Exercise 1: Implementing the Singleton Pattern

1. Create a New Java Project

I created a new Java project and named it SingletonPatternExample. This project will contain a logging class that follows the singleton pattern so that it has only one object in the whole program.

2. Define a Singleton Class

We created a class called `Logger`. This class should not allow multiple objects to be created. It will have a private static instance of itself. The constructor is private so that no one can create it from outside the class.

3. Implement the Singleton Pattern

Here is the code for the Logger class:

```
public class Logger {

    private static Logger instance = null;

    // private constructor so other classes can't make objects
    private Logger() {
        System.out.println("Logger initialized");
    }

    // public static method to return the instance
    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }

    // just a method to show logging message
    public void log(String message) {
        System.out.println("Log: " + message);
    }
}
```

4. Test the Singleton Implementation

To check if it really creates only one instance, I wrote a simple test class.

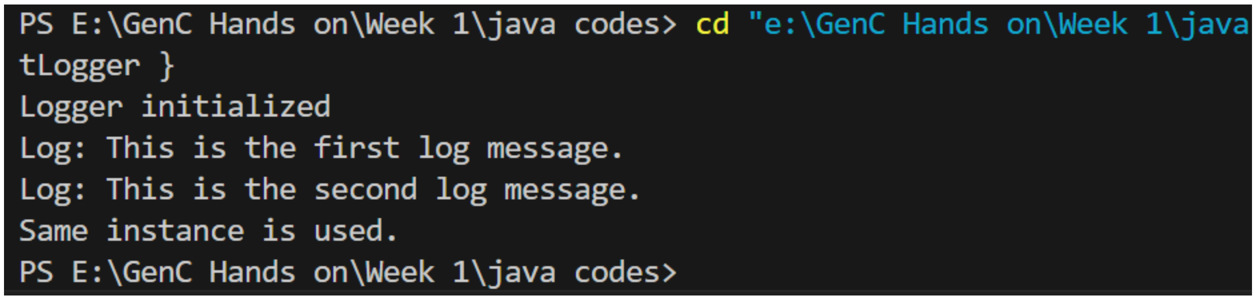
```
public class TestLogger {
    public static void main(String[] args) {

        Logger log1 = Logger.getInstance();
        Logger log2 = Logger.getInstance();

        log1.log("This is the first log message.");
        log2.log("This is the second log message.");

        if (log1 == log2) {
            System.out.println("Same instance is used.");
        } else {
            System.out.println("Different instances were created.");
        }
    }
}
```

OUTPUT:



```
PS E:\GenC Hands on\Week 1\java codes> cd "e:\GenC Hands on\Week 1\java
tLogger }
Logger initialized
Log: This is the first log message.
Log: This is the second log message.
Same instance is used.
PS E:\GenC Hands on\Week 1\java codes>
```

As we see in the output, the constructor message only printed once, which shows that only one object was created.

Conclusion:

Singleton is useful when we want to make sure that only one object exists in the whole application. This is common for loggers, configuration classes, and so on.

We can improve it by using a thread-safe singleton, but for this exercise, this is enough.