|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Object Oriented Programming** | **Semester:** | **III** |
| **Date of Performance:** | **18 / 08 / 2023** | **Batch No:** | **A - 3** |
| **Faculty Name:** | **Pragya Gupta** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

|  |  |  |
| --- | --- | --- |
| **Writing Program (07)** | **Performance in lab**  **and Viva (05 + 03)** | **Post lab questions, conclusion and completion (03 + 02 + 05)** |
|  |  |  |

**Experiment No: 4**

**Title: Concept of Inheritance and Abstract class**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Learn the concept of Inheritance, method overriding and Abstract class in Java |

|  |
| --- |
| **COs to be achieved:** |
| **CO2:** Understand concepts of Object-Oriented Programming and basic characteristics of Java. |

|  |
| --- |
| **Tools used:** |
| 1. Java Development Kit (JDK) 2. Visual Studio Code 3. [Inheritance in Java - GeeksforGeeks](https://www.geeksforgeeks.org/inheritance-in-java/) 4. [Java Method Overriding (programiz.com)](https://www.programiz.com/java-programming/method-overriding) |

|  |
| --- |
| **Theory:** |
| **(About Types of Inheritance and abstract class)**  **Inheritance:**  In Java, inheritance is a fundamental object-oriented programming concept that allows a new class (subclass or derived class) to inherit properties and behaviors (fields and methods) from an existing class (superclass or base class). There are several types of inheritance in Java:   1. **Single Inheritance:** A subclass inherits from a single superclass. Java supports single inheritance where a class can only extend one superclass. 2. **Multilevel Inheritance:** A class can inherit from another class, which in turn inherits from another class. This creates a chain of inheritance. 3. **Hierarchical Inheritance:** Multiple subclasses inherit from a single superclass. This leads to a hierarchical structure where a single superclass has multiple subclasses. 4. **Multiple Inheritance (via Interfaces):** Java does not support multiple inheritance of classes directly to avoid ambiguity. However, multiple inheritance can be achieved through interfaces. A class can implement multiple interfaces, allowing it to inherit method signatures from multiple sources. 5. **Hybrid Inheritance:** A combination of any of the above types of inheritance.   Let us try to understand with the example of class Animal and class Cat. In this case, Animal class can be considered as a parent class, whereas class Cat can be considered as a child class. A class in Java is the blueprint for creating objects. An object has properties and states. Properties are denoted using methods, while states are denoted using variables. When we design classes for our java program, we can design the parent class methods and states.  When we design our Cat class, we need not rewrite all the code for the cat class from scratch and can reuse the code from the animal class. The cat class can be designed to add further some more properties and can reuse the properties of the animal class. Here is the sample code for inheritance:  private String sound;  private String sleep;  private String legs;  private String diet; // Animal attribute  public String getSound() { // Animal method  //implementation of getSound() method here  }  public String getSleep() {  //implementation of getSleep() method here  }  public String getLegs() {  //implementation of getLegs() method here  }  public String getDiet() {  //implementation of getDiet() method here  }  }  class Cat extends Animal { //Cat class inherits methods of Animal  private String type; // Cat attribute  private String size;  private String breed;  public String getType() {  //implementation of getType() method here  }  public String getSize() {  //implementation of getSize() method here  }  public String getBreed() {  //implementation of getBreed() method here  }  public static void main(String[] args) {  Cat myCat= new Cat();  myCat.getSleep();  myCat.getLegs();  myCat.getBreed();  }  }  **Abstraction Class:**  An abstract class in Java is a class that cannot be instantiated on its own but serves as a blueprint for other classes. It can contain both concrete methods (methods with implementations) and abstract methods (methods without implementations). Abstract classes are often used to define common behaviors and characteristics that should be shared among subclasses.  Here are some key points about abstract classes in Java:   1. **Declaring an Abstract Class:**   To declare an abstract class, you use the abstract keyword before the class keyword in the class declaration. For example:  abstract class Shape {  // Fields, constructors, and methods can be defined here  }   1. **Abstract Methods:**   Abstract classes can include abstract methods, which are declared without implementations. Subclasses of an abstract class are required to provide concrete implementations for all the abstract methods. Abstract methods are declared using the abstract keyword and do not have a body. For example:  abstract class Shape {  abstract double area();  }   1. **Subclassing Abstract Classes:**   When you create a subclass of an abstract class, you must provide implementations for all the abstract methods declared in the abstract class. If any abstract method is left unimplemented in the subclass, that subclass itself becomes abstract and cannot be instantiated.  class Circle extends Shape {  double radius;  Circle(double radius) {  this.radius = radius;  }  @Override  double area() {  return Math.PI \* radius \* radius;  }  }   1. **Final and Abstract:** A class cannot be both abstract and final. The final keyword prevents a class from being subclassed, but an abstract class is meant to be subclassed. Therefore, these two keywords are mutually exclusive.   Abstract classes are a fundamental concept in object-oriented programming that enables you to create a hierarchy of related classes with shared behavior while allowing subclasses to specialize and provide their own implementations. |

|  |
| --- |
| **Code:** |
| 1. Create a class named *'Rectangle'* with two data members *'length'* and *'breadth'* and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class *'Square'* inherit the *'Rectangle'* class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square.   import java.util.\*;  class Rectangle  {      private double length;      private double breadth;      public Rectangle(double length, double breadth)      {          this.length = length;          this.breadth = breadth;      }      public double calculateArea()      {          return length \* breadth;      }      public double calculatePerimeter()      {          return 2 \* (length + breadth);      }  }  class Square extends Rectangle  {      public Square(double side)      {          super(side, side);      }  }  public class exp4\_q1  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("\n\nenter length of rectangle: ");          double rectLength = sc.nextDouble();          System.out.print("enter breadth of rectangle: ");          double rectBreadth = sc.nextDouble();          Rectangle rectangle = new Rectangle(rectLength, rectBreadth);          System.out.println("\nrectangle area: " + rectangle.calculateArea());          System.out.println("rectangle perimeter: " + rectangle.calculatePerimeter());          System.out.print("\nenter side of square: ");          double squareSide = sc.nextDouble();          Square square = new Square(squareSide);          System.out.println("\nsquare area: " + square.calculateArea());          System.out.println("square perimeter: " + square.calculatePerimeter());          sc.close();      }  }   1. Now repeat the above example to print the area of 10 squares. (Hint - use array of objects)   import java.util.\*;  class Rectangle  {      private double length;      private double breadth;      public Rectangle(double length, double breadth)      {          this.length = length;          this.breadth = breadth;      }      public double calculateArea()      {          return length \* breadth;      }      public double calculatePerimeter()      {          return 2 \* (length + breadth);      }  }  class Square extends Rectangle  {      public Square(double side)      {          super(side, side);      }  }  public class exp4\_q2  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          System.out.print("enter length of rectangle: ");          double rectLength = sc.nextDouble();          System.out.print("enter breadth of rectangle: ");          double rectBreadth = sc.nextDouble();          Rectangle rectangle = new Rectangle(rectLength, rectBreadth);          System.out.println("\nrectangle area: " + rectangle.calculateArea());          System.out.println("rectangle perimeter: " + rectangle.calculatePerimeter());          // creating array of 10 Square objects          Square[] squares = new Square[10];          for (int i = 0; i < squares.length; i++)          {              System.out.print("\nenter side of square " + (i + 1) + ": ");              double squareSide = sc.nextDouble();              squares[i] = new Square(squareSide);              System.out.println("area of square " + (i + 1) + ": " + squares[i].calculateArea());          }          sc.close();      }  }   1. Write a java program to create an abstract class named Shape that contains two integers and an empty method named print Area (). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contains only the method print Area () that prints the area of the given shape.   import java.util.\*;  abstract class Shape  {      protected double dimension1;      protected double dimension2;      public Shape(double dimension1, double dimension2)      {          this.dimension1 = dimension1;          this.dimension2 = dimension2;      }      public abstract double calculateArea();      public void printArea()      {          System.out.println("area: " + calculateArea());      }  }  class Rectangle extends Shape  {      public Rectangle(double length, double breadth)      {          super(length, breadth);      }      @Override      public double calculateArea()      {          return dimension1 \* dimension2;      }  }  class Triangle extends Shape  {      public Triangle(double base, double height)      {          super(base, height);      }      @Override      public double calculateArea()      {          return 0.5 \* dimension1 \* dimension2;      }  }  class Circle extends Shape  {      public Circle(double radius)      {          super(radius, 0);      }      @Override      public double calculateArea()      {          return Math.PI \* dimension1 \* dimension1;      }  }  public class exp4\_q3  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);            System.out.print("\n\nenter length of rectangle: ");          double rectLength = sc.nextDouble();          System.out.print("enter breadth of rectangle: ");          double rectBreadth = sc.nextDouble();          Rectangle rectangle = new Rectangle(rectLength, rectBreadth);          System.out.print("rectangle ");          rectangle.printArea();          System.out.print("\nenter base of triangle: ");          double base = sc.nextDouble();          System.out.print("enter height of triangle: ");          double height = sc.nextDouble();          Triangle triangle = new Triangle(base, height);          System.out.print("triangle ");          triangle.printArea();          System.out.print("\nenter radius of circle: ");          double radius = sc.nextDouble();          Circle circle = new Circle(radius);          System.out.print("circle ");          circle.printArea();          sc.close();      }  } |

|  |
| --- |
| **Output:** |
|  |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **Explain *super* keyword with all its usages. Support explanation with a program.**   The super keyword in Java is used to refer to the immediate parent class of a subclass. It is primarily used to access members (fields and methods) of the superclass from the subclass. The super keyword is often used when there is a need to differentiate between the members of the subclass and those of the superclass that have the same names. The super keyword has several usages:   * 1. **Accessing Superclass Members:** You can use the super keyword to access fields and methods of the superclass from within a subclass. This is helpful when a subclass overrides a method from the superclass and wants to call the overridden method of the superclass.   2. **Invoking Superclass Constructor:** When a subclass is instantiated, its constructor can call the constructor of the superclass using the super keyword. This is useful when you want to perform the initialization defined in the superclass before performing subclass-specific initialization.   Let's illustrate these usages with an example:  class Vehicle {  protected String brand;  public Vehicle(String brand) {  this.brand = brand;  }  public void displayBrand() {  System.out.println("Brand: " + brand);  }  }  class Car extends Vehicle {  private int year;  public Car(String brand, int year) {  super(brand); // Call to the superclass constructor  this.year = year;  }  public void displayYear() {  System.out.println("Year: " + year);  }  // Override the displayBrand method  @Override  public void displayBrand() {  super.displayBrand(); // Call the superclass method using super  System.out.println("Car Brand: " + brand); // Access subclass field  }  }  public class SuperKeywordExample {  public static void main(String[] args) {  Car car = new Car("Toyota", 2023);  car.displayYear();  car.displayBrand();  }  }  In this example, the Vehicle class is the superclass, and the Car class is the subclass. The Car class uses the super keyword to call the superclass constructor and access the overridden method of the superclass. It also demonstrates accessing a field of the subclass and the superclass with the same name using the super keyword to differentiate between them.  When you run the program, you'll see the following output:  Year: 2023  Brand: Toyota  Car Brand: Toyota  This demonstrates how the ‘super’ keyword can be used to interact with both superclass and subclass members effectively.   1. **Perform following virtual lab experiments.**    1. <https://java-iitd.vlabs.ac.in/exp/inheritance/>      * 1. <https://java-iitd.vlabs.ac.in/exp/method-overloading/>      * 1. <https://java-iitd.vlabs.ac.in/exp/method-overriding/> |

|  |
| --- |
| **Conclusion:** |
| In conclusion, the experiment successfully introduced and demonstrated fundamental concepts of object-oriented programming in Java, including inheritance, method overriding, and the role of abstract classes. Through practical implementation, a clear understanding of code reusability, polymorphism, and creating class hierarchies was achieved. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |