**Strings:**

**We can create String in 2 ways:**

1.Using string literal- String s = “Welcome”;

2.Using New Keyword- String s=new String(“Welcome”);

**Why java uses the concept of string literal?**

String s = “Welcome”;

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

**New keyword:-**

In such a case, JVM will create a new string object in normal (non pool) heap memory and the literal “Welcome” will be placed in the string constant pool.

The variable s will refer to the object in heap (non pool).

**String objects are stored in a special memory area known as string constant pool.**

**Suppose we create,**

String str="Welcome";

String str2="Welcome";

Firstly, JVM will not find any string object with the value “Welcome” in string constant pool, so it will create a new object.

After that it will find the string with the value “Welcome” in the pool, it will not create new object for the second reference str2but will return the reference to the same instance.

**Memory allotment of String:**

Whenever a String Object is created as a literal, the object will be created in the String constant pool.This allows JVM to optimize the initialization of String literal.

The string can also be declared using new operator i.e. dynamically allocated. In case of String are dynamically allocated they are assigned a new memory location in heap.

This string will not be added to String constant pool.

**If you want to store this string in the constant pool then you will need to “intern” it.**

String internedString = str.intern();

**In java, objects of String are immutable which means a constant and cannot be changed once created.**

**Why did the String pool move from PermGen to the normal heap area?**

PermGen space is limited, the default size is just 64 MB. It was a problem with creating and storing too many string objects in PermGen space. That’s why the String pool was moved to a larger heap area. To make Java more memory efficient, the concept of string literal is used. By the use of the ‘new’ keyword, The JVM will create a new string object in the normal heap area even if the same string object is present in the string pool.

**A clear picture how strings are store.**

**class** StringStorage {

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

    {

        String s1 = "TAT";

        String s2 = "TAT";

        String s3 = **new** String("TAT");

        String s4 = **new** String("TAT");

        System.out.println(s1);

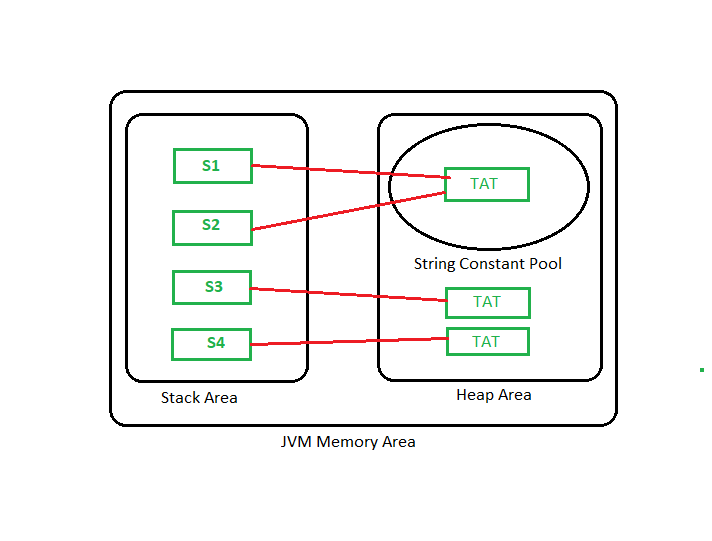
        System.out.println(s2);

        System.out.println(s3);

        System.out.println(s4);

    }

}



**class** Demo

{

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

    {

      String s="Sachin";

      s.concat(" Tendulkar");concat() method appends the string at the end

      System.out.println(s); will print Sachin because strings are immutable objects

    }

}

**Why string objects are immutable in java?**

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers 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.

**StringBuffer class in Java**

StringBuffer is a peer class of String that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences. StringBuffer may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are needed, to allow room for growth.

**Important Constructors of StringBuffer class**

StringBuffer(): creates an empty string buffer with the initial capacity of 16.

StringBuffer(String str): creates a string buffer with the specified string.

StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

**StringBuilder Class in Java**

StringBuilder in Java represents a mutable sequence of characters. Since the String Class in Java creates an immutable sequence of characters, the StringBuilder class provides an alternative to String Class, as it creates a mutable sequence of characters.

**Constructors in Java StringBuilder Class**

StringBuilder(): Constructs a string builder with no characters in it and an initial capacity of 16 characters.

StringBuilder(int capacity): Constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.

StringBuilder(CharSequence seq): Constructs a string builder that contains the same characters as the specified CharSequence.

StringBuilder(String str): Constructs a string builder initialized to the contents of the specified string.

**Difference between StringBuffer and String Bulider.**

However, the StringBuilder class differs from the StringBuffer class on the basis of synchronization. The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does. Therefore this class is designed for use as a drop-in replacement for StringBuffer in places where the StringBuffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to StringBuffer as it will be faster under most implementations. Instances of StringBuilder are not safe for use by multiple threads. If such synchronization is required then it is recommended that StringBuffer be used. String Builder is not thread-safe and high in performance compared to String buffer.