Java String

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

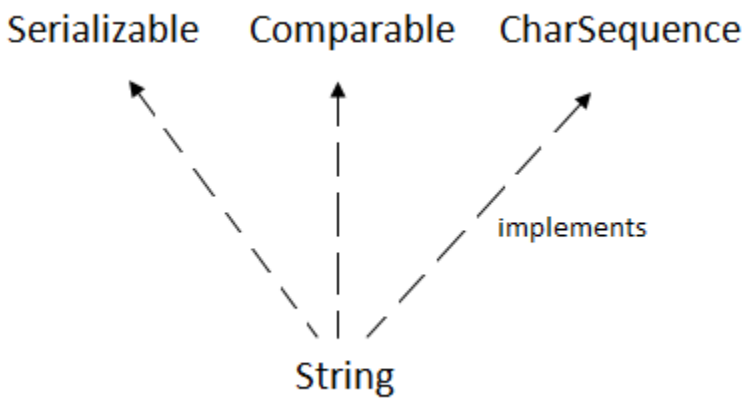
```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

```
1. String s="javatpoint";
```

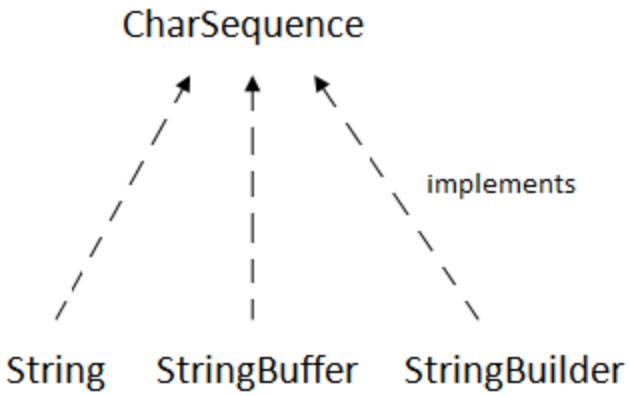
Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class
implements *Serializable*, *Comparable* and *CharSequence* interfaces.



CharSequence Interface

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

We will discuss about immutable string later. Let's first understand what is string in java and how to create the string object.

What is String in java

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

How to create String object?

There are two ways to create String object:

1. By string literal
2. By new keyword

---

1) String Literal

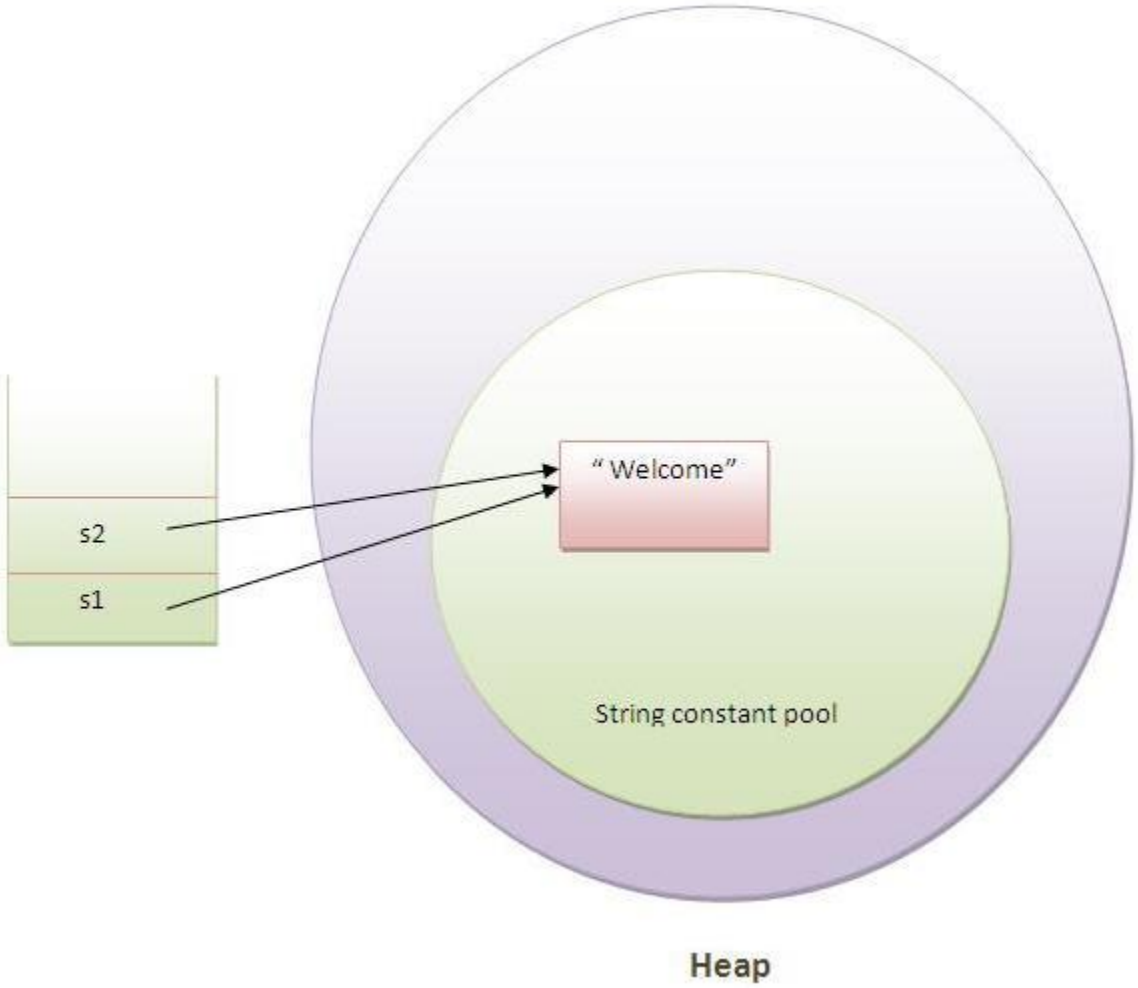
Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string

doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance



In the above example only one object will be created. 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 but will return the reference to the same instance.

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

Why java uses concept of string literal?

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

---

2) By new keyword

1. String s=**new** String("Welcome");//creates two objects and one reference variable

   In such 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).

Java String Example

```
1. public class StringExample{
2. public static void main(String args[]){
3. String s1="java";//creating string by java string literal
4. char ch[]={'s','t','r','i','n','g','s'};
5. String s2=new String(ch);//converting char array to string
6. String s3=new String("example");//creating java string by new keyword
7. System.out.println(s1);
8. System.out.println(s2);
9. System.out.println(s3);
10. }}
```

Test it Now

```
java
strings
example
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

Method	Description
char charAt(int index)	returns char value for the particular

	index
int length()	returns string length
String substring(int beginIndex)	returns substring for given begin index
String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string
boolean equals(Object another)	checks the equality of string with object
boolean isEmpty()	checks if string is empty
String concat(String str)	concatinates specified string
String replace(char old, char new)	replaces all occurrences of specified char value
String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
String[] split(String regex)	returns splitted string matching regex
String[] split(String regex, int limit)	returns splitted string matching regex and limit
int indexOf(int ch)	returns specified char value index
int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index

int indexOf(String substring)	returns specified substring index
int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
String toLowerCase()	returns string in lowercase.
String toUpperCase()	returns string in uppercase.
String trim()	removes beginning and ending spaces of this string.

Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

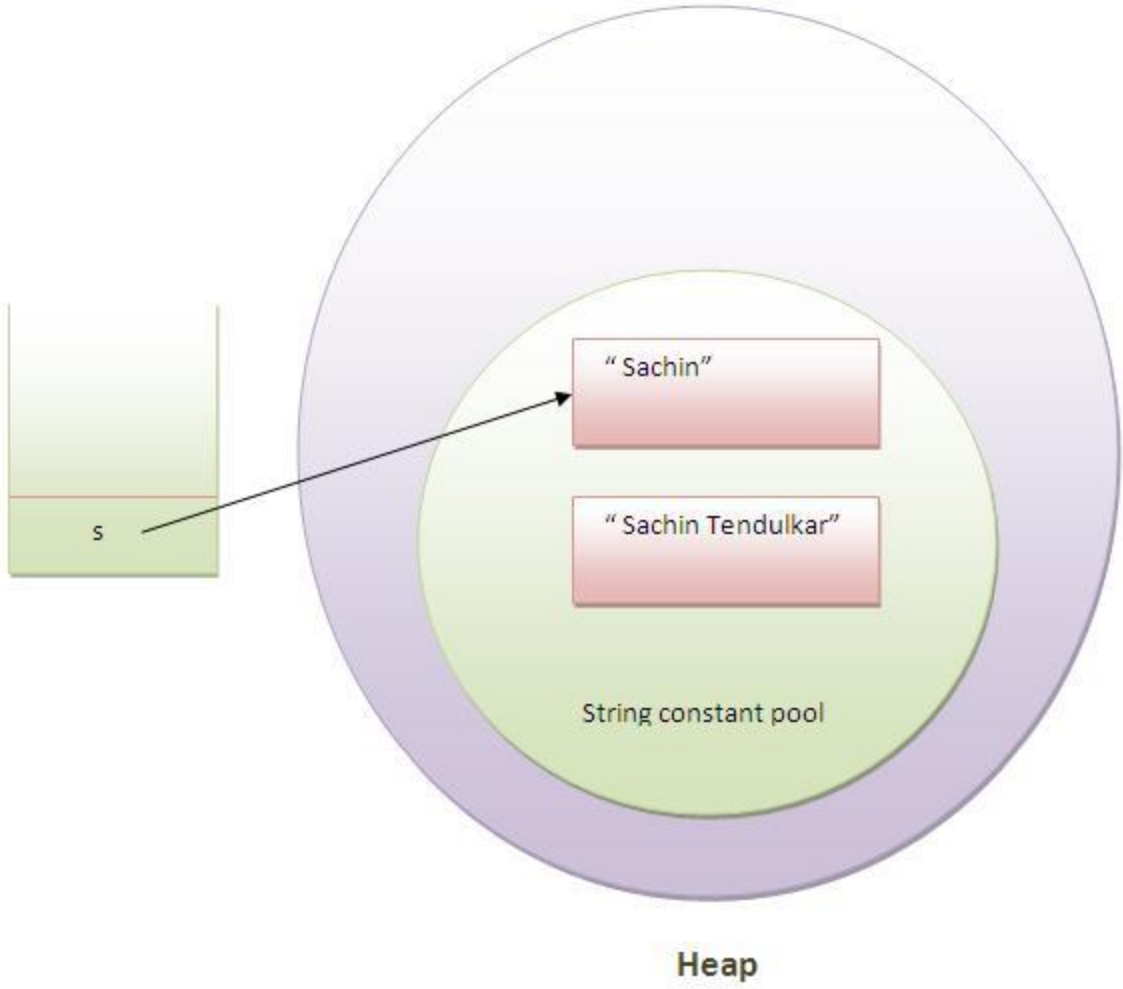
Let's try to understand the immutability concept by the example given below:

```
1. class Testimmutablestring{
2.  public static void main(String args[]){
3.    String s="Sachin";
4.    s.concat(" Tendulkar");//concat() method appends the string at the end
5.    System.out.println(s);//will print Sachin because strings are immutable objects
6.  }
7. }
```

Test it Now

```
Output:Sachin
```

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

```
1. class Testimmutablestring1{
2.  public static void main(String args[]){
3.    String s="Sachin";
4.    s=s.concat(" Tendulkar");
5.    System.out.println(s);
6.  }
7. }
```

Test it Now

```
Output:Sachin Tendulkar
```

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

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.

Java String compare

We can compare string in java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare string in java:

1. By equals() method
2. By = = operator
3. By compareTo() method

1) String compare by equals() method

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- **public boolean equals(Object another)** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.

```
1. class Teststringcomparison1{
2.  public static void main(String args[]){
3.    String s1="Sachin";
4.    String s2="Sachin";
5.    String s3=new String("Sachin");
6.    String s4="Saurav";
```

```
    System.out.println(s1.equals(s2));//true
    System.out.println(s1.equals(s3));//true
    System.out.println(s1.equals(s4));//false
 }
}
```

Test it Now

```
Output:true
       true
       false
```

```
class Teststringcomparison2{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="SACHIN";

   System.out.println(s1.equals(s2));//false
   System.out.println(s1.equalsIgnoreCase(s3));//true
 }
}
```

Test it Now

```
Output:false
       true
```

Click me for more about equals() method

---

2) String compare by == operator

The = = operator compares references not values.

```
class Teststringcomparison3{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   System.out.println(s1==s2);//true (because both refer to same instance)
   System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
 }
}
```

Test it Now

```
Output:true
```

```
        false
```

---

3) String compare by compareTo() method

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

- **s1 == s2** :0
- **s1 > s2** :positive value
- **s1 < s2** :negative value

```
1. class Teststringcomparison4{
2.  public static void main(String args[]){
3.    String s1="Sachin";
4.    String s2="Sachin";
5.    String s3="Ratan";
6.    System.out.println(s1.compareTo(s2));//0
7.    System.out.println(s1.compareTo(s3));//1(because s1>s3)
8.    System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
9.   }
10. }
```

Test it Now

```
Output:0
       1
       -1
```

String Concatenation in Java

In java, string concatenation forms a new string *that is* the combination of multiple strings. There are two ways to concat string in java:

1. By + (string concatenation) operator
2. By concat() method

1) String Concatenation by + (string concatenation) operator

Java string concatenation operator (+) is used to add strings. For Example:

```
1. class TestStringConcatenation1{
2.  public static void main(String args[]){
3.    String s="Sachin"+" Tendulkar";
4.    System.out.println(s);//Sachin Tendulkar
5.  }
6. }
```

Test it Now

```
Output:Sachin Tendulkar
```

The **Java compiler transforms** above code to this:

```
1. String s=(new StringBuilder()).append("Sachin").append(" Tendulkar).toString();
```

In java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also. For Example:

```
1. class TestStringConcatenation2{
2.  public static void main(String args[]){
3.    String s=50+30+"Sachin"+40+40;
4.    System.out.println(s);//80Sachin4040
5.  }
6. }
```

Test it Now

```
80Sachin4040
```

Note: After a string literal, all the + will be treated as string concatenation operator.

2) String Concatenation by concat() method

The String concat() method concatenates the specified string to the end of current string. Syntax:

```
1. public String concat(String another)
```

Let's see the example of String concat() method.

```
1. class TestStringConcatenation3{
```

```
2.   public static void main(String args[]){
3.     String s1="Sachin ";
4.     String s2="Tendulkar";
5.     String s3=s1.concat(s2);
6.     System.out.println(s3);//Sachin Tendulkar
7.    }
8.  }
```

Test it Now

```
Sachin Tendulkar
```

Java StringBuffer class

Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Important Constructors of StringBuffer class

1. **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

Important methods of StringBuffer class

1. **public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.
4. **public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.
5. **public synchronized StringBuffer reverse():** is used to reverse the string.
6. **public int capacity():** is used to return the current capacity.
7. **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.
8. **public char charAt(int index):** is used to return the character at the specified position.
9. **public int length():** is used to return the length of the string i.e. total number of characters.

10. **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
11. **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

```
1. class A{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.append("Java");//now original string is changed
5. System.out.println(sb);//prints Hello Java
6. }
7. }
```

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

```
1. class A{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }
```

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
1. class A{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }
```

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
1. class A{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }
7. }
```

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

```
1. class A{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }
7. }
```

6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
1. class A{
```

```
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. }
10. }
```

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
1. class A{
2. public static void main(String args[]){
3. StringBuffer sb=new StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. sb.ensureCapacity(10);//now no change
10. System.out.println(sb.capacity());//now 34
11. sb.ensureCapacity(50);//now (34*2)+2
12. System.out.println(sb.capacity());//now 70
13. }
14. }
```

Java StringBuilder class

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Important Constructors of StringBuilder class

1. **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
2. **StringBuilder(String str):** creates a string Builder with the specified string.
3. **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.

Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	is used to reverse the string.
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.

public char charAt(int index)	is used to return the character at the specified position.
public int length()	is used to return the length of the string i.e. total number of characters.
public String substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

Java StringBuilder Examples

Let's see the examples of different methods of StringBuilder class.

1) StringBuilder append() method

The StringBuilder append() method concatenates the given argument with this string.

```
1. class A{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello ");
4. sb.append("Java");//now original string is changed
5. System.out.println(sb);//prints Hello Java
6. }
7. }
```

2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

```
1. class A{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }
```

3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

```
1. class A{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }
```

4) StringBuilder delete() method

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

```
1. class A{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }
7. }
```

5) StringBuilder reverse() method

The reverse() method of StringBuilder class reverses the current string.

```
1. class A{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }
7. }
```

6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current

capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
1. class A{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. }
10. }
```

7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
1. class A{
2. public static void main(String args[]){
3. StringBuilder sb=new StringBuilder();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
9. sb.ensureCapacity(10);//now no change
10. System.out.println(sb.capacity());//now 34
11. sb.ensureCapacity(50);//now (34*2)+2
12. System.out.println(sb.capacity());//now 70
13. }
14. }
```

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.

Performance Test of String and StringBuffer

```
1. public class ConcatTest{
2.     public static String concatWithString()    {
3.         String t = "Java";
4.         for (int i=0; i<10000; i++){
5.             t = t + "Tpoint";
6.         }
7.         return t;
8.     }
9.     public static String concatWithStringBuffer(){
10.        StringBuffer sb = new StringBuffer("Java");
11.        for (int i=0; i<10000; i++){
12.            sb.append("Tpoint");
13.        }
14.        return sb.toString();
15.    }
16.    public static void main(String[] args){
17.        long startTime = System.currentTimeMillis();
18.        concatWithString();
19.        System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis
    ()-startTime)+"ms");
20.        startTime = System.currentTimeMillis();
21.        concatWithStringBuffer();
22.        System.out.println("Time taken by Concating with  StringBuffer: "+(System.currentTi
    meMillis()-startTime)+"ms");
```

```
    }
}
```

```
Time taken by Concating with String: 578ms
Time taken by Concating with  StringBuffer: 0ms
```

String and StringBuffer HashCode Test

As you can see in the program given below, String returns new hashcode value when you concat string but StringBuffer returns same.

```
public class InstanceTest{
    public static void main(String args[]){
        System.out.println("Hashcode test of String:");
        String str="java";
        System.out.println(str.hashCode());
        str=str+"tpoint";
        System.out.println(str.hashCode());

        System.out.println("Hashcode test of StringBuffer:");
        StringBuffer sb=new StringBuffer("java");
        System.out.println(sb.hashCode());
        sb.append("tpoint");
        System.out.println(sb.hashCode());
    }
}
```

```
Hashcode test of String:
3254818
229541438
Hashcode test of StringBuffer:
118352462
118352462
```

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	StringBuilder is *non-*

	safe. It means two threads can't call the methods of StringBuffer simultaneously.	*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 Example

```
1. public class BufferTest{
2.     public static void main(String[] args){
3.         StringBuffer buffer=new StringBuffer("hello");
4.         buffer.append("java");
5.         System.out.println(buffer);
6.     }
7. }
```

```
hellojava
```

StringBuilder Example

```
1. public class BuilderTest{
2.     public static void main(String[] args){
3.         StringBuilder builder=new StringBuilder("hello");
4.         builder.append("java");
5.         System.out.println(builder);
6.     }
7. }
```

```
hellojava
```

Performance Test of StringBuffer and StringBuilder

Let's see the code to check the performance of StringBuffer and StringBuilder classes.

```
1. public class ConcatTest{
2.     public static void main(String[] args){
3.         long startTime = System.currentTimeMillis();
4.         StringBuffer sb = new StringBuffer("Java");
```

```
5.        for (int i=0; i<10000; i++){
6.            sb.append("Tpoint");
7.        }
8.        System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis()
    - startTime) + "ms");
9.        startTime = System.currentTimeMillis();
10.       StringBuilder sb2 = new StringBuilder("Java");
11.       for (int i=0; i<10000; i++){
12.           sb2.append("Tpoint");
13.       }
14.       System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis(
    ) - startTime) + "ms");
15.   }
16. }
```

```
Time taken by StringBuffer: 16ms
Time taken by StringBuilder: 0ms
```