Concatenation**

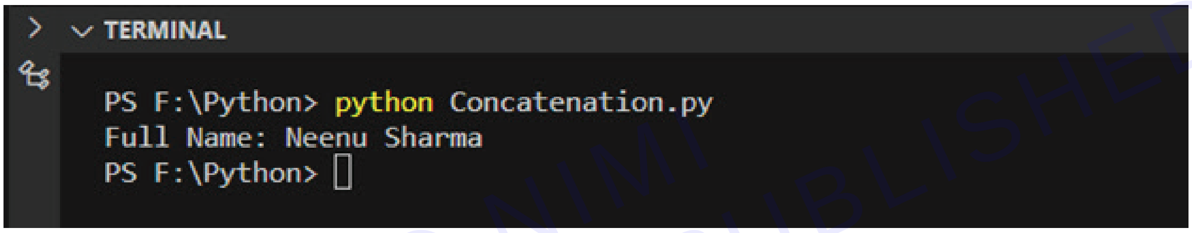
Code:

```
# Example 4: String Concatenation
# String variables
first_name = “Neenu”
last_name = “Sharma”
# Concatenate strings
full_name = first_name + “ “ + last_name
# Print the full name
print(“Full Name:”, full_name)
```

Explanation:

- String variables (first_name and last_name) are concatenated using the + operator.
- The print statement displays the full name.

Output:



```
PS F:\Python> python Concatenation.py
Full Name: Neenu Sharma
PS F:\Python> 
```

TASK 5: **Formatted String**

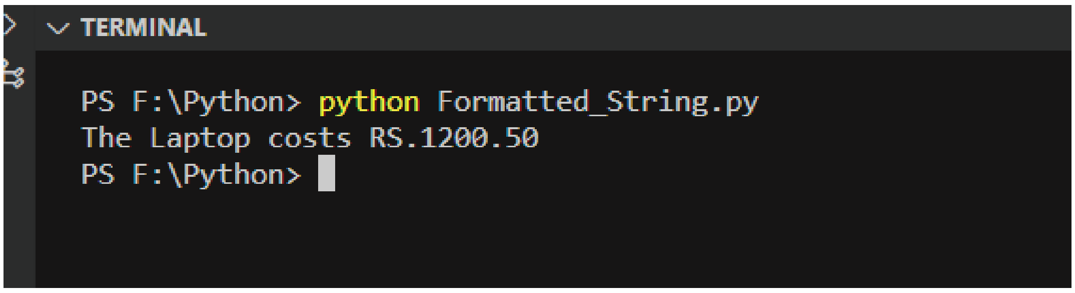
Code:

```
# Example 5: Formatted String
# Format string with variables
item = “Laptop”
price = 1200.50
# Print formatted string
print(f”The {item} costs RS.{price:.2f}”)
```

Explanation:

- A formatted string is created using an f-string, allowing variables (item and price) to be embedded directly into the string.
- The print statement displays the formatted string, including the price formatted with two decimal places.

Output:



```
PS F:\Python> python Formatted_String.py
The Laptop costs RS.1200.50
PS F:\Python> 
```

Related Exercises:

Here are the questions related to writing and testing a Python program to demonstrate print statements, comments, and different types of variables:

1 Print Statement: Write a Python program to display the message "Welcome to CSA_CITS!" using the print statement.

2 Comments: Create a Python program with both single-line and inline comments. Provide comments explaining the purpose of the code.

3 Integer Variable: Write a Python program that declares an integer variable (mark) and prints a message including the mark.

4 Float Variable: Develop a Python program with a float variable (side) representing the side of a square. Print a message displaying the side.

5 String Variable: Create a Python program declaring a stringvariable (city_name). Print a message including the city_name.(Eg. You are from Delhi).

EXERCISE 123 : Write and test a python program to perform data and data type operations, string operations, date, input and output, output formatting and operators

Objectives

At the end of this exercise you shall be able to

- develop python program to perform data and data type operations, string operations, date, input and output, output formatting and operators

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

Procedure

TASK 1**: Arithmetic Operators**

Code:

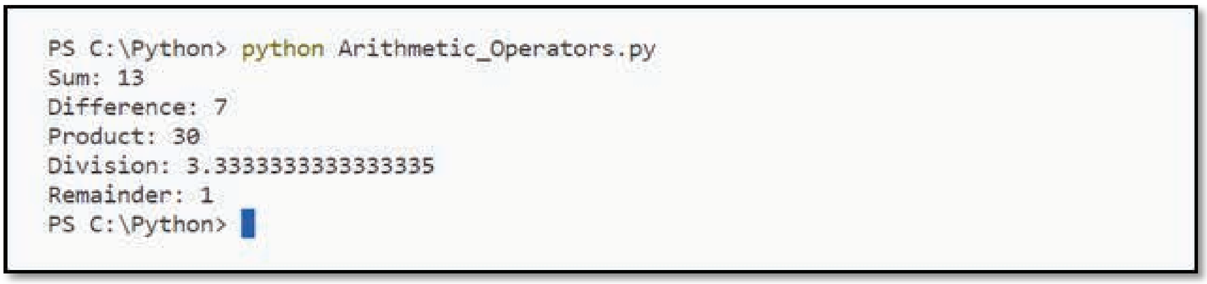
```
# Arithmetic Operators
num1 = 10
num2 = 3124
sum_result = num1 + num2
difference_result = num1 - num2
product_result = num1 * num2
division_result = num1 / num2
remainder_result = num1 % num2
print(f"Sum: {sum_result}")
print(f"Difference: {difference_result}")
print(f"Product: {product_result}")
print(f"Division: {division_result}")
print(f"Remainder: {remainder_result}")
```

Explanation:

- Addition (+): Adds num1 and num2 together, resulting in 13.
- Subtraction (-): Subtracts num2 from num1, resulting in 7.
- Multiplication (*): Multiplies num1 and num2, resulting in 30.
- Division (/): Divides num1 by num2, resulting in approximately 3.33333.
- Modulus (%): Computes the remainder when num1 is divided by num2, resulting in 1.

These are basic arithmetic operations showcasing how Python handles addition, subtraction, multiplication, division, and modulus operations. The results are then printed for each operation.

Output:



```
PS C:\Python> python Arithmetic_Operators.py
Sum: 13
Difference: 7
Product: 30
Division: 3.3333333333333335
Remainder: 1
PS C:\Python>
```

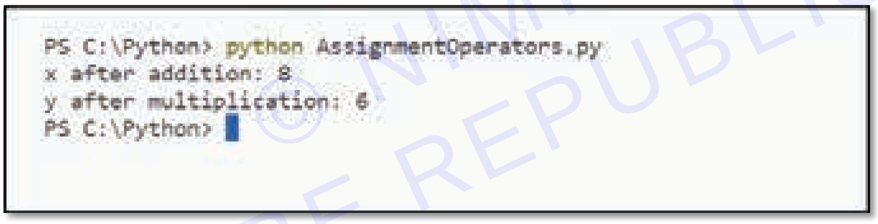
— — — — —

TASK 2**: Assignment Operators**

Code:

```
# Assignment Operators
x = 5
y = 3
x += y
print(f"x after addition: {x}")
y *= 2
print(f"y after multiplication: {y}")
```

Output:



```
PS C:\Python> python AssignmentOperators.py
x after addition: 8
y after multiplication: 6
PS C:\Python>
```

Explanation:

- Addition Assignment (+=): Adds the value of y to the current value of x and assigns the result back to x. In this case, x becomes 8 (original x value of 5 plus y value of 3).
- Multiplication Assignment (*=): Multiplies the value of y by 2 and assigns the result back to y. In this case, y becomes 6 (original y value of 3 multiplied by 2).
- These assignment operators provide a concise way to update the values of variables based on their current values. The results after each assignment operation are then printed for verification.

— — — — —

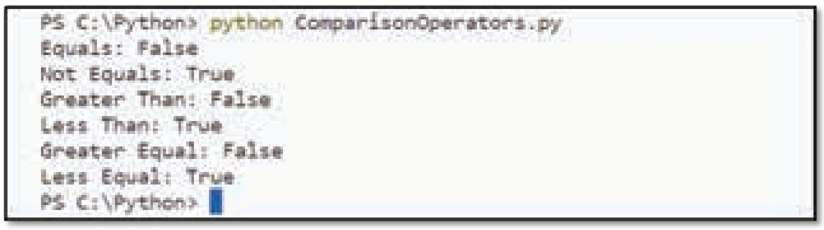
TASK 3 **: Comparison Operators**

Code:

```
# Comparison Operators
x = 10
y = 20
equals = x == y  # Equal to
not_equals = x !=y  # Not equal to
greater_than = x >y  # Greater than
less_than = x <y  # Less than
greater_equal = x >= y  # Greater than or equal to
```

```
less_equal = x <= y  # Less than or equal to
# Print each result on a separate line
print(f"Equals: {equals}")
print(f"Not Equals: {not_equals}")
print(f"Greater Than: {greater_than}")
print(f"Less Than: {less_than}")
print(f"Greater Equal: {greater_equal}")
print(f"Less Equal: {less_equal}")
```

Output:



```
PS C:\Python> python ComparisonOperators.py
Equals: False
Not Equals: True
Greater Than: False
Less Than: True
Greater Equal: False
Less Equal: True
PS C:\Python>
```

Explanation:

Here, we declare two variables x and y with values 10 and 20, respectively. Then, we use comparison operators to compare these values.

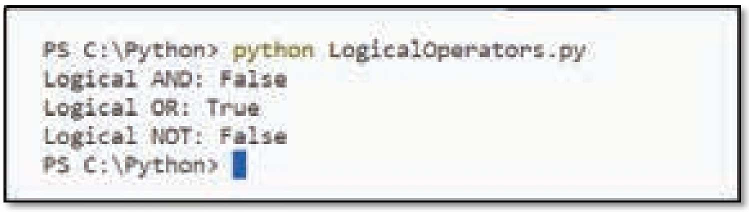
- equals: Checks if x is equal to y. If true, equals will be True, otherwise False.
- not_equals: Checks if x is not equal to y. If true, not_equals will be True, otherwise False.
- greater_than: Checks if x is greater than y. If true, greater_than will be True, otherwise False.
- less_than: Checks if x is less than y. If true, less_than will be True, otherwise False.
- greater_equal: Checks if x is greater than or equal to y. If true, greater_equal will be True, otherwise False.
- less_equal: Checks if x is less than or equal to y. If true, less_equal will be True, otherwise False.

TASK 4**: Logical Operators**

Code:

```
# Logical Operators
a = True
b = False
logical_and = a and b  # Logical AND
logical_or = a or b  # Logical OR
logical_not = not a  # Logical NOT
# Print each result on a separate line
print(f"Logical AND: {logical_and}")
print(f"Logical OR: {logical_or}")
print(f"Logical NOT: {logical_not}")
```

Output:



```
PS C:\Python> python LogicalOperators.py
Logical AND: False
Logical OR: True
Logical NOT: False
PS C:\Python>
```

Explanation:

- a is assigned the boolean value True.
- b is assigned the boolean value False.

1 Logical AND (and) Operator:
   - logical_and is assigned the result of the logical AND operation between a and b.
   - If both a and b are True, then logical_and will be True; otherwise, it will be False.

2. Logical OR (or) Operator:
   - logical_or is assigned the result of the logical OR operation between a and b.
   - If at least one of a or b is True, then logical_or will be True; otherwise, it will be False.

3. Logical NOT (not) Operator:
   - logical_not is assigned the result of the logical NOT operation on a.
   - If a is True, then logical_not will be False; if a is False, then logical_not will be True.

— — — — —

TASK 5**: Working with dates**

Code:

```
# Example 3: Working with dates
fromdatetime import datetime, timedelta
current_date = datetime.now()
future_date = current_date + timedelta(days=7)
print("Current Date:", current_date)
print("Future Date:", future_date)
```

Explanation:

In this example, we are working with dates using the datetime module in Python. Here's a breakdown of the code:

1 Fromdatetime import datetime, timedelta: This line imports the datetime class and the timedelta class from the datetime module. The datetime class is used for working with dates and times, and timedelta represents the difference between two dates or times.

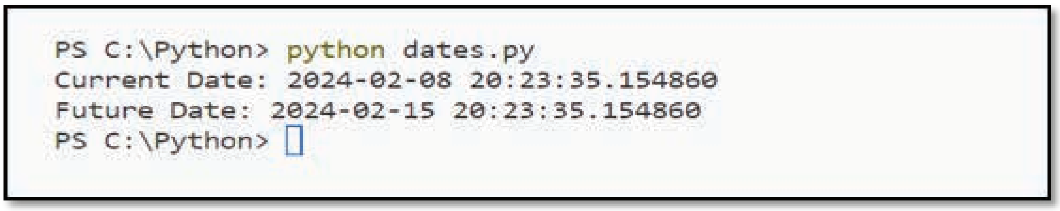
2 Current_date = datetime.now(): This line creates a datetime object representing the current date and time. The datetime.now() function returns the current date and time.

3 Future_date = current_date + timedelta(days=7): This line calculates a future date by adding a timedelta of 7 days to the current date. The timedelta(days=7) represents a duration of 7 days.

4 Print("Current Date:", current_date): This line prints the current date and time.

5 Print("Future Date:", future_date): This line prints the calculated future date, which is 7 days ahead of the current date.

Output:



```
PS C:\Python> python dates.py
Current Date: 2024-02-08 20:23:35.154860
Future Date: 2024-02-15 20:23:35.154860
PS C:\Python>
```

— — — — —

TASK 6**: Input and Output**

Code:

# Example 4: User input and output

user_input = input(“Enter your name: “)

print(“Hello, “ + user_input + “!”)

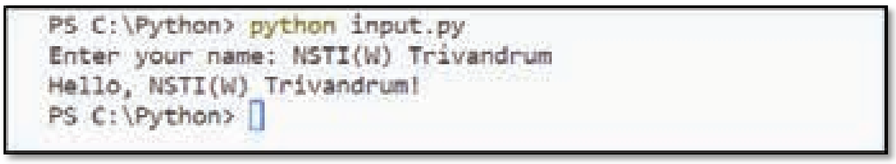
Explanation:

In this example, we demonstrate how to take user input and display output based on that input.

1 user_input = input(“Enter your name: “): This line prompts the user to enter their name. The input() function is used to receive input from the user, and the provided message (“Enter your name: “) serves as a prompt.

2 print(“Hello, “ + user_input + “!”): This line prints a greeting using the user’s input. The + operator is used to concatenate the strings, and the exclamation mark is included for emphasis.

Output:



```
PS C:\Python> python input.py
Enter your name: NSTI(W) Trivandrum
Hello, NSTI(W) Trivandrum!
PS C:\Python> 
```

— — — — —

TASK 7**: Output Formatting and Operators**

Code:

# Example 5: Output formatting and comparison operators

x = 10

y = 15

print(f”Is {x} equal to {y}? {x == y}”)

print(f”Is {x} not equal to {y}? {x != y}”)

Explanation:

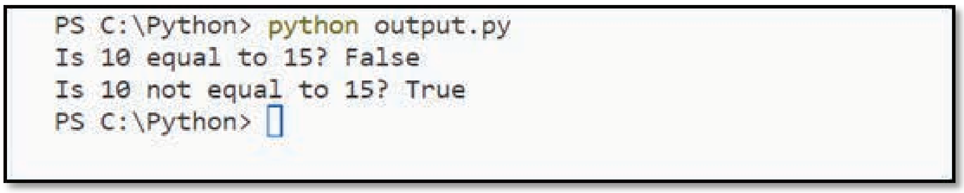
This example demonstrates output formatting and the use of comparison operators.

1 x = 10 and y = 15: These lines assign values to the variables x and y.

2 print(f”Is {x} equal to {y}? {x == y}”): This line uses an f-string to format the output. It checks whether x is equal to y using the equality operator (==) and prints the result.

3 print(f”Is {x} not equal to {y}? {x != y}”): Similarly, this line checks whether x is not equal to y using the inequality operator (!=) and prints the result.

Output:



```
PS C:\Python> python output.py
Is 10 equal to 15? False
Is 10 not equal to 15? True
PS C:\Python> 
```

Related Exercises

1 **Data Type Operations:**

- **Question:** Create a Python program that performs operations on different data types (int, float, string). Include arithmetic operations and display the results.

2 **String Operations:**

- **Question:** Write a Python program that manipulates strings. Perform operations like concatenation, slicing, and converting the case of the string. Display the modified strings.

3 **Date Operations:**

- **Question:** Develop a Python program that works with dates. Perform operations such as getting the current date, adding a specified number of days to the current date, and displaying the results.

4 **Input and Output:**

- **Question:** Create a Python program that takes user input, processes it, and displays the result. Include appropriate prompts for the user and handle different data types.

5 **Output Formatting and Operators:**

- **Question:** Write a Python program that uses output formatting and demonstrates the use of comparison operators. Include examples that format output using f-strings and perform comparisons between variables.

EXERCISE 124 : Determine the sequence of execution based on operator precedence

Objectives

At the end of this exercise you shall be able to

- develop python program to determine the sequence of execution based on operator precedence

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

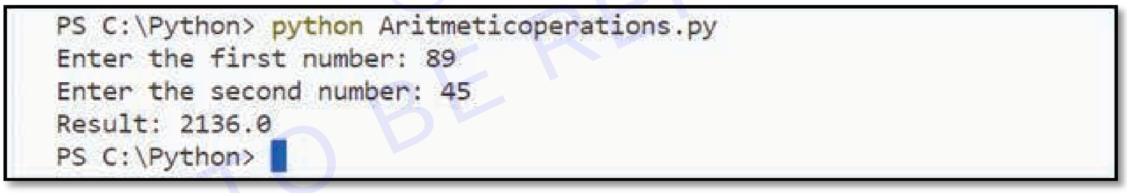
Procedure

TASK 1**: Arithmetic Operations**

Code:

```
# Program to demonstrate arithmetic operations based on precedence
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
result = num1 * (num2 + 3) / 2
print("Result:", result)
```

Output:



```
PS C:\Python> python Aritmeticoperations.py
Enter the first number: 89
Enter the second number: 45
Result: 2136.0
PS C:\Python>
```

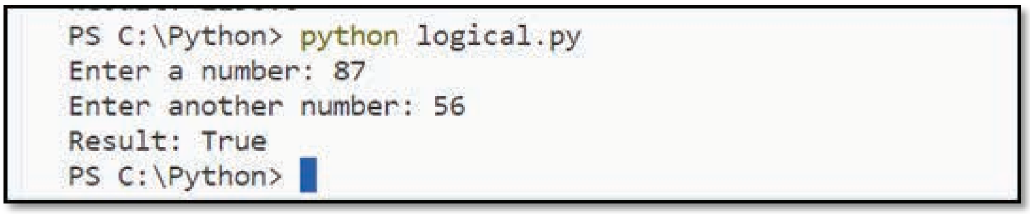
— — — — —

TASK 2**: Comparison and Logical Operators**

Code:

```
# Program to demonstrate comparison and logical operators based on precedence
x = int(input("Enter a number: "))
y = int(input("Enter another number: "))
result = x > 0 and (y % 2 == 0 or y > 10)
print("Result:", result)
```

Output:



```
PS C:\Python> python logical.py
Enter a number: 87
Enter another number: 56
Result: True
PS C:\Python>
```

— — — — —

TASK 3**: Bitwise Operations**

Code:

```
# Program to demonstrate bitwise operations based on precedence
a = int(input("Enter an integer: "))
b = int(input("Enter another integer: "))
result = (a & b) | (a ^ b)
print("Result:", result)
```

Explanation:

1 **User Input:**

- The program prompts the user to enter two integers (aandb).

2 **Bitwise Operations:**

- (a& b): Bitwise AND operation on a and b.
- (a ^ b): Bitwise XOR (exclusive OR) operation on a and b.
- (a & b) | (a ^ b): Bitwise OR operation on the results of the AND and XOR operations.

3 **Result Display:**

- The final result of the bitwise operations is stored in the variable result.
- The program prints the result to the console using print("Result:", result).

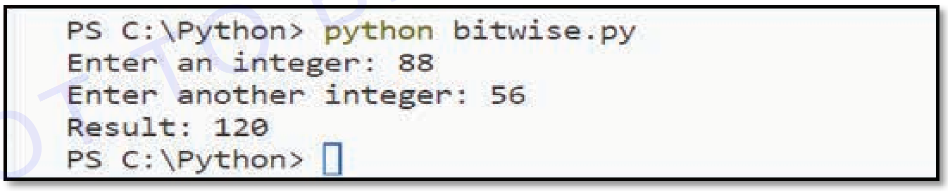
4 **Operator Precedence:**

- The & (AND) and ^ (XOR) operations have higher precedence than | (OR), so they are performed first.

5 **Observation:**

- Users can input integer values, and the program will demonstrate the bitwise AND, XOR, and OR operations based on the precedence of these bitwise operators.

Output:



```
PS C:\Python> python bitwise.py
Enter an integer: 88
Enter another integer: 56
Result: 120
PS C:\Python> 
```

— — — — —

TASK 4**: String Concatenation and Repetition**

Code:

```
# Program to demonstrate string concatenation and repetition based on precedence
str1 = input("Enter the first string: ")
str2 = input("Enter the second string: ")
result = str1 + " is " * 3 + str2
print("Result:", result)
```

Explanation:

1 **User Input:**

- The program prompts the user to enter two strings (str1 and str2).

2 **String Concatenation and Repetition:**

- str1 + " is " * 3 + str2: String concatenation of str1, the repeated string " is " three times, and str2.

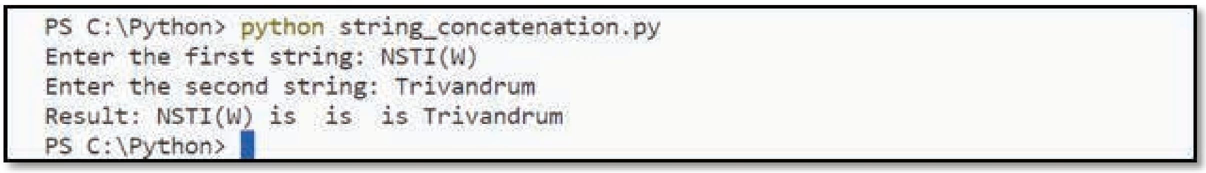
3 **Result Display:**

- The final result of the string concatenation and repetition is stored in the variable result.
- The program prints the result to the console using print("Result:", result).

4 **Observation:**

- Users can input any strings, and the program will demonstrate the concatenation of strings and the repetition of the middle string based on the precedence of string concatenation and repetition.

Output:



```
PS C:\Python> python string_concatenation.py
Enter the first string: NSTI(W)
Enter the second string: Trivandrum
Result: NSTI(W) is  is  is Trivandrum
PS C:\Python>
```

Related Exercises:

1 **String Operations:**

- Question 1: Write a Python program to concatenate two strings and print the result.

2 **Date Operations:**

- Question 2: Create a Python program that prints the current date and the date exactly one week from now.

3 **User Input and Output:**

- Question 3: Develop a Python program that takes user input for their name and greets them using the input.

4 **Output Formatting:**

- Question 4: Write a Python program that compares two numbers and prints the result in a formatted string.

5 **Bitwise Operations:**

- Question 5: Implement a Python program that takes two integers as input, performs bitwise AND and OR operations, and prints the result.

EXERCISE 125 : Construct and analyze code segments that use branching statements

Objectives

At the end of this exercise you shall be able to

- develop python program to Construct and analyze code segments that use branching statements.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

Procedure

Indentation

In Python, indentation is the whitespace (typically spaces or tabs) at the beginning of a line that determines the grouping of statements. Unlike many other programming languages that use curly braces {} or keywords like begin and end to indicate code blocks, Python relies on indentation for this purpose.

Indentation is not just for readability; it is a syntactical element in Python. Blocks of code with the same level of indentation are considered part of the same block or suite. Indentation is used to define the scope of control flow structures, such as loops, conditionals, functions, and classes.

For example, in the following Python code:

```
if x > 0:
print("x is positive")
    y = x * 2
print("Double of x is:", y)
```

The statements print("x is positive") and y = x * 2 are indented under the if x > 0: statement, indicating that they are part of the same block and will be executed only if the condition is true.

It's essential to maintain consistent indentation throughout your code to avoid syntax errors and to ensure that the structure of your code is correctly interpreted by the Python interpreter. Typically, four spaces are used for each level of indentation, although you can also use tabs or a different number of spaces as long as it is consistent within the same block.

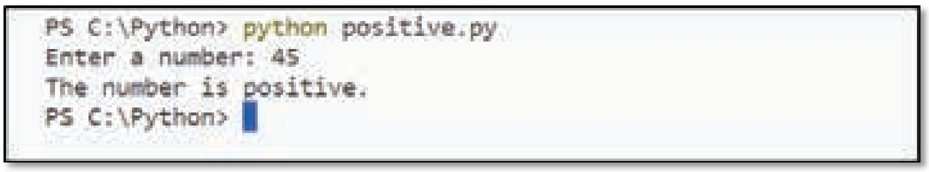
TASK 1**: If Statement**

Code:

```
# Program to check if a number is positive
num = int(input("Enter a number: "))
ifnum> 0:
print("The number is positive.")
```

In this example, if the entered number is greater than 0, it prints a message indicating that the number is positive.

Output:



```
PS C:\Python> python positive.py
Enter a number: 45
The number is positive.
PS C:\Python>
```

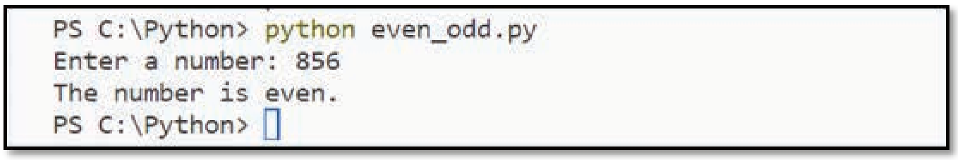
TASK 2**: If-Else Statement**

Code:

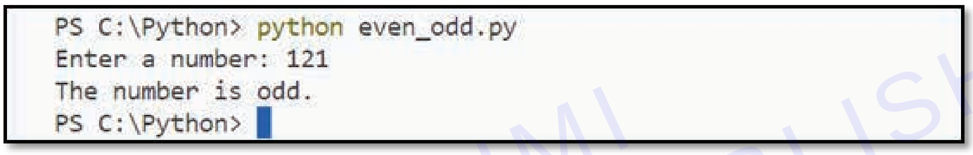
```
# Program to check if a number is even or odd
num = int(input("Enter a number: "))
ifnum % 2 == 0:
print("The number is even.")
else:
print("The number is odd.")
```

Here, the program checks if the entered number is even or odd. If the remainder when divided by 2 is 0, it prints a message indicating that the number is even. Otherwise, it prints a message indicating that the number is odd.

Output:



```
PS C:\Python> python even_odd.py
Enter a number: 856
The number is even.
PS C:\Python>
```



```
PS C:\Python> python even_odd.py
Enter a number: 121
The number is odd.
PS C:\Python>
```

TASK 3**: Nested if Statements**

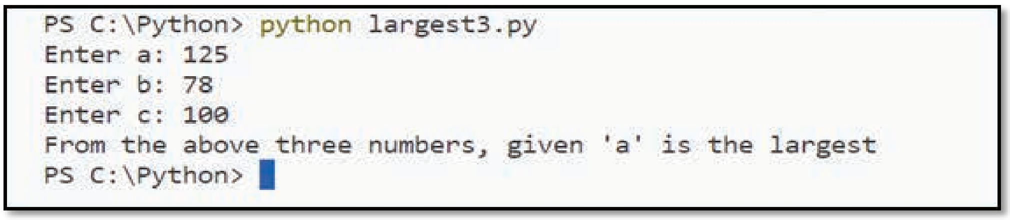
Code:

```
# Simple Python Program to print the largest of the three numbers.
# Taking user input for three numbers
a = int(input("Enter a: "))
b = int(input("Enter b: "))
c = int(input("Enter c: "))
# Checking if 'a' is the largest
if a > b and a > c:
    # If the condition is true, we will enter this block
    print("From the above three numbers, given 'a' is the largest")
# Checking if 'b' is the largest
if b > a and b > c:
    # If the condition is true, we will enter this block
    print("From the above three numbers, given 'b' is the largest")
# Checking if 'c' is the largest
if c > a and c > b:
    # If the condition is true, we will enter this block
    print("From the above three numbers, given 'c' is the largest")
```

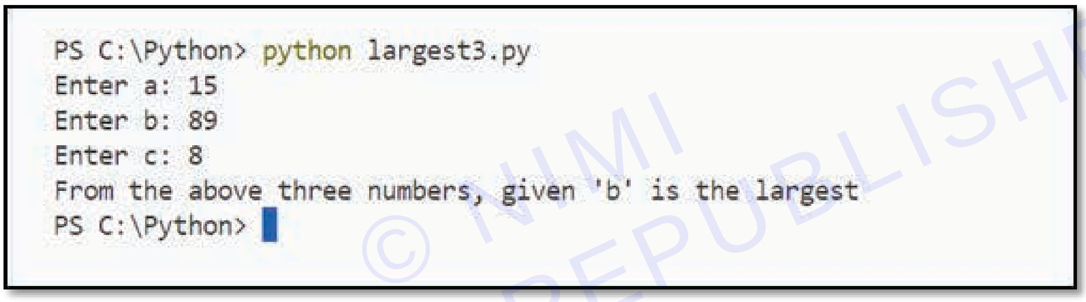
Explanation:

- a = int(input("Enter a: ")): Takes user input for the first number 'a' and converts it to an integer.
- b = int(input("Enter b: ")): Takes user input for the second number 'b' and converts it to an integer.
- c = int(input("Enter c: ")): Takes user input for the third number 'c' and converts it to an integer.
- The code uses a series of if statements to check which number among 'a', 'b', and 'c' is the largest.
- if a > b and a > c:: Checks if 'a' is greater than both 'b' and 'c'.
- print("From the above three numbers, given 'a' is the largest"): Prints a message if 'a' is the largest.
- Similar if statements and messages are present for 'b' and 'c'.

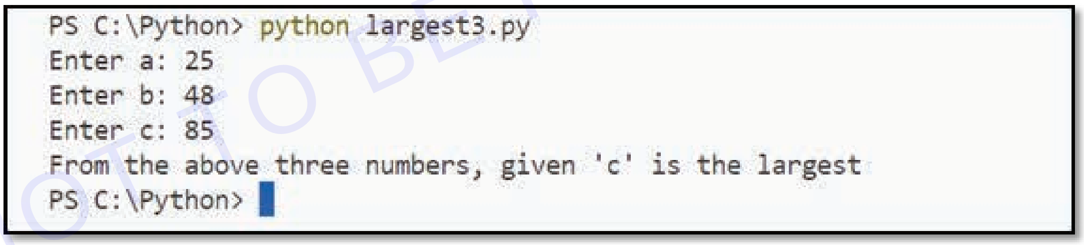
Output:



```
PS C:\Python> python largest3.py
Enter a: 125
Enter b: 78
Enter c: 100
From the above three numbers, given 'a' is the largest
PS C:\Python>
```



```
PS C:\Python> python largest3.py
Enter a: 15
Enter b: 89
Enter c: 8
From the above three numbers, given 'b' is the largest
PS C:\Python>
```



```
PS C:\Python> python largest3.py
Enter a: 25
Enter b: 48
Enter c: 85
From the above three numbers, given 'c' is the largest
PS C:\Python>
```

— — — — —

TASK 4**: Python program to understand the elif statement**

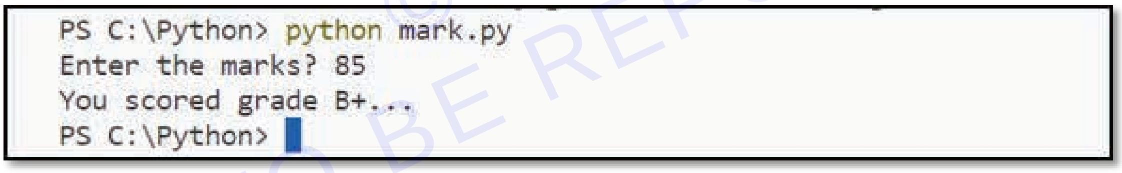
Code:

```
# Simple Python program to understand the elif statement
# Taking an integer input for marks dynamically
marks = int(input("Enter the marks? "))
# Checking various conditions using if-elif-else statements
if marks > 85 and marks <= 100:
    # If the condition is true, we will enter this block
    print("Congrats! You scored grade A...")
elif marks > 60 and marks <= 85:
    # If the condition is true, we will enter this block
    print("You scored grade B+...")
elif marks > 40 and marks <= 60:
    # If the condition is true, we will enter this block
    print("You scored grade B...")
elif marks > 30 and marks <= 40:
    # If the condition is true, we will enter this block
    print("You scored grade C...")
else:
    # If none of the above conditions are true, we will enter this block
    print("Sorry, you have failed.")
```

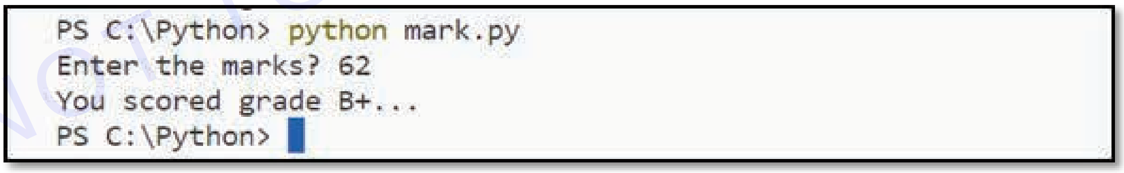
Explanation:

- marks = int(input("Enter the marks? ")): Takes user input for marks and converts it to an integer.
- The code uses if-elif-else statements to check different conditions based on the value of 'marks'.
- Conditions are checked in the order they appear, and the first true condition's block is executed.
- If marks are greater than 85 and less than or equal to 100, it prints "Congrats! You scored grade A..."
- If not, it moves to the next elif and checks if marks are greater than 60 and less than or equal to 85.
- The process continues for the other elif conditions.
- If none of the conditions are true, the else block is executed, and it prints "Sorry, you have failed."

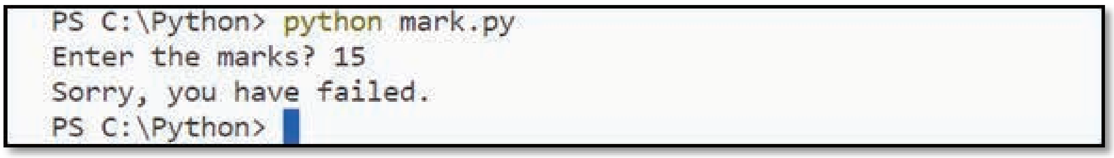
Output:



```
PS C:\Python> python mark.py
Enter the marks? 85
You scored grade B+...
PS C:\Python>
```



```
PS C:\Python> python mark.py
Enter the marks? 62
You scored grade B+...
PS C:\Python>
```



```
PS C:\Python> python mark.py
Enter the marks? 15
Sorry, you have failed.
PS C:\Python>
```

— — — — —

TASK 5**: Match Case**

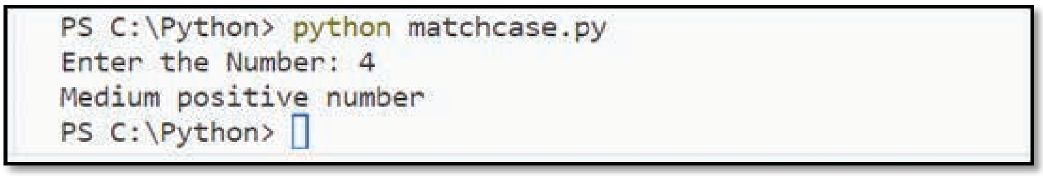
Code:

```
num=int(input("Enter the Number: "))
defdescribe_number(num):
    matchnum:
        case 0:
            return "Zero"
        case 1 | 2:
            return "Small positive number"
        case 3 | 4 | 5:
            return "Medium positive number"
        case _ if num> 5:
            return "Large positive number"
        case _:
            return "Negative number or other cases"
# Example usage
result = describe_number(num)
print(result)
```

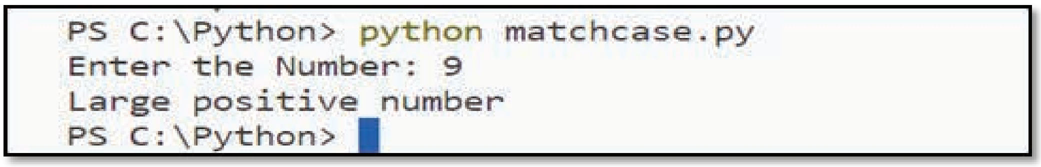
Explanation:

- num = int(input("Enter the Number: ")): This line prompts the user to enter a number, converts the input to an integer (int), and stores it in the variable num.
- defdescribe_number(num):: This line defines a function named describe_number that takes a single argument num.
- matchnum:: This is the beginning of a match statement, a new feature introduced in Python 3.10. It allows you to perform pattern matching on the value of num.
- case 0:: This line checks if num is equal to 0.
- return "Zero": If the value of num is 0, the function returns the string "Zero".
- case 1 | 2:: This line checks if num is either 1 or 2.
- return "Small positive number": If the value of num is 1 or 2, the function returns the string "Small positive number".
- case 3 | 4 | 5:: This line checks if num is either 3, 4, or 5.
- return "Medium positive number": If the value of num is 3, 4, or 5, the function returns the string "Medium positive number".
- case _ if num> 5:: This line is a wildcard case that matches any value greater than 5.
- return "Large positive number": If the value of num is greater than 5, the function returns the string "Large positive number".
- case _:: This line is another wildcard case that matches any other value.
- return "Negative number or other cases": If the value of num doesn't match any of the specific cases mentioned earlier, the function returns the string "Negative number or other cases".
- result = describe_number(num): This line calls the describe_number function with the user-inputted value num and stores the result in the variable result.
- print(result): Finally, this line prints the result, which is the description of the entered number based on the defined cases in the describe_number function.

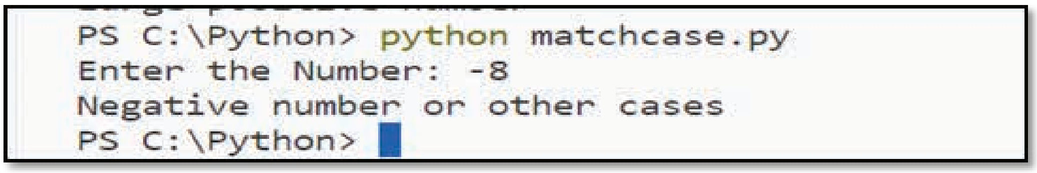
Output:



```
PS C:\Python> python matchcase.py
Enter the Number: 4
Medium positive number
PS C:\Python> 
```



```
PS C:\Python> python matchcase.py
Enter the Number: 9
Large positive number
PS C:\Python>
```



```
PS C:\Python> python matchcase.py
Enter the Number: -8
Negative number or other cases
PS C:\Python>
```

Related Exercises:

1 Check if a number is even or odd.

2 Determine if a given year is a leap year.

3 Find the maximum of three numbers.

4 Determine if a student has passed or failed based on the percentage obtained.

5 Check if a character is a vowel or a consonant.

6 Determine the type of a triangle based on its sides.(Scalene/ Isosceles/ Equilateral)

7 Calculate the factorial of a number.

8 Determine if a number is prime.

9 Check if a given string is a palindrome.

10 Convert a numerical grade (input the percentage of mark) to a letter grade.

EXERCISE 126 : Construct and analyze code segments that perform iteration

Objectives

At the end of this exercise you shall be able to

- develop python program to Construct and analyze code segments that perform iteration.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

Procedure

Python While loop

TASK 1**: Python Program for printing numbers from 1 to n**

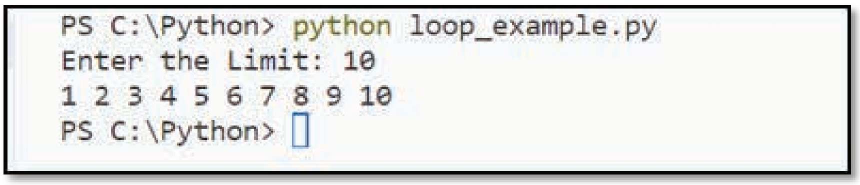
Code:

```
limit=int(input("Enter the Limit: "))
# Initialize the variable i with the value 1
i = 1
# The loop will continue as long as i is less than or equal to limit
whilei<= limit:
# Print the current value of i, followed by a space instead of a newline
    print(i, end=' ')
# Increment the value of i by 1 in each iteration
    i += 1
```

Explanation:

- i is initialized to 1.
- The while loop continues as long as the condition i<= limit is true.
- Inside the loop, print(i, end=' ') prints the current value of i followed by a space, without moving to the next line.
- i += 1 increments the value of i by 1 in each iteration of the loop.

Output:



```
PS C:\Python> python loop_example.py
Enter the Limit: 10
1 2 3 4 5 6 7 8 9 10
PS C:\Python> 
```

— — — — —

TASK 2**: While loops in Python for Printing those numbers divisible by either 5 or 7 within 1 to limit using a while loop**

Code:

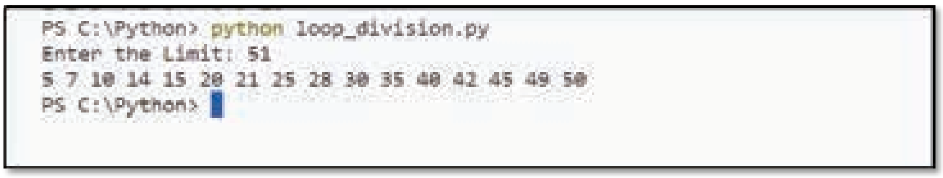
```
# Take user input to set the limit for the loop

limit = int(input("Enter the Limit: "))

# Initialize a variable i with the value 1

i = 1

# The loop will continue as long as i is less than the specified limit

whilei< limit:

# Check if the current value of i is divisible by 5 or 7

    ifi % 5 == 0 or i % 7 == 0:

# Print the current value of i, followed by a space instead of a newline

print(i, end=' ')

# Increment the value of i by 1 in each iteration

i+=1
```

Explanation:

- limit = int(input("Enter the Limit: ")): Takes user input to set the upper limit for the loop. The int() function is used to convert the input to an integer.
- i = 1: Initializes a variable i with the value 1.
- whilei< limit:: The while loop continues as long as i is less than the specified limit.
- if i % 5 == 0 or i % 7 == 0:: Checks whether the current value of i is divisible by 5 or 7. The % operator calculates the remainder after division.
- print(i, end=' '): If the condition in the if statement is true, it prints the current value of i, followed by a space instead of a newline.
- i += 1: Increments the value of i by 1 in each iteration of the loop.

The loop iterates through the numbers from 1 to the specified limit, and for each number, it checks if it is divisible by 5 or 7. If the condition is true, it prints the number. The loop continues until i reaches the specified limit.

Output:



```
PS C:\Python> python loop_division.py
Enter the Limit: 51
5 7 10 14 15 20 21 25 28 30 35 40 42 45 49 50
PS C:\Python>
```

— — — — —

TASK 3**: Write a program for the sum of squares of the first n natural numbers using a while loop**

Code:

```
# Python program example to show the use of while loop

# Take user input to set the limit for the loop

n = int(input("Enter the limit: "))

# Initializing summation and a counter for iteration

summation = 0

c = 1
```

```
while c <= n:  # Specifying the condition of the loop
    # Beginning the code block
    summation = c**2 + summation
    c = c + 1  # Incrementing the counter
# Print the final sum
print("The sum of squares is", summation)
```

Explanation:

1 n = int(input("Enter the limit: ")): Takes user input to set the upper limit for the loop. The int() function is used to convert the input to an integer.

2 summation = 0: Initializes a variable summation to 0. This variable will store the sum of squares.

3 c = 1: Initializes a counter variable c with the value 1.

4 while c <= n:: The while loop continues as long as c is less than or equal to the specified limit.

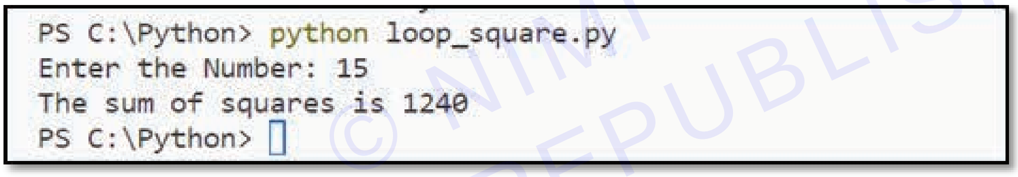
5 summation = c**2 + summation: Calculates the square of c and adds it to the current value of summation.

6 c = c + 1: Increments the value of c by 1 in each iteration of the loop.

7 After the loop, print("The sum of squares is", summation): Prints the final sum of squares.

This program is a straightforward example of using a while loop to iteratively calculate the sum of squares.

Output:



```
PS C:\Python> python loop_square.py
Enter the Number: 15
The sum of squares is 1240
PS C:\Python> 
```

Related Exercises:

Develop the following Python programs using while loop

1 Display Sum of digits of a number and its reverse

2 Print the cube of all numbers from 1 to a given number

3 Factorial of a number

4 Display the Fibonacci series upto 'n'

5 Write a program to display all prime numbers within a range

Python For Loop

TASK 1**: Python For Loop in Python String**

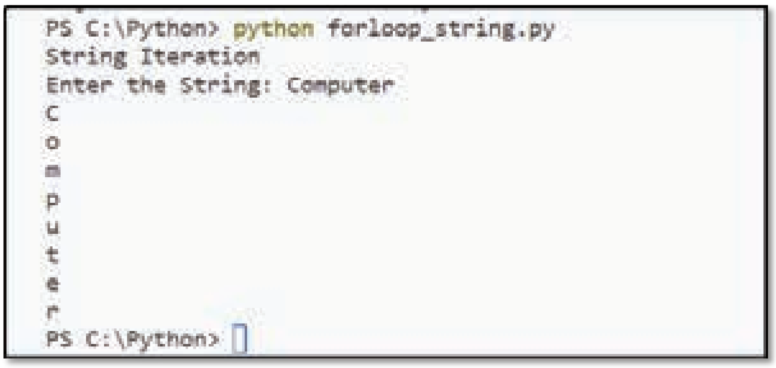
Code:

```
# Iterating over a String
print("String Iteration")
s = input("Enter the String: ")
foriin s:
    print(i)
```

Explanation:

This code uses a for loop to iterate over a string and print each character on a new line. The loop assigns each character to the variable i and continues until all characters in the string have been processed.

Output:



```
PS C:\Python> python forloop_string.py
String Iteration
Enter the String: Computer
C
o
m
p
u
t
e
r
PS C:\Python>
```

— — — — —

TASK 2**: Python For Loop with a step size**

Code:

limit=int(input("Enter the Limit: "))

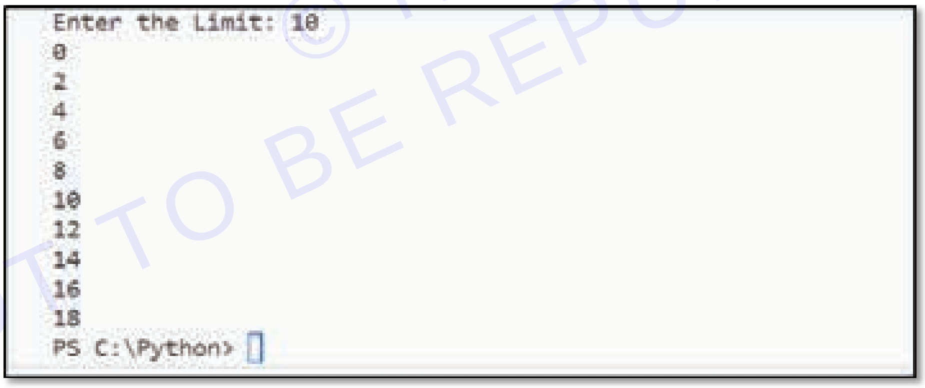
fori in range(0, 20, 2):

    print(i)

Explanation:

This code uses a for loop in conjunction with the range() function to generate a sequence of numbers starting from 0, up to (but not including) limit, and with a step size of 2. For each number in the sequence, the loop prints its value using the print() function.

Output:



```
Enter the Limit: 10
0
2
4
6
8
10
12
14
16
18
PS C:\Python>
```

— — — — —

TASK 3**: Python For Loop inside a For Loop**

Code:

limit=int(input("Enter the Limit: "))

fori in range(1, limit):

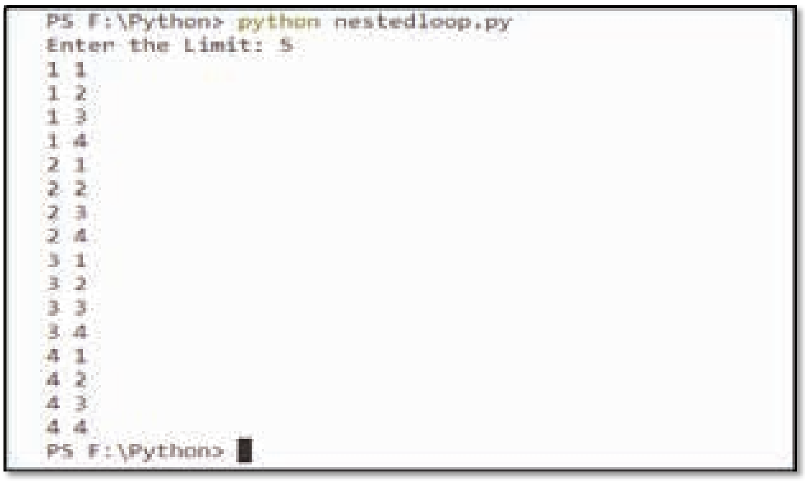
    for j in range(1, limit):

        print(i, j)

Explanation:

This code uses nested for loops to iterate over two ranges of numbers (1 to limit inclusive) and prints the value of i and j for each combination of the two loops. The inner loop is executed for each value of i in the outer loop.

Output:



```
PS F:\Python> python nestedloop.py
Enter the Limit: 5
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
4 1
4 2
4 3
4 4
PS F:\Python>
```

— — — — —

TASK 4 **: Right-angled pattern with characters**

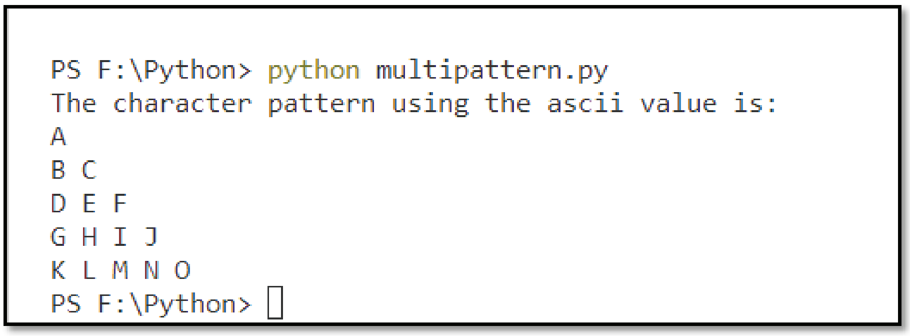
Code:

```
print(“The character pattern using the ascii value is: “)
asciiValue = 65     # here, we are giving the ASCII value of A
foriin range(0, 5):   # here, we are declaring the for loop for I values
    for j in range(0, i + 1):    # here, we are declaring the for loop for j values
        # Here, the below will convert the ASCII value to the character
        alphabate = chr(asciiValue)
        print(alphabate, end=’ ‘)     # Here, we are printing the alphabets
        asciiValue += 1
# Here, we are incrementing the asciivalue by 1 after each iteration
    print()
```

Explanation:

- The outer loop (for i in range(0, 5)) controls the number of rows in the pattern (5 rows in this case).
- The inner loop (for j in range(0, i + 1)) controls the number of characters in each row, and it increases with each iteration of the outer loop.
- chr(asciiValue) converts the current ASCII value to the corresponding character.
- print(alphabate, end=’ ‘) prints the character without moving to the next line.
- asciiValue += 1 increments the ASCII value for the next character in the sequence.
- print() moves to the next line for the next row in the pattern.

Output:



```
PS F:\Python> python multipattern.py
The character pattern using the ascii value is:
A
B C
D E F
G H I J
K L M N O
PS F:\Python>
```

Related Exercises:

Develop the following Python Programs using for loop|:

1 Display numbers from -10 to -1

2 Count the number of vowels in a string.

3 Print the following patterns.

```
* * * * *
* * * *
* * *
* *
*
```

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

4 Print numbers divisible by 3 or 5 from 1 to limit.

5 Count the total number of digits in a number.

EXERCISE 127 : Document code segments using comments and documentation strings

Objectives

At the end of this exercise you shall be able to

- develop python program to document code segments using comments and documentation strings.

Requirements

Tools/Materials

- PC/Laptop with Windows OS
- Latest Version of Python

Procedure

TASK 1**: Single-line comment**

Code:

```
# This is a single-line comment
variable = int(input("Enter the Value: "))  # Inline comment for variable assignment
# The next line prints the variable
print(variable)
```

Explanation:

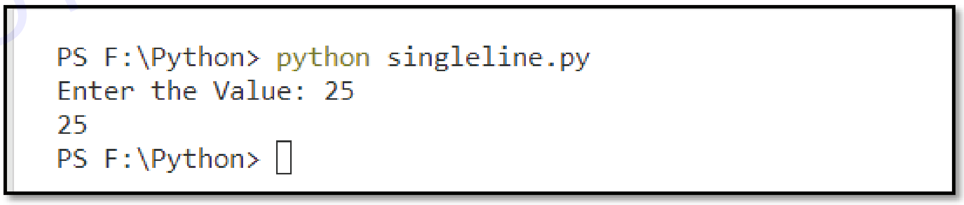
1 The first line is a single-line comment. Comments are ignored by the Python interpreter and are used to provide explanations or notes within the code.

2 The second line prompts the user to enter a value. The input() function takes user input as a string, and int() is used to convert it to an integer. The result is assigned to the variable.

3 The inline comment after the assignment explains that it is a comment related to the variable assignment.

4 The last line prints the value of the variable using the print() function.

Output:



```
PS F:\Python> python singleline.py
Enter the Value: 25
25
PS F:\Python> 
```

— — — — —

TASK 2**: Multi-line Comments**

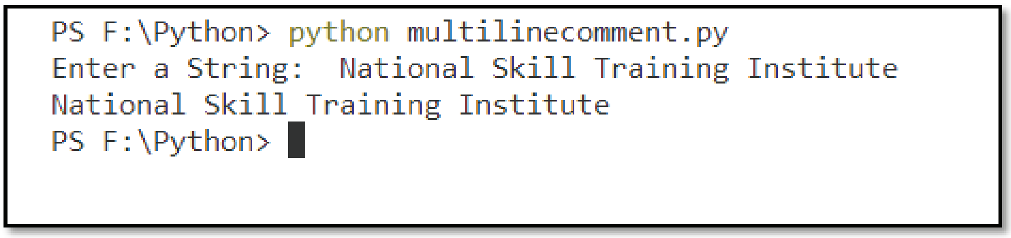
Code:

```
"""
This is a multi-line comment.
It can span multiple lines.
Use triple double-quotes or single-quotes.
"""
variable = input("Enter a String:  ")
print(variable)
```

Explanation:

- Multi-line comments are enclosed in triple double-quotes (“””).

Output:



```
PS F:\Python> python multilinecomment.py
Enter a String:  National Skill Training Institute
National Skill Training Institute
PS F:\Python>
```

— — — — —

TASK 3**: Documentation String (Docstring)**

Code:

```
defadd_numbers(a, b):
   “””
   This function adds two numbers.
   Args:
      a (int): The first number.
      b (int): The second number.
   Returns:
      int: The sum of the two numbers.
   “””
   result = a + b
   return result
# Example usage:
sum_result = add_numbers(5, 3)
print(“Sum:”, sum_result)
```

Explanation:

This code defines a function add_numbers that takes two integer parameters (a andb) and returns their sum. It includes a docstring that provides information about the function.

1 defadd_numbers(a, b)::

- The def keyword is used to define a function.
- add_numbers is the function name.
- (a, b) specifies the parameters the function takes.

2 “”” ... “””:

- The triple double-quoted string is a docstring.
- It serves as documentation for the function.
- It describes the purpose of the function, its parameters, and the return value.

3 result = a + b:

- This line calculates the sum of a and b and assigns it to the variable result.

4 return result:

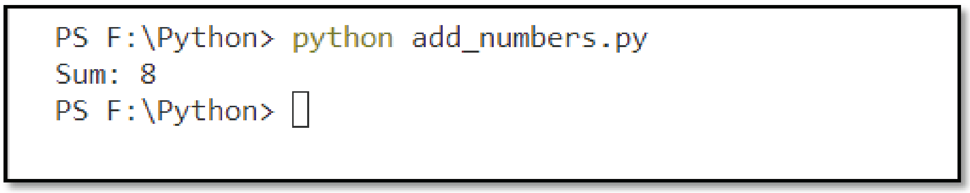
- The return statement is used to return the calculated sum.

5 Example Usage:

- sum_result = add_numbers(5, 3): Calls the add_numbers function with arguments 5 and 3 and assigns the result to sum_result.
- print(“Sum:”, sum_result): Prints the result of the addition.

This function is well-documented using a docstring, making it clear how to use it and what it does. It follows best practices for documenting functions in Python.

Output:



```
PS F:\Python> python add_numbers.py
Sum: 8
PS F:\Python>
```

— — — — —

TASK 4**: Python for loop with range function**

Code:

```
# Python Program to
# show range() basics
# printing a number
fori in range(10):
    print(i, end=” “)
# performing sum of first 10 numbers
sum = 0
fori in range(1, 10):
    sum = sum + i
print(“\nSum of first 10 numbers :”, sum)
```

Explanation:

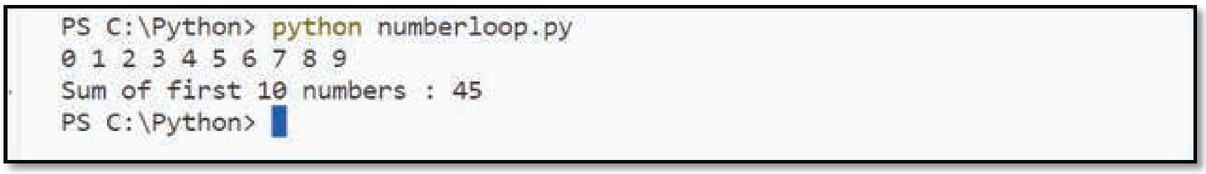
The Python range() function is used to generate a sequence of numbers. Depending on how many arguments the user is passing to the function, the user can decide where that series of numbers will begin and end as well as how big the difference will be between one number and the next.range() takes mainly three arguments.

- start: integer starting from which the sequence of integers is to be returned
- stop: integer before which the sequence of integers is to be returned.

The range of integers ends at a stop – 1.

- step: integer value which determines the increment between each integer in the sequence.

Output:



```
PS C:\Python> python numberloop.py
0 1 2 3 4 5 6 7 8 9
Sum of first 10 numbers : 45
PS C:\Python>
```

EXERCISE 128 : Write program in python using list, tuples, dictionaries and files

Objectives

At the end of this exercise you shall be able to

- develop python program to use list, tuples, dictionaries and files.

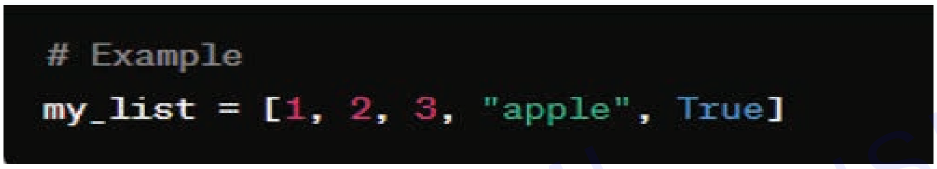
Procedure

Python List:

In Python, a list is a versatile and commonly used data structure that allows you to store and manipulate a collection of elements. Here's a brief practical explanation of Python lists:

Creating Lists:

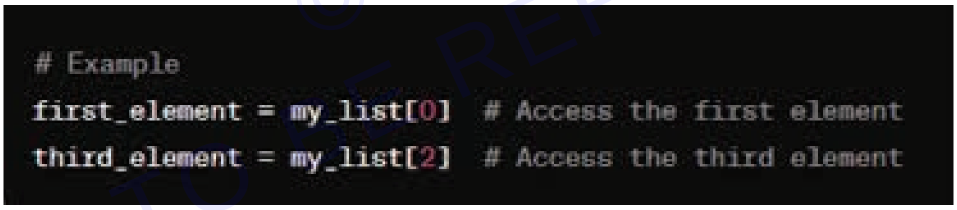
You can create a list by enclosing elements in square brackets [ ]. Elements can be of any data type, and a list can contain a mix of different types.



```
# Example
my_list = [1, 2, 3, "apple", True]
```

Accessing Elements:

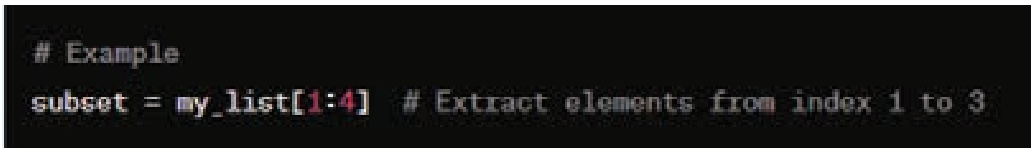
You can access elements in a list using indexing. Python uses 0-based indexing, meaning the first element is at index 0.



```
# Example
first_element = my_list[0]  # Access the first element
third_element = my_list[2]  # Access the third element
```

Slicing Lists:

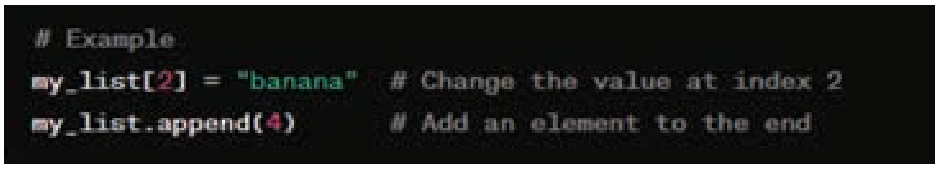
You can extract a portion of a list using slicing. Slicing is done using the colon : operator.



```
# Example
subset = my_list[1:4]  # Extract elements from index 1 to 3
```

Modifying Lists:

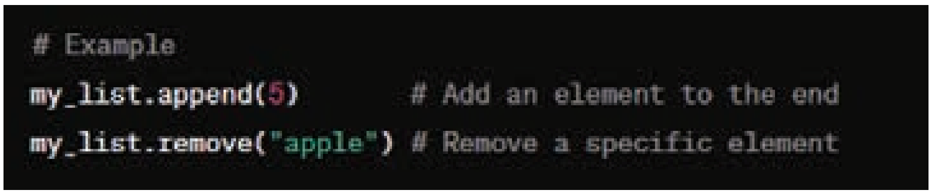
Lists are mutable, meaning you can change their elements.



```
# Example
my_list[2] = "banana"  # Change the value at index 2
my_list.append(4)      # Add an element to the end
```

List Methods:

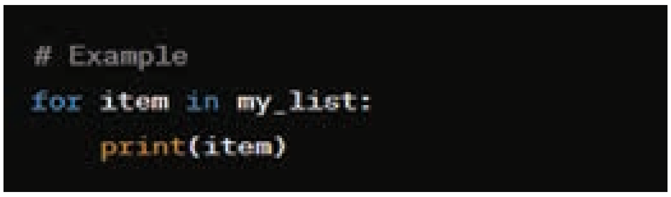
Python provides several built-in methods for manipulating lists, such as append(), remove(), pop(), extend(), and more.



```
# Example
my_list.append(5)          # Add an element to the end
my_list.remove("apple") # Remove a specific element
```

Iterating Through Lists:

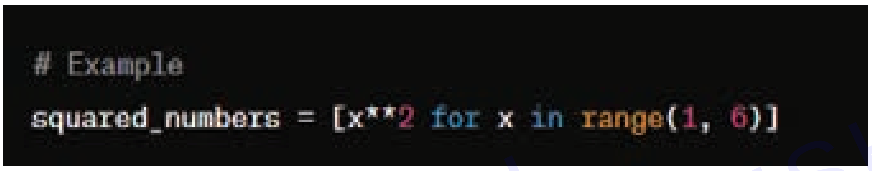
You can use loops to iterate through the elements of a list.



```
# Example
for item in my_list:
    print(item)
```

List Comprehensions:

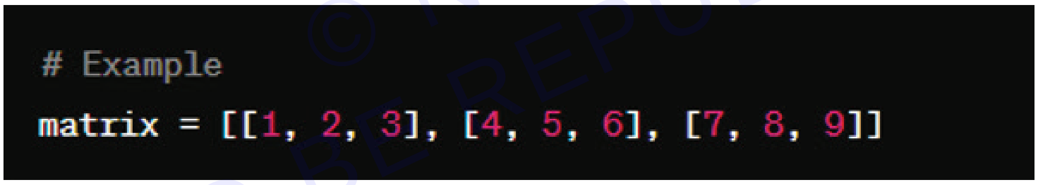
Python supports concise ways to create lists using list comprehensions.



```
# Example
squared_numbers = [x**2 for x in range(1, 6)]
```

Nested Lists:

Lists can contain other lists, creating nested structures.



```
# Example
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Here are five tasks demonstrating different aspects of Python lists:

— — — — —

TASK 1: **Basic List Operations**

# Creating a list

fruits = [“Grapes”, “Banana”, “Orange”,]

# Accessing elements

print(fruits[0])   # Output: Grapes

# Modifying list

fruits.append(“Apple”) # Adding a new element

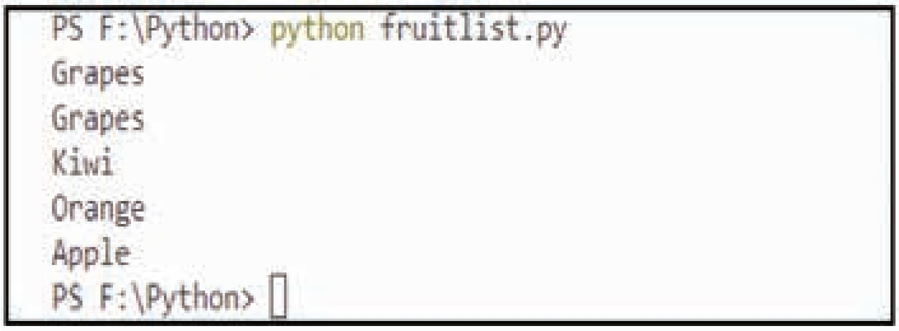
fruits[1] = “Kiwi”   # Modifying an element

# Iterating through the list

for fruit in fruits:

print(fruit)

Output:



```
PS F:\Python> python fruitlist.py
Grapes
Grapes
Kiwi
Orange
Apple
PS F:\Python>
```

— — — — —

TASK 2: **List Slicing**

# Creating a list

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

print (“\n Slicing the list from 2:7: “)

# Slicing the list

subset = numbers[2:7]   # Output: [3, 4, 5, 6, 7]

foriin subset :

print (i, end=” “)

print (“\n Odd Number slicing from the list:”)

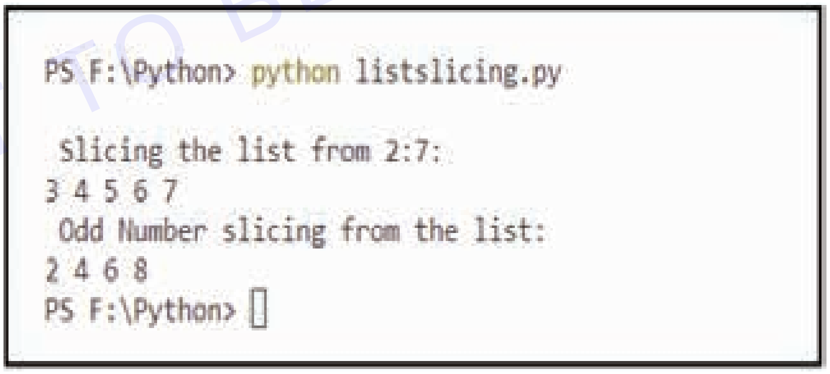
# Step in slicing

even_numbers = numbers[1::2] # Output: [2, 4, 6, 8]

foriineven_numbers :

print (i, end=” “)

Output:



```
PS F:\Python> python listslicing.py

 Slicing the list from 2:7:
3 4 5 6 7
 Odd Number slicing from the list:
2 4 6 8
PS F:\Python>
```

— — — — —

TASK 3: **List Comprehension**

# Creating a list

original_numbers = [1, 2, 3, 4, 5]

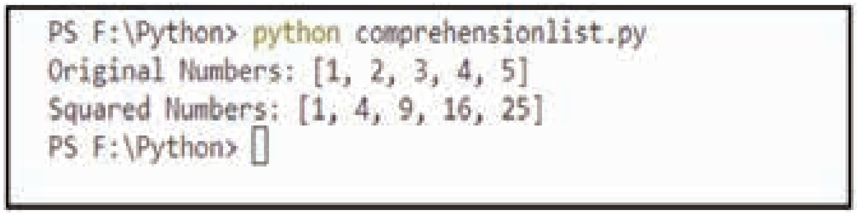
# Using list comprehension to create a new list

squared_numbers = [x**2for x inoriginal_numbers]

print(“Original Numbers:”, original_numbers)

print(“Squared Numbers:”, squared_numbers)

Output:



```
PS F:\Python> python comprehensionlist.py
Original Numbers: [1, 2, 3, 4, 5]
Squared Numbers: [1, 4, 9, 16, 25]
PS F:\Python> 
```

— — — — —

TASK 4: **Nested Lists**

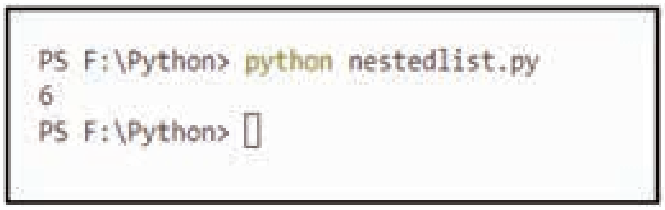
# Creating a nested list

matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Accessing elements in a nested list

print(matrix[1][2]) # Output: 6

Output:



```
PS F:\Python> python nestedlist.py
6
PS F:\Python> 
```

— — — — —

TASK 5: **List Method**

# Creating a list

my_list = [10, 20, 30, 40]

# Using list methods

my_list.append(50)   # Appending an element

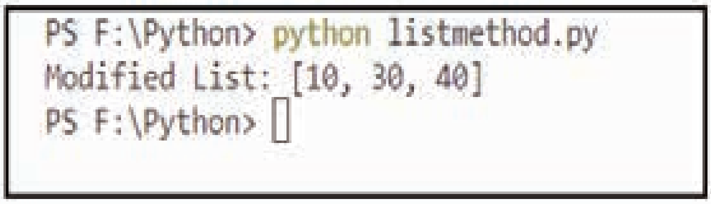
my_list.remove(20)   # Removing an element

popped_value = my_list.pop() # Popping and retrieving the last element

# Displaying the modified list

print("Modified List:", my_list)

Output:



```
PS F:\Python> python listmethod.py
Modified List: [10, 30, 40]
PS F:\Python> 
```

— — — — —

TASK 6: **Maximum and Minimum**

# Creating a list of numbers

numbers = []

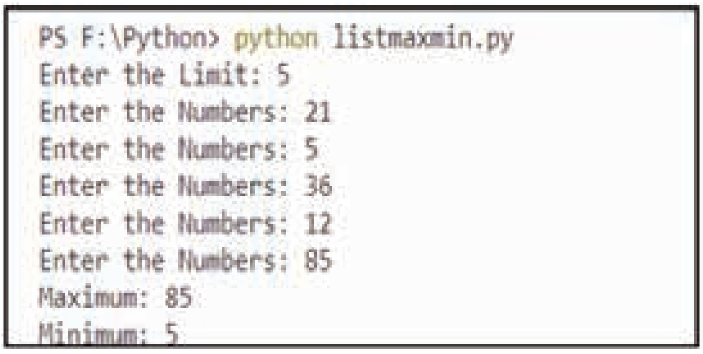
limit=int(input("Enter the Limit: "))

foriin range(0,limit):

numbers.append(int(input("Enter the Numbers: ")))

```
# Finding maximum and minimum
max_num = max(numbers)
min_num = min(numbers)
print("Maximum:", max_num)
print("Minimum:", min_num)
```

Output:



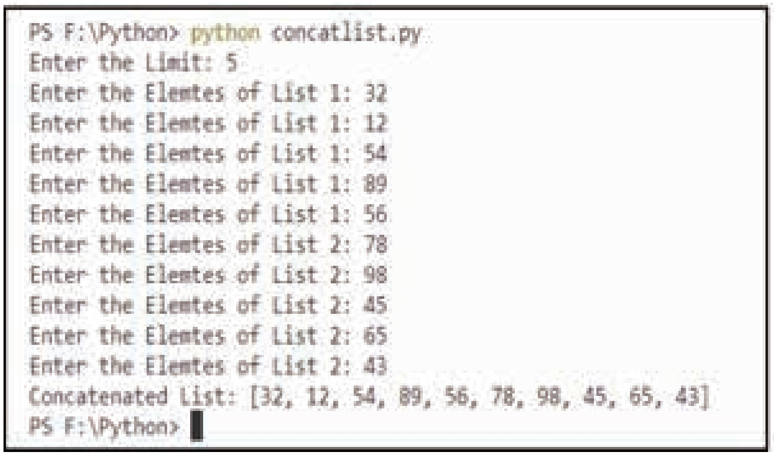
```
PS F:\Python> python listmaxmin.py
Enter the Limit: 5
Enter the Numbers: 21
Enter the Numbers: 5
Enter the Numbers: 36
Enter the Numbers: 12
Enter the Numbers: 85
Maximum: 85
Minimum: 5
```

— — — — —

TASK 7: **List Concatenation**

```
# Creating two lists
list1 = []
list2 = []
limit=int(input("Enter the Limit: "))
foriin range(0,limit):
list1.append(int(input("Enter the Elemtes of List 1: ")))
foriin range(0,limit):
list2.append(int(input("Enter the Elemtes of List 2: ")))
# Concatenating lists
result_list = list1 + list2
print("Concatenated List:", result_list)
```

Output:



```
PS F:\Python> python concatlist.py
Enter the Limit: 5
Enter the Elemtes of List 1: 32
Enter the Elemtes of List 1: 12
Enter the Elemtes of List 1: 54
Enter the Elemtes of List 1: 89
Enter the Elemtes of List 1: 56
Enter the Elemtes of List 2: 78
Enter the Elemtes of List 2: 98
Enter the Elemtes of List 2: 45
Enter the Elemtes of List 2: 65
Enter the Elemtes of List 2: 43
Concatenated List: [32, 12, 54, 89, 56, 78, 98, 45, 65, 43]
PS F:\Python>
```

Related Exercises:

1 Write a Python program to create a list of integers and display its elements.

2 Create a list of strings and print the length of the list.

3. Write a program to concatenate two lists.

4 Implement a Python program to find the sum of elements in a numeric list.

5 Create a list of numbers and remove the duplicates.

6 Write a program to reverse a list.

7 Generate a list of squares of numbers from 1 to 10 using list comprehension.

8 Create a new list containing only even numbers from an existing list.

9 Implement a program to add two matrices using lists.

10 Write a Python script to find the transpose of a matrix.

11 Create a list of words and sort them in alphabetical order.

12 Write a program to sort a list of numbers in descending order.

13 Implement a program to extract a sub-list from a given list.

14 Write a Python script to reverse every alternate sublist of a list.

15 Create a program to check if a specific element exists in a list.

16 Write a Python function to find the index of the first occurrence of an element in a list.

17 Implement a Python program to flatten a nested list.

18 Write a script to find the sum of each row in a matrix represented as a list of lists.

19 Create a program to find the second-largest element in a list.

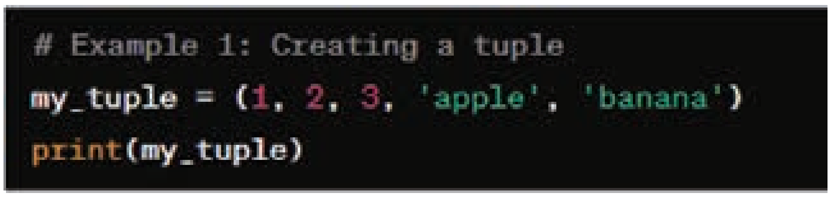
20 Implement a Python function to rotate a list by a given number of positions

— — — — —

Tuples

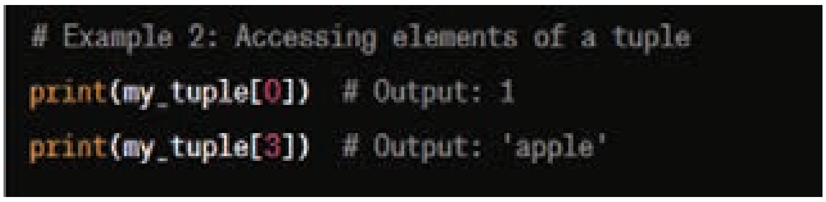
A tuple in Python is a collection of ordered, immutable elements. Tuples are similar to lists, but the key difference is that tuples are immutable, meaning their elements cannot be changed or modified once the tuple is created. Tuples are defined using parentheses ().

Creating a Tuple



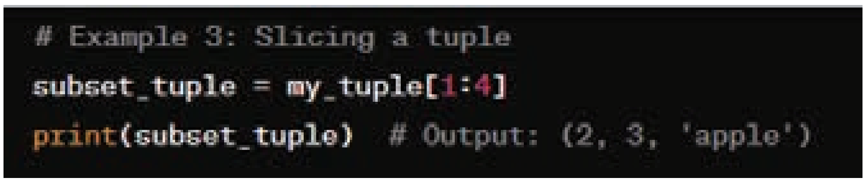
```
# Example 1: Creating a tuple
my_tuple = (1, 2, 3, 'apple', 'banana')
print(my_tuple)
```

Accessing Elements



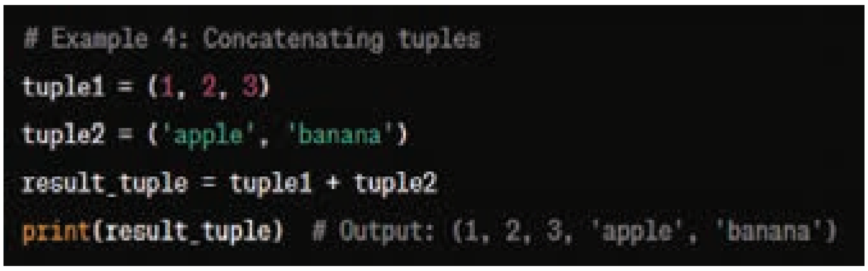
```
# Example 2: Accessing elements of a tuple
print(my_tuple[0])  # Output: 1
print(my_tuple[3])  # Output: 'apple'
```

Tuple Slicing



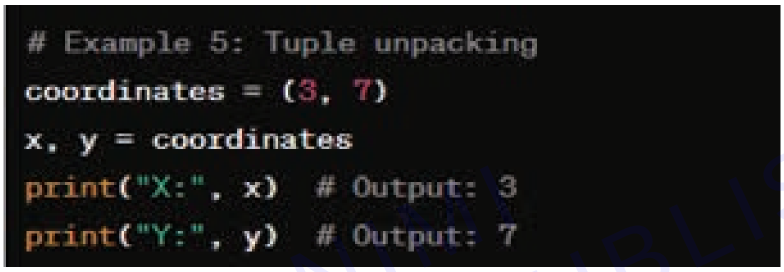
```
# Example 3: Slicing a tuple
subset_tuple = my_tuple[1:4]
print(subset_tuple)  # Output: (2, 3, 'apple')
```

Tuple Concatenation



```
# Example 4: Concatenating tuples
tuple1 = (1, 2, 3)
tuple2 = ('apple', 'banana')
result_tuple = tuple1 + tuple2
print(result_tuple)  # Output: (1, 2, 3, 'apple', 'banana')
```

Tuple Unpacking



```
# Example 5: Tuple unpacking
coordinates = (3, 7)
x, y = coordinates
print("X:", x)  # Output: 3
print("Y:", y)  # Output: 7
```

Tuples are often used when the order and immutability of elements are important. They are useful in scenarios where you want to represent a fixed collection of items that should not be modified during the program execution.

TASK 1: **Basic Tuple**

# Example 1: Creating a basic tuple

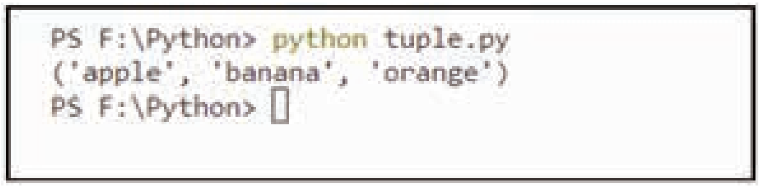
fruits = (‘apple’, ‘banana’, ‘orange’)

print(fruits)

# Output: (‘apple’, ‘banana’, ‘orange’)

Explanation: In this example, a tuple named fruits is created with three elements. Tuples are defined using parentheses.

Output:



```
PS F:\Python> python tuple.py
('apple', 'banana', 'orange')
PS F:\Python> 
```

— — — — —

TASK 2: **Mixed Tupls**

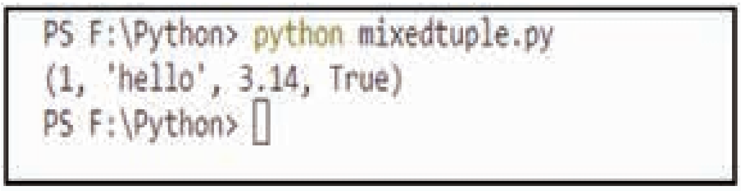
# Example 2: Tuples with mixed data types

mixed_tuple = (1, ‘hello’, 3.14, True)

print(mixed_tuple)

# Output: (1, ‘hello’, 3.14, True)

Output:



```
PS F:\Python> python mixedtuple.py
(1, 'hello', 3.14, True)
PS F:\Python> 
```

— — — — —

TASK 3: **Nested tuple**

# Read a nested tuple from the keyboard

# Input format: Enter a nested tuple as a string, e.g., (‘apple’, (1, 2, 3), [‘a’, ‘b’, ‘c’])

input_string = input(“Enter a nested tuple: “)

# Convert the input string to a tuple

nested_tuple = eval(input_string)

# Check if the entered value is a tuple

ifisinstance(nested_tuple, tuple):

    print(“Entered nested tuple:”, nested_tuple)

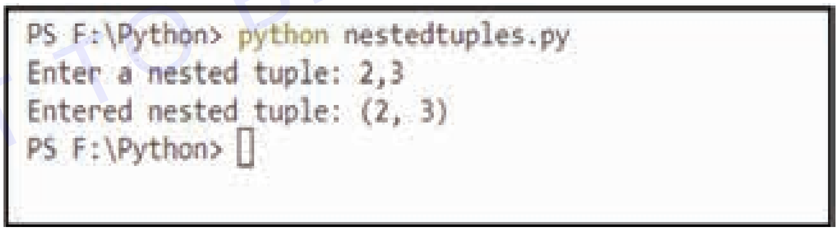
else:

    print(“Invalid input. Please enter a valid nested tuple.”)

Explanation:

- The program allows users to enter a nested tuple as a string.
- It then evaluates the string and converts it into a tuple.
- If the entered value is indeed a tuple, it is printed along with a confirmation message.
- If the input is not a tuple, an error message indicating invalid input is displayed.

Output:



```
PS F:\Python> python nestedtuples.py
Enter a nested tuple: 2,3
Entered nested tuple: (2, 3)
PS F:\Python> 
```

— — — — —

TASK 4: T**uple Operations**

Code:

#Tuple operations

tuple1 = (1, 2, 3)

tuple2 = (‘a’, ‘b’, ‘c’)

concatenated_tuple = tuple1 + tuple2

print(“Tuple1 is:”,tuple1)

print (“Tuple2 is:”,tuple2)

print(“concatenated_tuple is:”, concatenated_tuple)

# Output: (1, 2, 3, ‘a’, ‘b’, ‘c’)

Explanation:

1 **Tuple Initialization:**

- tuple1 is initialized with the values (1, 2, 3).
- tuple2 is initialized with the values ('a', 'b', 'c').

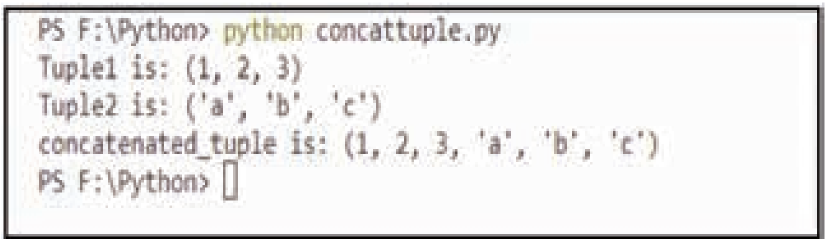
2 **Tuple Concatenation:**

- concatenated_tuple is created by concatenating tuple1 and tuple2. The + operator concatenates the two tuples.

3 **Printing the Tuples:**

- These lines print the original tuples (tuple1 and tuple2) and the result after concatenation (concatenated_tuple).

Output:



```
PS F:\Python> python concattuple.py
Tuple1 is: (1, 2, 3)
Tuple2 is: ('a', 'b', 'c')
concatenated_tuple is: (1, 2, 3, 'a', 'b', 'c')
PS F:\Python> 
```

Related Exercises:

1 Python program to find tuples which have all elements divisible by K from a list of tuples

2 Python program to find Tuples with positive elements in List of tuples

3 Python – Count tuples occurrence in list of tuples

4 Python – Removing duplicates from tuple

5 Python program to sort a list of tuples alphabetically

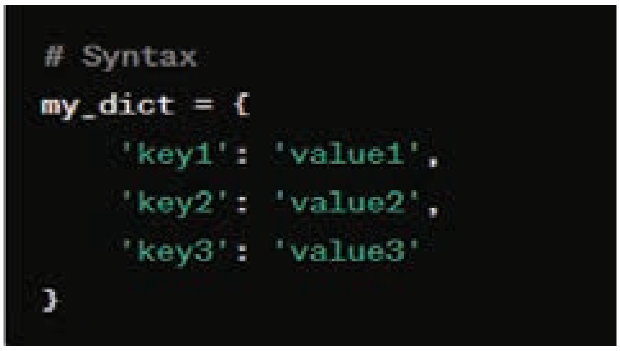
Dictionaries in Python

Dictionaries in Python are versatile data structures that allow you to store and retrieve data using key-value pairs. Here's a brief explanation of dictionaries:

Overview:

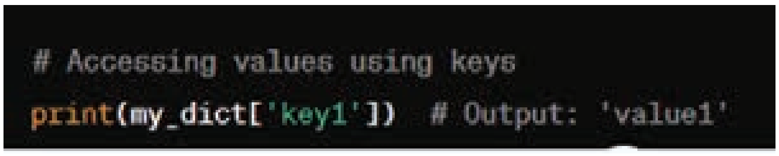
- A dictionary is an unordered collection of items.
- Each item in a dictionary consists of a key-value pair.
- Keys must be unique within a dictionary.
- Values can be of any data type, and they can be duplicates.

Creating a Dictionary:



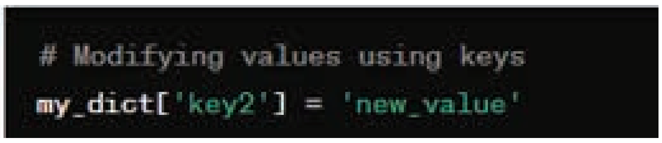
```
# Syntax
my_dict = {
    'key1': 'value1',
    'key2': 'value2',
    'key3': 'value3'
}
```

Accessing Values:



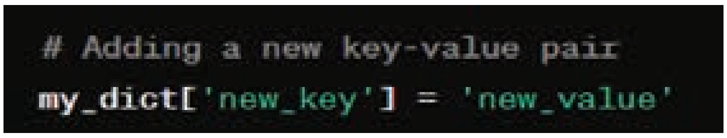
```
# Accessing values using keys
print(my_dict['key1'])  # Output: 'value1'
```

Modifying a Dictionary:



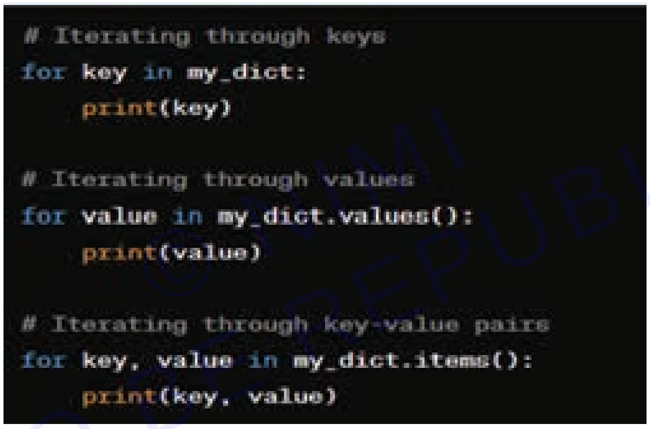
```
# Modifying values using keys
my_dict['key2'] = 'new_value'
```

Adding New Key-Value Pairs:



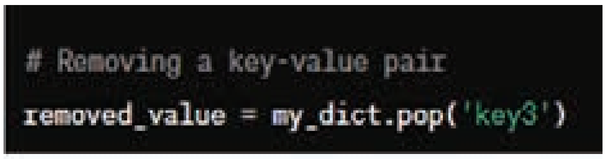
```
# Adding a new key-value pair
my_dict['new_key'] = 'new_value'
```

Iterating Through a Dictionary:



```
# Iterating through keys
for key in my_dict:
    print(key)

# Iterating through values
for value in my_dict.values():
    print(value)

# Iterating through key-value pairs
for key, value in my_dict.items():
    print(key, value)
```

Removing Key-Value Pairs:



```
# Removing a key-value pair
removed_value = my_dict.pop('key3')
```

Useful Methods:

- keys(): Returns a list of all keys.
- values(): Returns a list of all values.
- items(): Returns a list of key-value pairs as tuples.

— — — — —

TASK 1 : **Basic Dictionary**

# Creating a basic dictionary

student = {

    'name': 'SreeHari',

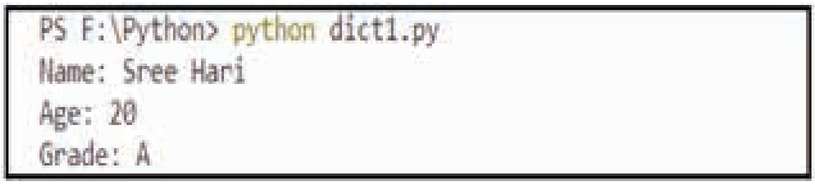
    'age': 20,

```
    'grade': 'A'
}
# Accessing dictionary elements
print("Name:", student['name'])
print("Age:", student['age'])
print("Grade:", student['grade'])
```

Explanation:

- A dictionary named student is created with keys ('name', 'age', 'grade') and corresponding values.
- Elements of the dictionary are accessed using keys, and their values are printed.

Output:



```
PS F:\Python> python dict1.py
Name: Sree Hari
Age: 20
Grade: A
```

— — — — —

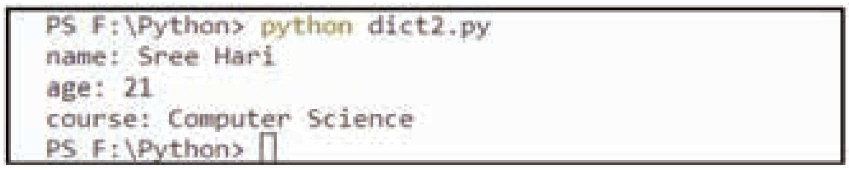
TASK 2: **Dictionary Operations**

```
student = {
    'name': 'SreeHari',
     'age': 20,
    'grade': 'A'
}
# Modifying dictionary
student['age'] = 21
student['course'] = 'Computer Science'
# Deleting a key-value pair
if'grade'in student:
    del student['grade']
# Iterating through dictionary items
for key, value instudent.items():
    print(f"{key}: {value}")
```

Explanation:

- The dictionary is modified by updating the 'age' and adding a new key 'course'.
- A key-value pair ('grade') is deleted using the del keyword.
- A loop iterates through the dictionary items, printing each key-value pair.

Output :



```
PS F:\Python> python dict2.py
name: Sree Hari
age: 21
course: Computer Science
PS F:\Python> 
```

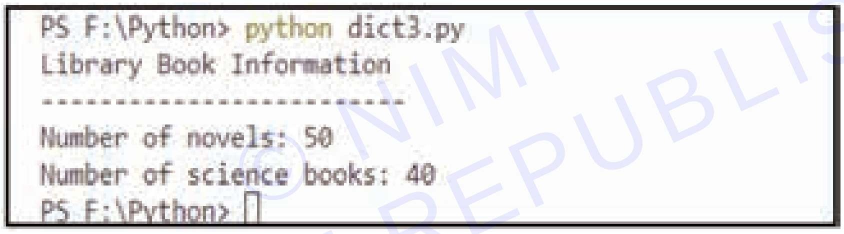
TASK 3 : **Nested Dictionary**

```
# Creating a nested dictionary
library = {
    'fiction': {'novels': 50, 'short stories': 30},
    'non-fiction': {'history': 20, 'science': 40}
}
# Accessing nested dictionary elements
print("Library Book Information")
print("------------------------")
print("Number of novels:", library['fiction']['novels'])
print("Number of science books:", library['non-fiction']['science'])
```

Explanation:

- The dictionary 'library' is nested with categories ('fiction', 'non-fiction') containing subcategories and their counts.
- Nested elements are accessed using multiple keys.

Output:



```
PS F:\Python> python dict3.py
Library Book Information
-------------------------
Number of novels: 50
Number of science books: 40
PS F:\Python>
```

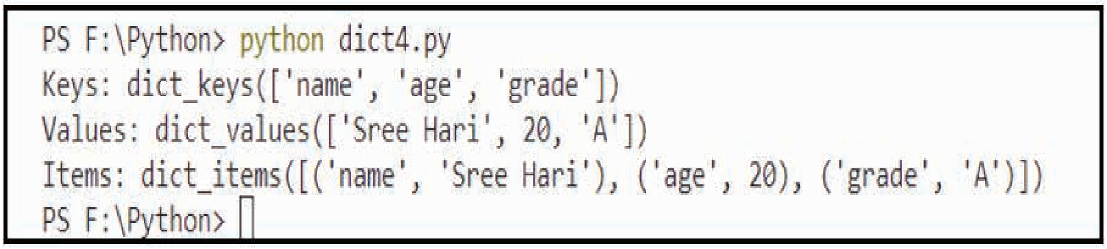
TASK 4: **Dictionary Methods**

```
student = {
    'name': 'SreeHari',
    'age': 20,
    'grade': 'A'
}
# Using dictionary methods
keys = student.keys()
values = student.values()
items = student.items()
print("Keys:", keys)
print("Values:", values)
print("Items:", items)
```

Explanation:

- The methods keys(), values(), and items() are used to retrieve keys, values, and key-value pairs, respectively.

Output:



```
PS F:\Python> python dict4.py
Keys: dict_keys(['name', 'age', 'grade'])
Values: dict_values(['Sree Hari', 20, 'A'])
Items: dict_items([('name', 'Sree Hari'), ('age', 20), ('grade', 'A')])
PS F:\Python> 
```

— — — — —

TASK 5: **Dictionary Comprehension**

# Dictionary comprehension

squared_numbers = {x: x**2for x inrange(1, 6)}

print(“Squared Numbers:”, squared_numbers)

Explanation:

- A dictionary is created using dictionary comprehension to store squared numbers from 1 to 5.

Output

Related Exercises:

1 Write a Python program to create an empty dictionary.

2 Create a dictionary with three key-value pairs representing student information (name, age, grade).

3 Access and print the value associated with the ‘age’ key in the student dictionary.

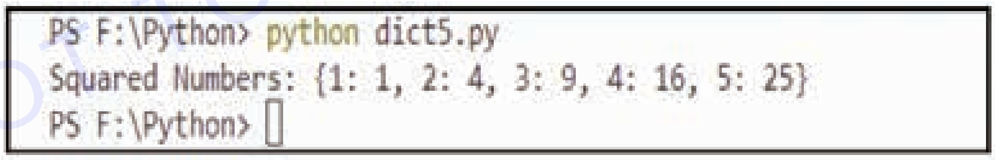
4 Modify the student dictionary to update the age to 22.

5 Add a new key-value pair to the student dictionary for the ‘course’ with the value ‘Computer Science’.

6 Remove the ‘grade’ key from the student dictionary if it exists.

7 Write a loop to iterate through the key-value pairs in the student dictionary and print them.

8 Use the keys(), values(), and items() methods to display the keys, values, and key-value pairs of the student dictionary.



```
PS F:\Python> python dict5.py
Squared Numbers: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
PS F:\Python> 
```

EXERCISE 129 : Write a python program depicting argument passing and using tuples, dictionaries as arguments

Objectives

At the end of this exercise you shall be able to

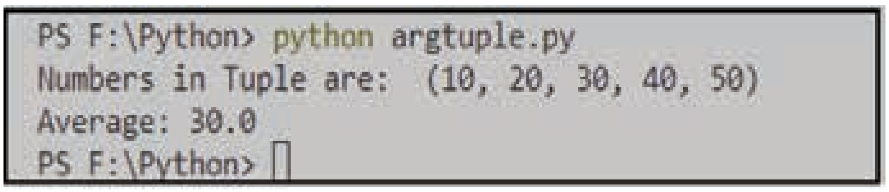
- develop python program to argument passing and using tuples, dictionaries as arguments.

Procedure

TASK 1: **Using Tuple**

```
defcalculate_average(*numbers):
    """
    This function takes variable positional arguments (numbers) as a tuple
    and calculates their average.
    Args:
        *numbers: Variable positional arguments (packed into a tuple).
    Returns:
        float: Average of the numbers.
    """
    ifnot numbers:
        return0  # Avoid division by zero
    average = sum(numbers) / len(numbers)
    return average
# Example usage:
result = calculate_average(10,20,30,40,50)
num=(10,20,30,40,50)
print("Numbers in Tuple are: ",num)
print("Average:", result)
```

Output:



```
PS F:\Python> python argtuple.py
Numbers in Tuple are:  (10, 20, 30, 40, 50)
Average: 30.0
PS F:\Python>
```

— — — — —

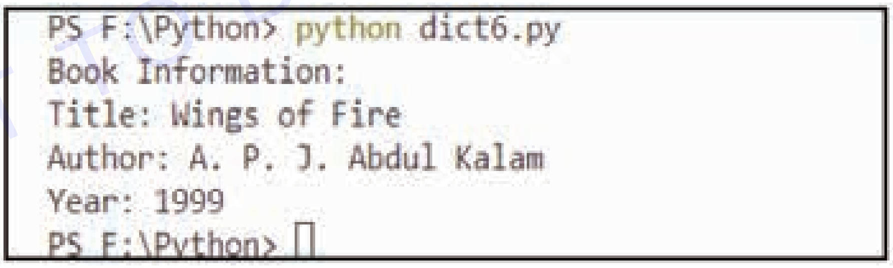
TASK 2: **Using Dictionary**

```
defprint_book_info(**book_details):
    """
    This function takes book details as keyword arguments packed into a dictionary.
    Args:
        **book_details: Book details (packed into a dictionary).
    """
    print("Book Information:")
    for key, value inbook_details.items():
        print(f"{key.capitalize()}: {value}")
# Example usage:
print_book_info(title="Wings of Fire ", author="A. P. J. Abdul Kalam", year=1999)
```

Explanation :

- The function print_book_info is defined to accept any number of keyword arguments. The double-asterisk ** before book_details allows the function to receive these keyword arguments and pack them into a dictionary called book_details.
- Inside the function, it prints "Book Information:" to indicate that it's displaying information about a book.
- It then iterates through the key-value pairs in the book_details dictionary using a for loop. For each key-value pair, it prints the key (capitalized using capitalize()) and the corresponding value.
- The print_book_info function can handle different sets of book details, and the keys and values will be printed in a readable format.
- In the example usage, the function is called with specific book details for "Wings of Fire" by A. P. J. Abdul Kalam, published in 1999.

Output:



```
PS F:\Python> python dict6.py
Book Information:
Title: Wings of Fire
Author: A. P. J. Abdul Kalam
Year: 1999
PS F:\Python> 
```

— — — — —

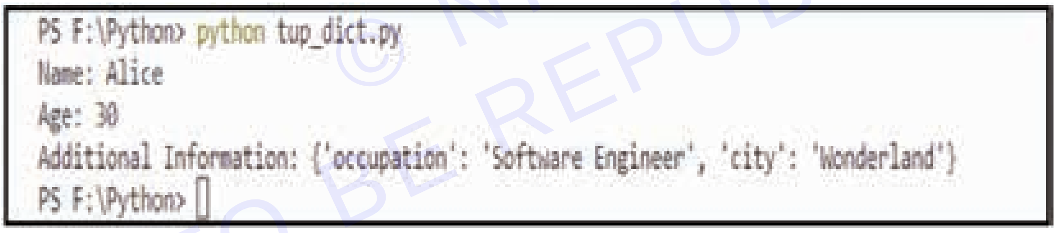
TASK 3: **Using Tuple & Dictionaries**

```
defprint_person_info(name, age, **additional_info):
    """
    This function takes required positional arguments (name, age)
    and additional keyword arguments packed into a dictionary.
    Args:
        name (str): Person's name.
        age (int): Person's age.
        **additional_info: Additional information (packed into a dictionary).
    """
    print("Name:", name)
    print("Age:", age)
    print("Additional Information:", additional_info)
# Example usage:
print_person_info("Alice", 30, occupation="Software Engineer", city="Wonderland")
```

Explanations :

- The function print_person_info is defined to accept two required positional arguments (name and age) and any number of additional keyword arguments. The double-asterisk ** before additional_info allows the function to receive these additional keyword arguments and pack them into a dictionary called additional_info.
- Inside the function, it prints the name and age of the person using the provided positional arguments (name and age).
- It then prints the additional information provided as keyword arguments, which are packed into the additional_info dictionary.
- The print_person_info function can handle different sets of additional information for a person, and it will print all the provided details.
- In the example usage, the function is called with the name "Alice", age 30, and additional information about her occupation and city.

Output:



```
PS F:\Python> python tup_dict.py
Name: Alice
Age: 30
Additional Information: {'occupation': 'Software Engineer', 'city': 'Wonderland'}
PS F:\Python> 
```

Related Exercises:

1 Program 1: Tuple as Argument

- Write a Python program that defines a function to calculate the average of numbers passed as a tuple.

2 Program 2: Dictionary as Argument

- Create a program that defines a function to display the details of a person. The function should take a dictionary with keys like 'name', 'age', and 'city' as arguments.

3 Program 3: Multiple Arguments and Default Values

- Write a program with a function that accepts multiple arguments, including a tuple and a dictionary. Set default values for some parameters.

EXERCISE 130 : Write a python program for importing a module

Objectives

At the end of this exercise you shall be able to

- develop python program for importing a module.

Procedure

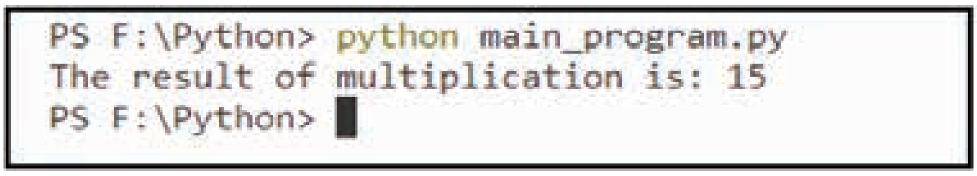
TASK 1: **Module contains a utility function**

Step1

- Create a module named utilities.py
- Save the following code in a file named utilities.py
- This module contains a utility function

```
# File: utilities.py
def multiply(a, b):
return a * b
# End of utilities.py
Step 2:
# Now, create a Python program to import and use the module
# File: main_program.py
# Import the module
import utilities
# Use the function from the imported module
result = utilities.multiply(5, 3)
# Display the result
print(f"The result of multiplication is: {result}")
# End of main_program.py
```

Output:



```
PS F:\Python> python main_program.py
The result of multiplication is: 15
PS F:\Python> █
```

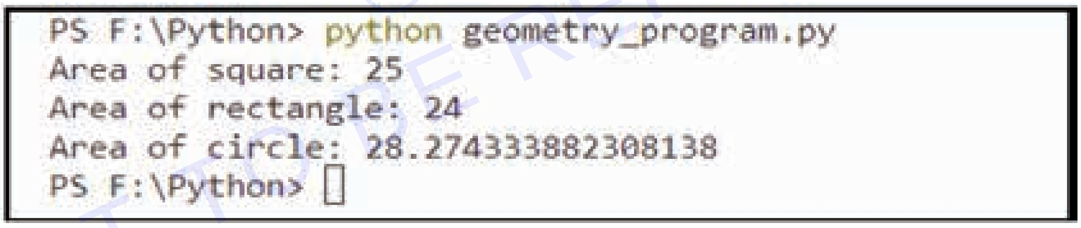
In this example, utilities.py is a separate Python file containing a function multiply. The main_program file imports this module and calls the multiply function to return the product of two numbers . Make sure both files (utilities. pyandmain_program.py) are in the same directory or provide the correct path when importing the module.

— — — — —

TASK 2: **Calculate the area of different shapes**

```
# File: geometry.py
import math
defcalculate_square_area(side):
 return side * side
defcalculate_rectangle_area(length, width):
return length * width
defcalculate_circle_area(radius):
returnmath.pi * radius * radius
# File: geometry_program.py
import geometry
# Calculate areas using functions from the geometry module
square_area = geometry.calculate_square_area(5)
rectangle_area = geometry.calculate_rectangle_area(6, 4)
circle_area = geometry.calculate_circle_area(3)
# Print the calculated areas
print("Area of square:", square_area)
print("Area of rectangle:", rectangle_area)
print("Area of circle:", circle_area)
```

Output:



```
PS F:\Python> python geometry_program.py
Area of square: 25
Area of rectangle: 24
Area of circle: 28.274333882308138
PS F:\Python> 
```

— — — — —

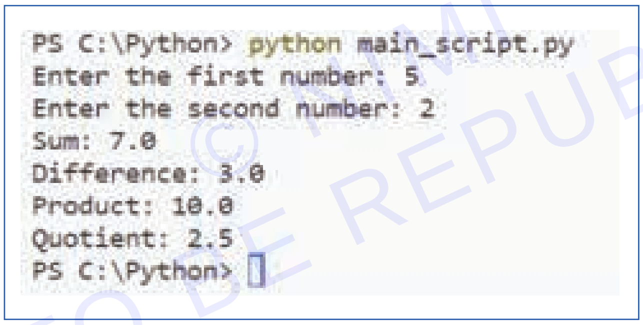
TASK 3: **Simple Arithmetic Operator**

Code:

```
# my_module.py
defadd(x, y):
returnx+y
defsubtract(x, y):
returnx-y
defmultiply(x, y):
returnx*y
defdivide(x, y):
ify!=0:
returnx/y
else:
```

```
return"Cannot divide by zero"
# main_script.py
importmy_module
# Getting user input for numbers
num1=float(input("Enter the first number: "))
num2=float(input("Enter the second number: "))
# Using arithmetic functions from my_module
sum_result=my_module.add(num1, num2)
difference_result=my_module.subtract(num1, num2)
product_result=my_module.multiply(num1, num2)
quotient_result=my_module.divide(num1, num2)
# Displaying the results
print(f"Sum: {sum_result}")
print(f"Difference: {difference_result}")
print(f"Product: {product_result}")
print(f"Quotient: {quotient_result}")
```

Output:



```
PS C:\Python> python main_script.py
Enter the first number: 5
Enter the second number: 2
Sum: 7.0
Difference: 3.0
Product: 10.0
Quotient: 2.5
PS C:\Python>
```

Related Exercises:

Exercise 1: Celsius to Fahrenheit Converter

Create a module temperature_converter.py with a function to convert Celsius to Fahrenheit and use it in a script.

Exercise 2: List Reverser

Create a module list_operations.py with a function to reverse a list and use it in a script.

Exercise 3: Palindrome Checker

Create a module palindrome_checker.py with a function to check if a given string is a palindrome and use it in a script.

Exercise 4: Date Formatter

Create a module date_formatter.py with a function to format a given date and use it in a script.

Exercise 5: Word Counter

Create a module word_counter.py with a function to count the number of words in a given text and use it in a script.

EXERCISE 131 : Use exception handling in python program

Objectives

At the end of this exercise you shall be able to

- develop python program for exception handling.

Procedure

Exception handling in Python is done using the try, except, else, and finally blocks. Here are five examples demonstrating different use cases for exception handling

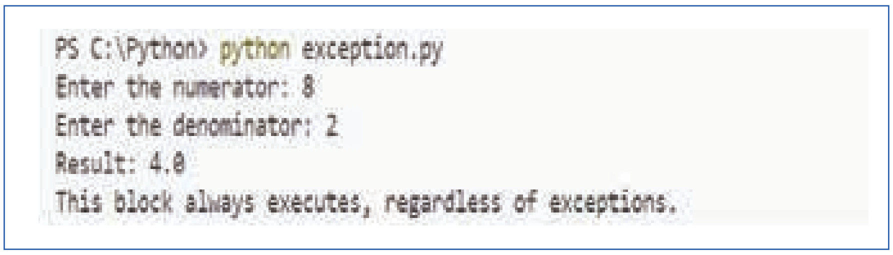
TASK 1: **Division by Zero**

```
# Example 1: Division by Zero
try:
numerator=int(input("Enter the numerator: "))
denominator=int(input("Enter the denominator: "))
result=numerator/denominator
exceptZeroDivisionError:
print("Error: Division by zero is not allowed.")
else:
print(f"Result: {result}")
finally:
print("This block always executes, regardless of exceptions.")
```

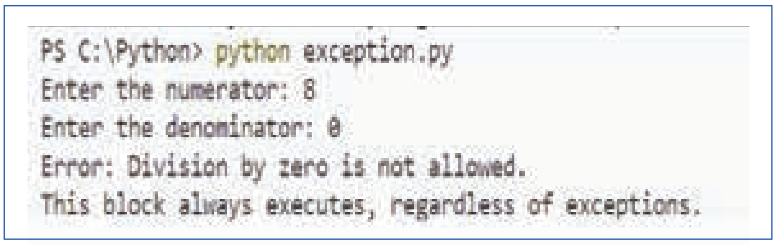
Explanation:

- We use try to enclose the code that might raise an exception.
- except Zero Division Error catches the specific exception if the denominator is zero.
- else block executes if there are no exceptions.
- finally block always executes, regardless of whether there is an exception or not.

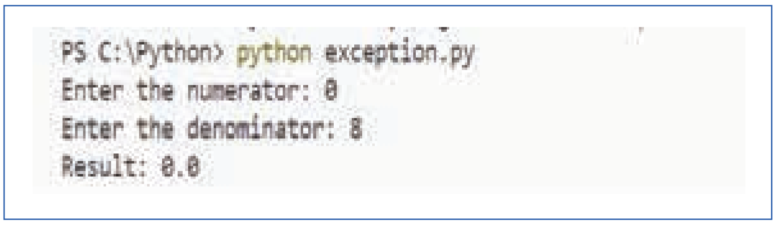
Output:



```
PS C:\Python> python exception.py
Enter the numerator: 8
Enter the denominator: 2
Result: 4.0
This block always executes, regardless of exceptions.
```



```
PS C:\Python> python exception.py
Enter the numerator: 8
Enter the denominator: 0
Error: Division by zero is not allowed.
This block always executes, regardless of exceptions.
```



```
PS C:\Python> python exception.py
Enter the numerator: 0
Enter the denominator: 8
Result: 0.0
```

— — — — —

TASK 2: **Invalid Input**

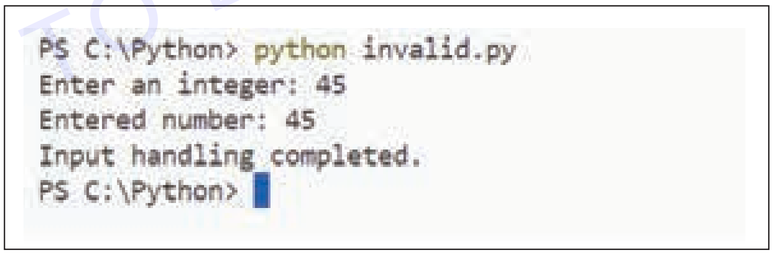
Code:

```
# Example 2: Invalid Input
try:
number=int(input("Enter an integer: "))
exceptValueError:
print("Error: Invalid input. Please enter an integer.")
else:
print(f"Entered number: {number}")
finally:
print("Input handling completed.")
```

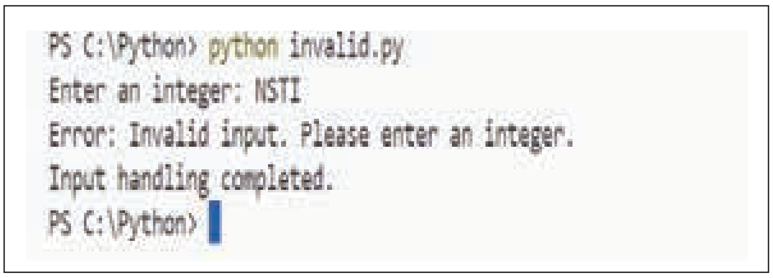
Explanation:

- Attempt to convert user input to an integer.
- exceptValueError catches the exception if the input is not a valid integer.
- else block executes if the input is a valid integer.
- finally block always executes.

Output:



```
PS C:\Python> python invalid.py
Enter an integer: 45
Entered number: 45
Input handling completed.
PS C:\Python>
```



```
PS C:\Python> python invalid.py
Enter an integer: NSTI
Error: Invalid input. Please enter an integer.
Input handling completed.
PS C:\Python>
```

— — — — —

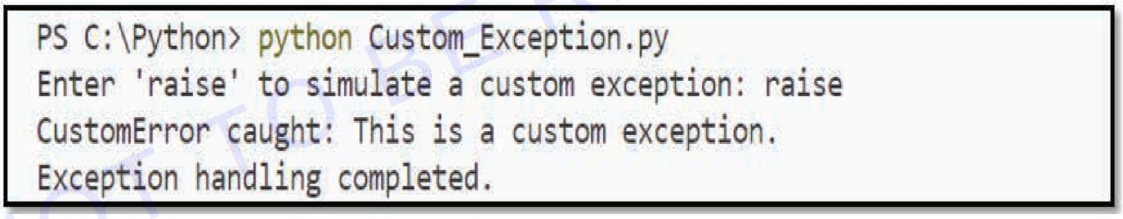
TASK 3: **Custom Exception**

```
# Example 3: Custom Exception
classCustomError(Exception):
 pass
try:
user_input=input("Enter 'raise' to simulate a custom exception: ")
ifuser_input.lower() =='raise':
raiseCustomError("This is a custom exception.")
exceptCustomErrorase:
print(f"CustomError caught: {e}")
else:
print("No exception raised.")
finally:
print("Exception handling completed.")
```

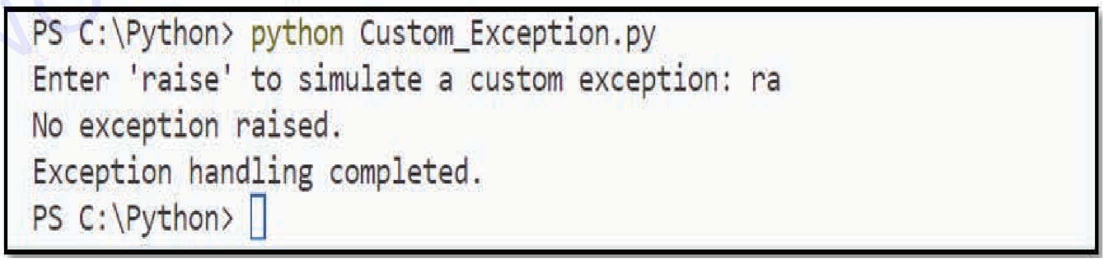
Explanation:

- We define a custom exception CustomError.
- If the user enters 'raise', we deliberately raise the custom exception.
- exceptCustomError as e catches the custom exception and prints the error message.
- else block executes if no exception is raised.
- finally block always executes.

Output:



```
PS C:\Python> python Custom_Exception.py
Enter 'raise' to simulate a custom exception: raise
CustomError caught: This is a custom exception.
Exception handling completed.
```



```
PS C:\Python> python Custom_Exception.py
Enter 'raise' to simulate a custom exception: ra
No exception raised.
Exception handling completed.
PS C:\Python>
```

— — — — —

TASK 4: **User Authentication**

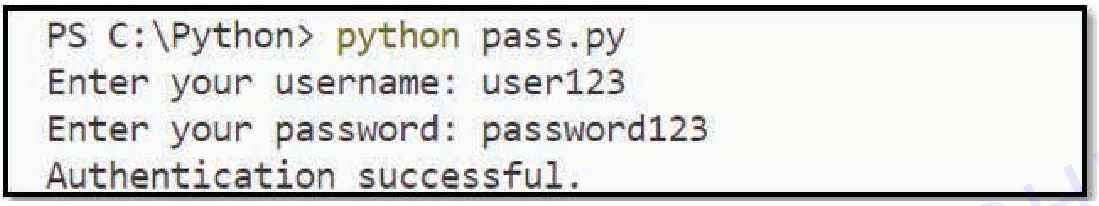
```
correct_username="user123"
correct_password="password123"
try:
username=input("Enter your username: ")
password=input("Enter your password: ")
```

```
ifusername!=correct_usernameorpassword!=correct_password:
raiseValueError(“Incorrect username or password.”)
exceptValueErrorase:
print(f”Authentication Error: {e}”)
else:
print(“Authentication successful.”)
```

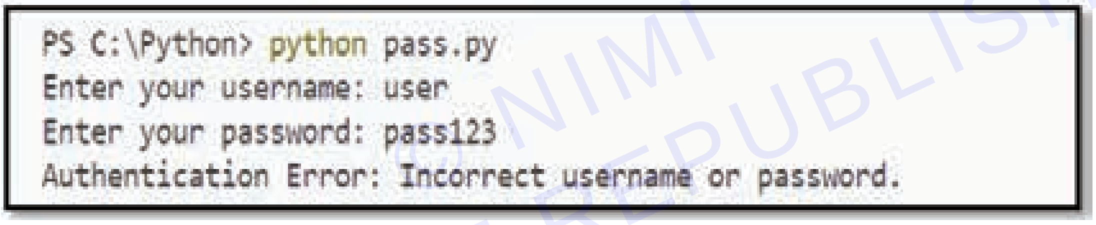
Explanation:

- The program expects the user to input a username and password.
- If the entered credentials do not match the correct ones, a ValueError is raised.
- The except block catches this exception and prints an authentication error message.
- If no exception occurs, the else block executes, indicating successful authentication.

Output:



```
PS C:\Python> python pass.py
Enter your username: user123
Enter your password: password123
Authentication successful.
```



```
PS C:\Python> python pass.py
Enter your username: user
Enter your password: pass123
Authentication Error: Incorrect username or password.
```

Related Exercises:

1 Write a program that tries to access an element at a specific index in a list. Handle the IndexError if the index is out of range.

2 Create a program that concatenates two strings entered by the user. Handle the TypeError if the user enters a non-string value

3 Write a program that asks the user to input their age. Raise a custom exception if the user enters a negative value.

4 Create a dictionary and try to access a key that doesn’t exist. Handle the KeyError gracefully.

5 Write a program that uses assert to check whether a given number is positive. Handle the AssertionError if the condition is not met.

EXERCISE 132 : Write a python program to use built in functions i.e. chr, cmp, compile, dir, eval, filter, hash, input, len, locals, long, max, pow, range, slice, tuple, Unicode, vars

Objectives

At the end of this exercise you shall be able to

- develop python programs to use built in functions i.e. chr, cmp, compile, dir, eval, filter, hash, input, len, locals, long, max, pow, range, slice, tuple, Unicode, vars.

Procedure

TASK 1 : **Write a program that takes an integer as input and uses the chr function to print the corresponding ASCII character**

Code:

```
# Exercise 1
num = int(input("Enter an integer: "))
char = chr(num)
print("Corresponding ASCII character:", char)
```

Explanation:

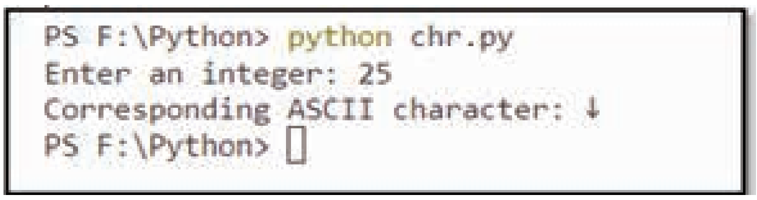
1 The program starts by prompting the user to enter an integer using input.

2 The entered value is converted to an integer using int(num).

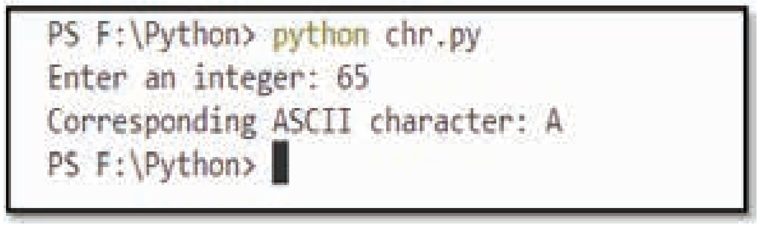
3 The chr function is then used to convert the integer to its corresponding ASCII character.

4 Finally, the result is printed, showing the ASCII character associated with the entered integer.

Output:



```
PS F:\Python> python chr.py
Enter an integer: 25
Corresponding ASCII character: ↓
PS F:\Python> 
```



```
PS F:\Python> python chr.py
Enter an integer: 65
Corresponding ASCII character: A
PS F:\Python> 
```

— — — — —

TASK 2: **Write a program that takes a Python code snippet as input, compiles it using the compile function, and executes it using exec**

Code:

# Exercise 2: compile and exec

code = input("Enter Python code snippet: ") # Prompt user for Python code

compiled_code = compile(code, '<string>', 'exec') # Compile the code

exec(compiled_code) # Execute the compiled code

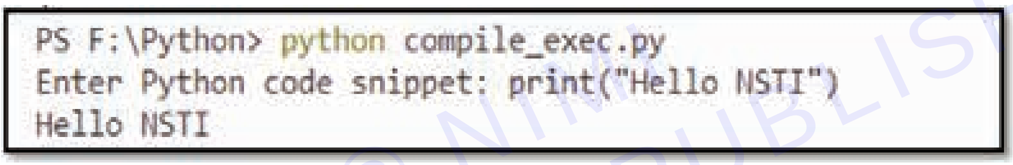
Explanation:

1 code = input("Enter Python code snippet: "): This line prompts the user to enter a Python code snippet, and the input is stored in the variable code.

2 compiled_code = compile(code, '<string>', 'exec'): The compile function is used to compile the entered code. The first argument is the code itself, the second argument ('<string>') is a filename to represent the code (it can be any string), and the third argument ('exec') specifies the compilation mode, which means the code will be executed as a series of statements.

3 exec(compiled_code): The exec function is then used to execute the compiled code. This function is used to dynamically execute Python code. The compiled code is passed as an argument to exec, and it is executed in the current global and local scopes.

Output:



```
PS F:\Python> python compile_exec.py
Enter Python code snippet: print("Hello NSTI")
Hello NSTI
```

— — — — —

TASK 3: **Write a program that uses the dir function to list all the attributes of a given object**

# Exercise 3: dir

obj = [1, 2, 3] # Example object (list)

attributes = dir(obj) # Use dir to get attributes of the object

print("Object attributes:", attributes) # Print the result

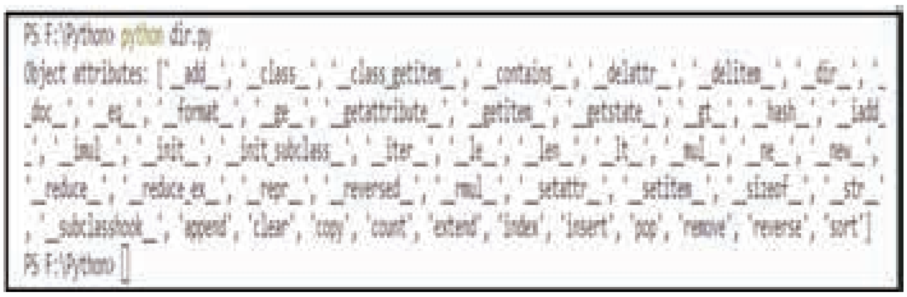
Explanation:

1 obj = [1, 2, 3]: This line creates an example object, in this case, a list [1, 2, 3].

2 attributes = dir(obj): The dir function is then used to get the attributes of the object obj. The dir function returns a list of names in the namespace of the object. In this case, it will return a list of attributes and methods available for the list object.

3 print("Object attributes:", attributes): Finally, the program prints the obtained attributes of the object. This helps in exploring the available attributes and methods for a given object.

Output:



```
PS F:\Python> python dir.py
Object attributes: ['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '_
_doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd_
_', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__'
, '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
PS F:\Python>
```

— — — — —

TASK 4: **Write a program that takes a mathematical expression as input from the user, evaluates it using the eval function, and prints the result**

Code:

```
# Exercise 4: eval and input
expression = input("Enter a mathematical expression: ")  # Prompt user for expression
result = eval(expression)  # Evaluate the expression using eval
print("Result:", result)  # Print the result
```

Explanation:

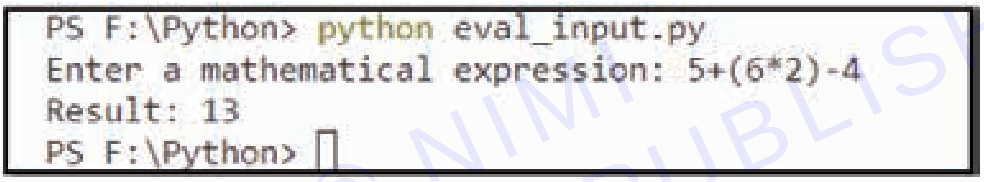
1 expression = input("Enter a mathematical expression: "): This line prompts the user to enter a mathematical expression. The entered expression is stored in the variable expression.

2 result = eval(expression): The eval function is then used to evaluate the mathematical expression stored in the variable expression. eval interprets the expression as a Python expression and returns the result.

3 print("Result:", result): Finally, the program prints the result of the evaluated expression.

So, this program demonstrates the use of the eval built-in function to dynamically evaluate a user-input mathematical expression. Users can input any valid Python expression, and the program will output the result of the evaluation.

Output :



```
PS F:\Python> python eval_input.py
Enter a mathematical expression: 5+(6*2)-4
Result: 13
PS F:\Python> 
```

TASK 5: **Write a program that uses the filter function to extract even numbers from a list**

Code:

```
# Exercise 5: filter
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # Example list
print("Numbers: ",numbers)
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))  # Use filter to get even numbers
print("Even numbers:", even_numbers)  # Print the result
```

Explanation:

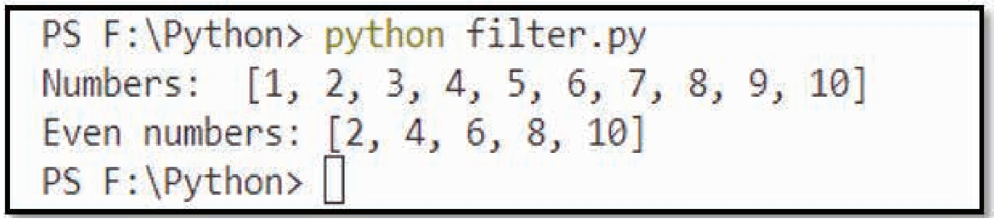
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: This line creates a list named numbers containing integer values from 1 to 10.

2 even_numbers = list(filter(lambda x: x % 2 == 0, numbers)): The filter function is used here to filter out even numbers from the numbers list. The lambda function lambda x: x % 2 == 0 checks if a number is even (x % 2 == 0). The filter function returns an iterator containing only the elements from the numbers list for which the lambda function returns True. list() is used to convert the iterator to a list.

3 print("Even numbers:", even_numbers): Finally, the program prints the list of even numbers extracted using the filter function.

So, this program demonstrates the use of the filter built-in function to selectively extract elements from a list based on a specified condition.

Output:



```
PS F:\Python> python filter.py
Numbers:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Even numbers: [2, 4, 6, 8, 10]
PS F:\Python> 
```

— — — — —

TASK 6: **Write a program that takes a string as input and prints its hashed value using the hash function**

Code:

string_to_hash = “hello”

hashed_value = hash(string_to_hash)

print(f”Hashed value of ‘{string_to_hash}’: {hashed_value}”)

Explanation:

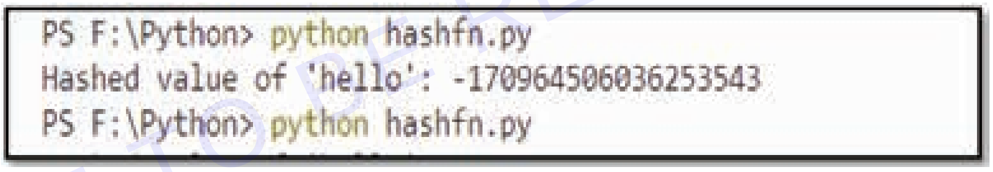
1 string_to_hash = “hello”: This line initializes a string variable string_to_hash with the value “hello”.

2 hashed_value = hash(string_to_hash): The hash() function is used to generate a hash value for the given string. It converts the string into a hash value, which is an integer representation of the string’s content. The resulting hash value is stored in the variable hashed_value.

3 print(f”Hashed value of ‘{string_to_hash}’: {hashed_value}”): This line prints the original string and its corresponding hash value. The f in print(f””) denotes an f-string, allowing variables to be directly inserted into the string. The output will show the original string and its associated hash value.

In summary, this program demonstrates how to calculate the hash value of a string using the hash() function in Python. The hash value is a numeric representation of the input string’s content.

Output:



```
PS F:\Python> python hashfn.py
Hashed value of 'hello': -1709645060362535 43
PS F:\Python> python hashfn.py
```

— — — — —

TASK 7: **Write a program that calculates and prints the length of a given string**

Code:

# 7.len

str1=”Hello, World!”

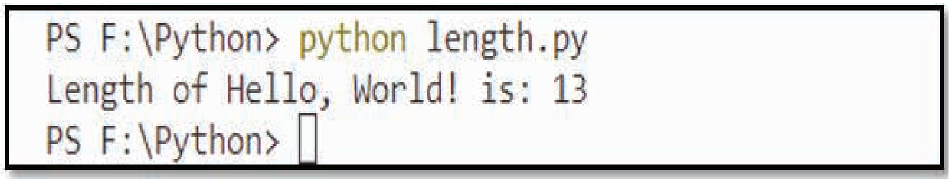
string_length = len(str1)

print(f”Length of {str1} is: {string_length}”)

Explanation:

1 This exercise demonstrates the use of the len built-in function, which is used to determine the length of a sequence or collection. In this case, the input is a string str1 (“Hello, World!”), and the len function is applied to calculate the length of the string. The result is then printed to the console. The len function is versatile and can be used with various iterable objects like strings, lists, tuples, etc.

Output:



```
PS F:\Python> python length.py
Length of Hello, World! is: 13
PS F:\Python>
```

— — — — —

TASK 8 : **Write a program that uses the locals function to display all local variables in the program**

Code:

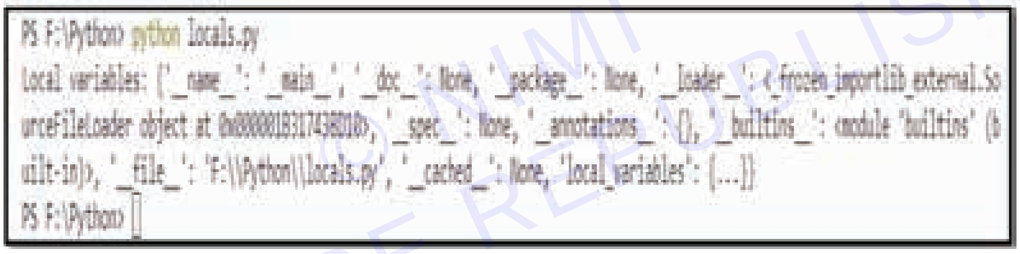
```
# 8. locals
local_variables = locals()
print(f"Local variables: {local_variables}")
```

Explanation:

1 The locals() function returns a dictionary containing the current local symbol table. In this example, locals() is called without any arguments, and it returns a dictionary containing all the local variables and their corresponding values within the current scope.

2 This can be particularly useful for debugging or inspecting the local variables within a function or code block. However, it's important to note that the result may vary depending on where locals() is called within the code.

Output:



```
PS F:\Python> python locals.py
Local variables: {'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <frozen_importlib_external.So
urceFileLoader object at 0x0000018317438D10>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (b
uilt-in)>, '__file__': 'F:\\Python\\locals.py', '__cached__': None, 'local_variables': {...}}
PS F:\Python>
```

— — — — —

TASK 9: **Write a program that finds and prints the maximum value from a given list of numbers using the max function**

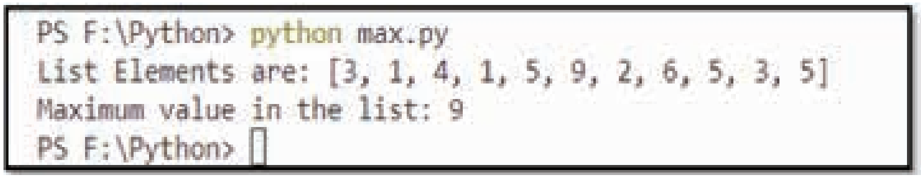
Code:

```
# 9. max
numbers_to_find_max = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
print("List Elements are:",numbers_to_find_max)
maximum_value = max(numbers_to_find_max)
print(f"Maximum value in the list: {maximum_value}")
```

Explanation:

1 The max() function is used to find the maximum value from a sequence (in this case, a list). In this example, the list numbers_to_find_max contains several numerical elements. The max() function is then applied to find and print the maximum value from the list.

Output:



```
PS F:\Python> python max.py
List Elements are: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
Maximum value in the list: 9
PS F:\Python>
```

TASK 10: **Write a program to display Even Numbers using range() in Python**

Code:

```
# Example 10: Using range with start, stop, and step
start=int(input("Enter the Start Value: "))
stop = int(input("Enter the Stop Value: "))
step = int(input("Enter the Step Value: "))
for num in range(start,stop,step):
print(num, end=' ')
```

Explanation:

1 User Input: The program uses input to prompt the user to enter the start, stop, and step values. The int(...) is used to convert the input to integers.

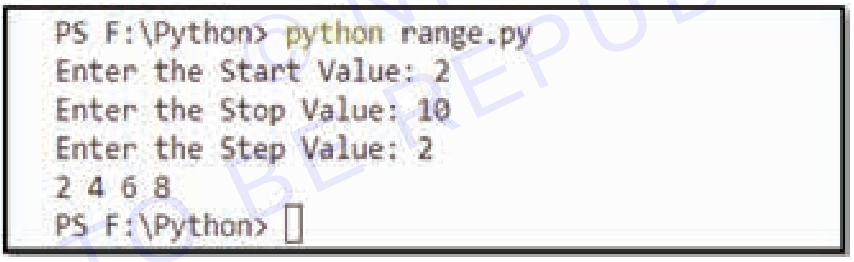
2 Range Function: The range(start, stop, step) function generates a sequence of numbers starting from start, up to (but not including) stop, with a specified step between numbers.

3 For Loop: The for loop iterates over the generated sequence of numbers. For each number, it executes the indented block of code.

4 Print Statement: Inside the loop, each number (num) is printed using the print function. The end=' ' argument ensures that the numbers are printed on the same line, separated by a space.

5 User Interaction: The user interaction allows for a dynamic experience where users can input their desired range and step, and the program displays the sequence accordingly.

Output:



```
PS F:\Python> python range.py
Enter the Start Value: 2
Enter the Stop Value: 10
Enter the Step Value: 2
2 4 6 8
PS F:\Python> 
```

— — — — —

TASK 11: **Write a program that takes two numbers (base and exponent) as input and calculates and prints the result of raising the base to the power of the exponent using the pow function**

Code:

```
# 11. pow
base=int(input("Enter the Base : "))
exponent = int(input("Enter the Exponent : "))
power_result = pow(base, exponent)
print(f"{base} raised to the power of {exponent}: {power_result}")
```

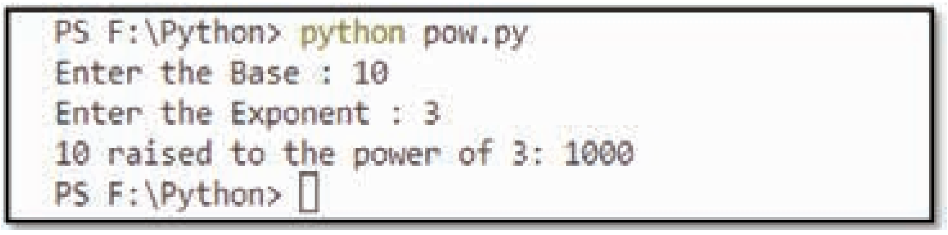
Explanation

1 **User Input:** The program prompts the user to enter the base and exponent values using input. The int(...) converts the entered values to integers.

2 **Pow Function:** The pow(base, exponent) function is used to calculate the power of the base raised to the specified exponent.

3 **Result Display:** The result of the power calculation is stored in the variable power_result and is then printed using the print statement. The formatted string (f"...") allows for a clear and concise display of the calculation.

Output:



```
PS F:\Python> python pow.py
Enter the Base : 10
Enter the Exponent : 3
10 raised to the power of 3: 1000
PS F:\Python> 
```

— — — — —

TASK 12: **Write a program that uses slicing to extract a substring from a given string**

Code:

```
# 12. slice
text = input("Enter the String: ")
print("Given String is ", text)
sliced_text = text[6:11]
print(f"Sliced text: {sliced_text}")
```

Explanation:

1 **User Input:** The program prompts the user to enter a string using input. The entered string is stored in the variable text.

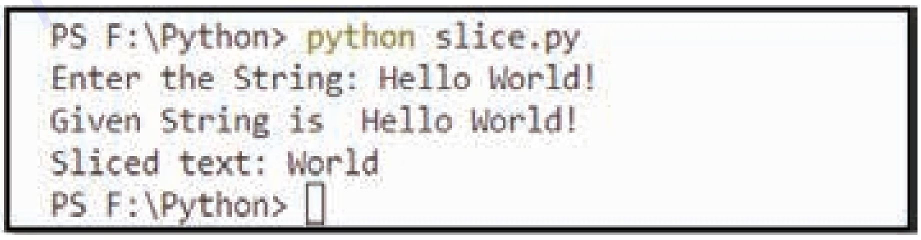
2 **Original Text Display:** The program prints the original string using print("Given String is ", text).

3 **Slicing:** The program uses slicing (text[6:11]) to extract a portion of the string. In Python, slicing is used to create a new sequence (in this case, a substring) by specifying a range of indices.

4 **Result Display:** The sliced text is stored in the variable sliced_text and is then printed using the print statement. The formatted string (f"...") allows for a clear display of the original and sliced text.

This example demonstrates how to use slicing to extract a substring from a user-entered string. The specific range [6:11] extracts characters from the 7th to the 10th positions (Python uses 0-based indexing). Users can input different strings, and the program will display the sliced portion.

Output:



```
PS F:\Python> python slice.py
Enter the String: Hello World!
Given String is  Hello World!
Sliced text: World
PS F:\Python> 
```

— — — — —

TASK 13: **Write a program that converts a list into a tuple using the tuple function**

Code :

```
# 13. tuple
example_list = [1, 2, 3, 4, 5]
print(" Given List is : ", example_list)
tuple_from_list = tuple(example_list)
print(f"Tuple from list: {tuple_from_list}")
```

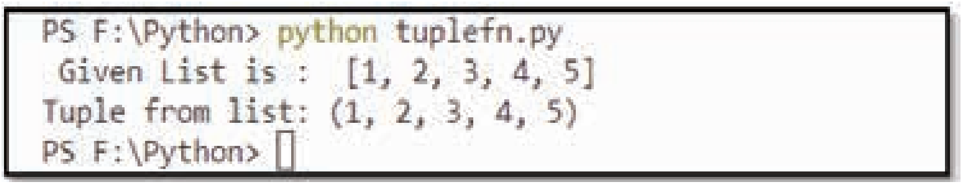
Explanation:

1 **List to Tuple Conversion:** The program creates a list example_list containing integers.

2 **Tuple Creation:** The tuple() function is used to convert the list into a tuple. The resulting tuple is assigned to the variable tuple_from_list.

3 **Display:** The program prints the tuple obtained from the list using the print statement. The formatted string (f"...") is used to include the tuple in the output.

Output:



```
PS F:\Python> python tuplefn.py
 Given List is :  [1, 2, 3, 4, 5]
Tuple from list: (1, 2, 3, 4, 5)
PS F:\Python> 
```

— — — — —

TASK 14 : **Write a program that uses the vars function to display all variables in the program as a dictionary**

Code:

```
# 14. vars
name = "John"
age = 25
country = "USA"
info_dict = vars()
print(f"Variables as dictionary: {info_dict}")
```

Explanation:

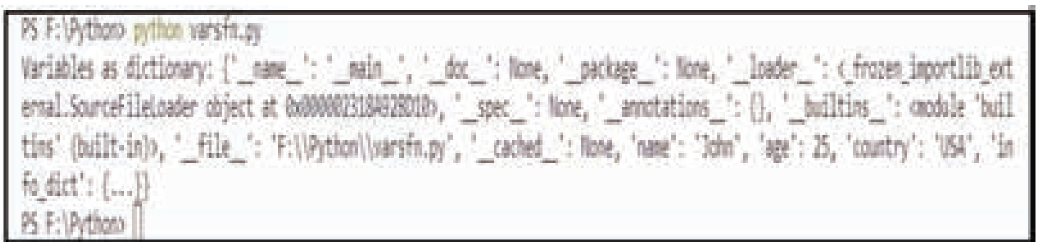
1 Variable Information: The program defines three variables: name, age, and country.

2 vars() Function: The vars() function is used to obtain a dictionary representing the current local symbol table. This includes all variables currently defined.

3 Display: The program prints the dictionary obtained from the local variables using the print statement. The formatted string (f"...") is used to include the dictionary in the output.

These examples showcase how to convert a list to a tuple and how to obtain variable information as a dictionary using the vars() function. The output will display the converted tuple and the dictionary of local variables.

Output:



```
PS F:\Python> python varsfn.py
Variables as dictionary: {'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <_frozen_importlib_ext
ernal.SourceFileLoader object at 0x000002318A928D10>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'bui
tins' (built-in)>, '__file__': 'F:\\Python\\varsfn.py', '__cached__': None, 'name': 'John', 'age': 25, 'country': 'USA', 'in
fo_dict': {...}}
PS F:\Python> 
```

Related Exercises

1 Write a program that takes an ASCII code as input and prints the corresponding character using the chr() function.

2 Write a program that compares two integers and prints whether they are equal, greater, or smaller using the cmp() function.

3 Create a program that takes a Python code snippet as input, compiles it using the compile() function, and then executes it using exec().

4 Write a program that defines a list and uses the dir() function to get its attributes. Print the list of attributes.

5 Develop a program that prompts the user for a mathematical expression, evaluates it using eval(), and prints the result.

6 Create a program that defines a list of numbers and uses the filter() function to obtain only the even numbers.

7 Write a program that takes a string as input, computes its hash using the hash() function, and prints the result.

8 Develop a program that uses the input() function to get the user's name and then prints a personalized greeting.

9 Create a program that takes a string as input and prints its length using the len() function.

10 Write a program that defines some local variables, uses the locals() function to obtain their dictionary, and prints the result.

11 Develop a program that defines a list of numbers, uses the max() function to find the maximum value, and prints the result.

12 Create a program that takes base and exponent values as input, calculates the power using pow(), and prints the result.

13 Write a program that takes a string as input, uses slicing to extract a substring, and prints the result.

14 Develop a program that defines a list and converts it into a tuple using the tuple() function. Print the resulting tuple.

15 Write a program that defines some variables, uses the vars() function to obtain their dictionary, and prints the result.

EXERCISE 133 : Write a python program to read and write into a file

Objectives

At the end of this exercise you shall be able to

- develop python programs to read and write into a file.

Procedure

In Python, file handling is an essential part of programming that allows you to work with files on your computer. You can read from existing files, write to new files, and perform various operations on files. Here's a brief overview of Python file handling:

Opening a File: To open a file, you can use the built-in open() function. It takes two parameters – the file name and the mode (read, write, or append).

```
content = file.read()  # Reads the entire content
line = file.readline()  # Reads one line
lines = file.readlines()  # Reads all lines into a list
```

Common modes:

- "r": Read (default mode).
- "w": Write (creates a new file or truncates an existing file).
- "a": Append (opens a file for appending new content).

Reading from a File: You can read the content of a file using various methods such as read(), readline(), or readlines().

```
content = file.read()  # Reads the entire content
line = file.readline()  # Reads one line
lines = file.readlines()  # Reads all lines into a list
```

Writing to a File: To write content to a file, open the file in write mode ("w") or append mode ("a") and use the write() method.

```
with open("example.txt", "w") as file:
    file.write("Hello, this is a new line.")
```

Closing a File: It's important to close the file after you're done with it. The with statement automatically closes the file when the block is exited.

```
with open("example.txt", "r") as file:
    content = file.read()
# File is automatically closed here
```

— — — — —

TASK 1: **Program code for read mode**

It is a read operation in Python. We open an existing file with the given code and then read it. The code is given below –

```
# Open the file in read mode
with open("file.txt", "r") as fileptr:
# Read the contents of the file
file_content = fileptr.read()
# Print the content
print(file_content)
```

Explanation:

with open("file.txt", "r") as fileptr:

- with: The with statement is used to wrap the execution of a block with methods defined by a context manager. It ensures that the file is properly closed after the block execution, even if an exception occurs.
- open("file.txt", "r"): The open function is used to open a file. The first argument is the filename ("file.txt" in this case), and the second argument is the mode ("r" for read mode).
- as fileptr: It assigns the file object returned by open to the variable fileptr. This variable is used to interact with the file.

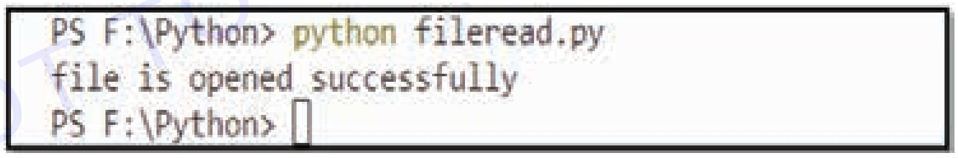
file_content = fileptr.read():

- fileptr.read(): The read method is called on the file object (fileptr). It reads the entire content of the file and returns it as a string, which is then assigned to the variable file_content.

print(file_content):

- print(file_content): This line prints the contents of the file, which were stored in the file_content variable.
- The use of the with statement is good practice because it automatically takes care of closing the file when the indented block is exited, ensuring proper resource management.

Output:



```
PS F:\Python> python fileread.py
file is opened successfully
PS F:\Python>
```

— — — — —

TASK 2: **Open the file in write mode**

```
# Open the file in write mode
with open("file.txt", "w") as fileptr:
# Write content to the file
fileptr.write("Hello, this is a sample line.\n")
fileptr.write("Writing another line to the file.\n")
print("Content has been written to the file.")
```

Explanation:

1 with open("file.txt", "w") as fileptr::

- This line opens the file named "file.txt" in write mode ("w").
- The with statement ensures that the file is properly closed after writing.

2 File Write:

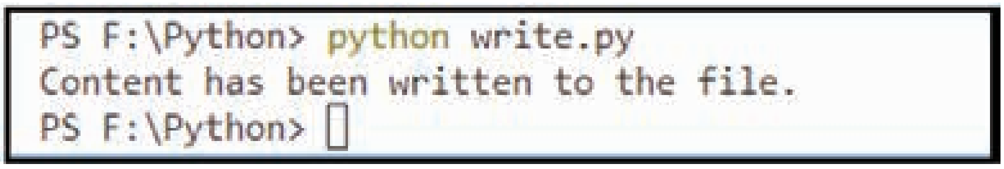
- fileptr.write("Hello, this is a sample line.\n"): Writes the first line to the file.
- fileptr.write("Writing another line to the file.\n"): Writes the second line to the file.
- The "\n" is used to add a newline character at the end of each line.

3 print("Content has been written to the file."):

- This line is not necessary for file writing but serves as a confirmation message that the content has been written.

  After running this code, you should find a file named "file.txt" in the same directory with the specified lines written to it.

Output;

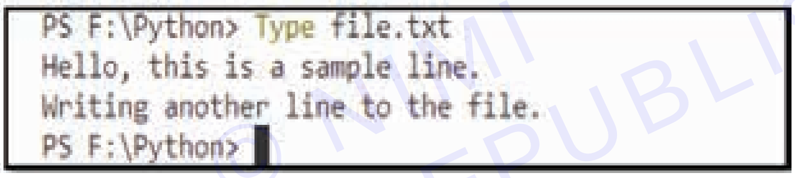


```
PS F:\Python> python write.py
Content has been written to the file.
PS F:\Python>
```

To see the content of the file named file.txt use the following command in the Command Prompt/Terminal.

F:\Python> Type file.txt

You will get the output like this



```
PS F:\Python> Type file.txt
Hello, this is a sample line.
Writing another line to the file.
PS F:\Python>
```

— — — — —

TASK 3 : **Python program that demonstrates file manipulation, including reading, writing, and appending to a file**

Code:

```
# Step 1: Open a file for writing
with open("sample_file.txt", "w") as fileptr:
# Step 2: Write content to the file
fileptr.write("Hello, this is a sample line.\n")
fileptr.write("Writing another line to the file.\n")
# Step 3: Open the same file for reading
with open("sample_file.txt", "r") as fileptr:
# Step 4: Read and print the file content
file_content = fileptr.read()
print("File Content (Read Mode):\n", file_content)
# Step 5: Open the same file for appending
with open("sample_file.txt", "a") as fileptr:
# Step 6: Append more content to the file
fileptr.write("Appending a new line to the file.\n")
# Step 7: Open the file again for reading
```

```
with open("sample_file.txt", "r") as fileptr:
# Step 8: Read and print the updated file content
updated_content = fileptr.read()
print("\nUpdated File Content (Read Mode):\n", updated_content)
```

Explanation:

1 **Opening File for Writing ("w"):**

- with open("sample_file.txt", "w") as fileptr: opens the file "sample_file.txt" in write mode.
- write method is used to write content to the file.

2 **Reading File Content ("r"):**

- with open("sample_file.txt", "r") as fileptr: opens the file in read mode to read its content.
- read method is used to read and print the file content.

3 **Appending to the File ("a"):**

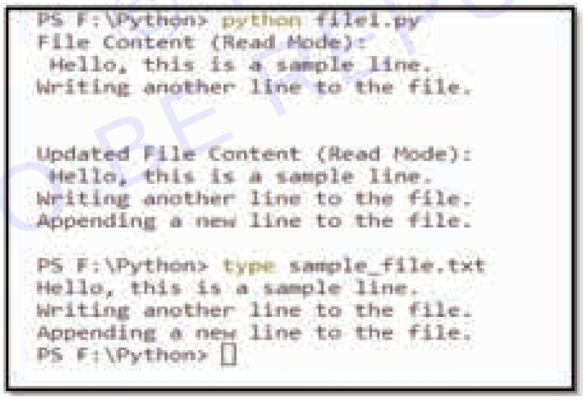
- with open("sample_file.txt", "a") as fileptr: opens the file in append mode.
- write method is used to append content to the file.

4 **Reading Updated File Content ("r"):**

- with open("sample_file.txt", "r") as fileptr: opens the file again in read mode.
- read method is used to read and print the updated file content.

After running this code, you should see the content of "sample_file.txt" displayed, including the appended line.

Output:



```
PS F:\Python> python file1.py
File Content (Read Mode):
 Hello, this is a sample line.
Writing another line to the file.


Updated File Content (Read Mode):
 Hello, this is a sample line.
Writing another line to the file.
Appending a new line to the file.

PS F:\Python> type sample_file.txt
Hello, this is a sample line.
Writing another line to the file.
Appending a new line to the file.
PS F:\Python> 
```

Related Exercises:

1 **File Reading:**

- Write a Python program to read the contents of a file and display them.
- Modify the program to read only the first N lines of the file.

2 **File Writing:**

- Create a Python program to write a list of strings to a file.
- Allow the user to input multiple lines of text and save them to a file.

3 **Appending to a File:**

- Write a program to append new data to an existing file.
- Allow the user to input additional lines and append them to the file.

EXERCISE 134 : Write a python program depicting argument passing and using tuples, dictionaries as arguments

Objectives

At the end of this exercise you shall be able to

- develop python programs depicting argument passing and using tuples, dictionaries as arguments.

Procedure

TASK 1: **Argument Passing with Tuples**

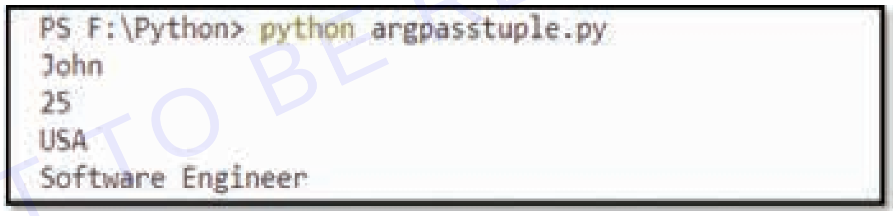
Code:

```
def display_info(*args):
for arg in args:
print(arg)
# Example usage
display_info(“John”, 25, “USA”, “Software Engineer”)
```

Explanation:

- The function display_info accepts variable-length positional arguments using *args.
- It prints each argument passed to the function.

Output:



```
PS F:\Python> python argpasstuple.py
John
25
USA
Software Engineer
```

— — — — —

TASK 2: **Argument Passing with Dictionaries**

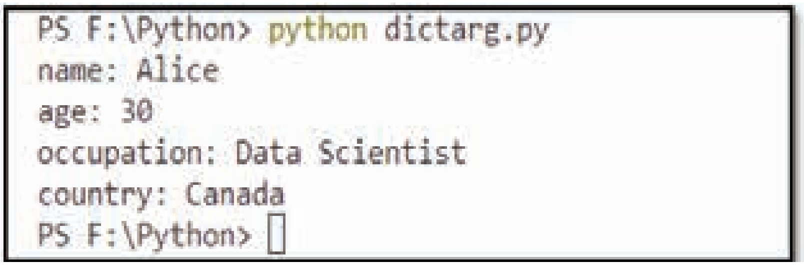
Code:

```
def display_user_info(**kwargs):
for key, value in kwargs.items():
print(f”{key}: {value}”)
# Example usage
display_user_info(name=”Alice”, age=30, occupation=”Data Scientist”, country=”Canada”)
```

Explanation:

- The function display_user_info accepts variable-length keyword arguments using **kwargs.
- It prints each key-value pair passed to the function.

Output:



```
PS F:\Python> python dictarg.py
name: Alice
age: 30
occupation: Data Scientist
country: Canada
PS F:\Python>
```

— — — — —

TASK 3: **Combined Use of Tuples and Dictionaries**

Code:

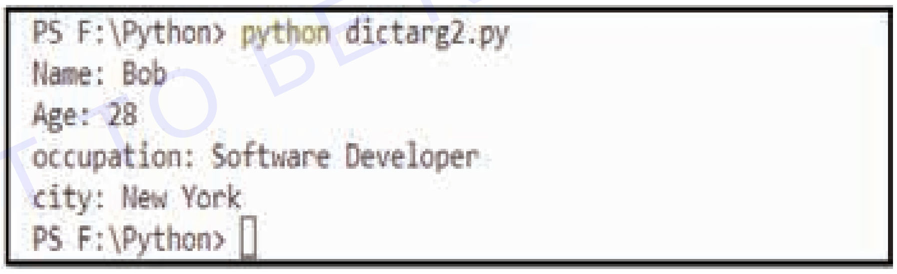
```
def display_person_info(name, age, **additional_info):
print("Name:", name)
print("Age:", age)
for key, value in additional_info.items():
print(f"{key}: {value}")
# Example usage
display_person_info("Bob", 28, occupation="Software Developer", city="New York")
```

Explanation:

- The function display_person_info has both positional and keyword arguments.
- It prints the name and age as positional arguments and any additional information as keyword arguments.

  These examples showcase different ways to use tuples and dictionaries for argument passing in Python functions.

Output:



```
PS F:\Python> python dictarg2.py
Name: Bob
Age: 28
occupation: Software Developer
city: New York
PS F:\Python>
```

Related Exercises:

Exercise 1: Tuples

1 Write a Python function that takes a tuple of numbers as input and returns their sum.

2 Create a program that prompts the user to enter values and stores them in a tuple. Display the tuple.

Exercise 2: Dictionaries

1 Implement a function that accepts a dictionary of student names and their corresponding grades. Print each student's name and grade.

2 Write a program to input key-value pairs from the user and store them in a dictionary. Display the dictionary.

Exercise 3: Combined Use

1 Define a function that takes a person's name and age as positional arguments and any additional information as keyword arguments. Display the name, age, and additional information.

2 Create a program that reads information about books (title, author, year) from the user and stores it in a dictionary. Display the dictionary.

EXERCISE 135 : Construct and analyze code segments that include List comprehensions, tuple, set and Dictionary comprehensions

Objectives

At the end of this exercise you shall be able to

- develop python programs to Construct and analyze code segments that include List comprehensions, tuple, set and Dictionary comprehensions

Procedure

Comprehensions in Python provide a concise way to create and manipulate sequences (lists, sets, dictionaries, etc.) based on existing sequences. There are mainly three types of comprehensions: list comprehensions, set comprehensions, and dictionary comprehensions.

1 **List Comprehensions:**

List comprehensions allow you to create new lists by applying an expression to each item in an existing iterable (list, tuple, string, etc.). They are a concise alternative to using loops.

Syntax:

new_list = [expression for item in iterable if condition]

— — — — —

TASK 1: **Squares of Numbers**

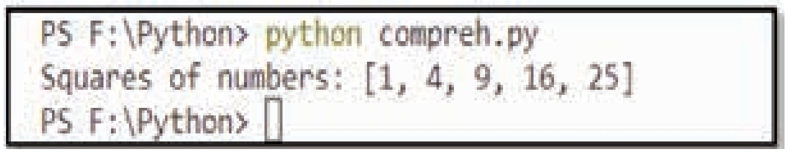
Code:

```
squares = [x**2 for x in range(1, 6)]
print("Squares of numbers:", squares)
```

Explanation:

This list comprehension creates a list of squares of numbers from 1 to 5.

Output:



```
PS F:\Python> python compreh.py
Squares of numbers: [1, 4, 9, 16, 25]
PS F:\Python> 
```

— — — — —

TASK 2: **Even Numbers**

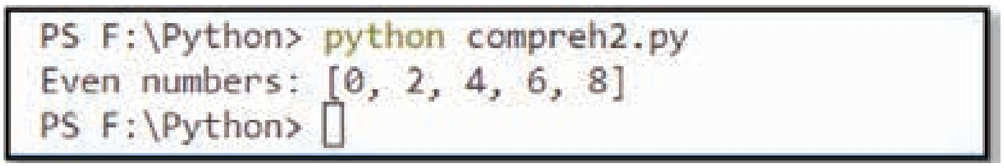
Code:

```
even_numbers = [x for x in range(10) if x % 2 == 0]
print("Even numbers:", even_numbers)
```

Explanation:

Explanation: This list comprehension generates a list of even numbers between 0 and 9.

Output:



```
PS F:\Python> python compreh2.py
Even numbers: [0, 2, 4, 6, 8]
PS F:\Python> 
```

Tuple Comprehensions:

1 Tuple comprehensions are similar to list comprehensions but create tuples instead. The syntax is the same as for list comprehensions.

— — — — —

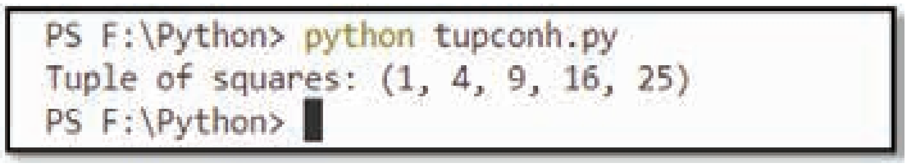
TASK 3: **Squares of Numbers**

Code:

```
tuple_of_squares = tuple(x**2 for x in range(1, 6))
print("Tuple of squares:", tuple_of_squares)
```

Explanation: This tuple comprehension creates a tuple of squares of numbers from 1 to 5.

Output :



```
PS F:\Python> python tupconh.py
Tuple of squares: (1, 4, 9, 16, 25)
PS F:\Python>
```

— — — — —

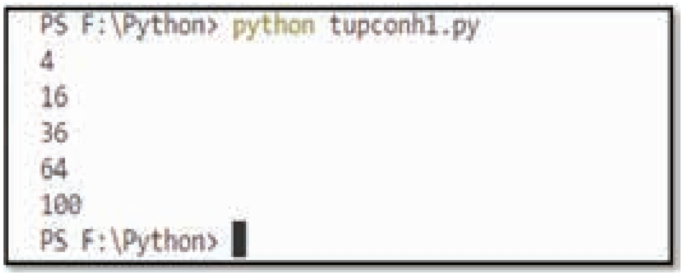
TASK 4: **Squares of even numbers from 1 to 10**

Code:

```
# Generating a tuple of squares of even numbers from 1 to 10
even_squares = (x**2 for x in range(1, 11) if x % 2 == 0)
# Iterating over the tuple of even squares
for square in even_squares:
    print(square)
```

Explanation :

- In this example, the generator expression (x**2 for x in range(1, 11) if x % 2 == 0) generates a tuple of squares of even numbers from 1 to 10.
- When you need to iterate over the items of the tuple, you can use a for loop or convert the generator expression into a tuple using the tuple() function:
- This approach is memory-efficient because it doesn't create the entire tuple in memory at once. Instead, it generates the items on-the-fly as needed.
- While tuple comprehensions don't exist explicitly in Python, you can achieve similar functionality using generator expressions to produce tuples.



```
PS F:\Python> python tupconh1.py
4
16
36
64
100
PS F:\Python>
```

Set Comprehensions:

A set is an unordered and mutable collection of unique elements. It is defined using curly braces {}. Sets are widely used when the presence of unique elements is crucial, and the order of elements doesn't matter.

Set comprehensions are similar to list comprehensions but create sets instead. They use curly braces {}.

Syntax:

new_set = {expression for item in iterable if condition}

— — — — —

TASK 5: **Cube of numbers from 1 to 5**

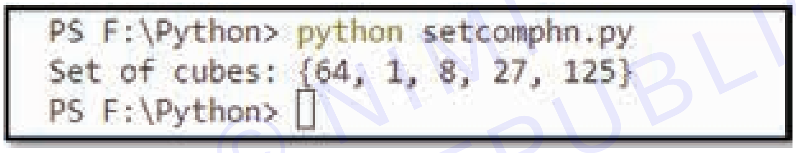
Code:

```
set_of_cubes = {x**3 for x in range(1, 6)}
 print("Set of cubes:", set_of_cubes)
```

Explanation:

- range(1, 6) generates numbers from 1 to 5 (inclusive).
- x**3 computes the cube of each number in the range.
- The set comprehension {x**3 for x in range(1, 6)} collects these cube values into a set.

  After executing the code, the set set_of_cubes will contain the cubes of numbers 1 through 5. The output will look like:



```
PS F:\Python> python setcomphn.py
Set of cubes: {64, 1, 8, 27, 125}
PS F:\Python> 
```

— — — — —

TASK 6: **Set Comprehension with Conversion**

Code:

```
# Create a set of uppercase characters in a string
text = "hello World"
print("Given Text is : ",text)
uppercase_set = {char.upper() for char in text if char.isalpha()}
print("Set of uppercase characters:", uppercase_set)
```

Explanation:

1 text = "hello World": Initializes a string variable named text with the value "hello World".

2 print("Given Text is : ", text): Prints the original text.

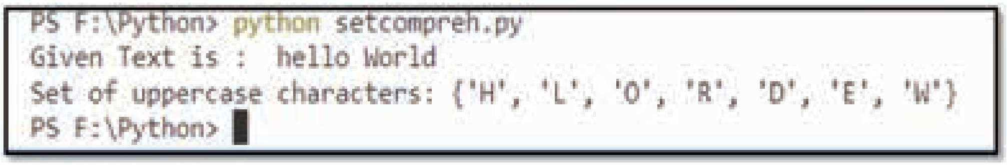
3 uppercase_set = {char.upper() for char in text if char.isalpha()}:

- This is a set comprehension.
- for char in text: Iterates over each character in the text.
- if char.isalpha(): Filters out non-alphabetic characters.
- char.upper(): Converts each character to uppercase.
- The result is a set containing the uppercase alphabetic characters from the original text.

4 print("Set of uppercase characters:", uppercase_set): Prints the set of uppercase characters obtained from the text.

So, the complete code takes a string, filters out non-alphabetic characters, converts the remaining characters to uppercase, and creates a set of those uppercase characters. The output displays the original text and the resulting set of uppercase characters.

Output:



```
PS F:\Python> python setcompreh.py
Given Text is :  hello World
Set of uppercase characters: {'H', 'L', 'O', 'R', 'D', 'E', 'W'}
PS F:\Python>
```

5 Dictionary Comprehensions:

Dictionary comprehensions allow you to create dictionaries. They use key-value pair expressions.

— — — — —

TASK 7: **Original list of fruits and their lengths**

# Original list of fruits and their lengths

fruits = [‘apple’, ‘banana’, ‘orange’, ‘kiwi’, ‘grape’]

fruit_lengths = {fruit: len(fruit) for fruit in fruits}

# Print the original list

print(“Original List of Fruits:”, fruits)

# Print the dictionary created using a comprehension

print(“Dictionary of Fruit Lengths:”, fruit_lengths)

Explanation:

1 fruits = [‘apple’, ‘banana’, ‘orange’, ‘kiwi’, ‘grape’]: Initializes a list of fruits.

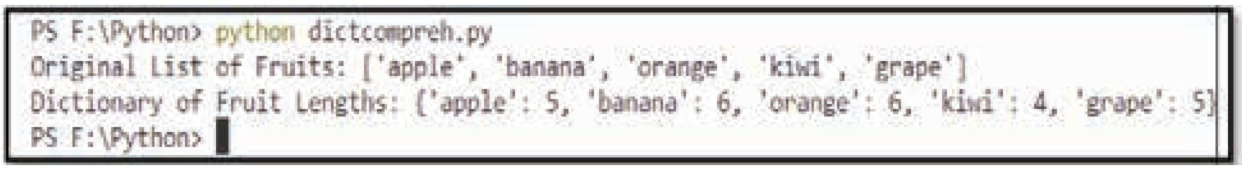
2 fruit_lengths = {fruit: len(fruit) for fruit in fruits}:

- This is a dictionary comprehension.
- for fruit in fruits: Iterates over each fruit in the list.
- {fruit: len(fruit)}: Creates a key-value pair in the dictionary, where the key is the fruit, and the value is the length of the fruit.

3 print(“Original List of Fruits:”, fruits): Prints the original list of fruits.

4 print(“Dictionary of Fruit Lengths:”, fruit_lengths): Prints the dictionary created using the comprehension.

Output:



```
PS F:\Python> python dictcompreh.py
Original List of Fruits: ['apple', 'banana', 'orange', 'kiwi', 'grape']
Dictionary of Fruit Lengths: {'apple': 5, 'banana': 6, 'orange': 6, 'kiwi': 4, 'grape': 5}
PS F:\Python>
```

— — — — —

TASK 8: **Temperatures in Celsius**

# Original list of temperatures in Celsius

temperatures_celsius = {‘city1’: 25, ‘city2’: 30, ‘city3’: 22, ‘city4’: 18}

# Dictionary comprehension to convert temperatures to Fahrenheit

temperatures_fahrenheit = {city: (temp * 9/5) + 32 for city, temp in temperatures_celsius.items()}

# Print the original temperatures

print(“Original Temperatures (Celsius):”, temperatures_celsius)

Print the converted temperatures

print("Temperatures (Fahrenheit):", temperatures_fahrenheit)

Explanation:

1 Temperatures_celsius = {'city1': 25, 'city2': 30, 'city3': 22, 'city4': 18}: Initializes a dictionary with city names as keys and temperatures in Celsius as values.

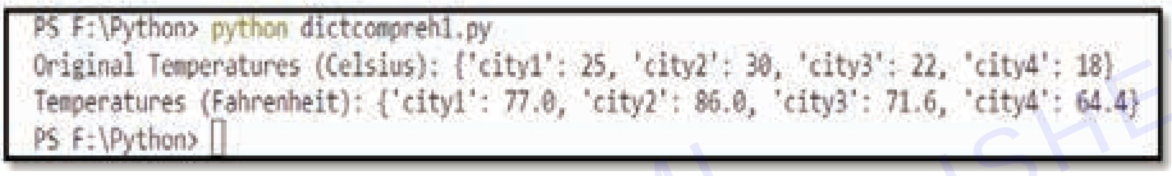
2. Temperatures_fahrenheit = {city: (temp * 9/5) + 32 for city, temp in temperatures_celsius.items()}:

- This is a dictionary comprehension.
- For city, temp in temperatures_celsius.items(): Iterates over each city and temperature in the original dictionary.
- {city: (temp * 9/5) + 32}: Creates a new key-value pair in the dictionary, where the key is the city, and the value is the temperature converted to Fahrenheit.

3 Print("Original Temperatures (Celsius):", temperatures_celsius): Prints the original temperatures in Celsius.

4 Print("Temperatures (Fahrenheit):", temperatures_fahrenheit): Prints the dictionary of temperatures converted to Fahrenheit.

Output:



```
PS F:\Python> python dictcompreh1.py
Original Temperatures (Celsius): {'city1': 25, 'city2': 30, 'city3': 22, 'city4': 18}
Temperatures (Fahrenheit): {'city1': 77.0, 'city2': 86.0, 'city3': 71.6, 'city4': 64.4}
PS F:\Python> []
```

Related Exercises :

List Comprehensions:

1 Create a list comprehension that generates the squares of numbers from 1 to 10.

2 Generate a list of even numbers from 1 to 20 using list comprehension.

3 Given a list of strings, create a new list containing the lengths of each string.

Tuple Comprehensions:

1 Generate a tuple comprehension that contains cubes of numbers from 1 to 5.

2 Create a tuple of characters at odd indices from a given string.

Set Comprehensions:

1 Generate a set comprehension that contains unique squares of numbers from 1 to 10.

2 Create a set of vowels present in a given string.

Dictionary Comprehensions:

1 Given two lists, create a dictionary where elements from the first list are keys, and elements from the second list are values.

2 Count the frequency of each character in a string and create a dictionary using dictionary comprehension.

3 Given a dictionary containing temperatures in Celsius, create a new dictionary with temperatures converted to Fahrenheit.

EXERCISE 136 : Perform basic operations using built-in modules

Objectives

At the end of this exercise you shall be able to

- develop python programs to Perform basic operations using built-in modules.

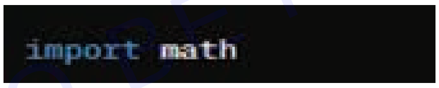
Procedure

TASK 1: **Math Module**

Code:

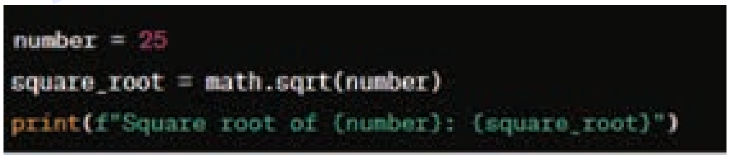
```
import math
# Example 1: Square Root
number = 25
square_root = math.sqrt(number)
print(f"Square root of {number}: {square_root}")
# Example 2: Power
base = 2
exponent = 3
power_result = math.pow(base, exponent)
print(f"{base} raised to the power of {exponent}: {power_result}")
```

Explanation:



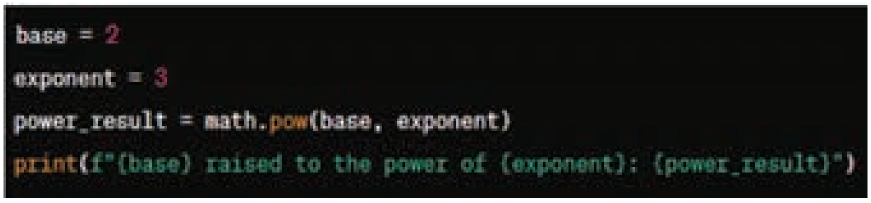
```
import math
```

- This line imports the math module, which provides access to various mathematical functions and constants in Python.



```
number = 25
square_root = math.sqrt(number)
print(f"Square root of {number}: {square_root}")
```

- In this section, the code calculates the square root of the number 25 using the sqrt() function from the math module. The result is stored in the variable square_root, and then it is printed out with an f-string.

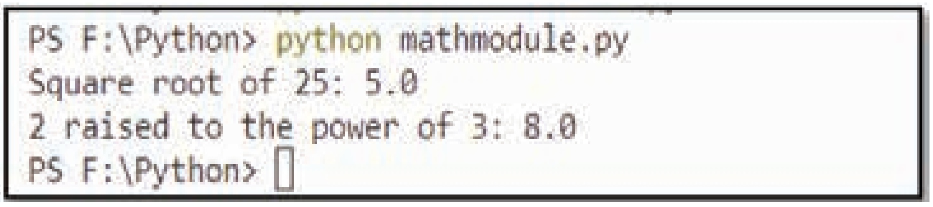


```
base = 2
exponent = 3
power_result = math.pow(base, exponent)
print(f"{base} raised to the power of {exponent}: {power_result}")
```

- Here, the code calculates 2 raised to the power of 3 using the pow() function from the math module. The result is stored in the variable power_result, and then it is printed out with an f-string.

  In summary, the code demonstrates how to use the sqrt() function to calculate square roots and the pow() function to calculate powers with the help of the math module.

Output:


```
PS F:\Python> python mathmodule.py
Square root of 25: 5.0
2 raised to the power of 3: 8.0
PS F:\Python> 
```

— — — — —

TASK 2: **Calendar Module**

import calendar

# Example 6: Display Calendar

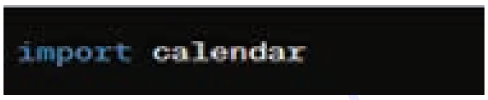
year = int(input(“Enter the Year: “))

month = int(input(“Enter the Month: “))

cal = calendar.month(year, month)

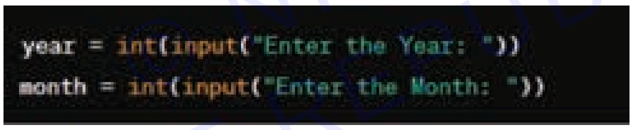
print(f”Calendar for {calendar.month_name[month]} {year}:\n{cal}”)

1 **Importing the calendar module:**


```
import calendar
```

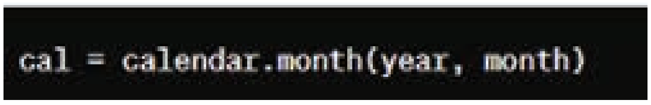
- This line imports the built-in calendar module, which provides functions to work with calendars.

2 **Getting User Input for Year and Month:**


```
year = int(input("Enter the Year: "))
month = int(input("Enter the Month: "))
```

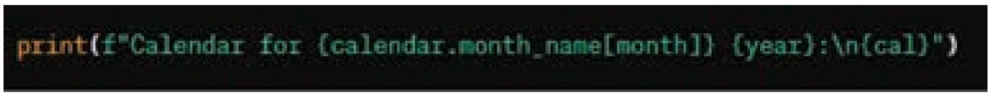
- These lines prompt the user to enter the year and month for which they want to display the calendar.
- The input function takes user input as a string, and int is used to convert it to integers.

3 **Generating Calendar:**


```
cal = calendar.month(year, month)
```

- This line uses the month function from the calendar module to generate a formatted calendar as a string for the specified year and month.
- The result is stored in the variable cal

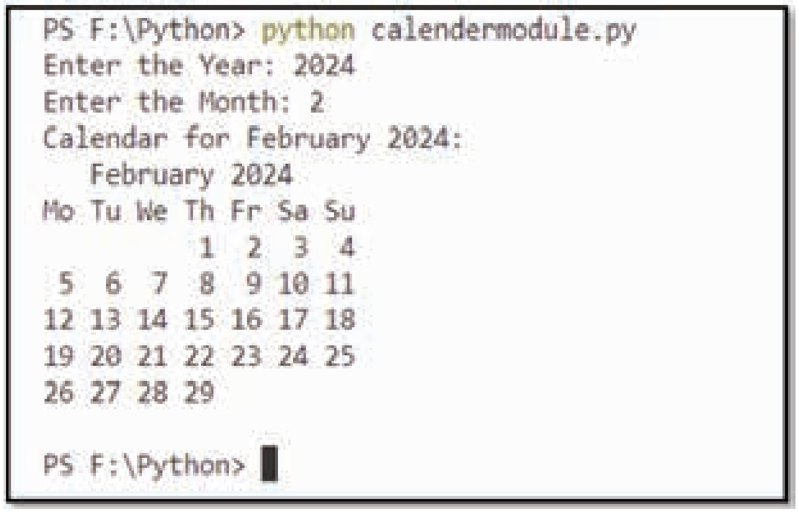
4 **Printing the Calendar:**


```
print(f"Calendar for {calendar.month_name[month]} {year}:\n{cal}")
```

- This line prints the formatted calendar.
- The f-string includes the name of the entered month and the entered year for clarity.
- The calendar.month_name[month] is used to get the name of the month based on the entered numeric value.
- The formatted calendar string (cal) is printed.

  In summary, the code takes user input for the year and month, uses the calendar module to generate a formatted calendar, and prints the calendar with additional information for better readability.

Output:



```
PS F:\Python> python calendermodule.py
Enter the Year: 2024
Enter the Month: 2
Calendar for February 2024:
   February 2024
Mo Tu We Th Fr Sa Su
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29

PS F:\Python> █
```

— — — — —

TASK 3: **Datetime Module**

Code:

```
from datetime import datetime, timedelta
# Example of working with dates and times
current_time = datetime.now()  # Get current date and time
formatted_time = current_time.strftime("%Y-%m-%d %H:%M:%S")  # Format the time as a string
# Example of adding and subtracting time
one_day_later = current_time + timedelta(days=1)
two_hours_earlier = current_time - timedelta(hours=2)
print(formatted_time)
print(one_day_later.strftime("%Y-%m-%d %H:%M:%S"))
print(two_hours_earlier.strftime("%Y-%m-%d %H:%M:%S"))
```

```
from datetime import datetime, timedelta
```

- This line imports the datetime class and the timedelta class from the datetime module. the datetime class is used to represent dates and times, and the timedelta class is used to represent the difference between two dates or times.

```
current_time = datetime.now()  # Get current date and time
```

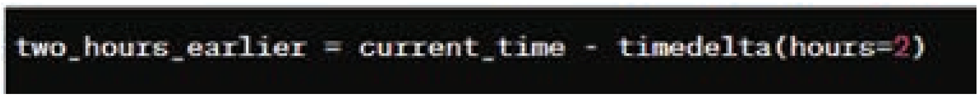
- This line creates a datetime object representing the current date and time.

```
formatted_time = current_time.strftime("%Y-%m-%d %H:%M:%S")
```

- The strftime method is used to format the datetime object as a string according to the specified format. In this case, it formats the date and time in the "Year-Month-Day Hour:Minute:Second" format.

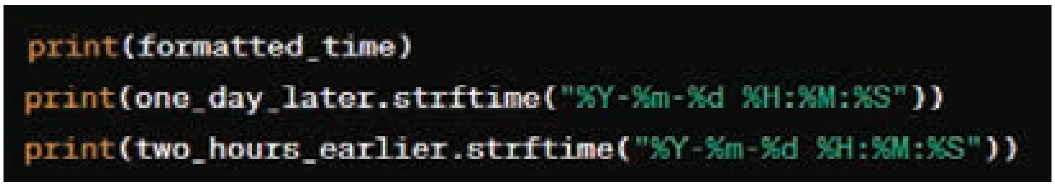
```
one_day_later = current_time + timedelta(days=1)
```

- This line creates a new datetime object representing a date and time one day later than the current time.



```
two_hours_earlier = current_time - timedelta(hours=2)
```

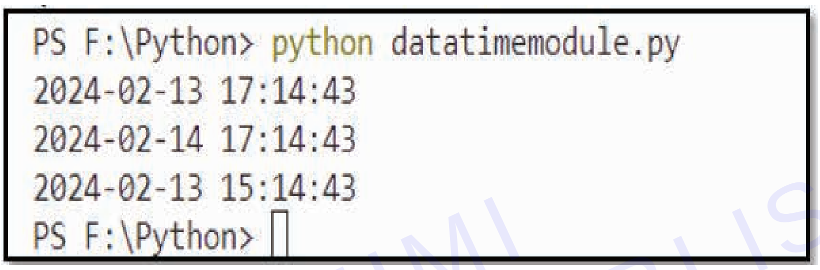
- This line creates a new datetime object representing a date and time two hours earlier than the current time
- Finally, the code prints out the formatted current time, the time one day later, and the time two hours earlier.



```
print(formatted_time)
print(one_day_later.strftime("%Y-%m-%d %H:%M:%S"))
print(two_hours_earlier.strftime("%Y-%m-%d %H:%M:%S"))
```

In summary, the code demonstrates working with dates and times, formatting them as strings, and performing basic arithmetic operations on them using the datetime and timedelta classes.

Output:



```
PS F:\Python> python datatimemodule.py
2024-02-13 17:14:43
2024-02-14 17:14:43
2024-02-13 15:14:43
PS F:\Python> 
```

Related Exercises:

1 Write a program that calculates the area of a circle given its radius using the math module.

2 Write a program that displays the current date and time using the datetime module.

3 Write a program that displays the calendar using the calendar module.

EXERCISE 137 : Solve complex computing problems by using builtin modules

Objectives

At the end of this exercise you shall be able to

- develop python programs to Solve complex computing problems by using builtin modules.

Procedure

TASK 1: **Finding the Square Root with the math Module**

Code:

```
import math
number = float(input("Enter a number: "))
square_root = math.sqrt(number)
print(f"The square root of {number} is {square_root}")
```

Explanation:

1 **Importing the math Module:**

- import math: This line imports the math module, which provides access to various mathematical functions and constants.

2 **Getting User Input:**

- number = float(input("Enter a number: ")): This line prompts the user to enter a number. The input is captured as a string using the input() function and then converted to a floating-point number using float().

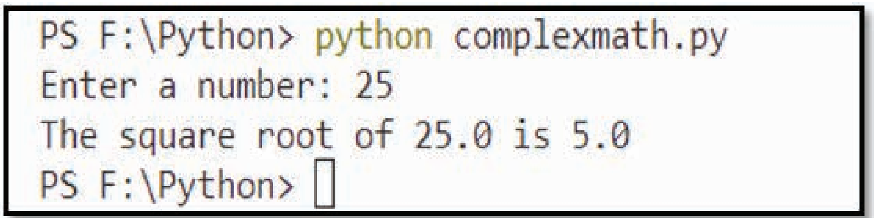
3 **Calculating the Square Root:**

- square_root = math.sqrt(number): This line calculates the square root of the input number using the sqrt() function from the math module.

4. **Displaying the Result:**

- print(f"The square root of {number} is {square_root}"): This line prints the calculated square root using an f-string. It displays the original number entered by the user along with its square root.

Output:



```
PS F:\Python> python complexmath.py
Enter a number: 25
The square root of 25.0 is 5.0
PS F:\Python> 
```

— — — — —

TASK 2: **Calculating Trigonometric Functions with the math Module**

Code:

```
import math
angle = float(input("Enter an angle in degrees: "))
sin_value = math.sin(math.radians(angle))
cos_value = math.cos(math.radians(angle))
```

print(f"The sin of {angle} degrees is {sin_value}")

print(f"The cos of {angle} degrees is {cos_value}")

Explanation:

1 Importing the math Module:

- import math: This line imports the math module, which provides mathematical functions and constants.

2 Getting User Input:

- angle = float(input("Enter an angle in degrees: ")): This line prompts the user to enter an angle in degrees, captures the input as a string, and converts it to a floating-point number using float().

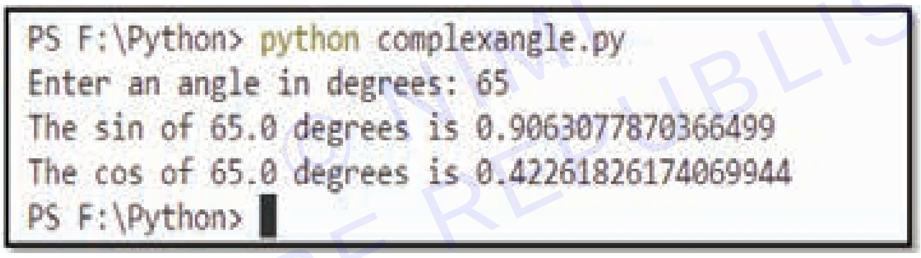
3 Calculating Sine and Cosine:

- sin_value = math.sin(math.radians(angle)): This line calculates the sine of the angle. The math.radians() function converts the angle from degrees to radians before applying the math.sin() function.
- cos_value = math.cos(math.radians(angle)): This line calculates the cosine of the angle in a similar way.

4 Displaying the Results:

- print(f"The sin of {angle} degrees is {sin_value}"): This line prints the calculated sine value for the entered angle.
- print(f"The cos of {angle} degrees is {cos_value}"): This line prints the calculated cosine value for the entered angle.

Output:



```
PS F:\Python> python complexangle.py
Enter an angle in degrees: 65
The sin of 65.0 degrees is 0.9063077870366499
The cos of 65.0 degrees is 0.42261826174069944
PS F:\Python>
```

— — — — —

TASK 3: **Generating Random Numbers with the random Module**

Code:

import random

random_number = random.randint(1, 100) # Generate a random integer between 1 and 100

print(f"Random number: {random_number}")

Explanation:

1 Importing the random Module:

- import random: This line imports the random module, which provides functions for generating pseudo-random numbers.

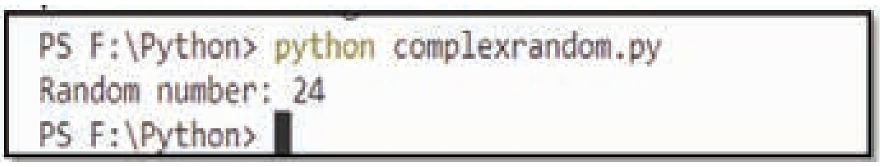
2 Generating a Random Integer:

- random_number = random.randint(1, 100): This line generates a random integer between 1 and 100 (inclusive) using the randint() function from the random module. The result is stored in the variable random_number.

3 Displaying the Result:

- print(f"Random number: {random_number}"): This line prints the generated random number to the console using an f-string.

Output:



```
PS F:\Python> python complexrandom.py
Random number: 24
PS F:\Python>
```

— — — — —

TASK 4: **Manipulating Dates and Times with the datetime Module**

Code:

```
from datetime import datetime, timedelta
current_time = datetime.now()
one_week_later = current_time + timedelta(weeks=1)
print(f"Current time: {current_time}")
print(f"One week later: {one_week_later}")
```

Explanation:

1 **Importing the datetime Module:**

- from datetime import datetime, timedelta: This line imports the datetime module, which provides classes for working with dates and times, and the timedelta class, which represents the difference between two dates or times.

2 **Getting the Current Time:**

- current_time = datetime.now(): This line gets the current date and time and assigns it to the variable current_time using the now() method of the datetime class.

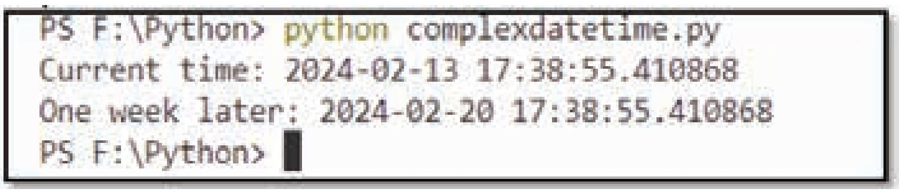
3 **Calculating One Week Later:**

- one_week_later = current_time + timedelta(weeks=1): This line adds one week (7 days) to the current time using the timedelta class. The result is stored in the variable one_week_later.

4 **Displaying the Results:**

- print(f"Current time: {current_time}"): This line prints the current time to the console using an f-string.
- print(f"One week later: {one_week_later}"): This line prints the calculated time one week later to the console using an f-string.

Output:



```
PS F:\Python> python complexdatetime.py
Current time: 2024-02-13 17:38:55.410868
One week later: 2024-02-20 17:38:55.410868
PS F:\Python>
```

Related Exercises

1 Write a Python program that generates and prints a random number between 1 and 50 using the random module

2 Create a program that takes user input for a temperature in Celsius and converts it to Fahrenheit using the appropriate formula. Utilize the math module for mathematical operations.

3 Develop a program that prompts the user to enter a specific date and then calculates and displays the date and time exactly one month later using the datetime module.

4 Build a Python program that takes an angle in degrees as input and calculates and prints the sine, cosine, and tangent values using functions from the math module.

5 Write a program that creates a new text file and writes the current date and time (timestamp) to it. Utilize the datetime module for timestamp generation.