EXPERIMENT – 04

Objective:

To develop Java programs that demonstrate the concepts of **inheritance** and **polymorphism**.

Theory:

Inheritance:

Inheritance allows a new class (subclass) to acquire the properties and behaviors of an existing class (superclass). This promotes code reusability and establishes a hierarchical relationship between classes.

Polymorphism:

Polymorphism enables a single interface to represent different underlying forms (data types). In Java, this is primarily achieved through:

- **Method Overriding**: A subclass provides a specific implementation of a method already defined in its superclass.
- **Method Overloading**: Multiple methods in the same class share the same name but have different parameters.

Types of Inheritance in Java:

Inheritance Type	Description	Java Support
Single	A class inherits from one superclass.	Supported
Multilevel	A class inherits from a subclass, forming a chain.	Supported
Hierarchical	Multiple classes inherit from a single superclass.	Supported
Multiple	A class inherits from multiple classes.	Not directly supported
Hybrid	A combination of two or more types of inheritance.	Not directly supported

Note: Java supports multiple inheritance through interfaces.

Practical Demonstrations:

1. Single Inheritance

```
class Vehicle {
   void startEngine() {
      System.out.println("Engine started.");
```

```
    }
}
class Car extends Vehicle {
    void drive() {
        System.out.println("Car is driving.");
    }
}
 class SingleInheritanceDemo {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.startEngine();
        myCar.drive();
    }
}
```

Output:

Engine started.
Car is driving.

Explanation:

- Car inherits the startEngine() method from Vehicle.
- Demonstrates single inheritance where Car is a subclass of Vehicle.

2. Multilevel Inheritance

```
class Animal {
    void eat() {
        System.out.println("Animal eats food.");
    }
}
class Mammal extends Animal {
    void walk() {
        System.out.println("Mammal walks.");
    }
```

```
}
class Dog extends Mammal {
    void bark() {
        System.out.println("Dog barks.");
    }
}
public class MultilevelInheritanceDemo {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();
        myDog.walk();
        myDog.bark();
    }
}
```

Output:

```
Animal eats food.
Mammal walks.
Dog barks.
```

Explanation:

- Dog inherits from Mammal, which in turn inherits from Animal.
- Demonstrates multilevel inheritance.

3. Polymorphism through Method Overriding

```
class Shape {
    void draw() {
        System.out.println("Drawing a shape.");
    }
}
class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing a circle.");
```

```
    }
}
public class PolymorphismDemo {
    public static void main(String[] args) {
        Shape myShape = new Circle();
        myShape.draw();
    }
}
```

Output:

Drawing a circle.

Explanation:

- Circle overrides the draw() method of Shape.
- Demonstrates runtime polymorphism where the method call is determined at runtime.

4. Multiple Inheritance via Interfaces

```
interface Flyable {
    void fly();
}
interface Swimmable {
    void swim();
}
class Duck implements Flyable, Swimmable {
    public void fly() {
        System.out.println("Duck flies.");
    }

    public void swim() {
        System.out.println("Duck swims.");
    }
}
public class MultipleInheritanceDemo {
    public static void main(String[] args) {
        Duck daffy = new Duck();
        daffy.fly();
        daffy.swim();
    }
}
```

Output:

Duck flies.
Duck swims.

Explanation:

- Duck implements both Flyable and Swimmable interfaces.
- Demonstrates multiple inheritance using interfaces.

Comparative Table: Inheritance vs. Polymorphism

Aspect	Inheritance	Polymorphism
Definition	Mechanism to derive new classes from existing ones.	Ability of different classes to respond to the same method call in different ways.
Purpose	Code reusability and hierarchical classification.	Flexibility and dynamic method invocation.
Types	Single, Multilevel, Hierarchical, etc.	Compile-time (overloading), Runtime (overriding).
Implementation	extends keyword for classes.	Method overriding and overloading.

Conclusion:

Through these examples, we've explored how Java facilitates **inheritance** to promote code reuse and establish relationships between classes, and how **polymorphism** allows for dynamic method invocation, enhancing flexibility and scalability in object-oriented programming.