**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Solution:**

**What is a Singleton Pattern?**

Singleton Design Pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to it. This pattern is particularly useful when exactly one object is needed to coordinate actions across the system.

**What are the Steps to follow when creating a Singleton Pattern?**

* **Private Constructor:** The Singleton class has a private constructor to prevent the instantiation of the class from external entities.
* **Private Instance:** The class contains a private static instance of itself.
* **Static Method:** A static method is provided to access the instance, and it ensures that only one instance is created if it doesn't exist.

**Real World Example:**

Think of Logger like a printer in a school.

* You don't want each student to bring their own printer.
* You want only one printer shared by everyone — that’s the singleton instance.

**When Singleton Pattern Is Commonly Used?**

* You need exactly one object to coordinate actions across the system.
* You want to control access to shared resources (like a log file, database connection, etc.).

**Explanation of the Java code:**

🔹 private static Logger singleInstance;

* This holds the only instance of the Logger class.
* It’s static so it belongs to the class and not to any object.

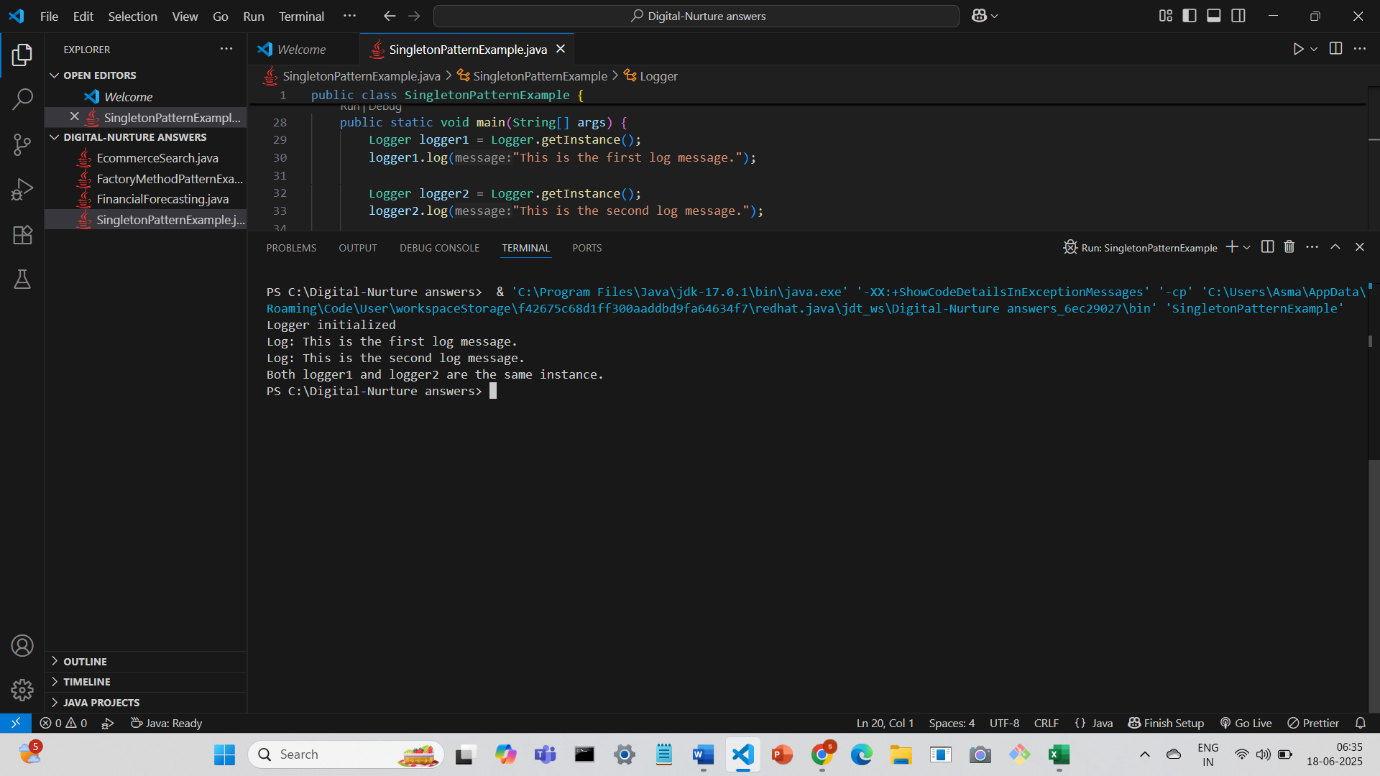
🔹 private Logger()

* The constructor is private, so other classes cannot use new Logger().
* This ensures that new instances can't be created accidentally.

🔹 public static Logger getInstance()

* This method is the only way to get the Logger instance.
* It creates the instance the first time it’s called (lazy initialization).
* After that, it just returns the already created instance.

**Output ScreenShot:**

****

**Conclusion:**

Because no matter how many times you call Logger.getInstance(), it always gives you the same object — only one instance exists in memory.