**FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG**

**Department of Computer Science and Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2025-2026** | **Estimated Time** | **Experiment No. 3 – 02 Hours** |
| **Course & Semester** | **S.E. CSE** | **Subject Name** | **Object Oriented**  **Programming with Java Lab** |
| **Module No.** | **03** | **Chapter Title** | Inheritance |
| **Experiment Type** | **Software Performance** | **Subject Code** | 25PCC12CS07 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of Student** | Atharva Dharmendra Jagtap | **Roll No.** | 10937 |
| **Date of Performance.:** |  | **Date of Submission.:** |  |
| **CO Mapping** | **CO2. Apply Object-Oriented Programming Principles to given problem.** | | |

**Objective of Experiment:**

To explore and demonstrate the concepts of Inheritance in Object-Oriented Programming (OOP),

**Pre-Requisite:** Any programming language like C, C++

**Tools:** Java IDLE

**Theory:**

**A. Inheritance**

Inheritance in Java is a fundamental concept of Object-Oriented Programming (OOP) that allows one class (called the subclass or child class) to inherit the fields and methods of another class (called the superclass or parent class). This promotes code reusability and establishes a natural hierarchy between classes.

// Superclass

class Animal {

void eat() {

System.out.println("This animal eats food.");

}

}

// Subclass

class Dog extends Animal {

void bark() {

System.out.println("The dog barks.");

}

// Overriding the eat method

@Override

void eat() {

System.out.println("The dog eats meat.");

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.eat(); // Calls the overridden eat method in Dog class

myDog.bark(); // Calls the bark method in Dog class

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.eat(); // Calls the overridden eat method in Dog class

myDog.bark(); // Calls the bark method in Dog class

}

}

// Fields

String model;

String color;

int year;

// Methods

void startEngine() {

System.out.println("Engine started.");

}

}

* **Creating Objects:** Objects are instances of a class.

**B. Interface**

An **interface** in Java is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Unlike a class, an interface cannot contain instance fields or constructors. Interfaces provide a way to achieve abstraction and multiple inheritance in Java.

interface Animal {

void sound();

}

class Dog implements Animal {

public void sound() {

System.out.println("Barks");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.sound(); // Output: Barks

}

}

// Fields

String model;

String color;

int year;

// Methods

void startEngine() {

System.out.println("Engine started.");

}

}

* **Creating Objects:** Objects are instances of a.

**Problem Description:**

**C. Abstract Class**

In Java, an abstract class is a class that cannot be instantiated on its own and is meant to be extended by other classes. An abstract class can contain both abstract methods (which have no body) and concrete methods (with a body). Abstract methods define a contract that subclasses must fulfill.

abstract class Animal {

// Abstract method (no implementation)

abstract void sound();

// Concrete method

void sleep() {

System.out.println("This animal is sleeping.");

}

}

class Dog extends Animal {

// Providing implementation for the abstract method

@Override

void sound() {

System.out.println("The dog barks.");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.sound(); // Output: The dog barks.

dog.sleep(); // Output: This animal is sleeping.

}

}

**D. Super keyword**

The super keyword in Java is a reference variable used to refer to the immediate parent class object. It is primarily used in inheritance to access members (methods, fields, and constructors) of the parent class from the child class.

**Accessing Parent Class Methods using super:**

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

@Override

void sound() {

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

}

void makeSound() {

super.sound(); // Calls the sound() method from Animal class

sound(); // Calls the overridden sound() method from Dog class

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.makeSound(); // Output: Animal makes a sound, Dog barks

}

}

**Post Lab Questions:**

**Accessing Parent Class Fields:**

class Animal {

String color = "Brown";

}

class Dog extends Animal {

String color = "Black";

void printColor() {

System.out.println("Dog color: " + color); // Black

System.out.println("Animal color: " + super.color); // Brown

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.printColor();

}

}

**Calling Parent Class Constructors:**

class Animal {

Animal() {

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

}

}

class Dog extends Animal {

Dog() {

super(); // Calls the constructor of the Animal class

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

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

// Output:

// Animal constructor called

// Dog constructor called

}

}

**E. Final keyword**

The final keyword in Java is used to apply restrictions on classes, methods, and variables. Once applied, the final keyword ensures that these elements cannot be modified in certain ways, enhancing security and predictability in your code.

**Problem Statement**

Create two interfaces ‘SimpleCalculator’ and ‘SciCalculator’ as shown in the above figure. Write a class ‘Calculator’ that implemets both the interfaces. Create object of class ‘Calculator’ and call the methods add(), sub(), factorial() and square() and display the result of each method.

**Code:**

// SimpleCalculator.java

interface SimpleCalculator {

    // Method for addition

    double add(double num1, double num2);

    // Method for subtraction

    double sub(double num1, double num2);

}

// SciCalculator.java

interface SciCalculator {

    // Method to calculate factorial (for non-negative integers)

    long factorial(int n);

    // Method to calculate the square of a number

    double square(double num);

}

// Calculator.java

class Calculator implements SimpleCalculator, SciCalculator {

    @Override

    public double add(double num1, double num2) {

        return num1 + num2;

    }

    @Override

    public double sub(double num1, double num2) {

        return num1 - num2;

    }

    @Override

    public long factorial(int n) {

        if (n < 0) {

            throw new IllegalArgumentException("Factorial is not defined for negative numbers.");

        }

        if (n == 0 || n == 1) {

            return 1;

        }

        long result = 1;

        for (int i = 2; i <= n; i++) {

            result \*= i;

        }

        return result;

    }

    @Override

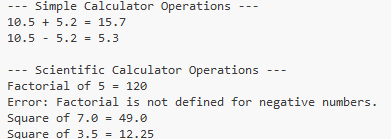
    public double square(double num) {

        return num \* num;

    }

}

**Output:**



**Post Lab Questions:**

|  |
| --- |
| **Q1. Create a Program to Demonstrate all types of Inheritance in Java and include the use of `super` keyword.** |
|  |
|  |

**Q1. Ans**

**Code:**

// Base class (Parent)

class Animal {

    String name;

    Animal(String name) {

        this.name = name;

        System.out.println("Animal Constructor called for: " + name);

    }

    void sound() {

        System.out.println(name + " makes a sound.");

    }

}

// ------------------- SINGLE INHERITANCE -------------------

class Dog extends Animal {

    Dog(String name) {

        super(name); // Call parent constructor

    }

    @Override

    void sound() {

        super.sound(); // Call parent method

        System.out.println(name + " barks.");

    }

}

// ------------------- MULTILEVEL INHERITANCE -------------------

class Puppy extends Dog {

    Puppy(String name) {

        super(name);

    }

    void behavior() {

        System.out.println(name + " is a cute puppy!");

    }

}

// ------------------- HIERARCHICAL INHERITANCE -------------------

class Cat extends Animal {

    Cat(String name) {

        super(name);

    }

    @Override

    void sound() {

        super.sound(); // Call parent method

        System.out.println(name + " meows.");

    }

}

class Cow extends Animal {

    Cow(String name) {

        super(name);

    }

    @Override

    void sound() {

        super.sound();

        System.out.println(name + " moos.");

    }

}

// ------------------- HYBRID INHERITANCE -------------------

// Hybrid = Combination of multiple inheritance styles (Java doesn’t support multiple inheritance with classes, but it can be achieved with interfaces)

// Interface for hybrid inheritance

interface Pet {

    void play();

}

// Dog is already extending Animal, now also implementing Pet

class PetDog extends Dog implements Pet {

    PetDog(String name) {

        super(name);

    }

    public void play() {

        System.out.println(name + " loves to play fetch!");

    }

}

// ------------------- MAIN DRIVER -------------------

public class InheritanceDemo {

    public static void main(String[] args) {

        System.out.println("\n--- Single Inheritance ---");

        Dog dog = new Dog("Buddy");

        dog.sound();

        System.out.println("\n--- Multilevel Inheritance ---");

        Puppy puppy = new Puppy("Max");

        puppy.sound();

        puppy.behavior();

        System.out.println("\n--- Hierarchical Inheritance ---");

        Cat cat = new Cat("Kitty");

        cat.sound();

        Cow cow = new Cow("Bessie");

        cow.sound();

        System.out.println("\n--- Hybrid Inheritance (with Interface) ---");

        PetDog petDog = new PetDog("Charlie");

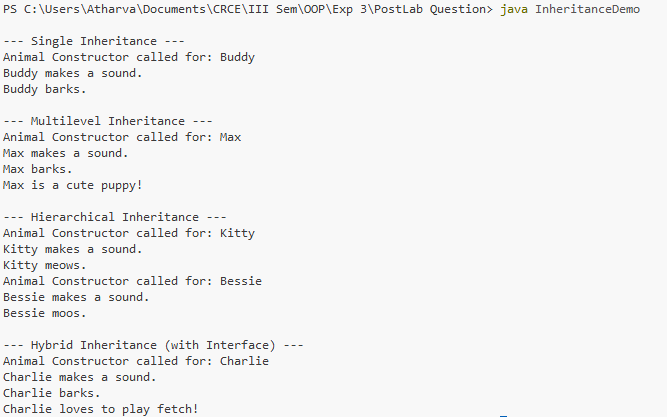
        petDog.sound();

        petDog.play();

    }

}

**Output:**



|  |  |  |  |
| --- | --- | --- | --- |
| **On time Completion and Submission (2)** | **Knowledge of the topic (4)** | **Implementation and Output (4)** | **Total (10)** |
|  |  |  |  |

|  |  |
| --- | --- |
| **References:Study Materials**  [**https://www.w3schools.com/java/**](https://www.w3schools.com/java/)  [**https://www.geeksforgeeks.org/java/**](https://www.geeksforgeeks.org/java/)  https://www.codecademy.com/learn/learn-java | **Video Channels**:  [**https://www.youtube.com/user/programmingwithmosh**](https://www.youtube.com/user/programmingwithmosh)  [**https://www.youtube.com/c/TheNetNinja**](https://www.youtube.com/c/TheNetNinja)  [**https://www.youtube.com/c/Freecodecamp**](https://www.youtube.com/c/Freecodecamp)  [**https://www.youtube.com/user/Simplilearn**](https://www.youtube.com/user/Simplilearn) |
| **Study Materials used for Demo**  <Add links here> | |

**Note:-students are expected to paste screenshot of the program with output**