Problem No: 04

Problem Name: Implement and design Singleton design pattern.

Objectives:

- To understand the Singleton Design Pattern and its purpose.
- To implement the Singleton Design Pattern using a class.
- To demonstrate the benefits of the Singleton pattern, such as ensuring only one instance of a class exists throughout the program.

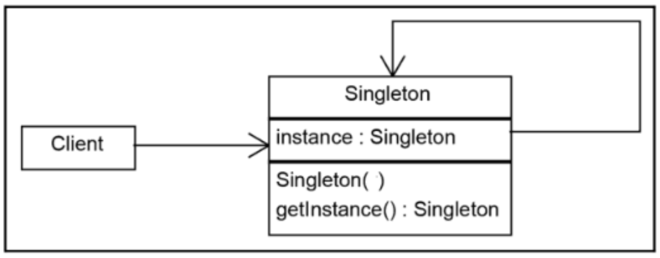
Theory:

The Singleton pattern restricts the instantiation of a class to one single object. It ensures that the class has only one instance and provides a global access point to that instance.

Real-life Example:

Database Connection: Often in applications, a database connection should be shared. If there were multiple instances of a database connection, it could lead to issues like resource wastage, inconsistent state, etc. The Singleton pattern ensures that only one database connection is created and used throughout the program.

UML/User-defined Class Design:



Program (Java):

```
public class Singleton {
    private static Singleton instance;
    private Singleton() {
        // Initialization code
    }
    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
    public void displayMessage() {
        System.out.println("Hello from Singleton!");
    }
    public static void main(String[] args) {
        // Getting the single instance of the Singleton class
        Singleton singleton = Singleton.getInstance();
        singleton.displayMessage();
        Singleton anotherInstance = Singleton.getInstance();
        System.out.println("Are both instances the same? " + (singleton == anotherInstance));
    }
}
```

Result and Discussion:

- The output of the displayMessage() method confirms that the Singleton class is functioning correctly.
- It will also print a message confirming that both singleton and anotherInstance refer to the same object.
- The program demonstrates the Singleton Design Pattern, ensuring that only one instance of the Singleton class is created.
- The use of the getInstance() method prevents the creation of multiple objects of the class, which is the main goal of the Singleton pattern.
- The key advantage of the Singleton pattern is that it provides a global point of access to the instance, which is particularly useful when managing shared resources like database connections, loggers, etc.
- The use of the private constructor and static getInstance() method ensures that the object is lazily initialized and only created when needed.