**Java provides three String classes:**

****1) java.lang.String                     (From JDK 1.0)****

****2) java.lang.StringBuffer            (From JDK 1.5)****

****3) java.lang.StringBuilder           (From JDK 1.5)****

1. All three classes implement ****Serializable**** and ****CharSequence**** interface.
2. In all three classes, ****toString()**** method is overrided. So. whenever you use reference variables of these three types, they will return contents of the objects not physical address of the objects.
3. ****hashCode()**** and ****equals()**** methods are overrided only in *java.lang.String* class but not in *java.lang.StringBuffer* and *java.lang.StringBuilder* classes.
4. There is no ****reverse()**** and ****delete()**** methods in String class. But, StringBuffer and StringBuilder have reverse() and delete() methods.
5. In case of String class, you can create the objects without ****new**** operator. But in case of StringBuffer and StringBuilder class, you have to use new operator to create the objects

# [An Example To Prove Strings Are Immutable](https://javaconceptoftheday.com/example-to-prove-strings-are-immutable/)

**public** **class** StringExamples

{

**public** **static** **void** main(String[] args)

    {

        String s1 = "JAVA";

        String s2 = "JAVA";

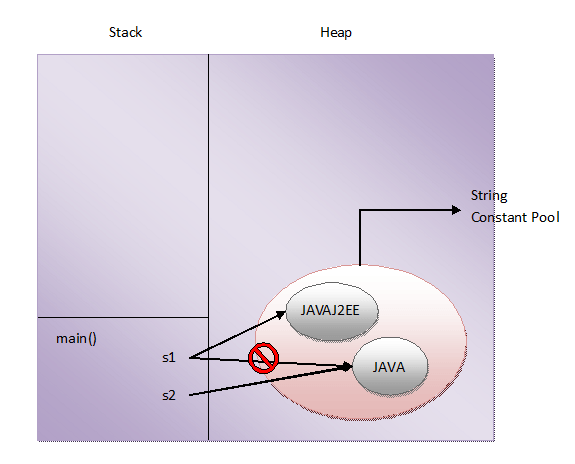
        System.out.println(s1 == s2);         //Output : true

        s1 = s1 + "J2EE";

        System.out.println(s1 == s2);         //Output : false

    }

}



## is new String() also immutable?

****The answer is Yes****. String objects created using new operator are also immutable although they are stored in the heap memory. This can be also proved with help of an example.

**public** **class** StringExamples

{

**public** **static** **void** main(String[] args)

    {

        String s1 = **new** String("JAVA");

        System.out.println(s1);         //Output : JAVA

        s1.concat("J2EE");

        System.out.println(s1);         //Output : JAVA

    }

}

****Immutability is the fundamental property of string objects. In whatever way you create the string objects, either using string literals or using new operator, they are immutable.****

****“==” operator****, ****equals() method**** and ****hashcode() method****s are used to check the equality of any type of objects in Java.

****“==” operator**** compares the two objects on their physical address. That means if two references are pointing to same object in the memory, then comparing those two references using “==” operator will return true. For example, if s1 and s2 are two references pointing to same object in the memory, then invoking ****s1 == s2**** will return true. This type of comparison is called ****“Shallow Comparison”****.

****equals() method****, if not overrided, will perform same comparison as “==” operator does i.e comparing the objects on their physical address. So, it is always recommended that you should override equals() method in your class so that it provides field by field comparison of two objects. This type of comparison is called ****“Deep Comparison”****.