**Ans 1) B)**

**CODE:-**

**// Main class OuterClass**

**public class OuterClass {**

**private int instanceVar = 10; // Instance variable**

**private static int staticVar = 20; // Static variable**

**// 1. Static Nested Class**

**public static class StaticNestedClass {**

**static {**

**System.out.println("Static block in StaticNestedClass executed.");**

**}**

**{**

**System.out.println("Instance block in StaticNestedClass executed.");**

**}**

**public StaticNestedClass() {**

**System.out.println("Constructor of StaticNestedClass executed.");**

**}**

**// Method to demonstrate accessing outer class's static and instance variables**

**public void demonstrateAccess(OuterClass outer) {**

**System.out.println("Accessing OuterClass static variable: " + staticVar);**

**System.out.println("Accessing OuterClass instance variable using outer: " + outer.instanceVar);**

**}**

**}**

**// 2. Final Nested Inner Class**

**public final class FinalInnerClass {**

**public final int finalVariable = 100; // Final variable**

**public final void demonstrateFinalBehavior() {**

**System.out.println("Final variable is immutable: " + finalVariable);**

**}**

**}**

**// 3. Overriding toString and demonstrating 'this' and 'super'**

**@Override**

**public String toString() {**

**return "OuterClass instance with instanceVar = " + instanceVar;**

**}**

**public static class StaticNestedClassWithOuterRef {**

**public void accessOuterClass(OuterClass outer) {**

**System.out.println("Accessing OuterClass using OuterClass.this: " + outer.toString());**

**}**

**}**

**// 4. Static method demonstrating restricted access**

**public static void demonstrateStaticAccess() {**

**System.out.println("Accessing static variable: " + staticVar);**

**// Instance variables can't be accessed directly in a static method.**

**// System.out.println(instanceVar); // This would cause a compilation error.**

**}**

**// Synchronized method to demonstrate thread-safe access**

**public void synchronizedAccessToStatic() {**

**synchronized (OuterClass.class) {**

**System.out.println("Thread-safe access to shared static variable: " + staticVar);**

**staticVar++; // Modify the shared static variable**

**}**

**}**

**// Utility class**

**public static final class UtilityClass {**

**public static void utilityMethod() {**

**System.out.println("Utility method called.");**

**}**

**}**

**// Main method for testing**

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

**// 1. Static Nested Class demonstration**

**StaticNestedClass nestedObj = new StaticNestedClass();**

**OuterClass outer = new OuterClass();**

**nestedObj.demonstrateAccess(outer);**

**// 2. Final Inner Class demonstration**

**FinalInnerClass innerObj = outer.new FinalInnerClass();**

**innerObj.demonstrateFinalBehavior();**

**// 3. Demonstrating 'this' and 'super'**

**StaticNestedClassWithOuterRef nestedWithOuter = new StaticNestedClassWithOuterRef();**

**nestedWithOuter.accessOuterClass(outer);**

**// 4. Static method demonstration**

**demonstrateStaticAccess();**

**// Thread-safe static access**

**outer.synchronizedAccessToStatic();**

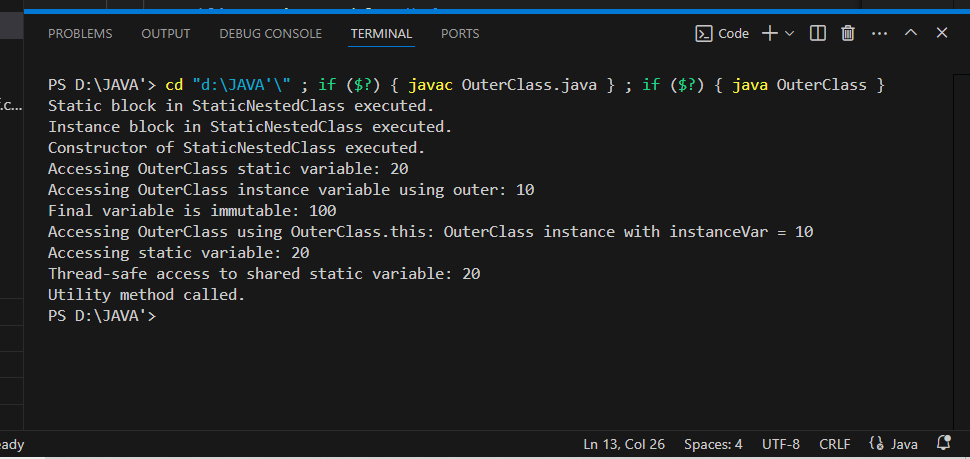
**// Utility class usage**

**UtilityClass.utilityMethod();**

**}**

**}**

**OUTPUT:-**

****

**EXPLANATION:-**

**Static Nested Class**:

* 1. A static nested class can be instantiated without an outer class instance.
  2. Includes:
     1. **Static block**: Runs once when the class is loaded.
     2. **Instance block**: Runs each time an object is created.
     3. **Constructor**: Initializes the nested class.
  3. Demonstrates accessing static and instance variables of the outer class using OuterClass.this.

**Final Nested Inner Class**:

* 1. Marked final to prevent subclassing.
  2. Contains a final variable to demonstrate immutability.
  3. Has a final method to prevent overriding.

this **and** super **Keywords:**

* 1. this: Used in the StaticNestedClassWithOuterRef to reference the outer class instance.
  2. super: Implicitly used in toString() when overriding from the Object class.

**Static Method in OuterClass:**

* 1. Demonstrates:
     1. Access to static variables.
     2. Restricted access to instance variables directly.
  2. Synchronized block in synchronizedAccessToStatic() ensures thread-safe access to staticVar.

**Utility Class**:

* 1. Marked final to prevent subclassing.
  2. Contains static methods only, emphasizing its role as a utility class.