**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";

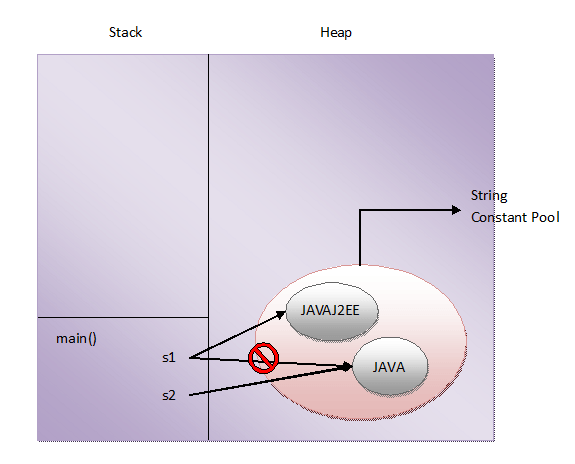
        System.out.println(s1 == s2);         //Output : true

        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”****.

In java.lang.String class, equals() method is overrided to provide the comparison of two string objects based on their contents. That means, any two string objects having same content will be equal according to equals() method. For example, if s1 and s2 are two string objects having the same content, then invoking ****s1.equals(s2)**** will return true.

****hashCode() method**** returns hash code value of an object in the Integer form. It is recommended that whenever you override equals() method, you should also override hashCode() method so that two equal objects according to equals() method must return same hash code values. This is the general contract between equals() and hashCode() methods that must be maintained all the time.

#### **Contract between hashCode() and equals()**

It is generally necessary to override the hashCode() method whenever equals() method is overridden, so as to maintain the general contract for the hashCode() method, which states that ****equal objects must have equal hash codes****.

* Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified.  
  This integer need not remain consistent from one execution of an application to another execution of the same application.
* If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
* It is not required that if two objects are unequal according to the [equals(java.lang.Object)](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html" \l "equals-java.lang.Object-) method, then calling the hashCode method on each of the two objects must produce distinct integer results.  
  However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

Define two string objects like below,

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "JAVA";    String s2 = "JAVA"; |

Now apply above methods on these two objects.

****s1 == s2**** —> will return true as both are pointing to same object in the constant pool.  
****s1.equals(s2)**** —> will also return true as both are referring to same object.  
****s1.hashCode() == s2.hashCode()**** —> It also returns true.

|  |  |
| --- | --- |
| 1  2  3 | String s1 = **new** String("JAVA");    String s2 = **new** String("JAVA"); |

****s1 == s2**** —> will return false because s1 and s2 are referring to two different objects in the memory.  
****s1.equals(s2)**** —> will return true as both the objects have same content.  
****s1.hashCode() == s2.hashCode()**** —> It will also return true because two equals string objects according to equals() method will have same hash code values.

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "JAVA";    String s2 = **new** String("JAVA"); |

****s1 == s2**** —> will return false because s1 and s2 are referring to two different objects in the memory.  
****s1.equals(s2)**** —> will return true as both the objects have same content.  
****s1.hashCode() == s2.hashCode()**** —> It will also return true.

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "0-42L";    String s2 = "0-43-"; |
|  |  |

****s1 == s2**** —> will return false as s1 and s2 are referring to two different objects in the memory. ****(Expected…)****  
****s1.equals(s2)**** —> It will also return false as both the objects have different content. ****(Expected…)****  
****s1.hashCode() == s2.hashCode()**** —> It will return true. ****(???….)****

This is because, ****two unequal string objects according to equals() method may have same hash code values****. Therefore, it is recommended not to use hashCode() method to compare two string objects. You may not get expected result.

## Conclusion :

When you want to check the equality of two string objects on their physical existence in the memory, then use “==” operator. If you want to check the equality of two string objects depending upon their contents, then use equals() method. It is recommended not to use hashCode() method to check the equality of two string objects. You may get unexpected result.

****java.lang.String**** class provides many methods to extract the characters from a string object:

1. charAt() Method
2. getChars() Method
3. toCharArray() Method
4. subString() Method

Java provides 4 methods to compare the strings.

1) ****equals()**** – This method returns true if contents of two string objects are same.  
2) ****equalsIgnoreCase()**** – This method compares two string objects but ignores the case of the characters when comparing.  
3) ****compareTo()**** – This method compares one string with another and returns an integer if the string is smaller or equal or greater than the other string.  
4) ****compareToIgnoreCase()**** – This method is same as compareTo() but ignores the case of the characters when comparing.

System.***out***.println("abc".compareTo("abc")); //Output : 0

System.***out***.println("abc".compareTo("abcd")); //Output : -1

System.***out***.println("abcd".compareTo("abc")); //Output : 1

**String Imp points:**  
  
Java doesn’t support operator overloading except ‘****+****‘ operator. ‘****+****‘ can be used for number addition as well as to concatenate two string objects. This is the special treatment given by the Java to string objects.

If you are performing lots of string concatenation in your code, then use either StringBuffer or StringBuilder classes. These two classes give better performance than String class.

Strings in java are backed by ****character array****. You can retrieve this array using toCharArray() method of String class.

# [20 Things You Should Know About Strings In Java](https://javaconceptoftheday.com/tutorial-examples-strings-in-java/)

****Strings in java**** are most used data types while developing any kind of applications. Hence, strings are treated as very special in java. This article contains 20 important points about strings in java. These points are also most discussed ones in the java interviews.

****1)**** In Java, you can create string objects in two ways. One is using ****new**** operator and another one is using string ****literals****.

|  |  |
| --- | --- |
| 1  2  3 | String s1 = "abc";        //Creating string object using string literal    String s2 = **new** String("abc");          //Creating string object using new operator |

****2)**** String objects created using string literals are stored in ****String Constant Pool**** and string objects created using new operator are stored in the ****heap memory****. Click [here](https://javaconceptoftheday.com/how-the-strings-are-stored-in-the-memory/" \o "How The Strings Are Stored In The Memory?" \t "https://javaconceptoftheday.com/tutorial-examples-strings-in-java/_blank) to see how strings are stored in the memory.

****3)**** What Is String Constant Pool?

String objects are most used data objects in Java. Hence, java has a special arrangement to store the string objects. String Constant Pool is one such arrangement. String Constant Pool is the memory space in heap memory specially allocated to store the string objects created using string literals. In String Constant Pool, there will be no two string objects having the same content.

Whenever you create a string object using string literal, JVM first checks the content of the object to be created. If there exist an object in the string constant pool with the same content, then it returns the reference of that object. It doesn’t create a new object. If the content is different from the existing objects then only it creates new object.

****4)**** String is a ****derived type****, not a primitive type like int, double etc. Strings are objects in java.

****5)**** String objects in java are ****immutable****. That means, once you create String objects, you can’t modify them. If you try to modify them, a new object will be created with modifications.

****6)**** To overcome the immutability of String objects, two more classes are introduced in Java. They are ****StringBuffer**** and ****StringBuilder**** classes. Objects of StringBuffer and StringBuilder class are mutable.

****7)**** All three classes – String, StringBuffer and StringBuilder are ****final****. That means you can’t extend them. All three classes are members of ****java.lang**** package.

****8)**** In all three classes – String, StringBuffer and StringBuilder, ****toString()**** method is overridden. That means, whenever you use references to objects of these classes, actual content of those objects will be retrieved.

****9)**** ****equals()**** and ****hashCode()**** methods are overridden in String class but they are not overridden in StringBuffer and StringBuilder classes.

****10)**** String and StringBuffer objects are ****thread safety**** where as StringBuilder objects are not thread safety.

****11)**** Using “****==****“, ****equals()**** and ****hashCode()**** on String objects.

All three – “==”, equals() and hashCode() are used to check the equality of two string objects. If you want to check the equality of two string objects based on their physical address, then use “==” operator. If you want to check the equality of two string objects based on their content, then use equals() method. It is recommended not to use hashCode() method to compare the string objects. You may get unexpected results. Click [here](https://javaconceptoftheday.com/when-to-use-equals-hashcode-on-strings/" \o "When To Use \“==\”, equals() And hashCode() On Strings" \t "https://javaconceptoftheday.com/tutorial-examples-strings-in-java/_blank) to see when to use “==”, equals() and hashcode() on strings.

****12)**** Strings in java are backed by ****character array****. You can retrieve this array using toCharArray() method of String class.

****13)**** If you are performing lots of string concatenation in your code, then use either StringBuffer or StringBuilder classes. These two classes give better performance than String class. Click [here](https://javaconceptoftheday.com/stringbuffer-stringbuilder-string-class/" \o "How StringBuffer And StringBuilder Differ From String Class" \t "https://javaconceptoftheday.com/tutorial-examples-strings-in-java/_blank) to see the differences between String, StringBuffer and StringBuilder classes.

****14)**** Java doesn’t support operator overloading except ‘****+****‘ operator. ‘****+****‘ can be used for number addition as well as to concatenate two string objects. This is the special treatment given by the Java to string objects.

****15)**** Java provides 4 methods to compare the strings.

1) ****equals()**** – This method returns true if contents of two string objects are same.  
2) ****equalsIgnoreCase()**** – This method compares two string objects but ignores the case of the characters when comparing.  
3) ****compareTo()**** – This method compares one string with another and returns an integer if the string is smaller or equal or greater than the other string.  
4) ****compareToIgnoreCase()**** – This method is same as compareTo() but ignores the case of the characters when comparing.

****16)**** You need not to create objects to access the String class methods. You can do so using ****string literals**** also. Look at the below example.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **public** **class** MainClass  {  **public** **static** **void** main(String[] args)      {          System.out.println("abc".charAt(0));          //Output : a            System.out.println("abc".equalsIgnoreCase("ABC"));      //Output : true            System.out.println("abc".compareTo("abc"));         //Output : 0            System.out.println("abc".indexOf('c'));        //Output : 2      }  } |

****17)**** What Is String Intern?

String object in the string constant pool is called as ****String Intern****. You can create an exact copy of heap memory string object in string constant pool. This process of creating an exact copy of heap memory string object in string constant pool is called ****interning****. ****intern()**** method is used for interning. Click [here](https://javaconceptoftheday.com/string-intern-java/" \o "What Is String Intern In Java?" \t "https://javaconceptoftheday.com/tutorial-examples-strings-in-java/_blank) to see more about string intern in java.

****18)**** ****indexOf(),**** ****lastIndexOf()**** and ****matches(String regex)**** are the methods to perform search within a string.

****19)**** Unlike in C and C++, Strings in java are not terminated with ****null**** character. Strings are treated as objects in java.

****20)**** Java provides lots of in built methods to manipulate the string objects.