**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

**Code**

class Logger {

// Step 2.1: Create a private static instance of Logger

private **static** Logger instance;

// Step 2.2: Make the constructor private to prevent instantiation

private Logger() {

System.out.println("Logger initialized.");

}

// Step 2.3: Provide a public static method to return the instance

public **static** Logger getInstance() {

**if** (instance == null) {

instance = new Logger ();

}

**return** instance;

}

public **void** log (String message) {

System.out.println("Log message: " + message);

}

}

public class test {

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

// Step 3.1: Get instances of Logger

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

// Step 3.2: Use the logger to log messages

logger1.log ("This is the first log message.");

logger2.log ("This is the second log message.");

// Step 3.3: Check if both logger instances are the same

**if** (logger1 == logger2) {

System.out.println("Both logger instances are the same (singleton works).");

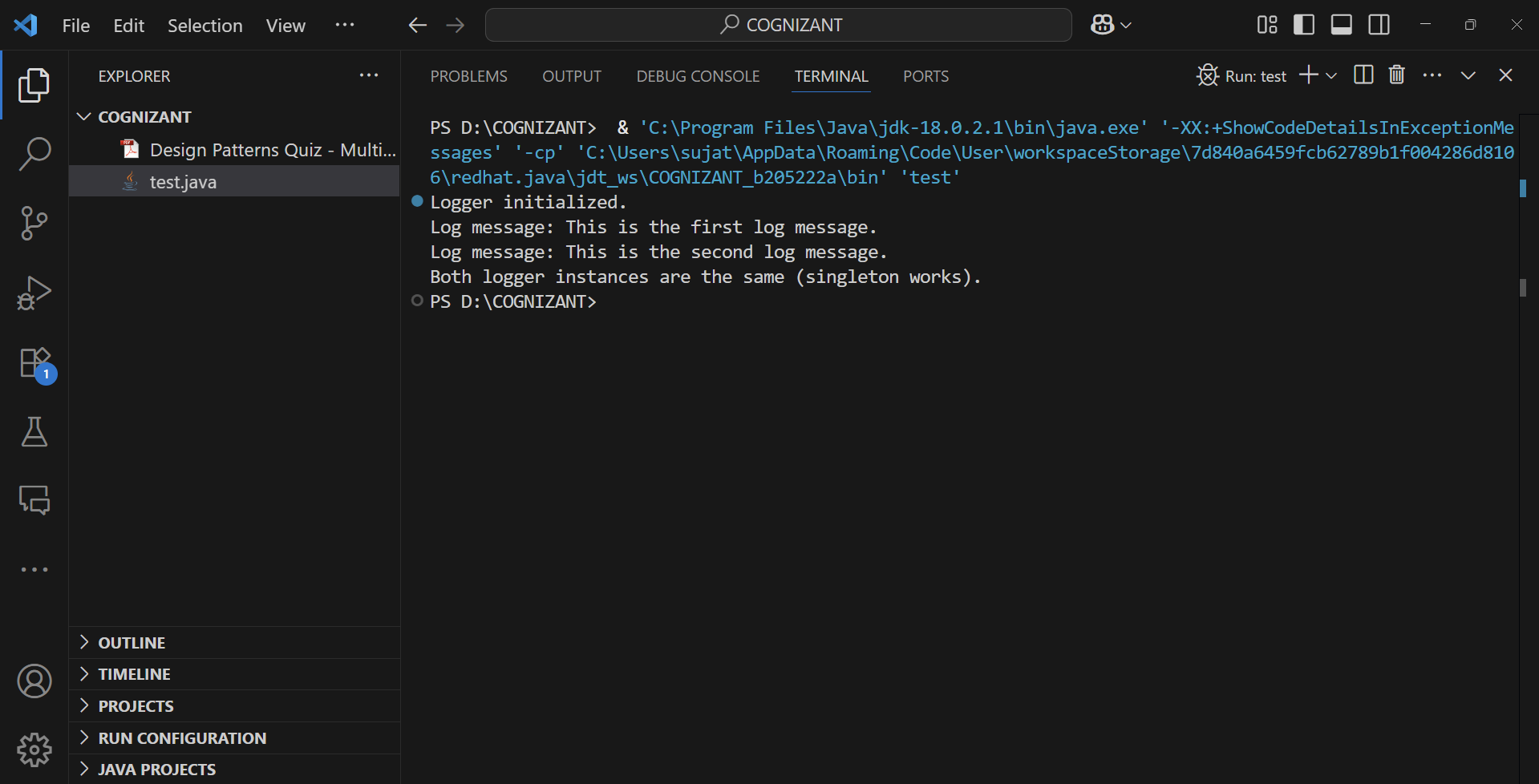
} **else** {

System.out.println("Different instances exist (singleton failed).");

}

}

}

**Output**