**Exercise 1: Implementing the Singleton Pattern**

**1. Scenario**

In a real-world application, a logging utility is used to keep track of system events, errors, and usage information. It’s important to maintain **only one instance** of the logger to ensure **consistent and centralized logging**. This is where the **Singleton Design Pattern** is helpful.

**2. Java Project Setup**

A new Java project is created and named:  
**SingletonPatternExample**

Within the project:

* A class named Logger is created to follow the Singleton design.
* A test class named Main is created to demonstrate the Singleton behavior.

**3. Singleton Pattern Implementation**

The Singleton pattern ensures that a class has only one instance and provides a global point of access to it.  
This is done using:

* A **private static instance variable** of the class.
* A **private constructor**, so the class cannot be instantiated from outside.
* A **public static method** (getInstance()) to return the single instance.

In the Logger class:

* The first call to getInstance() creates the object.
* All future calls return the same object.

This guarantees that only one instance exists throughout the application lifecycle.

**4. Testing the Singleton**

In the Main class:

* Two logger variables (logger1 and logger2) are created using Logger.getInstance().
* When compared using ==, both refer to the **same object**, confirming the Singleton pattern works.

Output shows:

* Logger is initialized only once.
* Both instances log messages using the same logger.
* The comparison confirms singleton behavior.