**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**

package mypackage;

public class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

**TestSingleton.java**

package mypackage;

public class TestSingleton {

public static void main(String[] args) {

System.out.println("Testing Singleton Pattern...");

Logger logger1 = Logger.getInstance();

logger1.log("First log message");

Logger logger2 = Logger.getInstance();

logger2.log("Second log message");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same. Singleton verified.");

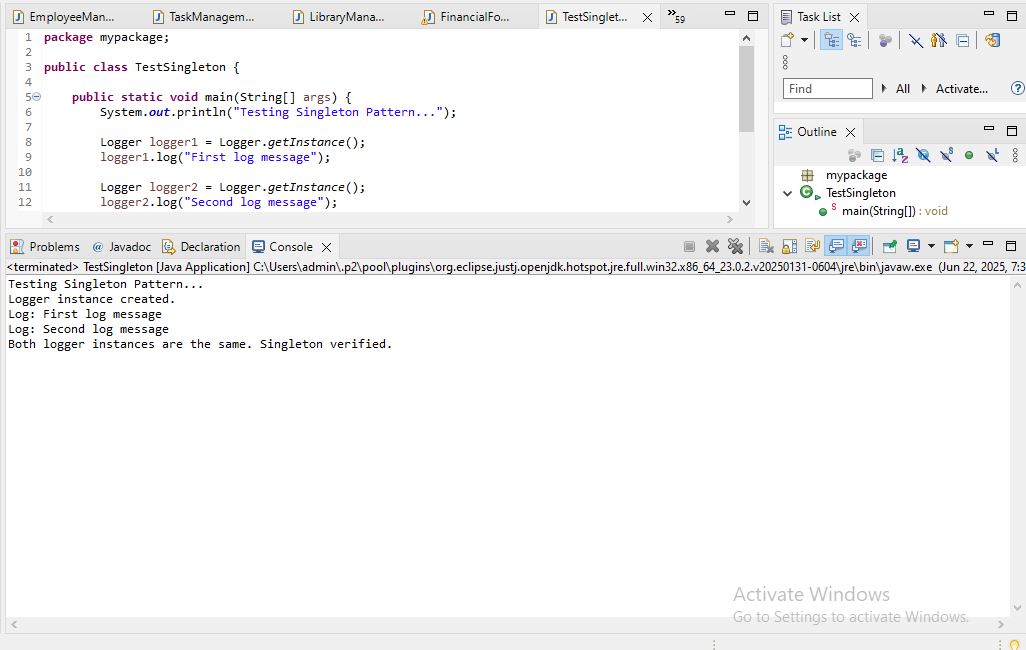
} else {

System.out.println("Different logger instances. Singleton failed.");

}

}

}



**Implementing the Singleton Pattern**  
We need to ensure that the Logger utility class has only one instance throughout the application to maintain consistent logging.

**1. Create a New Java Project:**

A new Java project is created with the name SingletonPatternExample.

**2. Define a Singleton Class:**

* The class Logger is defined with:
  + A private static variable instance to hold the single instance of Logger.
  + A private constructor to prevent external instantiation.
  + A public static method getInstance() to provide global access to the single instance.

**3. Implement the Singleton Pattern:**

* When getInstance() is called for the first time, a new instance of Logger is created.
* On subsequent calls, the existing instance is returned.
* This ensures that only one instance exists throughout the application lifecycle.

**4. Test the Singleton Implementation:**

* The TestSingleton class tests the Logger class.
* It retrieves the Logger instance multiple times.
* It compares the instances to verify that both references point to the same object, confirming the Singleton behavior.

**Advantages of Singleton Pattern:**

* Controls object creation and ensures only one instance.
* Saves memory by avoiding multiple instances.
* Useful for shared resources like logging, configuration, or database connections.

**Time Complexity:**

* The time complexity of getInstance() is O(1), as it performs a simple check and returns the instance.

**Possible Optimizations:**

* Use synchronization for thread-safety in multi-threaded environments.
* Use eager initialization or double-checked locking for advanced implementations.