**Strings**

**How to create Immutable class?**

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below:

**Example to create Immutable class**

|  |
| --- |
| In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method. |

1. public final class Employee{
2. final String pancardNumber;
4. public Employee(String pancardNumber){
5. this.pancardNumber=pancardNumber;
6. }
8. public String getPancardNumber(){
9. return pancardNumber;
10. }
12. }

The above class is immutable because:

* The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
* The class is final so we cannot create the subclass.
* There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

### Why string objects are immutable in java?

|  |
| --- |
| Because java uses the concept of string literal.Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java. |

**Where values Store in memory?**

String s="Sachin";

s.concat(" Tendulkar");//concat() method appends the string at the end  --------🡪goes to string constant pool with value Sachi RTendulkar

System.out.println(s);// Sachin----

# Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | String class is immutable. | StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

# Difference between StringBuffer and StringBuilder

There are many differences between StringBuffer and StringBuilder. A list of differences between StringBuffer and StringBuilder are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is less efficient than StringBuilder. | StringBuilder is more efficient than StringBuffer. |

StringBuffer sb1 = new StringBuffer(“microsoft”);

StringBuffer sb2 = new StringBuffer(“microsoft”);

System.out.println( sb1 == sb2 );//false

System.out.println( sb1.equals(sb2) ); //false------StringBuffer doesn't override equals(); its equals() method is the one inherited from java.lang.Object, which returns true if and only if "==" would be true. String, on the other hand, does override equals(); its definition returns true of the two objects contain the same sequence of characters.