**Exercise 1: Implementing the Singleton Pattern**

The Singleton design pattern ensures that a class has only one instance throughout the entire lifecycle of the application, and it provides a global point of access to that instance. In this exercise, we will create a Logger class to demonstrate how to implement this pattern.

**Steps:**

* **Private Static Instance**: We will have a private static variable to hold the single instance of the class.
* **Private Constructor**: The constructor is private to prevent the creation of objects from outside the class.
* **Public Static Method**: This method will return the single instance of the Logger class.

**Implement the Singleton Pattern:**

We have already implemented the Singleton pattern in the Logger class by following these key rules:

* The instance variable is static.
* The constructor is private, preventing direct instantiation.
* The getInstance() method ensures that only one instance is created.

**Summary of Steps:**

* **Singleton Class (Logger)**: We created a private static instance, a private constructor, and a public method to return the instance.
* **Test Class (SingletonTest)**: We created a test class to verify that the Logger class only creates a single instance and that all logging operations use this single instance.

**Output:**

LOG: This is the first log message.

LOG: This is the second log message.

Both logger1 and logger2 are the same instance.

**Conclusion:**

This implementation ensures that your logging utility will only have one instance throughout the application lifecycle, which is crucial for consistent logging and avoiding unnecessary resource consumption.