**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.

**Solution :**

**Logger.java :**

**package** logger;

**public** **class** Logger {

// Private static instance of the class

**private** **static** **final** Logger ***instance*** = **new** Logger();

// Private constructor

**private** Logger() {

System.***out***.println("Logger instance created");

}

// Public static method

**public** **static** Logger getInstance() {

**return** ***instance***;

}

// Logging method

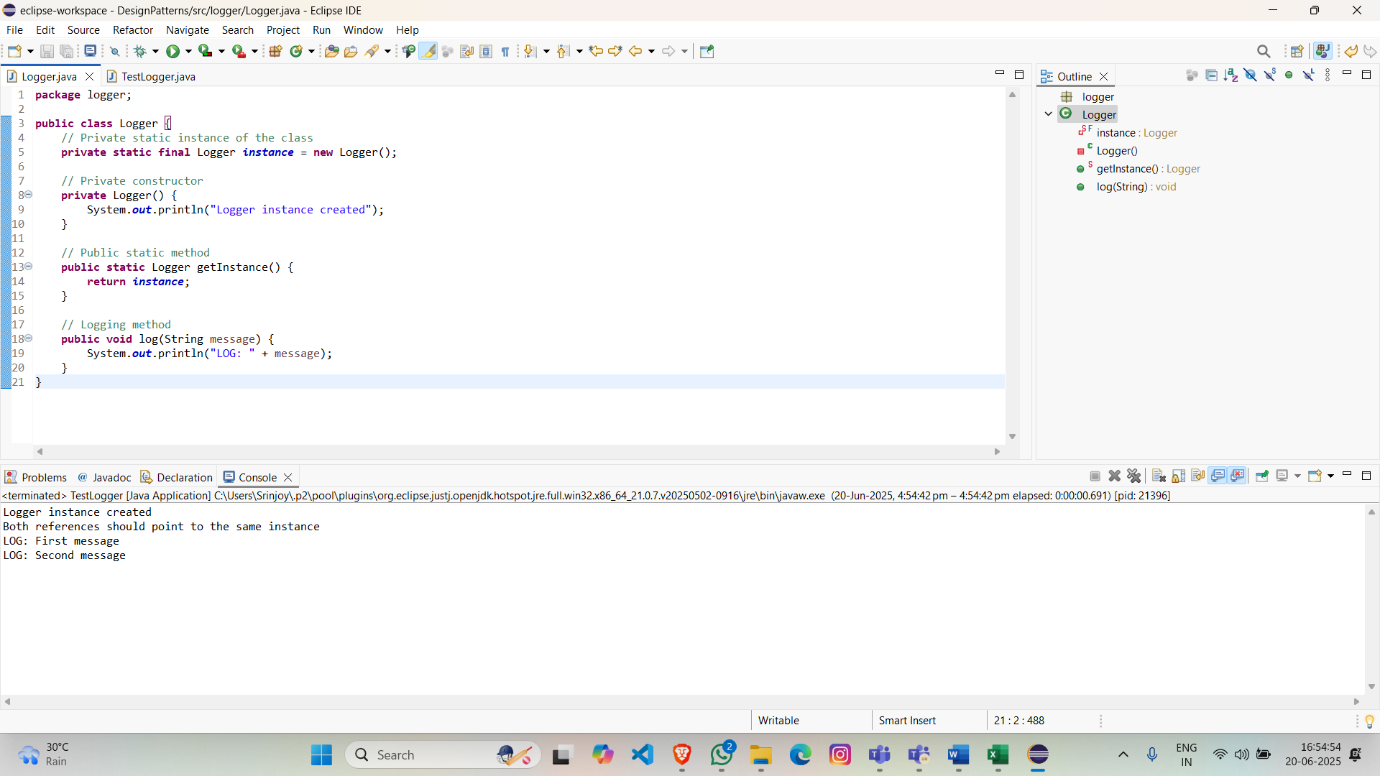
**public** **void** log(String message) {

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

}

}

**Screenshot:**



**TestLogger.java :**

**package** logger;

**public** **class** TestLogger {

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

// logger instances

Logger logger1 = Logger.*getInstance*();

Logger logger2 = Logger.*getInstance*();

// Verify singleton behavior

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

System.***out***.println("Both references should point to the same instance");

} **else** {

System.***out***.println("References will point to different instances which is wrong");

}

// Use logging functionality

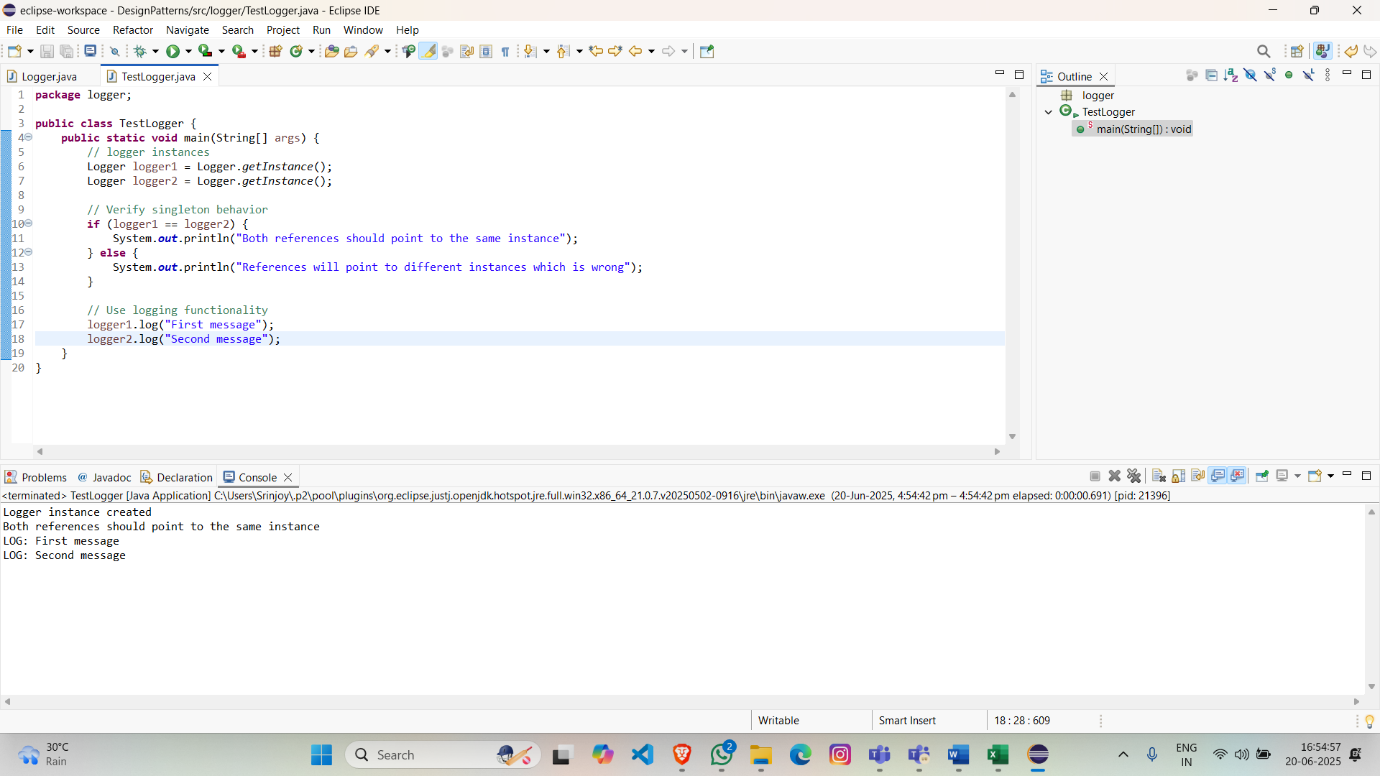
logger1.log("First message");

logger2.log("Second message");

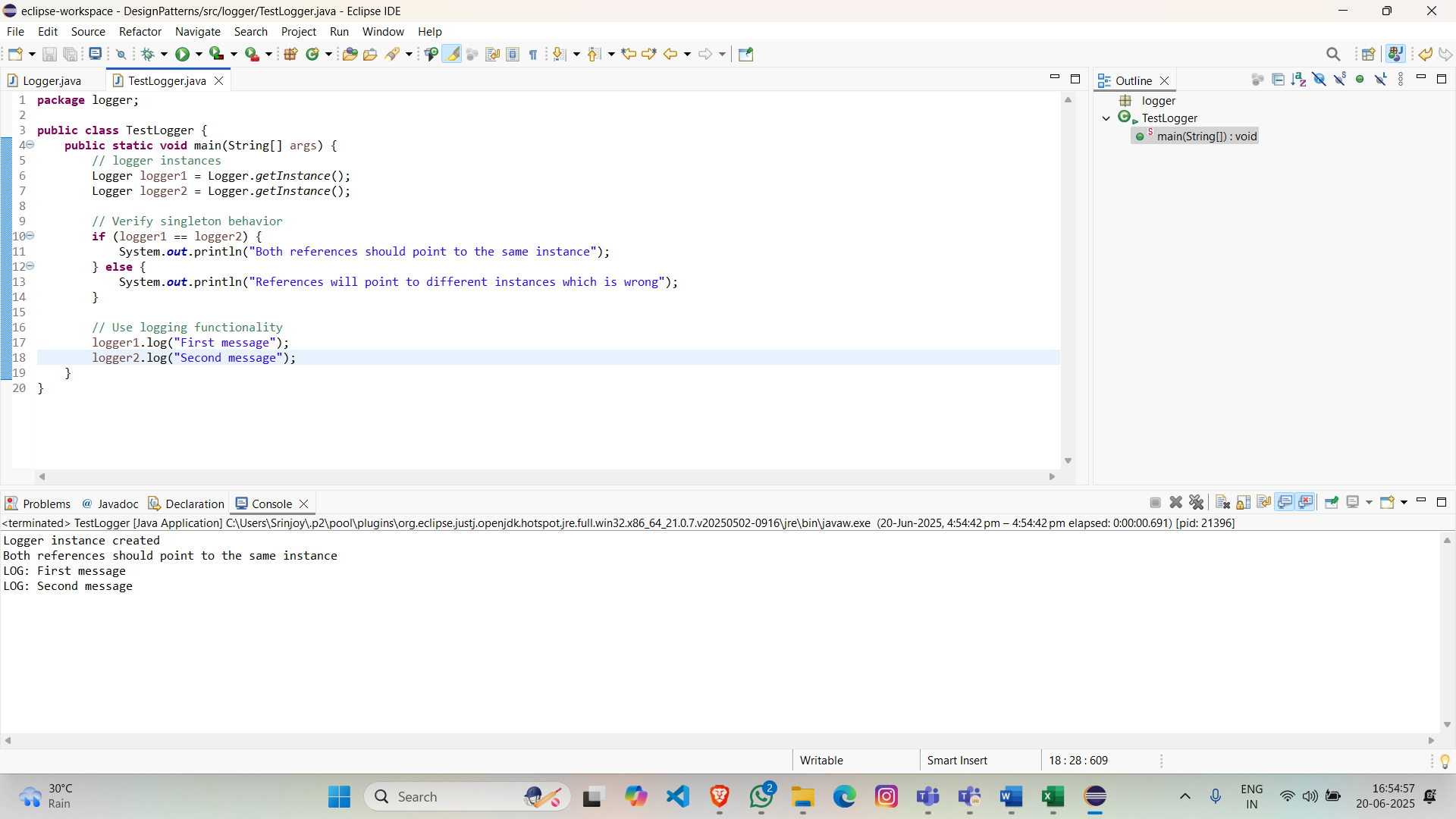
}

}

**Screenshot:**

****

**Output:**

****

**Key Points:**

* **Single Instance**: The static final instance ensures only one object is ever created.
* **Global Access**: The getInstance() method provides controlled access to the singleton instance.
* **Thread Safety**: Eager initialization is inherently thread-safe since the instance is created during class loading.
* **Prevention of External Instantiation**: The private constructor blocks direct object creation.