**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:

Create a New Java Project:

Create a new Java project named SingletonPatternExample.

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.

Implement the Singleton Pattern:

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

Test the Singleton Implementation:

Create a test class to verify that only one instance of Logger is created and used across the application.

**Program files:**

**Logger.java**

public class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() { //Can use synchronised in case of thread safety issues

if (instance == null) {

instance = new Logger();

}

return instance;}

public void log(String message) {

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

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First message from logger1");

Logger logger2 = Logger.getInstance();

logger2.log("Second message from logger2");

if (logger1 == logger2) {

System.out.println("Same instances");

} else {

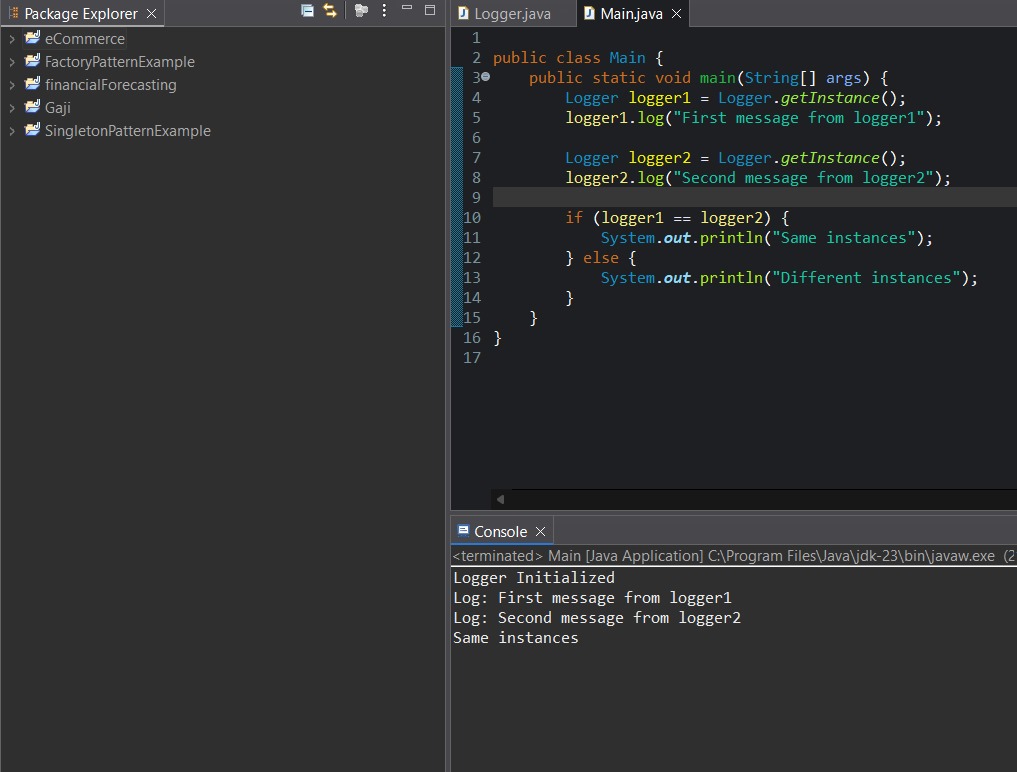
System.out.println("Different instances");

}

}

}

**Output:**

****