Lab No. : 04

Experiment Name : Design singeleton design pattern using UML .

Objectives:

1.Singleton pattern offers a straightforward solution for ensuring only one instance of a class is created, simplifying object management.

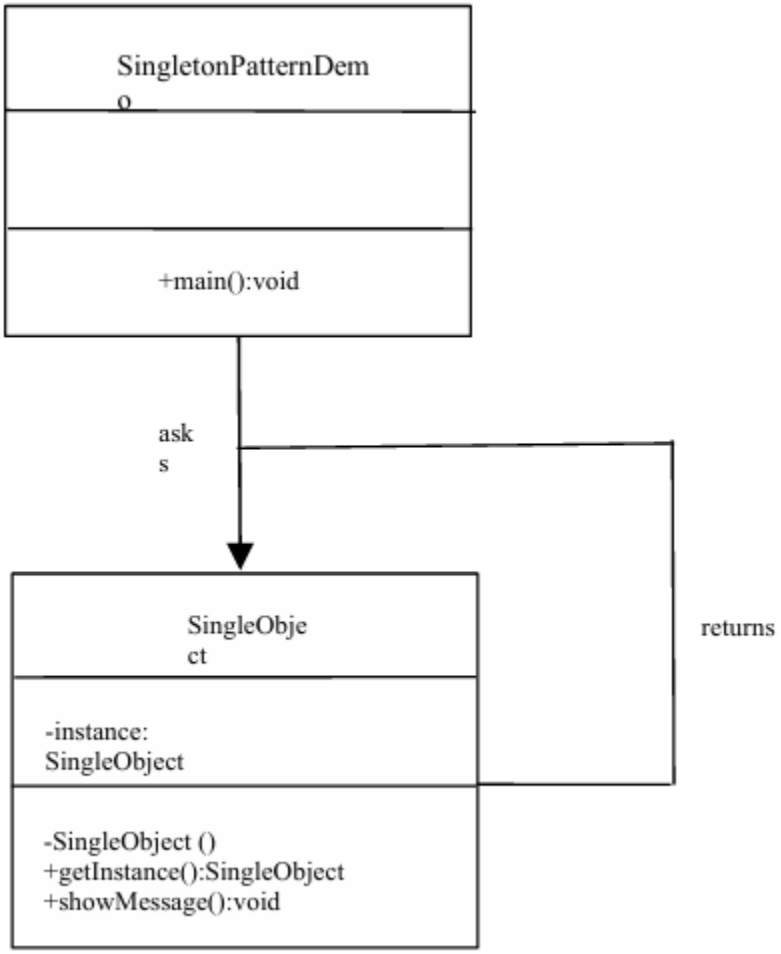
2. It provides an efficient method for object creation, promoting resource management and best practices in Java applications.

3. Singleton encapsulates object creation within a single class, ensuring centralized access and preventing accidental instantiation elsewhere in the codebase.

Theory:

The Singleton Pattern stands as one of the most ubiquitous design patterns in software engineering. Its simplicity and ease of implementation render it a go-to choice for ensuring that only a single instance of a class exists within a system. By encapsulating its constructor as private and maintaining a static instance of itself, the Singleton Pattern allows controlled access to this instance through a static method, thus offering a centralized point for accessing globally shared resources or managing state. However, the widespread use of the Singleton Pattern can sometimes lead to its misuse or overuse within software projects. In scenarios where its application isn't genuinely warranted, the drawbacks of employing it can overshadow its benefits. Instances of such misuse can result in tightly coupled code, reduced flexibility, and hindered testability, detracting from the overall maintainability and scalability of the software. Consequently, the Singleton Pattern occasionally garners criticism and is labeled as an antipattern or pattern singleton. This designation highlights instances where its adoption introduces unnecessary complexity or hampers the overall design of the system.

Implementation of singeleton design pattern using UML class design:



Implementation in Java:

Step 1 :

Create a Singleton Class.

SingleObject.java

```
public class SingleObject {
//create an object of SingleObject
private static SingleObject instance = new SingleObject();
//make the constructor private so that this class cannot be
//instantiated
private SingleObject(){}
```

```
//Get the only object available
public static SingleObject getInstance(){
return instance;
}
public void showMessage(){
System.out.println("Hello World!");
}
}
```

Step 2 :

Get the only object from the singleton class.

SingletonPatternDemo.java

```
public class SingletonPatternDemo {
public static void main(String[] args) {
//illegal construct
//Compile Time Error: The constructor SingleObject() is not visible
//SingleObject object = new SingleObject();
//Get the only object available
SingleObject object = SingleObject.getInstance();
//show the message
object.showMessage();
}
}
```

Step 3 :

Verify the output.

```
Hello World!
```

Discussion:

By implementing the Singleton pattern, we ensure that only one instance of a class exists throughout the application, offering centralized access and control. This pattern is particularly

useful in scenarios where resources need to be shared, configuration needs to be managed, or when a single point of control is required. Through the combination of UML diagrams and practical code examples, we have illustrated the effective implementation of the Singleton pattern in software design.