**Exercise 1: Implementing the Singleton Pattern**

* **Create a New Java Project:**

To create a new Java project named *SingletonPatternExample*, you would typically use an Integrated Development Environment (IDE) such as Eclipse, IntelliJ IDEA, or NetBeans. The process generally involves selecting "New Project" from the File menu, choosing "Java Project" as the project type, and then naming it *SingletonPatternExample*. After creating the project, you'd have a basic structure with a src folder for your source code.

* **Define a Singleton Class:**

In this step, you would create a new class named Logger within the src folder of your project. This class would have a private static instance of itself, ensuring that only one instance can exist. The constructor of the Logger class would be made private to prevent direct instantiation from outside the class. You would also include a public static method, often named *getInstance(),* to provide controlled access to the single instance of the class.

* **Implement the Singleton Pattern**:

Implementing the Singleton pattern involves ensuring that the Logger class can only be instantiated once. This is typically achieved by creating the instance when it's first requested (lazy initialization) and then always returning that same instance for subsequent requests. The *getInstance()* method would check if an instance already exists; if not, it would create one, and if it does, it would return the existing instance. This ensures that no matter how many times *getInstance()* is called, only one Logger object is ever created.

* **Test the Singleton Implementation:**

To test the Singleton implementation, you would create a separate test class, often named Main or *LoggerTest*. In this class, you would write code to get multiple instances of the Logger class using the *getInstance()* method. You would then compare these instances to verify that they are indeed the same object. This can be done by comparing the object references or by setting a property on one instance and checking if it's reflected in the other instance. If the Singleton pattern is implemented correctly, all instances should be identical, proving that only one instance of the Logger class exists throughout the application.

The output of the code --

