Week 6 - S6 - Core OOP - Inheritance - Practice Problem

Name:Ramesh Harisabapathi Chettiar

Date of Submission:23/09/25

QNO1→Understanding basic inheritance, constructor chaining, and super keyword usage

PROGRAM→

Program1 > VehicleProgram.java > VehicleProgram

```java
// File: Vehicle.java
class Vehicle {
    // TODO: Create protected fields for inheritance:
    protected String brand;
    protected String model;
    protected int year;
    protected String engineType;

    // TODO: Create private fields that require getter/setter access:
    private String registrationNumber;
    private boolean isRunning;

    // Getter and Setter for registrationNumber
    public String getRegistrationNumber() {
        return registrationNumber;
    }

    public void setRegistrationNumber(String registrationNumber) {
        this.registrationNumber = registrationNumber;
    }

    // Getter for isRunning (no setter, use start/stop)
    public boolean isRunning() {
        return isRunning;
    }

    // Default constructor
    Vehicle() {
        this.brand = "";
        this.model = "";
        this.engineType = "";
        this.registrationNumber = "";
        this.year = 0;
        this.isRunning = false;
        System.out.println("Vehicle default constructor called");
    }

    // Parameterized constructor
    Vehicle(String brand, String model, String engineType, int year) {
        this.brand = brand;
        this.model = model;
        this.engineType = engineType;
        this.year = year;
        this.registrationNumber = "REG" + ((int)(Math.random() * 10000));
        this.isRunning = false;
        System.out.println("Vehicle parameterized constructor called");
    }

    // Vehicle operations
    public void start() {
        isRunning = true;
        System.out.println("Vehicle started");
    }

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

    public String getVehicleInfo() {
        return "Brand: " + brand + ", Model: " + model + ", Year: " + year +
               ", Engine: " + engineType + ", Registration: " + registrationNumber +
```

```
            ", Running: " + isRunning;
    }

    public void displaySpecs() {
        System.out.println("Vehicle Specs:");
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
        System.out.println("Engine Type: " + engineType);
        System.out.println("Registration Number: " + registrationNumber);
        System.out.println("Is Running: " + isRunning);
    }
}

// Car class that extends Vehicle
class Car extends Vehicle {
    // Car-specific fields
    int numberOfDoors;
    String fuelType;
    String transmissionType;

    // Default constructor
    Car() {
        super();
        this.numberOfDoors = 4;
        this.fuelType = "Petrol";
        this.transmissionType = "Manual";
        System.out.println("Car default constructor called");
    }
```

```
    // Parameterized constructor
    Car(String brand, String model, String engineType, int year,
        int numberOfDoors, String fuelType, String transmissionType) {
        super(brand, model, engineType, year);
        this.numberOfDoors = numberOfDoors;
        this.fuelType = fuelType;
        this.transmissionType = transmissionType;
        System.out.println("Car parameterized constructor called");
    }

    // Override start()
    @Override
    public void start() {
        super.start();
        System.out.println("Car startup sequence: Checking seatbelts, mirrors, infotainment...");
    }

    // Override displaySpecs()
    @Override
    public void displaySpecs() {
        super.displaySpecs();
        System.out.println("Car Specs:");
        System.out.println("Number of Doors: " + numberOfDoors);
        System.out.println("Fuel Type: " + fuelType);
        System.out.println("Transmission Type: " + transmissionType);
    }

    // Car-specific methods
    public void openTrunk() {
        System.out.println("Trunk opened");
    }
```

```java
    // Parameterized constructor
    Car(String brand, String model, String engineType, int year,
        int numberOfDoors, String fuelType, String transmissionType) {
        super(brand, model, engineType, year);
        this.numberOfDoors = numberOfDoors;
        this.fuelType = fuelType;
        this.transmissionType = transmissionType;
        System.out.println("Car parameterized constructor called");
    }

    // Override start()
    @Override
    public void start() {
        super.start();
        System.out.println("Car startup sequence: Checking seatbelts, mirrors, infotainment...");
    }

    // Override displaySpecs()
    @Override
    public void displaySpecs() {
        super.displaySpecs();
        System.out.println("Car Specs:");
        System.out.println("Number of Doors: " + numberOfDoors);
        System.out.println("Fuel Type: " + fuelType);
        System.out.println("Transmission Type: " + transmissionType);
    }

    // Car-specific methods
    public void openTrunk() {
        System.out.println("Trunk opened");
    }
```

```java
        car1.playRadio();
        car1.stop();
        System.out.println(car1.getVehicleInfo());

        // 2. Create Car using parameterized constructor
        System.out.println("\n----Parameterized Constructor Test----");
        Car car2 = new Car("Toyota", "Camry", "Hybrid", 2022, 4, "Hybrid", "Automatic");
        car2.setRegistrationNumber("TN09AB1234");
        car2.displaySpecs();
        car2.start();
        car2.openTrunk();
        car2.playRadio();
        car2.stop();
        System.out.println(car2.getVehicleInfo());

        // Test inheritance and super keyword
        System.out.println("\n----Inheritance and Super Test----");
        System.out.println("Brand (protected): " + car2.brand);
        System.out.println("Model (protected): " + car2.model);
        car2.start(); // overridden
        car2.stop();  // inherited

        // Polymorphic behavior
        System.out.println("\n----Polymorphism Test----");
        Vehicle v = new Car("Honda", "City", "Petrol", 2021, 4, "Petrol", "Manual");
        v.start(); // Calls Car's overridden start()
        v.displaySpecs(); // Calls Car's overridden displaySpecs()
    }
}
```

OUTPUT→

```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Practise
Problems> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week
6\Practise Problems\Program1\" ; if ($?) { javac VehicleProgram.java } ; if ($?) { java VehicleProgram }
----Vehicle Default Constructor Test----
Vehicle default constructor called
Vehicle Specs:
Brand:
Model:
Year: 0
Engine Type:
Registration Number:
Is Running: false
Vehicle started
Vehicle stopped
Brand: , Model: , Year: 0, Engine: , Registration: , Running: false

----Vehicle Parameterized Constructor Test----
Vehicle parameterized constructor called
Vehicle Specs:
Brand: Suzuki
Model: Swift
Year: 2020
Engine Type: Petrol
Registration Number: TN01XY9876
Is Running: false
Vehicle started
Vehicle stopped
Brand: Suzuki, Model: Swift, Year: 2020, Engine: Petrol, Registration: TN01XY9876, Running: false
```

```
----Default Constructor Test----
Vehicle default constructor called
Car default constructor called
Vehicle Specs:
Brand:
Model:
Year: 0
Engine Type:
Registration Number:
Is Running: false
Car Specs:
Number of Doors: 4
Fuel Type: Petrol
Transmission Type: Manual
Vehicle started
Car startup sequence: Checking seatbelts, mirrors, infotainment...
Trunk opened
Radio playing music
Vehicle stopped
Brand: , Model: , Year: 0, Engine: , Registration: , Running: false
```

```
----Parameterized Constructor Test----
Vehicle parameterized constructor called
Car parameterized constructor called
Vehicle Specs:
Brand: Toyota
Model: Camry
Year: 2022
Engine Type: Hybrid
Registration Number: TN09AB1234
Is Running: false
Car Specs:
Number of Doors: 4
Fuel Type: Hybrid
Transmission Type: Automatic
Vehicle started
Car startup sequence: Checking seatbelts, mirrors, infotainment...
Trunk opened
Radio playing music
Vehicle stopped
Brand: Toyota, Model: Camry, Year: 2022, Engine: Hybrid, Registration: TN09AB1234, Running: false
```

```
----Inheritance and Super Test----
Brand (protected): Toyota
Model (protected): Camry
Vehicle started
Car startup sequence: Checking seatbelts, mirrors, infotainment...
Vehicle stopped

----Polymorphism Test----
Vehicle parameterized constructor called
Car parameterized constructor called
Vehicle started
Car startup sequence: Checking seatbelts, mirrors, infotainment...
Vehicle Specs:
Brand: Honda
Model: City
Year: 2021
Engine Type: Petrol
Registration Number: REG2762
Is Running: true
Car Specs:
Number of Doors: 4
Fuel Type: Petrol
Transmission Type: Manual
```

QNO2→Building deep inheritance hierarchies with proper constructor chaining

PROGRAM→

```java
// TODO: Create base class Animal:
class Animal {
    // TODO: Create protected fields:
    // - species (String)
    // - habitat (String)
    // - lifespan (int)
    // - isWildlife (boolean)
    protected String species;
    protected String habitat;
    protected int lifespan;
    protected boolean isWildlife;

    // TODO: Create constructor that:
    // - Takes all parameters
    // - Prints "Animal constructor: Creating [species]"
    public Animal(String species, String habitat, int lifespan, boolean isWildlife) {
        this.species = species;
        this.habitat = habitat;
        this.lifespan = lifespan;
        this.isWildlife = isWildlife;
        System.out.println("Animal constructor: Creating " + species);
    }

    // TODO: Create methods:
    // - eat() - prints "Animal is eating"
    // - sleep() - prints "Animal is sleeping"
    // - move() - prints "Animal is moving"
    // - getAnimalInfo() - returns formatted animal details
    public void eat() {
        System.out.println("Animal is eating");
    }
```

```java
    public void sleep() {
        System.out.println("Animal is sleeping");
    }

    public void move() {
        System.out.println("Animal is moving");
    }

    public String getAnimalInfo() {
        return "Species: " + species + ", Habitat: " + habitat +
               ", Lifespan: " + lifespan + " years, Wildlife: " + isWildlife;
    }
}

// TODO: Create intermediate class Mammal extends Animal:
class Mammal extends Animal {
    // TODO: Add mammal-specific fields:
    // - furColor (String)
    // - hasWarmBlood (boolean) - always true for mammals
    // - gestationPeriod (int) - days
    protected String furColor;
    protected boolean hasWarmBlood;
    protected int gestationPeriod;

    // TODO: Create constructor that:
    // - Takes Animal parameters plus mammal-specific parameters
    // - Calls super() with appropriate parameters
    // - Sets hasWarmBlood to true automatically
    // - Prints "Mammal constructor: Adding mammal traits"
    public Mammal(String species, String habitat, int lifespan, boolean isWildlife,
                  String furColor, int gestationPeriod) {
        super(species, habitat, lifespan, isWildlife);
```

```
        this.furColor = furColor;
        this.hasWarmBlood = true;
        this.gestationPeriod = gestationPeriod;
        System.out.println("Mammal constructor: Adding mammal traits");
    }

    // TODO: Override methods from Animal:
    // - Override move() to print "Mammal is walking/running"
    // - Call super.move() first, then add mammal-specific behavior
    @Override
    public void move() {
        super.move();
        System.out.println("Mammal is walking/running");
    }

    // TODO: Add mammal-specific methods:
    // - nurse() - prints "Mammal is nursing offspring"
    // - regulateTemperature() - prints "Maintaining body temperature"
    public void nurse() {
        System.out.println("Mammal is nursing offspring");
    }

    public void regulateTemperature() {
        System.out.println("Maintaining body temperature");
    }
}

// TODO: Create specific class Dog extends Mammal:
class Dog extends Mammal {
    // TODO: Add dog-specific fields:
    // - breed (String)
    // - isDomesticated (boolean)
    // - loyaltyLevel (int) - 1-10 scale
    // - favoriteActivity (String)
    private String breed;
    private boolean isDomesticated;
    private int loyaltyLevel;
    private String favoriteActivity;

    // TODO: Create multiple constructors with different chaining patterns:
    // Constructor 1: Basic dog with minimal parameters
    // - Calls super() with default mammal/animal values
    // - Sets dog-specific defaults
    public Dog() {
        super("Dog", "Domestic", 13, false, "Varied", 60);
        this.breed = "Unknown";
        this.isDomesticated = true;
        this.loyaltyLevel = 5;
        this.favoriteActivity = "Playing";
        System.out.println("Dog constructor: Creating default dog");
    }

    // Constructor 2: Detailed dog with all parameters
    // - Calls super() with all mammal/animal parameters
    // - Initializes all dog-specific fields
    // - Prints "Dog constructor: Creating [breed] dog"
    public Dog(String species, String habitat, int lifespan, boolean isWildlife,
               String furColor, int gestationPeriod,
               String breed, boolean isDomesticated, int loyaltyLevel, String favoriteActivity) {
        super(species, habitat, lifespan, isWildlife, furColor, gestationPeriod);
        this.breed = breed;
        this.isDomesticated = isDomesticated;
        this.loyaltyLevel = loyaltyLevel;
```

```
        this.favoriteActivity = favoriteActivity;
        System.out.println("Dog constructor: Creating " + breed + " dog");
    }

    // Constructor 3: Copy constructor
    // - Takes another Dog object as parameter
    // - Calls this() with parameters from source dog
    public Dog(Dog other) {
        this(other.species, other.habitat, other.lifespan, other.isWildlife,
             other.furColor, other.gestationPeriod,
             other.breed, other.isDomesticated, other.loyaltyLevel, other.favoriteActivity);
        System.out.println("Dog copy constructor: Copying " + other.breed + " dog");
    }

    // TODO: Override methods from the inheritance chain:
    // - Override eat() to show dog eating behavior
    // - Call super.eat() and add "wagging tail while eating"
    @Override
    public void eat() {
        super.eat();
        System.out.println("Dog is eating and wagging tail while eating");
    }

    // - Override move() to print "Dog is running and playing"
    @Override
    public void move() {
        super.move();
        System.out.println("Dog is running and playing");
    }

    // - Override sleep() to print "Dog is sleeping in doghouse"
    @Override
    public void sleep() {
        System.out.println("Dog is sleeping in doghouse");
    }

    // TODO: Add dog-specific methods:
    // - bark() - prints "Woof! Woof!"
    // - fetch() - prints "Dog is fetching the ball"
    // - showLoyalty() - prints loyalty level message
    public void bark() {
        System.out.println("Woof! Woof!");
    }

    public void fetch() {
        System.out.println("Dog is fetching the ball");
    }

    public void showLoyalty() {
        System.out.println("Loyalty Level: " + loyaltyLevel + " (1-10 scale)");
    }

    // TODO: Create method that demonstrates calling up the chain:
    // - demonstrateInheritance() - calls methods from all three levels
    public void demonstrateInheritance() {
        System.out.println("--- Demonstrating Inheritance Chain ---");
        eat();
        move();
        sleep();
        nurse();
        regulateTemperature();
        bark();
        fetch();
```

```java
    // - Override sleep() to print "Dog is sleeping in doghouse"
    @Override
    public void sleep() {
        System.out.println("Dog is sleeping in doghouse");
    }

    // TODO: Add dog-specific methods:
    // - bark() - prints "Woof! Woof!"
    // - fetch() - prints "Dog is fetching the ball"
    // - showLoyalty() - prints loyalty level message
    public void bark() {
        System.out.println("Woof! Woof!");
    }

    public void fetch() {
        System.out.println("Dog is fetching the ball");
    }

    public void showLoyalty() {
        System.out.println("Loyalty Level: " + loyaltyLevel + " (1-10 scale)");
    }

    // TODO: Create method that demonstrates calling up the chain:
    // - demonstrateInheritance() - calls methods from all three levels
    public void demonstrateInheritance() {
        System.out.println("--- Demonstrating Inheritance Chain ---");
        eat();
        move();
        sleep();
        nurse();
        regulateTemperature();
        bark();
        fetch();
        showLoyalty();
        System.out.println(getAnimalInfo());
        System.out.println("Fur Color: " + furColor);
        System.out.println("Breed: " + breed);
        System.out.println("Favorite Activity: " + favoriteActivity);
    }

    public String getBreed() {
        return breed;
    }
}

public class MultiLevelInheritanceDemo {
    Run main | Debug main
    public static void main(String[] args) {
        System.out.println("=== Basic Dog Constructor ===");
        Dog dog1 = new Dog();
        dog1.demonstrateInheritance();

        System.out.println("\n=== Detailed Dog Constructor ===");
        Dog dog2 = new Dog("Dog", "Home", 15, false, "Brown", 63,
                           "Labrador", true, 9, "Swimming");
        dog2.demonstrateInheritance();

        System.out.println("\n=== Copy Constructor ===");
        Dog dog3 = new Dog(dog2);
        dog3.demonstrateInheritance();

        System.out.println("\n=== Method Overriding Test ===");
        dog2.eat();
        dog2.move();
```

```
220         dog2.move();
221         dog2.sleep();
222
223         System.out.println("\n=== Access Inherited Members ===");
224         System.out.println("Species: " + dog2.species);
225         System.out.println("Habitat: " + dog2.habitat);
226         System.out.println("Fur Color: " + dog2.furColor);
227         System.out.println("Breed: " + dog2.getBreed());
228
229         System.out.println("\n=== Inheritance Chain Test ===");
230         System.out.println("dog2 instanceof Dog: " + (dog2 instanceof Dog));
231         System.out.println("dog2 instanceof Mammal: " + (dog2 instanceof Mammal));
232         System.out.println("dog2 instanceof Animal: " + (dog2 instanceof Animal));
233
234         System.out.println("\n=== Multiple Dog Objects ===");
235         Dog dog4 = new Dog("Dog", "Farm", 12, false, "White", 60,
236                            "Dalmatian", true, 8, "Running");
237         dog4.demonstrateInheritance();
238
239         Dog dog5 = new Dog("Dog", "Street", 10, true, "Black", 58,
240                            "Stray", false, 6, "Exploring");
241         dog5.demonstrateInheritance();
242     }
243 }
```

OUTPUT→

```
=== Basic Dog Constructor ===
Animal constructor: Creating Dog
Mammal constructor: Adding mammal traits
Dog constructor: Creating default dog
--- Demonstrating Inheritance Chain ---
Animal is eating
Dog is eating and wagging tail while eating
Animal is moving
Mammal is walking/running
Dog is running and playing
Dog is sleeping in doghouse
Mammal is nursing offspring
Maintaining body temperature
Woof! Woof!
Dog is fetching the ball
Loyalty Level: 5 (1-10 scale)
Species: Dog, Habitat: Domestic, Lifespan: 13 years, Wildlife: false
Fur Color: Varied
Breed: Unknown
Favorite Activity: Playing
```

```
=== Detailed Dog Constructor ===
Animal constructor: Creating Dog
Mammal constructor: Adding mammal traits
Dog constructor: Creating Labrador dog
--- Demonstrating Inheritance Chain ---
Animal is eating
Dog is eating and wagging tail while eating
Animal is moving
Mammal is walking/running
Dog is running and playing
Dog is sleeping in doghouse
Mammal is nursing offspring
Maintaining body temperature
Woof! Woof!
Dog is fetching the ball
Loyalty Level: 9 (1-10 scale)
Species: Dog, Habitat: Home, Lifespan: 15 years, Wildlife: false
Fur Color: Brown
Breed: Labrador
Favorite Activity: Swimming
```

```
=== Copy Constructor ===
Animal constructor: Creating Dog
Mammal constructor: Adding mammal traits
Dog constructor: Creating Labrador dog
Dog copy constructor: Copying Labrador dog
--- Demonstrating Inheritance Chain ---
Animal is eating
Dog is eating and wagging tail while eating
Animal is moving
Mammal is walking/running
Dog is running and playing
Dog is sleeping in doghouse
Mammal is nursing offspring
Maintaining body temperature
Woof! Woof!
Dog is fetching the ball
Loyalty Level: 9 (1-10 scale)
Species: Dog, Habitat: Home, Lifespan: 15 years, Wildlife: false
Fur Color: Brown
Breed: Labrador
Favorite Activity: Swimming
```

```
=== Method Overriding Test ===
Animal is eating
Dog is eating and wagging tail while eating
Animal is moving
Mammal is walking/running
Dog is running and playing
Dog is sleeping in doghouse

=== Access Inherited Members ===
Species: Dog
Habitat: Home
Fur Color: Brown
Breed: Labrador

=== Inheritance Chain Test ===
dog2 instanceof Dog: true
dog2 instanceof Mammal: true
dog2 instanceof Animal: true

=== Multiple Dog Objects ===
Animal constructor: Creating Dog
Mammal constructor: Adding mammal traits
Dog constructor: Creating Dalmatian dog
--- Demonstrating Inheritance Chain ---
Animal is eating
Dog is eating and wagging tail while eating
```

```
Animal is moving
Mammal is walking/running
Dog is running and playing
Dog is sleeping in doghouse
Mammal is nursing offspring
Maintaining body temperature
Woof! Woof!
Dog is fetching the ball
Loyalty Level: 8 (1-10 scale)
Species: Dog, Habitat: Farm, Lifespan: 12 years, Wildlife: false
Fur Color: White
Breed: Dalmatian
Favorite Activity: Running
Animal constructor: Creating Dog
Mammal constructor: Adding mammal traits
Dog constructor: Creating Stray dog
--- Demonstrating Inheritance Chain ---
Animal is eating
Dog is eating and wagging tail while eating
Animal is moving
Mammal is walking/running
Dog is running and playing
Dog is sleeping in doghouse
Mammal is nursing offspring
Maintaining body temperature
Woof! Woof!
```

```
Dog is fetching the ball
Loyalty Level: 8 (1-10 scale)
Species: Dog, Habitat: Farm, Lifespan: 12 years, Wildlife: false
Fur Color: White
Breed: Dalmatian
Favorite Activity: Running
Animal constructor: Creating Dog
Mammal constructor: Adding mammal traits
Dog constructor: Creating Stray dog
--- Demonstrating Inheritance Chain ---
Animal is eating
Dog is eating and wagging tail while eating
Animal is moving
Mammal is walking/running
Dog is running and playing
Dog is sleeping in doghouse
Mammal is nursing offspring
Maintaining body temperature
Woof! Woof!
Dog is fetching the ball
Loyalty Level: 6 (1-10 scale)
Species: Dog, Habitat: Street, Lifespan: 10 years, Wildlife: true
Fur Color: Black
Breed: Stray
Favorite Activity: Exploring
```