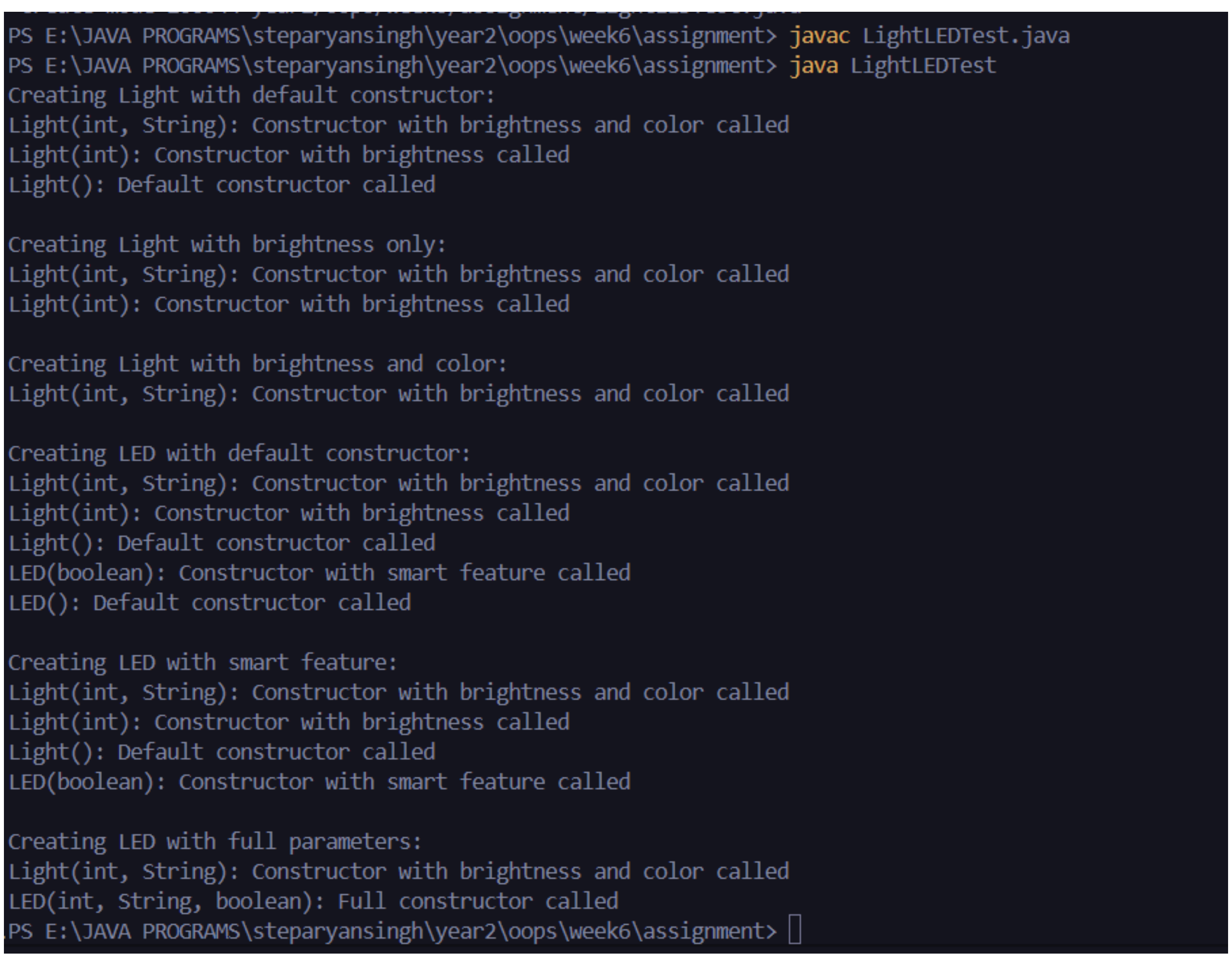
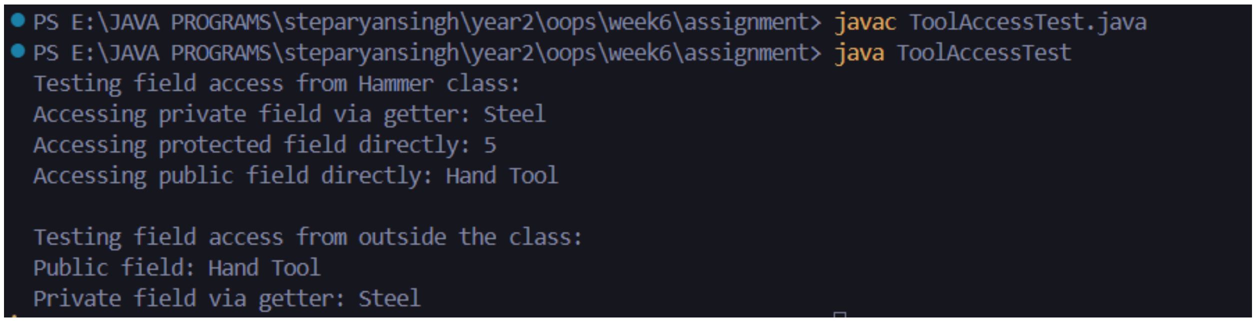
```
// HW PROBLEM 1: Light and LED Multiple Constructors
// Topic: Constructor Chaining with this() and super()
// Problem Statement:
// Create Light class with multiple constructors using this(). Create LED
class with
// constructors using both this() and super().
// Hints:
// ● Chain constructors using this() in same class
// ● Chain to parent using super() from child class
// ● Add print statements to trace constructor calls


public class LightLEDTest {

    // Light class with constructor chaining using this()
    static class Light {
        private int brightness;
        private String color;

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

        public Light(int brightness) {
            this(brightness, "White"); // Default color
            System.out.println("Light(int): Constructor with brightness
called");
        }

        public Light(int brightness, String color) {
            this.brightness = brightness;
            this.color = color;
            System.out.println("Light(int, String): Constructor with
brightness and color called");
        }
    }

    // LED class extending Light, using both this() and super()
    static class LED extends Light {
        private boolean isSmart;

        public LED() {
            this(false); // Default smart feature
            System.out.println("LED(): Default constructor called");
        }

        public LED(boolean isSmart) {
            super(); // Call Light's default constructor
            this.isSmart = isSmart;
            System.out.println("LED(boolean): Constructor with smart
feature called");
        }

        public LED(int brightness, String color, boolean isSmart) {
            super(brightness, color); // Call Light's parameterized
constructor
            this.isSmart = isSmart;
            System.out.println("LED(int, String, boolean): Full
constructor called");
        }
    }

    // Main method to test constructor chaining
    public static void main(String[] args) {
        System.out.println("Creating Light with default constructor:");
        Light light1 = new Light();

        System.out.println("\nCreating Light with brightness only:");
        Light light2 = new Light(75);

        System.out.println("\nCreating Light with brightness and color:");
        Light light3 = new Light(60, "Blue");

        System.out.println("\nCreating LED with default constructor:");
        LED led1 = new LED();

        System.out.println("\nCreating LED with smart feature:");
        LED led2 = new LED(true);
        System.out.println("\nCreating LED with full parameters:");
        LED led3 = new LED(80, "Red", true);
    }
}
```



```
PS E:\JAVA PROGRAMS\stepayansingh\year2\oops\week6\assignment> javac LightLEDTest.java
PS E:\JAVA PROGRAMS\stepayansingh\year2\oops\week6\assignment> java LightLEDTest
Creating Light with default constructor:
Light(int, String): Constructor with brightness and color called
Light(int): Constructor with brightness called
Light(): Default constructor called

Creating Light with brightness only:
Light(int, String): Constructor with brightness and color called
Light(int): Constructor with brightness called

Creating Light with brightness and color:
Light(int, String): Constructor with brightness and color called

Creating LED with default constructor:
Light(int, String): Constructor with brightness and color called
Light(int): Constructor with brightness called
Light(): Default constructor called
LED(boolean): Constructor with smart feature called
LED(): Default constructor called

Creating LED with smart feature:
Light(int, String): Constructor with brightness and color called
Light(int): Constructor with brightness called
Light(): Default constructor called
LED(boolean): Constructor with smart feature called

Creating LED with full parameters:
Light(int, String): Constructor with brightness and color called
LED(int, String, boolean): Full constructor called
PS E:\JAVA PROGRAMS\stepayansingh\year2\oops\week6\assignment>
```

```
// HW PROBLEM 2: Tool Access Levels
// Topic: Access Modifiers in Inheritance
// Problem Statement:
// Create Tool class with private, protected, and public fields. Create
Hammer class and
// test field accessibility.
```

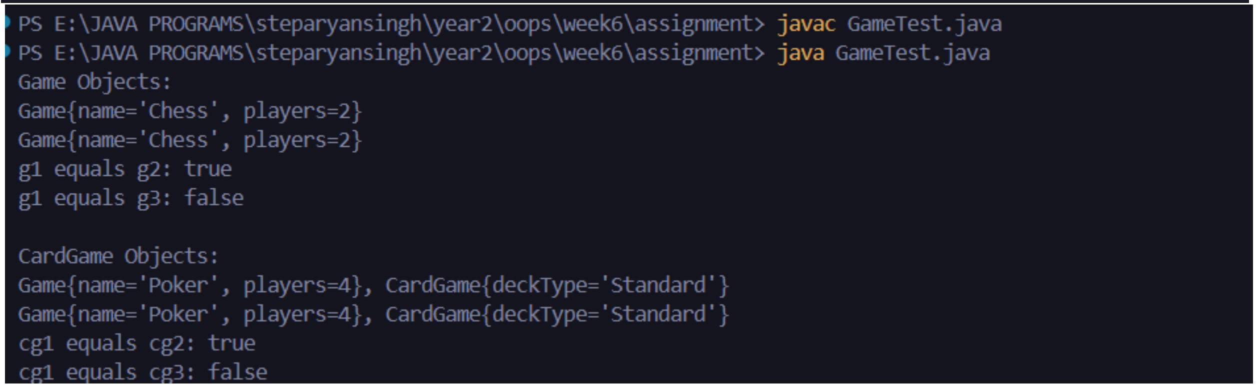
```java
// Hints:
// ● Try accessing different access level fields from child
// ● Use getters for private fields
// ● Document which fields are directly accessible



public class ToolAccessTest {

    // Tool class with private, protected, and public fields
    static class Tool {
        private String privateMaterial = "Steel";
        protected int protectedWeight = 5;
        public String publicType = "Hand Tool";

        // Getter for private field
        public String getPrivateMaterial() {
            return privateMaterial;
        }
    }

    // Hammer class extending Tool
    static class Hammer extends Tool {

        public void testAccess() {
            // System.out.println(privateMaterial); // ❌ Not accessible
directly
            System.out.println("Accessing private field via getter: " +
getPrivateMaterial()); // ✅

            System.out.println("Accessing protected field directly: " +
protectedWeight); // ✅
            System.out.println("Accessing public field directly: " +
publicType); // ✅
        }
    }

    // Main method to test field accessibility
    public static void main(String[] args) {
        Hammer hammer = new Hammer();
        System.out.println("Testing field access from Hammer class:");
        hammer.testAccess();

        System.out.println("\nTesting field access from outside the class:");
        System.out.println("Public field: " + hammer.publicType); // ✅
        // System.out.println("Protected field: " + hammer.protectedWeight); // ⚠️ Accessible only within same package or subclass
        // System.out.println("Private field: " + hammer.privateMaterial); // ❌ Not accessible
        System.out.println("Private field via getter: " + hammer.getPrivateMaterial()); // ✅
    }
}
```



```
PS E:\JAVA PROGRAMS\stepparyansingh\year2\oops\week6\assignment> javac ToolAccessTest.java
PS E:\JAVA PROGRAMS\stepparyansingh\year2\oops\week6\assignment> java ToolAccessTest
Testing field access from Hammer class:
Accessing private field via getter: Steel
Accessing protected field directly: 5
Accessing public field directly: Hand Tool

Testing field access from outside the class:
Public field: Hand Tool
Private field via getter: Steel
```

```java
// HW PROBLEM 3: Game and Card Game Objects
// Topic: Overriding Object Methods
// Problem Statement:

// 1

// 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
```

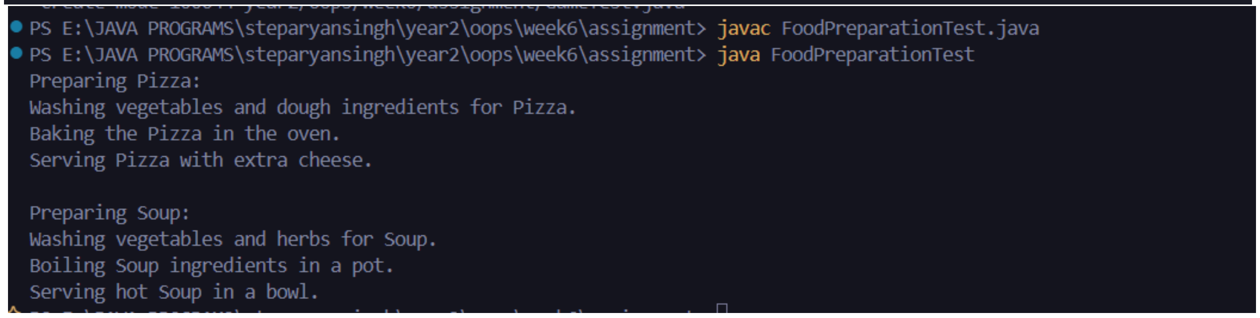
```
import java.util.Objects;

public class GameTest {

    // Base class Game
    static class Game {
        protected String name;
        protected int players;

        public 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 && Objects.equals(name,
game.name);
        }

        @Override
        public int hashCode() {
            return Objects.hash(name, players);
        }
    }

    // Subclass CardGame
    static class CardGame extends Game {
        private String deckType;

        public 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 Objects.equals(deckType, that.deckType);
        }

        @Override
        public int hashCode() {
            return Objects.hash(super.hashCode(), deckType);
        }
    }

    // Main method to test equality and string representation
    public static void main(String[] args) {
        Game g1 = new Game("Chess", 2);
        Game g2 = new Game("Chess", 2);
        Game g3 = new Game("Monopoly", 4);

        CardGame cg1 = new CardGame("Poker", 4, "Standard");
        CardGame cg2 = new CardGame("Poker", 4, "Standard");
        CardGame cg3 = new CardGame("Poker", 4, "Custom");

        System.out.println("Game Objects:");
        System.out.println(g1);
        System.out.println(g2);
        System.out.println("g1 equals g2: " + g1.equals(g2));
        System.out.println("g1 equals g3: " + g1.equals(g3));
        System.out.println("\nCardGame Objects:");
        System.out.println(cg1);
        System.out.println(cg2);
        System.out.println("cg1 equals cg2: " + cg1.equals(cg2));
        System.out.println("cg1 equals cg3: " + cg1.equals(cg3));
    }
}
```



```
PS E:\JAVA PROGRAMS\stepraryansingh\year2\oops\week6\assignment> javac GameTest.java
PS E:\JAVA PROGRAMS\stepraryansingh\year2\oops\week6\assignment> java GameTest.java
Game Objects:
Game{name='Chess', players=2}
Game{name='Chess', players=2}
g1 equals g2: true
g1 equals g3: false

CardGame Objects:
Game{name='Poker', players=4}, CardGame{deckType='Standard'}
Game{name='Poker', players=4}, CardGame{deckType='Standard'}
cg1 equals cg2: true
cg1 equals cg3: false
```

```
// HW PROBLEM 4: Food Preparation Template
// Topic: Template Method Pattern
// 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


public class FoodPreparationTest {

    // Abstract base class defining the template method
    static abstract class Food {
        // Template method
        public final void prepare() {
            wash();
            cook();
            serve();
        }

        // Steps to be implemented by subclasses
        protected abstract void wash();
        protected abstract void cook();
        protected abstract void serve();
    }

    // Concrete subclass: Pizza
    static class Pizza extends Food {
        @Override
        protected void wash() {
            System.out.println("Washing vegetables and dough ingredients
for Pizza.");
        }

        @Override
        protected void cook() {
            System.out.println("Baking the Pizza in the oven.");
        }

        @Override
        protected void serve() {
            System.out.println("Serving Pizza with extra cheese.");
        }
    }

    // Concrete subclass: Soup
    static class Soup extends Food {
        @Override
        protected void wash() {
            System.out.println("Washing vegetables and herbs for Soup.");
        }

        @Override
```

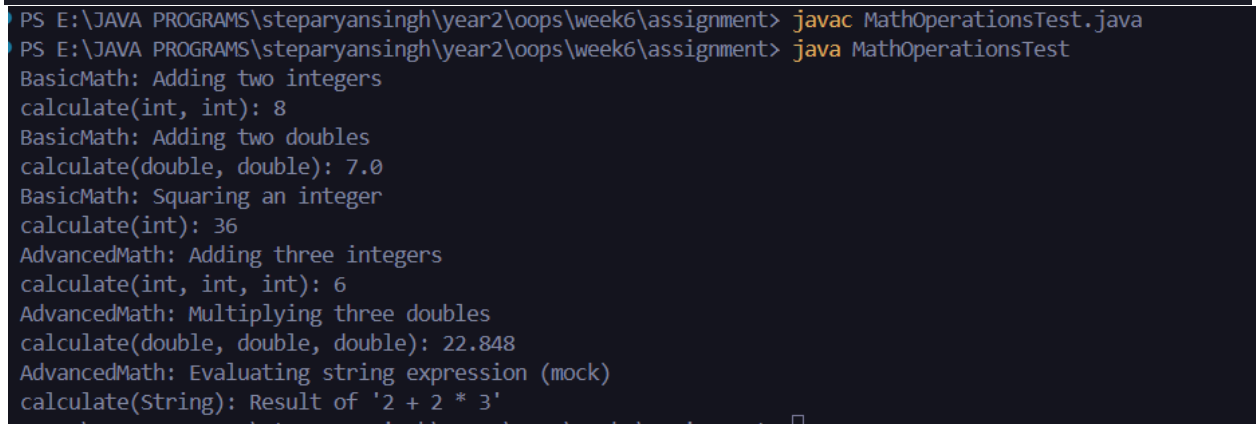
```java
        protected void cook() {
            System.out.println("Boiling Soup ingredients in a pot.");
        }

        @Override
        protected void serve() {
            System.out.println("Serving hot Soup in a bowl.");
        }
    }

    // Main method to test template method
    public static void main(String[] args) {
        System.out.println("Preparing Pizza:");
        Food pizza = new Pizza();
        pizza.prepare();

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



```
PS E:\JAVA PROGRAMS\stepar yansingh\year2\oops\week6\assignment> javac FoodPreparationTest.java
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week6\assignment> java FoodPreparationTest
Preparing Pizza:
Washing vegetables and dough ingredients for Pizza.
Baking the Pizza in the oven.
Serving Pizza with extra cheese.

Preparing Soup:
Washing vegetables and herbs for Soup.
Boiling Soup ingredients in a pot.
Serving hot Soup in a bowl.
```

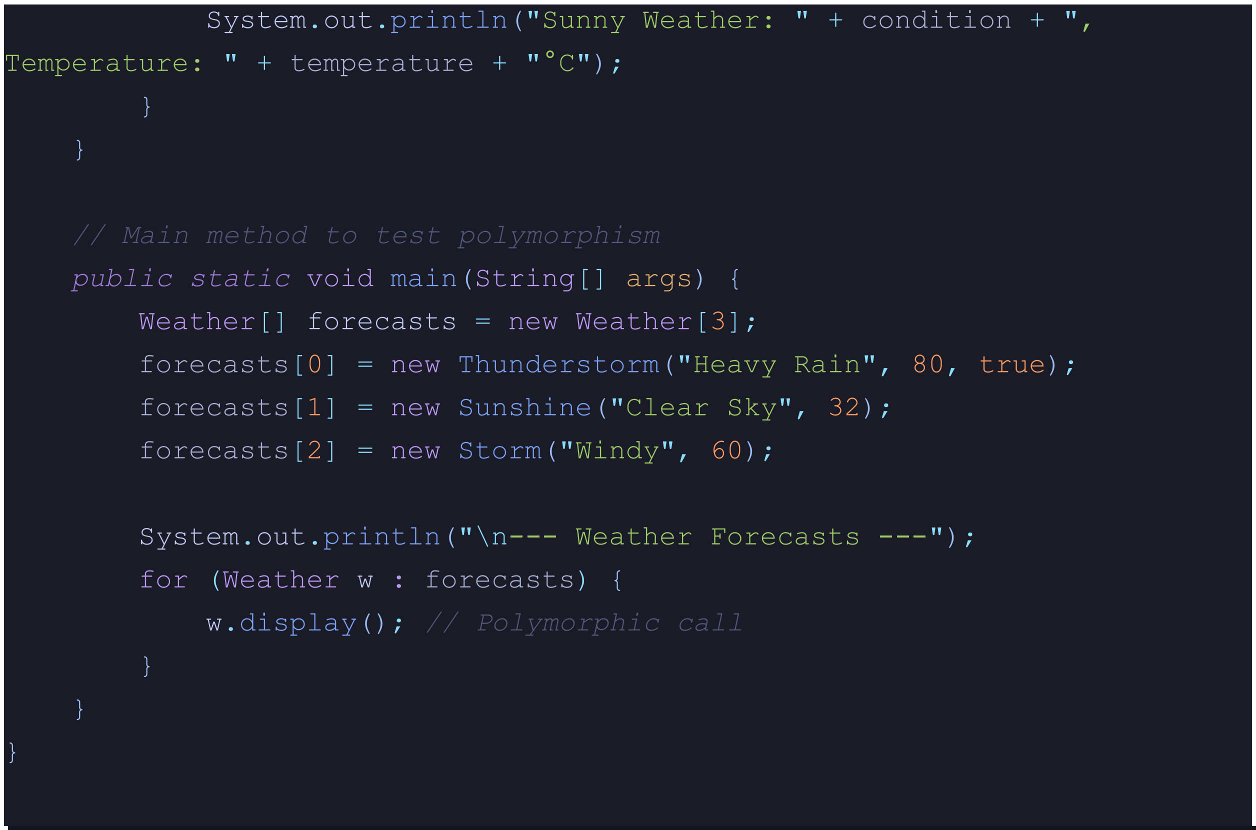
```java
// HW PROBLEM 5: Math Operations Inheritance
// Topic: Inheritance with Method Overloading
// Problem Statement:
// Create BasicMath with overloaded calculate() methods. Create
AdvancedMath
// extending it and adding more overloaded methods.
// Hints:
```

```java
// • Create multiple calculate() methods with different parameters

// 2

// • Child class inherits all overloaded methods
// • Add new overloaded methods in child class



public class MathOperationsTest {

    // Base class with overloaded calculate() methods
    static class BasicMath {
        public int calculate(int a, int b) {
            System.out.println("BasicMath: Adding two integers");
            return a + b;
        }

        public double calculate(double a, double b) {
            System.out.println("BasicMath: Adding two doubles");
            return a + b;
        }

        public int calculate(int a) {
            System.out.println("BasicMath: Squaring an integer");
            return a * a;
        }
    }

    // Subclass with additional overloaded calculate() methods
    static class AdvancedMath extends BasicMath {
        public int calculate(int a, int b, int c) {
            System.out.println("AdvancedMath: Adding three integers");
            return a + b + c;
        }

        public double calculate(double a, double b, double c) {
            System.out.println("AdvancedMath: Multiplying three doubles");
            return a * b * c;
        }
        public String calculate(String expression) {
            System.out.println("AdvancedMath: Evaluating string expression
(mock)");
            return "Result of '" + expression + "'";
        }
    }

    // Main method to test all overloaded methods
    public static void main(String[] args) {
        AdvancedMath math = new AdvancedMath();

        System.out.println("calculate(int, int): " + math.calculate(5,
3));
        System.out.println("calculate(double, double): " +
math.calculate(2.5, 4.5));
        System.out.println("calculate(int): " + math.calculate(6));
        System.out.println("calculate(int, int, int): " +
math.calculate(1, 2, 3));
        System.out.println("calculate(double, double, double): " +
math.calculate(1.2, 3.4, 5.6));
        System.out.println("calculate(String): " + math.calculate("2 + 2 *
3"));
    }
}
```

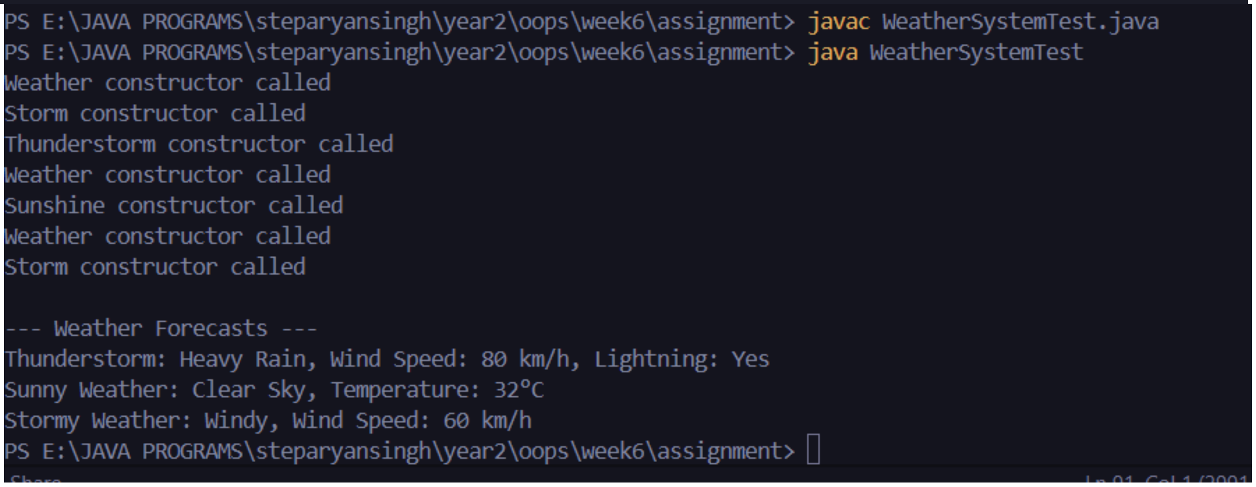


```
PS E:\JAVA PROGRAMS\stepar yansingh\year2\oops\week6\assignment> javac MathOperationsTest.java
PS E:\JAVA PROGRAMS\steparyansingh\year2\oops\week6\assignment> java MathOperationsTest
BasicMath: Adding two integers
calculate(int, int): 8
BasicMath: Adding two doubles
calculate(double, double): 7.0
BasicMath: Squaring an integer
calculate(int): 36
AdvancedMath: Adding three integers
calculate(int, int, int): 6
AdvancedMath: Multiplying three doubles
calculate(double, double, double): 22.848
AdvancedMath: Evaluating string expression (mock)
calculate(String): Result of '2 + 2 * 3'
```

```
// HW PROBLEM 6: Weather System Hierarchy
// Topic: Complete Inheritance Implementation
// 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



public class WeatherSystemTest {

    // Base class
    static class Weather {
        protected String condition;

        public Weather(String condition) {
            this.condition = condition;
            System.out.println("Weather constructor called");
        }

        public void display() {
            System.out.println("General Weather: " + condition);
        }
    }

    // Multilevel: Weather → Storm → Thunderstorm
    static class Storm extends Weather {
        protected int windSpeed;

        public Storm(String condition, int windSpeed) {
            super(condition);
            this.windSpeed = windSpeed;
            System.out.println("Storm constructor called");
        }
        @Override
        public void display() {
            System.out.println("Stormy Weather: " + condition + ", Wind
Speed: " + windSpeed + " km/h");
        }
    }

    static class Thunderstorm extends Storm {
        private boolean hasLightning;

        public Thunderstorm(String condition, int windSpeed, boolean
hasLightning) {
            super(condition, windSpeed);
            this.hasLightning = hasLightning;
            System.out.println("Thunderstorm constructor called");
        }

        @Override
        public void display() {
            System.out.println("Thunderstorm: " + condition + ", Wind
Speed: " + windSpeed +
                               " km/h, Lightning: " + (hasLightning ?
"Yes" : "No"));
        }
    }

    // Hierarchical: Weather → Sunshine
    static class Sunshine extends Weather {
        private int temperature;

        public Sunshine(String condition, int temperature) {
            super(condition);
            this.temperature = temperature;
            System.out.println("Sunshine constructor called");
        }

        @Override
        public void display() {
```



```java
            System.out.println("Sunny Weather: " + condition + ",
Temperature: " + temperature + "°C");
        }
    }

    // Main method to test polymorphism
    public static void main(String[] args) {
        Weather[] forecasts = new Weather[3];
        forecasts[0] = new Thunderstorm("Heavy Rain", 80, true);
        forecasts[1] = new Sunshine("Clear Sky", 32);
        forecasts[2] = new Storm("Windy", 60);

        System.out.println("\n--- Weather Forecasts ---");
        for (Weather w : forecasts) {
            w.display(); // Polymorphic call
        }
    }
}
```



```
PS E:\JAVA PROGRAMS\stepARYANSINGH\year2\oops\week6\assignment> javac WeatherSystemTest.java
PS E:\JAVA PROGRAMS\stepARYANSINGH\year2\oops\week6\assignment> java WeatherSystemTest
Weather constructor called
Storm constructor called
Thunderstorm constructor called
Weather constructor called
Sunshine constructor called
Weather constructor called
Storm constructor called

--- Weather Forecasts ---
Thunderstorm: Heavy Rain, Wind Speed: 80 km/h, Lightning: Yes
Sunny Weather: Clear Sky, Temperature: 32°C
Stormy Weather: Windy, Wind Speed: 60 km/h
PS E:\JAVA PROGRAMS\stepARYANSINGH\year2\oops\week6\assignment>
```