Week 6 - S6 - Core OOP - Inheritance - Assignment Problem (HW)

Name:Ramesh Harisabapathi Chettiar

Date of Submission:23/09/25

QNO1→

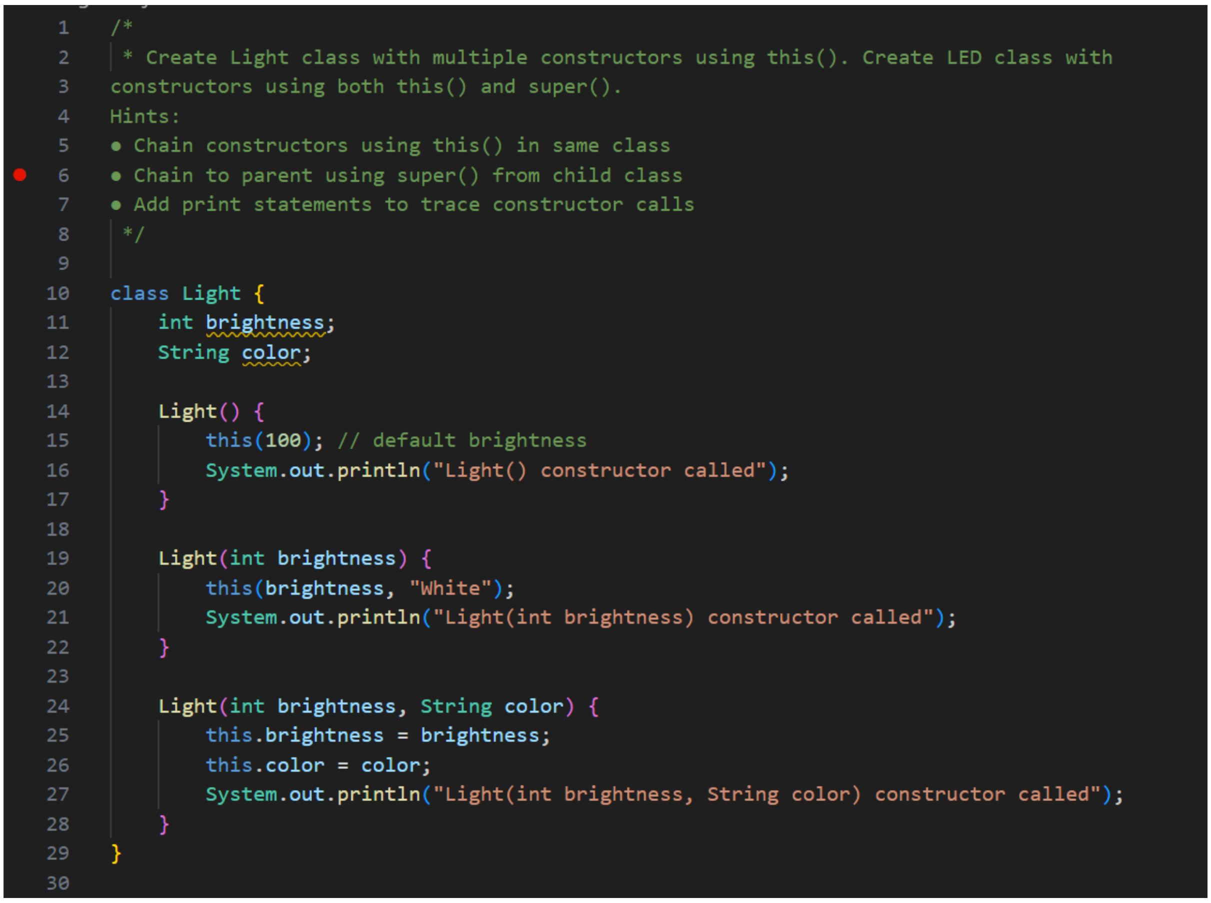
Problem Statement:

Create Light class with multiple constructors using this(). Create LED class with constructors using both this() and super().

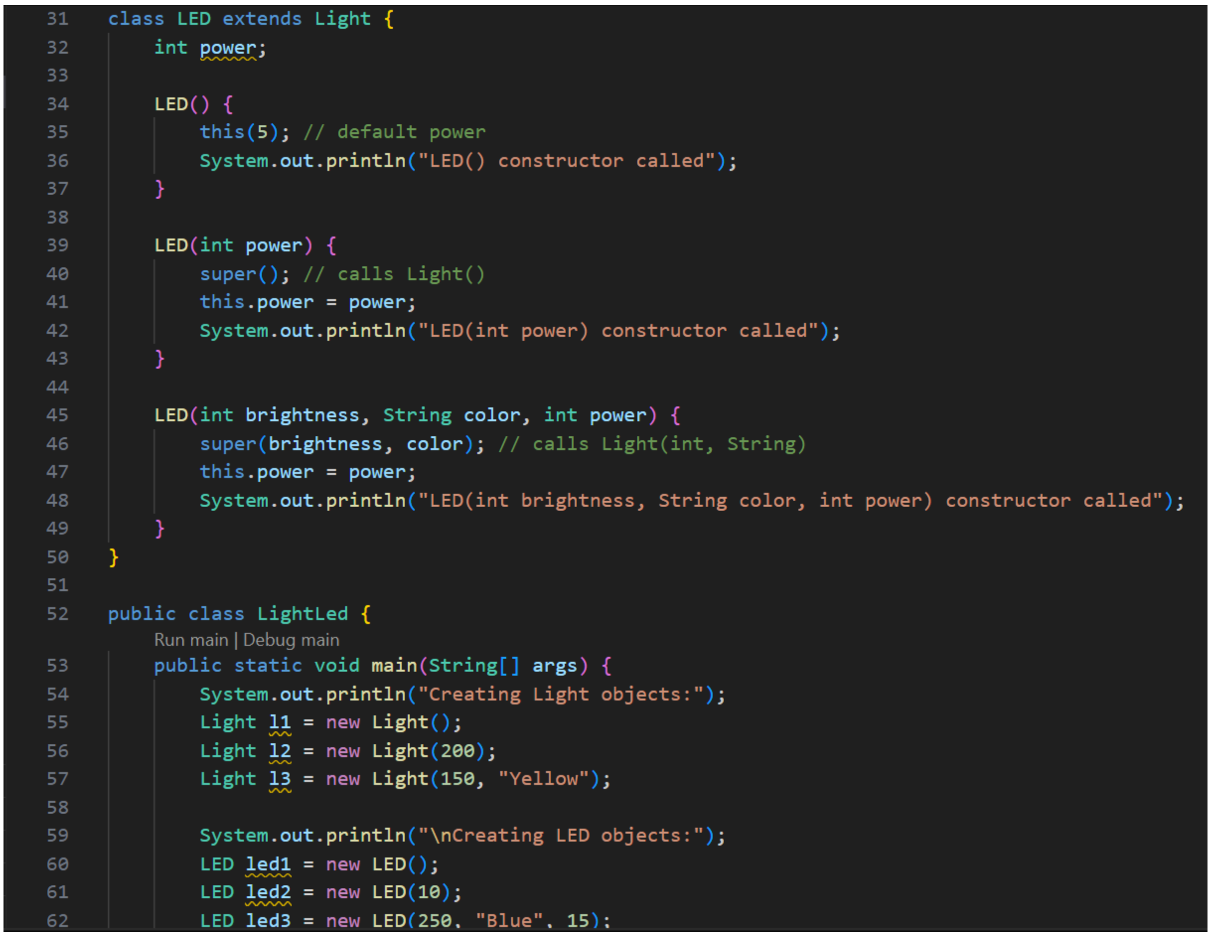
Hints:

- **Chain constructors using this() in same class**
- **Chain to parent using super() from child class**
- **Add print statements to trace constructor calls**

PROGRAM→

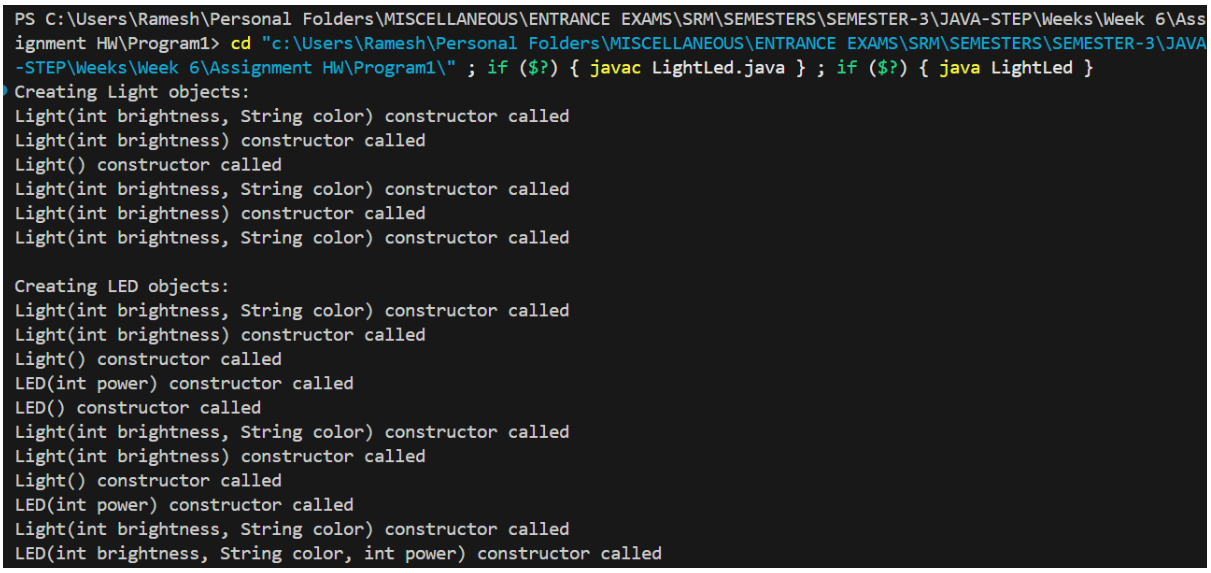


```java
/*
 * Create Light class with multiple constructors using this(). Create LED class with
constructors using both this() and super().
Hints:
• Chain constructors using this() in same class
• Chain to parent using super() from child class
• Add print statements to trace constructor calls
 */

class Light {
    int brightness;
    String color;

    Light() {
        this(100); // default brightness
        System.out.println("Light() constructor called");
    }

    Light(int brightness) {
        this(brightness, "White");
        System.out.println("Light(int brightness) constructor called");
    }

    Light(int brightness, String color) {
        this.brightness = brightness;
        this.color = color;
        System.out.println("Light(int brightness, String color) constructor called");
    }
}

```



```java
class LED extends Light {
    int power;

    LED() {
        this(5); // default power
        System.out.println("LED() constructor called");
    }

    LED(int power) {
        super(); // calls Light()
        this.power = power;
        System.out.println("LED(int power) constructor called");
    }

    LED(int brightness, String color, int power) {
        super(brightness, color); // calls Light(int, String)
        this.power = power;
        System.out.println("LED(int brightness, String color, int power) constructor called");
    }
}

public class LightLed {
    public static void main(String[] args) {
        System.out.println("Creating Light objects:");
        Light l1 = new Light();
        Light l2 = new Light(200);
        Light l3 = new Light(150, "Yellow");

        System.out.println("\nCreating LED objects:");
        LED led1 = new LED();
        LED led2 = new LED(10);
        LED led3 = new LED(250, "Blue", 15);
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Ass
ignment HW\Program1> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA
-STEP\Weeks\Week 6\Assignment HW\Program1\" ; if ($?) { javac LightLed.java } ; if ($?) { java LightLed }
Creating Light objects:
Light(int brightness, String color) constructor called
Light(int brightness) constructor called
Light() constructor called
Light(int brightness, String color) constructor called
Light(int brightness) constructor called
Light(int brightness, String color) constructor called

Creating LED objects:
Light(int brightness, String color) constructor called
Light(int brightness) constructor called
Light() constructor called
LED(int power) constructor called
LED() constructor called
Light(int brightness, String color) constructor called
Light(int brightness) constructor called
Light() constructor called
LED(int power) constructor called
Light(int brightness, String color) constructor called
LED(int brightness, String color, int power) constructor called
```

QNO 2→

Problem Statement:

Create Tool class with private, protected, and public fields. Create Hammer class and test field accessibility.

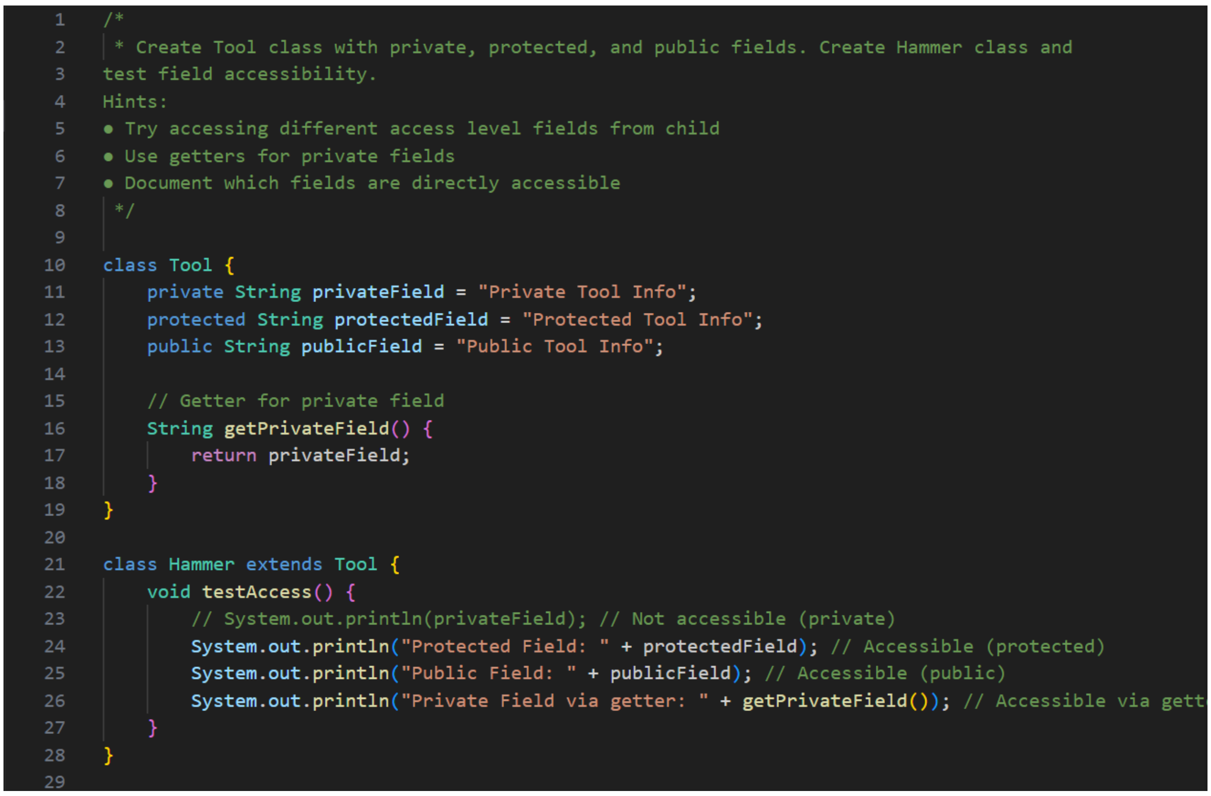
Hints:

● Try accessing different access level fields from child

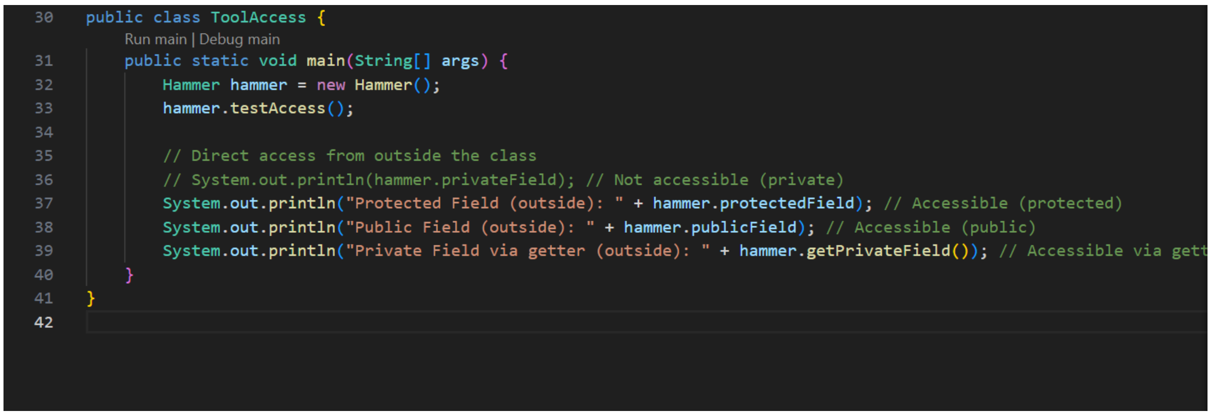
● Use getters for private fields

● Document which fields are directly accessible

PROGRAM→

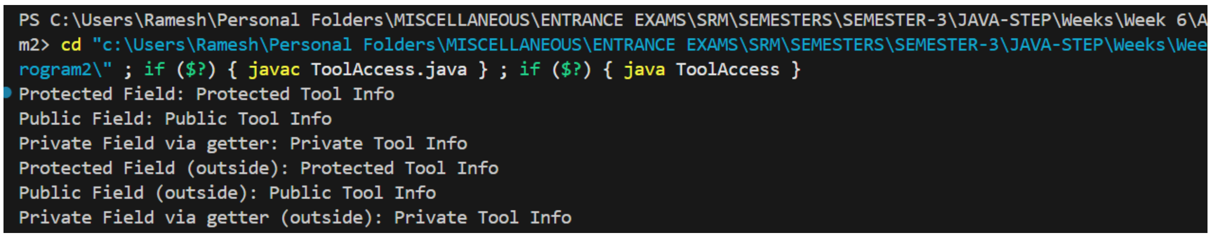


```java
/*
 * Create Tool class with private, protected, and public fields. Create Hammer class and
test field accessibility.
Hints:
● Try accessing different access level fields from child
● Use getters for private fields
● Document which fields are directly accessible
 */

class Tool {
    private String privateField = "Private Tool Info";
    protected String protectedField = "Protected Tool Info";
    public String publicField = "Public Tool Info";

    // Getter for private field
    String getPrivateField() {
        return privateField;
    }
}

class Hammer extends Tool {
    void testAccess() {
        // System.out.println(privateField); // Not accessible (private)
        System.out.println("Protected Field: " + protectedField); // Accessible (protected)
        System.out.println("Public Field: " + publicField); // Accessible (public)
        System.out.println("Private Field via getter: " + getPrivateField()); // Accessible via gett
    }
}

```



```java
public class ToolAccess {
    Run main | Debug main
    public static void main(String[] args) {
        Hammer hammer = new Hammer();
        hammer.testAccess();

        // Direct access from outside the class
        // System.out.println(hammer.privateField); // Not accessible (private)
        System.out.println("Protected Field (outside): " + hammer.protectedField); // Accessible (protected)
        System.out.println("Public Field (outside): " + hammer.publicField); // Accessible (public)
        System.out.println("Private Field via getter (outside): " + hammer.getPrivateField()); // Accessible via gett
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\A
m2> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Wee
rogram2\" ; if ($?) { javac ToolAccess.java } ; if ($?) { java ToolAccess }
Protected Field: Protected Tool Info
Public Field: Public Tool Info
Private Field via getter: Private Tool Info
Protected Field (outside): Protected Tool Info
Public Field (outside): Public Tool Info
Private Field via getter (outside): Private Tool Info
```

QNO 3→

Problem Statement:

Create Game class overriding toString() and equals(). Create CardGame extending Game and override these methods properly.

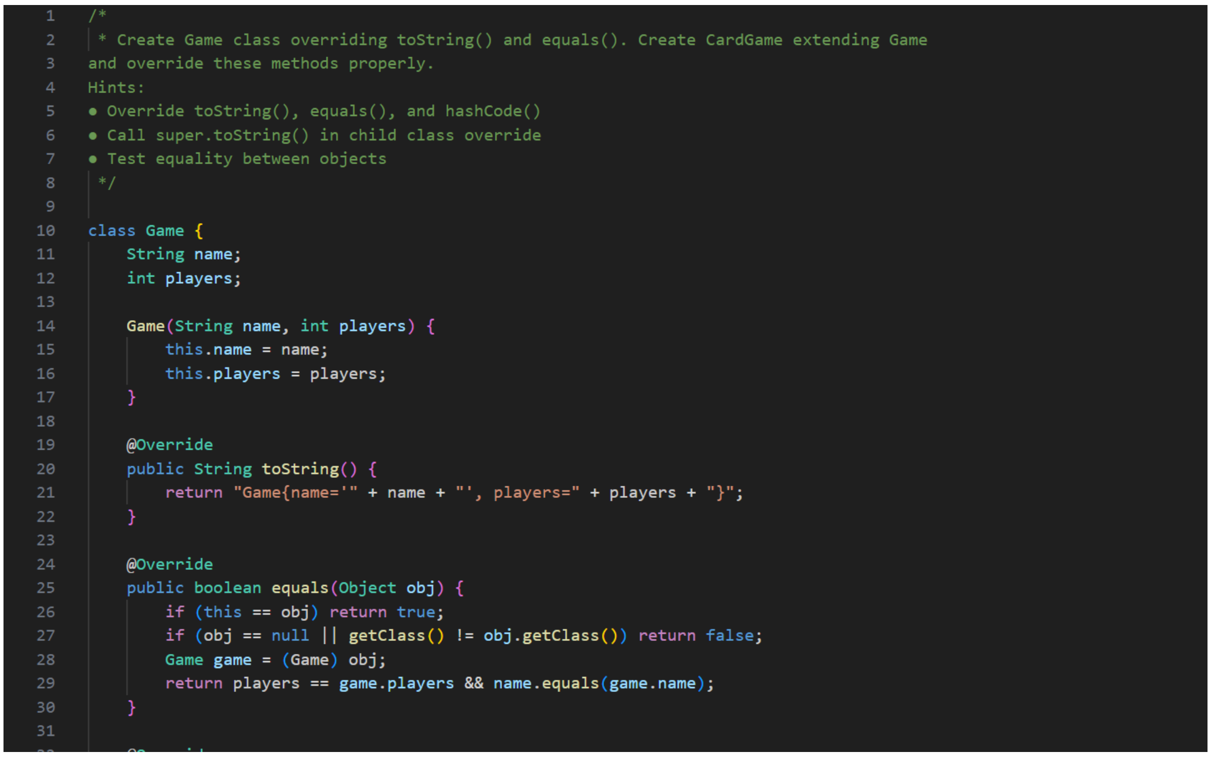
Hints:

● Override toString(), equals(), and hashCode()

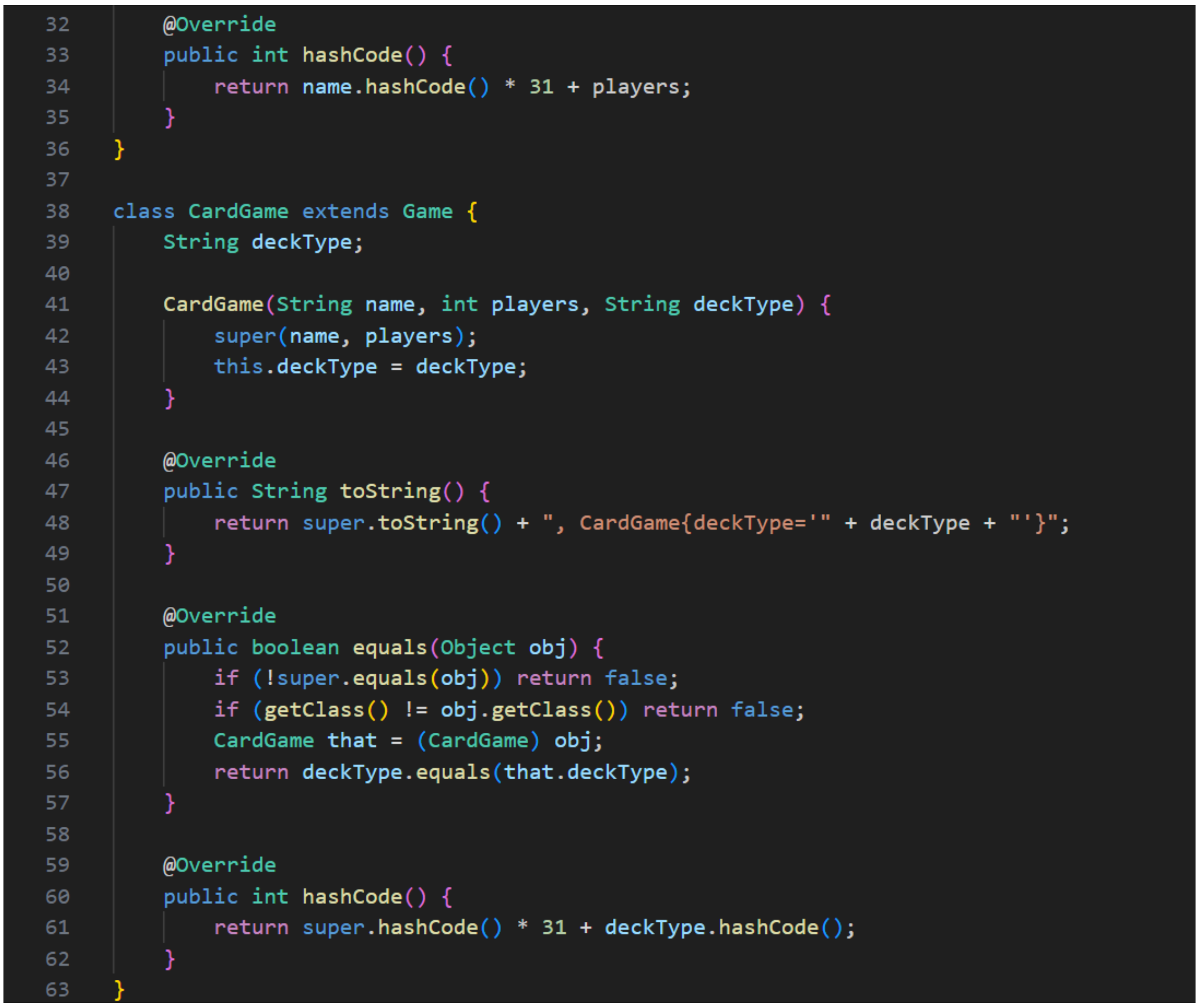
● Call super.toString() in child class override

● Test equality between objects

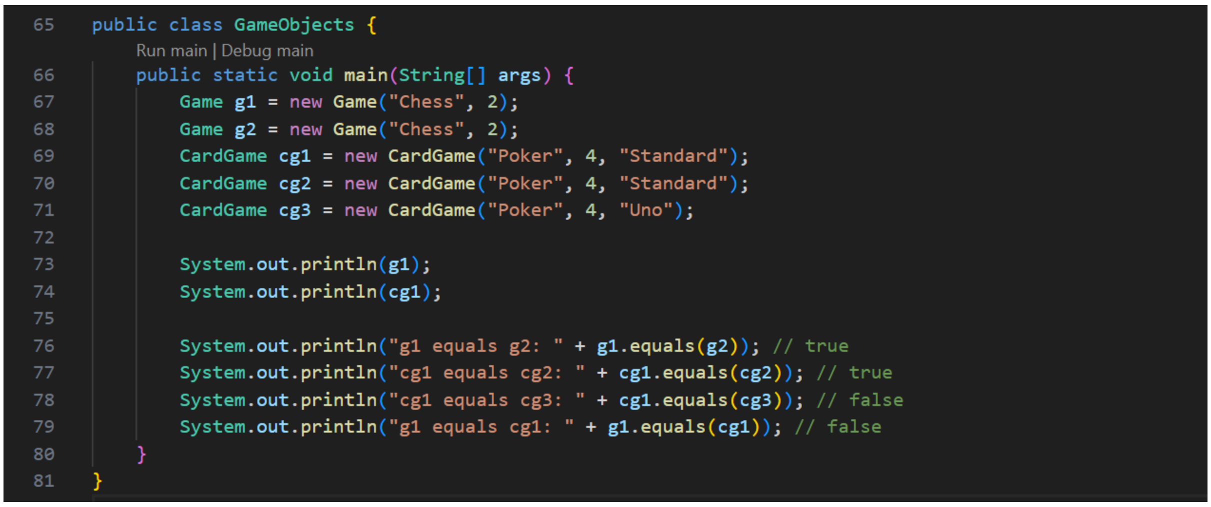
PROGRAM→



```java
/*
 * Create Game class overriding toString() and equals(). Create CardGame extending Game
and override these methods properly.
Hints:
• Override toString(), equals(), and hashCode()
• Call super.toString() in child class override
• Test equality between objects
 */

class Game {
    String name;
    int players;

    Game(String name, int players) {
        this.name = name;
        this.players = players;
    }

    @Override
    public String toString() {
        return "Game{name='" + name + "', players=" + players + "}";
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Game game = (Game) obj;
        return players == game.players && name.equals(game.name);
    }

```

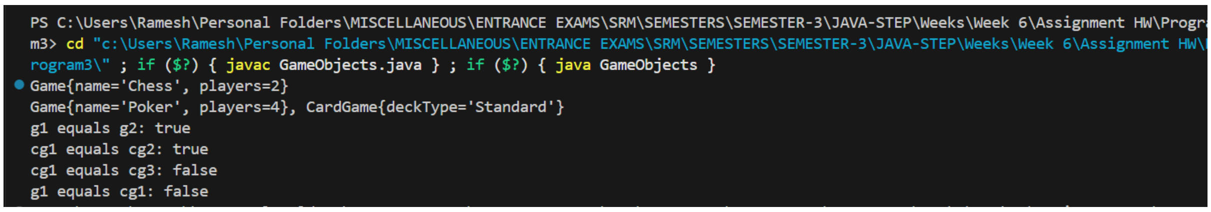


```java
    @Override
    public int hashCode() {
        return name.hashCode() * 31 + players;
    }
}

class CardGame extends Game {
    String deckType;

    CardGame(String name, int players, String deckType) {
        super(name, players);
        this.deckType = deckType;
    }

    @Override
    public String toString() {
        return super.toString() + ", CardGame{deckType='" + deckType + "'}";
    }

    @Override
    public boolean equals(Object obj) {
        if (!super.equals(obj)) return false;
        if (getClass() != obj.getClass()) return false;
        CardGame that = (CardGame) obj;
        return deckType.equals(that.deckType);
    }

    @Override
    public int hashCode() {
        return super.hashCode() * 31 + deckType.hashCode();
    }
}
```



```java
public class GameObjects {
    public static void main(String[] args) {
        Game g1 = new Game("Chess", 2);
        Game g2 = new Game("Chess", 2);
        CardGame cg1 = new CardGame("Poker", 4, "Standard");
        CardGame cg2 = new CardGame("Poker", 4, "Standard");
        CardGame cg3 = new CardGame("Poker", 4, "Uno");

        System.out.println(g1);
        System.out.println(cg1);

        System.out.println("g1 equals g2: " + g1.equals(g2)); // true
        System.out.println("cg1 equals cg2: " + cg1.equals(cg2)); // true
        System.out.println("cg1 equals cg3: " + cg1.equals(cg3)); // false
        System.out.println("g1 equals cg1: " + g1.equals(cg1)); // false
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignment HW\Progra
m3> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignment HW\P
rogram3\" ; if ($?) { javac GameObjects.java } ; if ($?) { java GameObjects }
Game{name='Chess', players=2}
Game{name='Poker', players=4}, CardGame{deckType='Standard'}
g1 equals g2: true
cg1 equals cg2: true
cg1 equals cg3: false
g1 equals cg1: false
```

QNO 4→

Problem Statement:

Create Food class with template method prepare() that calls wash(), cook(), serve().

Create Pizza and Soup with different implementations.

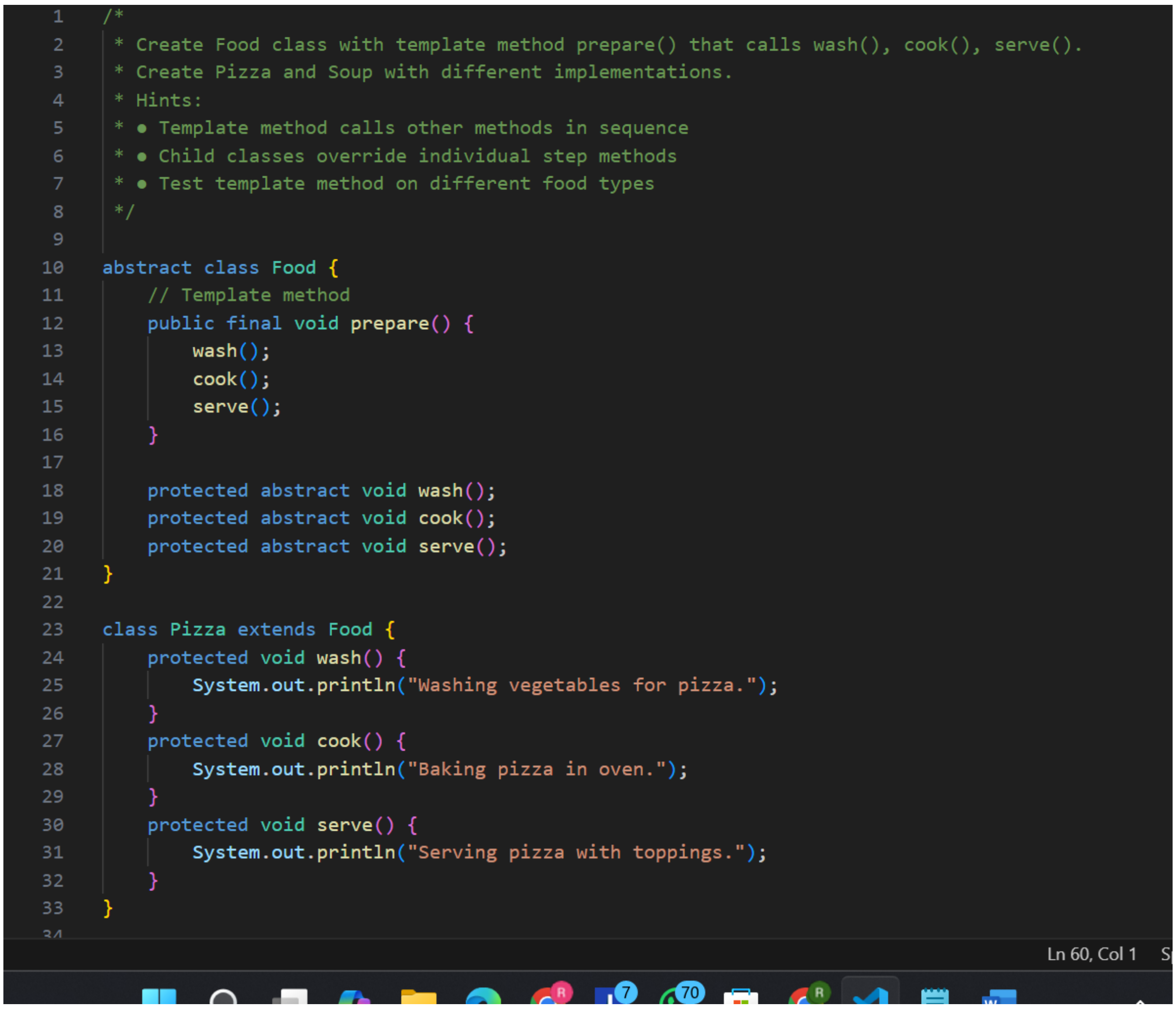
Hints:

● Template method calls other methods in sequence

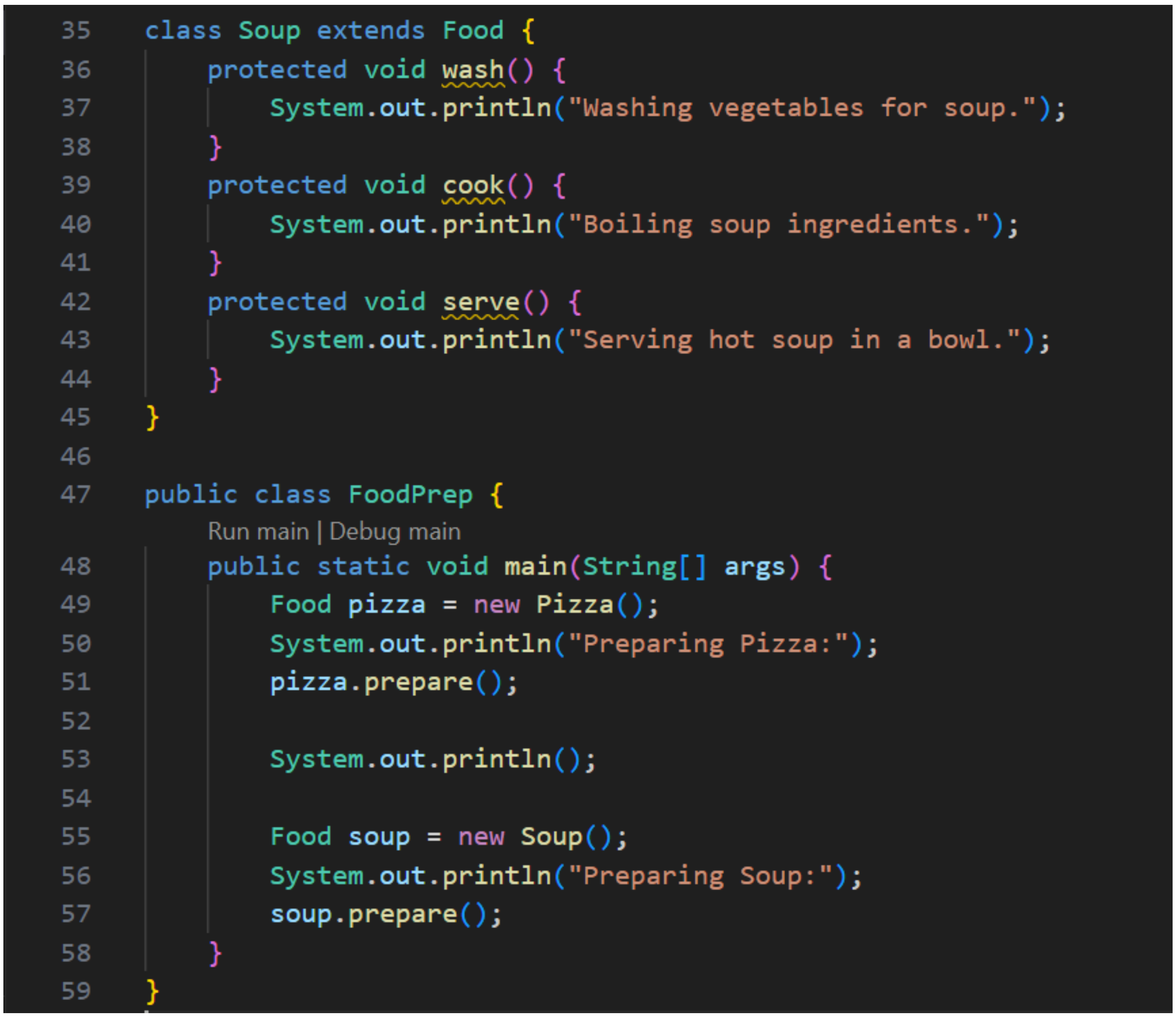
● Child classes override individual step methods

● Test template method on different food types

PROGRAM→

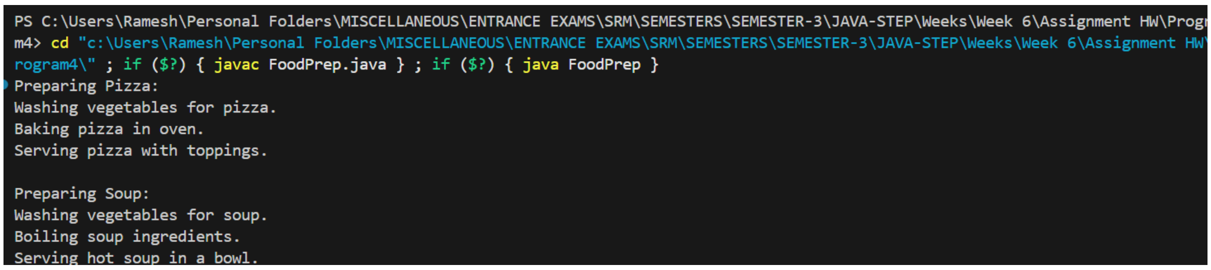


```java
/*
 * Create Food class with template method prepare() that calls wash(), cook(), serve().
 * Create Pizza and Soup with different implementations.
 * Hints:
 * • Template method calls other methods in sequence
 * • Child classes override individual step methods
 * • Test template method on different food types
 */

abstract class Food {
    // Template method
    public final void prepare() {
        wash();
        cook();
        serve();
    }

    protected abstract void wash();
    protected abstract void cook();
    protected abstract void serve();
}

class Pizza extends Food {
    protected void wash() {
        System.out.println("Washing vegetables for pizza.");
    }
    protected void cook() {
        System.out.println("Baking pizza in oven.");
    }
    protected void serve() {
        System.out.println("Serving pizza with toppings.");
    }
}
```



```java
class Soup extends Food {
    protected void wash() {
        System.out.println("Washing vegetables for soup.");
    }
    protected void cook() {
        System.out.println("Boiling soup ingredients.");
    }
    protected void serve() {
        System.out.println("Serving hot soup in a bowl.");
    }
}

public class FoodPrep {
    public static void main(String[] args) {
        Food pizza = new Pizza();
        System.out.println("Preparing Pizza:");
        pizza.prepare();

        System.out.println();

        Food soup = new Soup();
        System.out.println("Preparing Soup:");
        soup.prepare();
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignment HW\Progr
m4> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignment HW\
rogram4\" ; if ($?) { javac FoodPrep.java } ; if ($?) { java FoodPrep }
Preparing Pizza:
Washing vegetables for pizza.
Baking pizza in oven.
Serving pizza with toppings.

Preparing Soup:
Washing vegetables for soup.
Boiling soup ingredients.
Serving hot soup in a bowl.
```

QNO5→

Problem Statement:

Create BasicMath with overloaded calculate() methods. Create AdvancedMath

extending it and adding more overloaded methods.

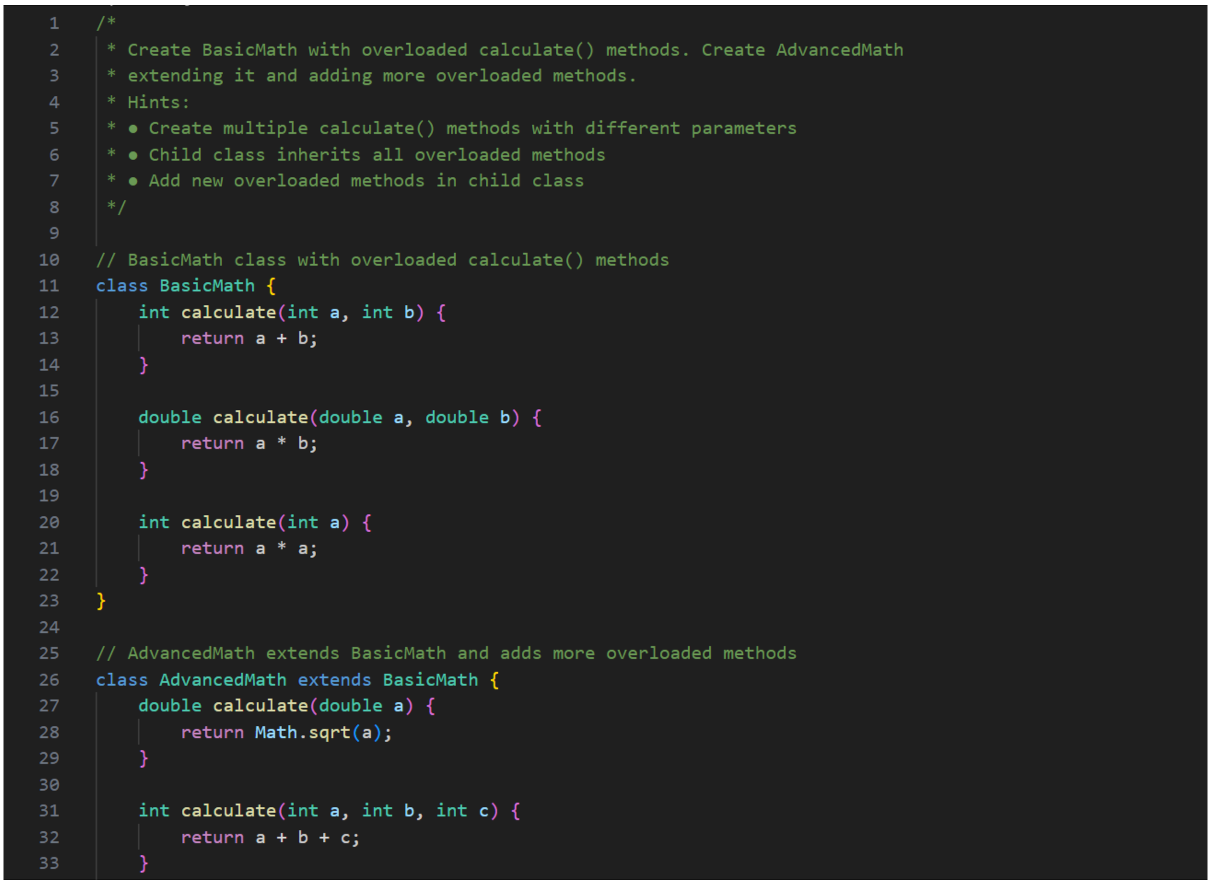
Hints:

● Create multiple calculate() methods with different parameters

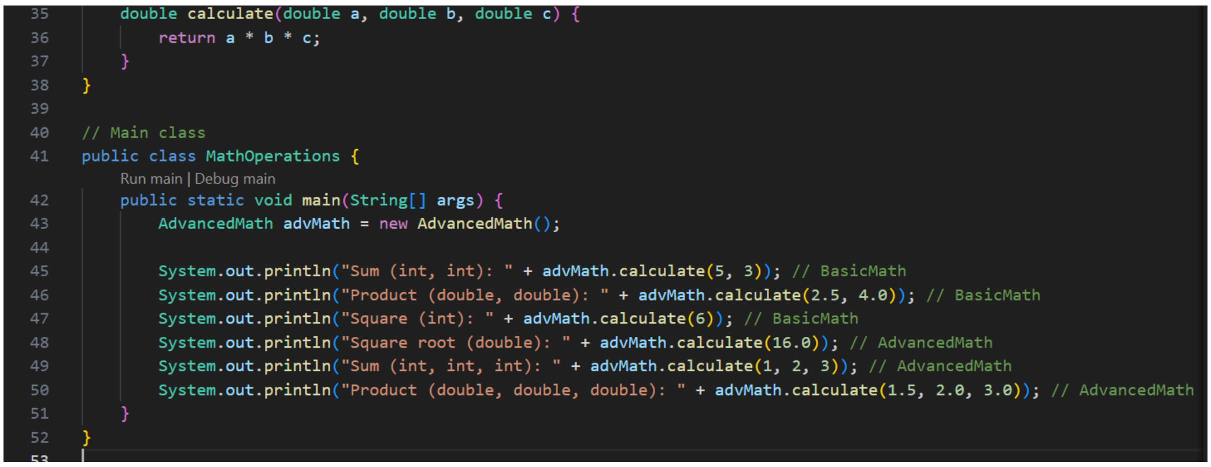
● Child class inherits all overloaded methods

● Add new overloaded methods in child class

PROGRAM→

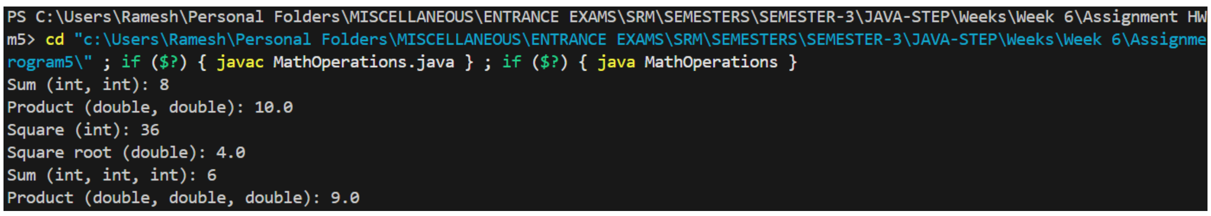


```java
/*
 * Create BasicMath with overloaded calculate() methods. Create AdvancedMath
 * extending it and adding more overloaded methods.
 * Hints:
 * • Create multiple calculate() methods with different parameters
 * • Child class inherits all overloaded methods
 * • Add new overloaded methods in child class
 */

// BasicMath class with overloaded calculate() methods
class BasicMath {
    int calculate(int a, int b) {
        return a + b;
    }

    double calculate(double a, double b) {
        return a * b;
    }

    int calculate(int a) {
        return a * a;
    }
}

// AdvancedMath extends BasicMath and adds more overloaded methods
class AdvancedMath extends BasicMath {
    double calculate(double a) {
        return Math.sqrt(a);
    }

    int calculate(int a, int b, int c) {
        return a + b + c;
    }
```



```java
    double calculate(double a, double b, double c) {
        return a * b * c;
    }
}

// Main class
public class MathOperations {
    public static void main(String[] args) {
        AdvancedMath advMath = new AdvancedMath();

        System.out.println("Sum (int, int): " + advMath.calculate(5, 3)); // BasicMath
        System.out.println("Product (double, double): " + advMath.calculate(2.5, 4.0)); // BasicMath
        System.out.println("Square (int): " + advMath.calculate(6)); // BasicMath
        System.out.println("Square root (double): " + advMath.calculate(16.0)); // AdvancedMath
        System.out.println("Sum (int, int, int): " + advMath.calculate(1, 2, 3)); // AdvancedMath
        System.out.println("Product (double, double, double): " + advMath.calculate(1.5, 2.0, 3.0)); // AdvancedMath
    }
}
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignment HW
m5> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignme
rogram5\" ; if ($?) { javac MathOperations.java } ; if ($?) { java MathOperations }
Sum (int, int): 8
Product (double, double): 10.0
Square (int): 36
Square root (double): 4.0
Sum (int, int, int): 6
Product (double, double, double): 9.0
```

QNO 6→

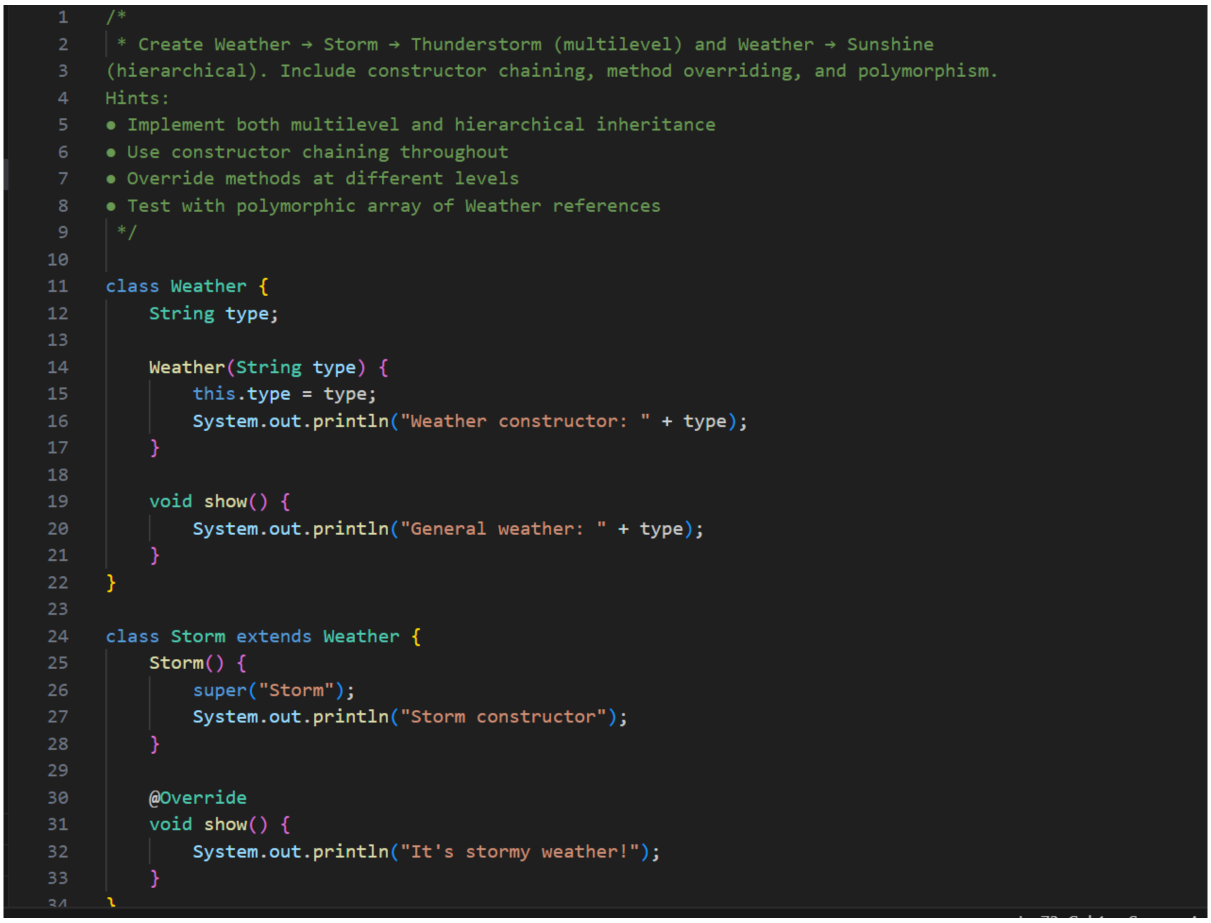
Problem Statement:

Create Weather → Storm → Thunderstorm (multilevel) and Weather → Sunshine (hierarchical). Include constructor chaining, method overriding, and polymorphism.

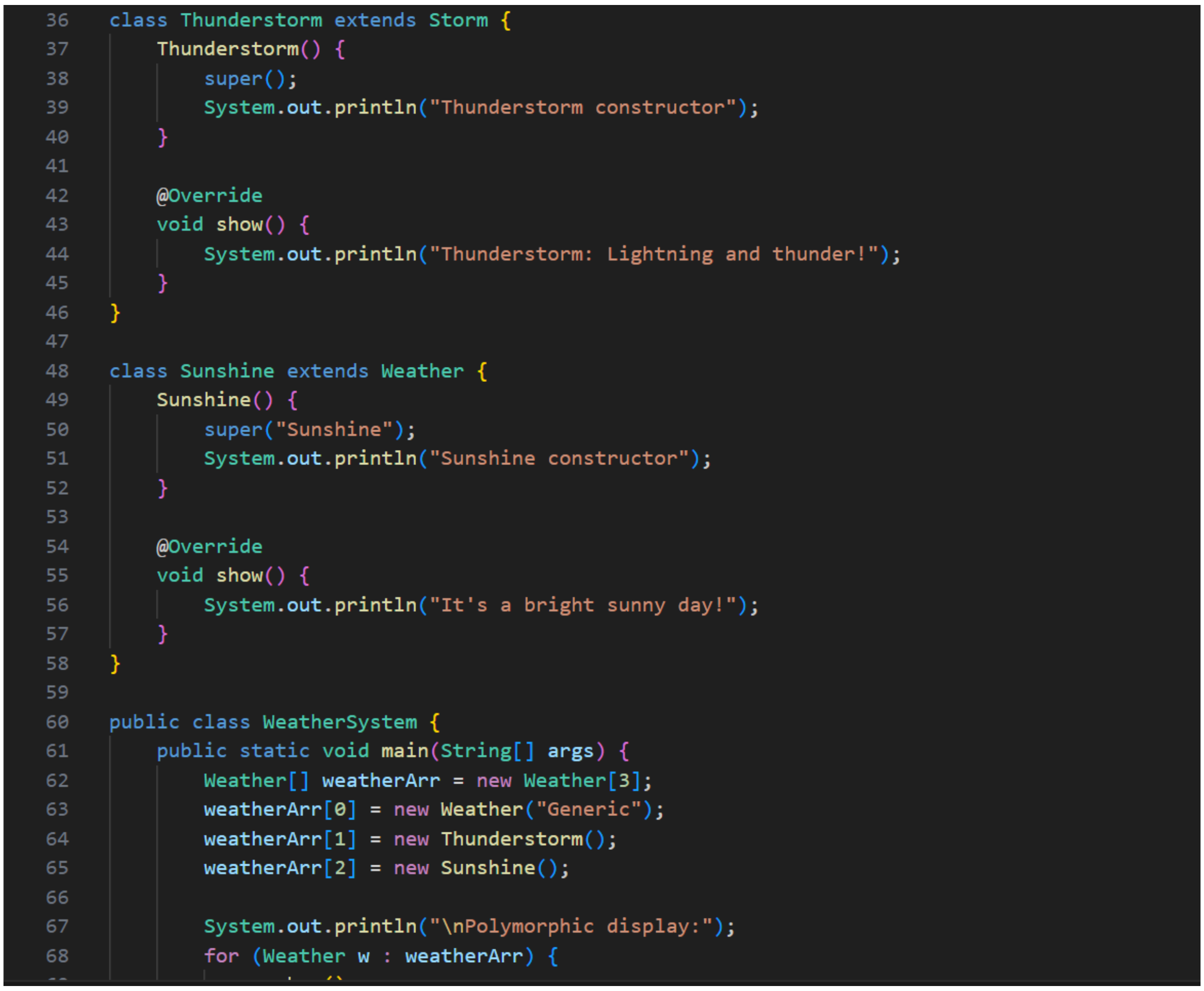
Hints:

- **Implement both multilevel and hierarchical inheritance**
- **Use constructor chaining throughout**
- **Override methods at different levels**
- **Test with polymorphic array of Weather references**

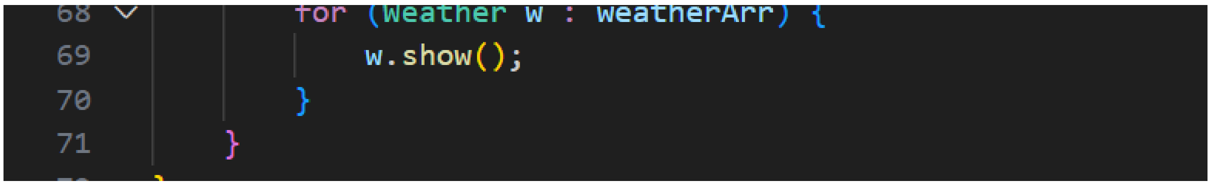
PROGRAM→



```java
/*
 * Create Weather → Storm → Thunderstorm (multilevel) and Weather → Sunshine
(hierarchical). Include constructor chaining, method overriding, and polymorphism.
Hints:
• Implement both multilevel and hierarchical inheritance
• Use constructor chaining throughout
• Override methods at different levels
• Test with polymorphic array of Weather references
 */

class Weather {
    String type;

    Weather(String type) {
        this.type = type;
        System.out.println("Weather constructor: " + type);
    }

    void show() {
        System.out.println("General weather: " + type);
    }
}

class Storm extends Weather {
    Storm() {
        super("Storm");
        System.out.println("Storm constructor");
    }

    @Override
    void show() {
        System.out.println("It's stormy weather!");
    }
}
```

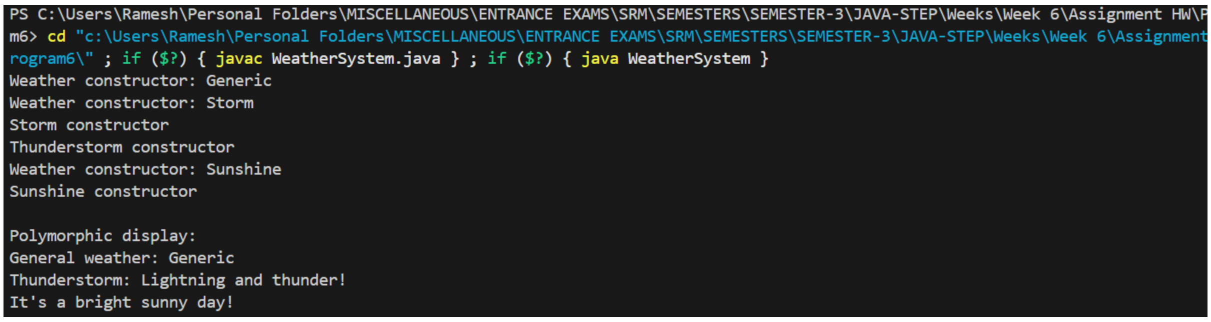


```java
class Thunderstorm extends Storm {
    Thunderstorm() {
        super();
        System.out.println("Thunderstorm constructor");
    }

    @Override
    void show() {
        System.out.println("Thunderstorm: Lightning and thunder!");
    }
}

class Sunshine extends Weather {
    Sunshine() {
        super("Sunshine");
        System.out.println("Sunshine constructor");
    }

    @Override
    void show() {
        System.out.println("It's a bright sunny day!");
    }
}

public class WeatherSystem {
    public static void main(String[] args) {
        Weather[] weatherArr = new Weather[3];
        weatherArr[0] = new Weather("Generic");
        weatherArr[1] = new Thunderstorm();
        weatherArr[2] = new Sunshine();

        System.out.println("\nPolymorphic display:");
        for (Weather w : weatherArr) {
```



```java
        for (Weather w : weatherArr) {
            w.show();
        }
    }
```

OUTPUT→



```
PS C:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignment HW\P
m6> cd "c:\Users\Ramesh\Personal Folders\MISCELLANEOUS\ENTRANCE EXAMS\SRM\SEMESTERS\SEMESTER-3\JAVA-STEP\Weeks\Week 6\Assignment
rogram6\" ; if ($?) { javac WeatherSystem.java } ; if ($?) { java WeatherSystem }
Weather constructor: Generic
Weather constructor: Storm
Storm constructor
Thunderstorm constructor
Weather constructor: Sunshine
Sunshine constructor

Polymorphic display:
General weather: Generic
Thunderstorm: Lightning and thunder!
It's a bright sunny day!
```