**Exercise 1: Implementing the Singleton Pattern**

**Overview**

This exercise implements the Singleton design pattern using a Logger class. The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance.

**Components**

**1. Logger Class (Singleton)**

The Logger class is the core of this implementation. It includes several key elements:

1. **Private Static Instance**:
   * A private static variable holds the single instance of the Logger class.
   * This ensures that only one instance of Logger exists across the entire application.
2. **Private Constructor**:
   * The constructor is made private to prevent direct instantiation of the Logger class from outside.
   * This is crucial for maintaining the single instance principle of the Singleton pattern.
3. **Public Static Access Method (getInstance)**:
   * This method provides global access to the single Logger instance.
   * It implements lazy initialization: the Logger instance is created only when this method is called for the first time.
   * Subsequent calls return the existing instance.
4. **Logging Functionality**:
   * Methods to set the log level and log messages are included.
   * These demonstrate the actual functionality of the Logger beyond just being a Singleton.

**2. SingletonTest Class (Main Application)**

This class serves as a test harness for the Singleton Logger:

1. **Instance Retrieval**:
   * It retrieves two instances of the Logger using getInstance().
2. **Functionality Testing**:
   * Sets different log levels and logs messages using both retrieved instances.
3. **Instance Comparison**:
   * Compares the two retrieved instances to verify they are indeed the same object.

**How It Works**

1. When the application starts, no Logger instance exists.
2. The first call to Logger.getInstance() creates the single instance.
3. Subsequent calls to getInstance() return this existing instance.
4. All parts of the application use this same Logger instance, ensuring consistent logging behavior.

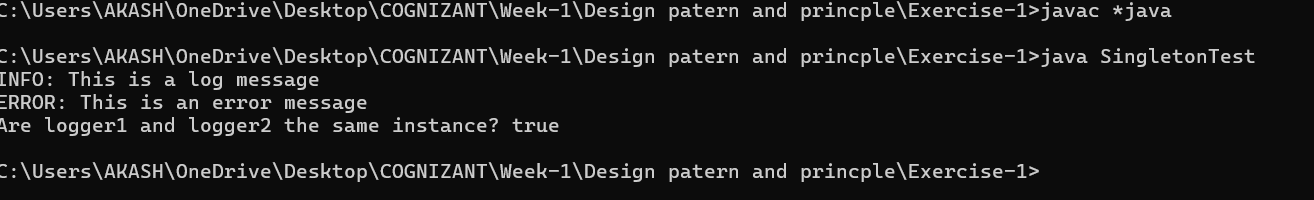
**Benefits of This Implementation**

1. **Global Access**: Provides a single point of access to the Logger throughout the application.
2. **Lazy Initialization**: The Logger is created only when it's first needed, saving resources.
3. **Consistency**: Ensures that all logging operations use the same instance, maintaining consistent behavior.

**Considerations**

1. **Thread Safety**: This basic implementation is not thread-safe. In a multi-threaded environment, additional synchronization would be needed.
2. **Testing Challenges**: Singletons can make unit testing more difficult as they introduce global state.

**Output**

****

**Conclusion**

This Singleton implementation provides a centralized logging mechanism for an application. It demonstrates how to ensure that only one instance of a class exists and how to provide global access to that instance. The pattern is particularly useful for managing shared resources like loggers, but should be used judiciously as it introduces global state into the application.