**Singleton Pattern Implementation for Logger**

1. First, create the Logger class (Logger.java)

public class Logger {

// The single instance of the class

private static Logger instance;

// Private constructor to prevent instantiation from outside

private Logger() {

// Initialization code if needed

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

}

// Public method to get the instance

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

// Example logging method

public void log(String message) {

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

}

}

2. Now create a test class (LoggerTest.java)

public class LoggerTest {

public static void main(String[] args) {

// Get logger instances

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

// Use the loggers

logger1.log("First log message");

logger2.log("Second log message");

// Check if both references point to the same object

if (logger1 == logger2) {

System.out.println("Both loggers are the same instance - Singleton works!");

} else {

System.out.println("Different instances created - Singleton failed!");

}

}

}

3. How This Works

1. Private Constructor: The `Logger` class has a private constructor so no other class can create new instances directly.

2. Static Instance: The class maintains a single static instance of itself.

3. getInstance() Method: This method:

- Checks if an instance already exists

- Creates one if it doesn't

- Returns the existing instance if it does

4. Thread Safety Note: This basic version isn't thread-safe. For a production environment, you'd want to add synchronization:

public static synchronized Logger getInstance() {

if (instance == null) {

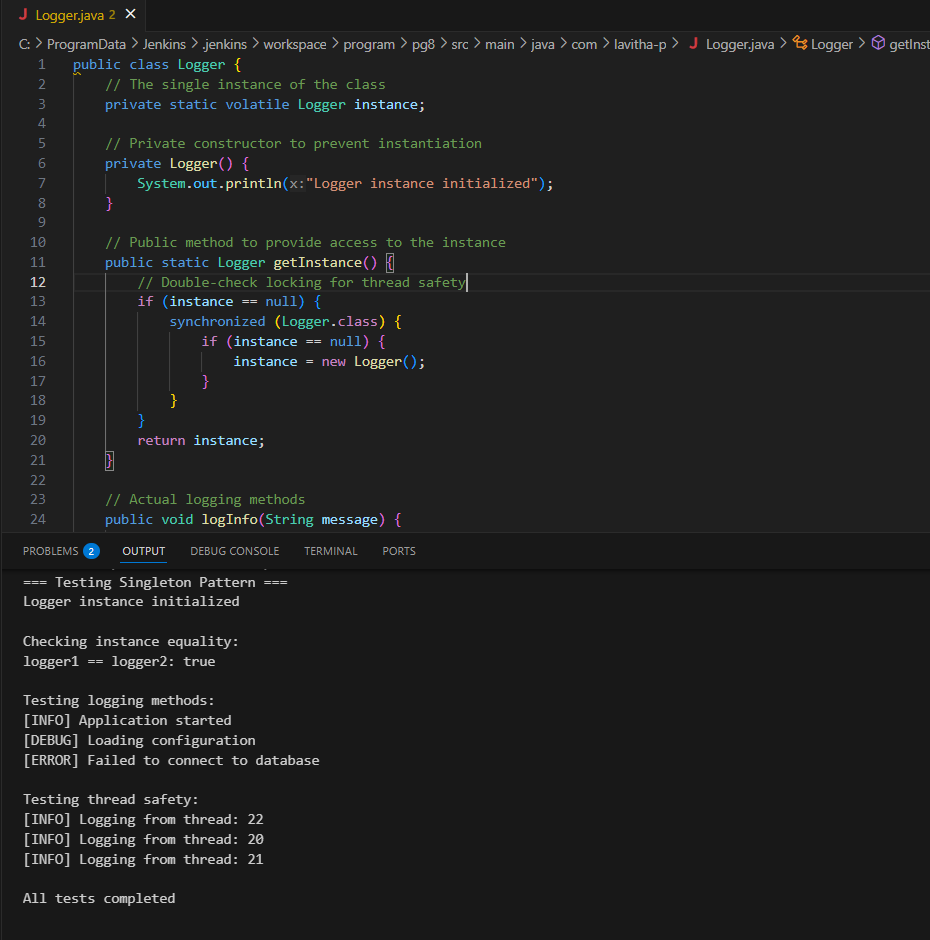
instance = new Logger();

}

return instance;

}

**Output:**

****