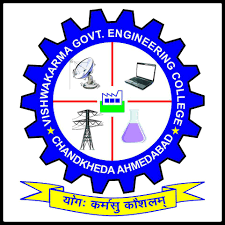
**A Laboratory Manual for**

**Object Oriented Programming -I(3140705)**

**B.E. Semester 4th**

**(Information Technology)**

**Institute logo**

****

**Vishwakarma Government Engineering College, Ahmedabad**

**Gujarat**



**Directorate of Technical Education,**

**Gandhinagar, Gujarat**

**Vishwakarma Government Engineering College,Ahmedabad**

**Certificate**

This is to certify that Mr. /Ms. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ Enrollment No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of B.E. Semester \_\_\_\_\_Computer Engineering of this Institute (GTU Code: 017 ) has satisfactorily completed the Practical / Tutorial work for the subject **Object Oriented Programming-I(3140705)** for the academic year 2023-24.

Place: \_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_

**Name and Sign of Faculty member**

**Head of the Department**

**Preface**

Main motto of any laboratory/practical/field work is for enhancing required skills as well as creating ability amongst students to solve real time problem by developing relevant competencies in psychomotor domain. By keeping in view, GTU has designed competency focused outcome-based curriculum for engineering degree programs where sufficient weightage is given to practical work. It shows importance of enhancement of skills amongst the students and it pays attention to utilize every second of time allotted for practical amongst students, instructors and faculty members to achieve relevant outcomes by performing the experiments rather than having merely study type experiments. It is must for effective implementation of competency focused outcome-based curriculum that every practical is keenly designed to serve as a tool to develop and enhance relevant competency required by the various industry among every student. These psychomotor skills are very difficult to develop through traditional chalk and board content delivery method in the classroom. Accordingly, this lab manual is designed to focus on the industry defined relevant outcomes, rather than old practice of conducting practical to prove concept and theory.

By using this lab manual students can go through the relevant theory and procedure in advance before the actual performance which creates an interest and students can have basic idea prior to performance. This in turn enhances pre-determined outcomes amongst students. Each experiment in this manual begins with competency, industry relevant skills, course outcomes as well as practical outcomes (objectives). The students will also achieve safety and necessary precautions to be taken while performing practical.

This manual also provides guidelines to faculty members to facilitate student centric lab activities through each experiment by arranging and managing necessary resources in order that the students follow the procedures with required safety and necessary precautions to achieve the outcomes. It also gives an idea that how students will be assessed by providing rubrics.

Java is a multi-platform, object-oriented, and network-centric language that can be used as a platform. It is a fast, secure, reliable programming language for coding everything from mobile apps and enterprise software to big data applications and server-side technologies. Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages.

Utmost care has been taken while preparing this lab manual however always there is chances of improvement. Therefore, we welcome constructive suggestions for improvement and removal of errors if any.

**Practical – Course Outcome matrix**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Course Outcomes (COs):**   1. Use various Java constructs, features and libraries for simple problems. 2. Demonstrate how to define and use classes, interfaces, create objects and methods, how to override and overload methods, compile and execute programs. 3. Write a program using exception handling, multithreading with synchronization. 4. Write a program using Files, binary I/O, collection Frameworks for a given problem. 5. Design and develop GUI based applications in a group using modern tools and frameworks. | | | | | | |
| **Sr. No.** | **Objective(s) of Experiment** | **CO1** | **CO2** | **CO3** | **CO4** | **CO5** |
| 1. | To learn basic java programming constructs. | **√** |  |  |  |  |
| 2. | To learn Arrays and Strings in Java. | **√** |  |  |  |  |
| 3. | To implement basic object-oriented concepts. |  | **√** |  |  |  |
| 4. | To implement inheritance and object-oriented concepts. |  | **√** |  |  |  |
| 5. | To demonstrate the use of abstract classes and interfaces. |  | **√** |  |  |  |
| 6. | To implement packages and exception handling in JAVA application. |  |  | **√** |  |  |
| 7. | To demonstrate I/O from files. |  |  |  | **√** |  |
| 8. | To learn JAVA FX UI Controls. |  |  |  |  | **√** |
| 9. | To implement event handling and animation. |  |  |  |  | **√** |
| 10. | To learn recursion and generics. |  |  |  | **√** |  |
| 11. | To demonstrate the use of Collection framework. |  |  |  | **√** |  |
| 12. | To demonstrate the use of multithreading. |  |  | **√** |  |  |

**Industry Relevant Skills**

The following industry relevant competency is expected to be developed in the student by undertaking the practical work of this laboratory.

1. Object oriented application development
2. Networking application development
3. GUI based application development

**Guidelines for Faculty members**

1. Teacher should provide the guideline with demonstration of practical to the students with all features.
2. Teacher shall explain basic concepts/theory related to the experiment to the students before starting of each practical
3. Involve all the students in performance of each experiment.
4. Teacher is expected to share the skills and competencies to be developed in the students and ensure that the respective skills and competencies are developed in the students after the completion of the experimentation.
5. Teachers should give opportunity to students for hands-on experience after the demonstration.
6. Teacher may provide additional knowledge and skills to the students even though not covered in the manual but are expected from the students by concerned industry.
7. Give practical assignment and assess the performance of students based on task assigned to check whether it is as per the instructions or not.
8. Teacher is expected to refer complete curriculum of the course and follow the guidelines for implementation.

**Instructions for Students**

1. Students are expected to carefully listen to all the theory classes delivered by the faculty members and understand the COs, content of the course, teaching and examination scheme, skill set to be developed etc.
2. Students shall organize the work in the group and make record of all observations.
3. Students shall develop maintenance skill as expected by industries.
4. Student shall attempt to develop related hand-on skills and build confidence.
5. Students shall make a small project/application in Java.
6. Student shall develop the habits of evolving more ideas, innovations, skills etc. apart from those included in scope of manual.
7. Student shall refer technical magazines and books.
8. Student should develop a habit of submitting the experimentation work as per the schedule and s/he should be well prepared for the same.

**Common Safety Instructions**

Students are expected to

1. Switch on the PC carefully (not to use wet hands)
2. Shutdown the PC properly at the end of your Lab
3. Carefully Handle the peripherals (Mouse, Keyboard, Network cable etc)
4. Use Laptop in lab after getting permission from Teacher

**Index**

**(Progressive Assessment Sheet)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sr. No. | Objective(s) of Experiment | Page No. | Date of performance | Date of submission | Assessment  Marks | Sign. of  Teacher with date | Remarks |
| 1. | To learn basic java programming constructs. |  |  |  |  |  |  |
| 2. | To learn Arrays and Strings in Java. |  |  |  |  |  |  |
| 3. | To implement basic object-oriented concepts. |  |  |  |  |  |  |
| 4. | To implement inheritance and object-oriented concepts. |  |  |  |  |  |  |
| 5. | To demonstrate the use of abstract classes and interfaces. |  |  |  |  |  |  |
| 6. | To implement packages and exception handling in JAVA application. |  |  |  |  |  |  |
| 7. | To demonstrate I/O from files. |  |  |  |  |  |  |
| 8. | To learn JAVA FX UI Controls. |  |  |  |  |  |  |
| 9. | To implement event handling and animation. |  |  |  |  |  |  |
| 10. | To learn recursion and generics. |  |  |  |  |  |  |
| 11. | To demonstrate the use of Collection framework. |  |  |  |  |  |  |
| 12. | To demonstrate the use of multithreading. |  |  |  |  |  |  |
| Total | | | | |  |  |  |

1. **COURSE OUTCOMES**

After learning the course, the students should be able to:

1. Use various Java constructs, features and libraries for simple problems.
2. Demonstrate how to define and use classes, interfaces, create objects and methods, how to override and overload methods, compile and execute programs.
3. Write a program using exception handling, multithreading with synchronization.
4. Write a program using Files, binary I/O, collection Frameworks for a given problem.
5. Design and develop GUI based applications in a group using modern tools and frameworks.
6. **TEACHING AND EXAMINATION SCHEME**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Teaching Scheme** | | | **Credits** | **Examination Marks** | | | | **Total Marks** |
| L | T | P | C | **Theory Marks** | | **Practical Marks** | |
| ESE (E) | PA (M) | ESE (V) | PA (I) |
| 4 | 0 | 2 | 5 | 70 | 30 | 30 | 20 | 150 |

1. **SUGGESTED LEARNING RESOURCES**

**Reference Books**:

1. Intro to Java Programming, 10th edition, Y.Daniel Liang, Pearson
2. Object oriented programming with Java , RajkumarBuyya,SThamaraiSelvi, Xingchen Chu, McGrawHill
3. Programming in Java, SachinMalhotra, SaurabhChoudhary, Oxford
4. Programming with JAVA , E Balagurusamy, McGrawHill
5. CORE JAVA volume -I Cay Horstmann, Pearson

**Major Equipment:** Computer, Laptop

**List of Open Source Software/learning website:**

https://docs.oracle.com/javase/tutorial/java/index.html

<https://www.tutorialspoint.com/JAVA/>

<https://dev.java/learn/>

https://www.codecademy.com/learn/learn-java

https://www.w3schools.com/java/

Java:

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.

**OpenJDK:**

OpenJDK (Open Java Development Kit) is a free and open-source implementation of the Java Platform, Standard Edition (Java SE). It is the result of an effort Sun Microsystems began in 2006. The implementation is licensed under the GPL-2.0-only with a linking exception. Were it not for the GPL linking exception, components that linked to the Java class library would be subject to the terms of the GPL license. OpenJDK is the official reference implementation of Java SE since version 7.

**JVM:**

The Java Virtual Machine, or JVM, executes live Java applications. Every JRE includes a default JRE, but developers are free to choose another that meets the specific resource needs of their applications.

**JRE:**

The Java Virtual Machine, or JVM, executes live Java applications. Every JRE includes a default JRE, but developers are free to choose another that meets the specific resource needs of their applications. The Java Virtual Machine, or JVM, executes live Java applications. Every JRE includes a default JRE, but developers are free to choose another that meets the specific resource needs of their applications.

**JAVA IDEs:**

IDEs typically provide a code editor, a compiler or interpreter and a debugger that the developer accesses through a unified graphical user interface (GUI). Here are a few popular Java IDEs:

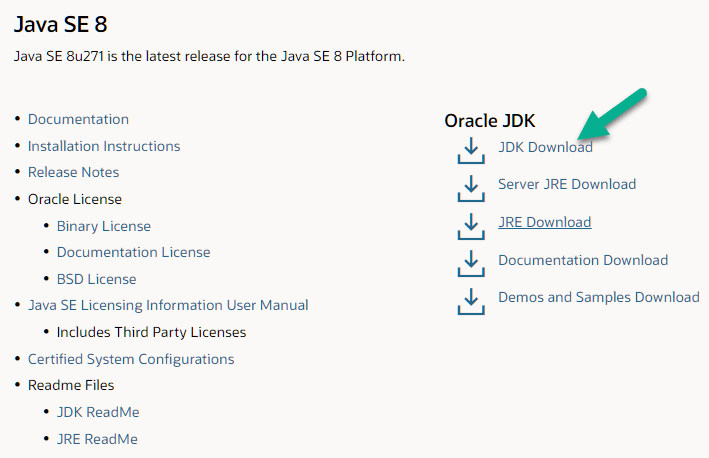
Eclipse: a Java-based open source platform that enables the creation of highly customized IDEs from plug-in components built by Eclipse members. The platform is user-friendly for beginners and also suitable for the creation of more sophisticated applications. Eclipse includes a lot of plug-ins that allow developers to develop and test code written in other languages.

NetBeans: a Java-based IDE and underlying application platform framework. In addition to Java, JavaScript and JavaFX, NetBeans supports C/C++, PHP, Groovy, and HTML5.

How to install Java for Windows:

Following are the steps on how to install Java in Windows 10 for JDK 8 free download for 32 bit or JDK8 download for Windows 64 bit and installation

**Step 1)** Go to <https://www.oracle.com/java/technologies/downloads/>. Click on JDK Download for Java download JDK 8.



**Step 2)** Next,

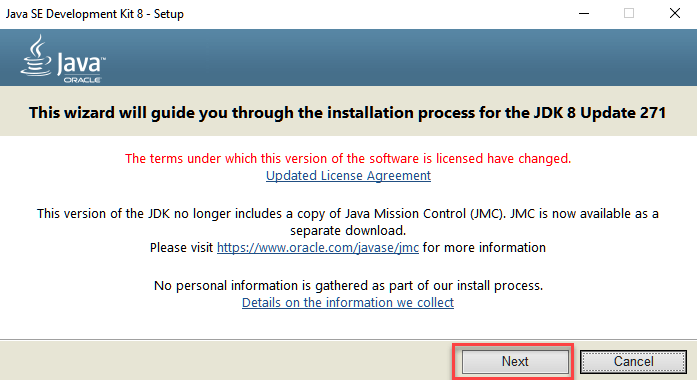
1. Accept License Agreement
2. Download Java 8 JDK for your version 32 bit or JDK download 64 bit.



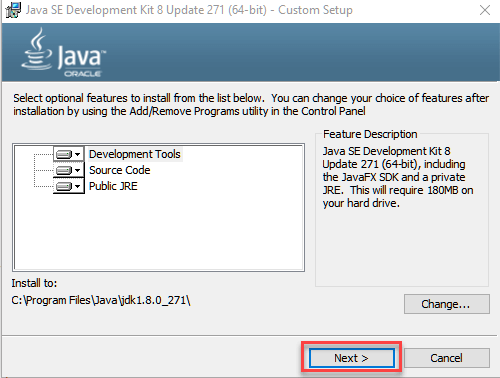
**Step 3)** When you click on the Installation link the popup will be open. Click on I reviewed and accept the Oracle Technology Network License Agreement for Oracle Java SE development kit and you will be redirected to the login page. If you don’t have an oracle account you can easily sign up by adding basics details of yours.



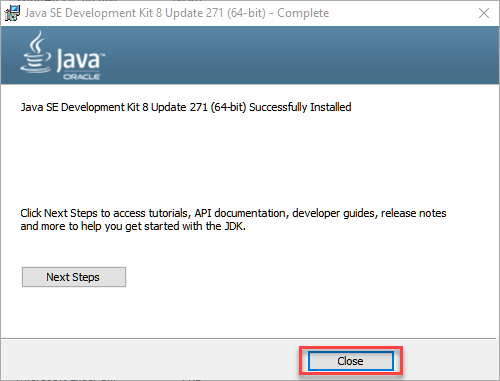
**Step 4)** Once the Java JDK 8 download is complete, run the exe for install JDK. Click Next



**Step 5)** Select the PATH to install Java in Windows… You can leave it Default. Click next.



**Step 6)** Once you install Java in windows, click Close



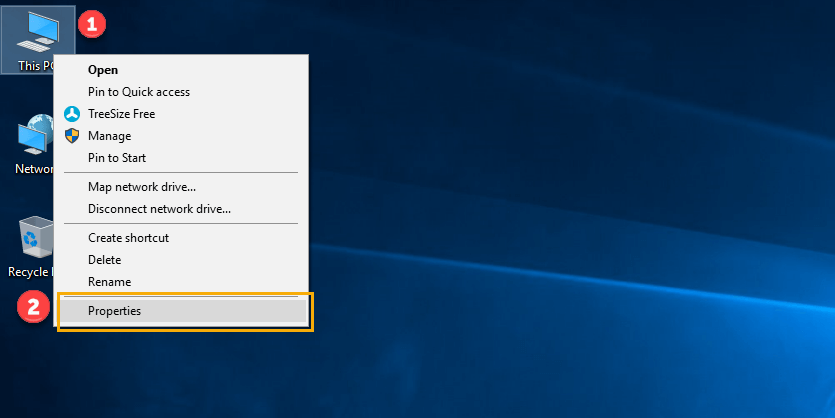
How to set Environment Variables in Java: Path and Classpath:

The PATH variable gives the location of executables like javac, java etc. It is possible to run a program without specifying the PATH but you will need to give full path of executable like **C:\Program Files\Java\jdk1.8.0\_271\bin\javac A.java** instead of simple **javac A.java**

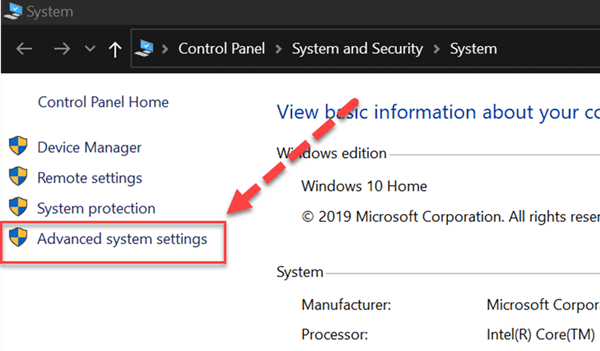
The CLASSPATH variable gives location of the Library Files.

Let’s look into the steps to set the PATH and CLASSPATH

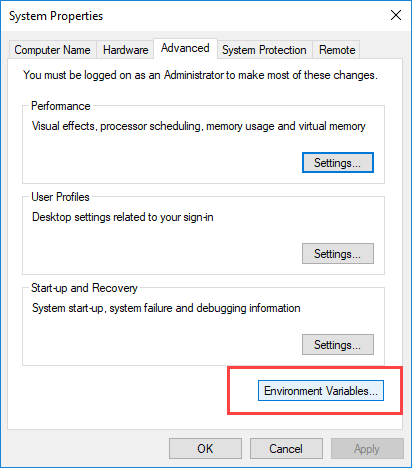
**Step 1)** Right Click on the My Computer and Select the properties



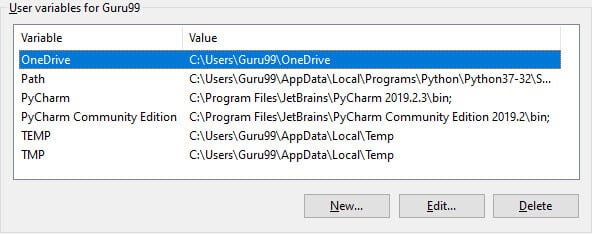
**Step 2)** Click on advanced system settings



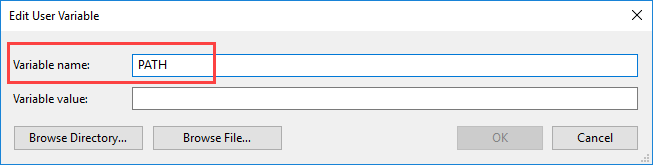
**Step 3)** Click on Environment Variables to set Java runtime environment



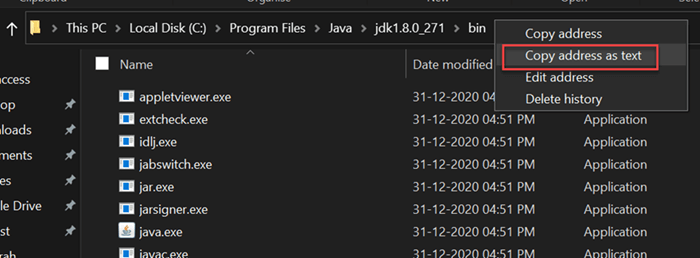
**Step 4)** Click on new Button of User variables



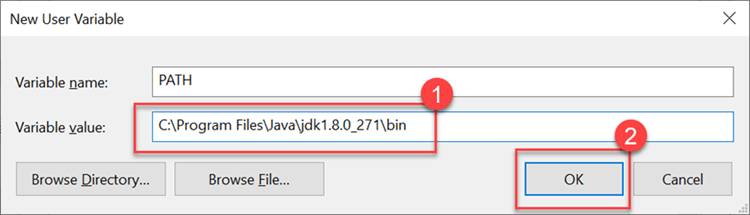
**Step 5)** Type PATH in the Variable name.



**Step 6)** Copy the path of bin folder which is installed in JDK folder.



**Step 7)** Paste Path of bin folder in Variable value. Click on OK Button.

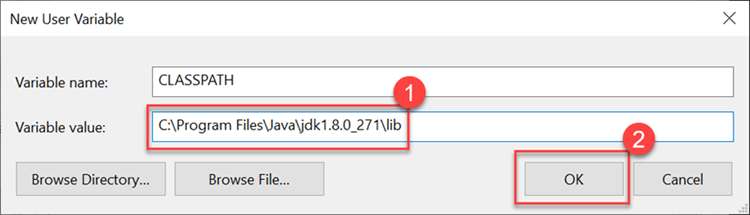


**Note:** In case you already have a PATH variable created in your PC, edit the PATH variable to

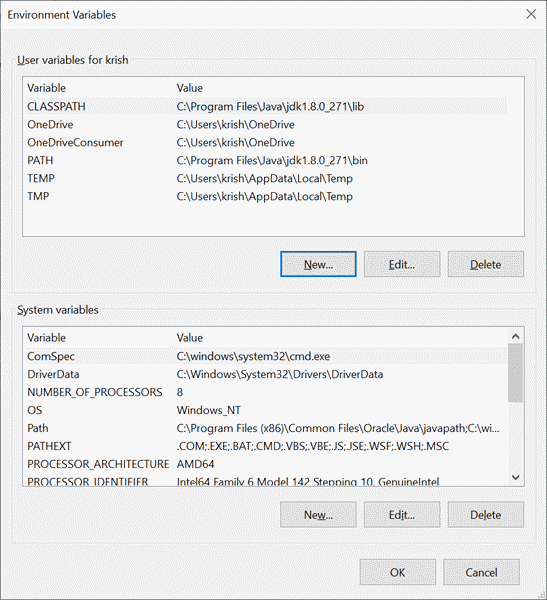
PATH = <JDK installation directory>\bin;%PATH%;

Here, %PATH% appends the existing path variable to our new value

**Step 8)** You can follow a similar process to set CLASSPATH.

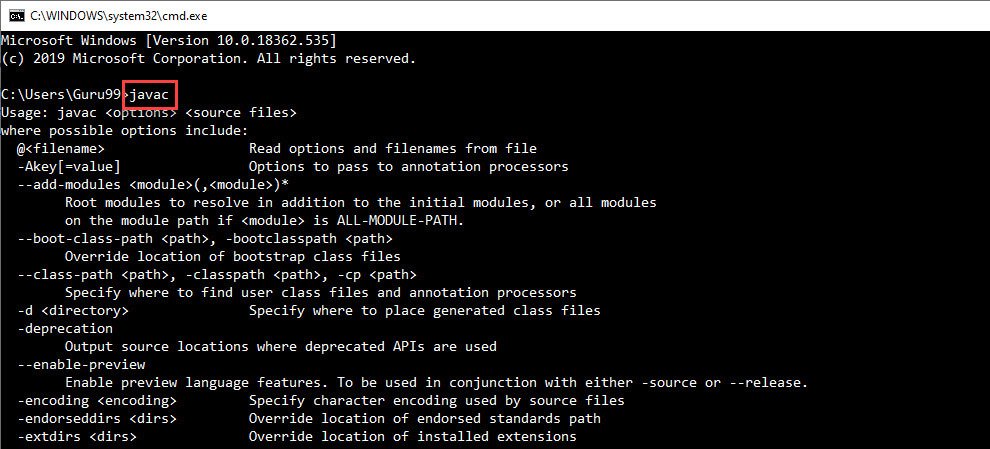


**Step 9)** Click on OK button



**Step 10)** Go to command prompt and type javac commands.

If you see a screen like below, Java is installed.



**Experiment No: 1**

**AIM: To learn basic JAVA programming constructs.**

**Date:**

**CO mapped: CO-1**

**Objectives:** (a) To learn and understand the different basic structures in java, such as syntax, logics, libraries and proper indentation.

**Background:**

**Java Variables**

A variable is a container that holds the value while the Java program is executed. A variable is assigned with a data type. Variable is a name of a memory location. There are three types of variables in java: local, instance, and static.

**Data Types in Java**

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

There are 8 types of primitive data types:

* boolean data type
* byte data type
* char data type
* short data type
* int data type
* long data type
* float data type
* double data type

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

**Operators in Java**

Operator in Java is a symbol that is used to perform operations. For example: +, -, \*, / etc.

There are many types of operators in Java which are given below:

* Unary Operator,
* Arithmetic Operator,
* Shift Operator,
* Relational Operator,
* Bitwise Operator,
* Logical Operator,
* Ternary Operator and
* Assignment Operator.

Java Control Statements

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

* Decision Making statements
  + if statements
  + switch statement
* Loop statements
  + do while loop
  + while loop
  + for loop
  + for-each loop
* Jump statements
  + break statement
  + continue statement

**Practical questions:**

1. Install JDK and IDE in your system. Write down the steps of installation with screenshots.
2. Write a Program that displays Welcome to Java, Learning Java Now and Programming is fun.
3. Write a program that solves the following equation and displays the value x and y:

a) 3.4x+50.2y=44.5 2) 2.1x+.55y=5.9 (Assume Cramer’s rule to solve equation

b) ax+by=e x=ed-bf/ad-bc cx+dy=f y=af-ec/ad-bc )

1. Write a program that reads a number in meters, converts it to feet, and displays the result.
2. Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing it by the square of your height in meters. Write a program that prompts the user to enter weight in pounds and height in inches and displays the BMI.

Note:- 1 pound=.45359237 Kg and 1 inch=.0254 meters.

1. Write a program that prompts the user to enter three integers and display the integers in decreasing order.
2. Write a program that prompts the user to enter a letter and check whether a letter is a vowel or constant.

**Additional programs:**

1. A cashier has currency notes of denominations 1, 2, 5, 10, 50 and 100. If the amount to be withdrawn is input through the keyboard, find the total number of currency notes of each denomination the cashier will have to give to the withdrawer.
2. If a five-digit number is input through the keyboard, write a program to print a new number by adding one to each of its digits. For example, if the number that is input is 12391 then the output should be displayed as 23502.
3. If lengths of three sides of a triangle are input through the keyboard, write a program to print the area of the triangle.
4. Write a program to produce the following patterns.

|  |  |
| --- | --- |
| **\*\*\*\***  **\*\*\***  **\*\***  **\*** | 1234  123  12  1 |
| 1234  567  89  0 | **\***  **\*\*\***  **\*\*\*\*\***  **\*\*\*\*\*\*\***  **\*\*\*\*\***  **\*\*\***  **\*** |

**Observations:**

**2) Program:-**

import java.util.\*;

public class Pra1\_2 {

public static void main(String[] args) {

Date t = new Date();

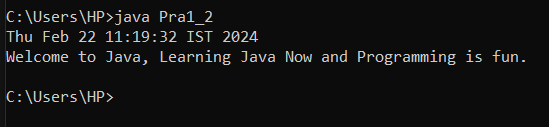
System.out.println(t);

System.out.println("Welcome to Java, Learning Java Now and Programming is fun.");

}

}

**Output:-**



**3) Program:-**

import java.util.\*;

public class Pra1\_3 {

public static void main(String[] args) {

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t);

float a, b, e, c, d, f;

System.out.print("Enter first equation {a b e}: ");

a = se.nextFloat();

b = se.nextFloat();

e = se.nextFloat();

System.out.print("Enter second equation {c d f}: ");

c = se.nextFloat();

d = se.nextFloat();

f = se.nextFloat();

float determinant = a \* d - b \* c;

if (determinant != 0) {

float x = (e \* d - b \* f) / determinant;

float y = (a \* f - e \* c) / determinant;

System.out.println("\nThe solution is x: " + x + " & y: " + y);

} else {

System.out.println("\nEquations have no unique solution.");

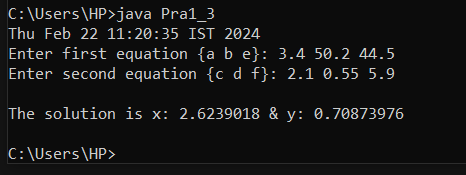
}

se.close();

}

}

**Output:-**



**4) Program:-**

import java.util.\*;

public class Pra1\_4 {

public static void main(String[] args) {

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t);

float meter, feet;

System.out.print("Enter the number: ");

meter = se.nextFloat();

feet = meter \* 3.281f;

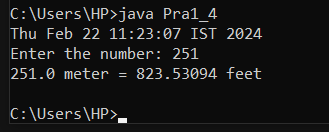
System.out.println(meter + " meter = " + feet + " feet");

se.close();

}

}

**Output:-**



**5) Program:-**

import java.util.\*;

public class Pra1\_5 {

public static void main(String[] args) {

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t);

float weight, height;

System.out.print("Enter your weight: ");

weight = se.nextFloat();

System.out.print("Enter your height: ");

height = se.nextFloat();

float weightInKg = 0.45359237f \* weight;

float heightInMeter = 0.0254f \* height;

float BMI = weightInKg / (float) Math.pow(heightInMeter, 2);

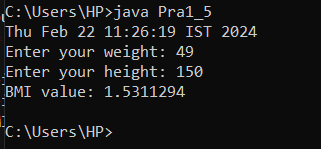
System.out.println("BMI value: " + BMI);

se.close();

}

}

**Output:-**

****

**6) Program:-**

import java.util.\*;

public class Pra1\_6 {

public static void main(String[] args) {

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t);

int num1, num2, num3;

System.out.println("Enter three integers:");

num1 = se.nextInt();

num2 = se.nextInt();

num3 = se.nextInt();

int[] arr = { num1, num2, num3 };

Arrays.sort(arr);

int temp = arr[2];

arr[2] = arr[0];

arr[0] = temp;

System.out.print("\nElements in a decreasing order: ");

for (int a : arr) {

System.out.print(a + " ");

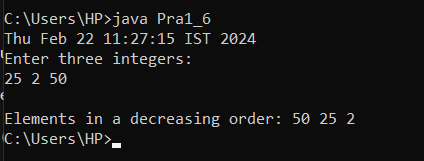
}

se.close();

}

}

**Output:-**

****

**7) Program:-**

import java.util.\*;

public class Pra1\_7 {

static boolean checkVowel(char ch) {

switch (ch) {

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

return true;

default:

return false;

}

}

public static void main(String[] args) {

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t);

String str;

do {

System.out.print("Enter a character: ");

str = se.next();

} while (str.length() != 1);

if (checkVowel((str.toLowerCase()).charAt(0))) {

System.out.println(str + " is vowel.");

} else {

System.out.println(str + " is constant.");

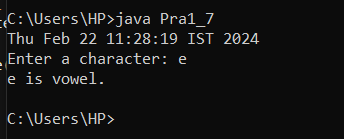
}

se.close();

}

}

**Output:-**



**Conclusion:** Java is a flexible language designed to offer a platform-independent and object-oriented environment for developing various applications. Java is a compiled programming language. However, it follows a two-step process of compilation and interpretation. Java source code (.java files) is first compiled into an intermediate form called bytecode by the Java Compiler. This bytecode is platform-independent and can be executed on any device that has a Java Virtual Machine (JVM). The Java Virtual Machine (JVM) then interprets the bytecode(.class file) and translates it into machine code. Java is a fast, secure, reliable programming language for coding everything from mobile apps and enterprise software to big data applications and server-side technologies.

**Quiz:**

# What is the primary purpose of Java??

# Answer:- Java is a high – level , object oriented language that is known for its portability , security and performance. Java is designed in Write Once, Run Anywhere (WORA) way. First source code of java is compiled into bytecode using compiler and then it is interpreted to machine code. Java code is transformed into bytecode, which is platform-independent and can be interpreted by the Java Virtual Machine (JVM) on any system. Java is used in a wide range of applications, including Mobile App Development, Web-based Applications, Desktop GUI Applications, Gaming Applications, etc.

# What is the main method in Java used for?

# Answer:- The main method in Java is the entry point of any Java application. When the Java Virtual Machine (JVM) starts executing a program, it first searches for the main method. Once found, it starts executing the code within it.

# JVM looks for the main method with the following signature:

# public static void main(String[] args)

# Here, public’s an access modifier, which specifies that the main method can be accessed globally. static, Allows the method to be called without creating an instance of the class containing it. void, Indicates that the method doesn't return any value. String[] args, This parameter is an array of strings that can hold arguments passed to the program from the command line. While not always used, it provides flexibility for passing input data during execution.

# How Java Language is Platform Independent?

# Answer:- Java achieves its platform independence through a two-step process involving byte code and the Java Virtual Machine (JVM). When compile a Java source code file, it is translated into an intermediate form called bytecode. Bytecode is platform-independent. Bytecode can be interpreted by the Java Virtual Machine (JVM) on any system. Each platform has its own Java Virtual Machine (JVM). Thus, Java is designed in Write Once, Run Anywhere (WORA) way.

# What is JVM and JRE?

# Answer:-

# JVM (Java Virtual Machine):- JVM is a virtual machine, it defines an execution environment for Java bytecode. It acts as an abstraction layer between the Java program and the underlying hardware and operating system. When you compile a Java source code, the Java compiler converts it into platform-independent bytecode. The JVM then interprets this bytecode into native machine code that can be executed by the host system.

# JRE (Java Runtime Environment):- The JRE is a software package that includes the actual implementation of the JVM along with Java libraries (Java API), and additional components required for running Java applications. The JRE itself is platform-dependent. However, the JVM within the JRE remains platform-independent. The JRE encapsulates the JVM, providing a complete environment for running Java programs. The JRE (Java Runtime Environment) is a part of the JDK (Java Development Kit)

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# https://www.javatpoint.com/

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks**  **­­­** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 2**

**AIM: To learn Arrays and Strings in Java.**

**Date:**

**CO mapped: CO-1**

**Objectives:**

1. Array manipulation: Learn how to create, populate, access, and modify arrays in Java.
2. String manipulation: Understand how to create and manipulate strings, including concatenation, comparison, and extraction of substrings.
3. Array and String methods: Explore common array and string methods available in Java's standard library.

**Background:**

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on the 1st index, and so on.

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

char[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=new String(ch);

is same as:

String s="javatpoint";

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

**Practical questions:**

1. Write a program that generate 6\*6 two-dimensional matrix, filled with 0’s and 1’s , display the matrix, check every raw and column have an odd number’s of 1’s.

2. Write a generic method that returns the minimum elements and their indices in a two dimensional array.

3. Write a method that returns a new array by eliminating the duplicate values in the array.

4. Write a program to add, subtract or multiply two 3\*3 integer arrays as per choice of user.

Sample Input:

Array 1:

1 2 3

4 5 6

7 8 9

Array 2:

5 6 7

1 2 0

4 3 2

Symbol: +

Sample Output:

6 8 10

5 7 6

11 11 11

5. Write a program to sort an array of 10 elements using selection sort.

6. Write a program that prompts the user to enter a string and displays the number of vowels and consonants in the string.

7. Write a program that prompts the user to enter two strings and displays the largest common prefix of the two strings.

8. Some websites impose certain rules for passwords. Write a method that checks whether a string is a valid password. Suppose the password rules are as follows: A password must have at least eight characters. A password consists of only letters and digits. A password must contain at least two digits. Write a program that prompts the user to enter a password and displays Valid Password if the rules are followed or Invalid Password otherwise.

**Observations:**

**1)Program:-**

import java.util.\*;

public class Pra2\_1 {

static String postfix(int d) {

return (switch (d % 10) {

case 1 -> (d == 11) ? "th" : "st";

case 2 -> (d == 12) ? "th" : "nd";

case 3 -> (d == 13) ? "th" : "rd";

default -> "th";

});

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

// Scanner se = new Scanner(System.in);

// System.out.println("Enter 6\*6 matrix with 0's & 1's: ");

int[][] arr = new int[6][6];

for (int i = 0; i < 6; i++) {

for (int j = 0; j < 6; j++) {

arr[i][j] = (int) (Math.random() \* 10) % 2;

// arr[i][j] = se.nextInt();

}

}

System.out.println("6\*6 Matrix:");

for (int[] row : arr) {

for (int e : row) {

System.out.print(e + " ");

}

System.out.println();

}

System.out.println("");

for (int i = 0; i < 6; i++) {

int row = 0;

for (int j = 0; j < 6; j++) {

row += arr[i][j];

}

if (row % 2 == 0) {

System.out.println(i + postfix(i) + " row has even 1's");

} else {

System.out.println(i + postfix(i) + " row has odd 1's");

}

}

System.out.println("");

for (int i = 0; i < 6; i++) {

int column = 0;

for (int j = 0; j < 6; j++) {

column += arr[j][i];

}

if (column % 2 == 0) {

System.out.println(i + postfix(i) + " column has even 1's");

} else {

System.out.println(i + postfix(i) + " column has odd 1's");

}

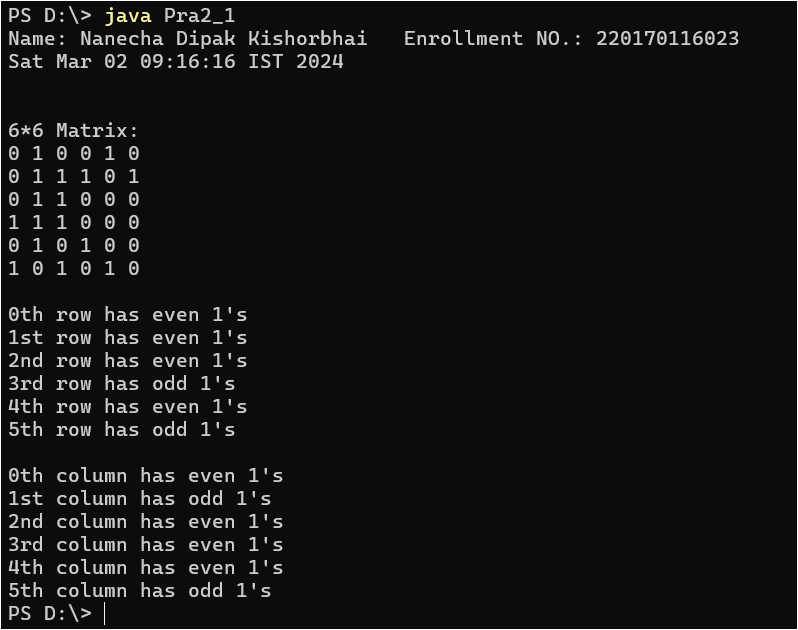
}

// se.close();

}

}

**Output:-**

****

**2)Program:-**

import java.util.\*;

public class Pra2\_2 {

static int[] minimum(int[][] arr) {

int min = arr[0][0], row = 0, column = 0;

for (int i = 0; i < arr.length; i++) {

for (int j = 0; j < arr[0].length; j++) {

if (arr[i][j] < min) {

min = arr[i][j];

row = i;

column = j;

}

}

}

return new int[] { min, row, column };

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.print("Enter size of the two dimensional array: ");

int size = se.nextInt();

// System.out.println("Enter " + size + "\*" + size + " matrix: ");

int[][] arr = new int[size][size];

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

arr[i][j] = (int) (Math.random() \* 100) % 50;

// arr[i][j] = se.nextInt();

}

}

System.out.println("\n" + size + "\*" + size + " Matrix:");

for (int[] row : arr) {

for (int e : row) {

System.out.print(e + " ");

}

System.out.println();

}

int[] min = minimum(arr);

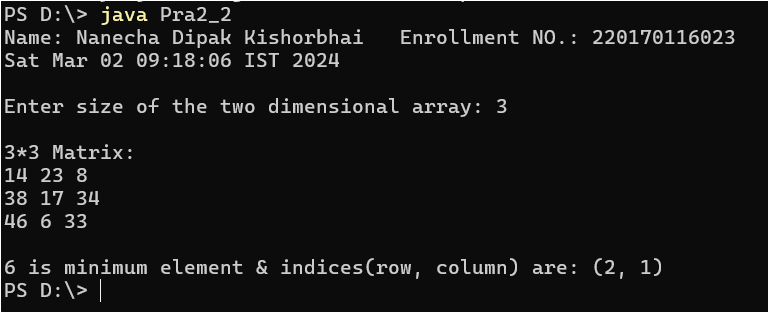
System.out.println("\n" + min[0] + " is minimum element & indices(row, column) are: (" + min[1] + ", " + min[2] + ")");

se.close();

}

}

**Output:-**



**3)Program:-**

import java.util.\*;

public class Pra2\_3 {

static int[] duplicateRemover(int[] arr) {

int[] a = new int[arr.length];

int k = 0;

for (int i = 0; i < arr.length; i++) {

boolean check = false;

for (int j = 0; j < k; j++) {

if (arr[i] == a[j]) {

check = true;

break;

}

}

if (check == false) {

a[k++] = arr[i];

}

}

return a;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.print("Enter size of an array: ");

int size = se.nextInt();

int[] arr = new int[size];

for (int i = 0; i < size; i++) {

System.out.print("Enter " + (i + 1) + " elemnt: ");

arr[i] = se.nextInt();

}

int[] a = duplicateRemover(arr);

System.out.print("\nReduce array: ");

for (int e : a) {

System.out.print(e + " ");

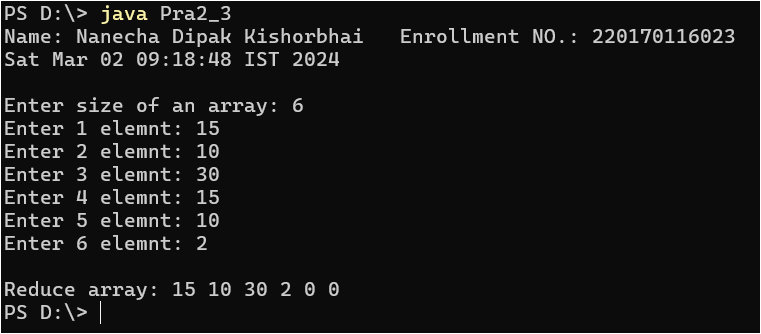
}

se.close();

}

}

**Output:-**

****

**4)Program:-**

import java.util.\*;

public class Pra2\_4 {

static int[][] addition(int[][] a, int[][] b) {

int[][] add = new int[a.length][a[0].length];

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

add[i][j] = a[i][j] + b[i][j];

}

}

return add;

}

static int[][] subtraction(int[][] a, int[][] b) {

int[][] sub = new int[a.length][a[0].length];

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

sub[i][j] = a[i][j] - b[i][j];

}

}

return sub;

}

static int[][] multiplication(int[][] a, int[][] b) {

int[][] multi = new int[a.length][a[0].length];

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

multi[i][j] = 0;

}

}

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

for (int row = 0; row < 3; row++) {

multi[i][j] += a[i][row] \* b[row][j];

}

}

}

return multi;

}

static void display(int[][] arr) {

for (int[] row : arr) {

for (int e : row) {

System.out.print(e + "\t");

}

System.out.println();

}

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

int[][] a = new int[3][3];

int[][] b = new int[3][3];

int[][] add = new int[3][3];

int[][] sub = new int[3][3];

int[][] multi = new int[3][3];

// System.out.println("Enter 3\*3 matrix a: ");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

a[i][j] = (int) (Math.random() \* 10) % 10;

// a[i][j] = se.nextInt();

}

}

// System.out.println("Enter 3\*3 matrix b: ");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

b[i][j] = (int) (Math.random() \* 10) % 10;

// b[i][j] = se.nextInt();

}

}

System.out.println("3\*3 Matrix a:");

display(a);

System.out.println("\n3\*3 Matrix b:");

display(b);

int choice;

do {

System.out.println("\nMenu:");

System.out.println("1) addition:");

System.out.println("2) subtraction:");

System.out.println("3) multiplication:");

System.out.println("0) exit:");

System.out.print("Enter your choice: ");

choice = se.nextInt();

switch (choice) {

case 1:

add = addition(a, b);

System.out.println("\nAfter addition: ");

display(add);

break;

case 2:

sub = subtraction(a, b);

System.out.println("\nAfter subtraction: ");

display(sub);

break;

case 3:

multi = multiplication(a, b);

System.out.println("\nAfter multiplication: ");

display(multi);

break;

case 0:

break;

default:

System.out.println("\nInvalid choice.");

break;

}

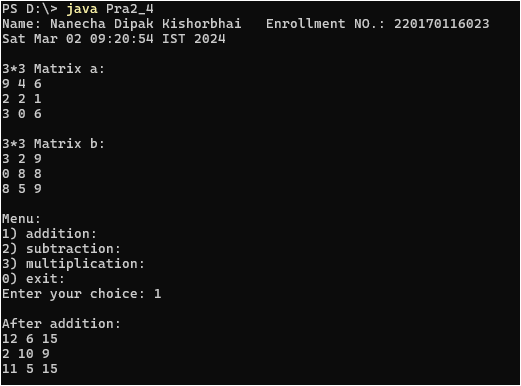
} while (choice != 0);

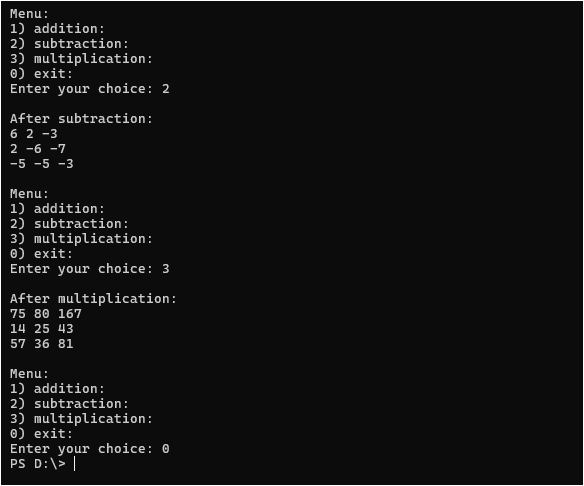
se.close();

}

}

**Output:-**



****

**5)Program:-**

import java.util.\*;

public class Pra2\_5 {

static int[] selectionSort(int[] arr) {

for (int i = 0; i < arr.length - 1; i++) {

int min = i;

for (int j = i + 1; j < arr.length; j++) {

if (arr[j] < arr[min]) {

min = j;

}

}

int temp = arr[min];

arr[min] = arr[i];

arr[i] = temp;

}

return arr;

}

static void display(int[] arr) {

for (int e : arr) {

System.out.print(e + " ");

}

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t + "\n");

int[] arr = new int[10];

for (int i = 0; i < 10; i++) {

arr[i] = (int) (Math.random() \* 100) % 50;

// System.out.print("Enter " + (i + 1) + " element: ");

// arr[i] = se.nextInt();

}

System.out.print("Array: ");

display(arr);

arr = selectionSort(arr);

System.out.print("\n\nSorted array: ");

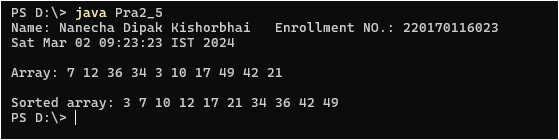
display(arr);

se.close();

}

}

**Output:-**

****

**6)Program:-**

import java.util.\*;

public class Pra2\_6 {

static boolean checkVowel(char ch) {

switch (ch) {

case 'a', 'e', 'i', 'o', 'u':

return true;

default:

return false;

}

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t + "\n");

System.out.print("Enter a string: ");

String str = se.nextLine();

int vowel = 0, consonant = 0;

for (char ch : str.toLowerCase().toCharArray()) {

if (!(Character.isLetter(ch))) {

continue;

} else if (checkVowel(ch)) {

vowel++;

} else {

consonant++;

}

}

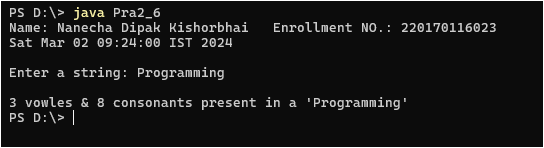
System.out.println("\n" + vowel + " vowles & "+ consonant + " consonants present in a '" + str + "'");

se.close();

}

}

**Output:-**

****

**7)Program:-**

import java.util.\*;

public class Pra2\_7 {

static String largeCommonPrefix(String str1, String str2) {

String prefix = "";

for (int i = 0, j = 0; i < str1.length() && j < str2.length(); i++, j++) {

if (str1.charAt(i) == str2.charAt(j)) {

prefix += str1.charAt(i);

} else {

break;

}

}

return prefix;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t + "\n");

System.out.print("Enter first String: ");

String str1 = se.nextLine();

System.out.print("Enter second String: ");

String str2 = se.nextLine();

String prefix = largeCommonPrefix(str1, str2);

if (prefix == "") {

System.out.println("\nNothing in common.");

} else {

System.out.println("\nLargest common prefix: " + prefix);

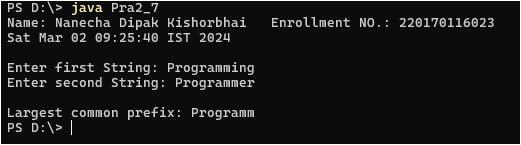
}

se.close();

}

}

**Output:-**

****

**8)Program:-**

import java.util.\*;

public class Pra2\_8 {

static boolean checkPassword(String str) {

if (str.length() < 8) {

return false;

}

int digit = 0;

for (char ch : str.toCharArray()) {

if (!(Character.isLetter(ch) || Character.isDigit(ch)))

return false;

if (Character.isDigit(ch)) {

digit++;

}

}

if (digit >= 2) {

return true;

}

return false;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Scanner se = new Scanner(System.in);

Date t = new Date();

System.out.println(t + "\n");

System.out.print("Enter password: ");

String str = se.nextLine();

if (checkPassword(str))

System.out.println("\n" + str + " is valid password");

else

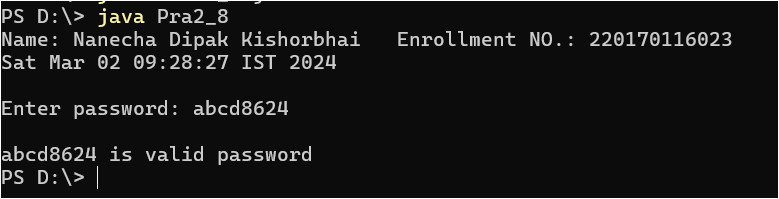
System.out.println("\nInvalid password");

se.close();

}

}

**Output:-**



**Conclusion:** An array is a fundamental data structure in Java. Arrays are collections of similar data types that are stored in contiguous memory locations and can be accessed by their indices. Strings are objects that represent sequences of characters and are immutable in Java. Strings can be declared and initialized using double quotes or the new keyword. Arrays can store primitive data types or objects of a class, while strings can only store characters. Arrays can be multidimensional, while strings are always one-dimensional. Arrays are mutable, meaning that their elements can be changed after initialization. Strings are immutable, meaning that any change to a string creates a new string object and does not affect the original string. Arrays and strings can be iterated using for loops, for-each loops, or while loops.

**Quiz:**

# What are ragged arrays in java and how are they implemented?

# Answer:- In Java, ragged arrays also known as irregular arrays, are a type of multidimensional array where the subarrays (inner arrays) can have different lengths. In a ragged array, each row can have a different number of elements. Unlike a regular (rectangular) array, where all rows have the same number of columns.

# Two ways of implementing ragged arrays are:-

# 1) Array of Arrays 2) ArrayList of ArrayLists Declaration of a ragged array using array of arrays:

# int[][] raggedArray = new int[size][];

# Differentiate String class and StringBuffer class.

# Answer:- String class are immutable, meaning contents of a String class cannot be changed once they are created. Any operation that seems to modify a String actually creates a new String object. StringBuffer class are mutable, meaning contents of a StringBuffer class can be changed directly using methods like append, replace, insert, and delete. The length of the String class is static, while the length of StringBuffer can be modified whenever required, as it is dynamic in behavior. In String class, More memory-intensive because of the immutability. Every modification creates a new object, leading to potential memory overhead. In StringBuffer class, Less memory-intensive because modifications happen in-place, avoiding the creation of new objects.

# How Create a two dimensional array. Instantiate and Initialize it?

# Answer:-

# Two ways of implementing 2D arrays are:-

# 1) Array of Arrays

# 2) ArrayList of ArrayLists

# Declaration of a two dimensional array: Specify the data type of the elements, variablle name and the number of rows and columns using square brackets. Ex:- int[][] twoDArray; Instantiation: Allocate memory for the array using the new keyword.

# twoDArray = new int[n][m];

# Initialization:

# Use nested loops to iterate over the rows and columns of the array, assigning values to each element. Initialization within declaration: Provide values directly within curly braces for each row, separated by commas. This method is convenient for smaller arrays with fixed values.

# In implementation of ArrayList of ArrayLists, each inner ArrayList represents a row of 2D structure.

# Explain the various String functions with their syntax.

# Answer:- length():

# Returns the length (number of characters) of the string. Syntax: str.length() charAt(index): Returns the character at the specified index. Syntax: str.charAt(index) concat(str): Appends the specified string to the end of the current string and returns the resulting string. Syntax: str.concat(" additional text") toLowerCase(): Converts all characters to lowercase and returns the resulting string.

# Syntax: str.toLowerCase() toUpperCase(): Converts all characters to uppercase and returns the resulting string.

# Syntax: str.toUpperCase() trim(): Removes leading and trailing whitespace characters and returns the resulting string.

# Syntax: str.trim() replace(oldChar, newChar):

# Replaces all occurrences of the old character with the new character.

# Syntax: str.replace(oldChar, newChar)

# equals(str):

# Compares the content of two strings for equality. Return true if equals otherwise return false.

# Syntax: str.equals(str) equalsIgnoreCase(str): Compares two strings ignoring their case. Return true if equals otherwise return false. Syntax: str.equalsIgnoreCase(str)

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 3**

**AIM: To implement basic object-oriented concepts.**

**Date:**

**CO mapped: CO-2**

**Objectives:**

1. To apply fundamental object-oriented principles, such as class design, encapsulation, inheritance, and polymorphism, to improve software modularity, code organization, and maintainability.
2. Implementing these basic object-oriented concepts in your software development practices will help you create more structured, maintainable, and reusable code, which is essential for building robust and scalable software systems.

**Background:**

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* Object: Any entity that has a state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical. An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.
* Class: Collection of objects is called class. It is a logical entity. A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.
* Inheritance: When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
* Polymorphism: If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.
* Abstraction: Hiding internal details and showing functionality is known as abstraction. For example, phone call, we don't know the internal processing. In Java, we use abstract class and interface to achieve abstraction.
* Encapsulation: Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines. A java class is an example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**Practical questions:**

1. Write a Java application which takes several command line arguments, which are supposed to be names of students and prints output as given below: (Suppose we enter 3 names then output should be as follows):

Number of arguments = 3

1: First Student Name is =Tom

2: Second Student Name is =Dick

3: Third Student Name is =Harry

(Hint: An array may be used for converting from numeric values from 1 to 20 into String.)

2. Design a class named Rectangle to represent a rectangle. The class contains: Two double data fields named width and height that specify the width and height of the rectangle. The default values are 1 for both width and height.

A no-arg constructor that creates a default rectangle.

A constructor that creates a rectangle with the specified width and height.

A method named getArea() that returns the area of this rectangle.

A method named getPerimeter() that returns the perimeter.

Write a test program that creates two Rectangle objects—one with width 4 and height 40 and the other with width 3.5 and height 35.9. Display the width, height, area, and perimeter of each rectangle in this order.

3. Define a class called Cartesian Point, which has two instance variables, x and y. Provide the methods get X() and get Y() to return the values of the x and y values respectively, a method called move() which would take two integers as parameters and change the values of x and y respectively, a method called display() which would display the current values of x and y. Now overload the method move() to work with single parameter, which would set both x and y to the same values,provide constructors with two parameters and overload to work with one parameter as well. Now define a class called Test Cartesian Point, with the main method to test the various methods in the Cartesian Point class.

4. Create a class Employee which has two private data members name and salary and it has two public member functions named as getData() and putData() where getData() gets name and salary from the user putData() displays name and salary for any user.

5. Define a class Time with hours and minutes astwo data members, add necessary member functions to initialize and display data of class. Do not use constructors in a class. Define a member function sum () which adds two Time objects. (Use the statements like T3.sum (T1, T2)).

6. Define Class named Point which represents 2-D Point, i.e P (x, y). Define Defaultconstructor to initialize both data member value 5, Parameterized constructor to initialize member according to value supplied by user and Copy Constructor. Define Necessary Function and Write a program to test class Point.

7. Create a class Account. It has three data member account id, name and balance. Define function to assign value and display value. Define function that search account number given by the user. If account number exists, print detail of that account. Write a program using array of object. Declare at least 5 account and print details.

**Observations:**

**1)Program:-**

import java.util.\*;

public class Pra3\_1 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

if (args.length < 1) {

System.out.println("Enter at least 1 command line argument.");

return;

}

int Arguments = args.length;

String[] numericToString = { "First", "Second", "Third", "Fourth", "Fifth", "Sixth", "Seventh", "Eighth", "Ninth", "Tenth", "Eleventh", "Twelfth", "Thirteenth", "Fourteenth", "Fifteenth", "Sixteenth", "Seventeenth", "Eighteenth", "Nineteenth", "Twentieth" };

System.out.println("Number of arguments = " + Arguments);

for (int i = 0; i < Arguments; i++) {

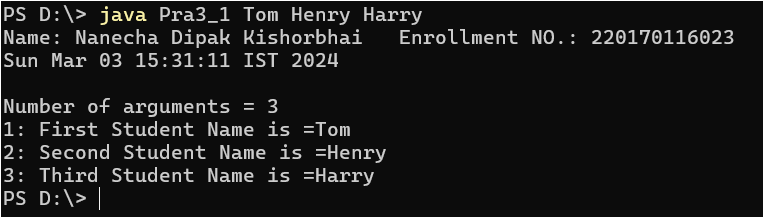
System.out.println((i + 1) + ": " + numericToString[i] + " Student Name is =" + args[i]);

}

}

}

**Output:-**

****

**2)Program:-**

import java.util.\*;

class Rectangles {

private double width, height;

Rectangles() {

this.width = 1;

this.height = 1;

}

Rectangles(double width, double height) {

this.width = width;

this.height = height;

}

double getArea() {

return width \* height;

}

double getPerimeter() {

return 2 \* (width + height);

}

void print() {

System.out.println("Width: " + this.width);

System.out.println("Height: " + this.height);

System.out.println("Area: " + (float)this.getArea());

System.out.println("Perimeter: " + this.getPerimeter());

}

}

public class Pra3\_2 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

double width, height;

System.out.print("Enter first rectangle's parameter(width, height): ");

width = se.nextDouble();

height = se.nextDouble();

Rectangles rec1 = new Rectangles(width, height);

System.out.print("Enter second rectangle's parameter(width, height): ");

width = se.nextDouble();

height = se.nextDouble();

Rectangles rec2 = new Rectangles(width, height);

System.out.println("\nFirst Rectangle:");

rec1.print();

System.out.println("\nSecond Rectangle:");

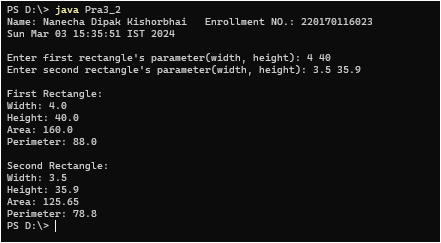
rec2.print();

se.close();

}

}

**Output:-**

****

**3)Program:-**

import java.util.\*;

class CartesianPoint {

private int x, y;

CartesianPoint(int x, int y) {

this.x = x;

this.y = y;

}

CartesianPoint(int z) {

this.x = this.y = z;

}

int getX() {

return this.x;

}

int getY() {

return this.y;

}

void move(int x, int y) {

this.x = x;

this.y = y;

}

void move(int z) {

this.x = this.y = z;

}

void display() {

System.out.println("X: " + x);

System.out.println("Y: " + y + "\n");

}

}

public class TestCartesianPoint {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

CartesianPoint p1 = new CartesianPoint(5, 4);

System.out.println("values of p1:");

p1.display();

System.out.println("After moving p1 to (17, 5):");

p1.move(17, 5);

p1.display();

CartesianPoint p2 = new CartesianPoint(20);

System.out.println("values of p2:");

p2.display();

System.out.println("After moving p2 to (6, 6):");

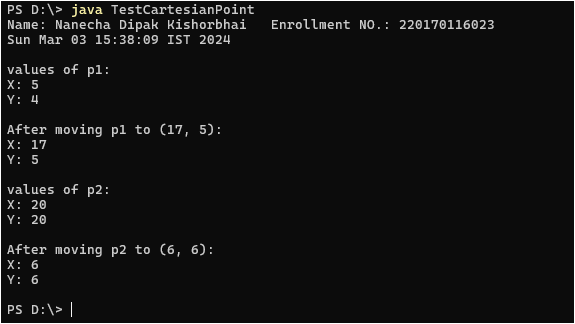
p2.move(6);

p2.display();

}

}

**Output:-**

****

**4)Program:-**

import java.util.\*;

class Employee {

private String name;

private float salary;

public void getData() {

Scanner se = new Scanner(System.in);

System.out.print("Enter name of the employee: ");

this.name = se.nextLine();

System.out.print("Enter salary: ");

this.salary = se.nextFloat();

se.close();

}

public void putData() {

System.out.println("\nName: " + this.name + " , salary: " + this.salary);

}

}

public class Pra3\_4 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Employee e1 = new Employee();

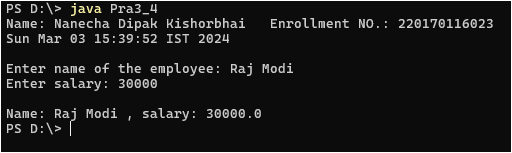
e1.getData();

e1.putData();

}

}

**Output:-**

****

**5)Program:-**

import java.util.\*;

class Time {

private int hours, minutes;

void setData(int hours, int minutes) {

if (hours >= 0 && hours < 24) {

this.hours = hours;

} else {

System.out.println("Error in seting hours " + hours);

System.exit(1);

}

if (minutes >= 0 && minutes < 60) {

this.minutes = minutes;

} else {

System.out.println("Error in seting minutes " + minutes);

System.exit(1);

}

}

int getHours() {

return this.hours;

}

int getMinutes() {

return this.minutes;

}

void print() {

System.out.println("Time: " + this.getHours() + " hours and " + this.getMinutes() + " minutes\n");

}

void sum(Time t1, Time t2) {

boolean carry = false;

if (t1.minutes + t2.minutes >= 60) {

carry = true;

}

this.minutes = (t1.minutes + t2.minutes) % 60;

this.hours = carry ? (t1.getHours() + t2.getHours() + 1) % 24 : (t1.getHours() + t2.getHours()) % 24;

}

}

public class Pra3\_5 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

int hour, minute;

Time t1 = new Time();

System.out.print("Enter T1 time(hour, minute): ");

hour = se.nextInt();

minute = se.nextInt();

t1.setData(hour, minute);

System.out.print("T1 ");

t1.print();

Time t2 = new Time();

System.out.print("Enter T2 time(hour, minute): ");

hour = se.nextInt();

minute = se.nextInt();

t2.setData(hour, minute);

System.out.print("T2 ");

t2.print();

Time t3 = new Time();

t3.sum(t1, t2);

System.out.println("\nT3 (sum of T1 & T2)");

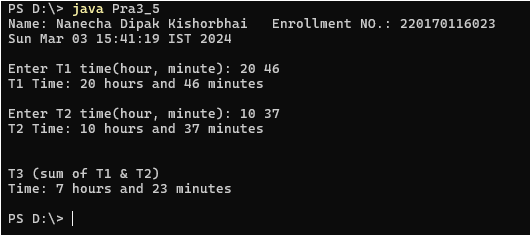
t3.print();

se.close();

}

}

**Output:-**

****

**6)Program:-**

import java.util.\*;

class Point {

private int x, y;

Point() {

this.x = 5;

this.y = 5;

}

Point(int x, int y) {

this.x = x;

this.y = y;

}

Point(Point p) {

this.x = p.x;

this.y = p.y;

}

void setX(int x) {

this.x = x;

}

void setY(int y) {

this.y = y;

}

int getX() {

return this.x;

}

int getY() {

return this.y;

}

void print() {

System.out.println("X: " + this.getX() + ", Y: " + this.getY());

}

}

public class Pra3\_6 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.println("Using default constructor:");

Point p1 = new Point();

System.out.println("p1 point");

p1.print();

System.out.println("\nUsing parameterized constructor:");

System.out.print("Enter value of x & y: ");

int x = se.nextInt();

int y = se.nextInt();

Point p2 = new Point(x, y);

System.out.println("p2 point");

p2.print();

System.out.println("\nUsing copy constructor:");

Point p3 = new Point(p2);

System.out.println("p3 point");

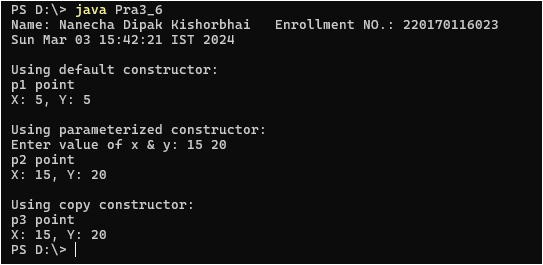
p3.print();

se.close();

}

}

**Output:-**

****

**7)Program:-**

import java.util.\*;

class Account {

private int accId;

private String name;

private float balance;

void setValue(int id, String name, float balance) {

this.accId = id;

this.name = name;

this.balance = balance;

}

int getId() {

return this.accId;

}

String getName() {

return this.name;

}

float getBalance() {

return this.balance;

}

void print() {

System.out.println("\nAccount id: " + this.getId());

System.out.println("Name: " + this.getName());

System.out.println("Balance: " + this.getBalance());

}

static Account searchAccount(Account[] cust, int data) {

for (Account acc: cust) {

if (acc.getId() == data) {

return acc;

}

}

return null;

}

}

public class Pra3\_7 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.print("how many details you want to fill: ");

int size = se.nextInt();

Account[] cust = new Account[size];

for (int i = 0; i < size; i++) {

System.out.println("\nEnter detail of " + (i + 1) + "'s customer (account id, name, balance): ");

cust[i] = new Account();

int id = se.nextInt();

se.nextLine();

String name = se.nextLine();

float balance = se.nextFloat();

cust[i].setValue(id, name, balance);

}

System.out.println("\nDetails of all accounts:");

for (Account acc : cust) {

acc.print();

}

System.out.print("\nEnter account number which to search: ");

int data = se.nextInt();

Account acc = Account.searchAccount(cust, data);

if (acc != null) {

acc.print();

} else {

System.out.println("\nAccount not found");

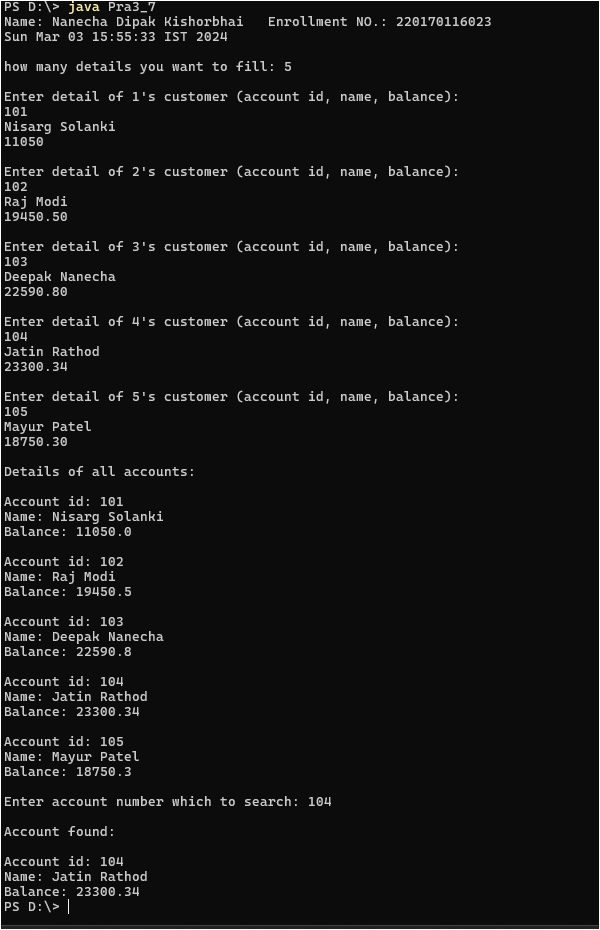
}

se.close();

}

}

**Output:-**

****

**Conclusion:** Object-Oriented design contain the core principles of structuring code around objects that encapsulate data and behavior. Classes act as templates/blueprints for creating objects, defining their attributes and methods. Objects represent real-world entities represented in code with attributes (data) and methods (behaviors). Abstraction involves focusing on essential details while concealing internal complexity. Inheritance facilitates the creation of new classes (subclasses) that inherit properties and behaviors from existing classes (superclasses), promoting code reusability and hierarchy. Polymorphism refers to the situation where a single task is executed in multiple ways.

**Quiz:**

# Explain the concept of encapsulation and why it is important in OOP.

# Answer:- Encapsulation is a fundamental concept in Object-Oriented Programming (OOP) that refers to the bundling of data (attributes or properties) and the methods (functions or procedures) that operate on that data into a single unit. This unit is typically a class, which acts like a blueprint for creating objects. Encapsulation creates a protective barrier around the data, restricting direct access from outside the class.

# Importance:-

# Data Hiding: Encapsulation allows the internal details (data and implementation) of an object to be hidden from the outside world.

# Abstraction: Users of the class interact with the object through its public interface (methods) that provide a safe and controlled way for external code to interact with the data.

# Flexibility: Encapsulation enhances code flexibility by allowing changes to the internal implementation of a class do not affect the external code that uses the class, as long as the public interface remains unchanged.

# What is a class, and how does it relate to objects in OOP?

# Answer:- In Object-Oriented Programming (OOP), a class acts as a blueprint or a template for creating objects. It defines a data structure that encapsulates properties (data) and behaviors (functions) that all objects of that type will share. A class itself doesn't exist in memory and doesn't hold any data. Classes also support concepts like inheritance and polymorphism, enhancing code organization and reusability.

# An Object is an instance of a class, representing a real-life entity. Objects are created based on the structure defined by the class. This process is called instantiation. It is a concrete realization of the class blueprint. If you consider a class as a blueprint for a house, then an object would be the actual house built based on that blueprint.

# Define and explain static and dynamic binding.

# Answer:-

# static binding also known as early binding, which refers to the process where the compiler determines which method to call during compile-time based on the type of reference variable. It's known as early binding because the binding between the method call and the method definition is resolved at compile-time and remains fixed during program execution. Static binding is often used for methods that are specific to a particular class and won't be overridden in subclasses.

# Dynamic binding also known as late binding, occurs when the method call is resolved at runtime based on the actual object type rather than the reference type. Dynamic binding is typically used for virtual methods that can be overridden in subclasses. It allows for polymorphic behavior, where different subclasses can provide their own implementations of the same method. Dynamic binding enables more flexible and polymorphic behavior in object-oriented programming languages.

# Explain the concept of method overloading and method overriding in OOP.

# Answer:-

# Method overloading is the ability to define multiple methods in the same class with the same name but different parameters. Overloading allows a class to have multiple methods with the same name performing different tasks. Overloaded methods are determined during compile-time based on the method signature. Method overriding occurs in inheritance structures. When a subclass inherits a method from its superclass(parent class) and redefines it with the same name, parameter list, and return type, it's called overriding. Method overriding Allows a subclass to provide a specific implementation of a method inherited from a superclass. Overridden methods are determined during runtime based on the actual type of the object.

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 4**

**AIM: To implement inheritance and object-oriented concepts.**

**Date:**

**CO mapped: CO-2**

**Objectives:**

1. To master the fundamental principles of inheritance and object-oriented concepts, enabling the design and development of efficient, maintainable, and scalable software solutions by leveraging the power of class hierarchies and code reuse.
2. Implementing these basic object-oriented concepts in your software development practices will help you create more structured, maintainable, and reusable code, which is essential for building robust and scalable software systems.

**Background:**

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system). The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Terms used in Inheritance

* Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* Sub Class/Child Class: Subclass is a class that inherits the other class. It is also called a derived class, extended class, or child class.
* Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* Reusability: As the name specifies, reusability is a mechanism that facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

**Practical questions:**

1. A set of 5 words (strings) will be taken as command line arguments. Write a program to reverse each word and check whether it is palindrome or not using method.

2. Define the class BankAccount to represent an account we open with bank. Define the subclasses SavingAccount and FixedDepositAccount. Implement the operations like openAccount(), deposit(), checkBalance(), withdraw() and calInterest() for these classes.

3. Write a program that finds area of any shape by overloading area () method for Square, Rectangle, Triangle and Square.

4. Write a program that finds Volume of any shape by overloading volume () method for Cube, Rectangular Cube and Sphere.

5. Write a Program to maintain employee’s information. Program should illustrate Inheritance concept. (Use your imagination to create class or subclass used for employee).

6. Create a base class Shape. Use this class to store two double type values that could be used to compute area of any shape. Derive two specific classes called Triangle and Rectangle from the base shape. Add to the base a member function getdata() to initialize base class data member and another member function display\_area() to compute and display the area of figures. (Use Method Overriding).

**Observations:**

**1)Program:-**

import java.util.\*;

public class Pra4\_1 {

public static boolean checkPalindrome(String str) {

String revString = "";

for (int i = str.length() - 1; i > -1; i--) {

revString += str.charAt(i);

}

if (revString.equals(str)) {

return true;

} else {

return false;

}

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

if (args.length != 5) {

System.out.println("Enter 5 words as a command line arguments.");

System.exit(1);

}

for (String str : args) {

if (checkPalindrome(str)) {

System.out.println(str + " is palindrome");

} else {

System.out.println(str + " is not a palindrome");

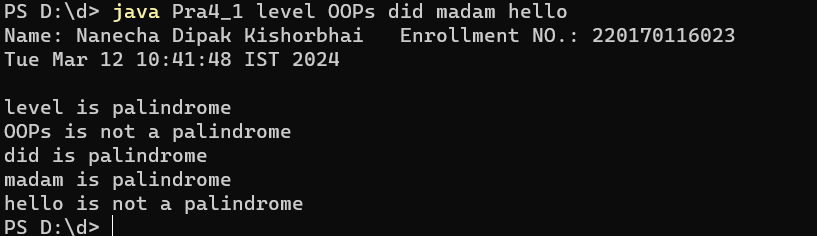
}

}

}

}

**Output:-**



**2)Program:-**

import java.util.\*;

class BankAccount {

private int accId;

private String name;

private double balance;

private int age;

void openAccount(Scanner se) {

System.out.println("Enter details of the customer(account id, name, balance, age): ");

this.accId = se.nextInt();

se.nextLine();

this.name = se.nextLine();

this.balance = se.nextDouble();

this.age = se.nextInt();

}

void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.printf("balance: %.2f\n", balance);

} else {

System.out.println("Invalid deposit amount.");

}

}

double checkBalance() {

return this.balance;

}

void withdraw(double amount) {

this.balance -= amount;

System.out.printf("After withdrawal balance is: %.2f\n", balance);

}

void calInterest() {}

int getAge() {

return this.age;

}

}

class SavingAccount extends BankAccount {

private static double interestRate = 0.04;

SavingAccount(Scanner se) {

super.openAccount(se);

}

void calInterest() {

double interest = checkBalance() \* interestRate;

System.out.print("After interest ");

super.deposit(interest);

}

void withdraw(double amount) {

double minBalance = 500.0;

if (checkBalance() - amount <= minBalance) {

System.out.println("\nCan't withdraw, minimum " + minBalance + " required in account");

return;

}

super.withdraw(amount);

}

}

class FixedDepositAccount extends BankAccount {

private static double interestRate;

private double depositTime;

FixedDepositAccount(Scanner se) {

super.openAccount(se);

System.out.print("How many years do you want to deposit money: ");

this.depositTime = se.nextDouble();

if (this.depositTime < 1)

interestRate = 0.06;

else if (this.depositTime >= 1 && this.depositTime < 5)

interestRate = 0.07;

else if (this.depositTime >= 5)

interestRate = 0.075;

if (getAge() >= 60)

interestRate += 0.01;

}

void calInterest() {

double interest = checkBalance() \* interestRate;

System.out.print("After interest ");

super.deposit(interest);

}

void withdraw(double amount) {

if (checkBalance() < amount) {

System.out.println("\nNot sufficient balance.");

return;

}

super.withdraw(amount);

}

}

public class Pra4\_2 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.println("Saving Account");

SavingAccount sAcc = new SavingAccount(se);

System.out.print("How much money do you want to withdraw: ");

double data = se.nextDouble();

sAcc.withdraw(data);

System.out.println("\nFixed Deposit Account");

FixedDepositAccount fdAcc = new FixedDepositAccount(se);

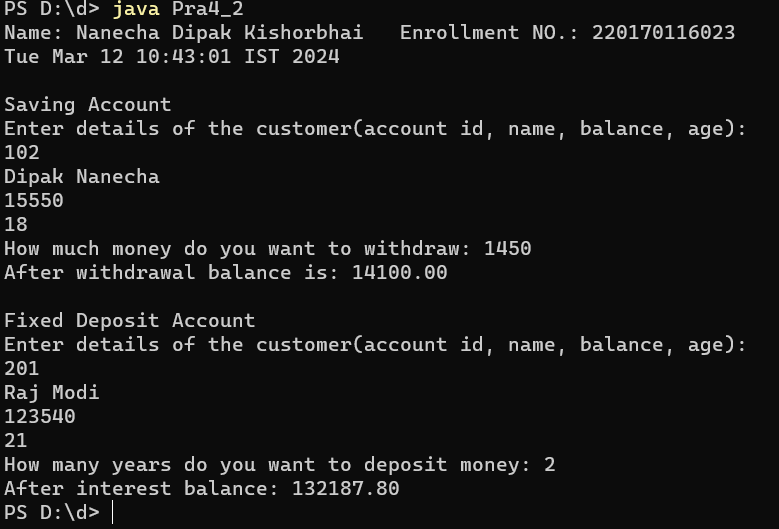
fdAcc.calInterest();

se.close();

}

}

**Output:-**



**3)Program:-**

import java.util.\*;

public class Pra4\_3 {

static float area(float side) {

return side \* side;

}

static float area(float width, float length) {

return width \* length;

}

static float area(float a, float b, float c) {

float s = (a + b + c) / 2;

float tempArea = s \* (s - a) \* (s - b) \* (s - c);

return (float) Math.pow(tempArea, 0.5);

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

System.out.println("Square(5): " + area(5));

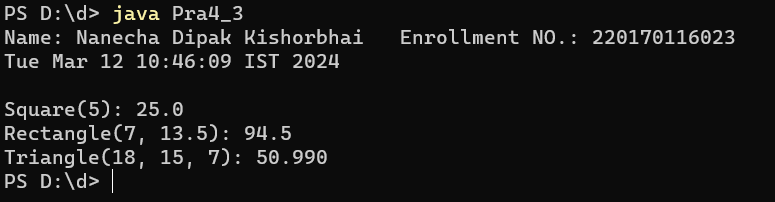
System.out.println("Rectangle(7, 13.5): " + area(7, 13.5f));

System.out.printf("Triangle(18, 15, 7): %.3f\n", area(18, 15, 7));

}

}

**Output:-**



**4)Program:-**

import java.util.\*;

public class Pra4\_4 {

static float volume(float side) {

return side \* side \* side;

}

static float volume(float length, float width, float height) {

return length \* width \* height;

}

static float volume(float side, String shape) {

if (shape.equalsIgnoreCase("y")) {

return (4 / 3.0f) \* (float) Math.PI \* side \* side \* side;

}

return 0;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

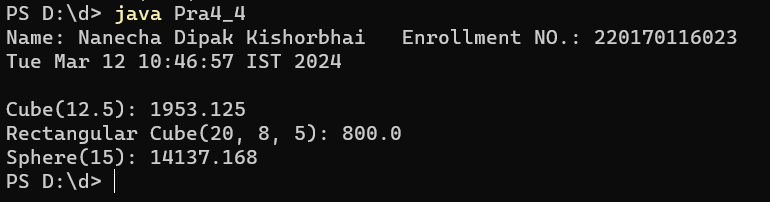
System.out.println("Cube(12.5): " + volume(12.5f));

System.out.println("Rectangular Cube(20, 8, 5): " + volume(20, 8, 5));

System.out.println("Sphere(15): " + volume(15, "y")); // Enter Y/y for Sphere

}

**Output:-**



**5)Program:-**

import java.util.\*;

class Employee {

private int empId;

private String empName;

private double empSalary;

Employee(int empId, String name, double salary) {

this.empId = empId;

this.empName = name;

this.empSalary = salary;

}

void print() {

System.out.println("Employee ID: " + empId);

System.out.println("Employee Name: " + empName);

System.out.println("Employee Salary: " + empSalary);

}

int getID() {

return this.empId;

}

}

class Department extends Employee {

private String departmentName;

Department(int empId, String name, double salary, String deptName) {

super(empId, name, salary);

this.departmentName = deptName;

}

void print() {

super.print();

System.out.println("Department: " + departmentName);

}

}

public class Pra4\_5 {

static void printDetaill(Employee[] emp, int[] idArray) {

System.out.println("Employee Details:\n");

for(int i = 0; i < emp.length; i++) {

for(int j = 0; j < idArray.length; j++) {

if (emp[i].getID() == idArray[j]) {

emp[i].print();

System.out.println("");

}

}

}

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

Employee[] emp = {

new Department(101, "Kishan Patel", 70000.0, "HR"), // human resources

new Employee(102, "Jatin Rathod", 40000.0),

new Department(103, "Raj Modi", 65000.0, "IT"),

new Department(104, "Ami Patel", 55000.0, "AD") // advertising department

};

int[] idArray = {102, 104};

Arrays.sort(idArray);

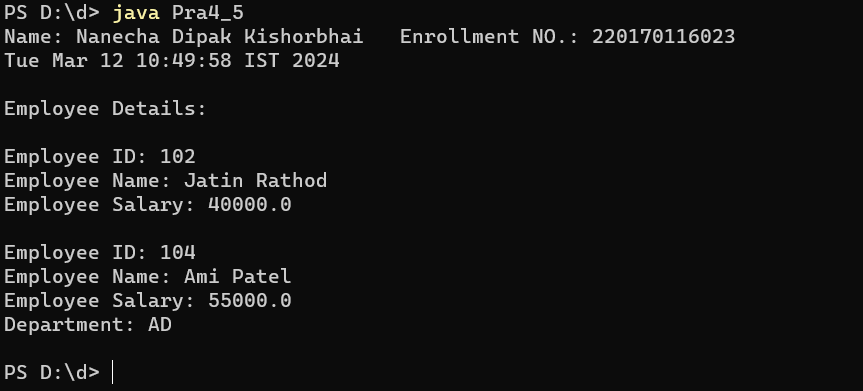
printDetaill(emp, idArray);

se.close();

}

}

**Output:-**



**6)Program:-**

import java.util.\*;

class Shape {

double a, b;

void getData(Scanner se) {

System.out.print("Enter dimension 1: ");

this.a = se.nextDouble();

System.out.print("Enter dimension 2: ");

this.b = se.nextDouble();

}

void displayArea() {}

}

class Triangle extends Shape {

private double c;

void getData(Scanner se) {

super.getData(se);

System.out.print("Enter dimension 3: ");

this.c = se.nextDouble();

}

void displayArea() {

double s = (a + b + c) / 2;

double tempArea = s \* (s - a) \* (s - b) \* (s - c);

System.out.printf("Area of the triangle: %.2f\n", Math.pow(tempArea, 0.5));

}

}

class Rectangle extends Shape {

void displayArea() {

System.out.printf("Area of the rectangle: %.2f\n", a \* b);

}

}

public class Pra4\_6 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.println("Triangle:");

Triangle tri = new Triangle();

tri.getData(se);

tri.displayArea();

System.out.println("\nRectangle:");

Rectangle r = new Rectangle();

r.getData(se);

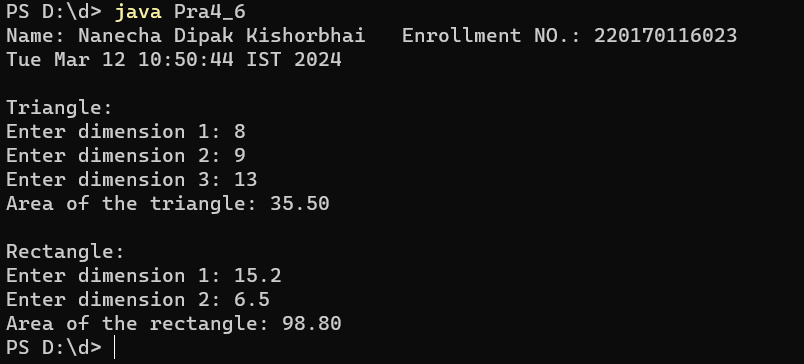
r.displayArea();

se.close();

}

}

**Output:-**



**Conclusion:** A class acts as a blueprint or a template for creating objects. Superclass(base class) is the class from where a subclass inherits the features. Subclass(derived class) is a class that inherits the other class. Encapsulation involves bundling the data(attributes) and the methods(functions) that operate on the data into a single unit known as a class. Inheritance allows a class (derived class) to inherit the characteristics and behaviors(methods) of another class(base class). Polymorphism allows objects of different types to be treated as objects of a common type.  
Abstraction: It is the process of hiding the complex implementation details and showing only the essential features of the object.

**Quiz:**

# What is inheritance in java? Explain different types of inheritance with proper example.

# Answer:- Inheritance is a fundamental concept in Object-Oriented Programming (OOP). It allows a class (derived class) to inherit the characteristics and behaviors (fields and methods) of another class (superclass or base class). The extends keyword is used to declare a subclass that inherits from a superclass. It's a way to reuse code and establish relationships between objects.

# Types of inheritance:-

# 1) Single Inheritance:

# A subclass inherits from only one superclass, meaning a class can extend only one other class. Ex:-

# class Animal { void eat() {} }

# class Dog extends Animal { void bark() {} } Here, Dog inherits eat() from Animal and adds its own method bark(). 2)Multilevel Inheritance: A subclass inherits from another subclass, forming a chain. Ex:- class A { void methodA() {} } class B extends A { void methodB() {} } class C extends B { void methodC() {} } Here, C is a subclass of B, and B is a subclass of A. So, C inherits from both A and B. 3)Hierarchical Inheritance:

# Multiple subclasses inherit from a single superclass. Ex:- class Shape { void draw() {} } class Circle extends Shape { void drawCircle() {} } class Rectangle extends Shape { void drawRectangle() {} } Here, Circle and Rectangle inherit from the common base class Shape. 4)Multiple Inheritance: Java doesn't support multiple inheritance for classes, However, multiple inheritance can be achieved through interfaces. An interface in Java allows a class to implement multiple interfaces. Ex:- interface A { void methodA(); }

# interface B { void methodB(); }

# class DerivedClass implements A, B { public void methodA() {} public void methodB() {} } Here, DerivedClass implements both interfaces A and B, achieving multiple inheritance through interfaces.

# Explain the use of final and Super keyword in JAVA.

# Answer:-

# final:-

# The final keyword is used to restrict the modifiability of something. Final Variables: Declaring a variable as final makes it a constant. The final keyword indicates that the variable's value cannot be changed after initialization.

# Final Methods: A method declared as final cannot be overridden by subclasses.

# Final Classes: A class declared as final cannot be subclassed. In other words, can't extend a final class. super:- The super keyword in Java is used to refer to the object of the parent class or to call methods from the parent class.

# It can be used to call the parent class constructor from the subclass constructor. using super() for no-parameter constructors and super(parameters) for parameterized constructors. It can be also used to call the parent class's method from your subclass method with using super.methodName(parameter).

# Define polymorphism with its need.

# Answer:- Polymorphism is a concept in object-oriented programming (OOP) that allows objects of different types to be treated as objects of a common type. It means "many forms". Compile-time(Static) Polymorphism: Defining multiple methods in the same class with the same name but different parameters.(method overloading) Runtime(Dynamic) Polymorphism: creating a method in a subclass that is already defined in its superclass.(method overriding)

# Code Reusability: Polymorphism allows for the creation of generic code that can work with objects of various types. Flexibility and Extensibility: Polymorphism allows for the creation of flexible code. New classes can be added without modifying existing code. Enhances Encapsulation: Polymorphism helps in achieving encapsulation by abstracting the implementation details from the external code.

# Explain about Encapsulation, Abstraction.

# Answer:-

# Encapsulation and abstraction are two fundamental concepts in object-oriented programming (OOP) that work together to create secure and reusable code.

# Encapsulation: It involves bundling the data (attributes or fields) and the methods (functions or procedures) that operate on the data into a single unit known as a class. The class serves as a blueprint for creating objects, and encapsulation ensures that the internal state of an object is hidden from the outside world. Encapsulation allows the control of access to the internal components of a class. By using access modifiers like private, protected, public & default.

# Abstraction: It is the process of hiding the complex implementation details and showing only the essential features of the object. In java, developers can create abstract classes or interfaces that define a set of methods without providing the implementation. Abstraction allows developers to focus on the essential features of an object while ignoring unnecessary details.

# Suggested Reference:

# 1. <https://www.tutorialspoint.com/java/>

# 2. <https://www.geeksforgeeks.org/>

# 3. <https://www.w3schools.com/java/>

# 4. <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 5**

**AIM: To demonstrate the use of abstract classes and interfaces.**

**Date:**

**CO mapped: CO-2**

**Objectives:**

1. To understand the purpose and usage of abstract classes and interfaces in object-oriented programming. Develop the ability to design and implement abstract classes and interfaces effectively to promote code reusability, ensure consistent behavior in class hierarchies, and facilitate the development of flexible and extensible software systems.
2. Abstract classes and interfaces are important OOP concepts that allow you to define common contracts and behaviors for classes. Achieving this objective will enable you to use these tools to create more modular and maintainable software, especially when dealing with class hierarchies and multiple implementations.

**Background:**

A class that is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have constructors and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

An interface in Java is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

**Practical questions:**

1. Describe abstract class called Shape which has three subclasses say Triangle, Rectangle, Circle. Define one method area() in the abstract class and override this area() in these three subclasses to calculate for specific object, i.e., area() of Triangle subclass should calculate area of triangle etc. Same for Rectangle and Circle.

2. Write a program that demonstrates the instance of operator. Declare interfaces I1 and I2. Interface I3 extends both of these interfaces. Also declare interface I4. Class X implements I3. Class W extends X and implements I4. Create an object of class W. Use the instance of operator to test if that object implements each of the interfaces and is of type X.

3. Write a java program to implement an interface called Exam with a method Pass (int mark) that returns a boolean. Write another interface called Classify with a method Division (int average) which returns a String. Write a class called Result which implements both Exam and Classify. The Pass method should return true if the mark is greater than or equal to 50 else false. The Division method must return "First" when the parameter average is 60 or more, "Second" when average is 50 or more but below 60, "No division" when average is less than 50.

**Observations:**

**1)Program:-**

import java.util.\*;

abstract class Shape {

abstract double area();

}

class Triangle extends Shape {

private double a, b, c;

Triangle(double a, double b, double c) {

this.a = a;

this.b = b;

this.c = c;

}

double area() {

double s = (a + b + c) / 2;

double tempArea = s \* (s - a) \* (s - b) \* (s - c);

return Math.sqrt(tempArea);

}

}

class Rectangle extends Shape {

private double width, length;

Rectangle(double width, double length) {

this.width = width;

this.length = length;

}

double area() {

return this.width \* this.length;

}

}

class Circle extends Shape {

private double radius;

Circle(double radius) {

this.radius = radius;

}

double area() {

return Math.PI \* this.radius \* this.radius;

}

}

public class P5\_1 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Triangle triangle = new Triangle(15, 24, 13);

Rectangle rectangle = new Rectangle(6.8, 8.3);

Circle circle = new Circle(2);

System.out.printf("Area of Triangle: %.2f\n", triangle.area());

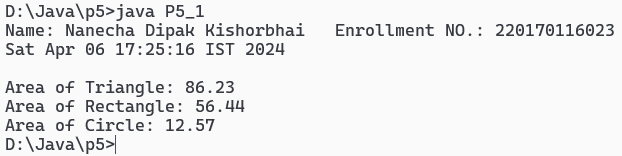
System.out.printf("Area of Rectangle: %.2f\n", rectangle.area());

System.out.printf("Area of Circle: %.2f", circle.area());

}

}

**Output:-**



**2)Program:-**

import java.util.\*;

interface I1 {}

interface I2 {}

interface I3 extends I1, I2 {}

interface I4 {}

class X implements I3 {}

class W extends X implements I4 {}

public class P5\_2 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

W wobj = new W();

if (wobj instanceof W)

System.out.println("wobj is instance of W");

if (wobj instanceof X)

System.out.println("wobj is instance of X");

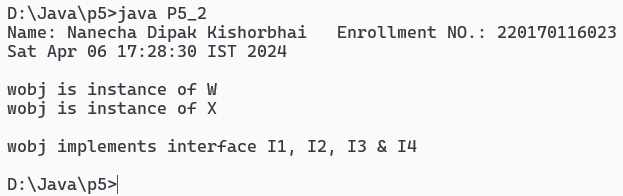
if (wobj instanceof I1 && wobj instanceof I2 && wobj instanceof I3 && wobj instanceof I4)

System.out.println("\nwobj implements interface I1, I2, I3 & I4");

}

}

**Output:-**

****

**3)Program:-**

import java.util.\*;

interface Exam {

boolean Pass(int mark);

}

interface Classify {

String Division(int average);

}

class Result implements Exam, Classify {

public boolean Pass(int mark) {

if (mark < 0 || mark > 100) {

System.out.println("Invalid mark");

System.exit(1);

}

return mark >= 50;

}

public String Division(int average) {

if (average >= 60) return "First";

else if (average >= 50) return "Second";

else return "No division";

}

}

public class P5\_3 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.print("How many subject is in this semester: ");

int sum = 0, sub = se.nextInt();

Result r = new Result();

for (int i = 0; i < sub; i++) {

System.out.print("\nEnter " + (i + 1) + " subject's mark: ");

int mark = se.nextInt();

sum += mark;

String pass = r.Pass(mark) ? "Pass" : "Fail";

System.out.println(pass + " in " + (i + 1) + " subject");

}

String div = r.Division(sum/sub);

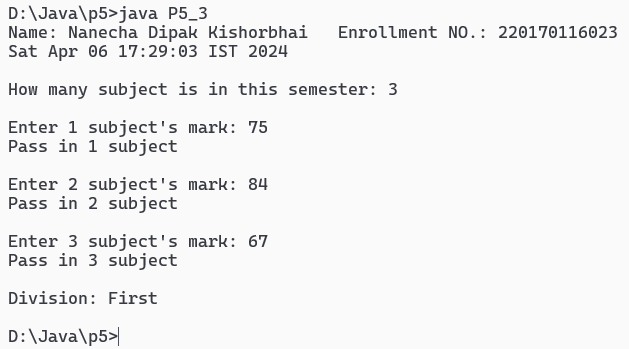
System.out.println("\nDivision: " + div);

se.close();

}

}

**Output:-**

****

**Conclusion:** Abstract classes and interfaces are both essential tools in Java for promoting code reusability and flexibility. Abstract classes allow you to define common behavior among related classes, while interfaces enable you to define contracts for unrelated classes. By utilizing abstract classes and interfaces effectively, we can create modular, maintainable, and extensible codebases that adhere to object-oriented principles such as encapsulation, inheritance, and polymorphism.

**Quiz:**

# 1. Explain how interfaces promote the concept of multiple inheritance in OOP.

# Answer:-

# Interfaces allow classes to inherit behaviors from multiple sources without introducing the complexities and ambiguities associated with traditional multiple inheritance. A class can implement multiple interfaces, each representing a distinct contract for behavior.

# By implementing multiple interfaces, a class can reuse code from various sources, promoting code reusability. This enables developers to compose classes with functionalities from different interfaces, facilitating modular and maintainable code.

# Classes implementing interfaces must adhere to the contract defined by the interface, ensuring consistency in behavior across different implementations. This promotes consistency and predictability in software design.

# Interfaces promote the principle of composition over inheritance, emphasizing the use of interfaces to define behaviors that can be composed together in various combinations, rather than relying solely on class inheritance hierarchies.

# 2. What is an interface, and how does it differ from an abstract class?

# Answer:-

# An interface is a contract that defines a set of method signatures without providing any implementation details. Classes that implement an interface must provide concrete implementations for all the methods declared in the interface. Interfaces facilitate polymorphism, multiple inheritance, and loose coupling between components in software systems.

# Method Implementation: An interface contains only method declarations without any implementation. Classes that implement an interface must provide concrete implementations for all the methods declared in the interface. An abstract class can contain both abstract methods (methods without implementation) and concrete methods (methods with implementation). Subclasses are required to implement abstract methods, but they can also choose to override or use the concrete methods provided by the abstract class.

# Inheritance: Interfaces support multiple inheritance, allowing a class to implement multiple interfaces.

# Abstract classes support single inheritance, meaning a class can inherit from only one abstract class. State: Interfaces cannot contain any instance variables. They define only behavior through method declarations. Abstract classes can contain instance variables, constructors, and other members in addition to methods. This allows abstract classes to define both behavior and state(instance variables).

# 3. When would you choose to use an abstract class over an interface, and vice versa, in your software design?

# Answer:-

# Use an Abstract Class when: When you need to provide a default implementation for some methods that subclasses can optionally override. Abstract classes can contain fields, constructors, and other members in addition to methods, enabling you to encapsulate both state and behavior within a single unit. Abstract class provide a structured way to define a base class with some common behavior while allowing subclasses to extend or override specific methods as needed. Use an Interface When:

# Interfaces declare method signatures without specifying the implementation, allowing implementing classes to provide their own implementation for the specified behavior. When you want to enable multiple inheritance of type (i.e., a class can implement multiple interfaces). When you're aiming for a high level of abstraction.

# 4. Can a class implement multiple interfaces? If so, what benefits does this provide?

# Answer:-

# Yes, a class can implement multiple interfaces. Implementing multiple interfaces allows a class to inherit behaviors and fulfill contracts defined by each interface.

# Implementing multiple interfaces enables classes to reuse code from various sources. By inheriting behaviors from multiple interfaces, a class can leverage functionalities defined in each interface, promoting code reusability and reducing redundancy in code. By implementing multiple interfaces, a class can exhibit polymorphic behavior, meaning it can be treated interchangeably with other classes that implement the same interfaces. This promotes flexibility in design and facilitates code reuse. Interfaces define what a class can do, not how it does it. By implementing multiple interfaces, a class can collaborate with different objects without relying on their specific implementations.

# 5. Can you declare an interface method static? Justify your answer.

# Answer:-

# Yes, in modern Java versions, we can declare a method as static within an interface.

# This change was introduced in Java 8. Declaring static methods in interfaces provides several benefits. First, It allows for the inclusion of utility methods directly within the interface, promoting code organization and reducing redundancy. static methods enable code sharing among classes that implement the interface, enhancing code reuse and maintainability. Additionally, adding static methods to interfaces supports backward compatibility, allowing interfaces to evolve over time without breaking existing implementations.

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 6**

**AIM: To implement packages and exception handling in JAVA application.**

**Date:**

**CO mapped: CO-3**

**Objectives:**

To effectively implement packages and exception handling in a Java application, organizing code into logical modules for improved maintainability, and ensuring robust error handling to enhance the application's reliability and user experience.

**Background:**

A java package is a group of similar types of classes, interfaces, and sub-packages. Package in java can be categorized in two forms, built-in package, and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc. Here, we will have the detailed learning of creating and using user-defined packages.

Exception Handling in Java is one of the powerful mechanisms to handle runtime errors so that the normal flow of the application can be maintained. In this practical, we will learn about Java exceptions, their types, and the difference between checked and unchecked exceptions.

Advantage of Java Package

1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2. Java package provides access protection.
3. Java package removes naming collision.

**Practical questions:**

1. Write a program in Java to develop user defined exception for “Divide by Zero” error.

2. Write a program in Java to demonstrate throw, throws, finally, multiple try block and multiple catch exception.

3. Write a small application in Java to develop Banking Application in which user deposits the amount Rs 1000.00 and then start withdrawing ofRs 400.00, Rs 300.00 and it throws exception "Not Sufficient Fund" when user withdraws Rs 500 thereafter.

4. Write an application that contains a method named average () has one argument that is an array of strings. It converts these to double values and returns their average. The method generates a NullPointerException,if an array elements is null or a NumberFormatException, if an element is incorrectly formatted. Include throws statement in method declaration.

5. Write an application that generates custom exception if first argument from command line argument is 0.

6. A marklist containing reg.no and marks for a subject is given.if the marks are <0,user-defined IllegalMarkException is thrown out and handled with the message "Illegal Mark". For all valid marks, the candidate will be declared as "PASS" if the marks are equal to or greater than 40, otherwise it will be declared as "FAIL".Write a class called IllegalMarkException.

7. Assume that there are two packages, student and exam. A student package contains Student class and the exam package contains Result class. Write a program that generates mark sheet for students.

8. Define a class A in package a pack. In class A, three variables are defined of access modifiers protected, private and public. Define class B in package bpack which extends A and write display method which accesses variables of class A. Define class C in package cpack which has one method display() in that create one object of class A and display its variables. Define class ProtectedDemo in package dpack in which write main() method. Create objects of class B and C and class display method for both these objects.

**Observations:**

**1)Program:-**

import java.util.\*;

class DivisionException extends Exception {

DivisionException() {

super("Can't divide by zero");

}

}

public class P6\_1 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.print("Enter a: "); // numerator

double a = se.nextDouble();

System.out.print("Enter b: "); // denominator

double b = se.nextDouble();

se.close();

double result;

try {

if(b == 0) throw new DivisionException();

result = a/b;

System.out.printf("Result(a/b): %.2f", result);

} catch (DivisionException e) {

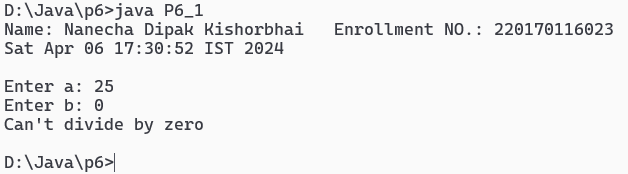
System.out.println(e.getMessage());

}

}

}

**Output:-**

****

**2)Program:-**

import java.util.\*;

class InvalidAge extends Exception {

InvalidAge (String str) {

super(str);

}

}

public class P6\_2 {

static void checkAge(int age) throws InvalidAge {

if (age < 0) {

throw new InvalidAge("Age can't be negative");

} else if (age < 18) {

throw new InvalidAge("You must be at least 18 years old");

} else {

System.out.println("\nYou are eligible");

}

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.print("Enter age to check if a person is eligible for driving: ");

try {

int age = se.nextInt();

checkAge(age);

} catch (InvalidAge e) {

System.out.println("\n" + e.getMessage());

} catch (InputMismatchException e) {

System.out.println("Enter age in integer");

} catch (Exception e) {

System.out.println("Some exception occurred:");

} finally {

se.close();

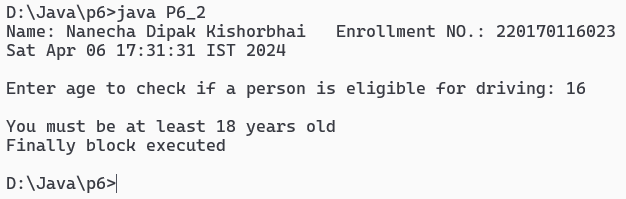
System.out.println("Finally block executed");

}

}

}

**Output:-**

****

**3)Program:-**

import java.util.\*;

class NotSufficientFundException extends Exception {

NotSufficientFundException() {

super("Not Sufficient Fund");

}

}

class Bank {

private int accNo;

private String custName;

private double balance;

Bank(int accNo, String name, double balance) {

this.accNo = accNo;

this.custName = name;

this.balance = balance;

}

void deposit(double data) {

this.balance += data;

System.out.println("After deposit balance: " + this.balance);

}

void withdraw(double data) throws NotSufficientFundException {

if (this.balance - data < 0) {

throw new NotSufficientFundException();

}

this.balance -= data;

System.out.println("After withdraw balance: " + this.balance);

}

void printDetail() {

System.out.println("\nAccount Number: " + this.accNo);

System.out.println("Name: " + this.custName);

System.out.println("Balance: " + this.balance);

}

}

public class P6\_3 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Bank acc = new Bank(101, "Raj Modi", 1000);

try {

acc.withdraw(400);

acc.withdraw(300);

acc.withdraw(500);

} catch (NotSufficientFundException e) {

acc.printDetail();

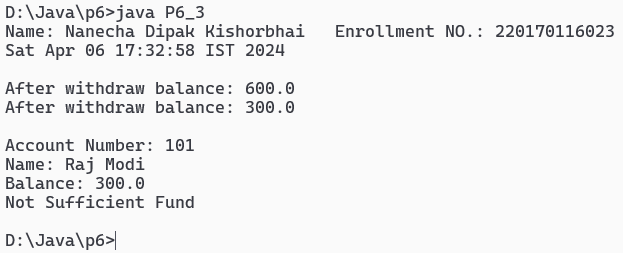
System.out.println(e.getMessage());

}

}

}

**Output:-**

****

**4)Program:-**

import java.util.\*;

public class P6\_4 {

static double average(String[] numbers) throws NullPointerException, NumberFormatException {

if (numbers == null) {

throw new NullPointerException("String array can't be null");

}

else if (numbers.length == 0) {

throw new NumberFormatException("String array is empty");

}

double sum = 0;

for (String str : numbers) {

if (str == null) {

throw new NullPointerException("Array element can't be null");

}

try {

double num = Double.parseDouble(str);

sum += num;

} catch (NumberFormatException e) {

throw new NumberFormatException(str + " is not in a valid number format");

}

}

return sum / numbers.length;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

String[] numbers = {"25", "36.s4", "32.7"};

System.out.print("Number array: ");

for (String e : numbers) {

System.out.print(e + " ");

}

System.out.println("");

try {

double avg = average(numbers);

System.out.printf("Average: %.2f\n", avg);

} catch (NullPointerException e) {

System.out.println("\n" + e.getMessage());

} catch (NumberFormatException e) {

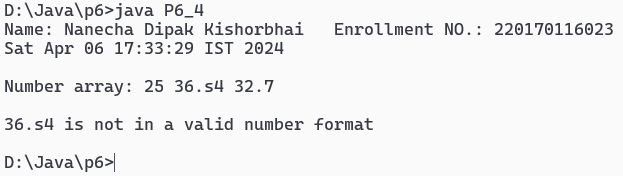
System.out.println("\n" + e.getMessage());

}

}

}

**Output:-**

****

**5)Program:-**

import java.util.\*;

class FirstElementZero extends Exception {

public String getMessage() {

return "First argument of command line argument can't be 0";

}

}

public class P6\_5 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

try {

if (args[0].equals("0")) throw new FirstElementZero();

else System.out.println("First argument is not 0");

} catch (FirstElementZero e) {

System.out.println(e.getMessage());

} catch(ArrayIndexOutOfBoundsException e) {

System.out.println("Enter atleast one command line argument");

} catch (Exception e) {

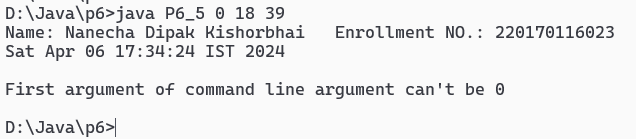
System.out.println("Some exception occurred:");

}

}

}

**Output:-**

****

**6)Program:-**

import java.util.\*;

class IllegalMarkException extends Exception {

IllegalMarkException(String str) {

super(str);

}

}

class MarkList {

private int regNo;

private int marks;

MarkList(int r, int marks) throws IllegalMarkException {

if (marks < 0) throw new IllegalMarkException("\nIllegal Mark for Reg No: " + r);

this.regNo = r;

this.marks = marks;

}

String checkResult(){

return (marks >= 40) ? "PASS" : "FAIL";

}

int getRegNo() {

return regNo;

}

}

public class P6\_6 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

MarkList[] list = new MarkList[3];

try {

list[0] = new MarkList(1, 65);

System.out.println("Reg No: " + list[0].getRegNo() + ", Result: " + list[0].checkResult());

list[1] = new MarkList(2, 35);

System.out.println("Reg No: " + list[1].getRegNo() + ", Result: " + list[1].checkResult());

list[2] = new MarkList(3, -2);

System.out.println("Reg No: " + list[2].getRegNo() + ", Result: " + list[2].checkResult());

} catch (IllegalMarkException e) {

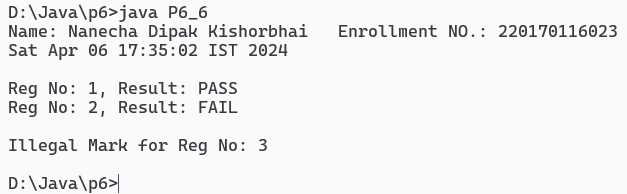
System.out.println(e.getMessage());

}

}

}

**Output:-**

****

**7)Program:-**

// student.Student

package student;

public class Student {

private int enrollNo, cnt;

private String name;

private String[] subjects;

private int[] marks;

public Student(int e, String name, int size) {

this.enrollNo = e;

this.name = name;

this.subjects = new String[size];

this.marks = new int[size];

cnt = 0;

}

public void addMarks(String sub, int marks) {

this.subjects[this.cnt] = sub;

this.marks[this.cnt] = marks;

this.cnt++;

}

public int getEnrollNo() { return this.enrollNo;}

public String getName() { return this.name;}

public String[] getSubjects() { return this.subjects;}

public int[] getMarks() { return this.marks;}

}

// exam.Result

package exam;

import student.Student;

public class Result {

private Student stu;

private int totalMark;

private String result;

public Result(Student s) {

stu = s;

totalMark = 0;

result = "";

}

public void markSheet() {

int check = 0;

System.out.println("MarkSheet");

System.out.println("Name: " + stu.getName());

System.out.println("Enrollment No: " + stu.getEnrollNo());

System.out.println("Subject\t|\tMarks\n");

for (int i = 0; i < stu.getSubjects().length; i++) {

System.out.println(stu.getSubjects()[i] + "\t|\t" + stu.getMarks()[i]);

totalMark += stu.getMarks()[i];

if(stu.getMarks()[i] < 33) check++;

}

result = check == 0 ? "Pass" : "Fail";

System.out.println("\nToal Marks: " + totalMark + "\nResult: " + result);

if(result.equals("Fail")) System.out.println("Current Backlog: " + check);

else System.out.println("Congratulations!! You have passed this exam.");

}

}

// using Student & Result to print mark sheet

import student.Student;

import exam.Result;

import java.util.\*;

public class P6\_7{

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

Student stu = new Student(101, "Raj Modi", 4);

stu.addMarks("OOPs", 73);

stu.addMarks("COA", 60);

stu.addMarks("OSV", 60);

stu.addMarks("DM", 63);

Result r = new Result(stu);

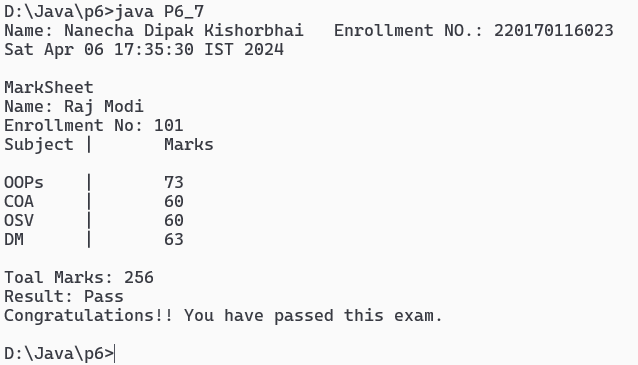
r.markSheet();

se.close();

}

}

**Output:-**

****

**8)Program:-**

// apack.A

package apack;

public class A {

public String A1 = "Public";

protected String A2 = "Protected";

private String A3 = "Private";

}

// bpack.B

package bpack;

import apack.A;

public class B extends A{

public void display() {

System.out.println("\nMethod of B:");

System.out.println("A1: " + A1);

System.out.println("A2: " + A2);

System.out.println("Other variables can't be accessible.");

}

}

// cpack.C

package cpack;

import apack.A;

public class C {

public void display() {

System.out.println("\nMethod of C:");

A aobj = new A();

System.out.println("A1: " + aobj.A1);

System.out.println("Other variables can't be accessible.");

}

}

// dpack.ProtectedDemo

package dpack;

import bpack.B;

import cpack.C;

import java.util.\*;

public class ProtectedDemo {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t);

B bobj = new B();

bobj.display();

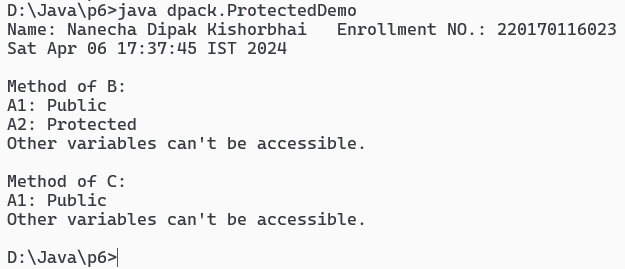
C cobj = new C();

cobj.display();

}

}

**Output:-**



**Conclusion:** Packages provide a mechanism to organize code into logical units, improving readability and maintainability for larger projects. By grouping related classes and interfaces together, we can keep codebase organized and prevent naming conflicts. Package in java can be categorized in two forms, built-in package, and user-defined package. Exception handling equips you to deal with unexpected errors that might arise during program execution. By utilizing try-catch blocks, exceptions can be gracefully handled, preventing program crashes and providing informative messages to the user. Effective use of both packages and exception handling leads to cleaner, more reliable, and easier-to-maintain Java programs.

**Quiz:**

# 1. Explain the benefits of organizing classes into packages in a Java application.

# Answer:-

# Modularity: Packages provide a way to organize related classes and interfaces, making the codebase modular and easier to manage. This modular structure enhances code readability and maintainability.

# Encapsulation: Packages allow developers to encapsulate classes and restrict access using access modifiers like public, private, protected, and default. This helps in controlling the visibility of classes and reduces the risk of unintended access or modification.

# Namespace Management: Packages provide a namespace mechanism to prevent naming conflicts. By grouping classes into packages with unique names, developers can avoid naming collisions, especially when integrating third-party libraries or frameworks.

# Reusability: Well-organized packages facilitate code reuse by enabling developers to easily locate and reuse classes across different projects

# 2. How do you declare and define a package in Java?

# Answer:-

# A package in Java is a way to organize classes and interfaces into namespaces, helping to avoid naming conflicts and providing a modular structure to codes.

# Declare the Package: At the beginning of each source file that belongs to the package, use the package keyword followed by the package name. This declaration should be the first non-comment statement in the file.

# Example: package example.mypackage; Here, this statement declares that the classes in the file belong to the example.mypackage package. Place Class Files in the Corresponding Directory Structure: Organize source code files in a directory structure that mirrors the package hierarchy.

# example

# └── mypackage

# └── MyClass.java

# 3. What is an exception in Java, and why is exception handling important in software development?

# Answer:-

# An exception in Java is an event that disrupts the normal flow of a program’s execution. It occurs during the runtime of a program and can be caused by various factors such as invalid user input, hardware malfunctions, or logical errors in the program. Exceptions are represented by objects of classes that are subclasses of the Throwable class.

# Error Recovery: Exception handling allows developers to gracefully recover from unexpected errors or exceptional situations. Instead of suddenly terminating the program, developers can catch and handle exceptions, providing alternative actions or error messages to users.

# Debugging and Maintenance: Exception handling facilitates debugging and maintenance efforts. When exceptions occur, they provide valuable information such as stack traces, which help developers identify the cause of errors and troubleshoot issues effectively.

# User Experience: Effective exception handling contributes to a positive user experience by providing informative error messages and guiding users through unexpected situations. Well-handled exceptions can prevent frustration and confusion among users when errors occur.

# 4. Explain the try-catch-finally block and its role in handling exceptions.

# Answer:-

# try block: The try block encloses the code that may throw exceptions. When an exception occurs within the try block, the execution of the try block is halted, and the Java Virtual Machine (JVM) looks for an appropriate catch block to handle the exception.

# catch block(s): A catch block follows the try block and is used to catch and handle specific types of exceptions. Each catch block specifies the type of exception it can handle. If an exception matches the type specified in a catch block, the corresponding catch block is executed to handle the exception. Multiple catch blocks can be chained together to handle different types of exceptions.

# finally block: The finally block, if present, follows the try-catch block. It contains code that is always executed, regardless of whether an exception occurs or not. The finally block is typically used to release resources, such as closing files or releasing database connections, that were acquired within the try block. It ensures that essential cleanup tasks are performed, even in the event of an exception.

# The role of the try-catch-finally block in handling exceptions: Catch and Handle Exceptions: By enclosing potentially error-prone code within a try block and providing corresponding catch blocks, developers can catch and handle exceptions gracefully, preventing program crashes and allowing for error recovery. Ensure Resource Cleanup: The finally block ensures that critical resources are properly released, regardless of whether an exception occurs or not. This helps prevent resource leaks and ensures that the program maintains proper resource management practices. Control Flow: The try-catch-finally block allows developers to control the flow of execution in response to exceptions.

# 5. What is difference between throw and throws?

# Answer:-

# throw: throw is a keyword used to explicitly throw an exception within a method or block of code. It is used when a specific condition arises that warrants an exceptional situation. When throw is encountered, it creates an instance of an exception class and raises it.

# The syntax is as follows:

# throw new SomeException("An error occurred");

# Here, SomeException is the type of exception being thrown, and “An error occurred” is an optional message associated with the exception. throws: throws is a keyword used in method declarations to indicate that the method may throw one or more types of exceptions. It specifies the exceptions that the method might propagate to its caller, allowing the caller to handle or propagate the exceptions further. The syntax is as follows:

# public void methodName() throws SomeException { // method implementation}

# Here, SomeException is the type of exception that the methodName method may throw. Multiple exceptions can be listed separated by commas.

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 7**

**AIM: To demonstrate I/O from files.**

**Date:**

**CO mapped: CO-4**

**Objectives:**

To showcase proficiency in reading data from and writing data to files in various formats using programming languages, demonstrating the ability to implement reliable and efficient file I/O operations, which are essential for tasks such as data storage, retrieval, and processing in software applications.

**Background:**

Java I/O (Input and Output) is used to process the input and produce the output. Java uses the concept of a stream to make I/O operations fast. The java.io package contains all the classes required for input and output operations.

**Practical questions:**

1. Write a program that removes all the occurrences of a specified string from a text file. For example, invoking java Practical7\_1 John filename removes the string John from the specified file. Your program should read the string as an input.

2. Write a program that will count the number of characters, words, and lines in a file. Words are separated by whitespace characters. The file name should be passed as a command-line argument.

3. Write a program to create a file named Practical7.txt if it does not exist. Write 100 integers created randomly into the file. Integers are separated by spaces in the file. Read the data back from the file and display the data in increasing order.

**Observations:**

**1)Program:-**

import java.util.\*;

import java.io.\*;

public class P7\_1 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

try {

File f1 = new File(args[1]);

File f2 = new File("temp.txt");

BufferedReader br = new BufferedReader(new FileReader(f1));

BufferedWriter bw = new BufferedWriter(new FileWriter(f2));

String str;

boolean firstLine = true;

while ((str = br.readLine()) != null) {

str = str.replaceAll(args[0], "");

if (!firstLine) {

bw.newLine(); // add a newline untill the last line

} else {

firstLine = false;

}

bw.write(str);

}

br.close();

bw.close();

if (f1.delete() == false) System.out.println("Can't delete the " + args[0] + " file");

if (f2.renameTo(f1) == false) System.out.println("Can't rename the temporary file");

System.out.println(args[0] + " successfully removed from the '" + args[1] + "' file.");

} catch (FileNotFoundException e) {

System.out.println(e.getMessage());

} catch (IOException e) {

System.out.println(e.getMessage());

} catch (Exception e) {

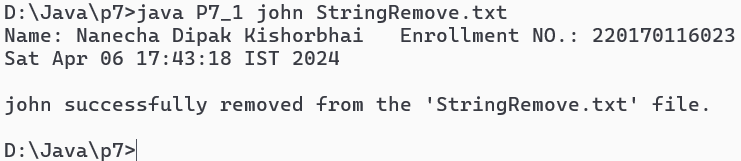
System.out.println(e.getMessage());

}

}

}

**Output:-**

****

**2)Program:-**

import java.util.\*;

import java.io.\*;

public class P7\_2 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

try {

FileReader fr = new FileReader(args[0]);

int chars = 0, word = 0, lines = 0, x;

while ((x = fr.read()) != -1) {

chars++;

if (x == (int)' ') word++;

else if (x == (int)'\n') lines++;

}

if ((chars == 0) && (word == 0) && (lines == 0)) System.out.println(args[0] + " is empty.");

else System.out.println("Characters: " + chars + ", Word: " + ++word + ", Lines: " + ++lines);

fr.close();

} catch (FileNotFoundException e) {

System.out.println(e.getMessage());

} catch (IOException e) {

System.out.println(e.getMessage());

} catch (Exception e) {

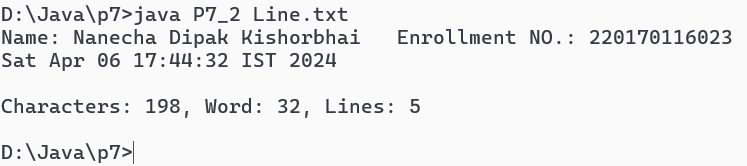
System.out.println(e.getMessage());

}

}

}

**Output:-**

****

**3)Program:-**

import java.util.\*;

import java.io.\*;

public class P7\_3 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

try {

FileWriter fw = new FileWriter("Array.txt");

for (int i = 0; i < 100; i++) {

fw.write(Integer.toString((int)(Math.random()\*1000)));

fw.append(' ');

}

fw.close();

int[] arr = new int[100];

FileReader fr = new FileReader("Array.txt");

int x, i = 0;

while ((x = fr.read()) != -1) {

String str = "";

while (x != (int)' ') {

str += Character.toString((char)x);

x = fr.read();

}

arr[i++] = Integer.parseInt(str);

}

fr.close();

Arrays.sort(arr);

System.out.println("Sorted array: ");

for (int e : arr) {

System.out.print(e + " ");

}

} catch (FileNotFoundException e) {

System.out.println(e.getMessage());

} catch (IOException e) {

System.out.println(e.getMessage());

} catch (Exception e) {

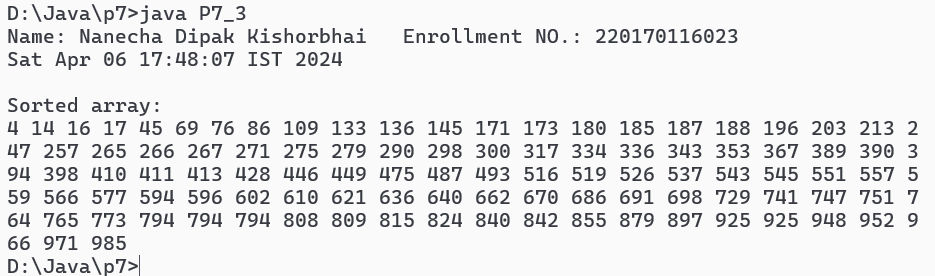
System.out.println(e.getMessage());

}

}

}

**Output:-**

****

**Conclusion:** File I/O is a fundamental concept in programming and is widely used in various applications, ranging from simple data processing tasks to complex file management systems. By demonstrating file I/O, developers gain insights into handling text-based and binary data, managing resources efficiently, dealing with exceptions, and ensuring data integrity and security. Java uses the concept of a stream to make I/O operations fast.

**Quiz:**

# 1. What is file input/output (I/O), and why is it important in software development?

# Answer:- File input/output (I/O) refers to the process of reading data from and writing data to files on a computer system. In software development, file I/O is crucial because it enables programs to store and retrieve information persistently.

# Data Persistence: File I/O allows software applications to store data permanently on disk storage. This is essential for saving user preferences, application state, and other important information that needs to persist beyond a single program execution. Data Sharing: provide a common medium for sharing data between different programs and systems. For example, one program may generate a file that another program can read and process, facilitating data exchange and interoperability. Data Backup: Files can be used for creating backups of critical data. By writing data to files, software developers can implement backup and recovery mechanisms to protect against data loss due to system failures or other unforeseen events. Platform Independence: File I/O operations are typically supported across different operating systems and programming languages, making it a versatile and widely applicable technique in software development.

# 2. What are the common modes for opening files, and how do they differ (e.g., read, write, append)?

# Answer:- In Java, when working with file I/O operations, there are several common modes for opening files. These modes determine how the file can be accessed and modified.

# Read Mode ("r"): This mode allows you to open a file for reading only. You can read data from the file but cannot modify or write to it. If the specified file does not exist, an IOException is thrown. Ex:- FileReader fileReader = new FileReader("filename.txt");

# Write Mode ("w"): This mode allows you to open a file for writing only. If the file already exists, it will be truncated (emptied) before writing new data to it. If the file does not exist, a new file will be created. Ex:- FileWriter fileWriter = new FileWriter("filename.txt"); Append Mode ("a"): This mode allows you to open a file for appending data. If the file exists, new data will be added to the end of the file. If the file does not exist, a new file will be created. Ex:- FileWriter fileWriter = new FileWriter("filename.txt", true); Read/Write Mode ("rw"): This mode allows you to open a file for both reading and writing. You can read data from the file and also write new data to it. If the file does not exist, a FileNotFoundException is thrown. Ex:- RandomAccessFile file = new RandomAccessFile("filename.txt", "rw");

# 3. Describe the concept of file streams and how they are used in file I/O operations.

# Answer:- File streams enable reading data from files into memory (input streams) and writing data from memory to files (output streams). They serve as the intermediary for transferring data between the program and external storage.

# Input Streams: Input streams are used for reading data from files into a Java program. They provide methods for reading bytes or characters sequentially from a file.

# Ex:- FileInputStream inputStream = new FileInputStream("input.txt");

# Output Streams: Output streams are used for writing data from a Java program to files. They provide methods for writing bytes or characters sequentially to a file. Ex:- FileOutputStream outputStream = new FileOutputStream("output.txt"); Character Streams: Character streams, such as FileReader and FileWriter, are used specifically for reading and writing text data represented as characters. They handle character encoding and decoding automatically, making it easier to work with text files. Ex:- FileReader fileReader = new FileReader("textfile.txt"); FileWriter fileWriter = new FileWriter("output.txt");

# Byte Streams: Byte streams, such as FileInputStream and FileOutputStream, are used for reading and writing binary data in the form of bytes. They are suitable for handling non-textual data, such as images, audio files, and serialized objects.

# Ex:- FileInputStream fileInputStream = new FileInputStream("image.png");

# FileOutputStream fileOutputStream = new FileOutputStream("output.png");

# 4. Write short notes about I/O stream classes.

# Answer:- I/O stream classes in Java provide the foundation for performing input and output operations, both for handling text-based data and binary data.

# Here are short notes about some of the commonly used I/O stream classes:

# InputStream and OutputStream: InputStream, Abstract class for reading input bytes. OutputStream, Abstract class for writing output bytes. Used for handling binary data. FileInputStream and FileOutputStream: FileInputStream, Reads bytes from a file. FileOutputStream, Writes bytes to a file. Used for reading and writing binary data to/from files. FileReader and FileWriter: FileReader, Reads characters from a file using the default character encoding. FileWriter, Writes characters to a file using the default character encoding. Used for reading and writing text data to/from files. BufferedReader and BufferedWriter: BufferedReader, Reads text from a character-input stream with buffering for efficiency. BufferedWriter: Writes text to a character-output stream with buffering for efficiency. Used for efficient reading and writing of text data.

# PrintStream and PrintWriter: PrintStream, Provides methods to print formatted representations of objects to an output stream. PrintWriter, Provides methods to write formatted text to an output stream. Used for formatted output to text files or other output streams.

# 

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 8**

**AIM: To learn JAVA FX UI Controls.**

**Date:**

**CO mapped: CO-5**

**Objectives:**

* 1. To gain proficiency in JavaFX UI controls, including understanding their features and capabilities, and developing the ability to create interactive and visually appealing user interfaces for Java applications. This knowledge will enable the design and development of user-friendly, responsive, and feature-rich graphical user interfaces (GUIs) in Java applications.
  2. Learning JavaFX UI controls is essential for creating modern and engaging graphical user interfaces for Java applications. This objective emphasizes not only understanding the various UI controls but also the practical skills to design and implement user interfaces effectively.

**Background:**

Every user interface considers the following three main aspects –

UI elements − These are the core visual elements that the user eventually sees and interacts with. JavaFX provides a huge list of widely used and common elements varying from basic to complex, which we will cover in this practical.

Layouts − They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in the Layout chapter.

Behavior − These are events that occur when the user interacts with UI elements.

JavaFX provides several classes in the package javafx.scene.control. To create various GUI components (controls), JavaFX supports several controls such as date picker, button text field, etc.

Each control is represented by a class; you can create a control by instantiating its respective class.

### Common elements in a JavaFX application

All JavaFX applications contain the following elements:

1. A main window, called a stage in JavaFX.
2. At least one Scene in the stage.
3. A system of panes and boxes to organize GUI elements in the scene.
4. One or more GUI elements, such as buttons and labels.

The usual procedure for setting up a scene is to build it from the bottom up. First, we make the GUI elements, then we make boxes and panes to organize the elements, and finally, we put everything in the scene.

All JavaFX elements such as boxes and panes that are meant to contain other elements have a child list that we can access via the getChildren() method. We put elements inside other elements by adding things to child lists. In the code above you can see the button and the label objects being added as children of a VBox, and the VBox, in turn, is set as the child of a StackPane.

In addition to setting the structure for the window, we also call methods designed to set the properties of various elements. For example, the code in this example uses the button's setText() method to set the text the button will display.

Follow the procedure outlined in the section above to make a new JavaFX application. Replace the start() method in the App class with the following code:

public void start(Stage primaryStage) {

Button btn = new Button();

btn.setText("Say 'Hello World'");

StackPane root = new StackPane();

VBox box = new VBox();

box.getChildren().add(btn);

Label label = new Label();

box.getChildren().add(label);

root.getChildren().add(box);

btn.setOnAction(new ClickHandler(label));

Scene scene = new Scene(root, 300, 250);

primaryStage.setTitle("Hello World!");

primaryStage.setScene(scene);

primaryStage.show();

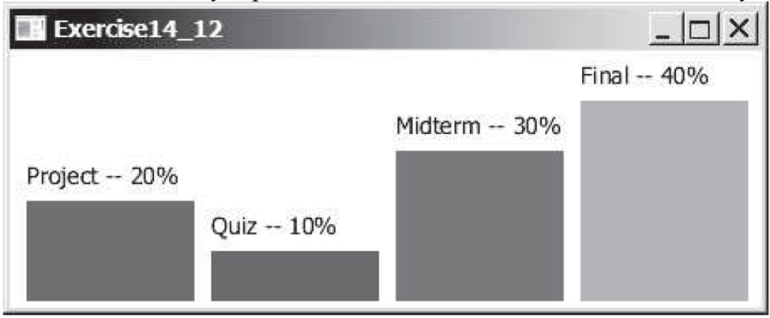
}

**Practical questions:**

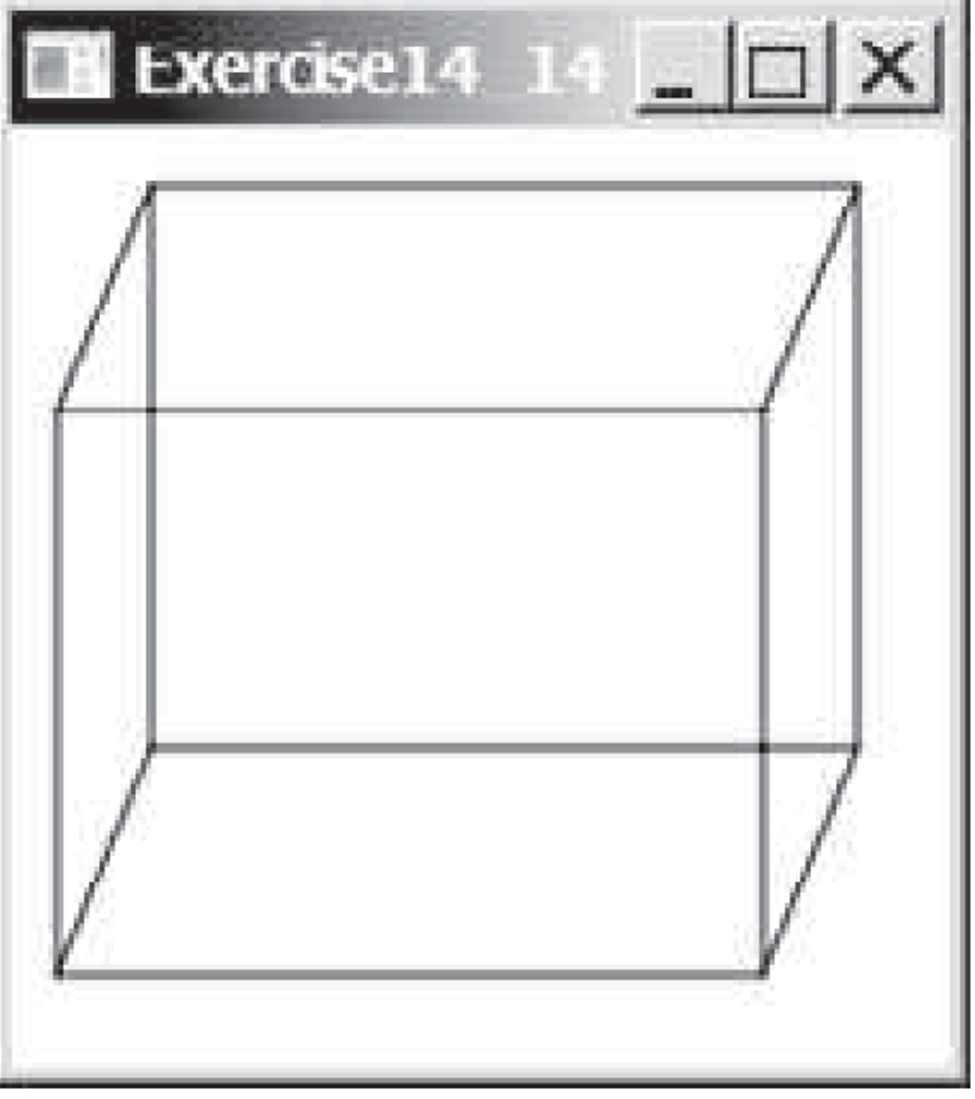
1. Write a program that displays five texts vertically, as shown in Figure. Set a random color and opacity for each text and set the font of each text to Times Roman, bold, italic, and 22 pixels.



2. Write a program that uses a bar chart to display the percentages of the overall grade represented by projects, quizzes, midterm exams, and the final exam, as shown in Figure b. Suppose that projects take 20 percent and are displayed in red, quizzes take 10 percent and are displayed in blue, midterm exams take 30 percent and are displayed in green, and the final exam takes 40 percent and is displayed in orange. Use the Rectangle class to display the bars. Interested readers may explore the JavaFXBarChart class for further study.



3. Write a program that displays a rectanguloid, as shown in Figure a. The cube should grow and shrink as the windo w grows or shrinks.



**Observations:**

**1)Program:-**

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.layout.HBox;

import javafx.scene.paint.Color;

import javafx.scene.text.Font;

import javafx.scene.text.FontPosture;

import javafx.scene.text.FontWeight;

import javafx.scene.text.Text;

import javafx.stage.Stage;

import java.util.\*;

public class P8\_1 extends Application {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t);

try {

launch(args);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

public void start(Stage stage) throws Exception {

HBox hbox = new HBox();

hbox.setAlignment(Pos.CENTER);

hbox.setPadding(new Insets(25, 25, 25, 25));

Text[] t = new Text[5];

for (int i = 0; i < 5; i++) {

t[i] = new Text("Java");

t[i].setFont(Font.font("Times Roman", FontWeight.BOLD, FontPosture.ITALIC, 22));

t[i].setFill(new Color(Math.random(), Math.random(), Math.random(), Math.random()));

t[i].setRotate(90);

hbox.getChildren().add(t[i]);

}

stage.setScene(new Scene(hbox));

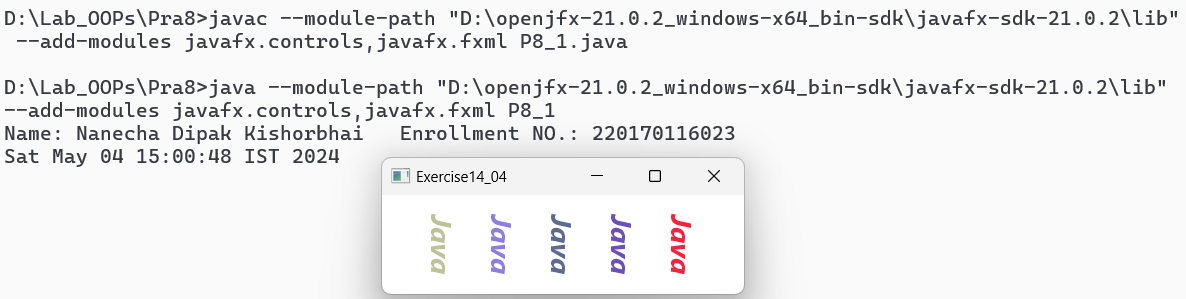
stage.setTitle("Exercise14\_04");

stage.show();

}

}

**Output:-**



**2)Program:-**

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.scene.paint.Color;

import javafx.scene.shape.Rectangle;

import javafx.scene.text.Text;

import javafx.stage.Stage;

import java.util.\*;

public class P8\_2 extends Application {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t);

try {

launch(args);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

public void start(Stage stage) throws Exception {

HBox hbox = new HBox();

hbox.setAlignment(Pos.CENTER);

hbox.setSpacing(10);

hbox.setPadding(new Insets(5, 5, 5, 5));

String[] components = { "Project", "Quiz", "Midterm", "Final" };

int[] percentages = { 20, 10, 30, 40 };

Color[] colors = { Color.RED, Color.BLUE, Color.GREEN, Color.ORANGE };

for (int i = 0; i < components.length; i++) {

VBox vbox = new VBox();

vbox.setAlignment(Pos.BOTTOM\_LEFT);

vbox.setSpacing(5);

Rectangle bar = new Rectangle();

bar.setFill(colors[i]);

bar.setWidth(100);

bar.setHeight(percentages[i] \* 3);

Text t = new Text(components[i] + " -- " + percentages[i] + "%");

vbox.getChildren().addAll(t, bar);

hbox.getChildren().add(vbox);

}

Scene scene = new Scene(hbox);

stage.setTitle("Exercise14\_12");

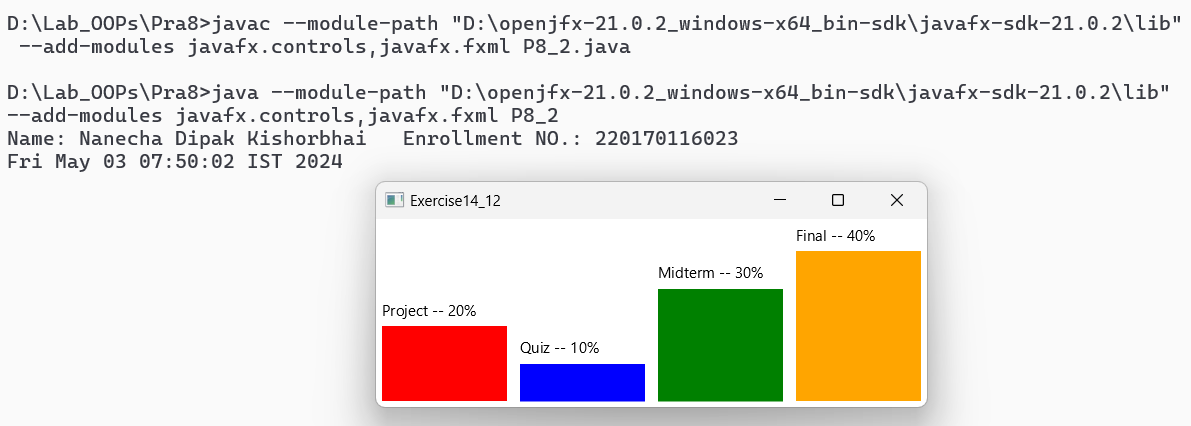
stage.setScene(scene);

stage.show();

}

}

**Output:-**



**3)Program:-**

**Output:-**

**Conclusion:** JavaFX UI controls offer a powerful and versatile toolkit for crafting user interfaces in Java applications. It streamline UI development. They're user-friendly, customizable for intuitive interfaces, and integrate with CSS for a polished look. JavaFX provides a huge list of widely used and common elements varying from basic to complex. Layouts define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). Behavior are events that occur when the user interacts with UI elements.

**Quiz:**

# Explain the evolution of Java GUI technologies since awt,swing and JavaFX.

# Answer:-

# AWT (Abstract Window Toolkit): AWT was the first GUI toolkit included in the Java Development Kit (JDK). It provided a set of classes for creating and managing GUI components such as windows, buttons, and text fields. AWI is platform-dependent, meaning the look and feel of the UI could vary depending on the operating system. Development with AWT could be complex and low-level. Swing (Java Foundation Classes): It was designed to address the limitations of AWT and provide a more powerful and flexible GUI framework. Swing components are entirely written in Java, making them platform-independent and consistent across different operating systems. Swing also provided features like double buffering for smoother graphics rendering and support for drag-and-drop operations. Provided a higher-level abstraction for easier GUI development compared to AWT. JavaFX: JavaFX is a modern GUI toolkit that offers a rich set of features for building desktop, mobile, and web applications. It provides a declarative language (FXML) for designing UI layouts, CSS for styling, and a powerful graphics engine for rendering. JavaFX supports hardware acceleration, which leads to better performance and smoother animations. With JavaFX, developers can create applications with modern UI designs and enhanced user experiences.

# What is the purpose of a TextField control, and how can it be used to collect user input?

# Answer:- A TextField control in GUI programming is a component that allows users to input a single line of text. Its purpose is to provide a user-friendly way for users to enter textual data into an application.

# Here's how a TextField control can be used to collect user input: Placement: Add the TextField control to your application's GUI layout, typically within a form or alongside a label indicating what kind of information is expected. User Interaction: When the user clicks or taps on the TextField, it becomes active for typing, allowing the user to input text. Data Capture: As the user types, the TextField captures the entered characters and stores them internally.

# Processing User Input: After retrieving the user input, the application can process it as needed, such as performing validation, saving to a database, or triggering certain actions based on the input provided by the user.

# How to create an ImageView from an Image, or directly from a file or a URL?.

# Answer:-

# Three ways of creating an ImageView depending on image source:

# 1. Using an existing Image object:

# If an Image object is already loaded, developers can directly create an ImageView with that Image as its argument. Ex:- Image image = new Image("path\_to\_image.jpg");

# ImageView imageView = new ImageView(image);

# 2. Creating an ImageView from a file path:

# We can't directly use a file path in the Image constructor. Instead, convert the file path to a URL first. Ex:- String imagePath = "path\_to\_image.jpg";

# Image image = new Image(new File(imagePath).toURI().toURL().toString());

# ImageView imageView = new ImageView(image);

# 3. Creating an ImageView from a URL:

# If image resides on the internet or a remote location, use the URL directly in the Image constructor. Ex:- String imageUrl = "https://www.example.com/image.png";

# Image image = new Image(imageUrl);

# ImageView imageView = new ImageView(image);

# What are the primary layout controls in JavaFX, and how do they impact the arrangement of UI components?

# Answer:-

# HBox (Horizontal Box): Arranges child nodes horizontally from left to right, similar to a row in a table. Nodes added to an HBox are placed side by side horizontally. Ideal for placing elements like buttons, labels, or text fields side-by-side.

# VBox (Vertical Box): Arranges child nodes vertically from top to bottom, similar to a column in a table. Each node added to a VBox is stacked vertically, one on top of the other. Suitable for stacking elements like labels, text areas, or list views on top of each other.

# FlowPane: FlowPane lays out its child nodes in a flowing manner, wrapping them to the next row or column when necessary. Useful for dynamically adding elements that might not have fixed sizes. GridPane: GridPane organizes its child nodes in a grid of rows and columns. Offers more precise control over positioning elements at specific locations within the grid. GridPane is flexible and allows complex layouts with components spanning multiple rows and columns. BorderPane: BorderPane divides the scene into five regions: top, bottom, left, right, and center. Components placed in these regions are resized automatically to fit within their respective regions. It's often used for creating a basic layout with a header, footer, sidebar, and main content area.

# What is CSS, and how is it used for styling JavaFX UI controls?

# Answer:- CSS (Cascading Style Sheets) is a language used to define the presentation of a document, typically a web page. It separates the content of a web page from its appearance, allowing for consistent styling and easier maintenance. Here's how CSS is used for styling JavaFX UI controls: Selectors: CSS selectors are used to target specific UI controls or groups of controls. Selectors can be based on the control's ID, class, type, or other attributes.

# Properties and Values: CSS properties are used to define various aspects of a control's appearance, such as its background color, font size, padding, and border style. Each property is assigned a value that specifies how the property should be applied.

# Style Sheets: CSS styles can be applied to JavaFX applications either inline within the Java code or externally through style sheets. External style sheets are often preferred as they provide better separation of concerns and make it easier to maintain and update the styling of the application.

# Inheritance and Cascading: CSS follows the principles of inheritance and cascading, allowing styles to be inherited from parent controls and overridden or combined with styles defined in other style sheets.

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 9**

**AIM: To implement event handling and animation.**

**Date:**

**CO mapped: CO-5**

**Objectives:**

To proficiently implement event handling and animation in software applications, fostering user interaction and engagement. Mastery of event-driven programming and animation techniques will empower the creation of dynamic, responsive, and visually captivating software experiences that cater to user needs and preferences.

**Background:**

### **Responding to user events**

For a GUI application to be interactive, various elements such as buttons have to be able to respond to interactions from the user, such as clicks. In GUI applications user actions such as mouse clicks and key presses are called *events*. To set up an element such as a button to respond to user events, we arrange to connect special *event handling* code to the button.

Our first example demonstrates one way to do this in JavaFX. The first step is to connect an object to the button as the button's event handler via the button's setOnAction() method. The requirement here is that the object that we link to the button has to implement a particular interface, the EventHandler<ActionEvent> interface. That interface has one method in it, a handle() method that will get called when the user clicks on the button.

For the event handler code to do something useful, it will typically need to have access to one or more elements in the scene that will be affected by the button click. In this example, clicking the button will trigger a change in the text displayed in a label in the scene. To make this all work, the class we set up needs to have a member variable that is a reference to the label object. The code in handle() will use that reference to change the text shown in the label when the user clicks on the button.

Also, insert the code for the following class at the bottom of the App.java file:

classClickHandler implements EventHandler<ActionEvent> {

publicClickHandler(Label label) {

this.label = label;

}

public void handle(ActionEventevt) {

label.setText("Hello, World!");

}

private Label label;

}

### **A better way to handle events**

Although the process for linking an event handler to a button is fairly straightforward, it is a little clunky. This process can get even more tedious when we start building applications with many buttons that need event handlers. As a fix for this, JavaFX allows us to use a simpler mechanism to set up event handlers.

To see how this mechanism works, remove the ClickHandler class completely and replace the line of code in start() that calls the button's setOnAction() method with this:

btn.setOnAction((e)->{label.setText("Hello, World!");});

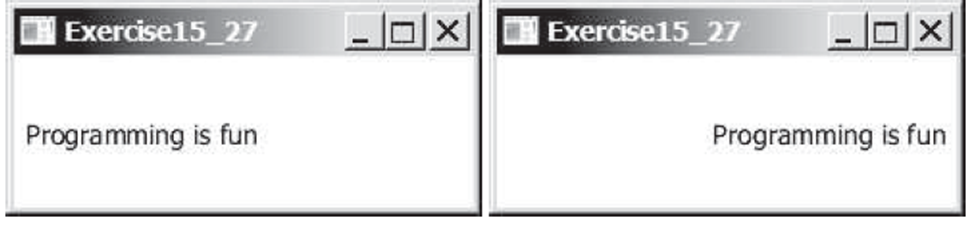
The ClickHandler class that you just eliminated had a handle() method in it. That method took a single parameter, which was an ActionEvent object, e. The code we just put in place of the original code contains a *lambda expression* mapping that parameter e to a chunk of code that will run when the event takes place. This code is the code that used to live in the body of the handle() method. The new statement saves a lot of space over the original. The lambda expression replaces the ClickHandler class and its handle() method with a simpler alternative.

**Practical questions:**

1. Write a program that can dynamically change the font of a text in a label displayed on a stack pane. The text can be displayed in bold and italic at the same time. You can select the font name or font size from combo boxes, as shown in Figure. The available font names can be obtained using Font.getFamilies(). The combo box for the font size is initialized with numbers from 1 to 100.



1. Write a program that displays a moving text, as shown in Figure. The text moves from left to right circularly. When it disappears in the right, it reappears from the left. The text freezes when the mouse is pressed and moves again when the button is released.



3. Create animation in Figure to meet the following requirements:

■ Allow the user to specify the animation speed in a text field.

■ Get the number of iamges and image’s file-name prefix from the user. For example, if the user enters n for the number of images and L for the image prefix, then the files are L1.gif, L2.gif, and so on, to Ln.gif. Assume that the images are stored in the image directory, a subdirectory of the program’s class directory. The animation displays the images one after the other.

■ Allow the user to specify an audio file URL. The audio is played while the animation runs.



**Observations:**

**1)Program:-**

import javafx.application.Application;

import javafx.collections.FXCollections;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.CheckBox;

import javafx.scene.control.ComboBox;

import javafx.scene.control.Label;

import javafx.scene.layout.HBox;

import javafx.scene.layout.StackPane;

import javafx.scene.layout.VBox;

import javafx.scene.text.Font;

import javafx.scene.text.FontPosture;

import javafx.scene.text.FontWeight;

import javafx.scene.text.Text;

import javafx.stage.Stage;

import java.util.\*;

public class P9\_1 extends Application {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t);

try {

launch(args);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

public void start(Stage stage) throws Exception {

Text tName = new Text("Font Name");

Text tSize = new Text("Font Size");

Label label = new Label("Programming is fun");

label.setFont(Font.font("Book Antiqua", FontWeight.NORMAL, 48));

ComboBox<String> fontNames = new ComboBox<>(FXCollections.observableArrayList(Font.getFamilies()));

fontNames.setValue("Book Antiqua");

fontNames.setOnAction(e -> label.setFont(Font.font(fontNames.getValue(), label.getFont().getSize())));

ComboBox<Integer> fontSize = new ComboBox<>(FXCollections.observableArrayList(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, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100));

fontSize.setValue(48);

fontSize.setOnAction(e -> label.setFont(Font.font(label.getFont().getName(), fontSize.getValue())));

CheckBox bold = new CheckBox("Bold");

CheckBox italic = new CheckBox("Italic");

bold.setOnAction(e -> {

FontWeight fontWeight = bold.isSelected() ? FontWeight.BOLD : FontWeight.NORMAL;

FontPosture fontPosture = italic.isSelected() ? FontPosture.ITALIC : FontPosture.REGULAR;

label.setFont(Font.font(label.getFont().getName(), fontWeight, fontPosture, label.getFont().getSize()));

});

italic.setOnAction(e -> {

FontWeight fontWeight = bold.isSelected() ? FontWeight.BOLD : FontWeight.NORMAL;

FontPosture fontPosture = italic.isSelected() ? FontPosture.ITALIC : FontPosture.REGULAR;

label.setFont(Font.font(label.getFont().getName(), fontWeight, fontPosture, label.getFont().getSize()));

});

HBox controls = new HBox(tName, fontNames, tSize, fontSize);

controls.setSpacing(10);

controls.setAlignment(Pos.TOP\_CENTER);

HBox checkboxes = new HBox(bold, italic);

checkboxes.setSpacing(10);

checkboxes.setAlignment(Pos.BOTTOM\_CENTER);

VBox vbox = new VBox(controls, label, checkboxes);

vbox.setAlignment(Pos.CENTER);

vbox.setSpacing(10);

Scene scene = new Scene(new StackPane(vbox), 500, 150);

stage.setScene(scene);

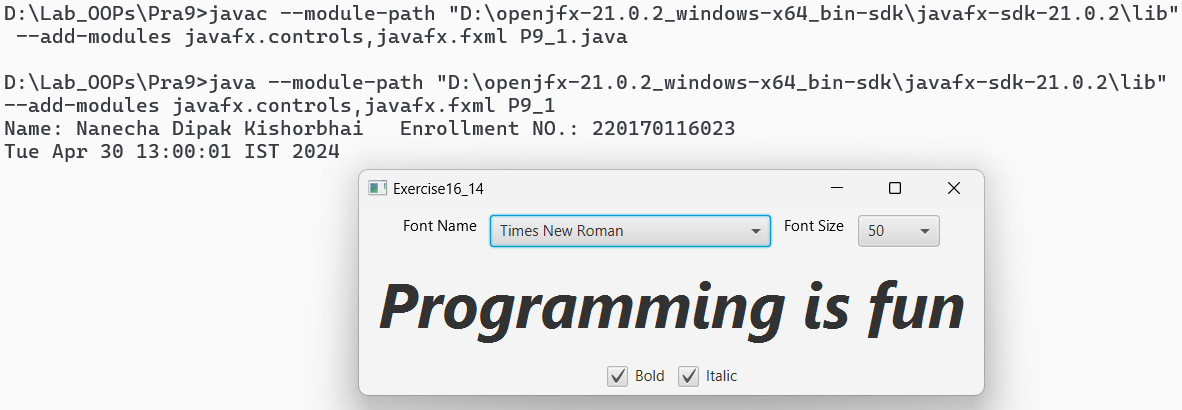
stage.setTitle("Exercise16\_14");

stage.show();

}

}

**Output:-**

****

**2)Program:-**

import javafx.animation.AnimationTimer;

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.input.MouseEvent;

import javafx.scene.layout.Pane;

import javafx.scene.text.Text;

import javafx.stage.Stage;

import java.util.\*;

public class P9\_2 extends Application {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t);

try {

launch(args);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

private boolean isPaused = false;

public void start(Stage stage) throws Exception {

Text movingText = new Text("Programming is fun");

movingText.setLayoutX(0);

movingText.setLayoutY(50); // height/2

Pane root = new Pane(movingText);

Scene scene = new Scene(root, 400, 100);

scene.addEventHandler(MouseEvent.MOUSE\_PRESSED, e -> isPaused = true);

scene.addEventHandler(MouseEvent.MOUSE\_RELEASED, e -> isPaused = false);

scene.heightProperty().addListener((obs, oldval, newVal) -> {

movingText.setLayoutY(newVal.doubleValue() / 2);

});

AnimationTimer timer = new AnimationTimer() {

public void handle(long now) {

if (!isPaused) {

movingText.setLayoutX((movingText.getLayoutX() + 2) % scene.getWidth());

}

}

};

timer.start();

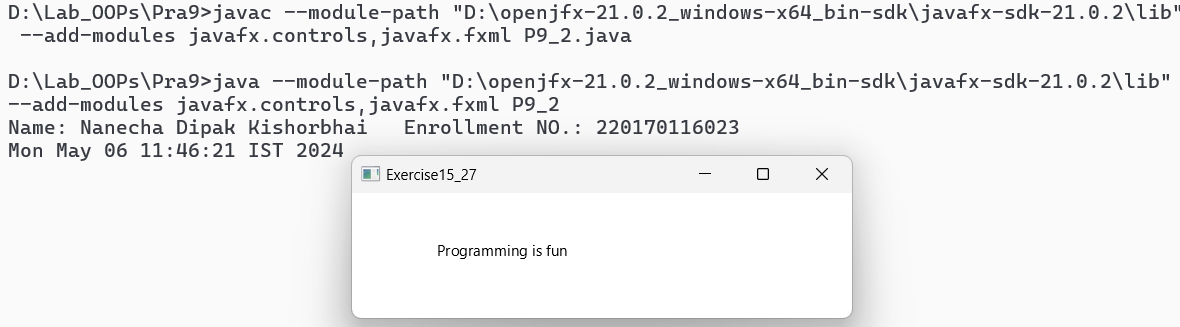
stage.setScene(scene);

stage.setTitle("Exercise15\_27");

stage.show();

}

}**Output:-**



**3)Program:-**

**Output:-**

**Conclusion:** Event handling allows programs to respond to user actions and external stimuli, creating a dynamic and responsive experience. Through event handling, developers can respond to user actions such as clicks, key presses, and mouse movements, enabling dynamic interactions with the application interface. Animation adds a dynamic layer, enhancing user engagement and providing visual feedback. Animation breathes life into your UI by adding visual effects and transitions. Animation techniques enables developers to convey information more effectively, guide user attention, and add visual appeal to their applications.

**Quiz:**

# How does event handling work in Java, and what is the event-driven programming model?

# Answer:- In Java, event handling is a mechanism that allows the programmer to respond to various events generated by the user or the system, such as mouse clicks, key presses, or actions on graphical user interface (GUI) components like buttons or menus. The event handling mechanism in Java follows a model known as the event-driven programming model. Event Sources: In Java, events are generated by event sources. These sources could be GUI components like buttons, text fields, or any other component that can interact with the user or the system. Event Object: An object that carries information about the event. This information can include the source of the event, the type of event and any additional details specific to the event. Event Listener: An object that listens for events from a specific source and defines how to respond to them. Event listeners implement interfaces that define methods for different event types. When an event occurs on a source object, it creates an event object containing relevant details. The source object then fires the event, notifying all registered listeners.

# What is the role of the java.awt.event and javafx.event packages in Java event handling?

# Answer:-

# java.awt.event: his package belongs to the older Abstract Window Toolkit (AWT) and the Swing GUI framework. This package provides classes and interfaces for handling various events generated by AWT components like buttons, text fields, menus, etc. Developers use these Package to create event listeners that respond to user actions such as mouse clicks, mouse movements, keyboard inputs, etc. javafx.event: This package is part of the modern JavaFX GUI framework. This package provides classes and interfaces for handling events in JavaFX applications. JavaFX event handling is typically done using lambda expressions or by implementing event handler interfaces.

# Explain: MouseEvent, KeyEvent, ActionEvent

# Answer:-

# MouseEvent: The MouseEvent class represents an event that occurs when the user interacts like clicking, dragging, hovering, or scrolling with the mouse. A MouseEvent object carries details like the type of event (click, press, release), location of the mouse pointer, and which mouse buttons were involved. KeyEvent: The KeyEvent class represents an event that occurs when the user interacts with the keyboard. A KeyEvent object provides information about the key that was pressed, its modifiers (like Ctrl, Shift), and whether it was pressed or released. ActionEvent: The ActionEvent class represents an event that occurs when a component (like a button) generates an action. An ActionEvent doesn't provide details about the specific user action that caused it. It's typically used with components that can perform actions, like buttons or menu items. Often, an ActionEvent might be generated in response to a MouseEvent (like a button click) or a KeyEvent (like pressing Enter on a button).

# What are the primary libraries or frameworks for creating animations in Java, and which one do you prefer?

# Answer:-

# 1) Swing: Swing is a GUI toolkit for Java that provides a set of lightweight components for building desktop applications. While Swing isn't specifically designed for animations, it provides basic building blocks like timers and custom painting mechanisms that can be used to create simple animations. Animations in Swing are typically achieved by repeatedly updating the state of Swing components within a loop or using timers to schedule updates. 2)JavaFX: JavaFX is a modern GUI framework with built-in animation features. It offers a declarative approach using CSS-like animation styles and a powerful Timeline class for managing animation sequences. I would choose JavaFX, because javaFX is often a popular choice due to its integration with the Java platform, rich set of features, and versatility for creating both simple and complex animations. It's supported by Oracle and has a vibrant community, making it a reliable option for many animation projects in Java.

# How to set the cycle count of an animation to infinite?

# Answer:-

# Using the INDEFINITE constant: JavaFX provides a static constant Timeline.INDEFINITE within the Animation class specifically for this purpose. We can set the cycle count of your animation using the setCycleCount method. Ex:

# AnimationObject.setCycleCount(Timeline.INDEFINITE);

# Setting a very high cycle count:

# While not strictly infinite, we can achieve a similar effect by setting the cycle count to a very high number.

# Ex:

# AnimationObject.setCycleCount(Integer.MAX\_VALUE);

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 10**

**AIM: To learn recursion and generics.**

**Date:**

**CO mapped: CO-4**

**Objectives:**

1. To develop a deep understanding of recursion and generics in programming. Mastery of recursion will enable the development of elegant and efficient algorithms for solving complex problems. Understanding generics will facilitate the creation of flexible, reusable, and type-safe code in various programming languages.
2. Learning recursion and generics is crucial for building efficient algorithms and writing more versatile and type-safe code in software development. Achieving this objective will help you become a more proficient and well-rounded programmer.

**Background:**

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called the recursive method.

Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time. Before generics, we can store any type of object in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of object.

**Practical questions:**

1. Write a recursive method that converts a decimal number into a binary number as a string. The method header is: public static String dec2Bin(int value)

Write a test program that prompts the user to enter a decimal number and displays its binary equivalent.

2. Write the following method that returns a new ArrayList. The new list contains the non-duplicate elements from the original list.

public static <E>ArrayList<E>removeDuplicates(ArrayList<E> list)

3. Implement the following method using binary search.

public static <E extends Comparable<E>>

intbinarySearch(E list, E key)

**Observations:**

**1)Program:-**

import java.util.\*;

public class P10\_1 {

public static String dec2Bin(int value) {

if (value == 0) return "0";

else if(value == 1) return "1";

String str = "";

str += dec2Bin(value / 2) + value % 2;

return str;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

System.out.print("Enter number: ");

int d = se.nextInt();

String str = dec2Bin(d);

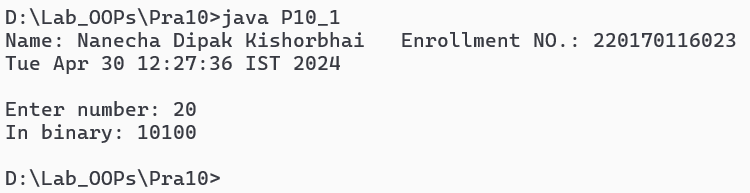
System.out.println("In binary: " + str);

se.close();

}

}

**Output:-**

****

**2)Program:-**

import java.util.\*;

public class P10\_2 {

public static <E> ArrayList<E> removeDuplicates(ArrayList<E> list) {

ArrayList<E> temp = new ArrayList<>();

for (E e : list) {

if (!temp.contains(e)) temp.add(e);

}

return temp;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

ArrayList<Integer> arr = new ArrayList<>();

System.out.println("Enter numbers in arraylist:");

System.out.println("Enter -1 for stop");

int num;

while (true) {

num = se.nextInt();

if(num == -1) break;

arr.add(num);

}

ArrayList<Integer> list = removeDuplicates(arr);

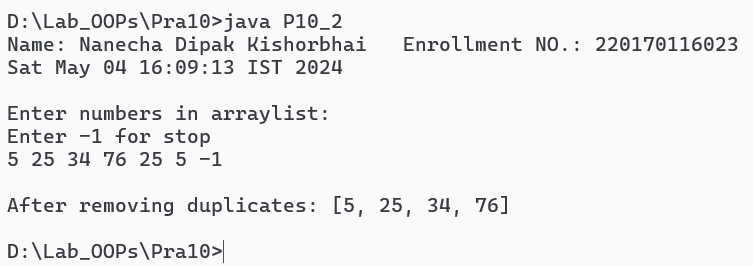
System.out.println("\nAfter removing duplicates: " + list);

se.close();

}

}

**Output:-**

****

**3)Program:-**

import java.util.\*;

public class P10\_3 {

public static <E extends Comparable<E>> int binarySearch(E[] list, E key) {

int lb = 0, ub = list.length - 1;

while (lb <= ub) {

int mid = (lb + ub) / 2;

int temp = list[mid].compareTo(key);

if (temp == 0) return mid;

else if (temp < 0) lb = mid + 1;

else if (temp > 0) ub = mid - 1;

}

return -1;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

Integer[] arr = { 5, 55, 33, 10, 2, 7, 88 };

Arrays.sort(arr);

for (Integer e : arr) {

System.out.print(e + " ");

}

System.out.print("\nEnter element which to find: ");

int key = se.nextInt();

int index = binarySearch(arr, key);

if (index != -1) System.out.println("\nElement " + key + " found at index: " + index);

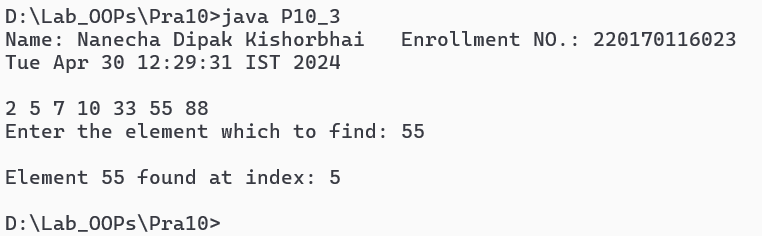
else System.out.println("\nElement not found");

se.close();

}

}

**Output:-**



**Conclusion:** Recursion in java is a process in which a method calls itself continuously. Recursion allows you to break down complex problems into smaller, self-similar ones, leading to concise and readable code. Each recursive call works on a smaller input until a base case is reached, which is a condition where the function returns without making any further recursive calls. Generics provide a means to create classes, interfaces, and methods that operate on objects of various types while providing compile-time type safety. They enable the creation of reusable, type-safe code components, promoting better code organization and reducing the likelihood of runtime errors.

**Quiz:**

# What is recursion in Java, and how does it differ from iteration in solving problems?

# Answer:- Recursion in Java is a programming technique where a method calls itself directly or indirectly. In recursion, a function solves a problem by breaking it down into smaller instances of the same problem. Each recursive call works on a smaller input until a base case is reached, which is a condition where the function returns without making any further recursive calls. Recursion utilizes the call stack to manage the state of each recursive call. On the other hand, iteration involves using loops (such as for, while, or do-while, for-each) to repeatedly execute a set of instructions until a specified condition is met. Iterative solutions usually involve maintaining state through variables and updating them within the loop until the desired result is achieved.

# What are the advantages and disadvantages of using recursion in Java?

# Answer:-

# Advantages: Recursive solutions can sometimes be simpler and more elegant than iterative solutions, especially for problems that inherently involve repeated subproblems or have a natural recursive structure. Like tree traversals or the factorial function, recursive code can be cleaner and easier to understand. Recursive functions can encapsulate complex logic into simpler, self-contained units and reusability. Recursive solutions often require fewer lines of code compared to their iterative counterparts, which can lead to more maintainable code. Disadvantages: Recursion is the risk of stack overflow errors, especially for problems with deep recursion or when dealing with large input sizes. Each recursive call consumes additional stack space. Debugging recursive code can be trickier compared to iterative code. Using recursion can lead to the risk of infinite recursion.

# What are generics in Java, and why are they used for creating parameterized types?

# Answer:- Generics in Java are a way to create parameterized types. Generics essentially allows to create classes, interfaces, and methods that can work with various data types without sacrificing type safety.

# Generics enable us to create classes, methods, and interfaces that can work with various data types. For instance, classes like HashSet, ArrayList, and HashMap utilize generics effectively. Generics enforce type safety at compile time. When using a generic class or method, we can provide the actual data type (like Integer, String) to fill in these placeholders, creating a parameterized type. This improves code maintainability and reduces redundancy.

# How to define Generic class? What are restrictions of generic programming?

# Answer:- Generic classes allow you to create blueprints for objects that can hold different types of data. In languages Java, generic classes are defined using type parameters enclosed in angle brackets (<>). Syntax:

# class MyClass<TypeParameter1, TypeParameter2, ...> {body of the class }

# Restrictions of Generic Programming:-

# We cannot directly use primitive types (int, char, etc.) as type parameters in generics. You can use their wrapper classes (e.g., Integer for int).

# Generic classes cannot have static fields whose type is a type parameter. Static fields need a fixed type at compile time.

# We cannot create, catch, or throw objects of parameterized types. Exceptions themselves are classes, and their type needs to be determined at compile time.

# Objects of parameterized types cannot be created, caught, or thrown as exception types require compile-time determination.

# Generic classes cannot have static fields whose type is a type parameter. Static fields require a fixed type at compile time.

# Can you provide an example of a generic class in Java, such as a generic ArrayList?

# Answer:-

# Example of a generic class:-

# public class GenericArrayList<T> {

# private T[] items;

# private int size;

# public GenericArrayList() {

# this.items = (T[]) new Object[10];

# this.size = 0;

# }

# public void add(T item) {

# if (size == items.length) { resize();}

# items[size++] = item;

# }

# private void resize() {

# T[] newItems = (T[]) new Object[items.length \* 2];

# System.arraycopy(items, 0, newItems, 0, items.length);

# this.items = newItems;

# }

# }

# This generic class GenericArrayList<T> takes a type parameter T. This allows the class to store elements of any data type T. The compiler ensures to only add elements of type T to the list, preventing type mismatches. Like:- GenericArrayList<String>: This would store a list of strings.

# 

# Suggested Reference:

# 1. <https://www.tutorialspoint.com/java/>

# 2. <https://www.geeksforgeeks.org/>

# 3. <https://www.w3schools.com/java/>

# 4. <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty

**Experiment No: 11**

**AIM: To demonstrate the use of Collection framework.**

**Date:**

**CO mapped: CO-4**

**Objectives:**

1. To proficiently demonstrate the use of Java's Collection framework, including understanding its core interfaces (List, Set, Map), implementing and manipulating data structures like lists, sets, and maps, and effectively applying collections for data storage, retrieval, and manipulation in Java applications.
2. Mastery of the Java Collection framework is essential for managing and organizing data efficiently in Java applications. This objective focuses on understanding the core collection interfaces and using them to build versatile data structures to meet various application needs.

**Background:**

The Collection in Java is a framework that provides architecture to store and manipulate a group of objects. Java Collections can achieve all the operations that you perform on data such as searching, sorting, insertion, manipulation, and deletion. Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

**Practical questions:**

1. Write a program that lets the user enter numbers from a graphical user interface and displays them in a text area, as shown in Figure. Use a linked list to store the numbers. Do not store duplicate numbers. Add the buttons Sort, Shuffle, and Reverse to sort, shuffle, and reverse the list.



2. Create two priority queues, {"George", "Jim", "John", "Blake", "Kevin", "Michael"} and {"George", "Katie", "Kevin", "Michelle", "Ryan"}, and find their union, difference, and intersection.

3. Store pairs of 10 states and its capital in a map. Your program should prompt the user to enter a state and should display the capital for the state.

**Observations:**

**1)Program:-**

import javafx.application.Application;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.scene.text.Text;

import javafx.stage.Stage;

import java.util.\*;

public class P11\_1 extends Application {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t);

try {

launch(args);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

public static String print(LinkedList<Integer> ll) {

String str = "";

for (int e : ll) {

str += e + " ";

}

return str;

}

public void start(Stage stage) throws Exception {

LinkedList<Integer> ll = new LinkedList<>();

Text t = new Text("Enter a number:");

TextField tf = new TextField();

TextArea ta = new TextArea("Empty");

ta.setEditable(false);

ta.setPrefSize(400, 50);

tf.setOnAction(e -> {

int num = Integer.parseInt(tf.getText());

if (ll.isEmpty()) {

ta.clear();

}

if (!ll.contains(num)) {

ll.add(num);

}

tf.clear();

ta.setText(print(ll));

});

Button sort = new Button("Sort");

sort.setOnAction(e -> {

Collections.sort(ll);

ta.setText(print(ll));

});

Button shuffle = new Button("Shuffle");

shuffle.setOnAction(e -> {

Collections.shuffle(ll);

ta.setText(print(ll));

});

Button reverse = new Button("Reverse");

reverse.setOnAction(e -> {

Collections.reverse(ll);

ta.setText(print(ll));

});

HBox hbox = new HBox(t, tf);

hbox.setSpacing(10);

hbox.setAlignment(Pos.CENTER);

HBox buttonBox = new HBox(sort, shuffle, reverse);

buttonBox.setSpacing(10);

buttonBox.setAlignment(Pos.CENTER);

VBox vbox = new VBox(hbox, ta, buttonBox);

vbox.setAlignment(Pos.CENTER);

Scene scene = new Scene(vbox);

stage.setTitle("Exercise20\_02");

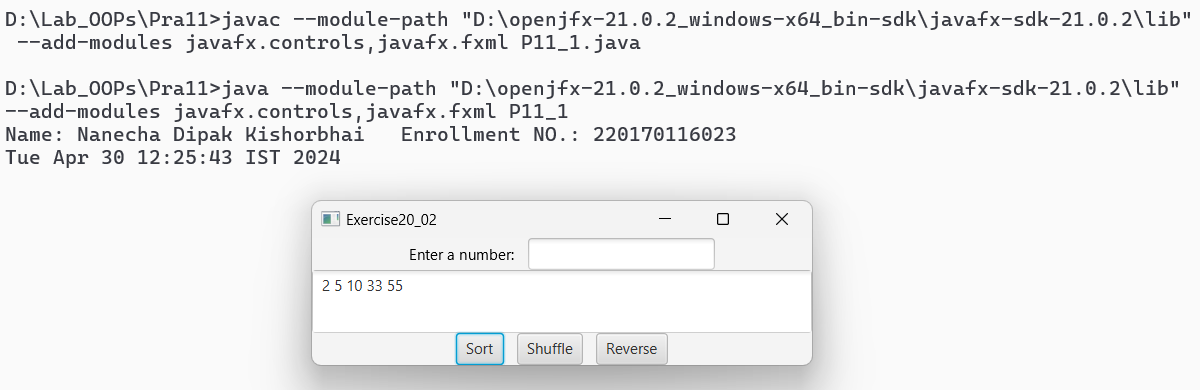
stage.setScene(scene);

stage.show();

}

}

**Output:-**



**2)Program:-**

import java.util.\*;

public class P11\_2 {

public static String print(PriorityQueue<String> queue) {

String str = "[";

PriorityQueue<String> temp = new PriorityQueue<>(queue);

while (!temp.isEmpty()) {

str += temp.poll() + ", ";

}

return str += queue.isEmpty()? "]" : "\b\b]";

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

String[] list1 = {"George", "Jim", "John", "Blake", "Kevin", "Michael"};

String[] list2 = {"George", "Katie", "Kevin", "Michelle", "Ryan"};

PriorityQueue<String> que1 = new PriorityQueue<>();

PriorityQueue<String> que2 = new PriorityQueue<>();

for (String str: list1)

que1.add(str);

for (String str : list2)

que2.add(str);

System.out.println("que1: " + print(que1) + "\nque2: " + print(que2));

// union

PriorityQueue<String> union = new PriorityQueue<>(que1);

for (String str : que2) {

if (!union.contains(str))

union.add(str);

}

System.out.println("\nUnion: " + print(union));

// Difference (que1 - que2)

PriorityQueue<String> difference1 = new PriorityQueue<>(que1);

difference1.removeAll(que2);

System.out.println("\nque1-que2: " + print(difference1));

// Difference (queue2 - queue1)

PriorityQueue<String> difference2 = new PriorityQueue<>(que2);

difference2.removeAll(que1);

System.out.println("\nque2-que1: " + print(difference2));

// Intersection

PriorityQueue<String> intersection = new PriorityQueue<>(que1);

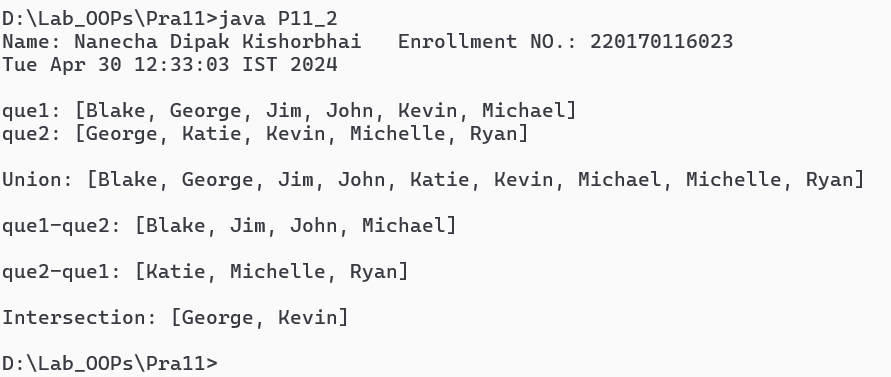
intersection.retainAll(que2);

System.out.println("\nIntersection: " + print(intersection));

}

}

**Output:-**

****

**3)Program:-**

import java.util.\*;

public class P11\_3 {

public static String toTitleCase(String s) {

if (Character.isLowerCase(s.toCharArray()[0]))

s = s.substring(0, 1).toUpperCase() + s.substring(1);

if (s.contains(" ")){

int index = s.indexOf(' ');

s = s.substring(0, index + 1) + toTitleCase(s.substring(index + 1));

}

return s;

}

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Scanner se = new Scanner(System.in);

HashMap<String, String> map = new HashMap<>();

map.put("Gujarat", "Gandhinagar");

map.put("Andhra Pradesh", "Amaravati");

map.put("Arunachal Pradesh", "Itanagar");

map.put("Assam", "Dispur");

map.put("Bihar", "Patna");

map.put("Chhattisgarh", "Raipur");

map.put("Goa", "Panaji");

map.put("Haryana", "Chandigarh");

map.put("Himachal Pradesh", "Shimla");

map.put("Jharkhand", "Ranchi");

System.out.print("Enter a state: ");

String state = toTitleCase(se.nextLine());

String capital = map.get(state);

if (capital != null)

System.out.println("The capital of " + state + " is " + capital);

else

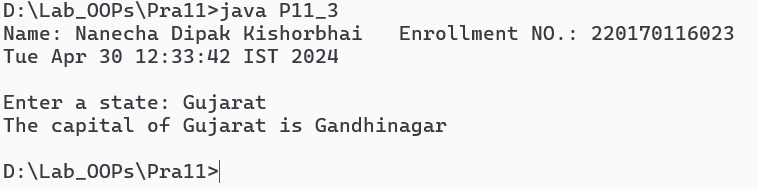
System.out.println("Capital not found for the " + state + " state.");

se.close();

}

}

**Output:-**

****

**Conclusion:** The Collection in Java is a framework that provides architecture to store and manipulate a group of objects. Java Collection means a single unit of objects. It provides a set of pre-defined data structures, like lists, sets, and maps, each with specific strengths for organizing and accessing data. The Collection framework in Java consists of several core interfaces, implementation classes, and algorithms to work with collections. The Java Collections Framework fosters code readability, reusability, and maintainability.

**Quiz:**

# Write a note on ‘Collection in JAVA’. Also discuss List and Enumeration Interface.

# Answer:- In Java, the Collection framework provides a set of classes and interfaces to represent groups of objects. It provides versatile data structures that can be used in various scenarios. The Collection framework in Java consists of several core interfaces, implementation classes, and algorithms to work with collections. Like: Collection, List, set, Map. List Interface:- The List interface extends the Collection interface and represents an ordered collection of elements.

# Implementations of the List interface include ArrayList, LinkedList, and Vector. Features:

# Allows duplicate elements,

# Maintains the order of elements as they are inserted,

# Provides methods to access elements by their index,

# Provides methods to add, remove, and modify elements.

# Enumeration Interface: The Enumeration interface in Java is an old interface used to iterate through a collection of objects. Implementations of the Enumeration Interface include Vector, Hashtable, Stack, and Properties.

# Features:

# Only iterate through elements in the order they were added.

# Can't go backward or access elements by index.

# Provides methods like hasMoreElements() and nextElement() to iterate through a collection.

# Differentiate between Enumeration and Iterator.

# Answer:-

# Enumeration: Supports only forward iteration. It provides methods like hasMoreElements() and nextElement() to move forward through the collection. We cannot modify the collection (add, remove, or change elements) while iterating using Enumeration. Enumeration originally designed to work with legacy collection types like Vector and Hashtable.

# Iterator: Supports both forward and backward iteration. It provides methods like hasNext() and next() for forward iteration, and hasPrevious() and previous() for backward iteration. Allows safe modification of the collection (add, remove, or change elements) while iterating using Iterator. Iterator works with all types of collections in the Java Collections Framework.

# Compare List, Set and Map interfaces. Also compare ArrayList, TreeSet and HashMap classes in java.

# Answer:-

# List Interface: Represents an ordered collection of elements allows duplicate elements, elements are stored in a sequence, and each element is associated with an index, common implementations include ArrayList, LinkedList, and Vector.

# Set Interface: Represents a collection of unique elements, does not allow duplicate elements, does not guarantee the order of elements, common implementations include HashSet, LinkedHashSet, and TreeSet. Map Interface: Represents a mapping between keys and values, contains key-value pairs, where each key is associated with exactly one value, does not allow duplicate keys, common implementations include HashMap, LinkedHashMap, and TreeMap. ArrayList Class: Implements the List interface, represents a dynamic array that can grow or shrink in size, provides fast random access to elements by index, allows duplicate elements. While average access is constant time (O(1)), adding or removing elements from the middle can sometimes involve shifting elements, leading to a worst-case time complexity of O(n). TreeSet Class: Implements the Set interface, represents a sorted set of elements stored in a tree structure (Red-Black tree), automatically sorts elements in natural order or using a specified comparator, does not allow duplicate elements, slower for insertion and removal compared to HashSet, but provides faster access to elements in sorted order. Access, insertion, and removal all have a time complexity of O(log n), which is generally efficient even for large datasets. HashMap Class: Implements the Map interface, represents a hash table-based implementation of the Map interface, provides fast insertion, deletion, and retrieval of key-value pairs, does not maintain the order of elements, allows one null key and multiple null values. Average time complexity of O(1) due to its use of hashing.

# Explain the unique features of Map interface.

# Answer:-

# Key-Value Pairs: Map stores data in key-value pairs. Each key acts like a unique identifier that points to a specific value. This allows for efficient retrieval of values based on their keys.

# Unique Keys: Map cannot contain duplicate keys. Each key can only map to a single value. When attempting to insert a duplicate key, the existing value associated with that key will be overwritten by the new value.

# Unordered: There's no guarantee of the order in which elements are stored in a Map. However, specific implementations like TreeMap may enforce a particular order.

# Collection Views: The Map interface provides methods to access its contents in different ways as follows,

# keySet() returns a Set of all the keys in the Map.

# values() returns a Collection of all the values in the Map.

# entrySet() returns a Set of all the key-value pair mappings in the Map .

# How do you perform common operations like sorting, searching, or filtering on Collections?

# Answer:-

# Sorting: To sort a collection of objects, you can use the Collections.sort() method for lists or arrays. This method sorts the elements of the collection in ascending order based on their natural ordering or a specified Comparator if provided.

# Searching: For searching elements in a collection, we can check if a specific element exists in a collection using methods like contains() (returns true/false).

# Filtering: To filter elements based on certain criteria, we can create a stream from the collection and then use methods like filter() to apply the predicate that defines the filtering criteria.

# Adding elements: Elements can be added to collections using methods like add() for lists, put() for maps, or specific methods like addFirst(), addLast() for queues.

# Removing elements: Elements can be removed from collections using methods like remove() for lists, remove() for sets, remove(Object key) for maps, or specific methods like removeFirst(), removeLast() for queues.

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty:

**Experiment No: 12**

**AIM: To demonstrate the use of multithreading.**

**Date:**

**CO mapped: CO-3**

**Objectives:**

1. To effectively demonstrate the use of multithreading in software applications, including creating and managing multiple threads, synchronizing their execution, and leveraging the power of concurrent programming to improve performance, responsiveness, and resource utilization.
2. Demonstrating the use of multithreading is crucial for building responsive and efficient software applications, and this objective emphasizes understanding the concepts and practical implementation of multithreading to achieve these goals.

**Background:**

Multithreading in Java is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory areas so saves memory, and context-switching between the threads takes less time than the process. Java Multithreading is mostly used in games, animation, etc.

**Practical questions:**

1. Write a program to create a thread extending Thread class and demonstrate the use of slip() method.

2. Write a program to create a thread implementing Runnable interface and demonstrate the use of join() method.

3. Write a program that launches 10 threads. Each thread adds 1 to a variable sum that initially is 0. Define an Integer wrapper object to hold sum. Run the program with and without synchronization to see its effect.

**Observations:**

**1)Program:-**

import java.util.\*;

class ThreadClass extends Thread {

ThreadClass(String name) {

super(name);

}

public void run() {

System.out.println(getName() + " is started");

for (int i = 0; i < 3; i++) {

System.out.println(getName() + " working: " + (i + 1));

try {

sleep(1000);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

System.out.println(getName() + " is finished");

}

}

public class P12\_1 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

ThreadClass th1 = new ThreadClass("Thread\_1");

ThreadClass th2 = new ThreadClass("Thread\_2");

th1.run();

th2.run();

try {

th1.join();

th2.join();

} catch (Exception e) {

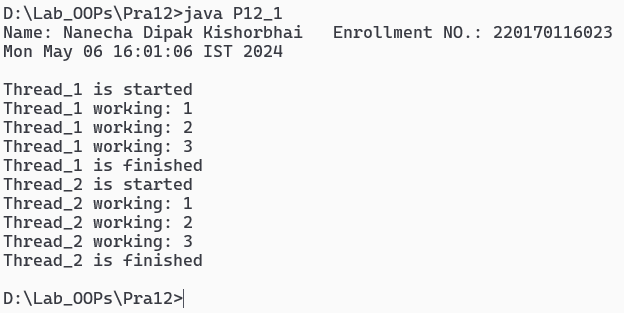
System.out.println(e.getMessage());

}

}

}

**Output:-**



**2)Program:-**

import java.util.\*;

class ThreadRunnable implements Runnable {

private String name;

private long delay;

ThreadRunnable(String name, long delay) {

this.name = name;

this.delay = delay;

}

public void run() {

System.out.println(name + " is started");

System.out.println(name + " will sleep for " + delay + " seconds");

try {

Thread.sleep((delay \* 1000)); // millisecond\*1000

} catch (Exception e) {

System.out.println(e.getMessage());

}

System.out.println(name + " is finished");

}

}

public class P12\_2 {

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Thread th1 = new Thread(new ThreadRunnable("Th1", 5));

Thread th2 = new Thread(new ThreadRunnable("Th2", 2));

th1.start();

th2.start();

try {

th1.join();

th2.join();

} catch (Exception e) {

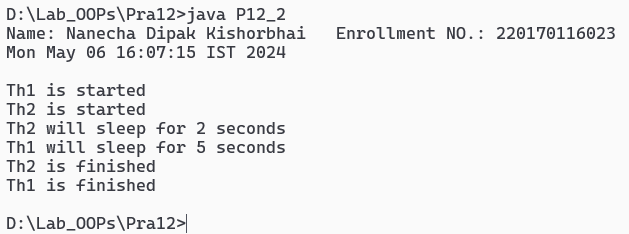
System.out.println(e.getMessage());

}

}

}

**Output:-**



**3)Program:-**

import java.util.\*;

public class P12\_3 {

private static Integer sum = 0;

private static Integer synSum = 0;

private static Object lock = new Object();

public static void main(String[] args) {

System.out.println("Name: Nanecha Dipak Kishorbhai\t Enrollment NO.: 220170116023");

Date t = new Date();

System.out.println(t + "\n");

Thread[] threads = new Thread[10];

for (int i = 0; i < threads.length; i++) {

threads[i] = new Thread(new Runnable() {

public void run() {

for (int i = 0; i < 100; i++) {

sum++;

}

}

});

threads[i].start();

}

for (int i = 0; i < threads.length; i++) {

try {

threads[i].join();

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

System.out.println("without synchronization: " + sum);

for (int i = 0; i < threads.length; i++) {

threads[i] = new Thread(new Runnable() {

public void run() {

for (int i = 0; i < 100; i++) {

synchronized (lock) {

synSum++;

}

}

}

});

threads[i].start();

}

for (int i = 0; i < threads.length; i++) {

try {

threads[i].join();

} catch (Exception e) {

System.out.println(e.getMessage());

}

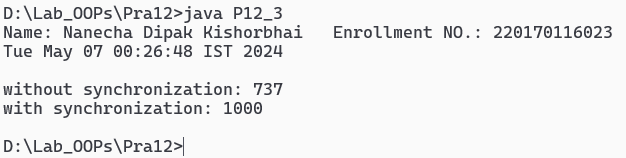
}

System.out.println("with synchronization: " + synSum);

}

}

**Output:-**



**Conclusion:** Multithreading in Java is a process of executing multiple threads simultaneously. A thread is a lightweight sub-process, the smallest unit of processing. Multithreading introduces complexities like synchronization and potential race conditions. Its utilization of a shared memory area by threads. They don't allocate separate memory areas so saves memory, and context-switching between the threads takes less time than the process. Java Multithreading is mostly used in games, animation, etc.

**Quiz:**

# Can you explain the difference between a process and a thread in the context of multithreading?

# Answer:-

# Process: A process can be thought of as an instance of a program that is being executed. Process has its own memory space, resources (like CPU, files), and execution context. Processes are managed by the operating system and can contain one or more threads. Creating and destroying processes is relatively expensive for the operating system.

# Thread: A thread is the smallest unit of execution within a process. Threads share the process's memory space and resources. Each thread has its own execution context, including its own stack, program counter, and register values.

# What are the different states in the lifecycle of a Java thread, and how does a thread transition between them?

# Answer:- A Java thread goes through various stages during its execution, known as its lifecycle.

# New: This is the initial state when a thread object is created using the Thread class constructor but haven't called the start() method yet.

# Runnable: When we call the start() method on the thread object, it transitions to the runnable state. This means the thread is ready to execute, but the CPU is currently executing another thread.

# Running: This is the active state where the thread is actually executing its tasks defined in the run() method.

# Blocked (Waiting): A thread enters this state when it is waiting for a resource that is currently held by another thread, or when it is waiting for a certain condition to be met.

# Timed Waiting: Similar to blocked state, but the thread waits for a specific amount of time. Methods such as sleep(millis) or join(timeout).

# Terminated (or Dead): This is the final state where the thread has finished executing its tasks or encountered an error that prevents it from continuing. The thread object is no longer usable.

# What is runnable interface? How can you use this interface in creating thread?

# Answer:- The Runnable interface in Java is a fundamental building block for multithreading. It defines a contract for any class whose instances can be executed by a thread.

# How to Use Runnable Interface for Thread Creation:

# Implement the Runnable interface: Create a class that implements the Runnable interface. Override the run() method in class which, represents the task that the thread will execute.

# Create a Thread object: Use the Thread class constructor and pass an instance of Runnable class that implements the run() method.

# Start the thread: Call the start() method on the Thread object to start the execution of the thread.

# Explain the concept of thread synchronization and the role of the synchronized keyword.

# Answer:- Thread synchronization is a mechanism used in multithreaded programming to control access to shared resources by multiple threads.It ensures that only one thread can access and modify a shared resource at a time, preventing data inconsistencies and race conditions. The synchronized keyword can be used in two ways: 1) Synchronized methods: Declaring a method as synchronized means that only one thread can execute that method on an object at a time. When a thread enters a synchronized method, it automatically acquires a lock on the object. This lock prevents other threads from entering the method until the first thread finishes and releases the lock.

# 2) Synchronized blocks: The synchronized keyword can be used to create synchronized blocks within a method. Synchronized block uses a lock on an object. Only one thread can enter the block at a time, while others trying to enter are blocked until the first thread exits.

# Explain: wait, sleep, notify and notify all.

# Answer:-

# wait: This method causes the currently executing thread to suspend its execution and relinquish the lock on the object it's synchronized on. The thread remains suspended until another thread calls notify or notifyAll on the same object. wait can only be called from within a synchronized block or method.

# sleep: sleep makes the current thread pause its execution for a specified time (in milliseconds) without releasing the lock on the object. This can be useful for introducing delays or pacing execution.

# notify: The notify() method is used to awaken a single thread that is waiting on the same object. When notify() is called, it selects one of the threads that are currently waiting on the object and signals it to continue execution. The awakened thread must reacquire the lock on the object before it can proceed.

# notifyAll: notifyAll wakes up all threads waiting on the same object. Similar to notify, the notifying thread retains the lock on the object after the call. Again, the order in which threads are awakened is not guaranteed.

# Suggested Reference:

# <https://www.tutorialspoint.com/java/>

# <https://www.geeksforgeeks.org/>

# <https://www.w3schools.com/java/>

# <https://www.javatpoint.com/>

# References used by the students: (Sufficient space to be provided)

# Rubric wise marks obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Criteria** | **Need Improvement** | **Good** | **Excellent** | **Total** |
| **Marks** | **Design of logic (4)**  **Correct output (4)**  **Mock viva test (2)** | Program has significant logic errors. (1)  Output has multiple errors. (1)  Delayed & only few correct answers (1) | Program has slight logic errors that do no significantly affect the results (2)  Output has minor errors. (2)  Partially correct response (1) | Program is logically well designed (3)  Program displays correct output with no errors (3)  All questions responded Correctly (2) |  |

# Signature of Faculty: