Experiment No. 3

Design and develop the programs for different types of inheritance.

Instructions:

This manual consists of three parts: **A) Theory and Concepts, B) Problems for Implementation, and C) Write-up Questions.**

1. Students must understand the **theory and concepts** provided before implementing the problem statement(s) for **Experiment 3**.
2. They should **practice the given code snippets** within the theory section.
3. Later, they need to **implement the problems provided**.
4. **Write-up:** Students are required to **write answers** to the questions on journal pages, **maintain a file**, and get it checked regularly. The file should include index, write-up, and implementation code with results.
5. Referencing: Include proper sources or references for the content used.
6. Use of Generative AI: Clearly mention if you have used any AI tools (e.g., ChatGPT, Copilot, Bard) to generate text, explanations, or code. Cite the AI-generated content appropriately in the write-up.

Part A. Theory and Concepts:

Inheritance in Java allows one class to inherit the properties and behaviors of another class. This means you can create new classes based on existing ones, reusing their methods and fields while adding new ones. Inheritance represents an **IS-A relationship**, also known as a parent-child relationship. **For example:**

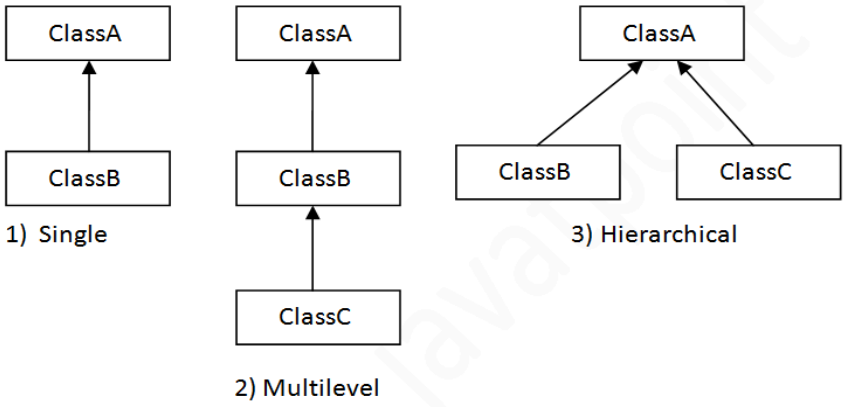
- A **Car** IS-A **Vehicle** because it inherits features from the **Vehicle** class.
- A **Dog** IS-A **Animal** because it inherits characteristics from the **Animal** class.

Types of Inheritance in Java

Java supports the following types of inheritance based on class hierarchy:

1. **Single-Level Inheritance** – A subclass inherits from a single superclass.
2. **Multilevel Inheritance** – A subclass inherits from another subclass, forming a chain.
3. **Hierarchical Inheritance** – Multiple subclasses inherit from the same superclass.

Note: Java does **not** support **Multiple Inheritance** (a class inheriting from multiple classes) and **Hybrid Inheritance** (a mix of multiple types) directly. However, **interfaces** can be used to achieve them.




```
class subclass-name extends superclass-name
{
     //Body of the subclass
}
```

Method Overriding

Method overriding occurs when a subclass provides a **new implementation** for a method that already exists in its superclass.

- The method in the subclass **must have the same name and signature** as the method in the superclass.
- When an overridden method is called, the subclass version is executed instead of the superclass version.

(1.1) Single Inheritance

Example of Single Inheritance

```
// Example 1:

class Base
{
    public void Method1()
    {
        System.out.println("Base Class Method");
    }
}
```

```
class Derived extends Base
{
    public void Method2()
    {
        System.out.println("Derived Class Method");
    }
}

class Test
{
    public static void main(String[] args)
    {
        Derived d = new Derived(); // creating object
        d.Method1(); // prints Base Class Method
        d.Method2(); // prints Derived Class Method
    }
}
```

Commands:

```
javac Base.java
java Test
```

Output:

```
Base Class Method
Derived Class Method
```

```
// Example 2: Let's take an example of a Parent-Child relationship
in programming:

class Animal1 {
    public void makeSound() {
        System.out.println("Animals make sounds");
    }
}

class Dog extends Animal1 {
    public void bark() {
        System.out.println("Dog barks");
    }
}

class Test1 {
    public static void main(String[] args) {
        Dog myDog = new Dog();  // Creating an object of subclass
        myDog.makeSound(); // Inherited method (prints: Animals make
sounds)
        myDog.bark();  // Own method (prints: Dog barks)
    }
}
```

Commands:

```
javac Animal.java
java Test
```

```
Output:
Animals make sounds
Dog barks
```

Real-World Example: Think of a Bank Account system.

- The Base class is `BankAccount` which has general properties like `balance` and methods like `deposit()` and `withdraw()`.
- The Child class `SavingsAccount` extends `BankAccount` and adds a method `addInterest()`.

Abstraction in Java

Abstraction means **hiding unnecessary details** and showing only the essential features of an object. For example:

- When you use an **ATM machine**, you just insert your card and enter a PIN. You don't see the internal working of the machine.
- In Java, abstraction is achieved using **Abstract Classes** and **Interfaces**.

Abstract Class

An **abstract class** is a class that contains **at least one abstract method** (a method without a body).

- It can have **both abstract and concrete (normal) methods**.
- It **cannot be instantiated** (you cannot create an object of an abstract class directly).
- It is used to define a blueprint for other classes.

Example of Abstract Class

```
abstract class Vehicle {
    abstract void start(); // Abstract method (no body)

    public void stop() {
        System.out.println("Vehicle stopped");
    }
}

class Car extends Vehicle {
    void start() {
        System.out.println("Car starts with a key");
```

```
    }

    public static void main(String args[]) {
        Car myCar = new Car();
        myCar.start();  // Calls overridden method
        myCar.stop();   // Calls concrete method from abstract class
    }
}
```

Commands:

```
javac Vehicle.java
java Car
```

Output:

```
Car starts with a key
Vehicle stopped
```

Real-World Example:

- **Shape class** (Abstract) → Can have methods like `calculateArea()`.
- **Circle and Rectangle** (Concrete subclasses) → Implement the `calculateArea()` method in their own way.

Rules of Abstract Classes and Methods

- An **abstract method** has no body and **must be implemented** by the subclass.
- If a class has an abstract method, it **must be declared as abstract**.
- An abstract class **can have constructors and static methods**.
- An abstract class **can have final methods** (which cannot be overridden).

Example of Abstract Method

```
abstract class Animal {
    abstract void makeSound();  // Abstract method
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog barks");
    }

    public static void main(String args[]) {
        Animal myPet = new Dog();
        myPet.makeSound();  // Prints: Dog barks
    }
}
```

```
Commands:
javac Animal.java
java Dog

Output:
Dog barks
```

(1.2) Multilevel Inheritance

Example of Multilevel Inheritance

Definition: A *child* class inherits from a *parent* class, and *another* class further inherits from that *child* class.

Example: *Puppy* inherits from *Dog*, which inherits from *Animal*.

```
// Example of Multilevel Inheritance

class Animal {
    void eat() {
        System.out.println("Animals eat food");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks");
    }
}

class Puppy extends Dog { // Multilevel inheritance
    void weep() {
        System.out.println("Puppy weeps");
    }
}

class Test {
    public static void main(String[] args) {
        Puppy myPuppy = new Puppy();
        myPuppy.eat(); // Inherited from Animal
        myPuppy.bark(); // Inherited from Dog
        myPuppy.weep(); // Own method
    }
}
```

```
Commands:
javac Animal.java
java Test

Output:
Animals eat food
Dog barks
```

```
Puppy weeps
```

(1.3) Hierarchical Inheritance

Example of Hierarchical Inheritance

Definition: *Multiple child classes* inherit from the same *parent* class.

Example: *Dog* and *Cat* both inherit from *Animal*.

```
// Example of Hierarchical Inheritance

class Animal {
    void makeSound() {
        System.out.println("Animals make sounds");
    }
}

class Dog extends Animal { // Hierarchical inheritance
    void bark() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal { // Hierarchical inheritance
    void meow() {
        System.out.println("Cat meows");
    }
}

class Test {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound(); // Inherited method
        myDog.bark(); // Own method

        Cat myCat = new Cat();
        myCat.makeSound(); // Inherited method
        myCat.meow(); // Own method
    }
}
```

Commands:

```
javac Animal.java
java Test
```

Output:

```
Animals make sounds
Dog barks
Animals make sounds
Cat meows
```

(1.4) Multiple Inheritance (Through Interfaces)

Example of Multiple Inheritance (Through Interfaces)

Definition: Java **does not support multiple inheritance** using classes to avoid ambiguity.

However, we can achieve it using **interfaces.**

Example: A *Smartphone* can inherit properties from both *Camera* and *Phone*.

```
// Example of Multiple Inheritance (Through Interfaces)

interface Camera {
    void clickPhoto();
}

interface Phone {
    void makeCall();
}

class Smartphone implements Camera, Phone { // Multiple inheritance
using interfaces
    public void clickPhoto() {
        System.out.println("Photo clicked");
    }

    public void makeCall() {
        System.out.println("Calling...");
    }
}

class Test {
    public static void main(String[] args) {
        Smartphone myPhone = new Smartphone();
        myPhone.clickPhoto();
        myPhone.makeCall();
    }
}
```

Commands:

```
javac Smartphone.java
java Test
```

Output:

```
Photo clicked
Calling...
```

(1.5) Hybrid Inheritance (Through Interfaces)

Example of Hybrid Inheritance (Through Interfaces)

Definition: A combination of multiple and other types of inheritance. Since Java **does not support multiple inheritance** with classes, hybrid inheritance is achieved using **interfaces**.

Example: A *HybridCar* inherits from both *Car* and *ElectricVehicle* using interfaces.

```
// Example of Hybrid Inheritance (Through Interfaces)

interface Car {
    void drive();
}

interface ElectricVehicle {
    void chargeBattery();
}

class HybridCar implements Car, ElectricVehicle { // Hybrid
inheritance using interfaces
    public void drive() {
        System.out.println("Driving hybrid car");
    }

    public void chargeBattery() {
        System.out.println("Charging battery");
    }
}

class Test {
    public static void main(String[] args) {
        HybridCar myCar = new HybridCar();
        myCar.drive();
        myCar.chargeBattery();
    }
}
```

Commands:

```
javac HybridCar.java
java Test
```

Output:

```
Driving hybrid car
Charging battery
```

Part B. Problems for Implementation:

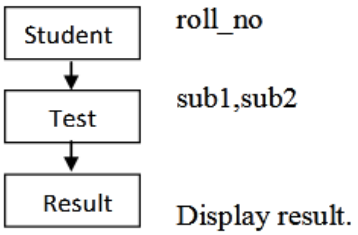
Aim: Implementation of a given problem statement/s for different types of inheritance.

1. Write a Java program to create a class known as "BankAccount" with methods called deposit() and withdraw(). Create a subclass called SavingsAccount that overrides the withdraw() method to prevent withdrawals if the account balance falls below one hundred.

2. Write a Java program that creates a class hierarchy for employees of a company. The base class should be Employee, with subclasses Manager, Developer, and Programmer. Each subclass should have properties such as name, address, salary, and job title. Implement methods for calculating bonuses, generating performance reports, and managing projects.

3. Implement Following:
    a. Create abstract class shapes with dim1, dim2 variables and abstract area() method. Class
    b. rectangle and triangle inherits shape class. Calculate area of rectangle and triangle.

4. Write a program to perform Multilevel Inheritance



Part C. Write-up Questions:

1. Explain **inheritance** in Java and its types.
2. Explain method **overloading** and method **overriding** with simple Java programs.
3. Describe the uses of the **super** keyword with an example.
4. Explain **abstract** methods and abstract classes.
5. Describe the uses of the **final** keyword.
6. Describe **garbage collection** in Java.

Conclusion:

Students should be able to write Java programs for different types of inheritance.