Ans 1) A)

**CODE:-**

**// ClassA: Base Class**

**class ClassA {**

**private int valueA; // Private variable for encapsulation**

**// Parameterized constructor**

**public ClassA(int valueA) {**

**this.valueA = valueA;**

**System.out.println("ClassA constructor called");**

**}**

**// Copy constructor**

**public ClassA(ClassA obj) {**

**this.valueA = obj.valueA;**

**}**

**// Getter for valueA**

**public int getValueA() {**

**return valueA;**

**}**

**// Setter for valueA**

**public void setValueA(int valueA) {**

**this.valueA = valueA;**

**}**

**}**

**// ClassB: Derived from ClassA**

**class ClassB extends ClassA {**

**protected int valueB; // Protected variable, accessible by derived classes**

**// Parameterized constructor**

**public ClassB(int valueA, int valueB) {**

**super(valueA); // Call ClassA's constructor**

**this.valueB = valueB;**

**System.out.println("ClassB constructor called");**

**}**

**// Copy constructor**

**public ClassB(ClassB obj) {**

**super(obj); // Call ClassA's copy constructor**

**this.valueB = obj.valueB;**

**}**

**// Private constructor to prohibit object creation**

**private ClassB() {**

**super(0); // Default value for ClassA**

**this.valueB = 0;**

**}**

**}**

**// ClassC: Derived from ClassB**

**class ClassC extends ClassB {**

**public int valueC; // Public variable**

**// Parameterized constructor**

**public ClassC(int valueA, int valueB, int valueC) {**

**super(valueA, valueB); // Call ClassB's constructor**

**this.valueC = valueC;**

**System.out.println("ClassC constructor called");**

**}**

**// Copy constructor**

**public ClassC(ClassC obj) {**

**super(obj); // Call ClassB's copy constructor**

**this.valueC = obj.valueC;**

**}**

**// Synchronized method to display all values**

**public synchronized void displayDetails() {**

**System.out.println("ValueA: " + getValueA()); // Access ClassA using getter**

**System.out.println("ValueB: " + valueB); // Access ClassB directly**

**System.out.println("ValueC: " + valueC); // Access ClassC directly**

**}**

**}**

**// Main class to test the program**

**public class Main {**

**public static void main(String[] args) {**

**// Create an object of ClassC**

**ClassC obj1 = new ClassC(10, 20, 30);**

**// Display details**

**obj1.displayDetails();**

**// Create a deep copy of obj1**

**ClassC obj2 = new ClassC(obj1);**

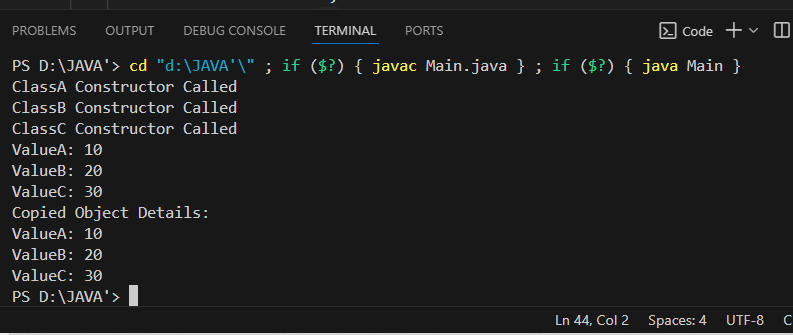
**System.out.println("Copied Object Details:");**

**obj2.displayDetails();**

**}**

**}**

**Output:-**

****

**Explanation :-**

· **Constructor Chaining:**

* ClassC uses super(valueA, valueB) to invoke ClassB's constructor.
* ClassB uses super(valueA) to invoke ClassA's constructor.
* This ensures that the constructors are called in a hierarchical order: ClassA -> ClassB -> ClassC.

· **Parameterized Constructors:**

* · Each class has a parameterized constructor to initialize its own variables.
* The constructor chaining ensures initialization of inherited variables.

· **Copy Constructors:**

* · Each class has a copy constructor that deep copies data from another object of the same class.
* **Example:** ClassC's copy constructor calls ClassB's copy constructor, which in turn calls ClassA's copy constructor.

· **Access Modifiers:**

* · valueA in ClassA is private and can only be accessed using getter and setter methods.
* valueB in ClassB is protected and directly accessible in derived classes.
* valueC in ClassC is public and directly accessible anywhere.

· **Synchronized Method:**

* · The displayDetails() method in ClassC is synchronized to ensure thread-safe access when displaying data.

· **Constraints:**

* · **No direct access to** valueA**:** ClassC accesses valueA using getValueA() from ClassA.
* **Prohibited object creation of** ClassB**:** A private no-argument constructor in ClassB ensures that no objects of ClassB can be created directly. Only derived classes or specific methods in ClassB can use it.