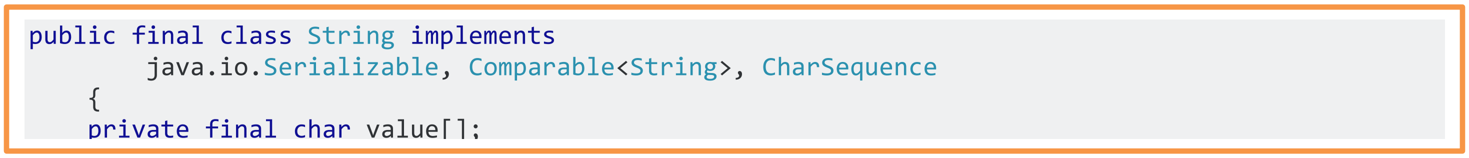
Java Strings

In java String is object that represents sequence of character values. Please note it is not array of char values because array is mutable and value can be changed. String implements CharSequence interface.

String implements these three interfaces

1. CharSequence
2. Serializable
3. Comparable



```
public final class String implements
        java.io.Serializable, Comparable<String>, CharSequence
    {
    private final char value[];
```

Java provides three classes for Strings

1. Java.lang.String
2. Java.lang.StringBuffer
3. Java.lang.StringBuilder

All these classes are final classes.

There is no reverse() or delete() method for String class but StringBuffer and StringBuilder have these methods.

String object can be created without using new operator but StringBuffer and StringBuilder can not be created without using new operator.

How String is stored in memory?

Strings are created using two ways- using literal and new keyword.

String str1=”Java”;

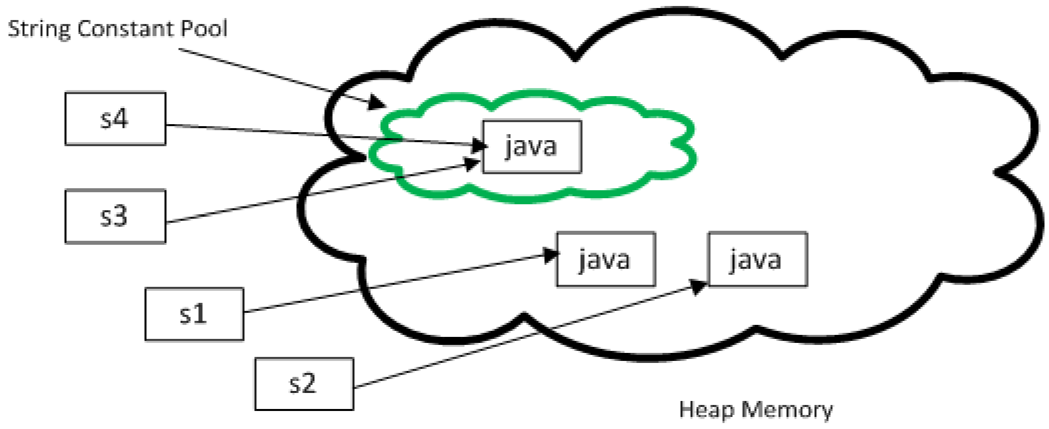
String str2=new String(“J2ee”);

In java allocated memory for a program is divided into two parts **stack and heap**. Stack is used for execution purpose while heap is used for storage purpose. Some part of heap memory is allocated to strings created using literal called ***String Constant Pool***. When string object is created using literal then it is stored in String Constant Pool but when it is created using new operator then it is stored in heap memory.

Note : There will be no two objects in the string constant pool having the same content.

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

But when we create string using new operator then new object is created whether content is same or not.



Let's try to understand using simple program.

```
public static void main(String[] args) {
            String s1=new String("java");
            String s2=new String("java");
            System.out.println(s1==s2);
            String s3="java";
            String s4="java";
            System.out.println(s3==s4);


        }
```

Output

False

True

For s1 and s2 two objects are be created in heap memory.

For s3 and s4 since content is same so both is referring to same object is String Constant Pool.

Note : == operator returns true if physical address of both the objects

Summary : there can not be two string objects with same content in the string constant pool. But, there can be two string objects with the same content in the heap memory.

String Comparison

Two methods and == operator used for this purpose

== Operator

The operator compares physical memory address of two objects. If same returns true.

```
            String s1="Java";
            String s2="Java";
            System.out.println("s1==s2 : "+(s1==s2));
            String s3=new String("Java");
            String s4=new String("Java");
            System.out.println("s3==s4 : "+(s3==s4));

      Output :

      s1==s2 : true

      s3==s4 : false
```

Equals method()

This is method of Object class. It is overridden in String class. if not overrided, will perform same comparison as "==" operator does i.e comparing the objects on their physical address.

It compares content of the object returns true if found same.

```
            String s1="Java";
            String s2="Java";
            System.out.println("s1.equals(s2)-"+s1.equals(s2));
            String s3=new String("Java");
            String s4=new String("Java");
            System.out.println("s3.equals(s4)-"+s3.equals(s4));
            String s3=new String("Java");
            String s4=new String("java");
            System.out.println("s3.equals(s4)-"+s3.equals(s4));
            output-
            s1.equals(s2)-true
            s3.equals(s4)-true

            s3.equals(s4)-false
```

hashCode() method

It returns hash code value of an object in integer form. Returns true if two strings have same hash code value. It is overridden in String class.

```
            String s1="Java";
            String s2=new String("Java");
            System.out.println("s1.hashCode()==s2.hashCode():
"+(s1.hashCode()==s2.hashCode()));
            String s3 = "0-41L";
            String s4 = "0-42-";
            System.out.println("s3.hashCode()==s4.hashCode():
"+(s3.hashCode()==s4.hashCode()));
            String s5="Java";
            String s6=new String("java");
            System.out.println("s5.hashCode()==s6.hashCode():
"+(s5.hashCode()==s6.hashCode()));

             Output
             -------------------
             s1.hashCode()==s2.hashCode(): true
             s3.hashCode()==s4.hashCode(): true
             s5.hashCode()==s6.hashCode(): false
```

Internal Implementation of hashCode() method

```
public int hashCode() {
  int h = hash;
  if (h == 0) {
    int off = offset;
    char val[] = value;
    int len = count;

      for (int i = 0; i < len; i++) {
        h = 31*h + val[off++];
      }
      hash = h;
    }
    return h;
```

The hash code for a String object is computed as –

s[0]*31^(n - 1) + s[1]*31^(n - 2) + ... + s[n - 1]

Using int arithmetic, where s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

So different Strings have same hash code value.

String Class

String class is a final class. So it’s objects are immutable i.e threadsafe.
Please Note : A final class can extend any non -final class.
But a class cannot extend any final class.

```
package com.main;
class B
{
}

final class A extends B
{

}

public class Test  {

      public static void main(String[] args) {
```

Class A is final but it can extend to class B.
But Test class cannot extend class A as it is final.

Constructors of String Class

```
/*Please not below syntax creates an empty String not null*/
             String str1=new String();
             System.out.println(str1.length());//0
             System.out.println(str1.isEmpty());//true
             /*Below constructor takes char array to create a String*/
             char[] charArray = {'J', 'A', 'V', 'A'};
             String str2=new String(charArray);
             System.out.println(str2);
             /*Below constructor accepts String as a argument*/
             String str3=new String("Java");
             /*Below constructor accepts a StringBuffer*/
             StringBuffer buffer=new StringBuffer("Java");
             String str4=new String(buffer);
             /*Below constructor accepts a StringBuilder*/
      StringBuilder builder=new StringBuilder("JAva");
      String str5=new String(builder);
```

Important String Methods

charAt(int n) method

It returns character at specified position n.
String s = "Wipro Technologies Hyderabad";
System.out.println(s.charAt(0));//Output is W
N must be in between 0 to s.length()-1 otherwise StringIndexOutOfBoundsException will be thrown.
getChar() method

This method copies the set of characters from the string into specified character array

public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

This method copies characters of a string object starting from **‘srcBegin’** to **‘srcEnd’** into character array **‘dst’** at the index **‘dstBegin’**. This method will also throw **StringIndexOutOfBoundsException** if ‘srcBegin’ or ‘srcEnd’ are not between 0 and length() – 1 or if characters extracted does not fit into destination array.

Please if size of dst array is less than number of characters copied then ArrayIndexOutOfBoundException is thrown.

```
String s = "Wipro Technologies Hyderabad";
               //Defining destination char array
             char[] dst = new char[20];
             //Copying the set of characters from s into dst.
             s.getChars(5, 18, dst, 2);

        for (char c : dst)
        {
            System.out.print(c);       //Output : --Technologies—
      }
```

toCharArray()

This metho converts String to Array.

```
String s = "Wipro Technologies Hyderabad";
               char[] stringArray=s.toCharArray();
```

public String substring(int beginIndex) –> This form returns sub string starting from **‘beginIndex’** to the end of the specified string.

public String substring(int beginIndex, int endIndex) –> This form returns sub string starting from **‘beginIndex’** to **‘endIndex’** of the specified string.

```
String str = "Wipro Technologies Hyderabad";
             System.out.println(str.substring(6));//output-Technologies Hyderabad
             System.out.println(str.substring(6, 18));//-Technologies
```

More String methods- replace(), replaceALl()

Some more Important methods

```
String str = "Wipro Technologies Hyderabad";
             String[] splitedString=str.split("\\s");
             System.out.println(splitedString[0]+"--"+splitedString[1]+"--
"+splitedString[2]);
             //Output-Wipro--Technologies--Hyderabad
             System.out.println(str.indexOf('o'));//output-4
             System.out.println(str.startsWith("Wipro"));//true
             System.out.println(str.endsWith("baad"));//false
             System.out.println(str.contains(" Hyd"));//true
             System.out.println(str.replace("Hyderabad", "Bengaluru"));
             System.out.println(str.replace("Hydaa", "Chennai"));//No exception, Just replace
will not work
             System.out.println(str);//String is immutable concept applies here
                System.out.println(str.replaceAll("[AEIOUaeiou]", "-"));//All vowel will be
                                           replaced
```

Output-

```
Wipro--Technologies--Hyderabad
4
true
false
true
Wipro Technologies Bengaluru
Wipro Technologies Hyderabad
Wipro Technologies Hyderabad
W-pr- T-chn-l-g--s Hyd-r-b-d
```

Converting Given Type to String Type

static String valueOf(int value)—It is static overloaded method of String class

```
Test t=new Test();
             int aInt=10;
             String aIntString=String.valueOf(aInt);
             System.out.println(aIntString instanceof String);//true
             double aDouble=50;
             String aDoubleString=String.valueOf(aDouble);
             System.out.println(aDoubleString instanceof String);//true
             boolean aBoolean=true;
             String aBooleanString=String.valueOf(aBoolean);
             System.out.println(aBooleanString instanceof String);//true
             String tObject=String.valueOf(t);
             System.out.println(tObject);//prints object com.main.Test@4e7a15b
             System.out.println(tObject instanceof String);//true
```

Example to prove that String is immutable

Immutability is fundamental property of String. In Java once String object is created

You cannot modify the content of the String. If you try a new String object will be created with modified content.
Let’s create two String objects using String Literal. String constant pool cannot contain duplicate Strings if so it will point to same object.

```
String str1="Java";
String str2="Java";
System.out.println(str1==str2);//true as both pointing to same object in String
constant pool
```

Let’s modify the content of String str1. Now str1 contains JavaJ2ee as one String
str1=str1+"J2ee";
Let’s compare str1 and str2 again.
System.out.println(str1==str2);//false
It is returning false as str2 is pointing to a new Object after modification.
Now Let’s create two Strings using new operator.
So these will be stored in Heap Memory area which can contain duplicate Strings So can we say Strings created using new operator is mutable?

```
String str=new String("Java");
             System.out.println(str);
             str.concat(" J2ee");//Now in heap a String Java J2ee is created.
             System.out.println(str);//output is Java only
             /*Reason is new object is created but there is no
              * reference to hold it. So content of str cannot be changed so string is
immutable */
             /*Now point same reference to newly created object*/
             str=str.concat(" J2ee");
                 System.out.println(str);//Java J2ee as now reference is pointing to newly
                                         created object
```

Conclusion : 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

Why we need String Constant pool?

Suppose we need thousands of String with same content. Just imagine if we create all those Strings in Heap Memory how much memory it will consume and lowers the application performance as it allows duplicate Strings to be stored.
Instead of that we can create only one String in String Constant Pool and use it. This will reduce memory consumption as well as application.
This is the reason why we need String Constant Pool.
String Intern in Java

Interning in String is process of creating exact copy Heap Memory String object to String Constant Pool. Using intern() Method we achieve this. String interning is used to save memory and for fast comparing.

```
String str1=new String("Java");
             /*str1 is in heap memory. we are creating exact
              * copy of str1 in reference str2 in String Constant Pool*/
             String str2=str1.intern();
             System.out.println(str1==str2);//False
             String str3="Java";
             System.out.println(str2==str3);//True
             //String created using literals are automatically interned
             String str4="J2ee";
             String str5=str4.intern();
             System.out.println(str4==str5);//True--Automatically interned because
String Constant pool cannot contain duplicate
```

After interning it takes less time to compare because it just checks reference while equals() method compares character by character wise.

```
String str1=new String("Java");
             String str2=new String("Java");
             Long startTime1=System.currentTimeMillis();
             System.out.println(str1.equals(str2));
             Long endTime1=System.currentTimeMillis();
             System.out.println("Time to compare without intering in ms: "+(endTime1-
startTime1));
             String str3=str1.intern();
             String str4=str2.intern();
             Long startTime2=System.currentTimeMillis();
             System.out.println(str3==str4);
             Long endTime2=System.currentTimeMillis();
             System.out.println("Time to Compare after intering in ms : "+(endTime2-
startTime2));
```

Output-

```
true
Time to compare without intering in ms : 1
true
Time to Compare after intering in ms : 0
```

StringBuffer Class

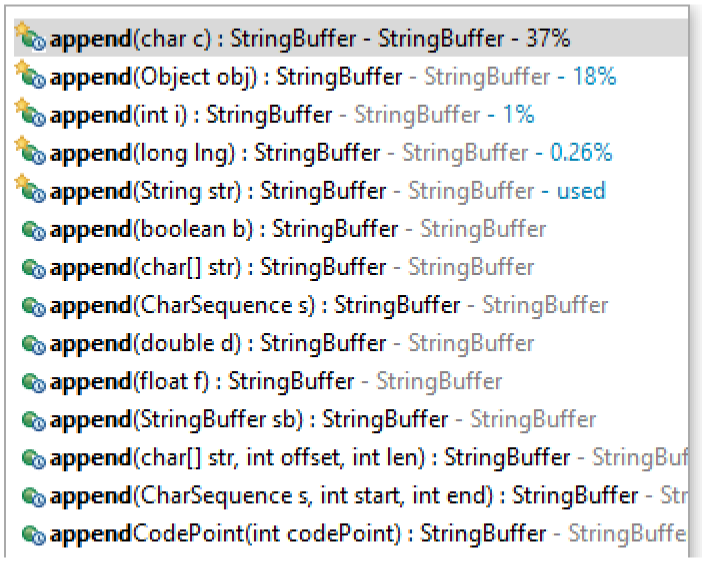
Both the classes are used to create mutable Strings. Once they are created, they cannot be modified. If you try to modify them, a new string object will be created with modified content. So modifying a string frequently causes memory issue. So we opt for StringBuilder or StringBuffer where strings needs frequent modifications in their content.

Strings using above classe can be created using new operator only and objects are stored in heap memory area only. Both the class have some special methods to achieve mutability.

Methods are- append(), replace(), delete(), reverse()

public synchronized StringBuffer append(String s)- Is used to append at the end of the string. It is an overloaded method.

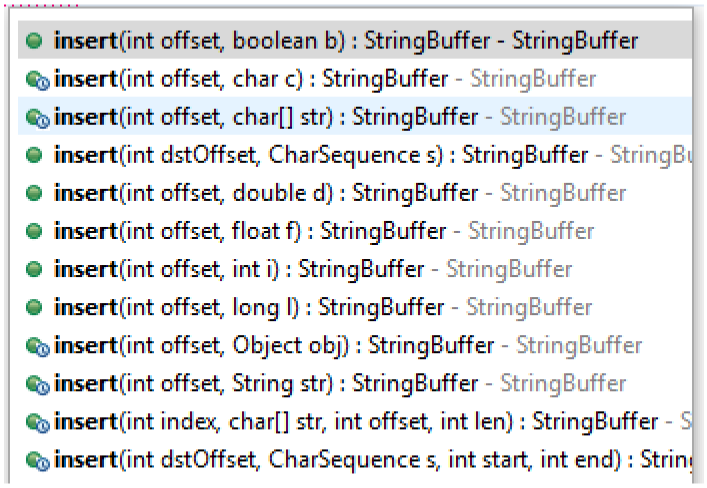
```
StringBuffer str= new StringBuffer("Java");
             str.append(" J2ee");
             System.out.println("After appending :"+str);//Java J2ee
```



```
append(char c) : StringBuffer - StringBuffer - 37%
append(Object obj) : StringBuffer - StringBuffer - 18%
append(int i) : StringBuffer - StringBuffer - 1%
append(long lng) : StringBuffer - StringBuffer - 0.26%
append(String str) : StringBuffer - StringBuffer - used
append(boolean b) : StringBuffer - StringBuffer
append(char[] str) : StringBuffer - StringBuffer
append(CharSequence s) : StringBuffer - StringBuffer
append(double d) : StringBuffer - StringBuffer
append(float f) : StringBuffer - StringBuffer
append(StringBuffer sb) : StringBuffer - StringBuffer
append(char[] str, int offset, int len) : StringBuffer - StringBuf
append(CharSequence s, int start, int end) : StringBuffer - Str
appendCodePoint(int codePoint) : StringBuffer - StringBuffe
```

public synchronized StringBuffer insert(int offset, String s)- is used to insert the specified string with this string at the specified position. It is also an overloaded method.

```
StringBuffer str= new StringBuffer("I am using version of Java");
             str.insert(10, " ");
             str.insert(11, 1.8);
             System.out.println(str);
             /*output- I am using 1.8 version of Java*/
```



```
insert(int offset, boolean b) : StringBuffer - StringBuffer
insert(int offset, char c) : StringBuffer - StringBuffer
insert(int offset, char[] str) : StringBuffer - StringBuffer
insert(int dstOffset, CharSequence s) : StringBuffer - StringBu
insert(int offset, double d) : StringBuffer - StringBuffer
insert(int offset, float f) : StringBuffer - StringBuffer
insert(int offset, int i) : StringBuffer - StringBuffer
insert(int offset, long l) : StringBuffer - StringBuffer
insert(int offset, Object obj) : StringBuffer - StringBuffer
insert(int offset, String str) : StringBuffer - StringBuffer
insert(int index, char[] str, int offset, int len) : StringBuffer - S
insert(int dstOffset, CharSequence s, int start, int end) : Strin
```

public synchronized StringBuffer replace(int startIndex, int endIndex, String str)- is used to replace the string from specified startIndex and endIndex. It is not Overloaded.

```
StringBuffer str= new StringBuffer("Java JSP");
             System.out.println(str.charAt(5));//J
             System.out.println(str.charAt(7));//P
             str.replace(5,8, "J2ee");
             System.out.println(str);//Java J2ee
```

public synchronized StringBuffer delete(int startIndex, int endIndex)- is used to delete the string from specified startIndex and endIndex.

```
StringBuffer str= new StringBuffer("Java JSP");
             str.delete(4,8);
             System.out.println(str);//Java
```

public synchronized StringBuffer reverse() -is used to reverse the string.

```
StringBuffer str= new StringBuffer("PSJ avaJ");
             str.reverse();
             System.out.println(str);//Java JSP
```

Note : capacity() and ensureCapacity() is only present in StringBufffer and StringBuilder class.
public int capacity()-is used to return the current capacity.
public void ensureCapacity(int minimumCapacity) -is used to ensure the capacity at least equal to the given minimum.
Common Methods Among three String classes- substring() , charAt(), length(), replace(), replaceAll()

StringBuilder Class

Both the classes are used to create mutable Strings. Once they are created, they cannot be modified. If you try to modify them, a new string object will be created with modified content. So modifying a string frequently causes memory issue. So we opt for StringBuilder or StringBuffer where strings needs frequent modifications in their content.

The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized

Strings using above classe can be created using new operator only and objects are stored in heap memory area only. Both the class have some special methods to achieve mutability.

Methods are- append(), replace(), delete(), reverse()

All these 4 methods in StringBuider class is non-synchronized.

1. Please Note : Unlike in C and C++, Strings in java are not terminated with **null** character. Strings are treated as objects in java.
2. String is not any keyword in Java. It's a class Name;
3. Java provides 2 more methods to compare the strings rather than equals() and equalsIgnoreCase().
   - compareTo()—compares string lexicographically .

- compareToIgnoreCase()-This method returns a negative integer, zero, or a positive integer as the specified String is greater than, equal to, or less than this String, ignoring case considerations.

```
String str1="Java";
    String str2="JAva";
    System.out.println(str1.compareToIgnoreCase(str2));//0
    String str3="Java J2ee";
    System.out.println(str1.compareToIgnoreCase(str3));//-5
    String str4="Ja";
    System.out.println(str1.compareToIgnoreCase(str4));//2
```

String vs StringBuffer vs StringBuilder

Because of thread safety property of String and StringBuffer classes, they reduces the performance of multithreaded applications.

Parameter	**String**	**StringBuffer**	**StringBuilder**
Immutability	Mutable	Immutable	Immutable
Object Creation	New or literals	Only new	Only new
Thread Safety	Yes	Yes	No
Performance	Slower	Slow	Fast
Concatenation	Slowest	Fast	Fastest

Note :

equals() and hashCode() Methods :
In StringBuffer and StringBuilder classes, equals() and hashCode methods are not overrided. Where as in String class they are overrided.
toString() Method :
toString() method is overrided in all three classes. You can also convert StringBuffer and StringBuilder objects to String type by calling toString() method on them.

Performance Test : String vs StrinBuffer vs StringBuilder

```
long startTime = System.currentTimeMillis();
        String str = new String("Java");
        for (int i=0; i<10000; i++){
            str.concat("J2ee");
        }
        System.out.println("Time taken by String: " + (System.currentTimeMillis() -
startTime) + "ms");

             startTime = System.currentTimeMillis();
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++){
            sb.append("J2ee");
        }
        System.out.println("Time taken by StringBuffer: " +
(System.currentTimeMillis() - startTime) + "ms");
        startTime = System.currentTimeMillis();
        StringBuilder sb2 = new StringBuilder("Java");
        for (int i=0; i<10000; i++){
            sb2.append("J2ee");
        }
        System.out.println("Time taken by StringBuilder: " +
(System.currentTimeMillis() - startTime) + "ms");
             Output-
```

Time taken by String: 6ms

Time taken by StringBuffer: 2ms

Time taken by StringBuilder: 1ms

How To Count Occurrences Of Each Character In String In Java? How to find duplicates in the given String?

```
Map<Character, Integer> characterCountMap=new HashMap<>();
             String inputString="Bitter Butter";
             char [] strArray=inputString.toCharArray();
             for(char c : strArray)
             {
                    if(characterCountMap.containsKey(c))
                    {
                           characterCountMap.put(c, characterCountMap.get(c)+1);
                    }
                    else
                    {
                           characterCountMap.put(c, 1);
                    }
             }

             System.out.println(characterCountMap);
             Iterator<Entry<Character, Integer>>
itr=characterCountMap.entrySet().iterator();
             while(itr.hasNext())
             {
                    Map.Entry<Character, Integer> pair=(Map.Entry<Character,
Integer>)itr.next();
                    if(pair.getValue()>1)
                    {
                           System.out.println(pair.getKey()+" is present
"+pair.getValue()+" times");
                    }
                                                   }
```

Output- (Here 1 space also counted)

```
{ =1, B=2, r=2, t=4, e=2, u=1, i=1}
```

B is present 2 times
r is present 2 times
t is present 4 times
e is present 2 times

String to int conversion

```
String a="12";
int b=Integer.parseInt(a);//12
int c=Integer.valueOf(a);//12
```

Int to String conversion

```
int a=12;
String b=String.valueOf(a);
String c=Integer.toString(a);
```

Interview Questions

1. What is similarity between String and StringBuffer?
   both are ThreadSafe
2. How to replace all white spaces from a String?
   Using replaceAll() method

```
String inputString="Wipro Technologies Chennai";
             String replaced=inputString.replaceAll("\\s", "");
                    System.out.println(replaced);
```

3. Can we call String class methods using string literals?
   Yes, we can call String class methods using string literals. Here are some examples,

```
"abc".charAt(0)

"abc".compareTo("abc")

"abc".indexOf('c')
```

4. **do you have any idea why strings have been made immutable in java?**

   a) Immutable strings increase security. As they can't be modified once they are created, so we can use them to store sensitive data like username, password etc.

   b) Immutable strings are thread safe. So, we can use them in a multi threaded code without synchronization.
   c) String objects are used in class loading. If strings are mutable, it is possible that wrong class is being loaded as mutable objects are modifiable.