Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJ19ITC802 **DATE:** 24-01-2024

COURSE NAME: Design Patterns Laboratory **CLASS:** BE - IT

EXPERIMENT NO. 1

CO/LO: Identify and apply the most suitable design pattern to address a given application design problem.

AIM: Code in any language of your choice to implement Singleton Pattern for creating a single object.

DESCRIPTION:

Singleton is a part of Gang of Four design pattern, and it is categorized under creational design patterns. In this article, we are going to take a deeper look into the usage of the Singleton pattern. It is one of the simplest design patterns in terms of the modelling but on the other hand, this is one of the most controversial patterns in terms of complexity of usage. Singleton pattern is a design pattern which restricts a class to instantiate its multiple objects. It is nothing but a way of defining a class. Class is defined in such a way that only one instance of the class is created in the complete execution of a program or project. It is used where only a single instance of a class is required to control the action throughout the execution. A singleton class shouldn't have multiple instances in any case and at any cost. Singleton classes are used for logging, driver objects, caching and thread pool, database connections.

SOURCE CODE:

```
public class Singleton {
    private static Singleton instance;

    private Singleton() {
        // Private constructor to prevent instantiation from outside
    }

    public static Singleton getInstance() {
        // Lazy initialization: create the instance only if it hasn't been created yet
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    public static void main(String[] args) {
        // Testing the Singleton class
```

```
        Singleton singleton1 = Singleton.getInstance();
        Singleton singleton2 = Singleton.getInstance();
        System.out.println(singleton1 == singleton2); // Output: true
    }
}
```

OUTPUT

```
True
```

CONCLUSION:

We create two instances of the Singleton class (singleton1 and singleton2) and check if they are the same object using the is operator. Since the Singleton pattern ensures that only one instance of the class can exist, singleton1 and singleton2 are the same object and the output is True.

REFERENCES:

[1] https://www.geeksforgeeks.org/singleton-design-pattern-introduction/?ref=lbp