**Laboratory Manual for**

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

**B.E. Semester 4th**

**(Computer Engineering)**



**Vishwakarma Government Engineering College, Chandkheda**

**Gujarat**



**Directorate of Technical Education,**

**Gandhinagar, Gujarat**

**Vishwakarma Government Engineering College, Chandkheda**

**Certificate**

***This is to certify that Mr./Ms. \_\_\_\_\_\_\_\_Thakar Atri Kamleshkumar\_\_\_\_\_\_\_\_\_ Enrollment No. 220170107141 of B.E. Semester 4, Computer Engineering of this Institute (GTU Code: 07) has satisfactorily completed the Practical work for the subject Object Oriented Programming 1-(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.

**DTE’s Vision**

* To provide globally competitive technical education
* Remove geographical imbalances and inconsistencies
* Develop student friendly resources with a special focus on girls’ education and support to

weaker sections

* Develop programs relevant to industry and create a vibrant pool of technical professionals

**Institute’s Vision**

* To create an ecosystem for proliferation of socially responsible and technically sound engineers, innovators and entrepreneurs.

**Institute’s Mission**

* To develop state-of-the-art laboratories and well-equipped academic infrastructure.
* To motivate faculty and staff for qualification up-gradation, and enhancement of subject knowledge.
* To promote research, innovation and real-life problem-solving skills.
* To strengthen linkages with industries, academic and research organizations.
* To reinforce concern for sustainability, natural resource conservation and social responsibility.

**Department’s Vision**

* To create an environment for providing value-based education in Computer Engineering through innovation, team work and ethical practices.

**Department’s Mission**

* To produce computer engineering graduates according to the needs of industry, government, society and scientific community.
* To develop state of the art computing facilities and academic infrastructure.
* To develop partnership with industries, government agencies and R & D organizations for knowledge sharing and overall development of faculties and students.
* To solve industrial, governance and societal issues by applying computing techniques.
* To create environment for research and entrepreneurship.

**Program Outcomes (POs)**

* 1. **Engineering knowledge:** Apply the knowledge of mathematics, science,engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
  2. **Problem analysis:** Identify, formulate, review research literature, and analyzecomplex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
  3. **Design/development of solutions:** Design solutions for complex engineeringproblems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
  4. **Conduct investigations of complex problems:** Use research-based knowledgeand research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
  5. **Modern tool usage:** Create, select, and apply appropriate techniques,resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
  6. **The engineer and society:** Apply reasoning informed by the contextualknowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
  7. **Environment and sustainability:** Understand the impact of the professionalengineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
  8. **Ethics:** Apply ethical principles and commit to professional ethics andresponsibilities and norms of the engineering practice.
  9. **Individual and team work:** Function effectively as an individual, and as amember or leader in diverse teams, and in multidisciplinary settings.
  10. **Communication:** Communicate effectively on complex engineering activitieswith the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
  11. **Project management and finance:** Demonstrate knowledge and understandingof the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
  12. **Life-long learning:** Recognize the need for, and have the preparation and abilityto engage in independent and life-long learning in the broadest context of technological change.

**Program Specific Outcomes (PSOs)**

* Sound knowledge of fundamentals of computer science and engineering including software and hardware.
* Develop the software using sound software engineering principles having web based/mobile based interface.
* Use various tools and technology supporting modern software frameworks for solving problems having large volume of data in the domain of data science and machine learning.

**Program Educational Objectives (PEOs)**

* Possess technical competence in solving real life problems related to Computing.
* Acquire good analysis, design, development, implementation and testing skills to formulate simple computing solutions to the business and societal needs.
* Provide requisite skills to pursue entrepreneurship, higher studies, research, and development and imbibe high degree of professionalism in the fields of computing.
* Embrace life-long learning and remain continuously employable.
* Work and excel in a highly competence supportive, multicultural and professional environment which abiding to the legal and ethical responsibilities.

**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 exception handling in JAVA application and Multithreading. |  |  |  |  |  |  |
| 7. | To demonstrate I/O from files. |  |  |  |  |  |  |
| 8. | To learn JAVA FX UI Controls and To implement event handling and animation. |  |  |  |  |  |  |
| 9. | To learn recursion and generics. |  |  |  |  |  |  |
| 10. | To demonstrate the use of Collection framework. |  |  |  |  |  |  |
| 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:

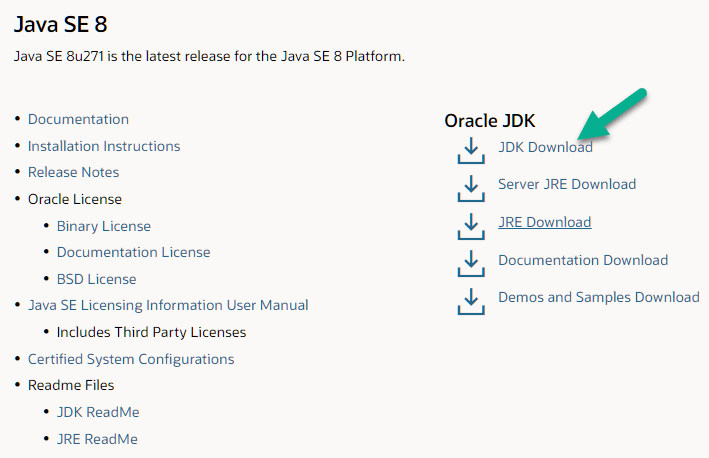
Eclipse: a Java-based open source platform that enables the creation of highly customized IDEs from plug-in components built by Eclipse members. The platform is user-friendly for beginners and also suitable for the creation of more sophisticated applications. Eclipse includes a lot of plug-ins that allow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

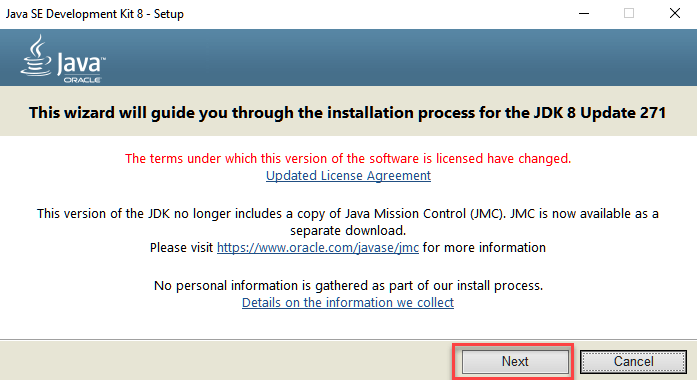
1. Accept License Agreement
2. Download Java 8 JDK for your version 32 bit or JDK download 64 bit.



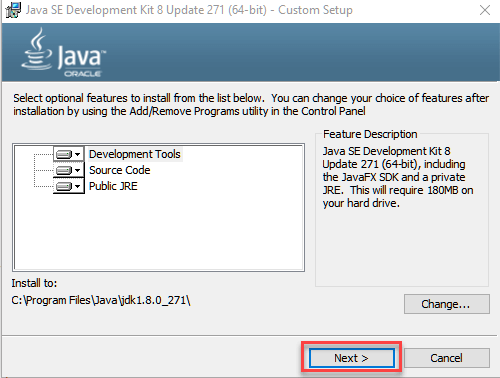
**Step 3)** When you click on the Installation link the popup will be open. Click on I reviewed and accept the Oracle Technology Network License Agreement for Oracle Java SE development kit and you will be redirected to the login page. If you don’t have an oracle account you can easily sign up by adding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



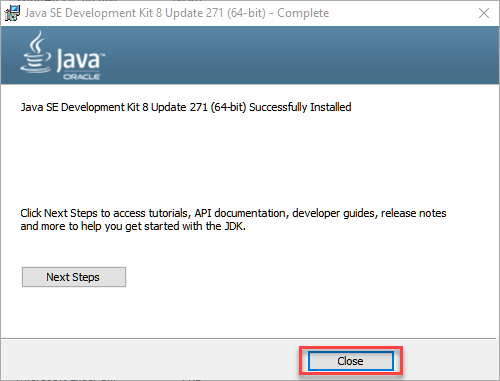
**Step 4)** Once the Java JDK 8 download is complete, run the exe for install JDK. Click Next



**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



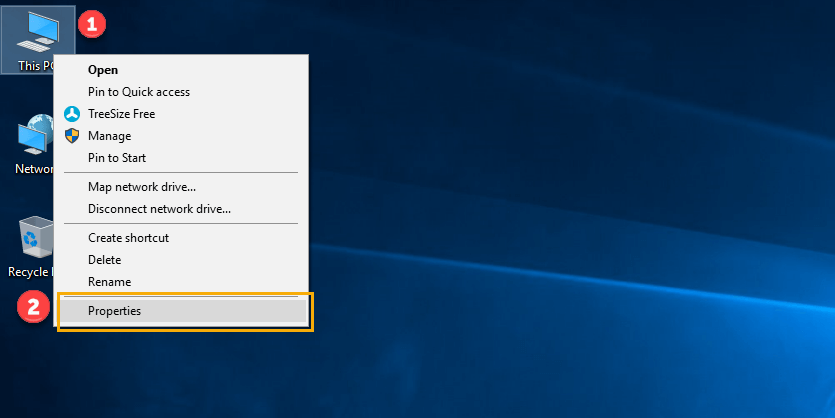
How to set Environment Variables in Java: Path and Classpath:

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**

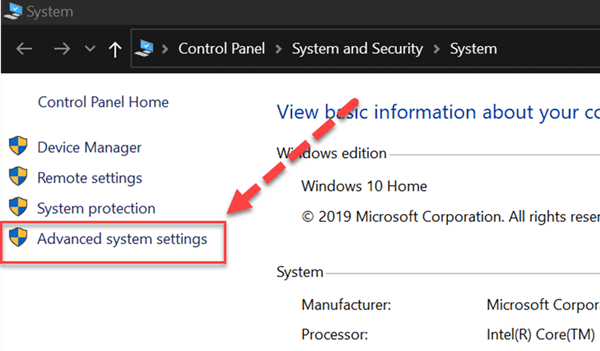
The CLASSPATH variable gives location of the Library Files.

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

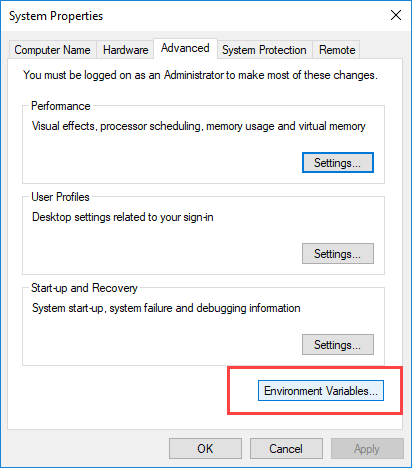
**Step 1)** Right Click on the My Computer and Select the properties



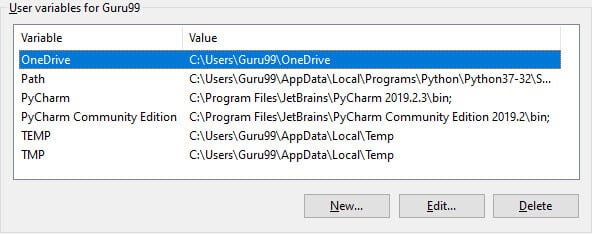
**Step 2)** Click on advanced system settings



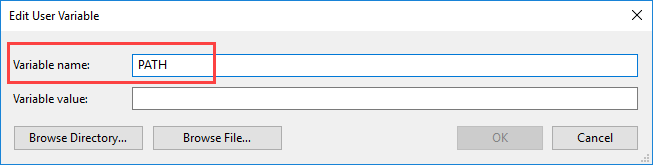
**Step 3)** Click on Environment Variables to set Java runtime environment



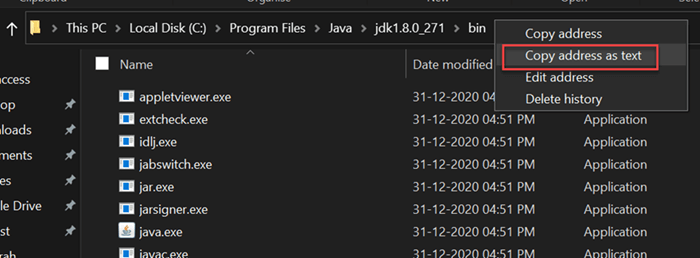
**Step 4)** Click on new Button of User variables



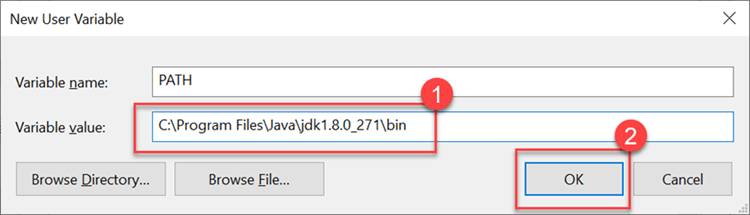
**Step 5)** Type PATH in the Variable name.



**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.

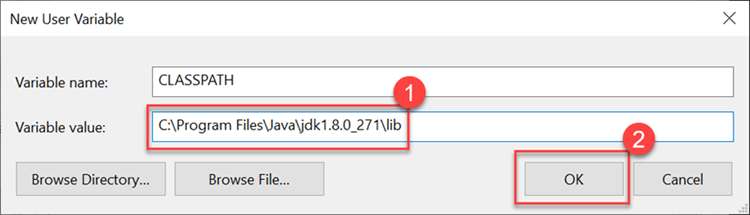


**Note:** In case you already have a PATH variable created in your PC, edit the PATH variable to

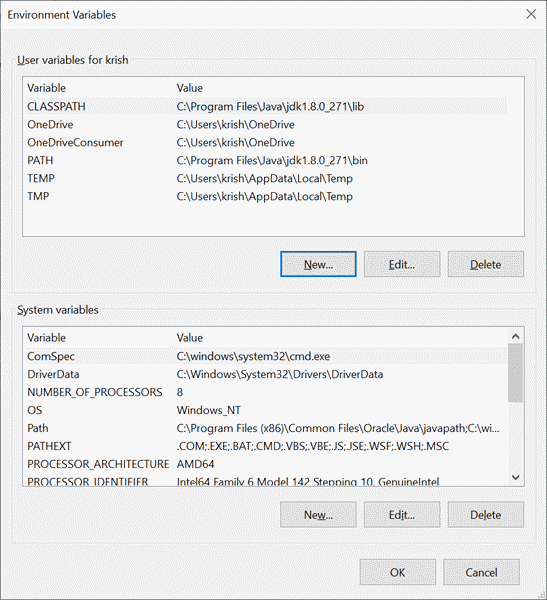
PATH = <JDK installation directory>\bin;%PATH%;

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

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

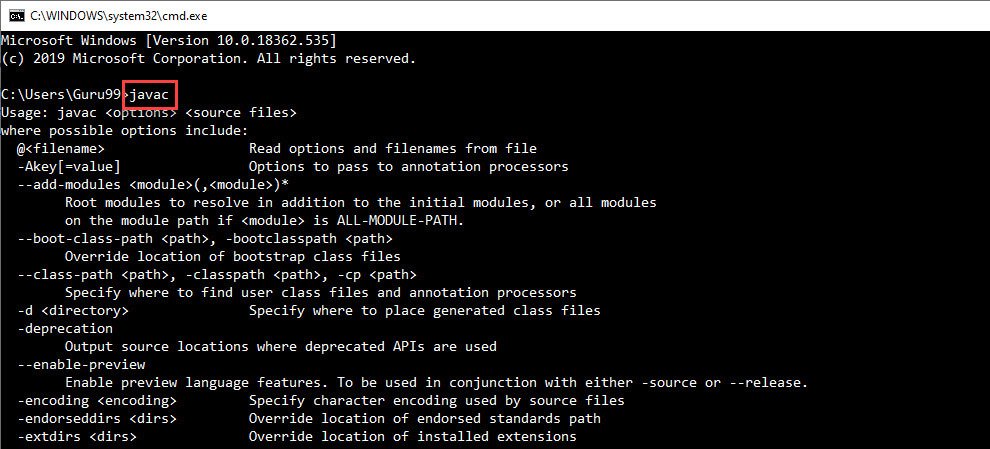


**Step 9)** Click on OK button



**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 | **\***  **\*\*\***  **\*\*\*\*\***  **\*\*\*\*\*\*\***  **\*\*\*\*\***  **\*\*\***  **\*** |

**Procedure:**

**Program 2:**

public class program\_2 {

    public static void main(String[] args) {

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

    }

}

**Program 3:**

public class program\_3 {

    public static void main(String[] args) {

        double a = 3.4, b = 50.2, e = 44.5, c = 2.1, d = .55, f = 5.9;

        double x = (e\*d-b\*f)/(a\*d-b\*c), y =(a\*f-e\*c)/(a\*d-b\*c);

        System.out.printf("x = %f\ny = %f",x,y);

    }

}

**Program 4:**

public class program\_4 {

    public static void main(String[] args) {

        int metres = 76;

        double feet = 3.28084 \* metres;

        System.out.printf("%d metres = %f feet", metres, feet);

    }

}

**Program 5:**

import java.util.Scanner;

public class program\_5 {

    public static void main(String[] args) {

        float weight, height\_in\_metres, bmi;

        Scanner sc = new Scanner(System.in);

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

        weight = sc.nextFloat();

        System.out.printf("Enter your height in metres: ");

        height\_in\_metres = sc.nextFloat();

        bmi = weight/(height\_in\_metres\*height\_in\_metres);

        System.out.printf("Your BMI is %f", bmi);

    }

}

**Program 6:**

import java.util.Scanner;

public class program\_6 {

    public static void main(String[] args) {

        int a, b, c;

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter first number: ");

        a = sc.nextInt();

        System.out.printf("Enter second number: ");

        b = sc.nextInt();

        System.out.printf("Enter third number: ");

        c = sc.nextInt();

        if (a >= b && a >= c) {

            if (b >= c) {

                System.out.printf("%d %d %d", a, b, c);

            } else {

                System.out.printf("%d %d %d", a, c, b);

            }

        } else if (b >= a && b >= c) {

            if (a >= c) {

                System.out.printf("%d %d %d", b, a, c);

            } else {

                System.out.printf("%d %d %d", b, c, a);

            }

        } else {

            if (a >= b) {

                System.out.printf("%d %d %d", c, a, b);

            } else {

                System.out.printf("%d %d %d", c, b, a);

            }

        }

        sc.close();

    }

}

**Program 7:**

import java.util.Scanner;

public class program\_7 {

    public static void main(String[] args) {

        char ch;

        Scanner sc = new Scanner(System.in);

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

        ch = sc.next().toLowerCase().charAt(0);

        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

            System.out.println("The given character is a vowel");

        } else {

            System.out.println("The given character is a consonant");

        }

        sc.close();

    }

}

**Program 8:**

import java.util.Scanner;

public class program\_8 {

    public static void main(String[] args) {

        int amount;

        int note\_1 = 0, note\_2 = 0, note\_5 = 0, note\_10 = 0, note\_50 = 0, note\_100 = 0;

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter the amount to be withdrawn: ");

        amount = sc.nextInt();

        while (amount != 0) {

            if (amount >= 100) {

                note\_100++;

                amount -= 100;

            } else if (amount >= 50) {

                note\_50++;

                amount -= 50;

            } else if (amount >= 10) {

                note\_10++;

                amount -= 10;

            } else if (amount >= 5) {

                note\_5++;

                amount -= 5;

            } else if (amount >= 2) {

                note\_2++;

                amount -= 2;

            } else {

                note\_1++;

                amount--;

            }

        }

        System.out.printf(

                "To withdraw the given amount you will require:\n%d notes of denomination 100\n%d notes of denomination 50\n%d notes of denomination 10\n%d notes of denomination 5\n%d notes of denomination 2\n%d notes of denomination 1\n",

                note\_100, note\_50, note\_10, note\_5, note\_2, note\_1);

        sc.close();

    }

}

**Program 9:**

import java.util.Scanner;

public class program\_9 {

    public static void main(String[] args) {

        int a;

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter a five digit number: ");

        a = sc.nextInt();

        System.out.println("The answer is " + (a+11111));

        sc.close();

    }

}

**Program 10:**

import java.util.Scanner;

import java.lang.Math;

public class program\_10 {

    public static void main(String[] args) {

        int a, b, c;

        double s, area;

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter the lengh of first side: ");

        a = sc.nextInt();

        System.out.printf("Enter the lengh of second side: ");

        b = sc.nextInt();

        System.out.printf("Enter the lengh of third side: ");

        c = sc.nextInt();

        s = (a + b + c) / 2.0;

        System.out.println(s);

        area = Math.sqrt(s \* (s - a) \* (s - b) \* (s - c));

        System.out.println("The area of the triangle is " + area + " sq units.");

        sc.close();

    }

}

**Program 11:**

public class program\_11 {

    public static void main(String[] args) {

        for (int i = 0; i < 4; i++) {

            for (int j = 0; j < 4 - i; j++) {

                System.out.printf("\* ");

            }

            System.out.println();

        }

        System.out.println();

        for (int i = 0; i < 4; i++) {

            for (int j = 0; j < 4 - i; j++) {

                System.out.printf("%d ", j + 1);

            }

            System.out.println();

        }

        System.out.println();

        int counter = 0;

        for (int i = 0; i < 4; i++) {

            for (int j = 0; j < 4 - i; j++) {

                counter++;

                if (counter == 10) {

                    System.out.printf("0");

                } else {

                    System.out.printf("%d ", counter);

                }

            }

            System.out.println();

        }

        System.out.println();

        int n = 4;

        for (int i = 1; i <= n; i++) {

            for (int j = 1; j <= n - i; j++) {

                System.out.print(" ");

            }

            for (int k = 1; k <= 2 \* i - 1; k++) {

                System.out.print("\*");

            }

            System.out.println();

        }

        for (int i = n - 1; i >= 1; i--) {

            for (int j = 1; j <= n - i; j++) {

                System.out.print(" ");

            }

            for (int k = 1; k <= 2 \* i - 1; k++) {

                System.out.print("\*");

            }

            System.out.println();

        }

    }

}

**Observations:**

**Program 2:**

****

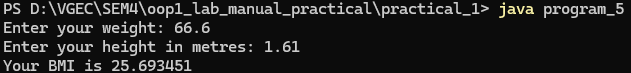
**Program 3:**

****

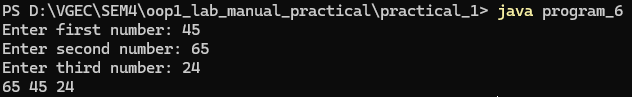
**Program 4:**

****

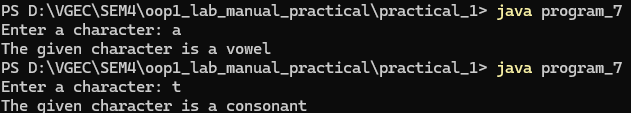
**Program 5:**

****

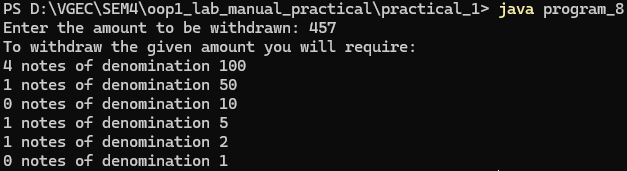
**Program 6:**

****

**Program 7:**

****

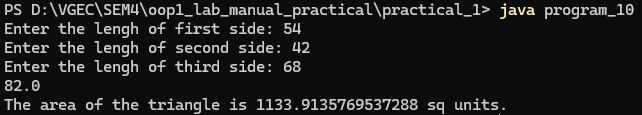
**Program 8:**

****

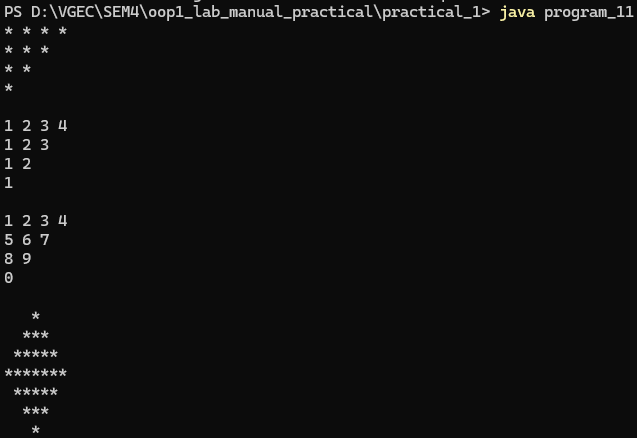
**Program 9:**

****

**Program 10:**

****

**Program 11:**

****

**Conclusion:** In this experiment we learned about basic concepts of java such as print statements, taking user inputs, processing inputs, control statements and loops to generate the desired output. We also learned some other concepts such as JVM, JRE, etc., how to install JDK and some basic things like purpose of java, platform independency, etc.

**Quiz:** (Sufficient space to be provided for the answers)

# What is the primary purpose of Java??

# Ans. The main goal of Java is to provide a platform-independent programming language and a runtime environment for developing and deploying software applications. Java was developed with the goal of "write once, run anywhere" (WORA), which means that Java code can be compiled into bytecode that can be used on any machine or platform equipped with a Java Virtual Machine (JVM). in, reassembled without want . This makes Java particularly suitable for developing web applications, enterprise software, mobile applications (Android), and other cross-platform applications. Additionally, Java emphasizes features such as simplicity, system orientation, and robustness, making it popular for a wide range of software development projects

# What is the main method in Java used for?

# Ans. In Java, a special method is a special method that acts as an entry point to a Java application. When you execute a Java program, the Java Virtual Machine (JVM) starts the execution of the program by finding a special method that contains the following signatures.

# public static void main(String[] args)

# Here is a breakdown of each section of the special rules –

# public: Indicates that the main method is accessible outside the class

# static: Indicates that the main method belongs to the class rather than any specific instance of the class.

# void: Indicates that the main method returns no value.

# Main: is the name of the method. This is the standard name that the JVM accepts as an entry point.

# String[] args: This represents a set of strings that can be used to pass arguments to a Java function at the command line. The main method usually contains the code that starts the program, sets the necessary resources, and starts the execution flow of the application. You can call other methods from the main method, instantiate objects, and perform the various tasks required to run the program.

# How Java Language is Platform Independent?

# Ans. Java achieves platform independence through several key mechanisms:

# Bytecode: When you compile Java source files, they are converted to bytecode instead of native machine code. Bytecode is a highly customized set of instructions designed to run with the Java Virtual Machine (JVM).

# Java Virtual Machine (JVM): Each platform has its own JVM implementation. The JVM looks down at the bytecode and executes on the underlying hardware. The JVM eliminates hardware and operating system specificity, and provides a consistent runtime environment for Java programs.

# Just-In-Time (JIT) compilation: Many modern JVM implementations include a Just-In-Time compiler. This compiler translates the bytecode into native machine code at runtime, optimizing the specific hardware platform on which the program is running.

# Class Libraries: Java provides a comprehensive set of libraries (Java API) that provide cross-platform functionality for communication, I/O operations, GUI development, etc. These libraries protect the developer from platform differences of the bottom .

# Write Once, Run Anywhere (WORA): Since Java code is compiled into bytecode, which can run on any JVM, Java programs can be executed on one platform and run on another platform each on which a corresponding JVM is installed. This principle is often summarized as "write once, run anywhere", emphasizing the independence of the Java platform. Overall, these features combine to make Java a platform-independent language, allowing developers to write code that can run on a variety of hardware operating systems without modification

# What is JVM and JRE?

# Ans. The JVM (Java Virtual Machine) and JRE (Java Runtime Environment) are key components of the Java platform:

# 1. JVM (Java Machine Operator):

# - A JVM is an abstract computing device that provides a runtime environment in which Java bytecode can be executed.

# - Abstracts the underlying hardware and operating system, and provides a consistent execution environment for Java programs.

# - JVM implementation is platform specific, which means that there are different implementations for different operating systems and architectures.

# - The primary responsibilities of the JVM are to load and execute Java bytecode, manage memory allocation and allocation, handle exceptions, and provide various runtime services.

# 2. JRE (Java Runtime Environment):

# - The JRE includes the JVM with additional libraries and resources needed to run Java applications.

# - In addition to the JVM, the JRE also includes features such as core libraries, help files, class libraries, Java runtime libraries, and configuration files.

# - Basically, the JRE provides the runtime environment needed to execute Java applications. - Developers and end users often use a JRE to install a Java application on their machines. In summary, a JVM is a virtual machine that acts as if it were Java bytecode, while a JRE provides a runtime environment that includes the JVM with the necessary libraries and resources to run Java applications.

# 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 | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# 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.

**Procedure:**

**Program 1:**

public class program\_1 {

    public static void main(String[] args) {

        int[][] matrix = new int[6][6];

        for (int i = 0; i < 6; i++) {

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

                matrix[i][j] = (int) (Math.random() \* 2);

            }

        }

        for (int i = 0; i < 6; i++) {

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

                System.out.printf("%d ", matrix[i][j]);

            }

            System.out.println();

        }

        System.out.println("Does every row has odd no. of 1's? " + (hasEveryRowOdd1s(matrix) ? "Yes" : "No"));

        System.out.println("Does every column has odd no. of 1's? " + (hasEveryColOdd1s(matrix) ? "Yes" : "No"));

    }

    public static boolean hasEveryRowOdd1s(int[][] m) {

        int count\_1 = 0;

        boolean hasEven = false;

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

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

                if (m[i][j] == 1) {

                    count\_1++;

                }

            }

            if (count\_1 % 2 != 0) {

                hasEven = true;

            } else {

                hasEven = false;

            }

            count\_1 = 0;

        }

        return hasEven;

    }

    public static boolean hasEveryColOdd1s(int[][] m) {

        int count\_1 = 0;

        boolean hasEven = false;

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

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

                if (m[j][i] == 1) {

                    count\_1++;

                }

            }

            if (count\_1 % 2 != 0) {

                hasEven = true;

            } else {

                hasEven = false;

            }

            count\_1 = 0;

        }

        return hasEven;

    }

}

**Program 3:**

import java.util.ArrayList;

public class program\_3 {

    public static void main(String[] args) {

        int[] arr = { 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 7, 8, 9, 9, 9 };

        ArrayList<Integer> ans = new ArrayList<>();

        int top = -1;

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

            if (ans.isEmpty()) {

                ans.add(arr[i]);

                top++;

            } else if (arr[i] != ans.get(top)) {

                ans.add(arr[i]);

                top++;

            }

        }

        System.out.printf("Before: [");

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

            if (i == arr.length - 1) {

                System.out.printf("%d", arr[i]);

            } else {

                System.out.printf("%d, ", arr[i]);

            }

        }

        System.out.println("]");

        System.out.println("After: " + ans);

    }

}

**Program 4:**

import java.util.Scanner;

public class program\_4 {

    public static void main(String[] args) {

        int arr\_1[][] = {

                { 1, 2, 3 },

                { 4, 5, 6 },

                { 7, 8, 9 }

        };

        int arr\_2[][] = {

                { 10, 11, 12 },

                { 13, 14, 15 },

                { 16, 17, 18 }

        };

        Scanner sc = new Scanner(System.in);

        char op;

        System.out.printf("Enter the operator(+, -, \*, /): ");

        op = sc.next().charAt(0);

        if (op == '+') {

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

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

                    System.out.printf("%d ", arr\_1[i][j] + arr\_2[i][j]);

                }

                System.out.println();

            }

        } else if (op == '-') {

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

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

                    System.out.printf("%d ", arr\_1[i][j] - arr\_2[i][j]);

                }

                System.out.println();

            }

        } else if (op == '\*') {

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

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

                    System.out.printf("%d ", arr\_1[i][j] \* arr\_2[i][j]);

                }

                System.out.println();

            }

        } else if (op == '/') {

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

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

                    System.out.printf("%d ", arr\_1[i][j] / arr\_2[i][j]);

                }

                System.out.println();

            }

        } else {

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

        }

        sc.close();

    }

}

**Program 5:**

public class program\_5 {

    public static void printArr(int[] arr) {

        int n = arr.length;

        System.out.printf("[");

        for (int i = 0; i < n; i++) {

            if (i == n - 1) {

                System.out.printf("%d]", arr[i]);

            } else {

                System.out.printf("%d, ", arr[i]);

            }

        }

        System.out.println();

    }

    public static void main(String[] args) {

        int[] arr = { 4, 7, 2, 1, 98, 3, 68, 9, 123, 54 };

        printArr(arr);

        int min\_index, temp, len = arr.length;

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

            min\_index = i;

            for (int j = i; j < len; j++) {

                if (arr[j] <= arr[min\_index]) {

                    min\_index = j;

                }

            }

            temp = arr[i];

            arr[i] = arr[min\_index];

            arr[min\_index] = temp;

        }

        printArr(arr);

    }

}

**Program 6:**

import java.util.Scanner;

public class program\_6 {

    private static boolean isVowel(char c) {

        c = Character.toLowerCase(c);

        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {

            return true;

        }

        return false;

    }

    private static int countVowels(String str) {

        int len = str.length(), vowels = 0;

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

            if (isVowel(str.charAt(i)) && str.charAt(i) != ' ') {

                vowels++;

            }

        }

        return vowels;

    }

    private static int countConsonants(String str) {

        int len = str.length(), consonants = 0;

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

            if ((!isVowel(str.charAt(i))) && str.charAt(i) != ' ') {

                consonants++;

            }

        }

        return consonants;

    }

    public static void main(String[] args) {

        String str;

        int vowels, consonants;

        Scanner sc = new Scanner(System.in);

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

        str = sc.nextLine();

        vowels = countVowels(str);

        consonants = countConsonants(str);

        System.out.printf("The entered string contains %d vowels and %d consonants", vowels, consonants);

        sc.close();

    }

}

**Program 7:**

import java.util.Scanner;

public class program\_7 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter the first string: ");

        String str1 = sc.nextLine();

        System.out.printf("Enter the second string: ");

        String str2 = sc.nextLine();

        String commonPrefix = findCommonPrefix(str1, str2);

        System.out.printf("Largest Common Prefix: %s", commonPrefix);

        sc.close();

    }

    static String findCommonPrefix(String str1, String str2) {

        int minLength = Math.min(str1.length(), str2.length());

        int index = 0;

        while (index < minLength && (str1.charAt(index) == str2.charAt(index))) {

            index++;

        }

        return str1.substring(0, index);

    }

}

**Program 8:**

import java.util.Scanner;

public class program\_8 {

    public static void main(String[] args) {

        String password;

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter a password: ");

        password = sc.next();

        boolean isValid = isValid(password);

        if (isValid) {

            System.out.println("Valid password!");

        } else {

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

        }

        sc.close();

    }

    private static boolean isValid(String pass) {

        if (pass.length() >= 8 && isAlnum(pass) && hasTwoDigits(pass)) {

            return true;

        }

        return false;

    }

    private static boolean isAlnum(String s) {

        boolean ans = false;

        int len = s.length();

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

            char c = s.charAt(i);

            if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {

                ans = true;

            } else {

                ans = false;

                break;

            }

        }

        return ans;

    }

    private static boolean hasTwoDigits(String s) {

        int count = 0;

        int len = s.length();

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

            if (s.charAt(i) >= '0' && s.charAt(i) <= '9') {

                count++;

            }

        }

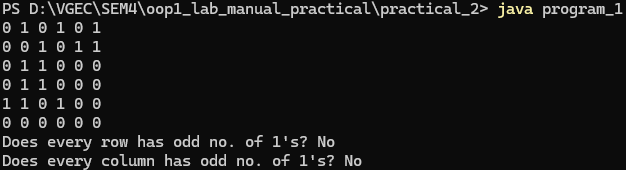
        return (count >= 2);

    }

}

**Observations:**

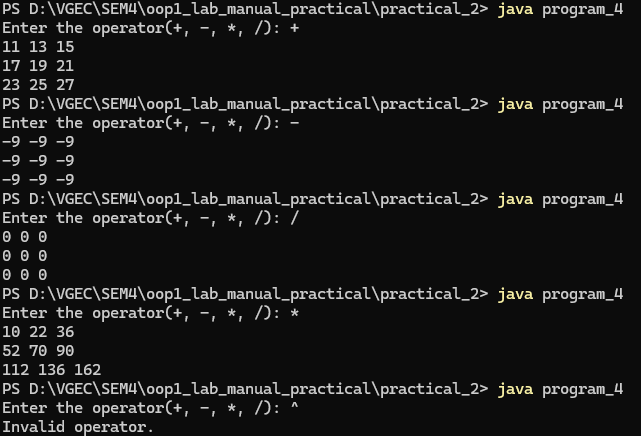
**Program 1:**

****

**Program 3:**

****

**Program 4:**

****

**Program 5:**

****

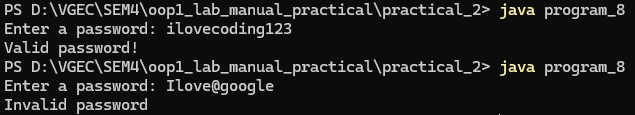
**Program 6:**

****

**Program 7:**

****

**Program 8:**

****

**Conclusion:** In this experiment we learnt many things about array. We learnt various operations on arrays like sorting array, removing duplicates along with some custom operations like checking validity of password, finding largest common prefix and finding number of vowels and consonants in a given string. We also learnt Arithmetic operations on 2D arrays. We also learnt generics in java.

**Quiz:** (Sufficient space to be provided for the answers)

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

# Ans. In Java, a ragged array, also known as a jagged array, is an array where each row can have a different length. This means that each row of the array can have a different number of elements. When you need to represent non-rectangular data structures, such as a matrix where each row contains a different number of elements

# Here's how you can use a ragged array in Java:

# public class RaggedArrayInstance {

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

# // Report an incised array

# int[][] raggedArray = new int[3][];

# 

# // Start each line with a different length

# raggedArray[0] = new int[]{1,2,3};

# raggedArray[1] = new int[][4,5};

# raggedArray[2] = new int[] {6, 7, 8, 9};

# 

# // Get the elements of the ragged array

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

# for (int j = 0; j < raggedArray[i].length; j++) {

# System.out.print(raggedArray[me][j] + "");

# }

# System.out.println();

# }

# }

# }

# In this example:

# - We declare a 2-dimensional array raggedArray with 3 rows, but we don't specify the size of the columns.

# - Provide an array of different size for each row and initialize each row with a different length.

# - We can access the elements of a ragged array by using nested loops, iterating on each row and iterating on each column in that row.

# Tear arrays are useful when you have data that is not neatly arranged in rectangles and offers great flexibility in representing such data.

# Differentiate String class and StringBuffer class.

# Ans. In Java, both the `String` class and the `StringBuffer` class are used to manipulate strings, but they differ in several key aspects:

# 1. Immutability:

# - String objects are immutable, meaning that once created, their values cannot be changed. Any operation that appears to modify a String actually creates a new String object.

# - StringBuffer objects, on the other hand, are mutable. You can modify the contents of a StringBuffer object without creating a new object each time.

# 2. Performance:

# - Because String objects are immutable, performing concatenation or modification operations on them can be inefficient, especially if done in loops. Each modification operation results in the creation of a new String object.

# - StringBuffer objects are designed for efficient string manipulation. They use a resizable buffer to store the string data, allowing for in-place modifications without the need to create new objects.

# 3. Thread Safety:

# - String objects are inherently thread-safe because they are immutable. Multiple threads can safely share and access String objects without risk of concurrent modification.

# - StringBuffer provides synchronized methods, making it safe for use in multi-threaded environments. However, this synchronization overhead can impact performance.

# - Java also provides a similar class called StringBuilder, which is similar to StringBuffer but is not synchronized, making it more efficient in single-threaded scenarios.

# Here's a summary of the differences between String and StringBuffer:

|  |  |  |
| --- | --- | --- |
| Aspect | String | StringBuffer |
| Immutability | Immutable | Mutable |
| Performance | Inefficient for mutations | Efficient for mutations |
| Thread Safety | Thread-safe | Thread-safe (synchronized) |

# In general, if you need to perform frequent string manipulations in a single-threaded environment, StringBuffer or StringBuilder is usually a better choice due to their mutability and better performance characteristics. If thread safety is not a concern and you don't need mutability, String is the preferred option due to its immutability, which can lead to simpler and more predictable code.

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

# Ans. public class TwoDimensionalArrayExample {

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

# // Declare a two-dimensional array

# int[][] twoDArray;

# 

# // Instantiate the two-dimensional array with specific dimensions

# twoDArray = new int[3][4]; // 3 rows, 4 columns

# 

# // Initialize the elements of the two-dimensional array

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

# for (int j = 0; j < twoDArray[i].length; j++) {

# twoDArray[i][j] = i \* twoDArray[i].length + j; // Assigning a value based on row and column index

# }

# }

# 

# // Print the elements of the two-dimensional array

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

# for (int j = 0; j < twoDArray[i].length; j++) {

# System.out.print(twoDArray[i][j] + " ");

# }

# System.out.println();

# }

# }

# }

# Explain the various String functions with their syntax.

# Ans. Here are explanations of various String functions with their syntax:

# 1. length(): Returns the length of the string.

# Syntax:

# int length()

# 

# 2. charAt(): Returns the character at the specified index.

# Syntax:

# char charAt(int index)

# 3. substring(): Returns a substring of the string.

# Syntax:

# String substring(int beginIndex)

# String substring(int beginIndex, int endIndex)

# 4. indexOf(): Returns the index of the first occurrence of a specified substring.

# Syntax:

# int indexOf(String str)

# int indexOf(String str, int fromIndex)

# 

# 5. lastIndexOf(): Returns the index of the last occurrence of a specified substring.

# Syntax:

# int lastIndexOf(String str)

# int lastIndexOf(String str, int fromIndex)

# 6. concat(): Concatenates the specified string to the end of this string.

# Syntax:

# String concat(String str)

# 7. toUpperCase(): Converts all characters in this string to uppercase.

# Syntax:

# String toUpperCase()

# 8. toLowerCase(): Converts all characters in this string to lowercase.

# Syntax:

# String toLowerCase()

# 9. trim(): Removes leading and trailing whitespace.

# Syntax:

# String trim()

# 10. startsWith(): Checks if this string starts with the specified prefix.

# Syntax:

# boolean startsWith(String prefix)

# 11. endsWith(): Checks if this string ends with the specified suffix.

# Syntax:

# boolean endsWith(String suffix)

# 12. replace(): Replaces all occurrences of a specified character or substring with another character or substring.

# Syntax:

# String replace(char oldChar, char newChar)

# String replace(CharSequence target, CharSequence replacement)

# These are some of the commonly used String functions in Java, along with their syntax.

# 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)

# JavaTPoint

# GFG

# W3schools

# ChatGPT

# Google

# Rubric wise marks obtained:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rubrics | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# 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.

**Procedure:**

**Program 1:**

public class program\_1 {

    public static void main(String[] args) {

        if (args.length == 0) {

            System.out.println("No student names provided.");

            return;

        }

        System.out.printf("No. of arguments  = %d:\n", args.length);

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

            String ordinal = getOrdinal(i + 1);

            System.out.printf("%s student name: %s\n", ordinal, args[i]);

        }

    }

    static String getOrdinal(int number) {

        if (number >= 10 && number <= 20) {

            return number + "th";

        }

        switch (number % 10) {

            case 1:

                return number + "st";

            case 2:

                return number + "nd";

            case 3:

                return number + "rd";

            default:

                return number + "th";

        }

    }

}

**Program 2:**

class rectangle {

    double width, height;

    rectangle() {

        width = height = 1;

    }

    rectangle(double w, double h) {

        width = w;

        height = h;

    }

    double getArea() {

        return width \* height;

    }

    double getPerimeter() {

        return 2 \* (width + height);

    }

}

public class program\_2 {

    public static void main(String[] args) {

        rectangle r1 = new rectangle();

        System.out.printf("Area of r1: %f\nPerimeter of r1: %f\n", r1.getArea(), r1.getPerimeter());

        rectangle r2 = new rectangle(3, 4);

        System.out.printf("Area of r2: %f\nPerimeter of r2: %f\n", r2.getArea(), r2.getPerimeter());

    }

}

**Program 3:**

class cartesianPoint {

    int x, y;

    cartesianPoint(int \_x, int \_y) {

        x = \_x;

        y = \_y;

    }

    cartesianPoint(int val) {

        x = y = val;

    }

    int getX(){

        return x;

    }

    int getY() {

        return y;

    }

    void move(int \_x, int \_y) {

        x = \_x;

        y = \_y;

    }

    void move(int val) {

        x = y = val;

    }

}

public class program\_3 {

    public static void main(String[] args) {

        cartesianPoint p1 = new cartesianPoint(4, 7);

        System.out.printf("For p1:\n");

        System.out.printf("Before:\nx: %d\ny: %d\n",p1.getX(),p1.getY());

        p1.move(8, 6);

        System.out.printf("After:\nx: %d\ny: %d\n",p1.getX(),p1.getY());

        cartesianPoint p2 = new cartesianPoint(5);

        System.out.printf("For p2:\n");

        System.out.printf("Before:\nx: %d\ny: %d\n",p2.getX(),p2.getY());

        p2.move(6);

        System.out.printf("After:\nx: %d\ny: %d\n",p2.getX(),p2.getY());

    }

}

**Program 4:**

import java.util.Scanner;

class employee {

    private String name;

    private int salary;

    employee(String name, int salary) {

        this.name = name;

        this.salary = salary;

    }

    void putData() {

        System.out.printf("Name: %s\nSalary: %d\n", name, salary);

    }

    void getData(String name, int salary) {

        this.name = name;

        this.salary = salary;

    }

}

public class program\_4 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        employee e1 = new employee("Tommy Vercetti", 120000);

        e1.putData();

        String name;

        int salary;

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

        name = sc.nextLine();

        System.out.printf("Enter the salary: ");

        salary = sc.nextInt();

        e1.getData(name, salary);

        e1.putData();

        sc.close();

    }

}

**Program 5:**

class time {

    private byte hour;

    private byte minute;

    void initialize(byte hour, byte minute) {

        this.hour = hour;

        this.minute = minute;

    }

    time sum(time t) {

        int h = this.hour + t.hour;

        int m = this.minute + t.minute;

        time t1 = new time();

        if (m >= 60) {

            h += m / 60;

            m %= 60;

        }

        t1.initialize((byte) h, (byte) m);

        return t1;

    }

    void printTime() {

        System.out.printf("%s : %s\n", this.hour, this.minute);

    }

}

public class program\_5 {

    public static void main(String[] args) {

        time t1 = new time();

        t1.initialize((byte) 21, (byte) 45);

        t1.printTime();

        time t2 = new time();

        t2.initialize((byte) 2, (byte) 56);

        t1 = t1.sum(t2);

        t1.printTime();

    }

}

**Program 6:**

class point {

    private int x, y;

    point() {

        this.x = 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 display() {

        System.out.printf("Coords = (%d, %d)\n", this.x, this.y);

    }

}

public class program\_6 {

    public static void main(String[] args) {

        point p1 = new point();

        point p2 = new point(4, 9);

        System.out.println("p1:");

        p1.display();

        System.out.println("p2:");

        p2.display();

        point p1\_copy = new point(p1);

        point p2\_copy = new point(p2);

        System.out.println("p1\_copy:");

        p1\_copy.display();

        System.out.println("p2\_copy:");

        p2\_copy.display();

    }

}

**Program 7:**

import java.util.Scanner;

class account {

    private int id, balance;

    private String name;

    account(int id, int balance, String name) {

        this.id = id;

        this.balance = balance;

        this.name = name;

    }

    void setData(int id, int balance, String name) {

        this.id = id;

        this.balance = balance;

        this.name = name;

    }

    void getData() {

        System.out.printf("Account ID: %d\nAccount Holder Name: %s\nBalance: %d\n", this.id, this.name, this.balance);

    }

    int getID() {

        return this.id;

    }

}

public class program\_7 {

    public static void main(String[] args) {

        account a[] = new account[5];

        a[0] = new account(0, 20000000, "Cesar");

        a[1] = new account(1, 999999999, "Carl Johnson");

        a[2] = new account(2, 56000, "Officer Tempenny");

        a[3] = new account(3, 43000, "Eddie Pulaski");

        a[4] = new account(4, 3400000, "Officer Herandez");

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter an account number: ");

        int acc\_no = sc.nextInt();

        boolean user\_found = false;

        for (int i = 0; i < 5; i++) {

            if (a[i].getID() == acc\_no) {

                a[i].getData();

                user\_found = true;

            }

        }

        if (!user\_found) {

            System.out.println("User not found!");

        }

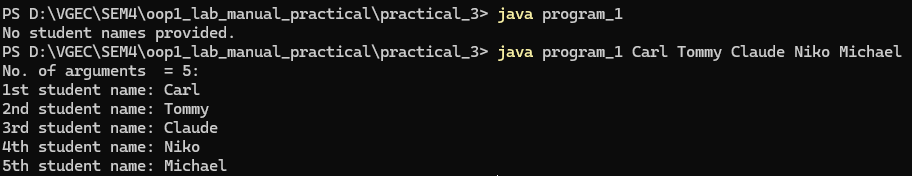
        sc.close();

    }

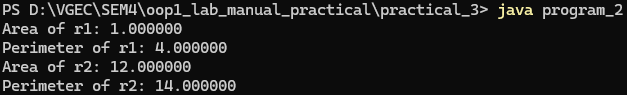
}

**Observations:**

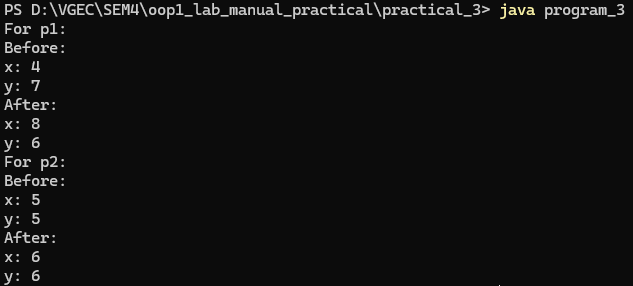
**Program 1:**

****

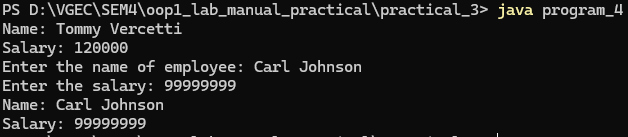
**Program 2:**

****

**Program 3:**

****

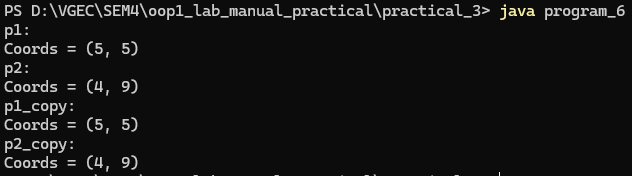
**Program 4:**

****

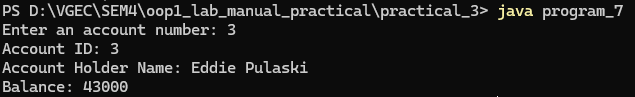
**Program 5:**

****

**Program 6:**

****

**Program 7:**

****

**Conclusion:** In this experiment we learnt to implement oops concepts in our code. We learnt about concepts like access controllers, constructor, methods, copy constructor and various other oops concepts.

**Quiz:** (Sufficient space to be provided for the answers)

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

# Ans. Encapsulation is one of the basic principles of object-oriented programming (OOP) and defines the combination of data (properties) and methods (behaviors) that operate on that data, called class In encapsulation an object the state is hidden from the outside world It is only possible on interfaces (routes).

# Here is why encapsulation is important in OOP:

# 1. Data Hiding: Encapsulation hides the contents of an object from the outside world. This prevents direct access to the product data, reducing the risk of inadvertent modification or misuse.

# 2. Abstractness: Encapsulation enables the concept of abstraction by defining an explicit interface for interaction with an object. Users do not need to know how product data is stored or processed internally; They just need to understand how to use its public methods.

# 3. Modularity: Encapsulation promotes modularity by encapsulating related data and methods in a single unit (class). This makes the code easier to understand, maintain, and modify, because changes to the class implementation information do not affect other parts of the implementation.

# 4. Code Reusability: Encapsulation facilitates code reusability by encapsulating common functions in reusable components (classes). These features can easily be reused in other program components or tasks without duplicating the code.

# Security: Encapsulation helps improve security by preventing direct access to the object state. Providing controlled methods, encapsulation can implement authentication, error checking, and security policies within the class

# Overall, encapsulation is a key principle in OOP as it promotes data hiding, abstraction, modularity, code reusability, and security, leading to more robust, maintainable, and scalable software systems.

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

# Ans. In object-oriented programming (OOP), a class is a blueprint for creating objects. It defines the attributes (data) and actions (methods) that objects in that class will have. The class acts as a template or prototype for individual objects.

# Here is a breakdown of the basic class components and their relationships to objects.

# 1. Attributes (Data): A class defines attributes that represent the state of objects created from that class. Attributes are also called member variables, fields, or properties. These properties describe the characteristics or properties of objects.

# 2. Methods (interactions): A class also defines methods (functions) that can represent the behavior or actions of an object of that class. Methods define operations that an object can perform on its data.

# 3. Object Instantiation: Once a class is defined, objects (instances) can be created from it using a process called instantiation. Each object created from a class has its own attributes, but they share the same methods defined by the class.

# 4. Relationship between Class and Object: Every object in OOP is an instance of a class. A class provides structure and behavior for objects, while objects represent individual instances of that class with their own unique state.

# 5. Inheritance: Classes can also participate in inheritance, where one class (subclass or child class) can inherit attributes and methods from another class (superclass or parent class). This allows code to be reused and relationships between classes to be created systematically.

# 6. Encapsulation: Classes encapsulate data and methods in a single object, providing a method for storing and extracting data. Encapsulation hides the contents of a class from external access, promoting modularity and security.

# In summary, a class in OOP is a template for creating objects, defining their attributes and behavior. Objects are instances of classes and represent individual objects with their own states and behaviors, while classes provide settings and behaviors shared by all instances of that class

# Define and explain static and dynamic binding.

# Ans. Static and dynamic binding are concepts related to polymorphism in object-oriented programming. They determine the method invocation mechanism at compile-time and runtime, respectively.

# 1. Static Binding:

# - Static binding (also known as early binding) refers to the process of linking a method call to the method implementation at compile-time.

# - In static binding, the compiler determines which method to call based on the reference type of the object, not the actual type of the object at runtime.

# - Static binding occurs for methods that are:

# - Final methods

# - Private methods

# - Static methods

# - Constructors

# Example:

# 

# class A {

# void display() {

# System.out.println("A's display");

# }

# }

# class B extends A {

# void display() {

# System.out.println("B's display");

# }

# }

# public class Main {

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

# A obj = new B();

# obj.display(); // Static binding: A's display is called at compile-time

# }

# }

# 

# 2. Dynamic Binding:

# - Dynamic binding (also known as late binding or runtime polymorphism) refers to the process of linking a method call to the method implementation at runtime.

# - In dynamic binding, the JVM determines which method to call based on the actual type of the object at runtime.

# - Dynamic binding occurs for non-static and non-private methods, i.e., instance methods overridden in subclasses.

# Example:

# class A {

# void display() {

# System.out.println("A's display");

# }

# }

# class B extends A {

# void display() {

# System.out.println("B's display");

# }

# }

# public class Main {

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

# A obj = new B();

# obj.display(); // Dynamic binding: B's display is called at runtime

# }

# }

# 

# In summary, static binding occurs at compile-time and is based on the reference type, while dynamic binding occurs at runtime and is based on the actual object type. Dynamic binding enables polymorphic behavior, allowing different subclasses to provide their own implementation of methods defined in the superclass.

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

# Ans. Method overloading and method overriding are two important concepts in object-oriented programming (OOP) that involve the usage of methods in classes. Here's an explanation of each:

# 1. Method Overloading:

# - Method overloading refers to defining multiple methods in a class with the same name but different parameter lists (i.e., different number or types of parameters).

# - The overloaded methods must have different parameter lists or types, or both.

# - Method overloading allows developers to provide multiple ways to call a method, making the code more flexible and readable.

# - Overloaded methods are resolved at compile-time based on the number and types of arguments passed to them.

# Example:

# 

# public class Calculator {

# // Overloaded methods

# public int add(int a, int b) {

# return a + b;

# }

# public double add(double a, double b) {

# return a + b;

# }

# }

# 

# 2. Method Overriding:

# - Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

# - The overriding method must have the same name, return type, and parameter list (or a covariant return type) as the method in the superclass.

# - Method overriding allows subclasses to provide their own implementation of inherited methods, enabling polymorphic behavior.

# - Overridden methods are resolved at runtime based on the actual type of the object.

# Example:

# 

# class Animal {

# void makeSound() {

# System.out.println("Animal makes a sound");

# }

# }

# class Dog extends Animal {

# @Override

# void makeSound() {

# System.out.println("Dog barks");

# }

# }

# In summary, method overloading involves defining multiple methods with the same name but different parameter lists within a class, while method overriding involves providing a specific implementation of a method that is already defined in a superclass within a subclass. Overloading is resolved at compile-time based on the method signature, whereas overriding is resolved at runtime based on the actual type of the object. Both concepts contribute to the flexibility and extensibility of object-oriented designs.

# 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)

# JavaTPoint

# W3Schools

# GFG

# ChatGPT

# Google

# Rubric wise marks obtained:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rubrics | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# 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).

**Procedure:**

**Program 1:**

public class program\_1 {

    private static *boolean* checkPalindrome(String *s*) {

        StringBuilder reverse = new StringBuilder();

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

            reverse.append(*s*.charAt(i));

        }

        return *s*.equals(reverse.toString());

    }

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

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

            System.out.printf("Is string %d palindrome? %b\n", i, checkPalindrome(*args*[i]));

        }

    }

}

**Program 2:**

class BankAccount {

    protected String accountNumber;

    private String holderName;

    protected *double* balance;

    public BankAccount(String *accountNumber*, String *holderName*, *double* *balance*) {

        this.accountNumber = *accountNumber*;

        this.holderName = *holderName*;

        this.balance = *balance*;

    }

    public *void* openAccount() {

        System.out.println("Account " + accountNumber + " opened for " + holderName);

    }

    public *void* deposit(*double* *amount*) {

        if (*amount* > 0) {

            balance += *amount*;

            System.out.println("Deposited " + *amount* + " into account " + accountNumber + ". New balance: " + balance);

        } else {

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

        }

    }

    public *void* withdraw(*double* *amount*) {

        if (*amount* > 0 && *amount* <= balance) {

            balance -= *amount*;

            System.out.println("Withdrew " + *amount* + " from account " + accountNumber + ". New balance: " + balance);

        } else {

            System.out.println("Insufficient funds for withdrawal.");

        }

    }

    public *void* checkBalance() {

        System.out.println("Balance in account " + accountNumber + ": " + balance);

    }

}

class SavingAccount extends BankAccount {

    private static final *double* INTEREST\_RATE = 0.05;

    public SavingAccount(String *accountNumber*, String *holderName*, *double* *balance*) {

        super(*accountNumber*, *holderName*, *balance*);

    }

    public *void* calInterest() {

*double* interest = balance \* INTEREST\_RATE;

        System.out.println("Interest calculated for saving account " + accountNumber + ": " + interest);

        deposit(interest);

    }

}

class FixedDepositAccount extends BankAccount {

    private static final *double* INTEREST\_RATE = 0.08; // 8% interest rate

    public FixedDepositAccount(String *accountNumber*, String *holderName*, *double* *balance*) {

        super(*accountNumber*, *holderName*, *balance*);

    }

    public *void* calInterest() {

*double* interest = balance \* INTEREST\_RATE;

        System.out.println("Interest calculated for fixed deposit account " + accountNumber + ": " + interest);

        deposit(interest);

    }

}

public class program\_2 {

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

        SavingAccount savingAcc = new SavingAccount("SA123", "Tommy Vercetti", 999999999);

        savingAcc.openAccount();

        savingAcc.checkBalance();

        savingAcc.deposit(500);

        savingAcc.checkBalance();

        savingAcc.withdraw(200);

        savingAcc.checkBalance();

        savingAcc.calInterest();

        savingAcc.checkBalance();

        FixedDepositAccount fdAcc = new FixedDepositAccount("FD456", "Carl Johnson", 670000000);

        fdAcc.openAccount();

        fdAcc.checkBalance();

        fdAcc.deposit(1000);

        fdAcc.checkBalance();

        fdAcc.withdraw(500);

        fdAcc.checkBalance();

        fdAcc.calInterest();

        fdAcc.checkBalance();

    }

}

**Program 3:**

class areaCalculator {

*double* find\_area(*double* *l*) {

        return *l* \* *l*;

    }

*double* find\_area(*double* *w*, *double* *h*) {

        return *w* \* *h*;

    }

*double* find\_area(*double* *b*, *double* *h*, *boolean* *isTriangle*) {

        if (*isTriangle*)

            return 0.5 \* *b* \* *h*;

        else

            return -1;

    }

}

public class program\_3 {

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

        areaCalculator a = new areaCalculator();

*double* square\_length = 5.7;

        System.out.printf("Area of square = %f sq. units\n", a.find\_area(square\_length));

*double* rectangle\_width = 7.6, rectangle\_height = 8.9;

        System.out.printf("Area of rectangle = %f sq. units\n", a.find\_area(rectangle\_width, rectangle\_height));

*double* triangle\_base = 9.6, triangle\_height = 7.3;

*boolean* isTriangle = true;

        System.out.printf("Area of triangle = %f sq. units\n", a.find\_area(triangle\_base, triangle\_height, isTriangle));

    }

}

**Program 4:**

class volumeCalculator {

*double* find\_volume(*double* *l*) {

        return *l* \* *l* \* *l*;

    }

*double* find\_volume(*double* *l*, *double* *b*, *double* *h*) {

        return *l* \* *b* \* *h*;

    }

*double* find\_volume(*double* *r*, *boolean* *isSphere*) {

        if (*isSphere*) {

*double* pi = 3.1415926;

            return (4 / 3) \* pi \* *r* \* *r* \* *r*;

        } else {

            return -1;

        }

    }

}

public class program\_4 {

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

        volumeCalculator v = new volumeCalculator();

*double* cube\_length = 6.5;

        System.out.printf("The volume of the cube is %f cu. units\n", v.find\_volume(cube\_length));

*double* rect\_cube\_length = 4.6, rect\_cube\_width = 7.8, rect\_cube\_height = 9.6;

        System.out.printf("The volume of the rectangular cube is %f cu. units\n",

                v.find\_volume(rect\_cube\_length, rect\_cube\_width, rect\_cube\_height));

*double* sphere\_radius = 3.7;

*boolean* isSphere = true;

        System.out.printf("The volume of the sphere is %f cu. units\n", v.find\_volume(sphere\_radius, isSphere));

    }

}

**Program 5:**

class employee {

    protected String name, empId, deptName;

    protected *int* salary, performanceScore;

    employee(String *empId*, String *name*, *int* *salary*, String *deptName*) {

        this.empId = *empId*;

        this.name = *name*;

        this.salary = *salary*;

        this.deptName = *deptName*;

        this.performanceScore = 0;

    }

*void* incrementSalary() {

        salary += 0.1 \* salary;

    }

*void* changePerformanceScore(*int* *score*) {

        performanceScore += *score*;

    }

*void* givePromotion() {

        if (this.performanceScore > 1500) {

            System.out.printf("Promoted Successfully.\n");

            this.performanceScore -= 1500;

            salary += 0.15 \* salary;

        } else {

            System.out.printf("Try harder.\n");

        }

    }

*void* printEmployee() {

        System.out.printf("{ID: %s, Name: %s, Dept name: %s, Salary: %d, Performance score: %d}\n", this.empId,

                this.name, this.deptName, this.salary, this.performanceScore);

    }

}

class programmer extends employee {

    protected String language, techStack;

    programmer(String *empId*, String *name*, *int* *salary*, String *language*, String *techStack*) {

        super(*empId*, *name*, *salary*, "IT");

        this.techStack = *techStack*;

        this.language = *language*;

    }

*void* incrementSalary() {

        salary += 0.2 \* salary;

    }

*void* printProgrammer() {

        printEmployee();

        System.out.printf("{Main language: %s, Tech stack: %s}\n", this.language, this.techStack);

    }

*void* changeLanguage(String *language*) {

        this.language = *language*;

    }

*void* changeTechStack(String *techStack*) {

        this.techStack = *techStack*;

    }

}

class manager extends employee {

    manager(String *empId*, String *name*, *int* *salary*) {

        super(*empId*, *name*, *salary*, "Management");

    }

*void* incrementSalary() {

        salary += 0.3 \* salary;

    }

*void* printManager() {

        printEmployee();

    }

    employee hire(String *empId*, String *name*, *int* *salary*, String *dept*) {

        employee e = new employee(*empId*, *name*, *salary*, *dept*);

        System.out.printf("Employee hired successfully!\n");

        return e;

    }

}

public class program\_5 {

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

        employee e1 = new employee("Emp001", "Carl Johnson", 200000, "Lawyer");

        e1.printEmployee();

        e1.changePerformanceScore(400);

        e1.printEmployee();

        e1.givePromotion();

        e1.changePerformanceScore(1500);

        e1.printEmployee();

        e1.givePromotion();

        e1.printEmployee();

        e1.incrementSalary();

        e1.printEmployee();

        System.out.println();

        programmer p1 = new programmer("Pro001", "Cesar", 400000, "C++", "MERN");

        p1.printProgrammer();

        p1.changePerformanceScore(400);

        p1.printProgrammer();

        p1.givePromotion();

        p1.changePerformanceScore(1500);

        p1.printProgrammer();

        p1.givePromotion();

        p1.printProgrammer();

        p1.incrementSalary();

        p1.printProgrammer();

        p1.changeLanguage("Rust");

        p1.printProgrammer();

        p1.changeTechStack("MEAN");

        p1.printProgrammer();

        System.out.println();

        manager m1 = new manager("Man001", "Tommy Vercetti", 999999999);

        m1.printManager();

        m1.changePerformanceScore(400);

        m1.givePromotion();

        m1.printManager();

        m1.changePerformanceScore(1500);

        m1.givePromotion();

        m1.printManager();

        m1.incrementSalary();

        m1.printManager();

        employee e2 = m1.hire("Emp002", "Trevor", 54000, "Service");

        e2.printEmployee();

    }

}

**Program 6:**

abstract class shape {

    protected *double* l, w;

*void* getData(*double* *l*, *double* *w*) {

        this.l = *l*;

        this.w = *w*;

    }

    abstract *void* displayArea();

}

class rectangle extends shape {

*void* displayArea() {

        System.out.printf("Area of rectangle = %f\n", this.l \* this.w);

    }

}

class triangle extends shape {

*void* displayArea() {

        System.out.printf("Area of traingle = %f\n", 0.5 \* this.l \* this.w);

    }

}

public class program\_6 {

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

        shape s;

        rectangle r = new rectangle();

        r.getData(5.7, 6.1);

        r.displayArea();

        triangle t = new triangle();

        t.getData(5.7, 6.1);

        t.displayArea();

        s = r;

        s.displayArea();

        s = t;

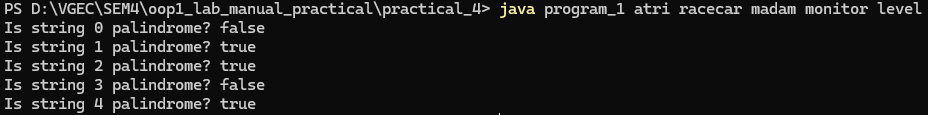
        s.displayArea();

    }

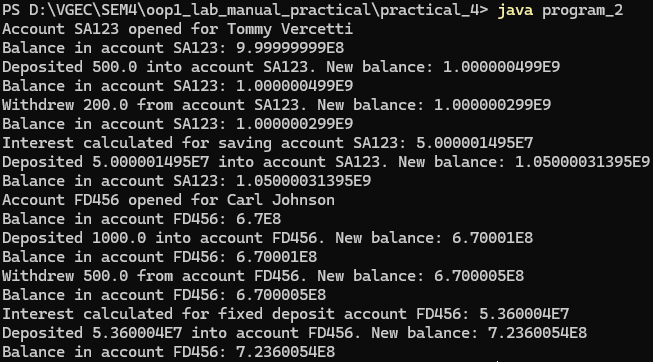
}

**Observations:**

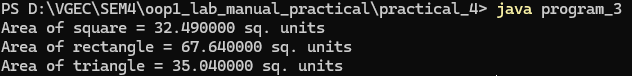
**Program 1:**

****

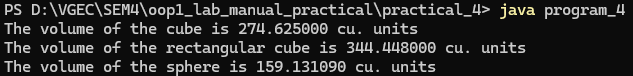
**Program 2:**

****

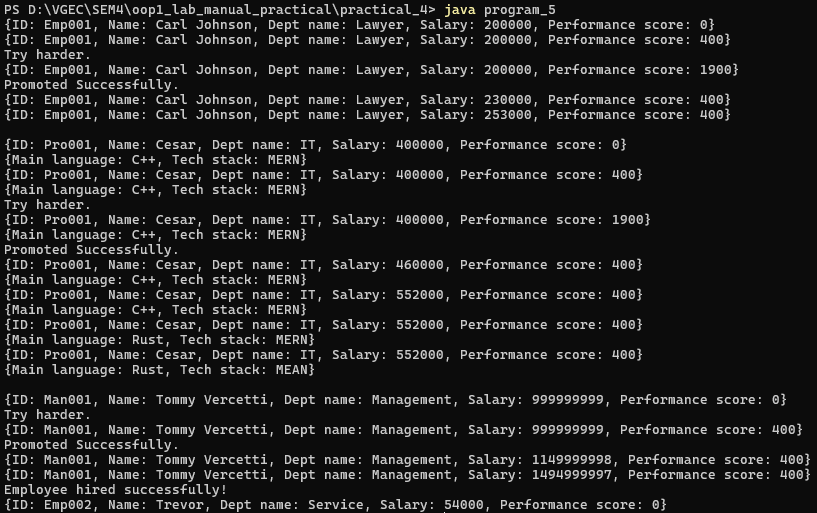
**Program 3:**

****

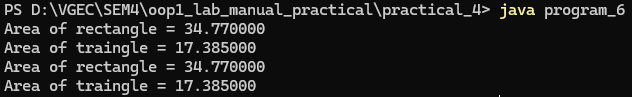
**Program 4:**

****

**Program 5:**

****

**Program 6:**

****

**Conclusion:** In this experiment we learnt about a concept of OOPs called inheritance and to implement it in code. We also learnt method over-riding, abstract class, etc. in this. We implemented real life scenarios in code using inheritance. E.g. Area calculator, Simple bank account management system, Volume calculator and Simple employee management system.

**Quiz:**

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

# Ans. Inheritance is a key feature of object-oriented programming (OOP) that allows a class to inherit properties and behaviors (methods) from another class. Inheritance enables code reuse and promotes the concept of "is-a" relationships between classes.

# In Java, there are different types of inheritance, including:

# 1. Single Inheritance:

# - Single inheritance refers to the concept where a subclass (child class) inherits from only one superclass (parent class).

# - The subclass inherits all the non-private fields and methods of its superclass.

# - Java does not support multiple inheritance to avoid ambiguity and complexities.

# Example:

# 

# class Animal {

# void eat() {

# System.out.println("Animal eats");

# }

# }

# class Dog extends Animal {

# void bark() {

# System.out.println("Dog barks");

# }

# }

# 

# 2. Multilevel Inheritance:

# - Multilevel inheritance refers to the concept where a subclass (child class) inherits from another subclass, creating a hierarchy of classes.

# - Each subclass inherits properties and behaviors from its immediate superclass.

# - This forms a chain of inheritance.

# Example:

# 

# class Animal {

# void eat() {

# System.out.println("Animal eats");

# }

# }

# class Dog extends Animal {

# void bark() {

# System.out.println("Dog barks");

# }

# }

# class Labrador extends Dog {

# void color() {

# System.out.println("Labrador is black");

# }

# }

# 

# 3. Hierarchical Inheritance:

# - Hierarchical inheritance refers to the concept where multiple subclasses (child classes) inherit from a single superclass (parent class).

# - Each subclass shares common properties and behaviors from the superclass but may have additional specific properties or behaviors.

# Example:

# 

# class Animal {

# void eat() {

# System.out.println("Animal eats");

# }

# }

# class Dog extends Animal {

# void bark() {

# System.out.println("Dog barks");

# }

# }

# class Cat extends Animal {

# void meow() {

# System.out.println("Cat meows");

# }

# }

# 

# In summary, inheritance in Java allows subclasses to inherit properties and behaviors from superclasses, enabling code reuse and promoting the "is-a" relationship between classes. Different types of inheritance, such as single, multilevel, and hierarchical inheritance, provide flexibility in designing class hierarchies to represent real-world entities effectively.

# To avoid ambiguity and complexity, Java does not support multiple properties

# Explain the use of final and Super keyword in JAVA

# Ans. In Java, the ‘final’ and ‘super’ keywords serve different purposes:

# 1. final Keyword:

# - The ‘final’ keyword is a modifier that can be applied to variables, methods, and classes.

# - When applied to a variable, it indicates that the variable's value cannot be changed once initialized (i.e., it becomes a constant).

# - When applied to a method, it indicates that the method cannot be overridden by subclasses.

# - When applied to a class, it indicates that the class cannot be subclassed (i.e., it cannot have subclasses).

# - Using ‘final’ provides immutability, security, and optimization benefits.

# Example:

# 

# final int MAX\_VALUE = 100;

# final void printMessage() {

# System.out.println("This method cannot be overridden.");

# }

# final class MyClass {

# // Class definition

# }

# 

# 2. super Keyword:

# - The ‘super’ keyword in Java is used to refer to the superclass (parent class) of the current object.

# - It can be used to access superclass methods and variables, call superclass constructors, and differentiate between overridden and superclass methods.

# - ‘super()’ is used to call the constructor of the superclass from the subclass constructor. It must be the first statement in the subclass constructor.

# Example:

# 

# class Animal {

# void eat() {

# System.out.println("Animal eats");

# }

# }

# class Dog extends Animal {

# void eat() {

# super.eat(); // Call superclass method

# System.out.println("Dog eats");

# }

# }

# 

# Example with constructor:

# 

# class Animal {

# Animal() {

# System.out.println("Animal constructor");

# }

# }

# class Dog extends Animal {

# Dog() {

# super(); // Call superclass constructor

# System.out.println("Dog constructor");

# }

# }

# 

# In summary, the `final` keyword is used to create constants, prevent method overriding, or restrict subclassing, while the `super` keyword is used to access superclass members, call superclass constructors, and differentiate between superclass and subclass methods. Both keywords play important roles in Java programming, providing control over class behavior and inheritance.

# Define polymorphism with its need.

# Ans. Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent different underlying forms (types) and behaviors.

# There are two types of polymorphism in Java:

# 1. Compile-time Polymorphism (Method Overloading):

# - Compile-time polymorphism, also known as static polymorphism or early binding, occurs when multiple methods in a class have the same name but different parameter lists.

# - The compiler determines which method to call based on the number and types of arguments passed to it.

# - Method overloading is an example of compile-time polymorphism.

# Example:

# 

# class Calculator {

# int add(int a, int b) {

# return a + b;

# }

# double add(double a, double b) {

# return a + b;

# }

# }

# 

# 2. Run-time Polymorphism (Method Overriding):

# - Run-time polymorphism, also known as dynamic polymorphism or late binding, occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

# - The method to be invoked is determined at runtime based on the actual type of the object.

# - Method overriding is an example of run-time polymorphism.

# Example:

# 

# class Animal {

# void makeSound() {

# System.out.println("Animal makes a sound");

# }

# }

# class Dog extends Animal {

# @Override

# void makeSound() {

# System.out.println("Dog barks");

# }

# }

# 

# Need for Polymorphism:

# 1. Code Reusability: Polymorphism promotes code reusability by allowing methods with the same name to be used across different classes or subclasses. This reduces code duplication and promotes a modular approach to programming.

# 2. Flexibility and Extensibility: Polymorphism makes the code more flexible and extensible by allowing different implementations of methods to be provided by subclasses. This enables easy modification and enhancement of functionality without affecting existing code.

# 3. Abstraction: Polymorphism enables the use of abstract interfaces or superclass types to represent objects of different concrete types. This simplifies the code and promotes a high level of abstraction, allowing developers to focus on the common behavior rather than specific implementations.

# 4. Dynamic Behavior: Polymorphism allows the behavior of objects to be determined dynamically at runtime based on their actual types. This enables the implementation of complex and dynamic behaviors in applications.

# In summary, polymorphism is a powerful mechanism in OOP that promotes code reuse, flexibility, abstraction, and dynamic behavior, leading to more modular, extensible, and maintainable software systems.

# Explain about Encapsulation, Abstraction.

# Ans. Encapsulation and abstraction are two important concepts in object-oriented programming (OOP) that promote code organization, modularity, and information hiding.

# 1. Encapsulation:

# - Encapsulation is the bundling of data (attributes) and methods (behaviors) that operate on that data into a single unit, called a class.

# - It hides the internal state and implementation details of an object from the outside world, providing controlled access to the object's properties and behaviors through well-defined interfaces (public methods).

# - Encapsulation helps ensure data integrity, security, and maintainability by preventing direct access to an object's internal state and providing controlled access through methods.

# - It also promotes the concept of data abstraction by separating the interface from the implementation details.

# Example:

# 

# public class Car {

# private String model;

# private int year;

# // Getter and setter methods

# public String getModel() {

# return model;

# }

# public void setModel(String model) {

# this.model = model;

# }

# public int getYear() {

# return year;

# }

# public void setYear(int year) {

# this.year = year;

# }

# }

# 

# 2. Abstraction:

# - Abstraction is the process of representing the essential features of an object while hiding the unnecessary details.

# - It focuses on what an object does rather than how it does it, providing a simplified and conceptual view of complex systems.

# - Abstraction allows developers to create models of real-world entities as classes with properties and behaviors, abstracting away the complexities of the underlying implementation.

# - It promotes code reusability, maintainability, and extensibility by providing a clear separation between interface and implementation.

# Example:

# 

# public abstract class Shape {

# // Abstract method representing the area of the shape

# public abstract double area();

# }

# public class Circle extends Shape {

# private double radius;

# // Constructor and implementation of abstract method

# public Circle(double radius) {

# this.radius = radius;

# }

# @Override

# public double area() {

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

# }

# }

# 

# In summary, encapsulation and abstraction are key principles in OOP that promote modular, maintainable, and scalable software designs. Encapsulation hides the internal state of objects and provides controlled access through methods, while abstraction focuses on representing essential features and behaviors while hiding unnecessary details. Together, they enable effective data management, code organization, and system design in OOP.

# 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)

# JavaTPoint

# GFG

# W3Schools

# Rubric wise marks obtained:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rubrics | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# 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.

**Procedure:**

**Program 1:**

abstract class shape {

    abstract *void* displayArea();

}

class traingle extends shape {

*double* a, b, c;

    traingle(*double* *a*, *double* *b*, *double* *c*) {

        this.a = *a*;

        this.b = *b*;

        this.c = *c*;

    }

    @*Override*

*void* displayArea() {

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

*double* a\_2 = s \* (s - a) \* (s - b) \* (s - c);

*double* a = Math.sqrt(a\_2);

        System.out.printf("Area of the triangle is: %f\n", a);

    }

}

class rectangle extends shape {

*double* l, w;

    rectangle(*double* *l*, *double* *w*) {

        this.l = *l*;

        this.w = *w*;

    }

    @*Override*

*void* displayArea() {

        System.out.printf("Area of the rectangle is: %f\n", this.l \* this.w);

    }

}

class circle extends shape {

*double* r;

    circle(*double* *r*) {

        this.r = *r*;

    }

    @*Override*

*void* displayArea() {

        System.out.printf("Area of the circle is: %f\n", 3.1415926 \* this.r \* this.r);

    }

}

public class program\_1 {

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

        shape s;

        traingle t = new traingle(4.38, 6.24, 9.35);

        t.displayArea();

        s = t;

        s.displayArea();

        rectangle r = new rectangle(4.65, 6.24);

        r.displayArea();

        s = r;

        s.displayArea();

        circle c = new circle(4.24);

        c.displayArea();

        s = c;

        s.displayArea();

    }

}

**Program 2:**

interface I1 {

}

interface I2 {

}

interface I3 extends *I1*, *I2* {

}

interface I4 {

}

class X implements I3 {

}

class W extends X implements I4 {

}

public class program\_2 {

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

        W obj = new W();

        if (obj instanceof I1) {

            System.out.println("obj implements interface I1");

        }

        if (obj instanceof I2) {

            System.out.println("obj implements interface I2");

        }

        if (obj instanceof I3) {

            System.out.println("obj implements interface I3");

        }

        if (obj instanceof I4) {

            System.out.println("obj implements interface I4");

        }

        if (obj instanceof X) {

            System.out.println("obj is of type X");

        }

    }

}

**Program 3:**

import java.util.Scanner;

interface exam {

*boolean* pass(*int* *marks*);

}

interface classify {

    String division(*int* *average*);

}

class result implements exam, classify {

    private *int* marks;

    public *boolean* pass(*int* *marks*) {

        return *marks* >= 50;

    }

    public String division(*int* *average*) {

        if (*average* >= 60) {

            return "First";

        } else if (*average* >= 50) {

            return "Second";

        } else {

            return "No Division";

        }

    }

}

public class program\_3 {

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

        Scanner sc = new Scanner(System.in);

        System.out.printf("Enter your marks: ");

*int* marks = sc.nextInt();

        result r = new result();

        System.out.printf("Has student passed the exam? %b\n", r.pass(marks));

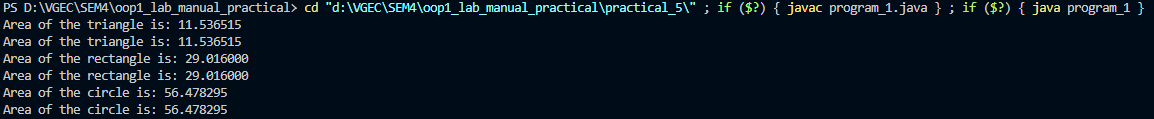
        System.out.printf("Division: %s\n", r.division(marks));

    }

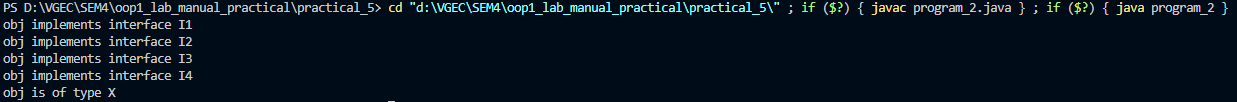
}

**Observations:**

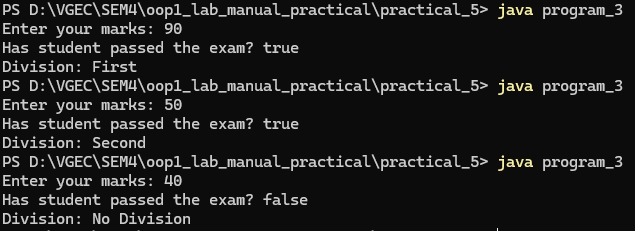
**Program 1:**

****

**Program 2:**

****

**Program 3:**

****

**Conclusion:** In this experiment we learned about some more advanced concepts of java named as abstract classes and interfaces. Abstract class is a class which has minimum 1 abstract method but an interface must contain abstract methods. Interface also allows multiple inheritance in java. We use ‘extends’ keyword for child class of a particular parent class and we use ‘implements’ keyword for a child class of an interface but we use ‘extends’ keyword for a child interface of a particular parent interface.

**Quiz:** (Sufficient space to be provided for the answers)

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

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

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

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

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

# 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 | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# Signature of Faculty:

**Experiment No: 6**

**A.**

**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.

**B.**

**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 sleep() 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.

4. Write a program that demonstrate thread priority four threads each with a different priority level then the other are started objects and not the behave of each Thread

5. Write a program that demonstrate use of Executor Framework in mutitasking.

6. Write a program for handling producer consumer problem.

**Procedure:**

**//Write program here**

**Observations:**

**//Write program output here**

**Conclusion:** (Sufficient space to be provided)

**Quiz:** (Sufficient space to be provided for the answers)

# Can you explain the difference between a process and a thread in the context of multithreading?

# What are the different states in the lifecycle of a Java thread, and how does a thread transition between them?

# What is runnable interface? How can you use this interface in creating thread?

# Explain the concept of thread synchronization and the role of the synchronized keyword.

# Explain: wait, sleep, notify and notify all.

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

1. Explain the benefits of organizing classes into packages in a Java application**.**
2. What is an exception in Java, and why is exception handling important in software development?
3. Explain the try-catch-finally block and its role in handling exceptions.
4. What is difference between throw and throws?

# 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 | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# 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:** Put Output of the program

**Conclusion:** (Sufficient space to be provided)

**Quiz:** (Sufficient space to be provided for the answers)

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

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

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

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

# 

# 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 | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# Signature of Faculty:

**Experiment No: 8**

**A.**

**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();

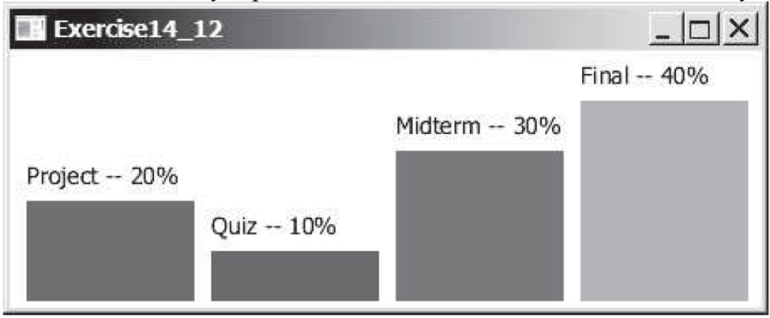
}

**Practical questions:**

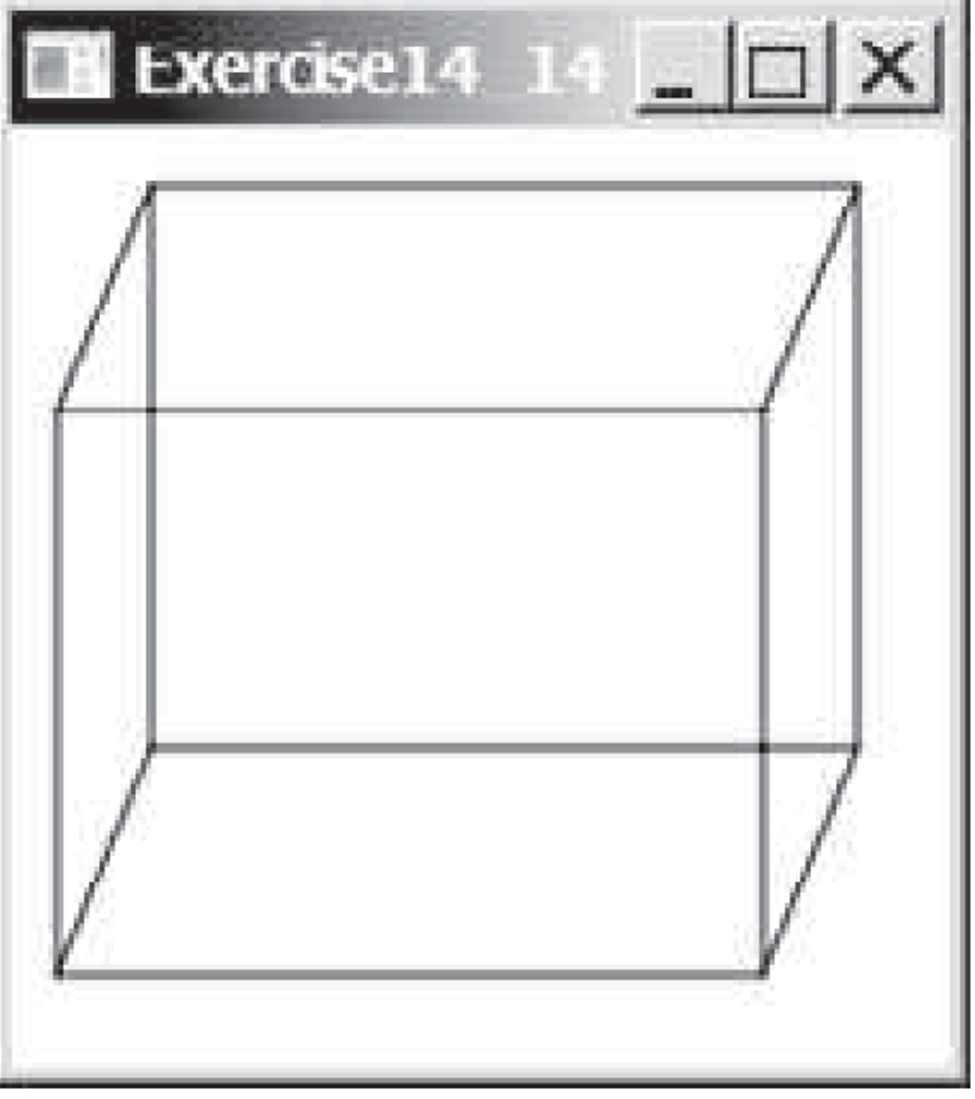
1. Write a program that displays five texts vertically, as shown in Figure. Set a random color and opacity for each text and set the font of each text to Times Roman, bold, italic, and 22 pixels.



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 window grows or shrinks.



**B.**

**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!");});

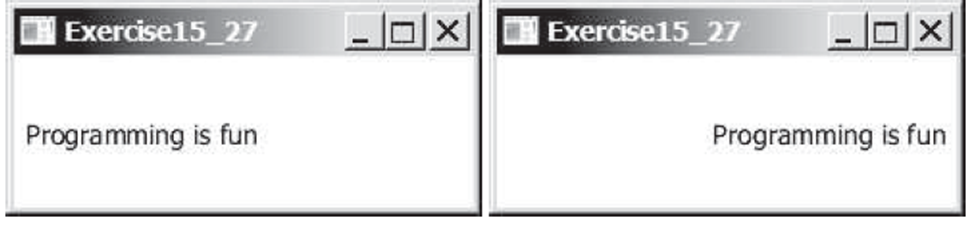
The ClickHandler class that you just eliminated had a handle() method in it. That method took a single parameter, which was an ActionEvent object, e. The code we just put in place of the original code contains a *lambda expression* mapping that parameter e to a chunk of code that will run when the event 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.



**Procedure:**

**//Write program here**

**Observations:**

**//Write program output here**

**Conclusion:** (Sufficient space to be provided)

**Quiz:** (Sufficient space to be provided for the answers)

# How does event handling work in Java, and what is the event-driven programming model?

# What is the role of the java.awt.event and javafx.event packages in Java event handling?

# Explain: MouseEvent, KeyEvent, ActionEvent

# What are the primary libraries or frameworks for creating animations in Java, and which one do you prefer?

# How to set the cycle count of an animation to infinite?

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

# What is the purpose of a TextField control, and how can it be used to collect user input?

# How to create an ImageView from an Image, or directly from a file or a URL?.

# What are the primary layout controls in JavaFX, and how do they impact the arrangement of UI components?

# What is CSS, and how is it used for styling JavaFX UI controls?

# 

# 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 | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# Signature of Faculty:

**Experiment No: 9**

**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)

**Procedure:**

**//Write program here**

**Observations:**

**//Write program output here**

**Conclusion:** (Sufficient space to be provided)

**Quiz:** (Sufficient space to be provided for the answers)

# What is recursion in Java, and how does it differ from iteration in solving problems?

# What are the advantages and disadvantages of using recursion in Java?

# What are generics in Java, and why are they used for creating parameterized types?

# How to define Generic class? What are restrictions of generic programming?

# Can you provide an example of a generic class in Java, such as a generic ArrayList?

# 

# 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 | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# Signature of Faculty:

**Experiment No: 10**

**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.

**Procedure:**

**//Write program here**

**Observations:**

**//Write program output here**

**Conclusion:** (Sufficient space to be provided)

**Quiz:** (Sufficient space to be provided for the answers)

# Write a note on ‘Collection in JAVA’. Also discuss List and Enumeration Interface.

# Differentiate between Enumeration and Iterator.

# Compare List, Set and Map interfaces. Also compare ArrayList, TreeSet and HashMap classes in java.

# Explain the unique features of Map interface.

# How do you perform common operations like sorting, searching, or filtering on Collections?

# 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 | Knowledge (2) | | Problem Recognition(2) | | Logic Building (2) | | Completeness and accuracy (2) | | Ethics (2) | | Total |
| Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg. (1) | Good (2) | Avg.(1) | Good (2) | Avg. (1) |
| Marks |  |  |  |  |  |  |  |  |  |  |  |

# Signature of Faculty: