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:

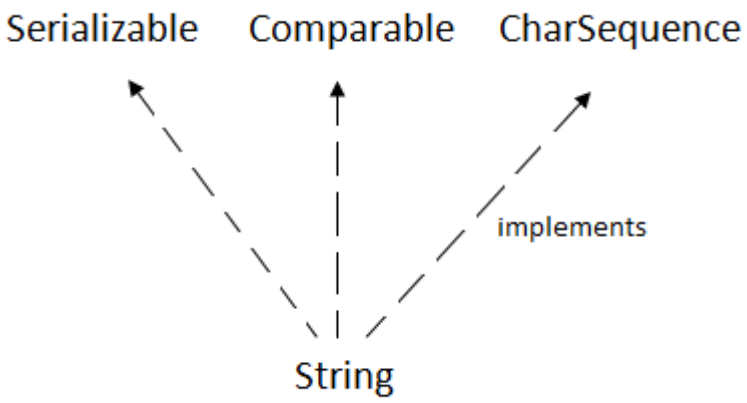
```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

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

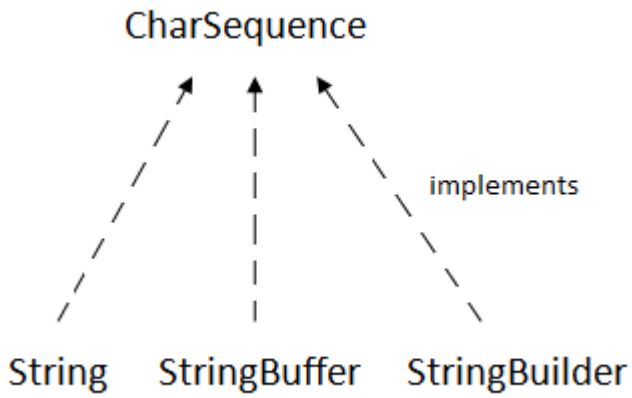
Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



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

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 a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

---

1) String Literal

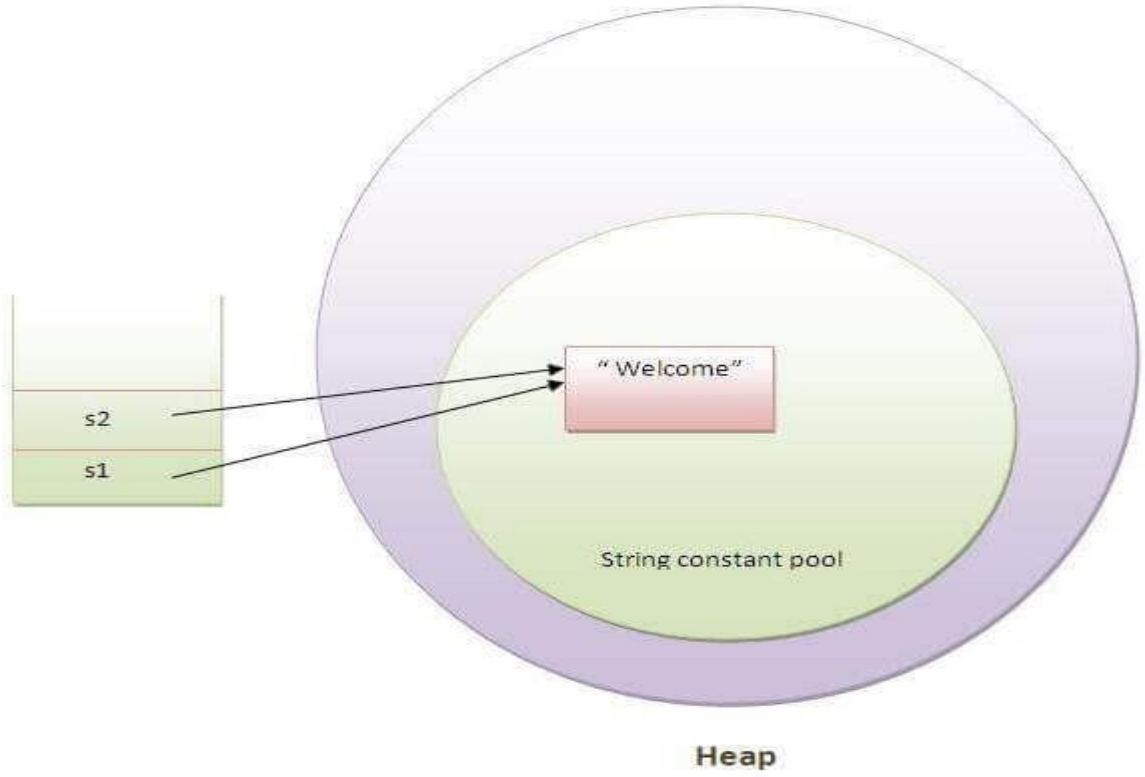
Java String literal is created by using double quotes. For Example:

```
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 the 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";//It  doesn't  create  a  new  instance
```

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



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, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

Why Java uses the concept of String literal?

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

2) By new keyword

```
1. String s=new String("Welcome");//creates two objects and one r
```

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 a 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.}}
```

java strings example

Java String class methods

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

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	static String format(String format, Object... args)	returns a formatted string.
4	static String format(Locale l, String format, Object... args)	returns formatted string with given locale.
5	String substring(int beginIndex)	returns substring for given begin index.
6	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
7	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
8	static String join(CharSequence delimiter, CharSequence... elements)	returns a joined string.
9	static String join(CharSequence	returns a joined string.

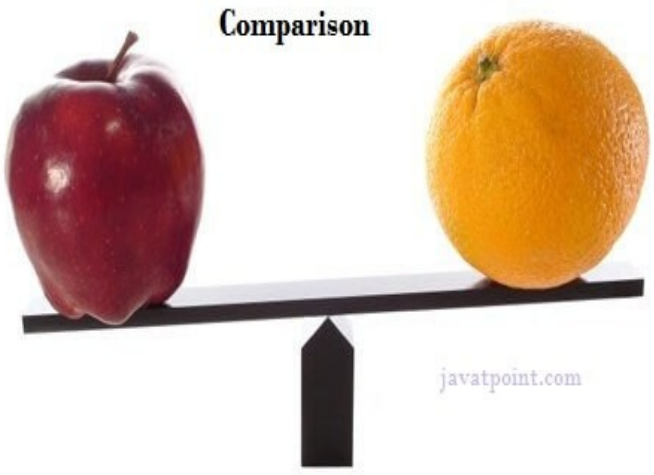
	delimiter, Iterable<? extends CharSequence> elements)	
10	boolean equals(Object another)	checks the equality of string with the given object.
11	boolean isEmpty()	checks if string is empty.
12	String concat(String str)	concatenates the specified string.
13	String replace(char old, char new)	replaces all occurrences of the specified char value.
14	String replace(CharSequence old, CharSequence new)	replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16	String[] split(String regex)	returns a split string matching regex.
17	String[] split(String regex, int limit)	returns a split string matching regex and limit.
18	String intern()	returns an interned string.
19	int indexOf(int ch)	returns the specified char value index.
20	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
21	int indexOf(String substring)	returns the specified substring

		index.
22	int indexOf(String substring, int fromIndex)	returns the specified substring index starting with given index.
23	String toLowerCase()	returns a string in lowercase.
24	String toLowerCase(Locale l)	returns a string in lowercase using specified locale.
25	String toUpperCase()	returns a string in uppercase.
26	String toUpperCase(Locale l)	returns a string in uppercase using specified locale.
27	String trim()	removes beginning and ending spaces of this string.
28	static String valueOf(int value)	converts given type into string. It is an overloaded method.

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";
7.   System.out.println(s1.equals(s2));//true
8.   System.out.println(s1.equals(s3));//true
```

```
9.    System.out.println(s1.equals(s4));//false
10. }
11.}
```

Test it Now

```
Output:true
           true
           false
```

```
1. class  Teststringcomparison2{
2.  public  static  void  main(String  args[]){
3.    String  s1="Sachin";
4.    String  s2="SACHIN";
5.
6.    System.out.println(s1.equals(s2));//false
7.    System.out.println(s1.equalsIgnoreCase(s2));//true
8.  }
9. }
```

Test it Now

Output:

```
false
true
```

strings

2) String compare by == operator

The = = operator compares references not values.example

```
class  Teststringcomparison3{
1. public  static  void  main(String  args[]){
2.  String  s1="Sachin";
3.  String  s2="Sachin";
4.  String  s3=new  String("Sachin");
5.  System.out.println(s1==s2);//true  (because  both  refer  to  same  instance)

6.  System.out.println(s1==s3);//false(because  s3  refers  to  instance  created
in  nonpool)
7. }
8.}
```

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. }
```

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)
```

```
class TestStringConcatenation3{
1. public static void main(String args[]){
2.  String s1="Sachin ";
3.  String s2="Tendulkar";
4.  String s3=s1.concat(s2);
5.  System.out.println(s3);//Sachin Tendulkar
6.  }
7.}
```

Substring in Java

A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.

Note: Index starts from 0.

You can get substring from the given string object by one of the two methods:

1. **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).

2. **public String substring(int startIndex, int endIndex):** This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

In case of string:

- **startIndex:** inclusive
- **endIndex:** exclusive

Let's understand the startIndex and endIndex by the code given below.

```
1. String  s="hello";
2. System.out.println(s.substring(0,2));//he
```

In the above substring, 0 points to h but 2 points to e (because end index is exclusive).

Example of java substring

```
1. public class TestSubstring{
2.  public static void main(String args[]){
3.   String s="SachinTendulkar";
4.   System.out.println(s.substring(6));//Tendulkar
5.   System.out.println(s.substring(0,6));//Sachin
6.  }
7. }
```

Test it Now

```
Tendulkar
Sachin
```

**********methods**********

Java String toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

```
1. String  s="Sachin";
2. System.out.println(s.toUpperCase());//SACHIN
```

```
3. System.out.println(s.toLowerCase());//sachin
4. System.out.println(s);//Sachin(no change in original)
```
Test it Now

```
SACHIN
sachin
Sachin
```

ToCHar ARRAY method

The java string toCharArray() method converts this string into character array. It returns a newly created character array, its length is similar to this string

```
1. public class StringToCharArrayExample{
2. public static void main(String args[]){
3. String s1="hello";
4. char[] ch=s1.toCharArray();
5. for(int i=0;i<ch.length;i++){
6. System.out.print(ch[i]);
7. }
8. }}
```

Output:

```
hello
```

Java String trim() method

The string trim() method eliminates white spaces before and after string.

```
1. String s="    Sachin    ";
2. System.out.println(s);//    Sachin
3. System.out.println(s.trim());//Sachin
```
Test it Now

```
    Sachin
Sachin
```

Java String startsWith() and endsWith() method

```
1. String s="Sachin";
2.  System.out.println(s.startsWith("Sa"));//true
3.  System.out.println(s.endsWith("n"));//true
```
Test it Now

```
true
true
```

Java String charAt() method

The string charAt() method returns a character at specified index.

```
String  s="Sachin";
System.out.println(s.charAt(0));//S
System.out.println(s.charAt(3));//h
```

Test it Now

```
S
h
```

Java String length() method

The string length() method returns length of the string.

```
String  s="Sachin";
System.out.println(s.length());//6
```

Test it Now

```
6
```

Java String intern() method

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

```
String  s=new  String("Sachin");
String  s2=s.intern();
System.out.println(s2);//Sachin
```

Test it Now

```
Sachin
```

Java String valueOf() method

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

```
int  a=10;
String  s=String.valueOf(a);
```

3. System.out.println(s+10);

Output:

```
1010
```

Java String substring() method example

```
1. public class SubstringExample{
2. public static void main(String args[]){
3. String s1="javatpoint";
4. System.out.println(s1.substring(2,4));//returns va
5. System.out.println(s1.substring(2));//returns vatpoint
6. }}
```

Test it Now

```
va
vatpoint
```

Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

```
1. public class SplitExample{
2. public static void main(String args[]){
3. String s1="java string split method by javatpoint";
4. String[] words=s1.split("\\s");//splits the string based on whitespace
5. //using java foreach loop to print elements of string array
6. for(String w:words){
7. System.out.println(w);
8. }
9. }}
```

Test it Now

```
java
string
split
method
by
javatpoint
```

Java String split() method with regex and length example

```
1. public class SplitExample2{
2. public static void main(String args[]){
3. String s1="welcome to split world";
4. System.out.println("returning words:");
5. for(String w:s1.split("\\s",0)){
6. System.out.println(w);
7. }
8. System.out.println("returning words:");
9. for(String w:s1.split("\\s",1)){
10.System.out.println(w);
11.}
12.System.out.println("returning words:");
13.for(String w:s1.split("\\s",2)){
14.System.out.println(w);
15.}
16.
17.}}
```

Test it Now

```
returning words:
welcome
to
split
world
returning words:
welcome to split world
returning words:
welcome
to split world
```

Java String split() method with regex and length example 2

Here, we are passing split limit as a second argument to this function. This limits the number of splitted strings.

```
1. public class SplitExample3 {
2.   public static void main(String[] args) {
3.     String str = "Javatpointttt";
4.     System.out.println("Returning words:");
5.     String[] arr = str.split("t", 0);
6.     for (String w : arr) {
7.       System.out.println(w);
8.     }
9.     System.out.println("Split array length: "+arr.length);
10.  }
11.}
```

```
Returning words:
Java
poin
Split array length: 2
```

Java String startsWith()

The **java string startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

```
public class StartsWithExample{
1.public static void main(String args[]){
2.String s1="java string split method by javatpoint";
3.System.out.println(s1.startsWith("ja"));
4.System.out.println(s1.startsWith("java string"));
5.}}
```

6.Output:

```
true
true
```

7.

Java String replace() method /replaceAll

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

```
String s1="Java is a programming language. Java is a platform. Java is a
n Island.";
```

```
1.String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
2.System.out.println(replaceString);
```

Output:

```
Kava is a programming language. Kava is a platform. Kava is an Island.
```

Java String replaceAll() example: replace character

Let's see an example to replace all the occurrences of **a single character**.

```
1. public class ReplaceAllExample1{
2. public static void main(String args[]){
3. String s1="javatpoint is a very good website";
4. String replaceString=s1.replaceAll("a","e");//replaces all occurrences of "a" to "e"
5. System.out.println(replaceString);
6. }}
```

Test it Now

```
jevetpoint is e very good website
```

Java String replaceAll() example: remove white spaces

Let's see an example to remove all the occurrences of **white spaces**.

```
1. public class ReplaceAllExample3{
2. public static void main(String args[]){
3. String s1="My name is Khan. My name is Bob. My name is Sonoo.";
4. String replaceString=s1.replaceAll("\\s","");
5. System.out.println(replaceString);
6. }}
```

Test it Now

```
MynameisKhan.MynameisBob.MynameisSonoo.
```

Java String lastIndexOf()

The **java string lastIndexOf()** method returns last index of the given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

Java String lastIndexOf() method example

```
1. public class LastIndexOfExample{
```

```
2. public static void main(String args[]){
3. String s1="this is index of example";//there are 2 's' characters in this s
   entence
4. int index1=s1.lastIndexOf('s');//returns last index of 's' char value
5. System.out.println(index1);//6
6. }}
```

Test it Now

Output:

Java String join()

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added

Returns

joined string with delimiter

Java String join() method example

```
1. public class StringJoinExample{
2. public static void main(String args[]){
3. String joinString1=String.join("-","welcome","to","javatpoint");
4. System.out.println(joinString1);
5. }}
```

Test it Now

```
welcome-to-javatpoint
```

Java String join() Method Example 2

We can use delimeter to format the string as we did in the below example to show date and time.

```
1. public class StringJoinExample2 {
2.   public static void main(String[] args) {
3.     String date = String.join("/","25","06","2018");
4.     System.out.print(date);
5.     String time = String.join(":", "12","10","10");
```

```
6.      System.out.println(" "+time);
7.    }
8. }
```

25/06/2018 12:10:10

Java String isEmpty()

The **java string isEmpty()** method checks if this string is empty or not. It returns *true*, if length of string is 0 otherwise false. In other words, true is returned if string is empty otherwise it returns false

Java String isEmpty() method example

```
1. public class IsEmptyExample{
2. public static void main(String args[]){
3. String s1="";
4. String s2="javatpoint";
5.
6. System.out.println(s1.isEmpty());
7. System.out.println(s2.isEmpty());
8. }}
```

Test it Now

true

false

Java String isEmpty() Method Example 2

```
1. public class IsEmptyExample2 {
2.   public static void main(String[] args) {
3.     String s1="";
4.     String s2="Javatpoint";
5.     // Either length is zero or isEmpty is true
6.     if(s1.length()==0 || s1.isEmpty())
7.       System.out.println("String s1 is empty");
8.     else System.out.println("s1");
9.     if(s2.length()==0 || s2.isEmpty())
10.      System.out.println("String s2 is empty");
11.    else System.out.println(s2);
```

```
12.   }
13.}
```

```
String s1 is empty
Javatpoint
```

Java String indexOf()

The **java string indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

Java String indexOf() method example

```
1. public class IndexOfExample{
2. public static void main(String args[]){
3. String s1="this is index of example";
4. //passing substring
5. int index1=s1.indexOf("is");//returns the index of is substring
6. int index2=s1.indexOf("index");//returns the index of index substring
7. System.out.println(index1+"   "+index2);//2  8
8.
9. //passing substring with from index
10.int index3=s1.indexOf("is",4);//returns the index of is substring after 4th
   index
11.System.out.println(index3);//5  i.e. the index of another is
12.
13.//passing char value
14.int index4=s1.indexOf('s');//returns the index of s char value
15.System.out.println(index4);//3
16.}}
```

Test it Now

```
2  8
5
3
```

Java String indexOf(String substring, int fromIndex) Method Example

This method takes substring and index as arguments and returns index of first character occured after the given fromIndex.

```
1. public class IndexOfExample3 {
2.   public static void main(String[] args) {
3.     String s1 = "This is indexOf method";
4.     // Passing substring and index
5.     int index = s1.indexOf("method", 10); //Returns the index of this sub string
6.     System.out.println("index of substring "+index);
7.     index = s1.indexOf("method", 20); // It returns -1 if substring does not found
8.     System.out.println("index of substring "+index);
9.   }
10.}
```

Test it Now

```
index of substring 16
index of substring -1
```

Java String indexOf(int char, int fromIndex) Method Example

This method takes char and index as arguments and returns index of first character occured after the given fromIndex.

```
1. public class IndexOfExample4 {
2.   public static void main(String[] args) {
3.     String s1 = "This is indexOf method";
4.     // Passing char and index from
5.     int index = s1.indexOf('e', 12); //Returns the index of this char
6.     System.out.println("index of char "+index);
7.   }
8. }
```

Test it Now

```
index of char 17
```

Java String getChars()

The **java string getChars()** method copies the content of this string into specified char array. There are 4 arguments passed in getChars() method. The signature of getChars() method is given below:

```
1. public class StringGetCharsExample{
2. public static void main(String args[]){
3. String str = new String("hello javatpoint how r u");
4.    char[] ch = new char[10];
5.    try{
6.      str.getChars(6, 16, ch, 0);
7.      System.out.println(ch);
8.    }catch(Exception ex){System.out.println(ex);}
9. }}
```

Test it Now

Output:

```
javatpoint
```

Java String getBytes()

The **java string getBytes()** method returns the byte array of the string. In other words, it returns sequence of bytes.

Java String getBytes() method example

```
1. public class StringGetBytesExample{
2. public static void main(String args[]){
3. String s1="ABCDEFG";
4. byte[] barr=s1.getBytes();
5. for(int i=0;i<barr.length;i++){
6. System.out.println(barr[i]);
7. }
8. }}
```

Test it Now

Output:

```
65
66
67
```

68

69

70

71

Java String getBytes() Method Example 2

This method returns a byte array that again can be passed to String constructor to get String.

```
1. public class StringGetBytesExample2 {
2.   public static void main(String[] args) {
3.     String s1 = "ABCDEFG";
4.     byte[] barr = s1.getBytes();
5.     for(int i=0;i<barr.length;i++){
6.       System.out.println(barr[i]);
7.     }
8.     // Getting string back
9.     String s2 = new String(barr);
10.    System.out.println(s2);
11.  }
12.}
```

Test it Now

Output:

```
65
66
67
68
69
70
71
ABCDEFG
```

Java String contains()

The **java string contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*

Java String contains() method example

```
1. class ContainsExample{
2. public static void main(String args[]){
```

```
3. String name="what do you know about me";
4. System.out.println(name.contains("do you know"));
5. System.out.println(name.contains("about"));
6. System.out.println(name.contains("hello"));
7. }}
```

Test it Now

```
true
true
false
```

Java String contains() Method Example 2

The contains() method searches case sensitive char sequence. If the argument is not case sensitive, it returns false. Let's see an example below.

```
public class ContainsExample2 {

1.   public static void main(String[] args) {
2.     String str = "Hello Javatpoint readers";
3.     boolean isContains = str.contains("Javatpoint");
4.     System.out.println(isContains);
5.     // Case Sensitive
6.     System.out.println(str.contains("javatpoint")); // false
7.   }
8. }
```

```
true
false
```

STRING Buffer

Java StringBuffer class

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

changed

Important Constructors of StringBuffer class

Constructor	Description
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.

Important methods of StringBuffer class

Modifier and Type	Method	Description
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.
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.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	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.

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 StringBufferExample{
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 StringBufferExample2{
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 StringBufferExample3{
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 StringBufferExample4{
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 StringBufferExample5{
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 StringBufferExample6{
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 StringBufferExample7{
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

Constructor	Description

StringBuilder()	creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str)	creates a string Builder with the specified string.
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 StringBuilderExample{
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 StringBuilderExample2{
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 StringBuilderExample3{
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  StringBuilderExample4{
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  StringBuilderExample5{
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  StringBuilderExample6{
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 StringBuilderExample7{
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()	StringBuffer class doesn't override the equals() method of Object class.

	method.	

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.

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;
3.
4. public Employee(String pancardNumber){
5. this.pancardNumber=pancardNumber;
6. }
7.
8. public String getPancardNumber(){
9. return pancardNumber;
10.}
11.
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.

Java toString() method

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

Advantage of Java toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

Understanding problem without toString() method

Let's see the simple code that prints reference.

```
1. class Student{
2.  int rollno;
3.  String name;
4.  String city;
5.
6.  Student(int rollno, String name, String city){
7.  this.rollno=rollno;
8.  this.name=name;
9.  this.city=city;
10. }
11.
12. public static void main(String args[]){
13.  Student s1=new Student(101,"Raj","lucknow");
14.  Student s2=new Student(102,"Vijay","ghaziabad");
15.
```

```
16.  System.out.println(s1);//compiler writes here s1.toString()
17.  System.out.println(s2);//compiler writes here s2.toString()
18. }
19.}
```

```
Output:Student@1fee6fc
          Student@1eed786
```

As you can see in the above example, printing s1 and s2 prints the hashcode values of the objects but I want to print the values of these objects. Since java compiler internally calls toString() method, overriding this method will return the specified values. Let's understand it with the example given below:

Example of Java toString() method

```
class Student{
1.  int rollno;
2.  String name;
3.  String city;
4.
5.  Student(int rollno, String name, String city){
6.  this.rollno=rollno;
7.  this.name=name;
8.  this.city=city;
9.  }
10.
11. public String toString(){//overriding the toString() method
12. return rollno+" "+name+" "+city;
13. }
14. public static void main(String args[]){
15.  Student s1=new Student(101,"Raj","lucknow");
16.  Student s2=new Student(102,"Vijay","ghaziabad");
17.
18.  System.out.println(s1);//compiler writes here s1.toString()
19.  System.out.println(s2);//compiler writes here s2.toString()
20. }
21.}
```

download this example of toString method

```
Output:101 Raj lucknow
```

102 Vijay ghaziabad

How to convert an Array to String in Java?

- Difficulty Level : **Medium**
- Last Updated : 30 Sep, 2019

Below are the various methods to convert an Array to String in Java:

1. **Arrays.toString() method**: Arrays.toString() method is used to return a string representation of the contents of the specified array. The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (a comma followed by a space). It returns "null" if the array is null.

```
// Java program to demonstrate
// working of Arrays.toString()

import java.io.*;
import java.util.*;

class GFG {
    public static void main(String[] args)
    {

        // Let us create different types of arrays and
        // print their contents using Arrays.toString()
        boolean[] boolArr
            = new boolean[] { true, true, false, true };
        char[] charArr
            = new char[] { 'g', 'e', 'e', 'k', 's' };
        double[] dblArr
            = new double[] { 1, 2, 3, 4 };
        int[] intArr
            = new int[] { 1, 2, 3, 4 };
        Object[] objArr
            = new Object[] { 1, 2, 3, 4 };

        System.out.println(
            "Boolean Array: "
            + Arrays.toString(boolArr));
        System.out.println(
            "Character Array: "
            + Arrays.toString(charArr));
        System.out.println(
```

```
            "Double Array: "
            + Arrays.toString(dblArr));
        System.out.println(
            "Integer Array: "
            + Arrays.toString(intArr));
        System.out.println(
            "Object Array: "
            + Arrays.toString(objArr));
    }
}
```

Output:

```
Boolean Array: [true, true, false, true]

Character Array: [g, e, e, k, s]

Double Array: [1.0, 2.0, 3.0, 4.0]

Integer Array: [1, 2, 3, 4]

Object Array: [1, 2, 3, 4]
```