Immutable Class:

String & all wrapper classes are an immutable class.

Immutable means once an object is created, we cannot change its content. In Java, all the [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/) (like Integer, Boolean, Byte, Short) and String class is immutable.

How to create Immutable class:

* + The class must be declared as final so that child classes can’t be created.
  + Data members in the class must be declared private so that direct access is not allowed.
  + Data members in the class must be declared as final so that we can’t change the value of it after object creation.
  + A parameterized constructor should initialize all the fields performing a deep copy so that data members can’t be modified with an object reference.
  + Deep Copy of objects should be performed in the getter methods to return a copy rather than returning the actual object reference)

***Note:*** There should be no setters or in simpler terms, there should be no option to change the value of the instance variable.

*Code:*

final class Create{

Private int I;

Create(int i){

this.i=i;

}

Public Create Modify(int i){

If(this.i==i) return this;

Else return(new create(i));

}

Public svm(){

Create c1=new Create(10);

Create c2=C1.Modify(100);

Create c3 =C1.Modify(10);

Syso(c1==c2)//false;

Syso(c1==c3);//true

Create c4=C1.Modify(100);

Sop(c2==c4);//false

}

Difference between immutable and singleton:

An immutable object is initialized by its constructor only, while a singleton is instantiated by a static method. A set of functions (or static methods) which manipulate some shared mutable state constitute a singleton. This implies that every mutable member of a singleton collection is itself a singleton.

Singleton object will have only one object and multiple states and Immutable object will have each new object with each time the state is changed.

Creation of Singleton class:

Private Constructor-So that instance will create only from within static method of class;

Use Factory method & null if the class has already been instantiated.

Override clone() & return the above created object to surpress the object creation through cloning process.

**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.e, to the same object");

        }

**else** {

            // Print statement

            System.out.println(

                "Three objects DO NOT point to the same memory location on the heap");

        }

    }

}

Thread Safe Singleton

public class ASingleton {

private static volatile ASingleton instance;  
        private static Object mutex = new Object();

private ASingleton() {  
        }

public static ASingleton getInstance() {  
                ASingleton result = instance;  
                if (result == null) {  
                        synchronized (mutex) {  
                                result = instance;  
                                if (result == null)  
                                        instance = result = new ASingleton();  
                        }  
                }  
                return result;  
        }

}

Volatile keyword :-

A volatile keyword is **used to modify the value of a variable by different threads**. It is also used to make classes thread-safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem. The volatile keyword can be used either with primitive type or objects.

Transient Keyword :-

transient is **a variables modifier used in serialization**. At the time of serialization, if we don't want to save value of a particular variable in a file, then we use transient keyword. When JVM comes across transient keyword, it ignores original value of the variable and save default value of that variable data type.