**WEEK 6 – HW**

Q1. Light and LED Multiple Constructors

Soln.

class Light {

protected String type;

protected int wattage;

Light() {

this("Generic Light", 0);

System.out.println("Light default constructor called");

}

Light(String type) {

this(type, 60);

System.out.println("Light single-parameter constructor called");

}

Light(String type, int wattage) {

this.type = type;

this.wattage = wattage;

System.out.println("Light two-parameter constructor called");

}

void showDetails() {

System.out.println("Type: " + type + ", Wattage: " + wattage);

}

}

class LED extends Light {

private String color;

LED() {

this("LED", 10, "White");

System.out.println("LED default constructor called");

}

LED(String type, int wattage) {

super(type, wattage);

System.out.println("LED two-parameter constructor called");

}

LED(String type, int wattage, String color) {

super(type, wattage);

this.color = color;

System.out.println("LED three-parameter constructor called");

}

@Override

void showDetails() {

super.showDetails();

System.out.println("Color: " + color);

}

}

public class Main {

public static void main(String[] args) {

System.out.println("Creating Light objects:");

Light l1 = new Light();

l1.showDetails();

System.out.println();

Light l2 = new Light("Halogen");

l2.showDetails();

System.out.println();

Light l3 = new Light("CFL", 25);

l3.showDetails();

System.out.println("\nCreating LED objects:");

LED led1 = new LED();

led1.showDetails();

System.out.println();

LED led2 = new LED("LED", 15);

led2.showDetails();

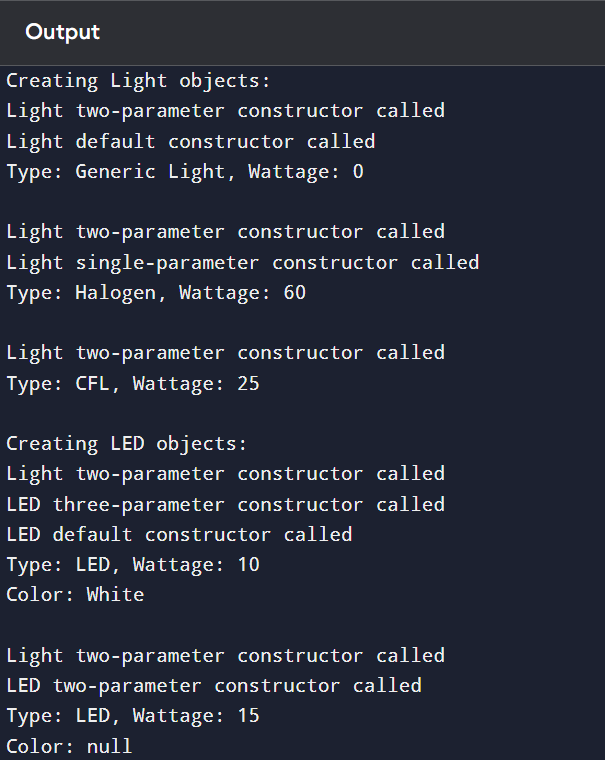
System.out.println();

LED led3 = new LED("LED", 20, "Blue");

led3.showDetails();

}

}



Q2. Tool Access Levels

Soln.

class Tool {

private String type;

protected String material;

public String brand;

Tool(String type, String material, String brand) {

this.type = type;

this.material = material;

this.brand = brand;

}

public String getType() {

return type;

}

}

class Hammer extends Tool {

private double weight;

Hammer(String type, String material, String brand, double weight) {

super(type, material, brand);

this.weight = weight;

}

void showDetails() {

// System.out.println("Type: " + type);

System.out.println("Type: " + getType());

System.out.println("Material: " + material);

System.out.println("Brand: " + brand);

System.out.println("Weight: " + weight);

}

}

public class Main {

public static void main(String[] args) {

Hammer h = new Hammer("Hammer", "Steel", "Stanley", 2.5);

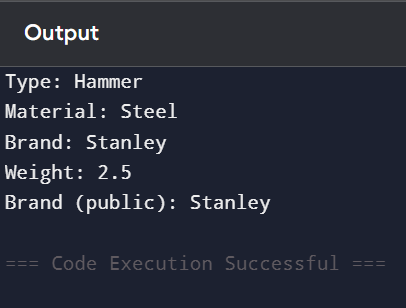
h.showDetails();

outside package without inheritance

System.out.println("Brand (public): " + h.brand);

}

}



Q3. Game and Card Game Objects

Soln.

class Game {

protected String name;

protected 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);

}

@Override

public int hashCode() {

return name.hashCode() + players;

}

}

class CardGame extends Game {

private 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 (this.getClass() != obj.getClass()) return false;

CardGame cardGame = (CardGame) obj;

return deckType.equals(cardGame.deckType);

}

@Override

public int hashCode() {

return super.hashCode() + deckType.hashCode();

}

}

public class Main {

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, "Custom");

System.out.println("Testing toString():");

System.out.println(g1);

System.out.println(cg1);

System.out.println("\nTesting equals():");

System.out.println("g1 equals g2: " + g1.equals(g2));

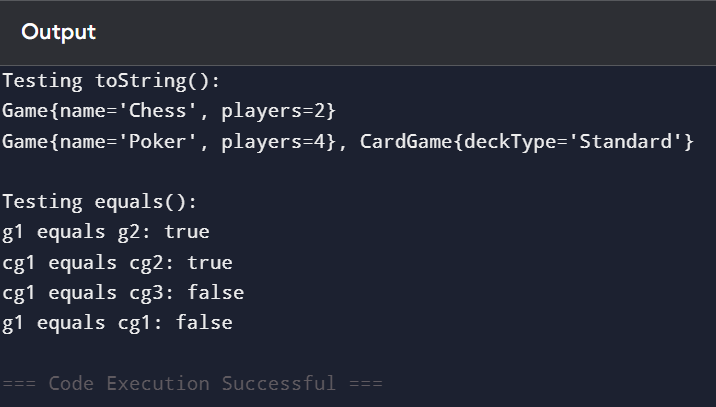
System.out.println("cg1 equals cg2: " + cg1.equals(cg2));

System.out.println("cg1 equals cg3: " + cg1.equals(cg3));

System.out.println("g1 equals cg1: " + g1.equals(cg1));

}

}



Q4. Food Preparation Template

Soln.

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 {

@Override

protected void wash() {

System.out.println("Washing vegetables and 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.");

}

}

class Soup extends Food {

@Override

protected void wash() {

System.out.println("Washing vegetables for soup.");

}

@Override

protected void cook() {

System.out.println("Boiling vegetables to make soup.");

}

@Override

protected void serve() {

System.out.println("Serving hot soup in a bowl.");

}

}

public class Main {

public static void main(String[] args) {

Food pizza = new Pizza();

Food soup = new Soup();

System.out.println("Preparing Pizza:");

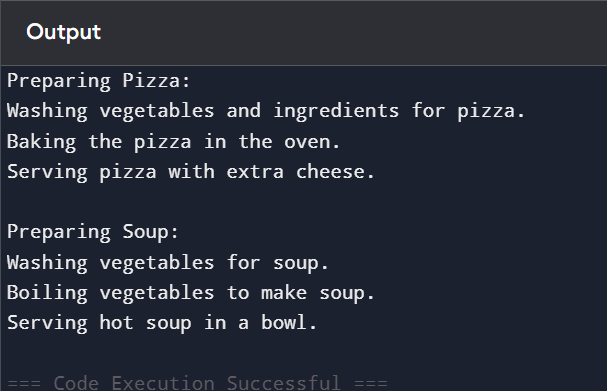
pizza.prepare();

System.out.println("\nPreparing Soup:");

soup.prepare();

}

}



Q5. Math Operations Inheritance

Soln.

class BasicMath {

int calculate(int a, int b) {

return a + b;

}

double calculate(double a, double b) {

return a + b;

}

int calculate(int a, int b, int c) {

return a + b + c;

}

}

class AdvancedMath extends BasicMath {

double calculate(double a, double b, double c) {

return a \* b \* c;

}

double calculate(int a, double b) {

return a \* b;

}

}

public class Main {

public static void main(String[] args) {

BasicMath bm = new BasicMath();

AdvancedMath am = new AdvancedMath();

System.out.println("BasicMath Operations:");

System.out.println("Sum of 2 ints: " + bm.calculate(5, 10));

System.out.println("Sum of 2 doubles: " + bm.calculate(5.5, 4.5));

System.out.println("Sum of 3 ints: " + bm.calculate(1, 2, 3));

System.out.println("\nAdvancedMath Operations:");

System.out.println("Product of 3 doubles: " + am.calculate(2.0, 3.0, 4.0));

System.out.println("Product of int and double: " + am.calculate(5, 2.5));

System.out.println("\nUsing inherited methods from BasicMath in AdvancedMath:");

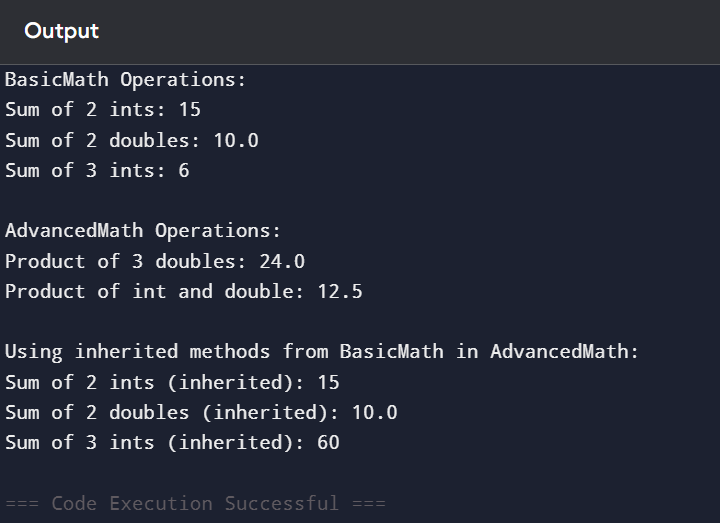
System.out.println("Sum of 2 ints (inherited): " + am.calculate(7, 8));

System.out.println("Sum of 2 doubles (inherited): " + am.calculate(3.3, 6.7));

System.out.println("Sum of 3 ints (inherited): " + am.calculate(10, 20, 30));

}

}



Q6. Weather System Hierarchy

Soln.

class Weather {

protected String description;

Weather(String description) {

this.description = description;

System.out.println("Weather constructor called");

}

void showWeather() {

System.out.println("General Weather: " + description);

}

}

class Storm extends Weather {

protected int windSpeed;

Storm(String description, int windSpeed) {

super(description);

this.windSpeed = windSpeed;

System.out.println("Storm constructor called");

}

@Override

void showWeather() {

System.out.println("Storm: " + description + ", Wind Speed: " + windSpeed + " km/h");

}

}

class Thunderstorm extends Storm {

private boolean lightning;

Thunderstorm(String description, int windSpeed, boolean lightning) {

super(description, windSpeed);

this.lightning = lightning;

System.out.println("Thunderstorm constructor called");

}

@Override

void showWeather() {

System.out.println("Thunderstorm: " + description + ", Wind Speed: "

+ windSpeed + " km/h, Lightning: " + lightning);

}

}

class Sunshine extends Weather {

private int temperature;

Sunshine(String description, int temperature) {

super(description);

this.temperature = temperature;

System.out.println("Sunshine constructor called");

}

@Override

void showWeather() {

System.out.println("Sunshine: " + description + ", Temperature: " + temperature + "°C");

}

}

public class Main {

public static void main(String[] args) {

Weather[] forecasts = new Weather[3];

forecasts[0] = new Storm("Heavy Rain", 80);

forecasts[1] = new Thunderstorm("Severe Thunderstorm", 100, true);

forecasts[2] = new Sunshine("Clear Sky", 30);

System.out.println("\nForecasts:");

for (Weather w : forecasts) {

w.showWeather();

}

}

}

