**Singleton Pattern Explaination**

**Purpose of the Code**

This program demonstrates the **Singleton Design Pattern**, which ensures that only **one instance** of a class is created throughout the application. It’s commonly used for things like logging, configuration, or managing shared resources.

**Class Structure**

The Logger class is designed in such a way that no other class can create its object directly using new. That’s because the constructor Logger() is marked private. This is key to controlling instance creation.

**Static Instance**

Inside the class, there’s a private static variable instance that holds the single instance of the Logger class. Initially, it is set to null.

**getInstance() Method**

This is the method that returns the instance of the Logger:

* If instance is null, it creates a new one.
* If it already exists, it just returns the same object.  
  This way, only **one object is ever created**, no matter how many times getInstance() is called.

**log() Method**

This is a regular method that takes a message and prints it with a “LOG:” prefix. It simulates how a real-world logger might work.

**Main Method Execution**

In main():

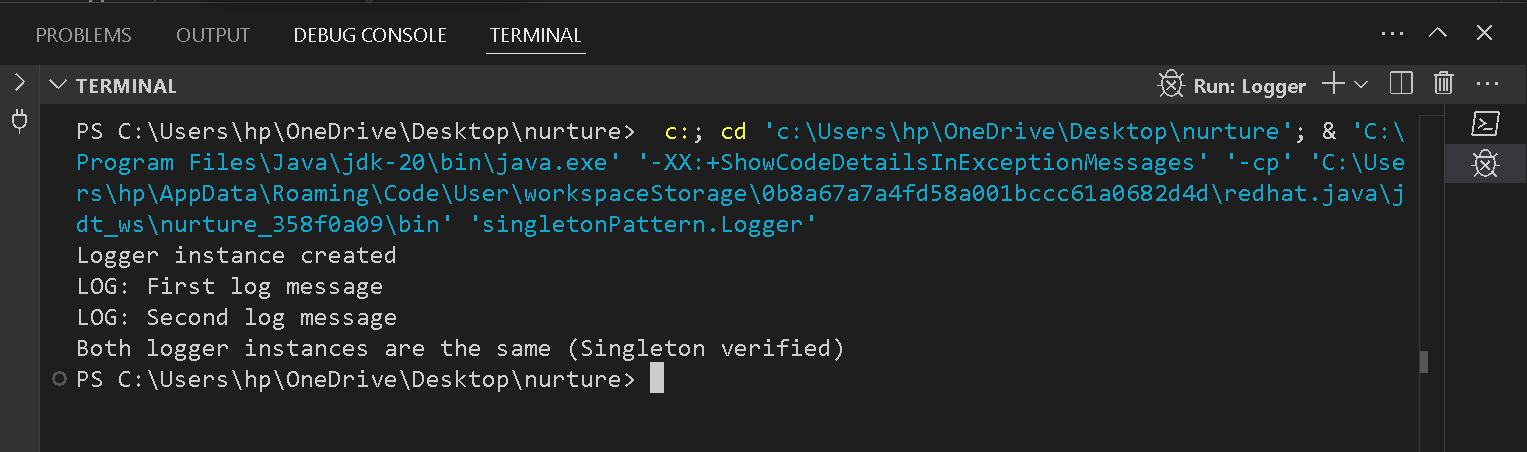
logger1 is created using Logger.getInstance() and logs a message.

* Then logger2 also calls Logger.getInstance(). Since the instance already exists, it reuses the same one.
* Finally, it compares logger1 and logger2 and confirms they are the **same instance**, proving that the singleton pattern worked.

**Result**

* Only one instance of Logger is ever created.
* All log messages go through that same instance.
* The output confirms the Singleton pattern was correctly implemented.

**Output🡺**

****