Creational Design Pattern

Singleton Design

Aim:
The aim is to ensure that a class has only one instance globally accessible, facilitating efficient resource management and providing a centralized point for accessing and controlling shared state within the application.

Description:
The provided Java code implements the Singleton design pattern, ensuring that only one instance of the `Singleton` class exists. The class contains a private static instance and a private constructor to prevent external instantiation. Access to the singleton instance is provided through a static method `getInstance()`. The `TestClass` demonstrates obtaining and modifying singleton instances, showcasing the pattern's ability to maintain a single global point of access to shared resources. This design enhances resource management and facilitates centralized control of global state within the application. Through thread-safe instantiation and efficient memory usage, the Singleton pattern promotes efficient and organized software design.

Code:

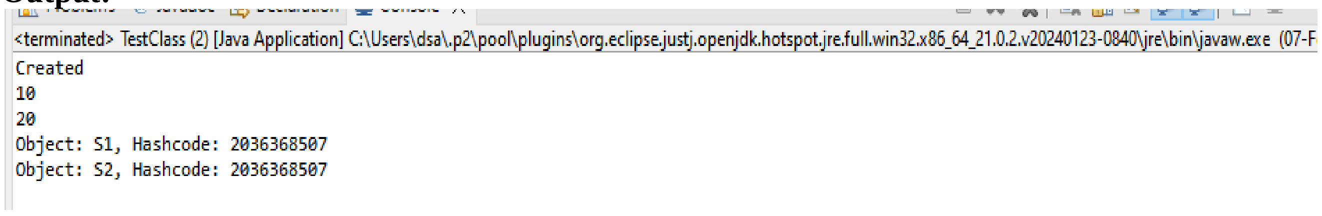
1. Singleton.java

```
package design;

public class Singleton {
	private static Singleton soleInst=new Singleton();
	public int i;
	private Singleton() {
		System.out.println("Created");
	}

	public static Singleton getInstance() {
		return soleInst;
	}

	public int getI() {
		return i;
	}

	public static void getSoleInstance(Singleton soleInst) {
		Singleton.soleInst = soleInst;
}
	public void setI(int i) {
		this.i = i;
	}

}
```

2. TestClass.java

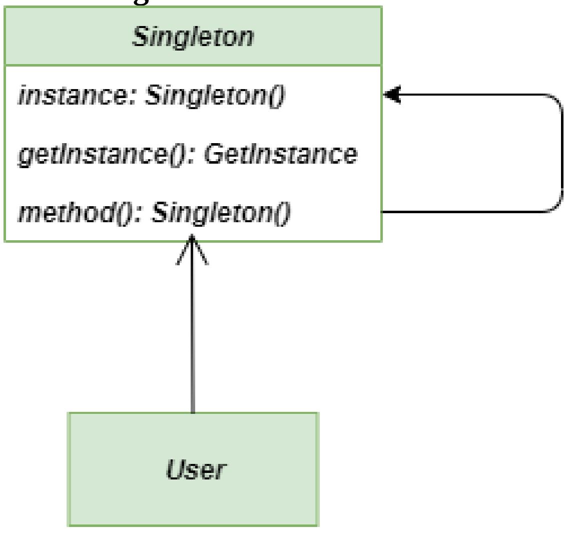
```
package design;

public class TestClass {
	public static void main(String[] args) {
		Singleton s1 = Singleton.getInstance();
		Singleton s2 = Singleton.getInstance();
		s1.setI(5);
		s2.setI(10);
		System.out.println(s1.getI());
		s2.i = s1.i + s2.i;
		System.out.println(s2.getI());
		print("S1",s1);
		print("S2",s2);
	}
	static void print(String name, Singleton obj) {
		System.out.println(String.format("Object:	%s,	Hashcode:	%d",	name,
obj.hashCode()));
	}
}
```

Output:



```
<terminated> TestClass (2) [Java Application] C:\Users\dsa\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.2.v20240123-0840\jre\bin\javaw.exe  (07-F
Created
10
20
Object: S1, Hashcode: 2036368507
Object: S2, Hashcode: 2036368507
```

Class Diagram:



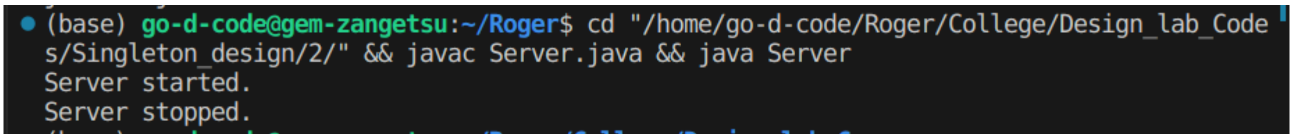
Aim:
The aim is to ensure that a class has only one instance globally accessible, facilitating efficient resource management and providing a centralized point for accessing and controlling shared state within the application.

Description:
This Java code implements a server using the Singleton design pattern. It ensures that only one instance of the server exists throughout the application, with lazy initialization for efficiency. The `getInstance()` method provides access to the singleton instance, while a private constructor prevents direct instantiation. The `Server` class includes start and stop methods for server functionality. This design facilitates centralized control and resource management in server-based applications, ensuring consistency and efficient resource utilization.

Code:

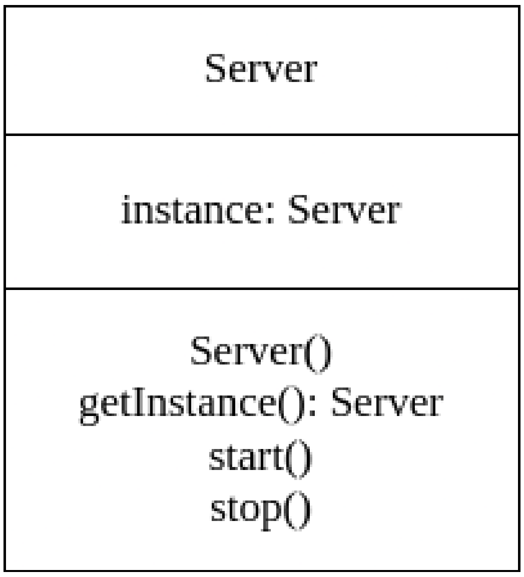
1. Server.java
```
public class Server {
    // Private static instance of the Server class
    private static Server instance;

    // Private constructor to prevent instantiation from outside
    private Server() {
        // Initialization code here
    }

    // Static method to get the instance of the Server class
    public static Server getInstance() {
        // Lazy initialization: Create instance if null
        if (instance == null) {
            synchronized (Server.class) {
                if (instance == null) {
                    instance = new Server();
                }
            }
        }
        return instance;
    }

    // Other methods of the Server class can be added here

    public void start() {
        System.out.println("Server started.");
    }

    public void stop() {
        System.out.println("Server stopped.");
    }

    public static void main(String[] args) {
        // Get the instance of the server
        Server server = Server.getInstance();
        // Use the server
        server.start();
        // ... do something with the server

        // Stop the server
        server.stop();
    }
}
```

Output:



```
● (base) go-d-code@gem-zangetsu:~/Roger$ cd "/home/go-d-code/Roger/College/Design_lab_Code
  s/Singleton_design/2/" && javac Server.java && java Server
  Server started.
  Server stopped.
```

Class Diagram:


Server
instance: Server
Server()
getInstance(): Server
start()
stop()