

## INHERITANCE

### Aim:

To understand and implement inheritance concepts in Java.

### PRE LAB EXERCISE

#### QUESTIONS

- ✓ What is inheritance?

Inheritance is a feature of Java in which one class acquires the properties and behaviors (fields and methods) of another class.

- The existing class is called **Parent (or Super) class**
- The new class is called **Child (or Sub) class**

- ✓ What is code reusability?

Code reusability means using existing code again instead of writing it repeatedly.

In Java, this is mainly achieved using **inheritance**, methods, and classes.

It reduces repetition and makes programs easier to maintain.

- ✓ What is the use of extends keyword?

The extends keyword is used to **inherit** one class from another in Java.

It allows a child class to use the properties and methods of the parent class.

### IN LAB EXERCISE

#### Objective:

To implement all types of inheritance.

## **PROGRAMS:**

### **Student Result System (Single Inheritance)**

#### **Question:**

A school wants to store student details and calculate marks. Create a base class Student and a derived class Result.

#### **Code:**

```
class Student {  
    String name;  
    int rollNo;  
  
    void getDetails() {  
        name = "Ruddhida";  
        rollNo = 230;  
    }  
}  
  
class Result extends Student {  
    int marks = 85;  
  
    void display() {  
        System.out.println("Name: " + name);  
        System.out.println("Roll No: " + rollNo);  
        System.out.println("Marks: " + marks);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Result r = new Result();  
    }  
}
```

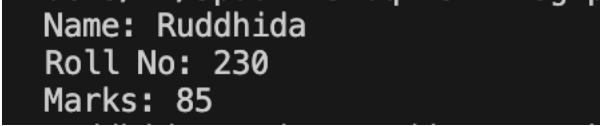
```
        r.getDetails();  
        r.display();  
    }  
}
```

**Output:**

Name: Ruddhida

Roll No: 230

Marks: 85

A screenshot of a terminal window showing the output of a Java program. The text is white on a black background. It displays three lines: "Name: Ruddhida", "Roll No: 230", and "Marks: 85".

```
Name: Ruddhida  
Roll No: 230  
Marks: 85
```

## 2. Bank Account System (Hierarchical Inheritance)

**Question:**

A bank has Savings and Current accounts. Both inherit from a common Account class.

**Code:**

```
class Account {  
    void showAccountType() {  
        System.out.println("Bank Account");  
    }  
}  
  
class SavingsAccount extends Account {  
    void interest() {  
        System.out.println("Savings Account gives interest");  
    }  
}
```

```

    }
}

class CurrentAccount extends Account {
    void overdraft() {
        System.out.println("Current Account supports overdraft");
    }
}

public class Main {
    public static void main(String[] args) {
        SavingsAccount s = new SavingsAccount();
        CurrentAccount c = new CurrentAccount();

        s.showAccountType();
        s.interest();

        c.showAccountType();
        c.overdraft();
    }
}

```

### **Output:**

Bank Account

Savings Account gives interest

Bank Account

Current Account supports overdraft

```
Bank Account
Savings Account gives interest
Bank Account
Current Account supports overdraft
```

### 3. Vehicle System (Multilevel Inheritance)

#### Question:

A company classifies vehicles as Vehicle → Car → ElectricCar.

#### Code:

```
class Vehicle {
    void start() {
        System.out.println("Vehicle starts");
    }
}

class Car extends Vehicle {
    void fuelType() {
        System.out.println("Car uses petrol");
    }
}

class ElectricCar extends Car {
    void battery() {
        System.out.println("Electric car uses battery");
    }
}

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

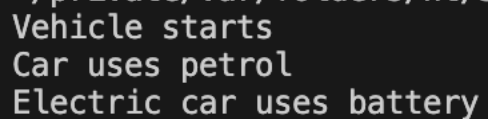
```
ElectricCar e = new ElectricCar();  
e.start();  
e.fuelType();  
e.battery();  
}  
}
```

**Output:**

Vehicle starts

Car uses petrol

Electric car uses battery



```
Vehicle starts  
Car uses petrol  
Electric car uses battery
```

## POST LAB EXERCISE

- ✓ Why Java does not support multiple inheritance using classes and how it is implemented?

Java does not support multiple inheritance using classes to avoid ambiguity, which is known as the **Diamond Problem**. In multiple inheritance, if a class inherits from two parent classes that have the same method, the compiler gets confused about which method to call, leading to ambiguity.

To avoid this problem, Java does not allow multiple inheritance with classes. Instead, Java implements multiple inheritance using **interfaces**. A class can implement multiple interfaces, allowing it to inherit behavior from more than one source without ambiguity.

- ✓ What is the role of the super keyword? Give examples.

The super keyword in Java is used to refer to the immediate parent class. It is mainly used for three purposes:

1. To call the parent class constructor.
2. To call the parent class method.
3. To access the parent class variable.

Example:

If a child class wants to use a method or variable of its parent class, it can use super. Also, if the parent class constructor needs to be called, super() is used inside the child class constructor.

- ✓ Can a child class access private members of the parent class? Why?

No, a child class cannot access private members of the parent class. This is because private members are accessible only within the same class in which they are declared. They are not visible to any other class, including subclasses.

If a child class needs to access a private member, the parent class should provide a public or protected getter method to allow controlled access.

- ✓ Explain why hybrid inheritance is not supported in Java.

Hybrid inheritance is not supported in Java through classes because it can lead to **ambiguity and the Diamond Problem**. Hybrid inheritance is a combination of more than one type of inheritance, usually involving multiple inheritance.

Since Java does not allow multiple inheritance with classes, any inheritance model that indirectly creates multiple inheritance (such as hybrid inheritance) is also not allowed. This is because if a class inherits the same method from two parent classes, the compiler will be confused about which method to call, leading to ambiguity.

To avoid this problem, Java restricts multiple and hybrid inheritance with classes. However, Java supports hybrid inheritance using **interfaces**, because interfaces do not contain method implementation (until Java 8), so ambiguity does not occur.

### Result:

Thus the different types of inheritance were implemented and executed successfully.

### ASSESSMENT

Description	Max Marks	Marks Awarded
Pre Lab Exercise	5	
In Lab Exercise	10	

Post Lab Exercise	5	
Viva	10	
<b>Total</b>	<b>30</b>	
<b>Faculty Signature</b>		