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:

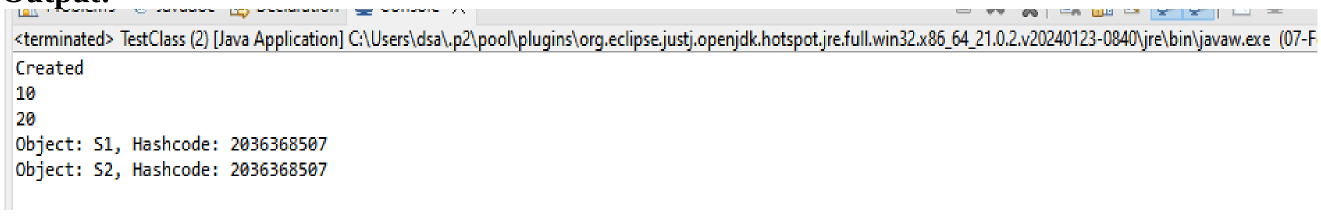
1. Singleton.java

```
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

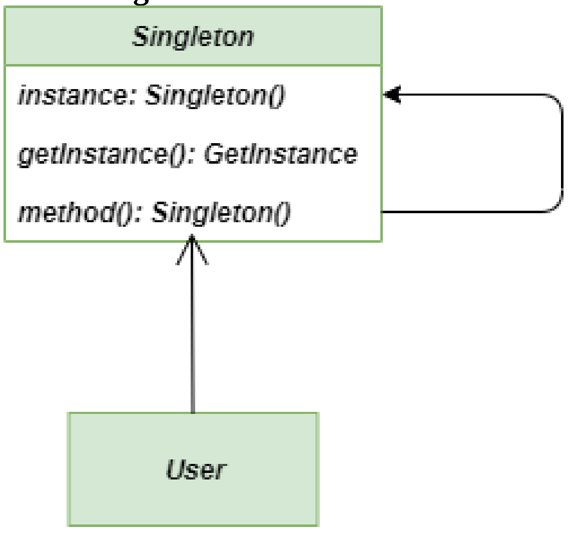
```
package design;

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.

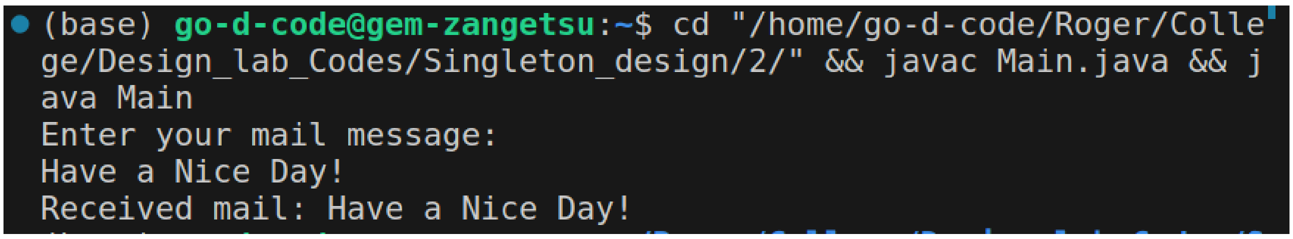
Description:
This Java code implements the Singleton design pattern with a `MailBox` class, ensuring there's only one instance of the mailbox throughout the program. The `MailBox` class has a private constructor and a static method `getInstance()` to retrieve the singleton instance, creating it if it doesn't exist. It also includes a method `receiveMail()` to handle incoming messages. The `Main` class demonstrates the usage by prompting the user to enter a mail message, obtaining the singleton instance of `MailBox`, and then receiving the mail message through the `receiveMail()` method. This pattern ensures centralized access to the mailbox instance, promoting efficient resource management and organized handling of mail messages.

Code:

1. Main.java

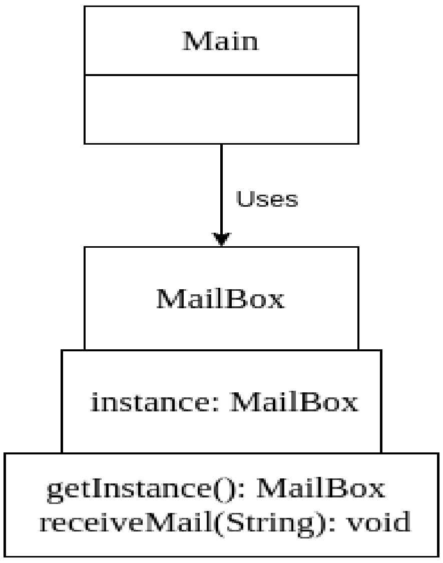
```
import java.util.Scanner;

// Singleton MailBox class
class MailBox {
    // Private static instance variable
    private static MailBox instance;

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

    // Public static method to get the instance
    public static MailBox getInstance() {
        if (instance == null) {
            instance = new MailBox();
        }
        return instance;
    }

    // Method to handle incoming mail
    public void receiveMail(String message) {
        System.out.println("Received mail: " + message);
    }
}

// Main class to demonstrate usage
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt user to enter mail message
        System.out.println("Enter your mail message:");
        String message = scanner.nextLine();
        // Get the instance of the Singleton MailBox
        MailBox mailBox = MailBox.getInstance();

        // Use the mailBox instance to receive mail
        mailBox.receiveMail(message);

        scanner.close();
    }
}
```

Output:



```
● (base) go-d-code@gem-zangetsu:~$ cd "/home/go-d-code/Roger/Colle
  ge/Design_lab_Codes/Singleton_design/2/" && javac Main.java && j
  ava Main
  Enter your mail message:
  Have a Nice Day!
  Received mail: Have a Nice Day!
```

Class Diagram:


Main
Uses
MailBox
instance: MailBox
getInstance(): MailBox
receiveMail(String): void