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:

o Create a new Java project named SingletonPatternExample.

2. Define a Singleton Class:

o Create a class named Logger that has a private static instance of itself.

o Ensure the constructor of Logger is private.

o Provide a public static method to get the instance of the Logger class.

3. Implement the Singleton Pattern:

o Write code to ensure that the Logger class follows the Singleton design pattern.

4. Test the Singleton Implementation:

o Create a test class to verify that only one instance of Logger is created and used across the application.  
  
CODE SECTION:  
  
**Logger.java**  
  
package src;

public class Logger {

    private static Logger uniqueLogger;

    private Logger() {

        System.out.println(">>> Logger Initialized!");

    }

    public static Logger getInstance() {

        if (uniqueLogger == null) {

            uniqueLogger = new Logger();

        }

        return uniqueLogger;

    }

    public void log(String message) {

        System.out.println("[LOG]: " + message);

    }

}  
  
**TestSingleton.java**  
  
package src;

public class TestSingleton {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        logger1.log("Message from logger1");

        logger2.log("Message from logger2");

        if (logger1 == logger2) {

            logger1.log("logger1 and logger2 are the same instance!");

        } else {

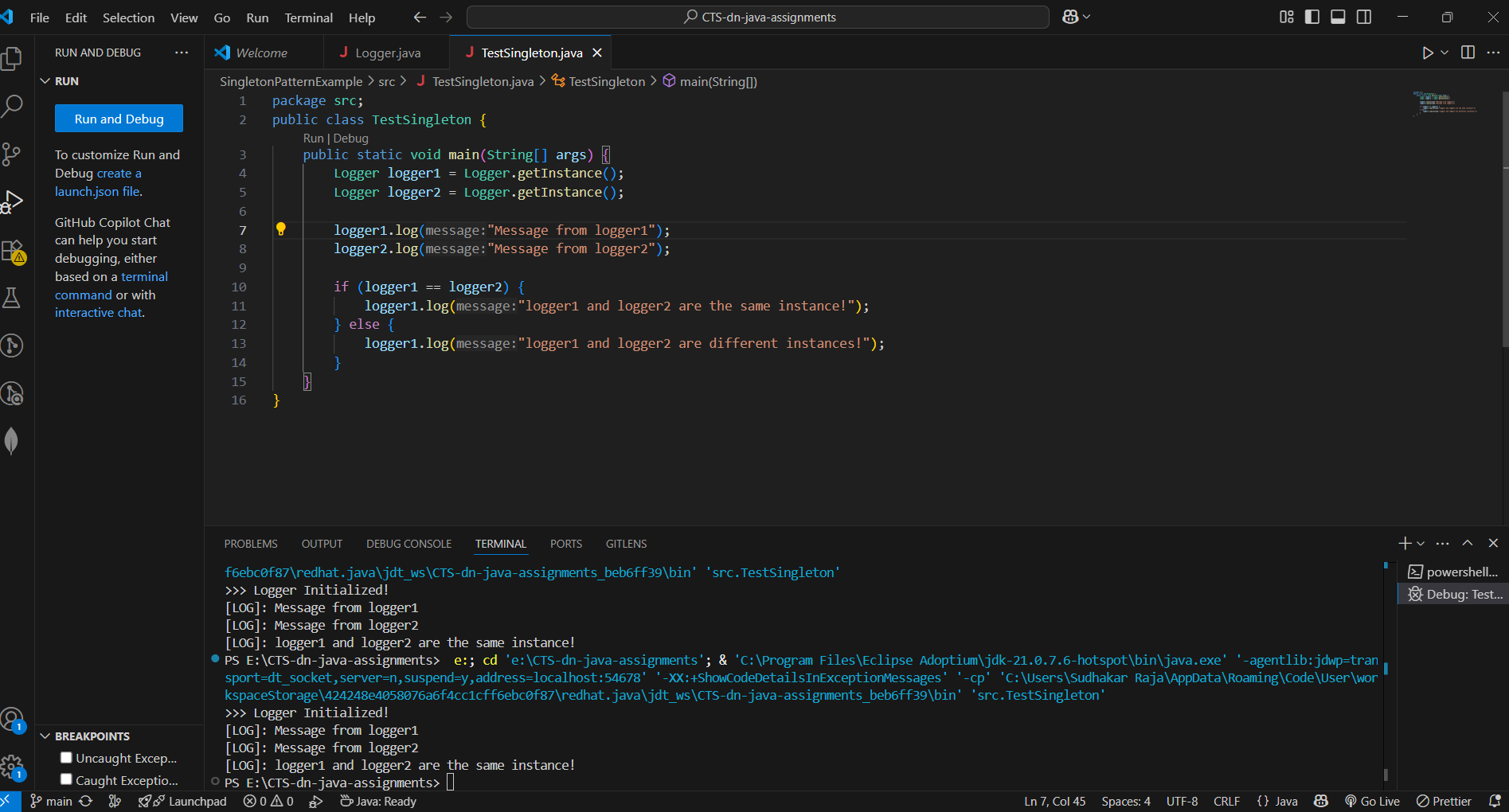
            logger1.log("logger1 and logger2 are different instances!");

        }

    }

}

**OUTPUT:**



**Conclusion:**  
 The assignment successfully demonstrates the Singleton design pattern where only one Logger instance is created and reused.