**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.

**Project: SingletonPatternExample**

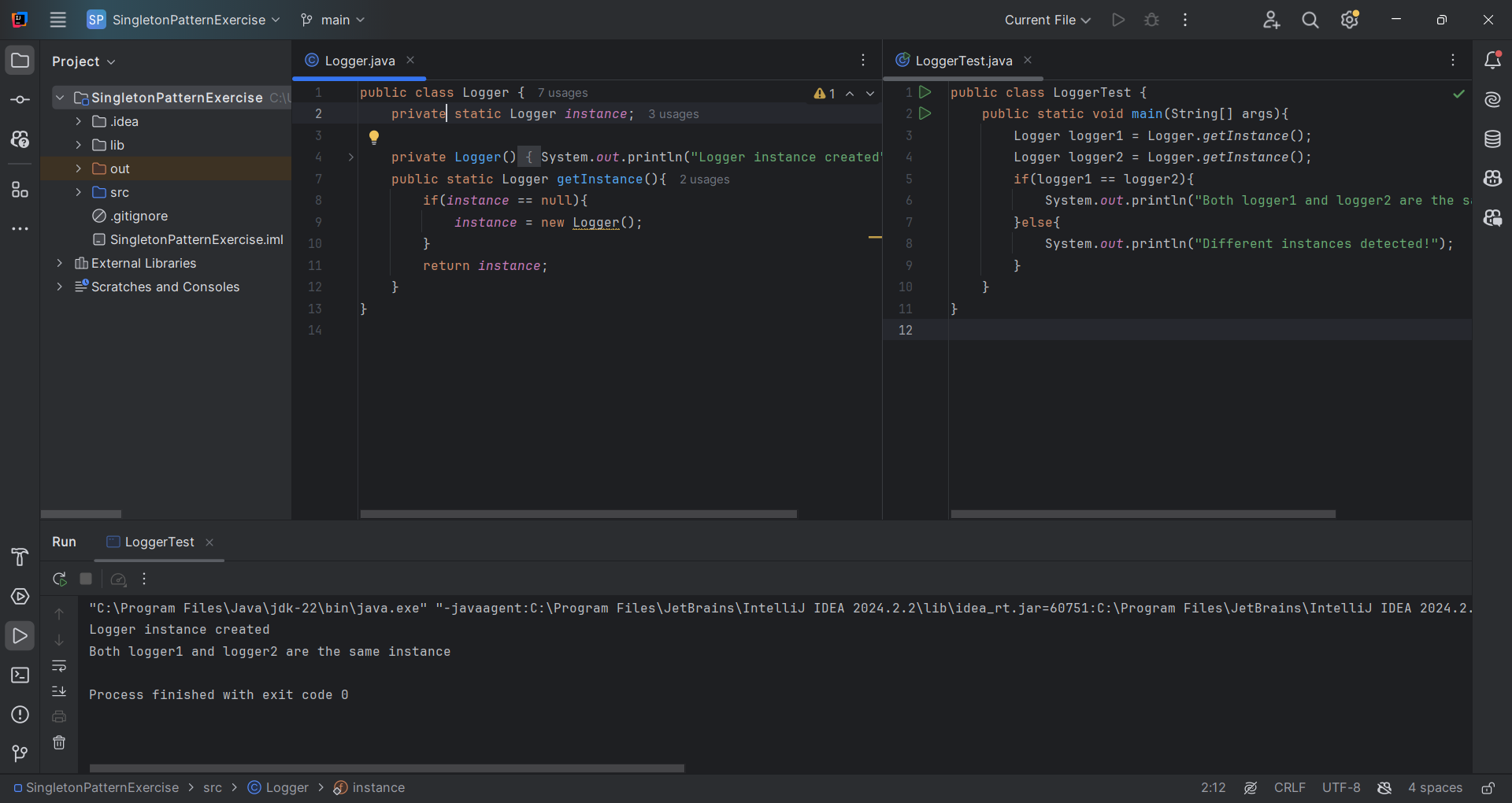
**Logger.java**

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*;  
 }  
}

**LoggerTest.java**

public class LoggerTest {  
 public static void main(String[] args){  
 Logger logger1 = Logger.*getInstance*();  
 Logger logger2 = Logger.*getInstance*();  
   
 if(logger1 == logger2){  
 System.*out*.println("Both logger1 and logger2 are the same instance");  
 }else{  
 System.*out*.println("Different instances detected!");  
 }  
 }  
}

**Output:**



**Explanation:**

The Logger.java class ensures that only one instance is ever created during the application’s lifetime.

This is achieved by making the constructor private, so it can only be accessed within the class itself. This prevents objects from being created using the new keyword from outside the class.

A public static method called getInstance() is defined to return the single instance of the class. This method creates an object only if the instance is null (if the object hasn’t already been created).

The LoggerTest.java class is used to test whether multiple objects are created. Two references are obtained by calling the getInstance() method from the Logger class. Both references should point to the same instance. Otherwise, the Singleton pattern has not been implemented correctly.