Singleton Class in Java

Difficulty Level : Easy Last Updated : 24 Jan, 2022

In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time. After the first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class through any instance, affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined.

Remember the key points while defining class as a singleton class that is while designing a singleton class:

1. Make a constructor private.
2. Write a static method that has the return type object of this singleton class. Here, the concept of Lazy initialization is used to write this static method.

Purpose of Singleton Class

The primary purpose of a Single class is to restrict the limit of the number of object creation to only one. This often ensures that there is access control to resources, for example, socket or database connection.

The memory space wastage does not occur with the use of the singleton class because it restricts the instance creation. As the object creation will take place only once instead of creating it each time a new request is made.

We can use this single object repeatedly as per the requirements. This is the reason why the multi-threaded and database applications mostly make use of the Singleton pattern in Java for caching, logging, thread pooling, configuration settings, and much more.

For example, there is a license with us, and we have only one database connection or suppose if our JDBC driver does not allow us to do multithreading, then Singleton class comes into the picture and makes sure that at a time, only a single connection or a single thread can access the connection.

How to Design/Create a Singleton Class in Java?

To create a singleton class, we must follow the steps, given below:

1. Ensure that only one instance of the class exists.

2. Provide global access to that instance by

- Declaring all constructors of the class to be private.
- Providing a static method that returns a reference to the instance. The lazy initialization concept is used to write the static methods.
- The instance is stored as a private static variable.

Example of singleton classes is **Runtime class, Action Servlet, Service Locator**. Private constructors and factory methods are also an example of the singleton class.

Difference between Normal Class and Singleton Class

We can distinguish a Singleton class from the usual classes with respect to the process of instantiating the object of the class. To instantiate a normal class, we use a java constructor. On the other hand, to instantiate a singleton class, we use the getInstance() method.

The other difference is that a normal class vanishes at the end of the lifecycle of the application while the singleton class does not destroy with the completion of an application.

Forms of Singleton Class Pattern

There are two forms of singleton design pattern, which are:

- **Early Instantiation:** The object creation takes place at the load time.
- **Lazy Instantiation:** The object creation is done according to the requirement.

Implementation: Let us brief how the singleton class varies from the normal class in java. Here the difference is in terms of instantiation as for normal class we use constructor, whereas for singleton class we use *getInstance() method* which we will be peeking out in example 1 as depicted below. In general, in order to avoid confusion, we may also use the class name as method name while defining this method which will be as depicted in example 2 below as follows.

Example 1:

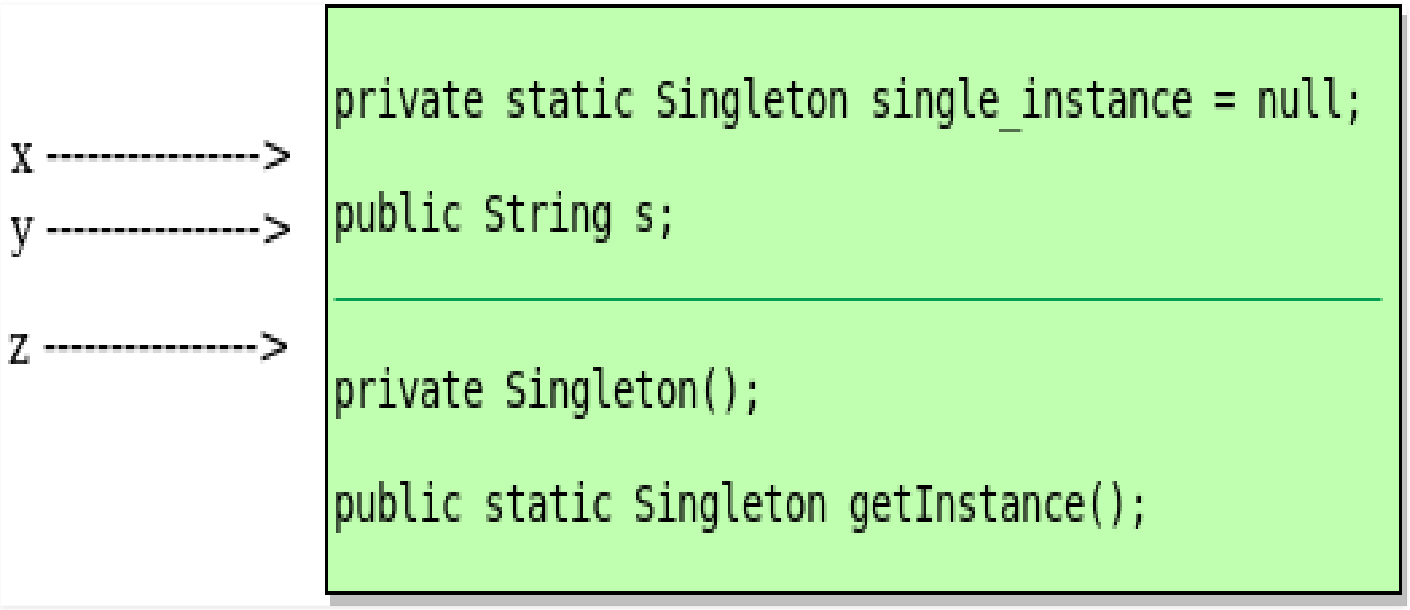
```
// Java program implementing Singleton class
// with using  getInstance() method

// Class 1
// Helper class
class Singleton {
    // Static variable reference of single_instance
    // of type Singleton
    private static Singleton single_instance = null;

    // Declaring a variable of type String
    public String s;

    // Constructor
    // Here we will be creating private constructor
    // restricted to this class itself
    private Singleton()
    {
        s = "Hello I am a string part of Singleton class";
    }

    // Static method
    // Static method to create instance of Singleton class
    public static Singleton getInstance()
    {
        if (single_instance == null)
            single_instance = new Singleton();

        return single_instance;
    }
}

// Class 2
// Main class
class GFG {
    // Main driver method
    public static void main(String args[])
    {
        // Instantiating Singleton class with variable x
        Singleton x = Singleton.getInstance();

        // Instantiating Singleton class with variable y
        Singleton y = Singleton.getInstance();

        // Instantiating Singleton class with variable z
        Singleton z = Singleton.getInstance();

        // Printing the hash code for above variable as
        // declared
        System.out.println("Hashcode of x is "
                           + x.hashCode());
        System.out.println("Hashcode of y is "
                           + y.hashCode());
        System.out.println("Hashcode of z is "
                           + z.hashCode());
```

```
            // Condition check
            if (x == y && y == z) {

                // Print statement
                System.out.println(
                    "Three objects point to the same memory location on the heap i.
            }

            else {
                // Print statement
                System.out.println(
                    "Three objects DO NOT point to the same memory location on the
            }
        }
    }
```

Output

```
Hashcode of x is 558638686
Hashcode of y is 558638686
Hashcode of z is 558638686
Three objects point to the same memory location on the heap i.e, to the same c
```

Output explanation:



In a singleton class, when we first-time call the ***getInstance() method***, it creates an object of the class with the name single_instance and returns it to the variable. Since single_instance is static, it is changed from null to some object. Next time, if we try to call the getInstance() method since single_instance is not null, it is returned to the variable,

instead of instantiating the Singleton class again. This part is done by if condition.
In the main class, we instantiate the singleton class with 3 objects x, y, z by calling the static *method getInstance()*. But actually, after the creation of object x, variables y and z are pointed to object x as shown in the diagram. Hence, if we change the variables of object x, that is reflected when we access the variables of objects y and z. Also if we change the variables of object z, that is reflected when we access the variables of objects x and y.
Now we are done with covering all aspects of example 1 and have implemented the same, now we will be implementing Singleton class with method name as that of the class name.

Example 2:

```
// Java program implementing Singleton class
// with method name as that of class

// Class 1
// Helper class
class Singleton {
    // Static variable single_instance of type Singleton
    private static Singleton single_instance = null;

    // Declaring a variable of type String
    public String s;

    // Constructor of this class
    // Here private constructor is is used to
    // restricted to this class itself
    private Singleton()
    {
        s = "Hello I am a string part of Singleton class";
    }

    // Method
    // Static method to create instance of Singleton class
    public static Singleton Singleton()
    {
        // To ensure only one instance is created
        if (single_instance == null) {
            single_instance = new Singleton();
        }
        return single_instance;
    }
}

// Class 2
```

```
// Main class
class GFG {
    // Main driver method
    public static void main(String args[])
    {
        // Instantiating Singleton class with variable x
        Singleton x = Singleton.Singleton();

        // Instantiating Singleton class with variable y
        Singleton y = Singleton.Singleton();

        // instantiating Singleton class with variable z
        Singleton z = Singleton.Singleton();

        // Now  changing variable of instance x
        // via toUpperCase() method
        x.s = (x.s).toUpperCase();

        // Print and display commands
        System.out.println("String from x is " + x.s);
        System.out.println("String from y is " + y.s);
        System.out.println("String from z is " + z.s);
        System.out.println("\n");

        // Now again changing variable of instance x
        z.s = (z.s).toLowerCase();

        System.out.println("String from x is " + x.s);
        System.out.println("String from y is " + y.s);
        System.out.println("String from z is " + z.s);
    }
}
```

Output

```
String from x is HELLO I AM A STRING PART OF SINGLETON CLASS
String from y is HELLO I AM A STRING PART OF SINGLETON CLASS
String from z is HELLO I AM A STRING PART OF SINGLETON CLASS


String from x is hello i am a string part of singleton class
String from y is hello i am a string part of singleton class
String from z is hello i am a string part of singleton class
```

Output explanation: In the singleton class, when we first-time call Singleton() method, it creates an object of class Singleton with the name single_instance and returns it to the

variable. Since single_instance is static, it is changed from null to some object. Next time if we try to call Singleton() method, since single_instance is not null, it is returned to the variable, instead of instantiating the Singleton class again.

This article is contributed by **Pavan Gopal Rayapati**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.