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

Name:Ramesh Harisabapathi Chettiar

Date of Submission:23/09/25

Qno1→Problem Statement:

Create a Fruit class with color and taste fields. Create an Apple class that extends

Fruit and adds variety field.

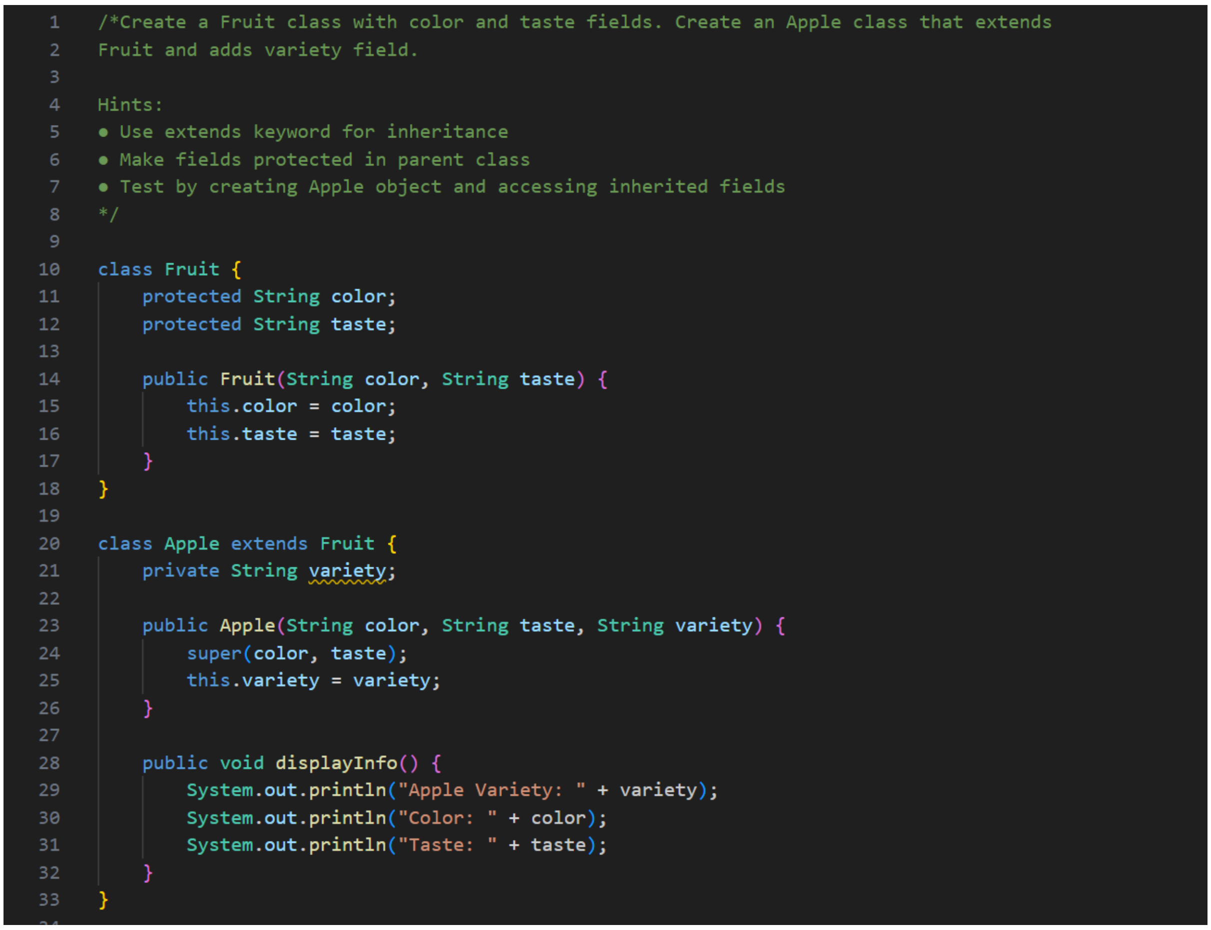
Hints:

● Use extends keyword for inheritance

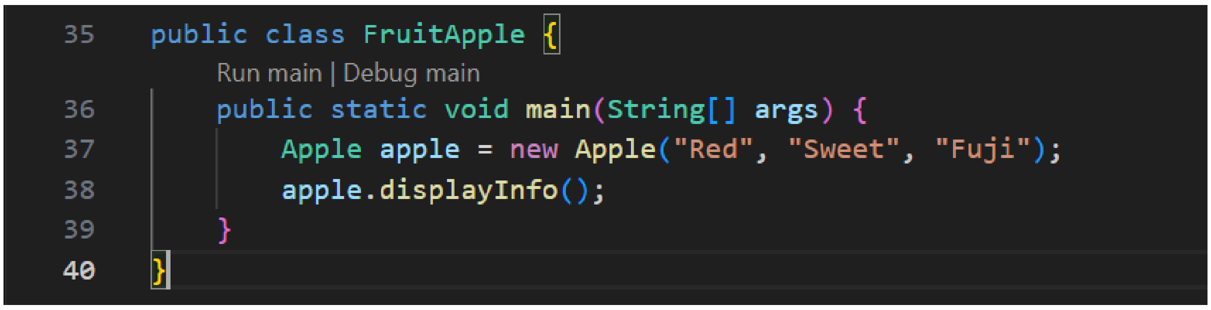
● Make fields protected in parent class

● Test by creating Apple object and accessing inherited fields

PROGRAM→

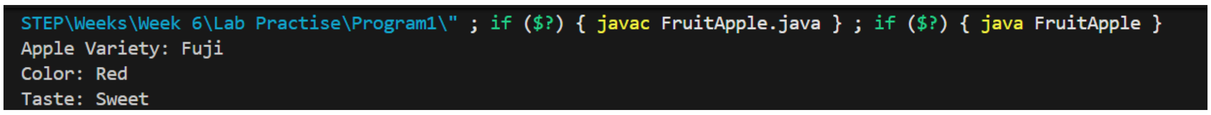


```java
/*Create a Fruit class with color and taste fields. Create an Apple class that extends
Fruit and adds variety field.

Hints:
● Use extends keyword for inheritance
● Make fields protected in parent class
● Test by creating Apple object and accessing inherited fields
*/

class Fruit {
    protected String color;
    protected String taste;

    public Fruit(String color, String taste) {
        this.color = color;
        this.taste = taste;
    }
}

class Apple extends Fruit {
    private String variety;

    public Apple(String color, String taste, String variety) {
        super(color, taste);
        this.variety = variety;
    }

    public void displayInfo() {
        System.out.println("Apple Variety: " + variety);
        System.out.println("Color: " + color);
        System.out.println("Taste: " + taste);
    }
}
```



```java
public class FruitApple {
    public static void main(String[] args) {
        Apple apple = new Apple("Red", "Sweet", "Fuji");
        apple.displayInfo();
    }
}
```

OUTPUT→



```
STEP\Weeks\Week 6\Lab Practise\Program1\" ; if ($?) { javac FruitApple.java } ; if ($?) { java FruitApple }
Apple Variety: Fuji
Color: Red
Taste: Sweet
```

QNO2→Problem Statement:

Create Phone class with brand and model. Create SmartPhone class extending Phone with operatingSystem field. Use constructor chaining.

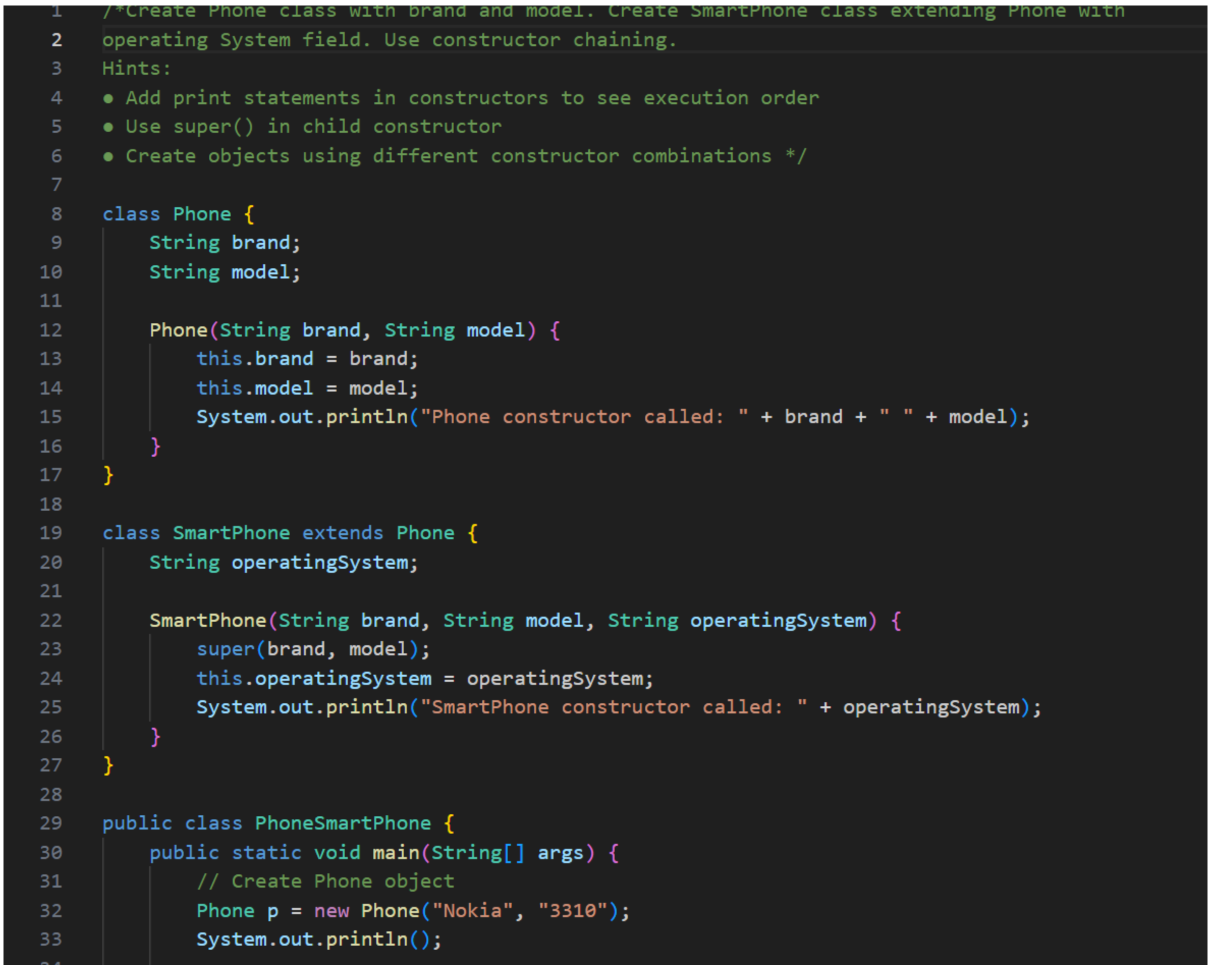
Hints:

● Add print statements in constructors to see execution order

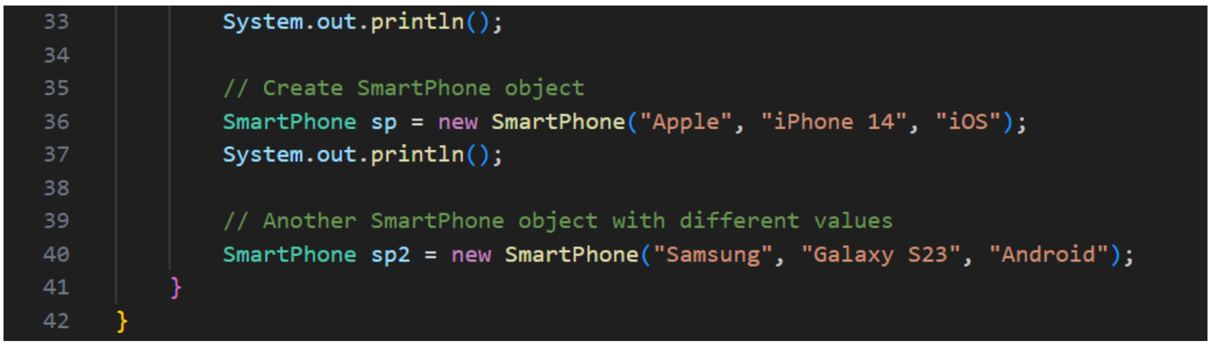
● Use super() in child constructor

● Create objects using different constructor combinations

PROGRAM→

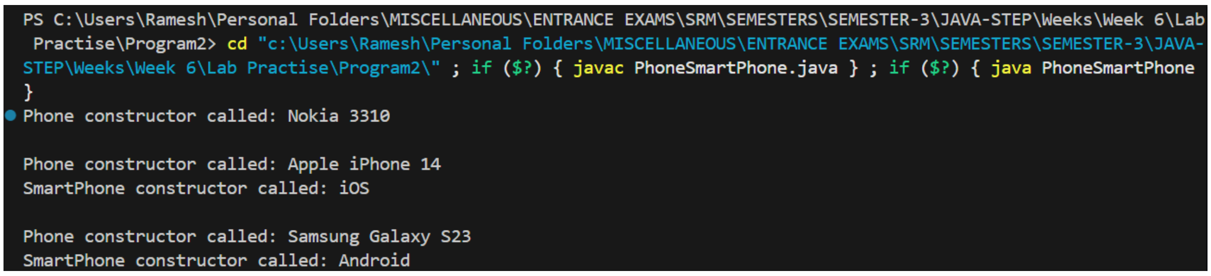


```java
/*Create Phone class with brand and model. Create SmartPhone class extending Phone with
operating System field. Use constructor chaining.
Hints:
• Add print statements in constructors to see execution order
• Use super() in child constructor
• Create objects using different constructor combinations */

class Phone {
    String brand;
    String model;

    Phone(String brand, String model) {
        this.brand = brand;
        this.model = model;
        System.out.println("Phone constructor called: " + brand + " " + model);
    }
}

class SmartPhone extends Phone {
    String operatingSystem;

    SmartPhone(String brand, String model, String operatingSystem) {
        super(brand, model);
        this.operatingSystem = operatingSystem;
        System.out.println("SmartPhone constructor called: " + operatingSystem);
    }
}

public class PhoneSmartPhone {
    public static void main(String[] args) {
        // Create Phone object
        Phone p = new Phone("Nokia", "3310");
        System.out.println();
```



```java
        System.out.println();

        // Create SmartPhone object
        SmartPhone sp = new SmartPhone("Apple", "iPhone 14", "iOS");
        System.out.println();

        // Another SmartPhone object with different values
        SmartPhone sp2 = new SmartPhone("Samsung", "Galaxy S23", "Android");
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Lab
 Practise\Program2> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-
STEP\Weeks\Week 6\Lab Practise\Program2\" ; if ($?) { javac PhoneSmartPhone.java } ; if ($?) { java PhoneSmartPhone
}
Phone constructor called: Nokia 3310

Phone constructor called: Apple iPhone 14
SmartPhone constructor called: iOS

Phone constructor called: Samsung Galaxy S23
SmartPhone constructor called: Android
```

QNO3→

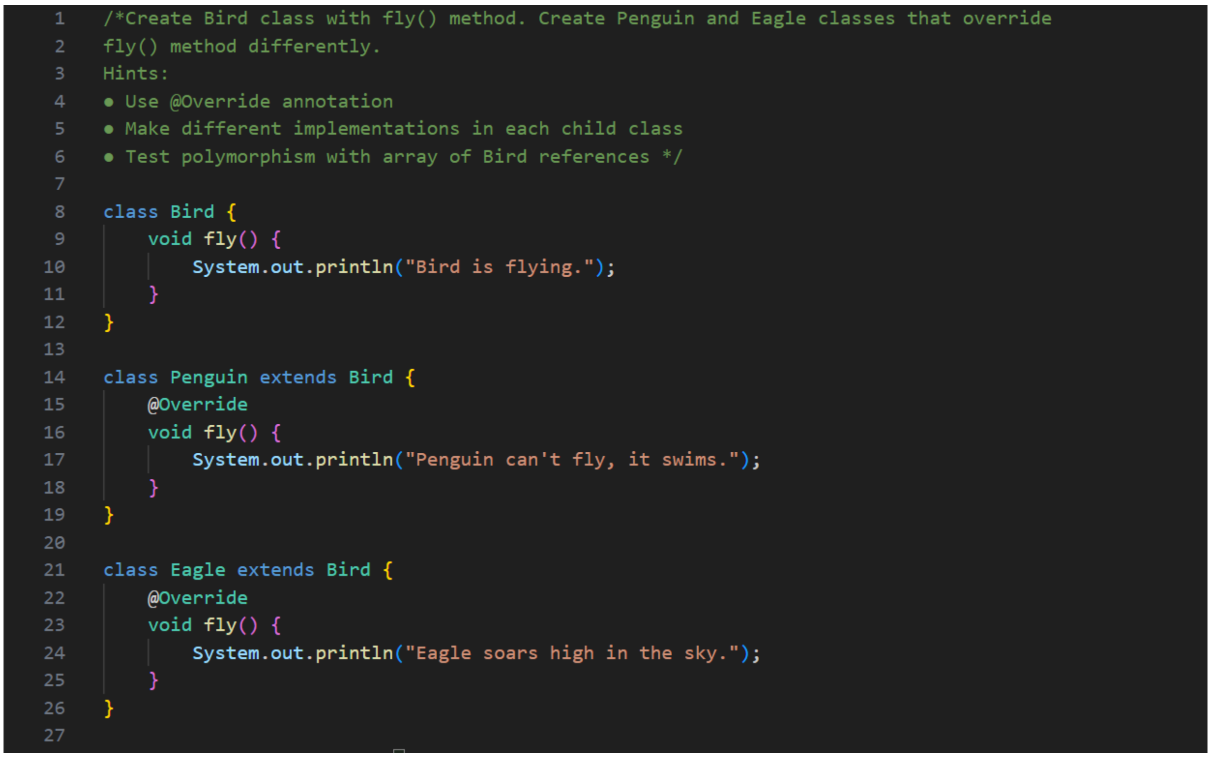
Problem Statement:

Create Bird class with fly() method. Create Penguin and Eagle classes that override fly() method differently.

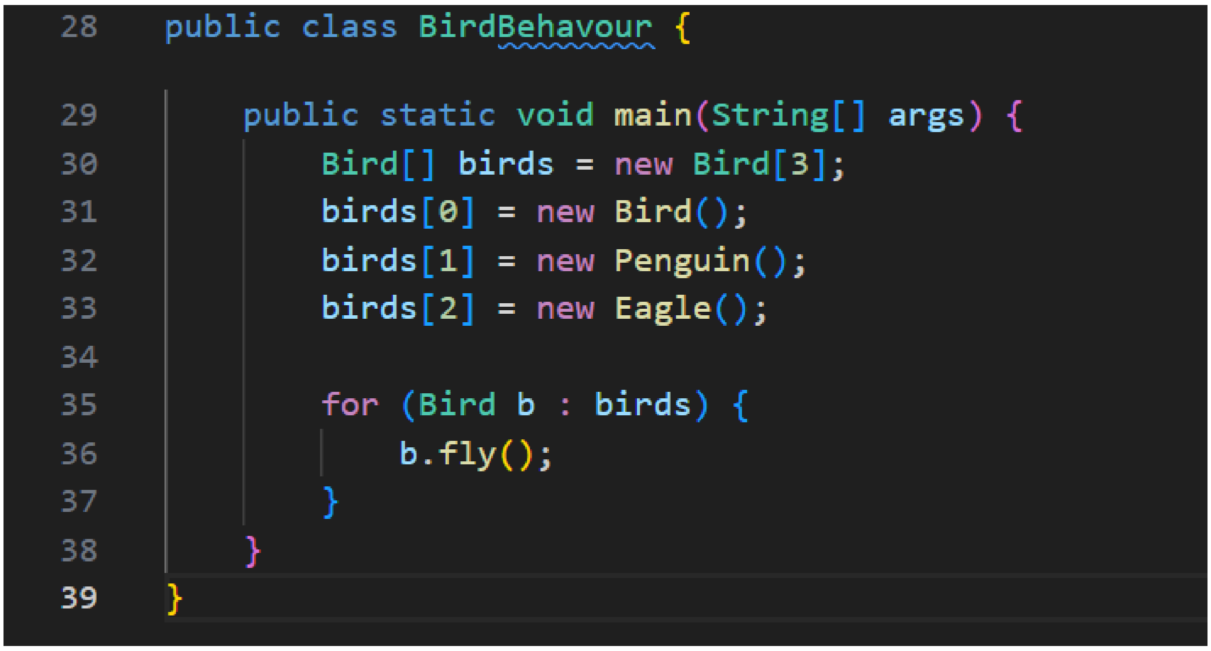
Hints:

- **Use @Override annotation**
- **Make different implementations in each child class**
- **Test polymorphism with array of Bird references**

PROGRAM→

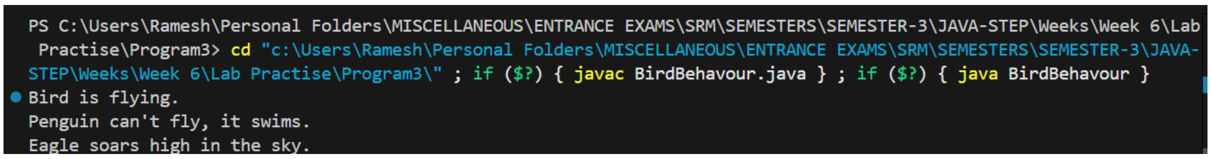


```java
/*Create Bird class with fly() method. Create Penguin and Eagle classes that override
fly() method differently.
Hints:
• Use @Override annotation
• Make different implementations in each child class
• Test polymorphism with array of Bird references */

class Bird {
    void fly() {
        System.out.println("Bird is flying.");
    }
}

class Penguin extends Bird {
    @Override
    void fly() {
        System.out.println("Penguin can't fly, it swims.");
    }
}

class Eagle extends Bird {
    @Override
    void fly() {
        System.out.println("Eagle soars high in the sky.");
    }
}

```



```java
public class BirdBehavour {

    public static void main(String[] args) {
        Bird[] birds = new Bird[3];
        birds[0] = new Bird();
        birds[1] = new Penguin();
        birds[2] = new Eagle();

        for (Bird b : birds) {
            b.fly();
        }
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Lab
 Practise\Program3> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-
STEP\Weeks\Week 6\Lab Practise\Program3\" ; if ($?) { javac BirdBehavour.java } ; if ($?) { java BirdBehavour }
Bird is flying.
Penguin can't fly, it swims.
Eagle soars high in the sky.
```

QNO 4→

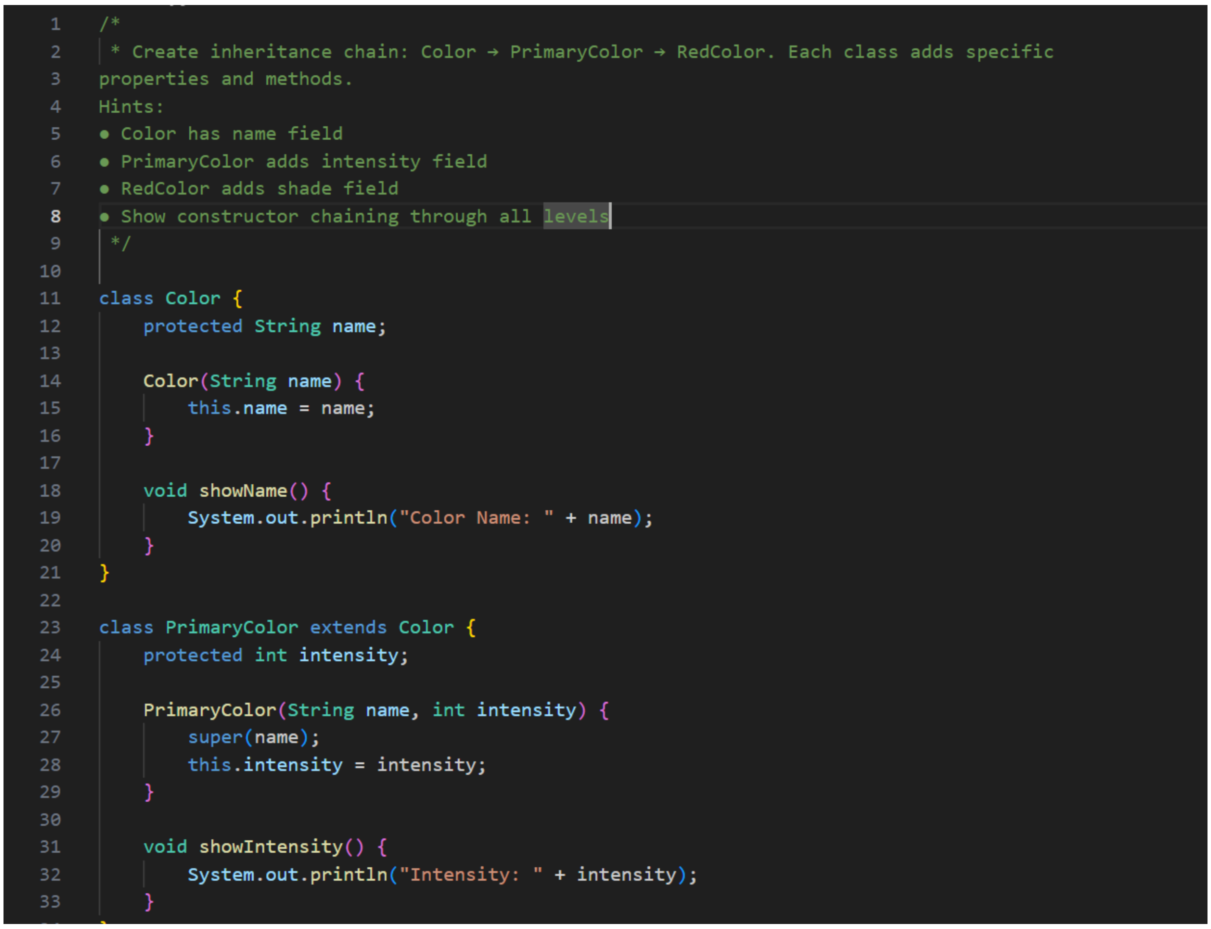
Problem Statement:

Create inheritance chain: Color → PrimaryColor → RedColor. Each class adds specific properties and methods.

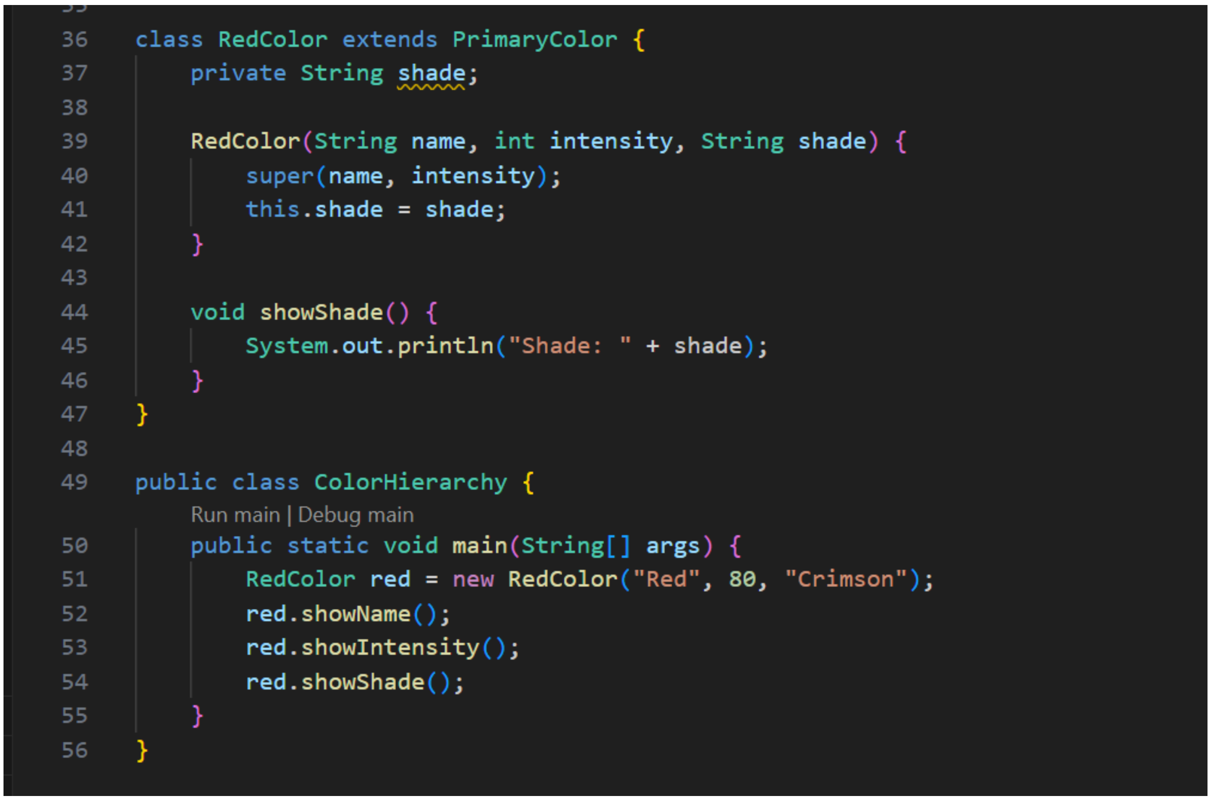
Hints:

- **Color has name field**
- **PrimaryColor adds intensity field**
- **RedColor adds shade field**
- **Show constructor chaining through all levels**

PROGRAM→

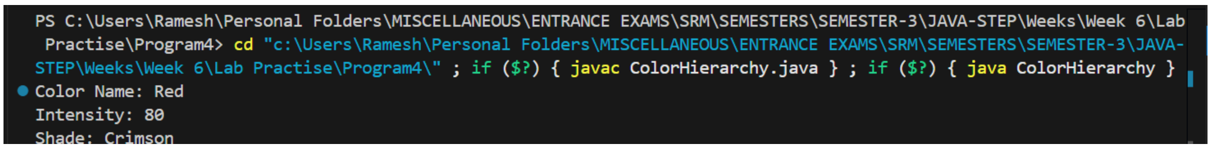


```java
/*
 * Create inheritance chain: Color → PrimaryColor → RedColor. Each class adds specific
properties and methods.
Hints:
• Color has name field
• PrimaryColor adds intensity field
• RedColor adds shade field
• Show constructor chaining through all levels
 */

class Color {
    protected String name;

    Color(String name) {
        this.name = name;
    }

    void showName() {
        System.out.println("Color Name: " + name);
    }
}

class PrimaryColor extends Color {
    protected int intensity;

    PrimaryColor(String name, int intensity) {
        super(name);
        this.intensity = intensity;
    }

    void showIntensity() {
        System.out.println("Intensity: " + intensity);
    }
```



```java
class RedColor extends PrimaryColor {
    private String shade;

    RedColor(String name, int intensity, String shade) {
        super(name, intensity);
        this.shade = shade;
    }

    void showShade() {
        System.out.println("Shade: " + shade);
    }
}

public class ColorHierarchy {
    public static void main(String[] args) {
        RedColor red = new RedColor("Red", 80, "Crimson");
        red.showName();
        red.showIntensity();
        red.showShade();
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Lab
 Practise\Program4> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-
STEP\Weeks\Week 6\Lab Practise\Program4\" ; if ($?) { javac ColorHierarchy.java } ; if ($?) { java ColorHierarchy }
Color Name: Red
Intensity: 80
Shade: Crimson
```

QNO 5→

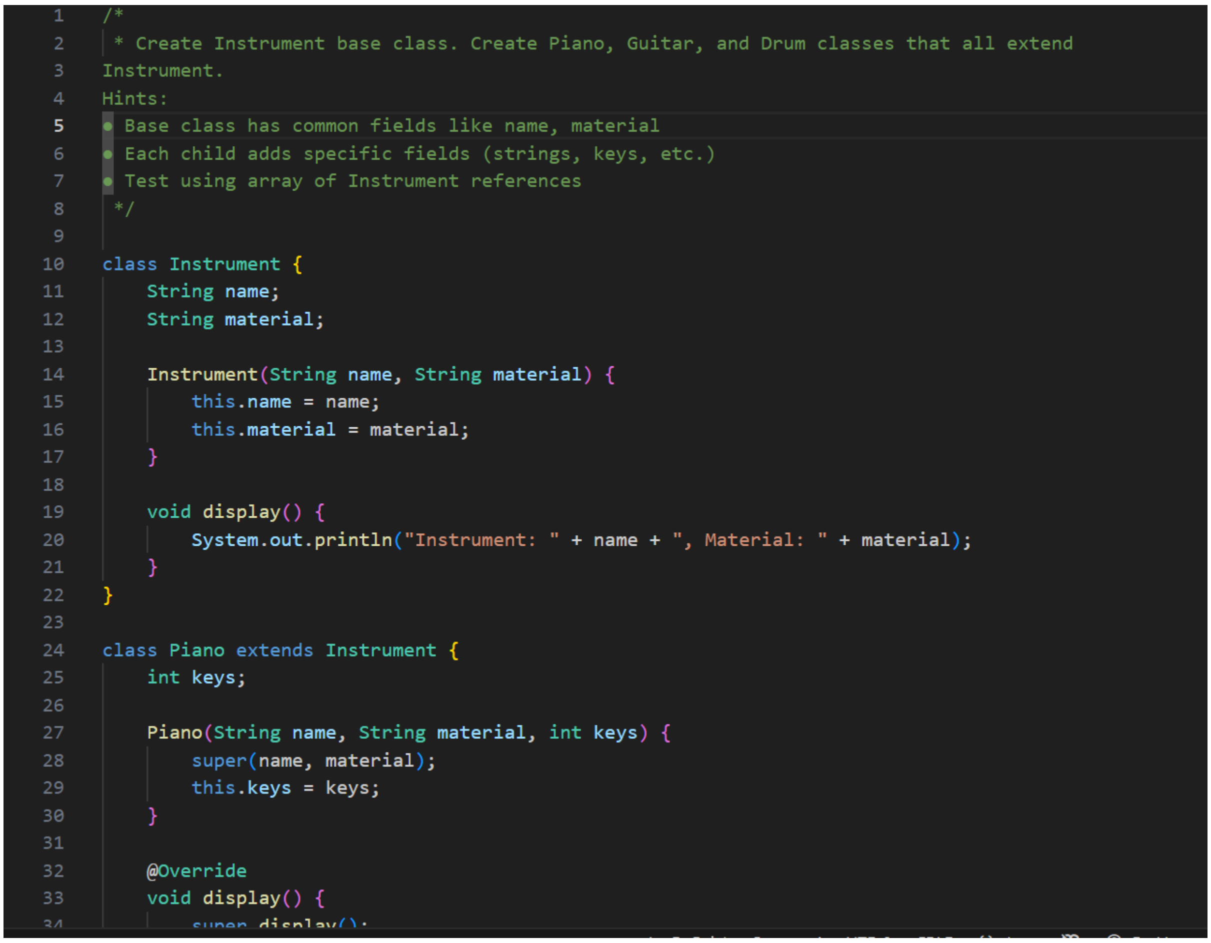
Problem Statement:

Create Instrument base class. Create Piano, Guitar, and Drum classes that all extend Instrument.

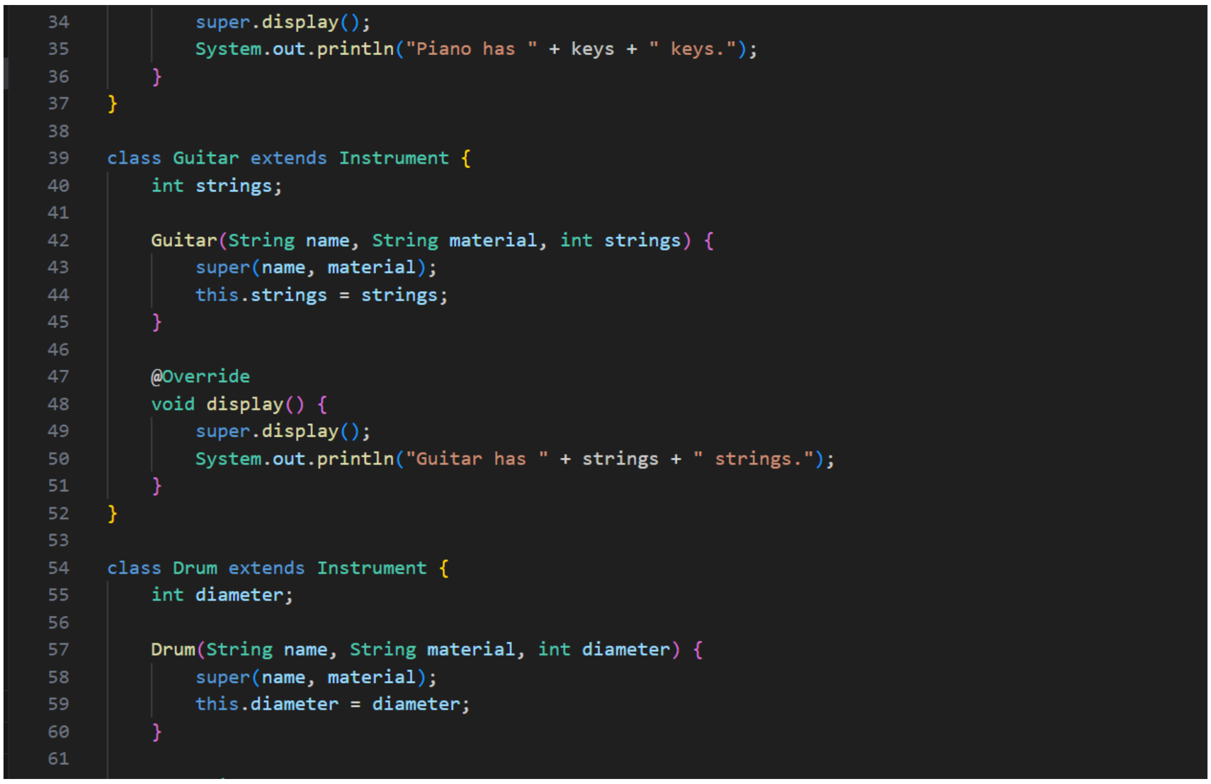
Hints:

- **Base class has common fields like name, material**
- **Each child adds specific fields (strings, keys, etc.)**
- **Test using array of Instrument references**

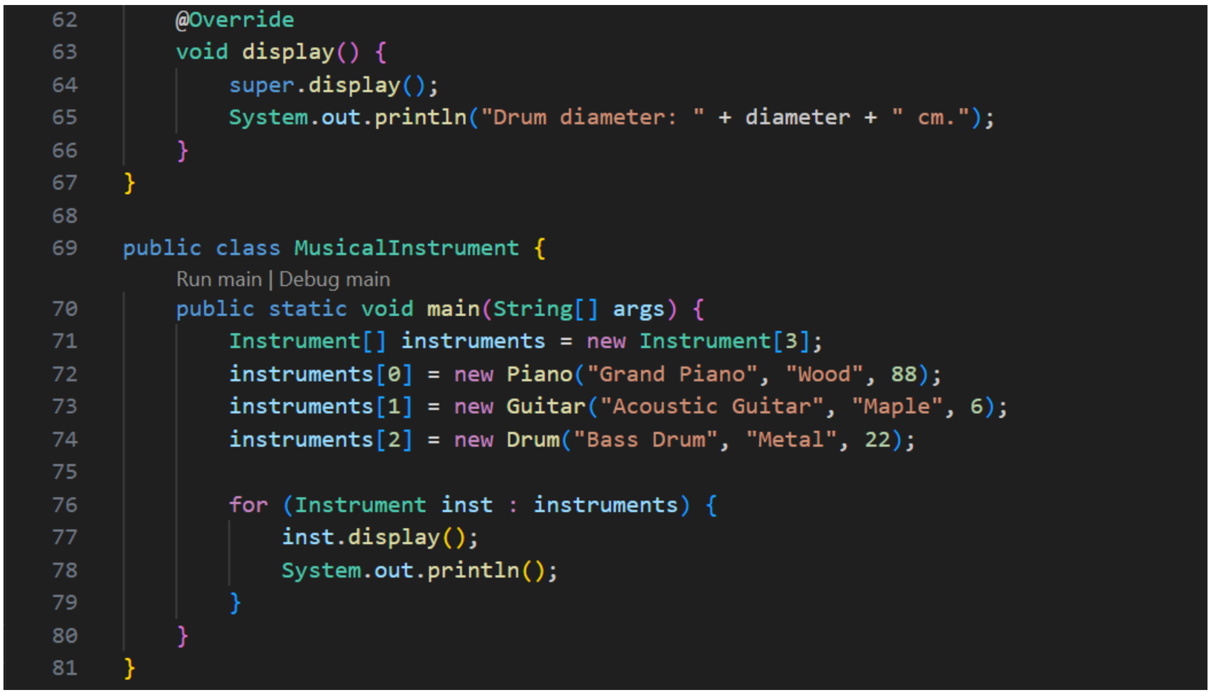
PROGRAM→



```java
/*
 * Create Instrument base class. Create Piano, Guitar, and Drum classes that all extend
Instrument.
Hints:
• Base class has common fields like name, material
• Each child adds specific fields (strings, keys, etc.)
• Test using array of Instrument references
 */

class Instrument {
    String name;
    String material;

    Instrument(String name, String material) {
        this.name = name;
        this.material = material;
    }

    void display() {
        System.out.println("Instrument: " + name + ", Material: " + material);
    }
}

class Piano extends Instrument {
    int keys;

    Piano(String name, String material, int keys) {
        super(name, material);
        this.keys = keys;
    }

    @Override
    void display() {
```

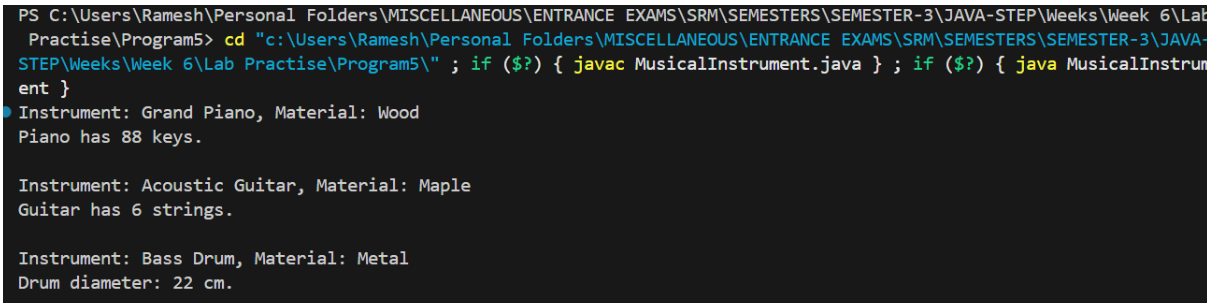


```java
        super.display();
        System.out.println("Piano has " + keys + " keys.");
    }
}

class Guitar extends Instrument {
    int strings;

    Guitar(String name, String material, int strings) {
        super(name, material);
        this.strings = strings;
    }

    @Override
    void display() {
        super.display();
        System.out.println("Guitar has " + strings + " strings.");
    }
}

class Drum extends Instrument {
    int diameter;

    Drum(String name, String material, int diameter) {
        super(name, material);
        this.diameter = diameter;
    }

```



```java
    @Override
    void display() {
        super.display();
        System.out.println("Drum diameter: " + diameter + " cm.");
    }
}

public class MusicalInstrument {
    Run main | Debug main
    public static void main(String[] args) {
        Instrument[] instruments = new Instrument[3];
        instruments[0] = new Piano("Grand Piano", "Wood", 88);
        instruments[1] = new Guitar("Acoustic Guitar", "Maple", 6);
        instruments[2] = new Drum("Bass Drum", "Metal", 22);

        for (Instrument inst : instruments) {
            inst.display();
            System.out.println();
        }
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Lab
 Practise\Program5> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-
STEP\Weeks\Week 6\Lab Practise\Program5\" ; if ($?) { javac MusicalInstrument.java } ; if ($?) { java MusicalInstrum
ent }
Instrument: Grand Piano, Material: Wood
Piano has 88 keys.

Instrument: Acoustic Guitar, Material: Maple
Guitar has 6 strings.

Instrument: Bass Drum, Material: Metal
Drum diameter: 22 cm.
```

QNO 6→

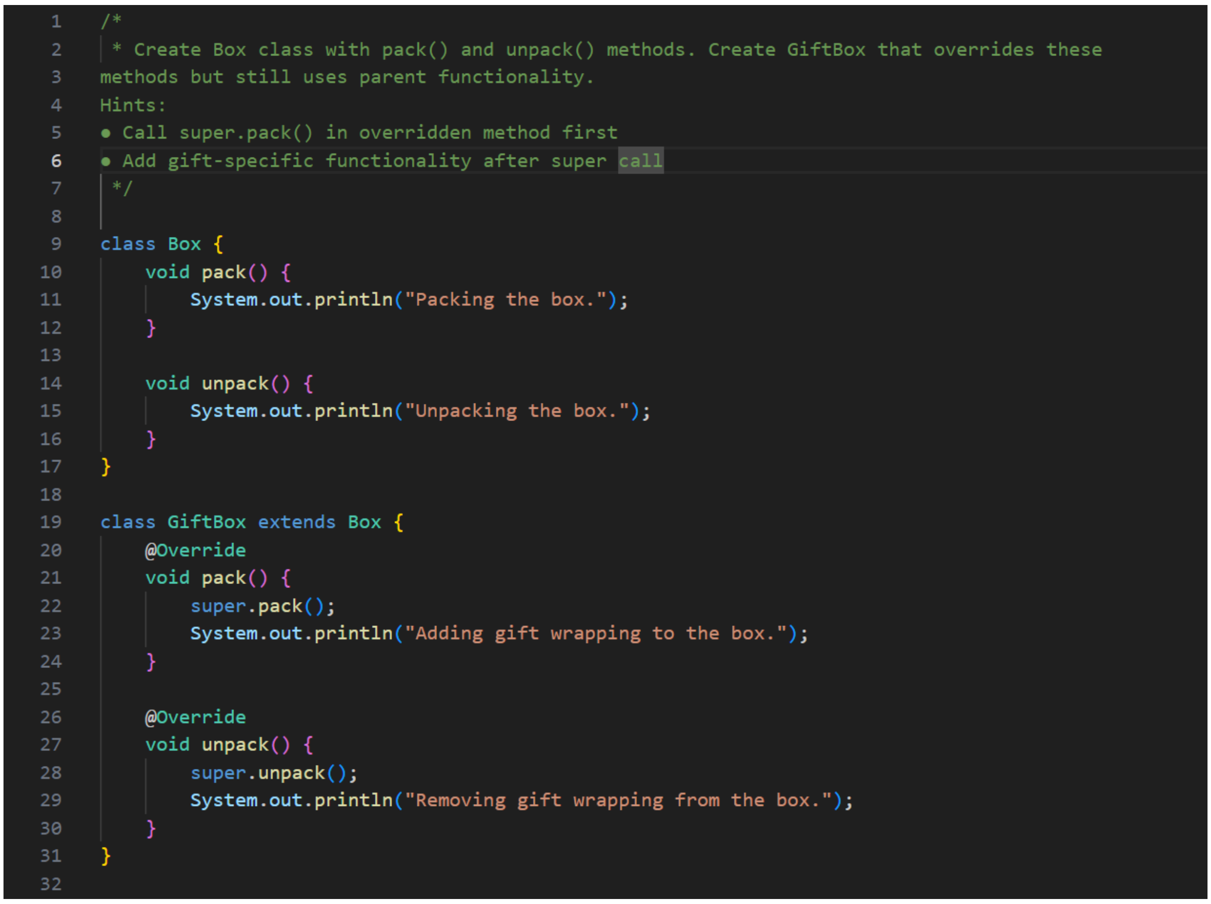
Problem Statement:

Create Box class with pack() and unpack() methods. Create GiftBox that overrides these methods but still uses parent functionality.

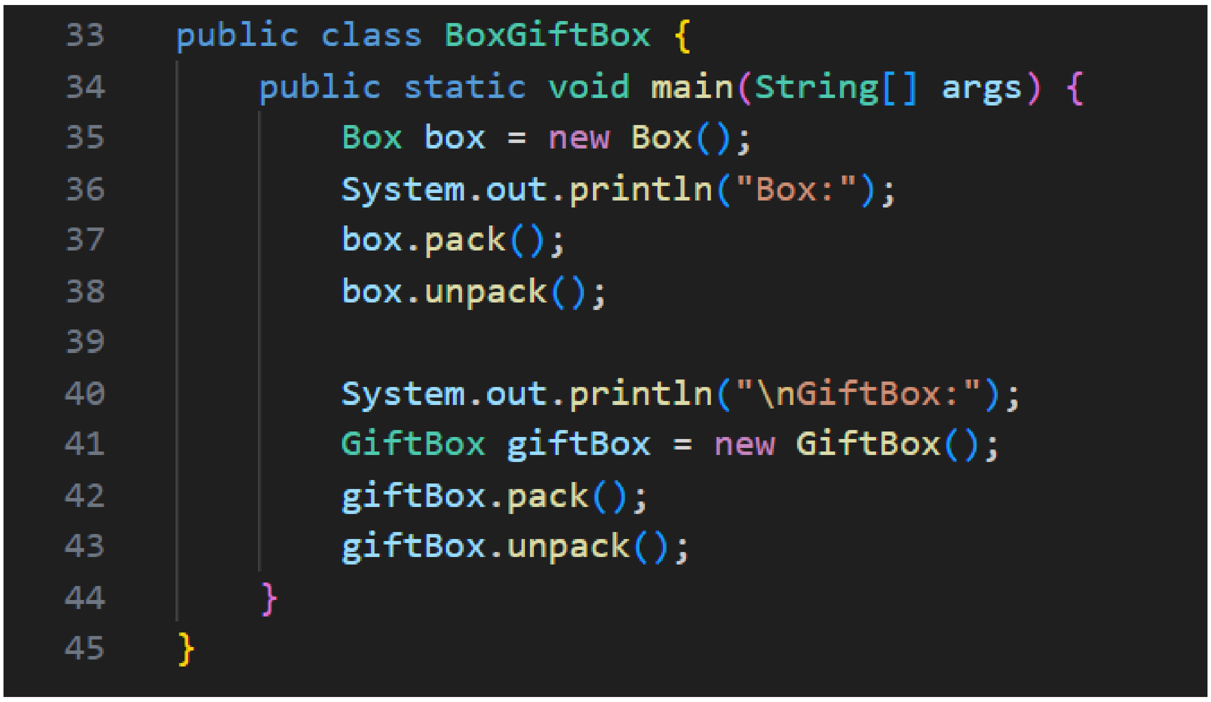
Hints:

- **Call super.pack() in overridden method first**
- **Add gift-specific functionality after super call**
- **Show enhanced behavior while preserving original**

PROGRAM→

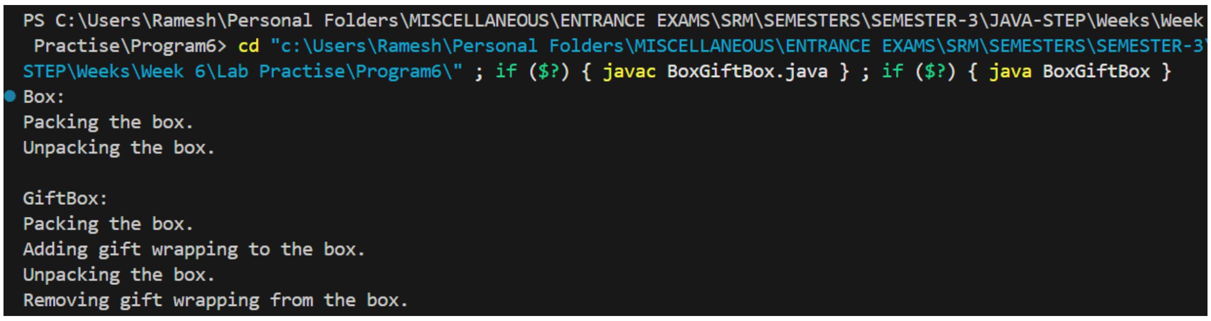


```java
/*
 * Create Box class with pack() and unpack() methods. Create GiftBox that overrides these
methods but still uses parent functionality.
Hints:
• Call super.pack() in overridden method first
• Add gift-specific functionality after super call
 */

class Box {
    void pack() {
        System.out.println("Packing the box.");
    }

    void unpack() {
        System.out.println("Unpacking the box.");
    }
}

class GiftBox extends Box {
    @Override
    void pack() {
        super.pack();
        System.out.println("Adding gift wrapping to the box.");
    }

    @Override
    void unpack() {
        super.unpack();
        System.out.println("Removing gift wrapping from the box.");
    }
}

```



```java
public class BoxGiftBox {
    public static void main(String[] args) {
        Box box = new Box();
        System.out.println("Box:");
        box.pack();
        box.unpack();

        System.out.println("\nGiftBox:");
        GiftBox giftBox = new GiftBox();
        giftBox.pack();
        giftBox.unpack();
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week
 Practise\Program6> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3
STEP\Weeks\Week 6\Lab Practise\Program6\" ; if ($?) { javac BoxGiftBox.java } ; if ($?) { java BoxGiftBox }
Box:
Packing the box.
Unpacking the box.

GiftBox:
Packing the box.
Adding gift wrapping to the box.
Unpacking the box.
Removing gift wrapping from the box.
```