# 🧪 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

class Light {

public Light() {

this("Default Light");

System.out.println("Light no-arg constructor");

}

public Light(String type) {

System.out.println("Light constructor with type: " + type);

}

}

class LED extends Light {

public LED() {

this("Default LED");

System.out.println("LED no-arg constructor");

}

public LED(String ledType) {

super();

System.out.println("LED constructor with ledType: " + ledType);

}

}

public class LightTest {

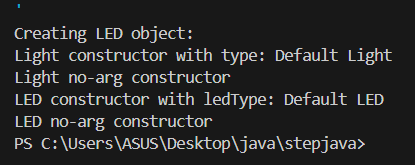
public static void main(String[] args) {

System.out.println("Creating LED object:");

LED led = new LED();

}

}



# 🧪 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.

## 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 Info";

protected String protectedField = "Protected Info";

public String publicField = "Public Info";

public String getPrivateField() {

return privateField;

}

}

class Hammer extends Tool {

public void showAccess() {

// System.out.println(privateField); // Not accessible

System.out.println(protectedField); // Accessible

System.out.println(publicField); // Accessible

System.out.println(getPrivateField()); // Access via getter

}

}

public class ToolTest {

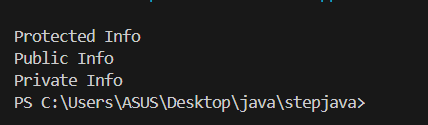
public static void main(String[] args) {

Hammer hammer = new Hammer();

hammer.showAccess();

}

}



# 🧪 HW PROBLEM 3: Game and Card Game Objects

**Topic: Overriding Object Methods**

## 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

class Game {

String name;

public Game(String name) {

this.name = name;

}

@Override

public String toString() {

return "Game: " + name;

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (!(obj instanceof Game)) return false;

Game other = (Game) obj;

return this.name.equals(other.name);

}

@Override

public int hashCode() {

return name.hashCode();

}

}

class CardGame extends Game {

int numberOfCards;

public CardGame(String name, int numberOfCards) {

super(name);

this.numberOfCards = numberOfCards;

}

@Override

public String toString() {

return super.toString() + ", Cards: " + numberOfCards;

}

@Override

public boolean equals(Object obj) {

if (!super.equals(obj)) return false;

if (!(obj instanceof CardGame)) return false;

CardGame other = (CardGame) obj;

return this.numberOfCards == other.numberOfCards;

}

@Override

public int hashCode() {

return super.hashCode() + numberOfCards;

}

}

public class GameTest {

public static void main(String[] args) {

Game game1 = new Game("Chess");

CardGame cardGame1 = new CardGame("Poker", 52);

CardGame cardGame2 = new CardGame("Poker", 52);

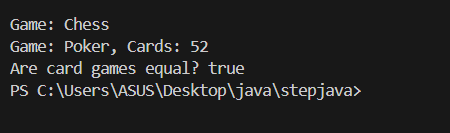
System.out.println(game1.toString());

System.out.println(cardGame1.toString());

System.out.println("Are card games equal? " + cardGame1.equals(cardGame2));

}

}



# 🧪 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

abstract class Food {

public final void prepare() {

wash();

cook();

serve();

}

abstract void wash();

abstract void cook();

abstract void serve();

}

class Pizza extends Food {

void wash() {

System.out.println("Washing ingredients for Pizza");

}

void cook() {

System.out.println("Baking Pizza in the oven");

}

void serve() {

System.out.println("Serving hot Pizza");

}

}

class Soup extends Food {

void wash() {

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

}

void cook() {

System.out.println("Boiling Soup on stove");

}

void serve() {

System.out.println("Serving warm Soup");

}

}

public class FoodTest {

public static void main(String[] args) {

Food pizza = new Pizza();

Food soup = new Soup();

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

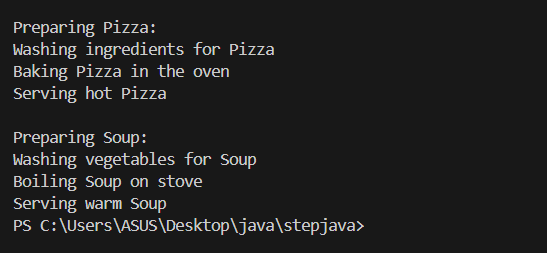
pizza.prepare();

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

soup.prepare();

}

}



# 🧪 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:

* Create multiple calculate() methods with different parameters
* Child class inherits all overloaded methods
* Add new overloaded methods in child class

class BasicMath {

public int calculate(int a, int b) {

return a + b;

}

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

return a + b + c;

}

}

class AdvancedMath extends BasicMath {

public double calculate(double a, double b) {

return a \* b;

}

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

return a \* b \* c;

}

}

public class MathTest {

public static void main(String[] args) {

AdvancedMath math = new AdvancedMath();

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

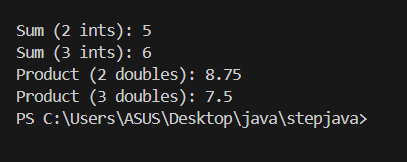
System.out.println("Sum (3 ints): " + math.calculate(1, 2, 3));

System.out.println("Product (2 doubles): " + math.calculate(2.5, 3.5));

System.out.println("Product (3 doubles): " + math.calculate(1.5, 2.0, 2.5));

}

}



# 🧪 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

class Weather {

public Weather() {

System.out.println("Weather created");

}

public void display() {

System.out.println("General weather information");

}

}

class Storm extends Weather {

public Storm() {

super();

System.out.println("Storm created");

}

@Override

public void display() {

System.out.println("Stormy weather with heavy rain");

}

}

class Thunderstorm extends Storm {

public Thunderstorm() {

super();

System.out.println("Thunderstorm created");

}

@Override

public void display() {

System.out.println("Thunderstorm with lightning and thunder");

}

}

class Sunshine extends Weather {

public Sunshine() {

super();

System.out.println("Sunshine created");

}

@Override

public void display() {

System.out.println("Clear and sunny weather");

}

}

public class WeatherTest {

public static void main(String[] args) {

Weather[] weathers = {new Thunderstorm(), new Sunshine()};

for (Weather w : weathers) {

w.display();

}

}

}

