# Singleton Class in Java – Early Loading, Lazy Loading (with thread safe) & Enum

## Singleton Class in Java

A Singleton Class in Java is a class whose instance is created only once and the same instance is used throughout the application.

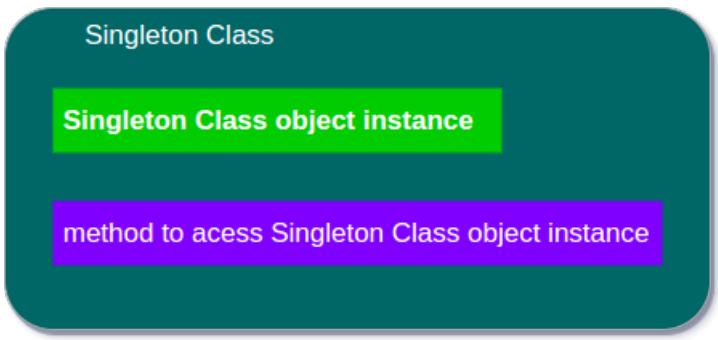
In practice Singleton is design pattern which allows the creation of instance only once. The Singleton also provides a means to access the only global instance either making the instance public or a public method to the private instance.

### When is Singleton Class used ?

Singleton Class is used when a single state has to be maintained through out the application life-cycle. As an example, for logging purposes, the same logger may have to used across all the classes and the logger has to be instantiated only once.

### What is the Structure of Singleton Class ?

Singleton Class should be a static class that has a static and final variable which holds the instance of Singleton class. And also there is a method to provide access to this instance for the other classes. Following is a picture that depicts the structure of Singleton Class :




Structure of Singleton Class in Java

## How to realize Singleton Class in Java

Singleton Class in Java could be realized in one of the four following ways :

Singleton Class in Java could be realized in one of the four following ways :

1. Early Loading
2. Lazy Loading
3. Lazy Loading – Thread Safe
4. Enum type

## Singleton Class in Java using Early Loading

The instance could be initialized directly during the instance declaration. JVM Java Virtual Machine takes care of creation of instance of the Singleton class. An example is shown below.

**SingletonCls.java**

```
/**
* Singleton Class in Java with Early loading
*/
public class SingletonCls {
    // singleton instance, this instance is created in JVM during start of the applicati
    // which is early loading
    private static final SingletonCls singletonInst = new SingletonCls();

    // a variable of singleton class
    private String message = "";

    // making constructor private so that no other class could use the default construct
    private SingletonCls() {
    }

    // the method which gives access to the only instance of SingletonCls
    public static SingletonCls getInstance(){
        return singletonInst;
    }

    // getter for the vairable message
    public String getMessage() {
        return message;
    }

    // setter for the variable message
    public void setMessage(String message) {
        this.message = message;
    }

}
```

Accessing the above Singleton class in another class is shown using ExamplePgm.java as following.

**ExamplePgm.java**

```
/**
 * Example class that demonstrates the usage of Singleton class
 */
```

```
 */
public class ExamplePgm {

    public static void main(String[] args) {
        // access instance of SingletonCls using getInstance() method
        SingletonCls instance = SingletonCls.getInstance();
        instance.setMessage("This message is set in main of ExamplePgm");
        displayMsg();
    }

    public static void displayMsg(){
        SingletonCls instance = SingletonCls.getInstance();
        System.out.println(instance.getMessage());
    }
}
```

Output to the console when the above program is run is shown below :

```
Starting of ExamplePgm..
This message is set in main of ExamplePgm
```

## Singleton Class in Java using Lazy Loading

The instance could be initialized only when the Singleton Class is used for the first time. Doing so creates the instance of the Singleton class in the JVM Java Virtual Machine during the execution of the method, which gives access to the instance, for the first time.  An example is given below.

**SingletonCls.java**

```
/**
* Singleton Class in Java with Lazy loading
*/
public class SingletonCls {
    // singleton instance declaration
    private static SingletonCls singletonInst;

    // a variable of singleton class
    private String message = "";

    // making constructor private so that no other class could use the default construct
    private SingletonCls() {
        System.out.println("Singleton instance created.");
    }

    // the method which gives access to the only instance of SingletonCls
    public static SingletonCls getInstance(){
        if(singletonInst==null){
            singletonInst = new SingletonCls();
            System.out.println("SingletonCls instance created for the first time.");
        }
        return singletonInst;
    }

    // getter for the variable message
    public String getMessage() {
```

```
        return message;
    }

    // setter for the variable message
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Accessing the above Singleton class in another class is shown using ExamplePgm.java as following.

**ExamplePgm.java**

```
/**
 * Example class that uses Singleton class and its variables
 */
public class ExamplePgm {

    public static void main(String[] args) {
        System.out.println("Starting of ExamplePgm..");
        // access instance of SingletonCls using getInstance() method
        SingletonCls instance = SingletonCls.getInstance();
        instance.setMessage("This message is set in main of ExamplePgm");
        displayMsg();
    }

    public static void displayMsg(){
        // SingletonCls.getInstance() gets the instance that is already created during t
        SingletonCls instance = SingletonCls.getInstance();
        System.out.println(instance.getMessage());
    }
}
```

Output to the console when the above program is run is shown below :

```
Starting of ExamplePgm..
Singleton instance created.
SingletonCls instance created for the first time.
This message is set in main of ExamplePgm
```

## Singleton Class in Java using Lazy Loading that is thread safe

The instance could be initialized only when the Singleton Class is used for the first time. Doing so creates the instance of the Singleton class in the JVM Java Virtual Machine during the execution of the method, which gives access to the instance, for the first time. In multi-threaded environment, it is a possible situation that two threads might enter the getInstance() method at a same time, which could create two instances of Singleton class which is not desirable and might make the program behavior unpredictable. To make the creation of Singleton instance thread safe, the method getInstance() is **synchronized** so that the method is executed by only one thread at a time.  An example is shown below.

**SingletonCls.java**

```
/**
* Singleton Class in Java with Lazy loading
*/
public class SingletonCls {
    // singleton instance declaration
    private static SingletonCls singletonInst;

    // a variable of singleton class
    private String message = "";

    // making constructor private so that no other class could use the default construct
    private SingletonCls() {
        System.out.println("Singleton instance created.");
    }

    // the method which gives access to the only instance of SingletonCls, is thread saf
    public static synchronized SingletonCls getInstance(){
        if(singletonInst==null){
            singletonInst = new SingletonCls();
            System.out.println("SingletonCls instance created for the first time.");
        }
        return singletonInst;
    }

    // getter for the variable message
    public String getMessage() {
        return message;
    }

    // setter for the variable message
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Accessing the above Singleton class in another class is shown using ExamplePgm.java as following.

**ExamplePgm.java**

```
/**
 * Example class that uses Singleton class and its variables
 */
public class ExamplePgm {

    public static void main(String[] args) {
        System.out.println("Starting of ExamplePgm..");
        // access instance of SingletonCls using getInstance() method
        SingletonCls instance = SingletonCls.getInstance();
        instance.setMessage("This message is set in main of ExamplePgm");
        displayMsg();
    }

    public static void displayMsg(){
        // SingletonCls.getInstance() gets the instance that is already created during t
        SingletonCls instance = SingletonCls.getInstance();
```

```
        System.out.println(instance.getMessage());
    }
}
```

Output to the console when the above program is run is shown below :

```
Starting of ExamplePgm..
Singleton instance created.
SingletonCls instance created for the first time.
This message is set in main of ExamplePgm
```

## Singleton Class in Java using Enum type

Consider the following block of code:

```
public enum SingletonEnumEx {
    INSTANCE;
}
```

Enum type SingletonEnumEx has an implicitly declared **public static final** field of type SingletonEnumEx for each constant declared in the body of SingletonEnumEx. In other words JVM considers the constant INSTANCE is of type SingletonEnumEx and also public static final. This is the behavior of class instance required by design of Singleton.

The enum members are explained clearly in the java doc here[https://docs.oracle.com/javase/specs/jls/se8/html/jls-8.html#jls-8.9.3].

Java Compiler takes care of the implicit declaration and is thread safe.

Thus, using this behavior, Singleton Class could be achieved using enum as shown in the following example. Enum type for Singleton design.

**SingletonEnumEx.java**

```
/**
 * Singleton Design implementation using enum
 */
public enum SingletonEnumEx {
    // the variable INSTANCE shall be compiled to a public static final field of type Si
    INSTANCE;
    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
```

```
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Class that uses SingletonEnumEx

**ExamplePgm.java**

```
/**
 * Example class that uses Singleton class and its variables
 */
public class ExamplePgm {

    public static void main(String[] args) {
        System.out.println("Starting of ExamplePgm..");
        // access instance of SingletonCls using getInstance() method
        SingletonEnumEx.INSTANCE.setMessage("This message is set in main of ExamplePgm")
        displayMsg();
    }

    public static void displayMsg(){
        System.out.println(SingletonEnumEx.INSTANCE.getMessage());
    }
}
```

## Conclusion

Concluding this Java Tutorial on Singleton Class in Java, we have seen four ways of programming the design pattern.


**Java Tutorial**

- Java Tutorial
- Java Introduction
- Java Installation
- IDEs for Java Development
- Java HelloWorld Program
- Java Program Structure
- Java Datatypes
- Java Variable Types
- Java Access Modifiers

- Java Operators
- Java Decision Making
- Java Loops
- Java Array
- Java OOPs
- Java String
- Java Exception Handling
- Java File Operations
- Java Date & Time
- Java MySQL
- Java Random
- Java Math