**Singleton Design Pattern**

**Assignment - 1**

Name - Heet Dobariya Roll No. - 22BCP177 Group - G5

* **Singleton Design :**

The Singleton design pattern is a creational pattern that ensures a class has only one instance and provides a global point of access to that instance. In other words, it restricts the instantiation of a class to a single object and provides a way to access that instance from any point in the application.

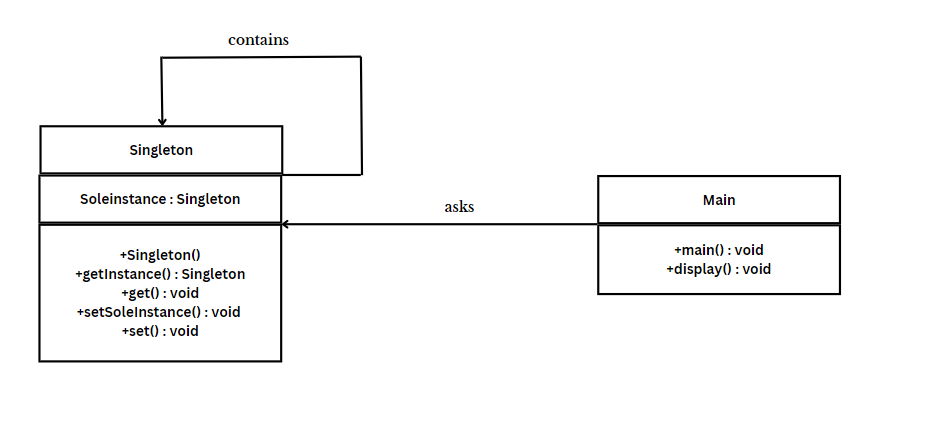
* The key features of a Singleton pattern include:

1. Single Instance: There is only one instance of the class that is created and shared.
2. Global Access: The instance is globally accessible, meaning that it can be accessed from any part of the application.
3. Lazy Initialization (optional): The instance is created only when it is first needed, not necessarily when the program starts.

A common implementation of the Singleton pattern involves a private constructor to prevent direct instantiation, a static method to provide access to the instance, and a static variable to hold the single instance.

It's important to note that while the Singleton pattern has its use cases, it should be used judiciously as it introduces a global state, which might lead to issues such as tight coupling and difficulties in unit testing. Additionally, in multi-threaded environments, special attention must be paid to ensure thread safety during the initialization of the singleton instance.

* **Program :** Give an example of Singleton Design Pattern.
* **UML Diagram :**



* **Code :**

1. **For Lazy Initialization -**

public class Singleton

{

// private static Singleton soleInstance = new Singleton();

private static Singleton soleInstance;

public int i;

private Singleton() // private constructor

{

System.out.println("Created...");

}

public static Singleton getInstance() // global access point

{

if(soleInstance == null)

{

soleInstance = new Singleton();

}

return soleInstance;

}

public int get()

{

return i;

}

public static void setSoleInstance(Singleton soleInstance)

{

Singleton.soleInstance = soleInstance;

}

public void set(int i)

{

this.i = i;

}

}

public class TestClass

{

public static void main(String[] args)

{

// Get the Singleton instance

Singleton s1 = Singleton.getInstance();

Singleton s2 = Singleton.getInstance();

s1.set(5);

s2.set(10);

System.out.println(s1.get());

s2.i = s1.i + s2.i;

System.out.println(s2.get());

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()));

}

}

1. **For Eager Initialization -**

public class Singleton

{

// Private static instance variable

private static Singleton soleInstance = new Singleton();

public int i;

// Private constructor to prevent instantiation from outside the class

private Singleton()

{

// Initialize the Singleton instance

System.out.println("Created...");

}

// Public method to provide global access to the instance

public static Singleton getInstance()

{

return soleInstance;

}

public int get()

{

return i;

}

public static void setSoleInstance(Singleton soleInstance)

{

Singleton.soleInstance = soleInstance;

}

public void set(int i)

{

this.i = i;

}

}

public class TestClass

{

public static void main(String[] args)

{

// Get the Singleton instance

Singleton s1 = Singleton.getInstance();

Singleton s2 = Singleton.getInstance();

s1.set(5);

s2.set(10);

System.out.println(s1.get());

s2.i = s1.i + s2.i;

System.out.println(s2.get());

print("S1",s1);

print("S2",s2);

}

static void print(String name, Singleton obj)

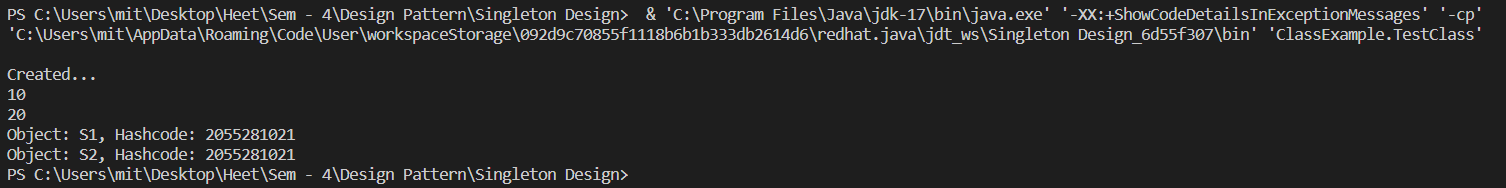
{

System.out.println(String.format("Object: %s, Hashcode: %d", name, obj.hashCode()));

}

}

* **Output :**

****