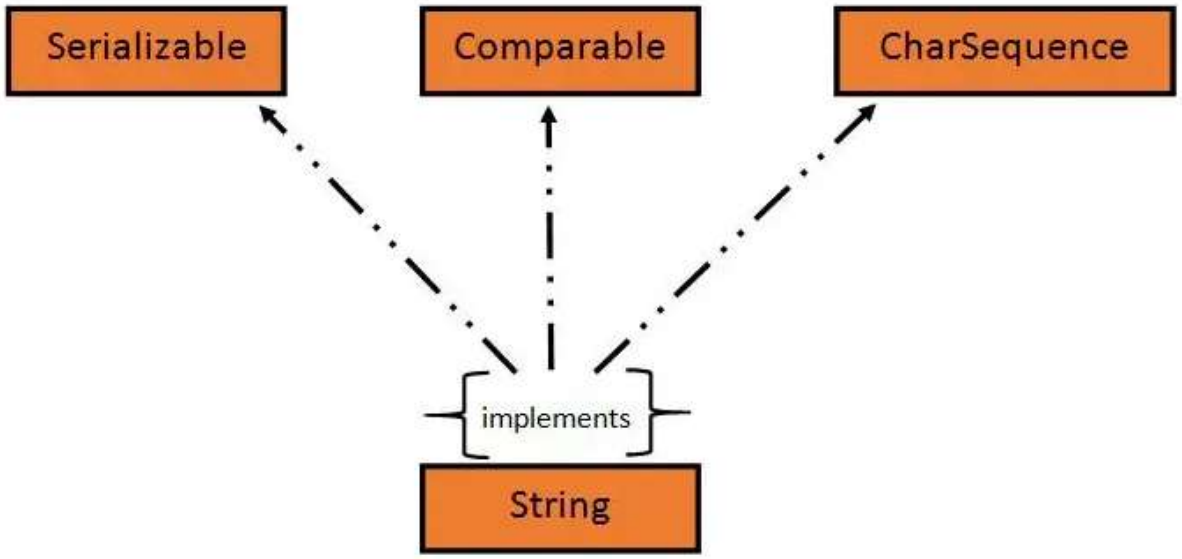
Introduction to Java Strings

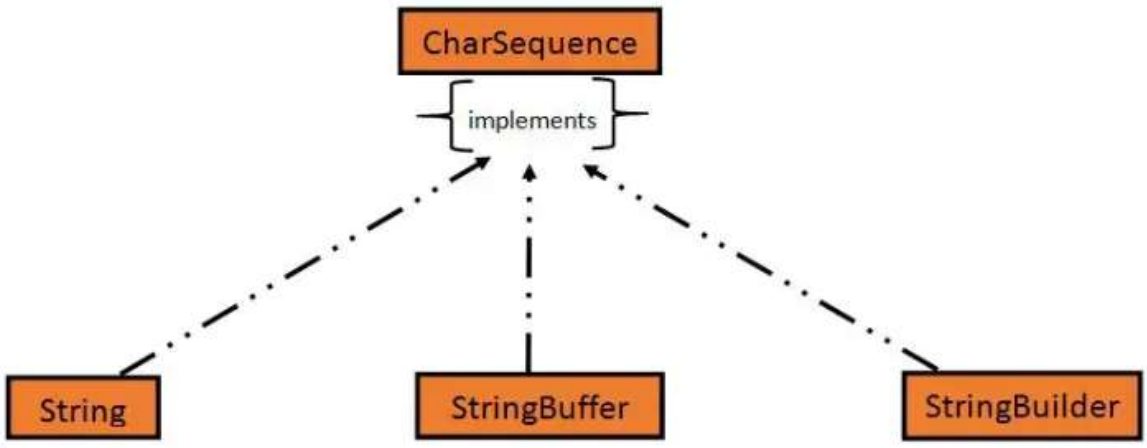
String is an object that represents sequence of characters. In Java, String is represented by String class which is located into java.lang package.

It is probably the most commonly used class in java library. In java, every string that we create is actually an object of type String. One important thing to notice about string object is that string objects are immutable that means once a string object is created it cannot be changed.

The Java String class implements Serializable, Comparable and CharSequence interface that we have represented using the below image.



In Java, CharSequence Interface is used for representing a sequence of characters. CharSequence interface is implemented by String, StringBuffer and StringBuilder classes. These three classes can be used for creating strings in java.



What is an Immutable object?

An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper classes objects are immutable.

Creating a String object

String can be created in number of ways, here are a few ways of creating string object.

1) Using a String literal

String literal is a simple string enclosed in double quotes " ". A string literal is treated as a String object.

```
public class Demo{
   public static void main(String[] args) {
        String s1 = "Hello Java";
        System.out.println(s1);
   }
}
```

2) Using new Keyword

We can create a new string object by using new operator that allocates memory for the object.

```
public class Demo{
   public static void main(String[] args) {
        String s1 = new String("Hello Java");
        System.out.println(s1);
   }
}
```

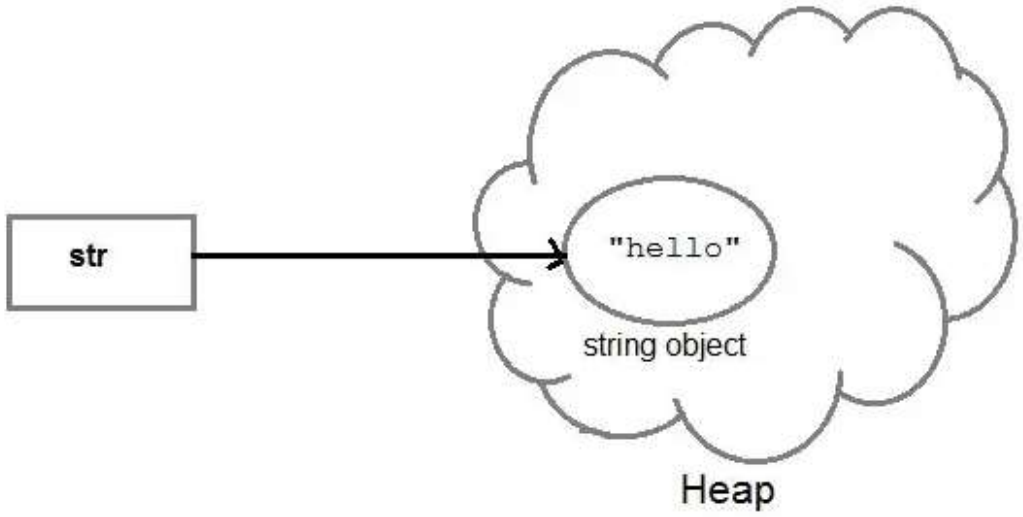
Each time we create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned. If string does not exist in the pool, a new string object is created, and

is placed in the pool. String objects are stored in a special memory area known as string constant pool inside the heap memory.

String objects and How they are stored

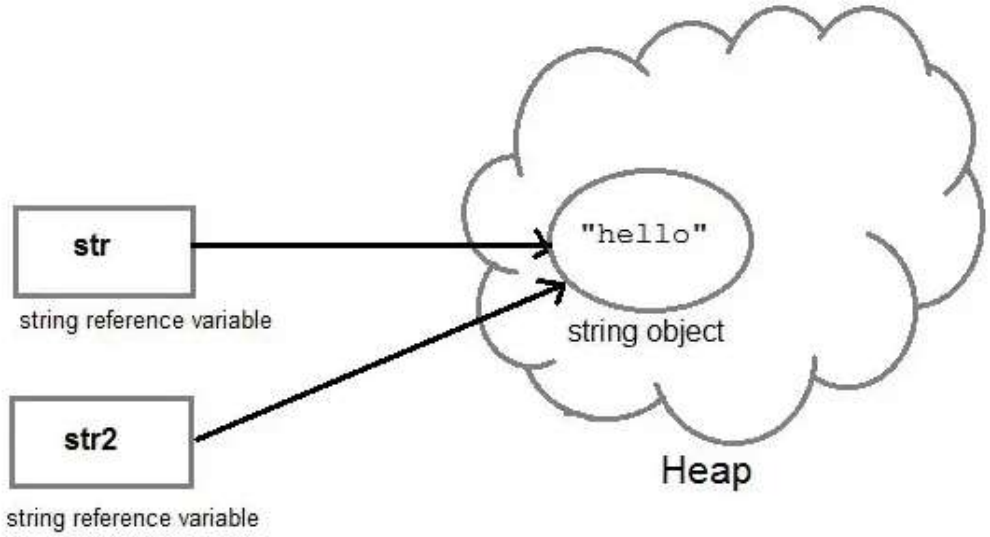
When we create a new string object using string literal, that string literal is added to the string pool, if it is not present there already.

String str= "Hello";



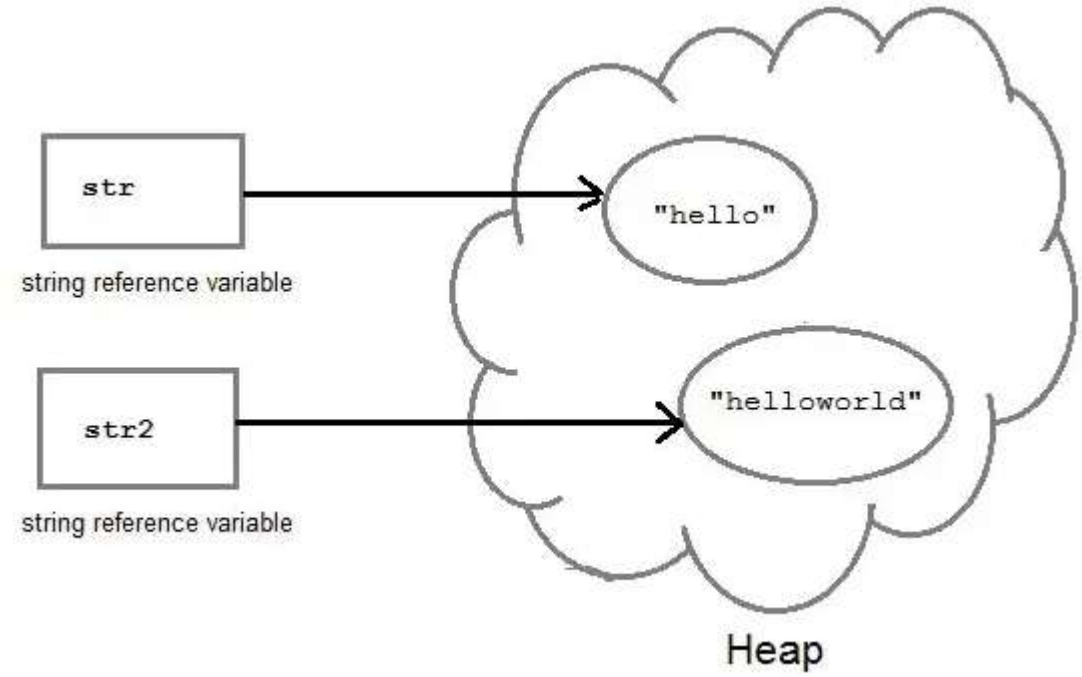
And, when we create another object with same string, then a reference of the string literal already present in string pool is returned.

String str2 = str;



But if we change the new string, its reference gets modified.

str2=str2.concat("world");



- **Concatenating String**

There are 2 methods to concatenate two or more string.

1) Using concat() method

2) Using + operator

1) Using concat() method

Concat() method is used to add two or more string into a single string object. It is string class method and returns a string object.

```
class Demo{
   public static void main(String[] args) {
        String s = "Hello";
        String str = "Java";
        String str1 = s.concat(str);
        System.out.println(str1);
```

```
    }
}
```

2) Using + operator

Java uses "+" operator to concatenate two string objects into single one. It can also concatenate numeric value with string object. See the below example.

```
class Demo{
    public static void main(String[] args) {
        String s = "Hello";
        String str = "Java";
        String str1 = s+str;
        String str2 = "Java"+11;
        System.out.println(str1);
        System.out.println(str2);
    }
}
```

- **String Comparison**

To compare string objects, Java provides methods and operators both. So we can compare string in following three ways.

Using equals() method

Using == operator

By CompareTo() method

Using equals() method

equals() method compares two strings for equality. Its general syntax is,

boolean equals (Object str)

Example

It compares the content of the strings. It will return true if string matches, else returns false.

```
class Demo{
   public static void main(String[] args) {
        String s = "Hell";
        String s1 = "Hello";
        String s2 = "Hello";
        boolean b = s1.equals(s2);
        System.out.println(b);
        b =     s.equals(s1);
        System.out.println(b);
   }
}
```

Using == operator

The double equal (==) operator compares two object references to check whether they refer to same instance. This also, will return true on successful match else returns false.

```
class Demo{
   public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Java";
        String s3 = new String ("Java");
        boolean b = (s1 == s2);
        System.out.println(b);
        b =     (s1 == s3);
```

```
        System.out.println(b);
    }
}
```

By compareTo() method

String compareTo() method compares values and returns an integer value which tells if the string compared is less than, equal to or greater than the other string. It compares the String based on natural ordering i.e alphabetically. Its general syntax is.

Syntax:

```
int compareTo(String str)
```

Example:

```
class HelloWorld{
    public static void main(String[] args) {
        String s1 = "Abhi";
        String s2 = "Viraaj";
        String s3 = "Abhi";
        int a = s1.compareTo(s2);     //return -21 because s1 < s2
        System.out.println(a);
        a = s1.compareTo(s3);     //return 0 because s1 == s3
        System.out.println(a);
        a = s2.compareTo(s1);     //return 21 because s2 > s1
        System.out.println(a);
    }
}
```