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:

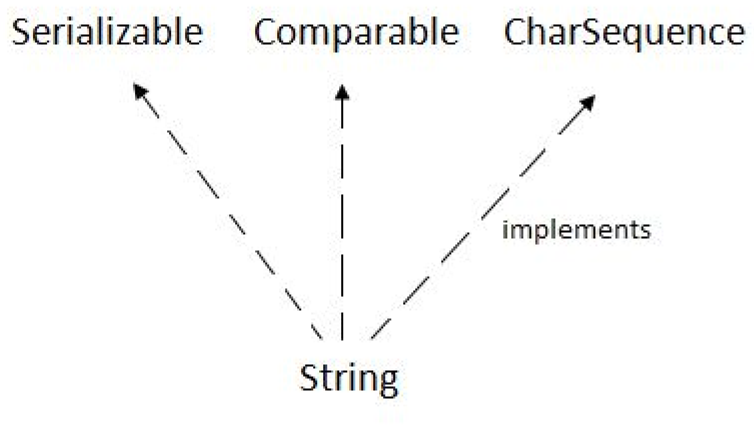
```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

is same as:

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

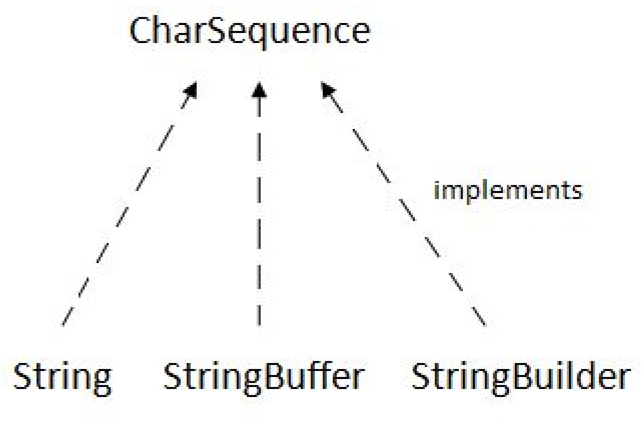
Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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.

We will discuss 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 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

---

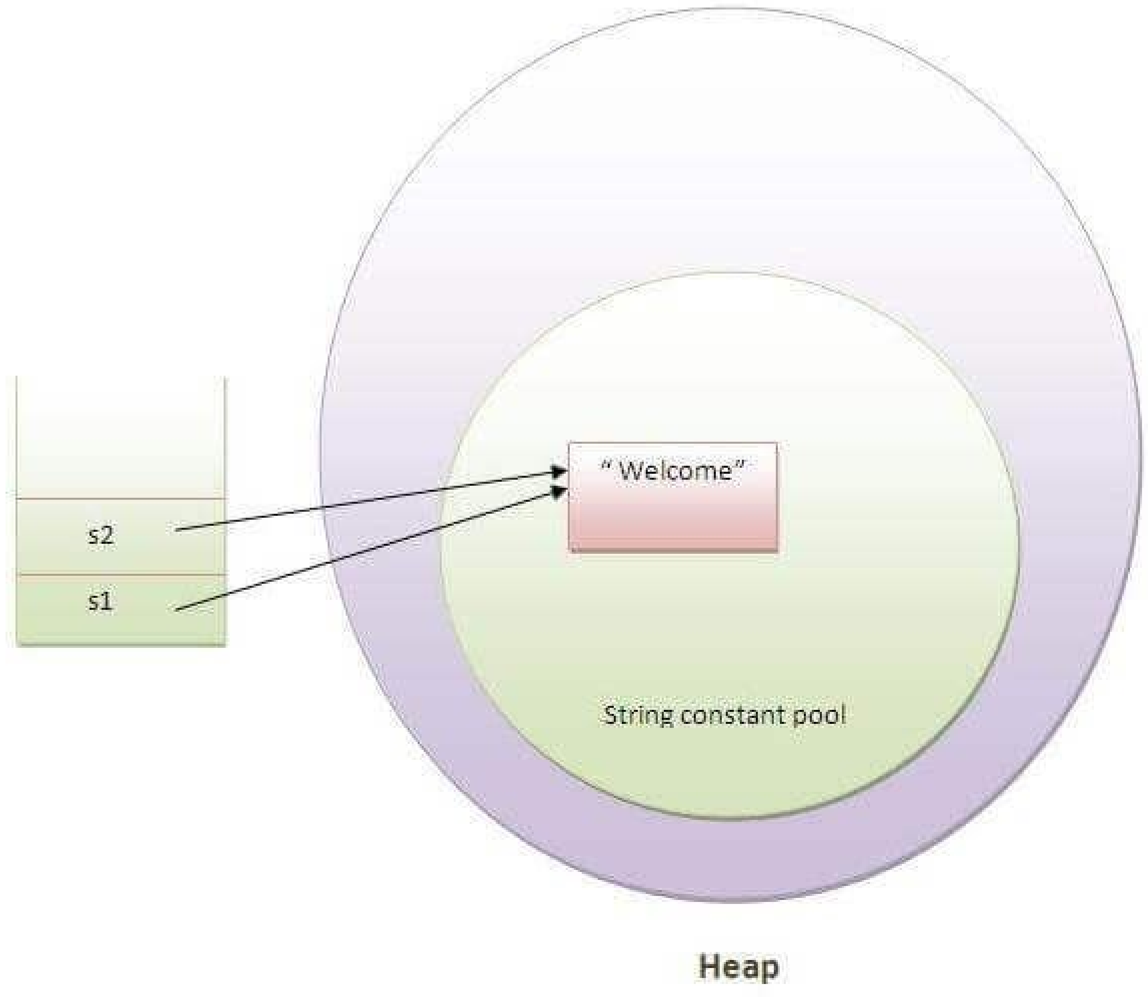
1) String Literal

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
```



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.

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

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

```java
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 a heap (non-pool).

Java String Example

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

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.

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 delimiter, Iterable<? extends CharSequence> elements)	returns a joined string.
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.

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.

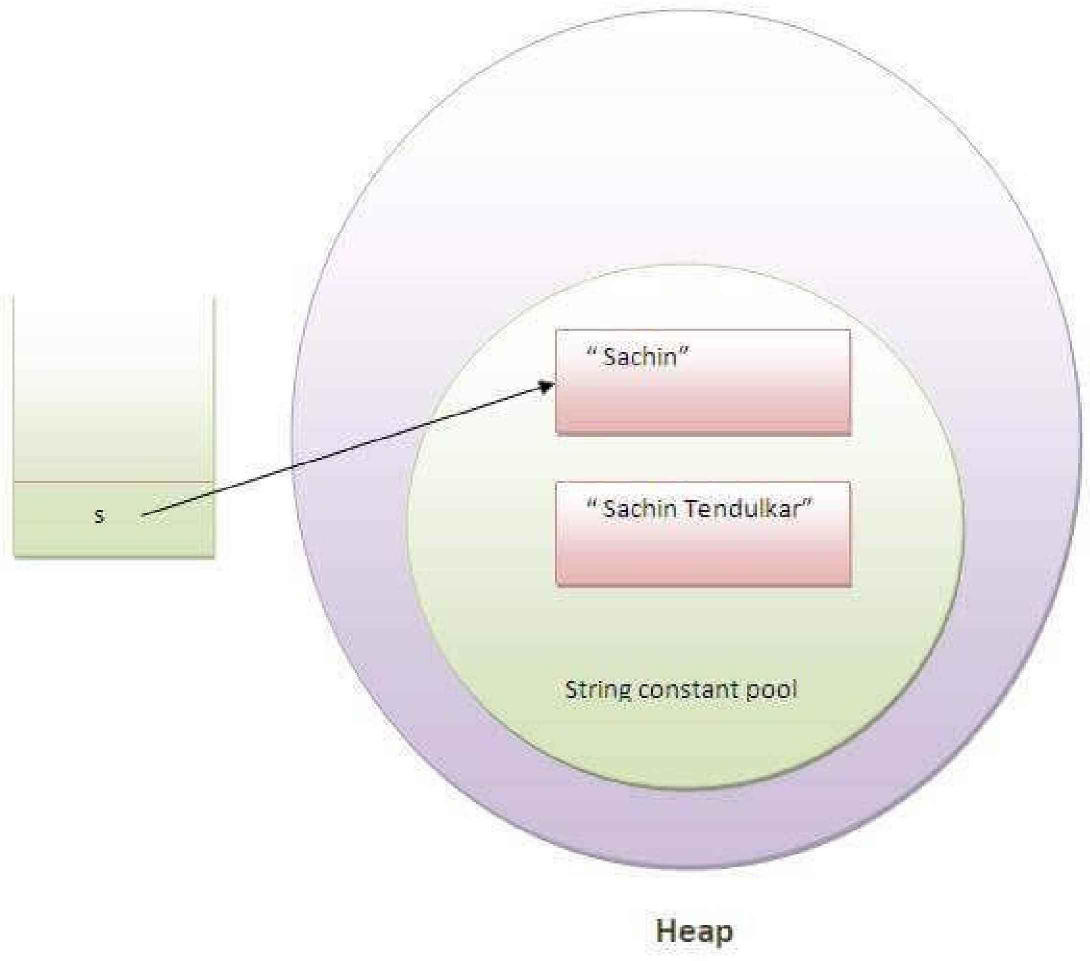
Let's try to understand the immutability concept by the example given below:

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

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:

```
class Testimmutablestring1{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s);
 }
}
```

Test it Now

Output:Sachin Tendulkar

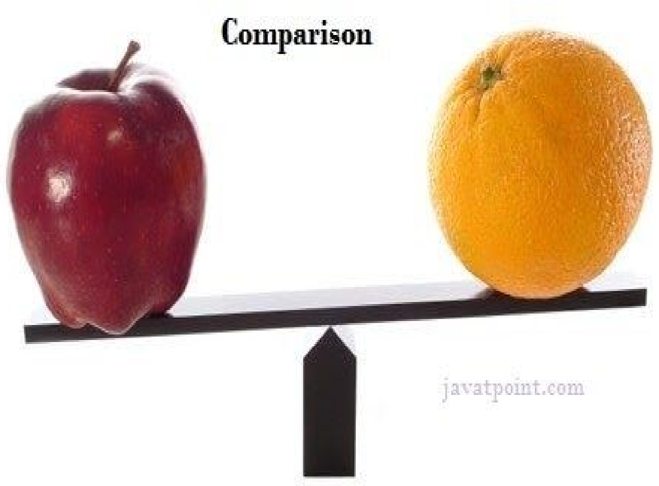
In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

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.

Next »« Prev

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.

```java
class Teststringcomparison1{
 public static void main(String args[]){
   String s1="Sachin";
   String s2="Sachin";
   String s3=new String("Sachin");
   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
```

```java
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(s2));//true
 }
}
```

Test it Now

Output:

```
false
true
```

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:

```java
class TestStringConcatenation1{
 public static void main(String args[]){
   String s="Sachin"+" Tendulkar";
   System.out.println(s);//Sachin Tendulkar
  }
 }
```

Test it Now

Output:Sachin Tendulkar

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

```
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:

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

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:

```
public String concat(String another)
```

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

```
class TestStringConcatenation3{
 public static void main(String args[]){
   String s1="Sachin ";
   String s2="Tendulkar";
   String s3=s1.concat(s2);
```

```
    System.out.println(s3);//Sachin Tendulkar
  }
}
```

Test it Now

```
Sachin Tendulkar
```

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.

```
String s="hello";
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

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

Test it Now

```
Tendulkar
Sachin
```

Java String class methods

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

Let's see the important methods of String class.

Java String toUpperCase() and toLowerCase() method

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

```java
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
```

Test it Now

```
SACHIN
sachin
```

Sachin

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.

1. String s="Sachin";
2. System.out.println(s.charAt(0));//S
3. 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);
System.out.println(s+10);
```

Output:

```
1010
```

Java String replace() method

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

```
1. String s1="Java is a programming language. Java is a platform. Java is an Island.";
2. String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
3. System.out.println(replaceString);
```

Output:

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

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.

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

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

StringBuffer		
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.

```java
class StringBufferExample{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

2) StringBuffer insert() method

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

```java
class StringBufferExample2{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

3) StringBuffer replace() method

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

```java
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

4) StringBuffer delete() method

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

```
class StringBufferExample4{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

5) StringBuffer reverse() method

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

```
class StringBufferExample5{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

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.

```java
class StringBufferExample6{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}
```

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.

```java
class StringBufferExample7{
public static void main(String args[]){
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

ava 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.

```java
class StringBuilderExample{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.append("Java");//now original string is changed
System.out.println(sb);//prints Hello Java
}
}
```

2) StringBuilder insert() method

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

```java
class StringBuilderExample2{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello ");
sb.insert(1,"Java");//now original string is changed
System.out.println(sb);//prints HJavaello
}
}
```

3) StringBuilder replace() method

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

```java
class StringBuilderExample3{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJavalo
}
}
```

4) StringBuilder delete() method

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

```java
class StringBuilderExample4{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.delete(1,3);
System.out.println(sb);//prints Hlo
}
}
```

5) StringBuilder reverse() method

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

```
class StringBuilderExample5{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Hello");
sb.reverse();
System.out.println(sb);//prints olleH
}
}
```

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.

```
class StringBuilderExample6{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
}
}
```

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.

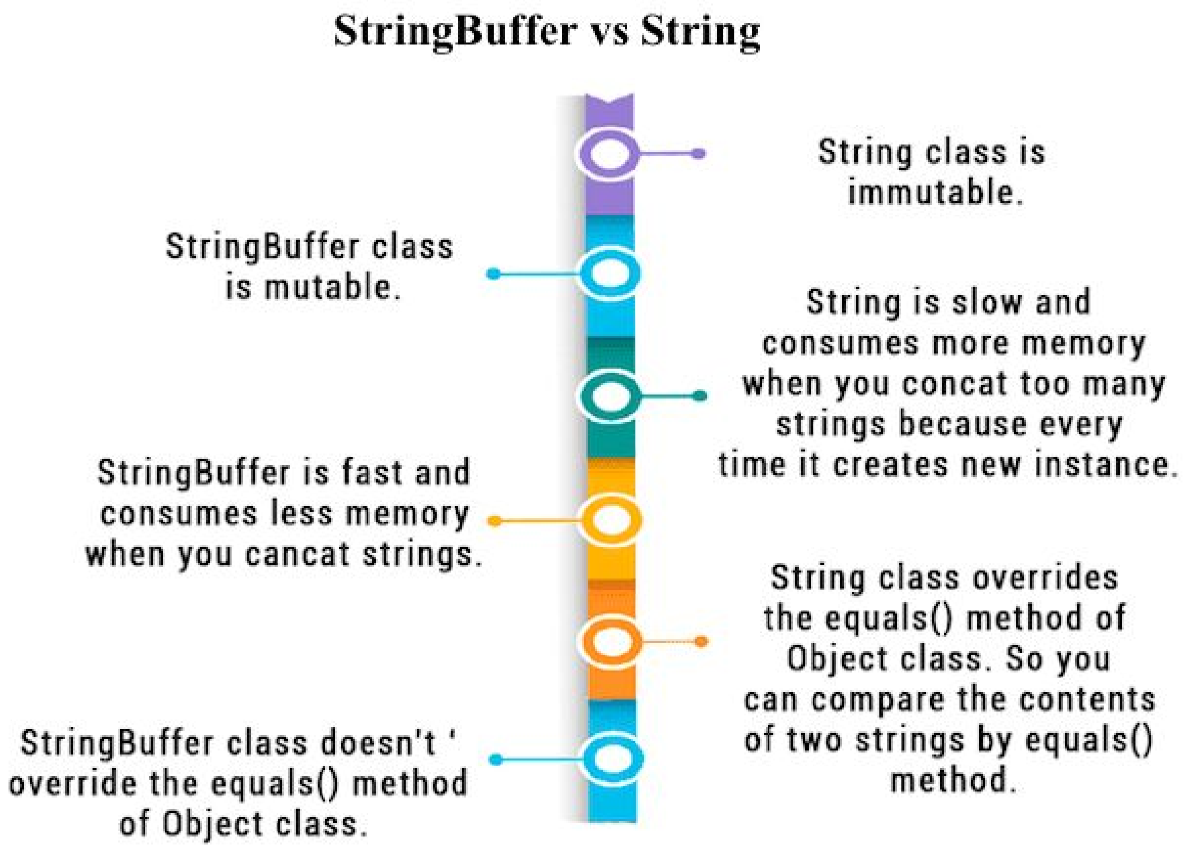
```
class StringBuilderExample7{
public static void main(String args[]){
StringBuilder sb=new StringBuilder();
System.out.println(sb.capacity());//default 16
sb.append("Hello");
System.out.println(sb.capacity());//now 16
sb.append("java is my favourite language");
System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
sb.ensureCapacity(10);//now no change
System.out.println(sb.capacity());//now 34
sb.ensureCapacity(50);//now (34*2)+2
System.out.println(sb.capacity());//now 70
}
}
```

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.


StringBuffer vs String
String class is immutable.
StringBuffer class is mutable.
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.
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

```
public class ConcatTest{
    public static String concatWithString()    {
        String t = "Java";
        for (int i=0; i<10000; i++){
            t = t + "Tpoint";
        }
        return t;
    }
    public static String concatWithStringBuffer(){
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++){
            sb.append("Tpoint");
        }
        return sb.toString();
    }
    public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        concatWithString();
        System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");
        startTime = System.currentTimeMillis();
        concatWithStringBuffer();
        System.out.println("Time taken by Concating with  StringBuffer: "+(System.currentTimeMillis()-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
```

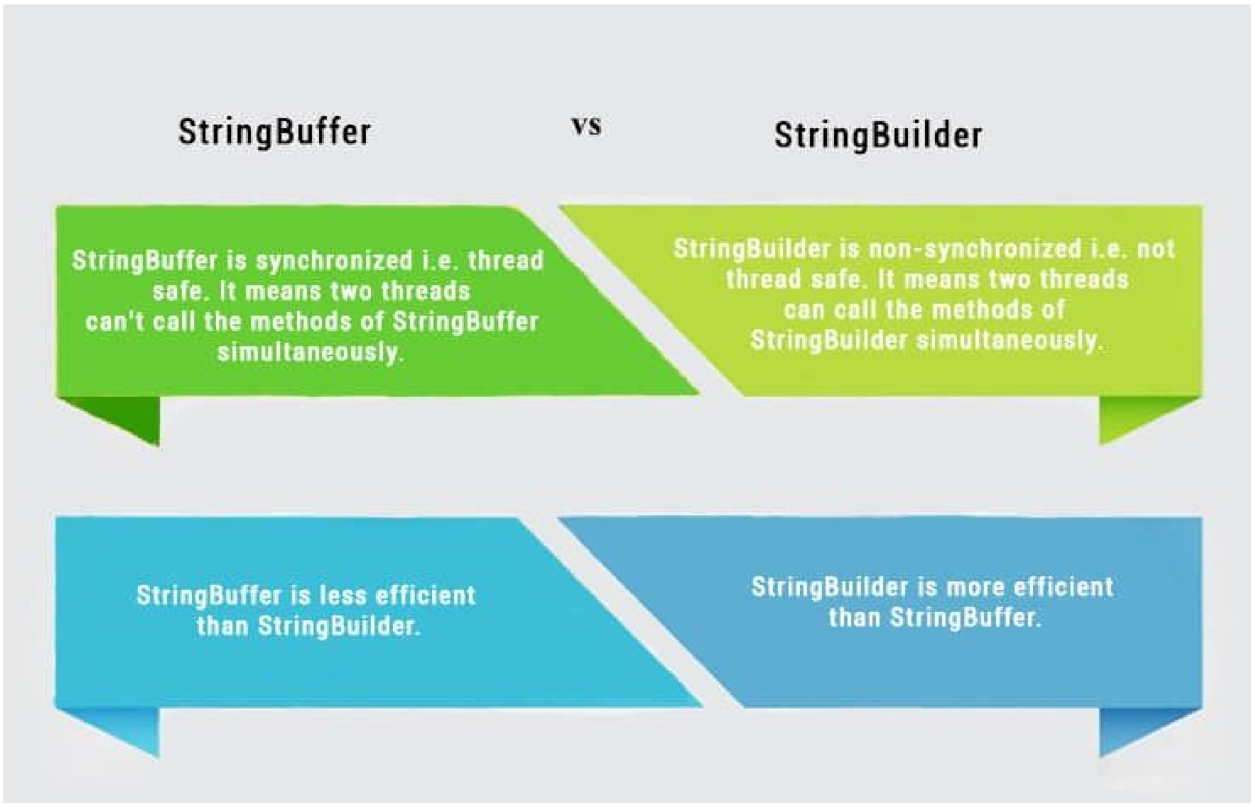
Difference between StringBuffer and StringBuilder

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.

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.



StringBuffer vs StringBuilder

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.

StringBuffer is less efficient than StringBuilder.

StringBuilder is more efficient than StringBuffer.

StringBuffer Example

```
//Java Program to demonstrate the use of StringBuffer class.
public class BufferTest{
    public static void main(String[] args){
        StringBuffer buffer=new StringBuffer("hello");
        buffer.append("java");
        System.out.println(buffer);
    }
}
```

```
hellojava
```

StringBuilder Example

```
//Java Program to demonstrate the use of StringBuilder class.
public class BuilderTest{
    public static void main(String[] args){
        StringBuilder builder=new StringBuilder("hello");
        builder.append("java");
        System.out.println(builder);
    }
}
```

```
hellojava
```

Performance Test of StringBuffer and StringBuilder

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

```
//Java Program to demonstrate the performance of StringBuffer and StringBuilder classes.
public class ConcatTest{
    public static void main(String[] args){
        long startTime = System.currentTimeMillis();
        StringBuffer sb = new StringBuffer("Java");
        for (int i=0; i<10000; i++){
            sb.append("Tpoint");
        }
        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("Tpoint");
        }
        System.out.println("Time taken by StringBuilder: " +
    (System.currentTimeMillis() - startTime) + "ms");
    }
}
```

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

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.

```
public final class Employee{
final String pancardNumber;

public Employee(String pancardNumber){
this.pancardNumber=pancardNumber;
}

public String getPancardNumber(){
return pancardNumber;
}

}
```

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.
- TThese points makes this class as immutable.

  here is no setter methods i.e. we have no option to change the value of the instance variable.0

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.

```
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
 }

 public static void main(String args[]){
   Student s1=new Student(101,"Raj","lucknow");
   Student s2=new Student(102,"Vijay","ghaziabad");

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

```
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

Now let's see the real example of toString() method.

```java
class Student{
 int rollno;
 String name;
 String city;

 Student(int rollno, String name, String city){
 this.rollno=rollno;
 this.name=name;
 this.city=city;
}

public String toString(){//overriding the toString() method
```

```
 return rollno+" "+name+" "+city;
}
public static void main(String args[]){
  Student s1=new Student(101,"Raj","lucknow");
  Student s2=new Student(102,"Vijay","ghaziabad");

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

```
Output:101 Raj lucknow
       102 Vijay ghaziabad
```

StringTokenizer in Java

1. StringTokenizer
2. Methods of StringTokenizer
3. Example of StringTokenizer

The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

Constructors of StringTokenizer class

There are 3 constructors defined in the StringTokenizer class.

Constructor	Description

StringTokenizer(String str)	creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	creates StringTokenizer with specified string and delimeter.
StringTokenizer(String str, String delim, boolean returnValue)	creates StringTokenizer with specified string, delimeter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Methods of StringTokenizer class

The 6 useful methods of StringTokenizer class are as follows:

Public method	**Description**
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object.
String nextToken(String delim)	returns the next token based on the delimeter.
boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

Simple example of StringTokenizer class

Let's see the simple example of StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

```
import java.util.StringTokenizer;
public class Simple{
 public static void main(String args[]){
   StringTokenizer st = new StringTokenizer("my name is khan"," ");
     while (st.hasMoreTokens()) {
         System.out.println(st.nextToken());
     }
   }
}
```

```
Output:my
       name
       is
       khan
```

Example of nextToken(String delim) method of StringTokenizer class

```
import java.util.*;

public class Test {
   public static void main(String[] args) {
       StringTokenizer st = new StringTokenizer("my,name,is,khan");

      // printing next token
      System.out.println("Next token is : " + st.nextToken(","));
   }
}
```

Output:Next token is : my

StringTokenizer class is deprecated now. It is recommended to use split() method of String class or regex (Regular Expression).

Java String Methods

ava String charAt()

The **java string charAt()** method returns *a char value at the given index number*.

The index number starts from 0 and goes to n-1, where n is length of the string. It returns **StringIndexOutOfBoundsException** if given index number is greater than or equal to this string length or a negative number.

Internal implementation

```java
public char charAt(int index) {
       if ((index < 0) || (index >= value.length)) {
           throw new StringIndexOutOfBoundsException(index);
       }
       return value[index];
   }
```

Signature

The signature of string charAt() method is given below:

```java
public char charAt(int index)
```

Parameter

index : index number, starts with 0

Returns

A char value

Specified by

CharSequence interface, located inside java.lang package.

Throws

StringIndexOutOfBoundsException : if index is negative value or greater than this string length.

Java String charAt() method example

```java
public class CharAtExample{
public static void main(String args[]){
String name="javatpoint";
char ch=name.charAt(4);//returns the char value at the 4th index
System.out.println(ch);
}}
```

Test it Now

Output:

```
t
```

StringIndexOutOfBoundsException with charAt()

Let's see the example of charAt() method where we are passing greater index value. In such case, it throws StringIndexOutOfBoundsException at run time.

```java
public class CharAtExample{
public static void main(String args[]){
String name="javatpoint";
char ch=name.charAt(10);//returns the char value at the 10th index
System.out.println(ch);
}}
```

Output:

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
String index out of range: 10
at java.lang.String.charAt(String.java:658)
at CharAtExample.main(CharAtExample.java:4)
```

Java String charAt() Example 3

Let's see a simple example where we are accessing first and last character from the provided string.

```java
public class CharAtExample3 {
    public static void main(String[] args) {
    String str = "Welcome to Javatpoint portal";
    int strLength = str.length();
    // Fetching first character
    System.out.println("Character at 0 index is: "+ str.charAt(0));
    // The last Character is present at the string length-1 index
    System.out.println("Character at last index is: "+ str.charAt(strLength-1));
    }
}
```

Output:

```
Character at 0 index is: W
Character at last index is: l
```

Java String charAt() Example 4

Let's see an example where we are accessing all the elements present at odd index.

```java
public class CharAtExample4 {
    public static void main(String[] args) {
        String str = "Welcome to Javatpoint portal";
        for (int i=0; i<=str.length()-1; i++) {
            if(i%2!=0) {
                System.out.println("Char at "+i+" place "+str.charAt(i));
            }
        }
    }
}
```

Output:

```
Char at 1 place e
Char at 3 place c
Char at 5 place m
Char at 7 place
Char at 9 place o
Char at 11 place J
Char at 13 place v
Char at 15 place t
Char at 17 place o
Char at 19 place n
Char at 21 place
Char at 23 place o
Char at 25 place t
Char at 27 place l
```

Java String charAt() Example 5

Let's see an example where we are counting frequency of a character in the string.

```
public class CharAtExample5 {
    public static void main(String[] args) {
        String str = "Welcome to Javatpoint portal";
        int count = 0;
        for (int i=0; i<=str.length()-1; i++) {
            if(str.charAt(i) == 't') {
                count++;
            }
        }
        System.out.println("Frequency of t is: "+count);
    }
}
```

Output:

```
Frequency of t is: 4
```

Java String concat

The **java string concat()** method *combines specified string at the end of this string*. It returns combined string. It is like appending another string.

Internal implementation

```
public String concat(String str) {
        int otherLen = str.length();
        if (otherLen == 0) {
            return this;
        }
        int len = value.length;
        char buf[] = Arrays.copyOf(value, len + otherLen);
        str.getChars(buf, len);
        return new String(buf, true);
    }
```

Signature

The signature of string concat() method is given below:

```
public String concat(String anotherString)
```

Parameter

anotherString : another string i.e. to be combined at the end of this string.

Returns

combined string

Java String concat() method example

```java
public class ConcatExample{
public static void main(String args[]){
String s1="java string";
s1.concat("is immutable");
System.out.println(s1);
s1=s1.concat(" is immutable so assign it explicitly");
System.out.println(s1);
}}
```

Test it Now

```
java string

java string is immutable so assign it explicitly
```

Java String concat() Method Example 2

Let's see an example where we are concatenating multiple string objects.

```java
public class ConcatExample2 {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "Javatpoint";
        String str3 = "Reader";
        // Concatenating one string
        String str4 = str1.concat(str2);
        System.out.println(str4);
        // Concatenating multiple strings
        String str5 = str1.concat(str2).concat(str3);
        System.out.println(str5);
    }
}
```

Output:

```
HelloJavatpoint

HelloJavatpointReader
```

Java String concat() Method Example 3

Let's see an example where we are concatenating spaces and special chars to the string object.

```java
public class ConcatExample3 {
    public static void main(String[] args) {
        String str1 = "Hello";
        String str2 = "Javatpoint";
        String str3 = "Reader";
        // Concatenating Space among strings
        String str4 = str1.concat(" ").concat(str2).concat(" ").concat(str3);
        System.out.println(str4);
        // Concatenating Special Chars
        String str5 = str1.concat("!!!");
        System.out.println(str5);
        String str6 = str1.concat("@").concat(str2);
        System.out.println(str6);
    }
}
```

Output:

```
Hello Javatpoint Reader

Hello!!!

Hello@Javatpoint
```

Next »« Prev

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*.

Internal implementation

```
public boolean contains(CharSequence s) {
       return indexOf(s.toString()) > -1;
   }
```

Signature

The signature of string contains() method is given below:

```
public boolean contains(CharSequence sequence)
```

Parameter

sequence : specifies the sequence of characters to be searched.

Returns

true if sequence of char value exists, otherwise **false**.

Throws

NullPointerException : if sequence is null.

Java String contains() method example

```java
class ContainsExample{
public static void main(String args[]){
String name="what do you know about me";
System.out.println(name.contains("do you know"));
System.out.println(name.contains("about"));
System.out.println(name.contains("hello"));
}}
```

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.

```java
public class ContainsExample2 {
    public static void main(String[] args) {
        String str = "Hello Javatpoint readers";
        boolean isContains = str.contains("Javatpoint");
        System.out.println(isContains);
        // Case Sensitive
        System.out.println(str.contains("javatpoint")); // false
    }
}
```

```
true
false
```

Java String contains() Method Example 3

The contains() method is helpful to find a char-sequence in the string. We can use it in control structure to produce search based result. Let us see an example below.

```
public class ContainsExample3 {
    public static void main(String[] args) {
        String str = "To learn Java visit Javatpoint.com";
        if(str.contains("Javatpoint.com")) {
            System.out.println("This string contains javatpoint.com");
        }else
            System.out.println("Result not found");
    }
}
```

Output:

```
This string contains javatpoint.com
```

Java String endsWith()

The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

Internal implementation

```
public boolean endsWith(String suffix) {
     return startsWith(suffix, value.length - suffix.value.length);
 }
```

Signature

The syntax or signature of endsWith() method is given below.

```
public boolean endsWith(String suffix)
```

Parameter

suffix : Sequence of character

Returns

true or false

Java String endsWith() method example

```
public class EndsWithExample{
public static void main(String args[]){
String s1="java by javatpoint";
System.out.println(s1.endsWith("t"));
System.out.println(s1.endsWith("point"));
}}
```

Test it Now

Output:

```
true
true
```

Java String endsWith() Method Example 2

```java
public class EndsWithExample2 {
    public static void main(String[] args) {
        String str = "Welcome to Javatpoint.com";
        System.out.println(str.endsWith("point"));
        if(str.endsWith(".com")) {
            System.out.println("String ends with .com");
        }else System.out.println("It does not end with .com");
    }
}
```

Output:

```
false
String ends with .com
```

Java String equals()

The **java string equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of Object class.

Internal implementation

```java
public boolean equals(Object anObject) {
       if (this == anObject) {
           return true;
       }
       if (anObject instanceof String) {
           String anotherString = (String) anObject;
           int n = value.length;
           if (n == anotherString.value.length) {
               char v1[] = value;
               char v2[] = anotherString.value;
               int i = 0;
               while (n-- != 0) {
                   if (v1[i] != v2[i])
                           return false;
                   i++;
               }
               return true;
           }
       }
       return false;
   }
```

Signature

```java
public boolean equals(Object anotherObject)
```

Parameter

anotherObject : another object i.e. compared with this string.

Returns

true if characters of both strings are equal otherwise **false**.

Overrides

equals() method of java Object class.

Java String equals() method example

```java
public class EqualsExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equals(s2));//true because content and case is same
System.out.println(s1.equals(s3));//false because case is not same
System.out.println(s1.equals(s4));//false because content is not same
}}
```

Test it Now

```
true
false
false
```

Java String equals() Method Example 2

The equals() method compares two strings and can be used in if-else control structure.

```
public class EqualsExample {
    public static void main(String[] args) {
        String s1 = "javatpoint";
        String s2 = "javatpoint";
        String s3 = "Javatpoint";
        System.out.println(s1.equals(s2)); // True because content is same
        if (s1.equals(s3)) {
            System.out.println("both strings are equal");
        }else System.out.println("both strings are unequal");
    }
}
```

```
true
both strings are unequal
```

Java String equals() Method Example 3

Let's see one more example to test the equality of string present in the list.

```
import java.util.ArrayList;
public class EqualsExample3 {
    public static void main(String[] args) {
        String str1 = "Mukesh";
        ArrayList<String> list = new ArrayList<>();
        list.add("Ravi");
        list.add("Mukesh");
        list.add("Ramesh");
        list.add("Ajay");
        for (String str : list) {
            if (str.equals(str1)) {
                System.out.println("Mukesh is present");
            }
        }
    }
}
```

```
Mukesh is present
```

Java String equalsIgnoreCase()

The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string. It is like equals() method but doesn't check case. If any character is not matched, it returns false otherwise it returns true.

Internal implementation

```
public boolean equalsIgnoreCase(String anotherString) {
       return (this == anotherString) ? true
               : (anotherString != null)
               && (anotherString.value.length == value.length)
               && regionMatches(true, 0, anotherString, 0, value.length);
   }
```

Signature

```
public boolean equalsIgnoreCase(String str)
```

Parameter

str : another string i.e. compared with this string.

Returns

It returns **true** if characters of both strings are equal ignoring case otherwise **false**.

Java String equalsIgnoreCase() method example

```java
public class EqualsIgnoreCaseExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equalsIgnoreCase(s2));//true because content and case both are same
System.out.println(s1.equalsIgnoreCase(s3));//true because case is ignored
System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same
}}
```

Test it Now

```
true
true
false
```

Java String format()

The **java string format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()* method.

The format() method of java language is like *sprintf()* function in c language and *printf()* method of java language.

Internal implementation

```
public static String format(String format, Object... args) {
       return new Formatter().format(format, args).toString();
   }
```

Signature

There are two type of string format() method:

```
public static String format(String format, Object... args)
and,
public static String format(Locale locale, String format, Object... args)
```

Parameters

locale : specifies the locale to be applied on the format() method.

format : format of the string.

args : arguments for the format string. It may be zero or more.

Returns

formatted string

Throws

NullPointerException : if format is null.

IllegalFormatException : if format is illegal or incompatible.

Java String format() method example

```java
public class FormatExample{
public static void main(String args[]){
String name="sonoo";
String sf1=String.format("name is %s",name);
String sf2=String.format("value is %f",32.33434);
String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0

System.out.println(sf1);
System.out.println(sf2);
System.out.println(sf3);
}}
```

Test it Now

```
name is sonoo
value is 32.334340
value is                  32.334340000000
```

Java String Format Specifiers

Here, we are providing a table of format specifiers supported by the Java String.

Format	Data Type	Output

Specifier		
%a	floating point (except *BigDecimal*)	Returns Hex output of floating point number.
%b	Any type	"true" if non-null, "false" if null
%c	character	Unicode character
%d	integer (incl. byte, short, int, long, bigint)	Decimal Integer
%e	floating point	decimal number in scientific notation
%f	floating point	decimal number
%g	floating point	decimal number, possibly in scientific notation depending on the precision and value.
%h	any type	Hex String of value from hashCode() method.
%n	none	Platform-specific line separator.
%o	integer (incl. byte, short, int, long, bigint)	Octal number
%s	any type	String value
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	%t is the prefix for Date/Time conversions. More formatting flags are needed after this. See Date/Time conversion below.

%x	integer (incl. byte, short, int, long, bigint)	Hex string.

Java String format() Method Example 2

This method supports various data types and formats them into a string type. Let us see an example.

```
public class FormatExample2 {
    public static void main(String[] args) {
        String str1 = String.format("%d", 101);          // Integer value
        String str2 = String.format("%s", "Amar Singh"); // String value
        String str3 = String.format("%f", 101.00);       // Float value
        String str4 = String.format("%x", 101);          // Hexadecimal value
        String str5 = String.format("%c", 'c');          // Char value
        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(str5);
    }

}
```

Test it Now

```
101
Amar Singh
101.000000
65
c
```

Java String format() Method Example 3

Apart from formatting, we can set width, padding etc. of any value. Let us see an example where we are setting width and padding for an integer value.

```java
public class FormatExample3 {
    public static void main(String[] args) {
        String str1 = String.format("%d", 101);
        String str2 = String.format("|%10d|", 101);  // Specifying length of integer
        String str3 = String.format("|%-10d|", 101); // Left-justifying within the specified width
        String str4 = String.format("|% d|", 101);
        String str5 = String.format("|%010d|", 101); // Filling with zeroes
        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(str5);
    }
}
```

Test it Now

```
101
|       101|
|101       |
| 101|
|0000000101|
```

Java String getBytes()

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

Signature

There are 3 variant of getBytes() method. The signature or syntax of string getBytes() method is given below:

```
public byte[] getBytes()
public byte[] getBytes(Charset charset)
public byte[] getBytes(String charsetName)throws
UnsupportedEncodingException
```

Internal implementation

```
public byte[] getBytes() {
        return StringCoding.encode(value, 0, value.length);
    }
```

Returns

sequence of bytes.

Java String getBytes() method example

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

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.

```java
public class StringGetBytesExample2 {
    public static void main(String[] args) {
        String s1 = "ABCDEFG";
        byte[] barr = s1.getBytes();
        for(int i=0;i<barr.length;i++){
            System.out.println(barr[i]);
        }
        // Getting string back
        String s2 = new String(barr);
        System.out.println(s2);
    }
}
```

Test it Now

Output:

```
65
66
67
68
```

```
69
70
71
```

```
ABCDEFG
```

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:

Internal implementation

```java
void getChars(char dst[], int dstBegin) {
        System.arraycopy(value, 0, dst, dstBegin, value.length);
    }
```

Signature

The signature or syntax of string getChars() method is given below:

```java
public void getChars(int srcBeginIndex, int srcEndIndex, char[] destination, int dstBeginIndex)
```

Returns

It doesn't return any value.

Throws

It throws StringIndexOutOfBoundsException if beginIndex is greater than endIndex.

Java String getChars() method example

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

Test it Now

Output:

```
javatpoint
```

Java String getChars() Method Example 2

It throws an exception if index value exceeds array range. Let's see an example.

```
public class StringGetCharsExample2 {
    public static void main(String[] args) {
        String str = new String("Welcome to Javatpoint");
        char[] ch  = new char[20];
        try {
            str.getChars(1, 26, ch, 0);
            System.out.println(ch);
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output:

```
java.lang.StringIndexOutOfBoundsException: offset 10, count 14, length 20
```

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.

Internal implementation

```
public int indexOf(int ch) {
        return indexOf(ch, 0);
    }
```

Signature

There are 4 types of indexOf method in java. The signature of indexOf methods are given below:

No.	Method	Description
1	int indexOf(int ch)	returns index position for the given char value
2	int indexOf(int ch, int fromIndex)	returns index position for the given char value and from index
3	int indexOf(String substring)	returns index position for the given substring

4	int indexOf(String substring, int fromIndex)	returns index position for the given substring and from index

Parameters

ch: char value i.e. a single character e.g. 'a'

fromIndex: index position from where index of the char value or substring is retured

substring: substring to be searched in this string

Returns

index of the string

Java String indexOf() method example

```java
public class IndexOfExample{
public static void main(String args[]){
String s1="this is index of example";
//passing substring
int index1=s1.indexOf("is");//returns the index of is substring
int index2=s1.indexOf("index");//returns the index of index substring
System.out.println(index1+"  "+index2);//2 8

//passing substring with from index
int index3=s1.indexOf("is",4);//returns the index of is substring after 4th index
System.out.println(index3);//5 i.e. the index of another is

//passing char value
int index4=s1.indexOf('s');//returns the index of s char value
System.out.println(index4);//3
}}
```

Test it Now

```
2  8
5
3
```

Java String indexOf(String substring) Method Example

This method takes substring as an argument and returns index of first character of the substring.

```
public class IndexOfExample2 {
    public static void main(String[] args) {
        String s1 = "This is indexOf method";
        // Passing Substring
        int index = s1.indexOf("method"); //Returns the index of this substring
        System.out.println("index of substring "+index);
    }

}
```

Test it Now

```
index of substring 16
```

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*.

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

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*.

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

Test it Now

```
index of char 17
```

Next »« Prev

Java String intern()

The **java string intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from memory, if it is created by new keyword. It creates exact copy of heap string object in string constant pool.

Signature

The signature of intern method is given below:

```java
public String intern()
```

Returns

interned string

Java String intern() method example

```java
public class InternExample{
public static void main(String args[]){
String s1=new String("hello");
String s2="hello";
String s3=s1.intern();//returns string from pool, now it will be same as s2
System.out.println(s1==s2);//false because reference variables are pointing to different instance
System.out.println(s2==s3);//true because reference variables are pointing to same instance
}}
```

Test it Now

```
false
true
```

Java String intern() Method Example 2

Let's see one more example to understand the string intern concept.

```
public class InternExample2 {
    public static void main(String[] args) {
        String s1 = "Javatpoint";
        String s2 = s1.intern();
        String s3 = new String("Javatpoint");
        String s4 = s3.intern();
        System.out.println(s1==s2); // True
        System.out.println(s1==s3); // False
        System.out.println(s1==s4); // True
        System.out.println(s2==s3); // False
        System.out.println(s2==s4); // True
        System.out.println(s3==s4); // False
    }
}
```

Test it Now

```
true
false
true
false
true
false
```

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.

The isEmpty() method of String class is included in java string since JDK 1.6.

Internal implementation

```
public boolean isEmpty() {
        return value.length == 0;
    }
```

Signature

The signature or syntax of string isEmpty() method is given below:

```
public boolean isEmpty()
```

Returns

true if length is 0 otherwise false.

Since

1.6

Java String isEmpty() method example

```
public class IsEmptyExample{
public static void main(String args[]){
String s1="";
String s2="javatpoint";

System.out.println(s1.isEmpty());
System.out.println(s2.isEmpty());
}}
```

Test it Now

```
true
false
```

Java String isEmpty() Method Example 2

```
public class IsEmptyExample2 {
    public static void main(String[] args) {
        String s1="";
        String s2="Javatpoint";
        // Either length is zero or isEmpty is true
        if(s1.length()==0 || s1.isEmpty())
            System.out.println("String s1 is empty");
        else System.out.println("s1");
        if(s2.length()==0 || s2.isEmpty())
            System.out.println("String s2 is empty");
        else System.out.println(s2);
```

```
   }
}
```

```
String s1 is empty
Javatpoint
```

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. The join() method is included in java string since JDK 1.8.

There are two types of join() methods in java string.

Signature

The signature or syntax of string join method is given below:

```
public static String join(CharSequence delimiter, CharSequence... elements)
and
public static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)
```

Parameters

delimiter : char value to be added with each element

elements : char value to be attached with delimiter

Returns

joined string with delimiter

Throws

NullPointerException if element or delimiter is null.

Since

1.8

Java String join() method example

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

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.

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

25/06/2018 12:10:10

Next »« Prev

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.

Internal Implementation

```
public int lastIndexOf(int ch) {
        return lastIndexOf(ch, value.length - 1);
    }
```

Signature

There are 4 types of lastIndexOf method in java. The signature of lastIndexOf methods are given below:

No.	Method	Description
1	int lastIndexOf(int ch)	returns last index position for the given char value
2	int lastIndexOf(int ch, int fromIndex)	returns last index position for the given char value and from index

3	int lastIndexOf(String substring)	returns last index position for the given substring
4	int lastIndexOf(String substring, int fromIndex)	returns last index position for the given substring and from index

Parameters

ch: char value i.e. a single character e.g. 'a'

fromIndex: index position from where index of the char value or substring is retured

substring: substring to be searched in this string

Returns

last index of the string

Java String lastIndexOf() method example

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

Test it Now

Output:

```
6
```

Java String lastIndexOf(int ch, int fromIndex) Method Example

Here, we are finding last index from the string by specifying *fromIndex*

```
public class LastIndexOfExample2 {
    public static void main(String[] args) {
        String str = "This is index of example";
        int index = str.lastIndexOf('s',5);
        System.out.println(index);
    }
}
```

Test it Now

Output:

```
3
```

Java String lastIndexOf(String substring) Method Example

It returns the last index of the substring.

```java
public class LastIndexOfExample3 {
    public static void main(String[] args) {
        String str = "This is last index of example";
        int index = str.lastIndexOf("of");
        System.out.println(index);
    }
}
```

Test it Now

Output:

```
19
```

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

It returns the last index of the substring from the *fromIndex*.

```java
public class LastIndexOfExample4 {
    public static void main(String[] args) {
        String str = "This is last index of example";
        int index = str.lastIndexOf("of", 25);
        System.out.println(index);
        index = str.lastIndexOf("of", 10);
        System.out.println(index); // -1, if not found
    }
}
```

Test it Now

Output:

```
19
-1
```

Java String length()

The **java string length()** method length of the string. It returns count of total number of characters. The length of java string is same as the unicode code units of the string.

Internal implementation

```
public int length() {
        return value.length;
    }
```

Signature

The signature of the string length() method is given below:

```
public int length()
```

Specified by

CharSequence interface

Returns

length of characters

Java String length() method example

```java
public class LengthExample{
public static void main(String args[]){
String s1="javatpoint";
String s2="python";
System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string
System.out.println("string length is: "+s2.length());//6 is the length of python string
}}
```

Test it Now

```
string length is: 10
string length is: 6
```

Java String length() Method Example 2

```java
public class LengthExample2 {
    public static void main(String[] args) {
        String str = "Javatpoint";
        if(str.length()>0) {
            System.out.println("String is not empty and length is: "+str.length());
        }
        str = "";
        if(str.length()==0) {
            System.out.println("String is empty now: "+str.length());
        }
    }
}
```

```
String is not empty and length is: 10
String is empty now: 0
```

Java String replace()

The **java string replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Since JDK 1.5, a new replace() method is introduced, allowing you to replace a sequence of char values.

Internal implementation

```java
public String replace(char oldChar, char newChar) {
        if (oldChar != newChar) {
            int len = value.length;
            int i = -1;
            char[] val = value; /* avoid getfield opcode */

            while (++i < len) {
                if (val[i] == oldChar) {
                    break;
                }
            }
            if (i < len) {
```

```java
            char buf[] = new char[len];
            for (int j = 0; j < i; j++) {
                buf[j] = val[j];
            }
            while (i < len) {
                char c = val[i];
                buf[i] = (c == oldChar) ? newChar : c;
                i++;
            }
            return new String(buf, true);
        }
    }
    return this;
}
```

Signature

There are two type of replace methods in java string.

```java
public String replace(char oldChar, char newChar)
and
public String replace(CharSequence target, CharSequence replacement)
```

The second replace method is added since JDK 1.5.

Parameters

oldChar : old character

newChar : new character

target : target sequence of characters

replacement : replacement sequence of characters

Returns

replaced string

Java String replace(char old, char new) method example

```
public class ReplaceExample1{
public static void main(String args[]){
String s1="javatpoint is a very good website";
String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
System.out.println(replaceString);
}}
```

Test it Now

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

Java String replace(CharSequence target, CharSequence replacement) method example

```
public class ReplaceExample2{
public static void main(String args[]){
String s1="my name is khan my name is java";
String replaceString=s1.replace("is","was");//replaces all occurrences of "is" to "was"
System.out.println(replaceString);
}}
```

Test it Now

```
my name was khan my name was java
```

Java String replace() Method Example 3

```java
public class ReplaceExample3 {
    public static void main(String[] args) {
        String str = "oooooo-hhhh-oooooo";
        String rs = str.replace("h","s"); // Replace 'h' with 's'
        System.out.println(rs);
        rs = rs.replace("s","h"); // Replace 's' with 'h'
        System.out.println(rs);
    }
}
```

```
oooooo-ssss-oooooo
oooooo-hhhh-oooooo
```

Java String replaceAll()

The **java string replaceAll()** method returns a string replacing all the sequence of characters matching regex and replacement string.

Internal implementation

```java
public String replaceAll(String regex, String replacement) {
        return Pattern.compile(regex).matcher(this).replaceAll(replacement);
}
```

Signature

```java
public String replaceAll(String regex, String replacement)
```

Parameters

regex : regular expression

replacement : replacement sequence of characters

Returns

replaced string

Java String replaceAll() example: replace character

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

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

Test it Now

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

Java String replaceAll() example: replace word

Let's see an example to replace all the occurrences of **single word or set of words**.

```
public class ReplaceAllExample2{
public static void main(String args[]){
```

```java
String s1="My name is Khan. My name is Bob. My name is Sonoo.";
String replaceString=s1.replaceAll("is","was");//replaces all occurrences of "is" to "was"
System.out.println(replaceString);
}}
```

Test it Now

```
My name was Khan. My name was Bob. My name was Sonoo.
```

Java String replaceAll() example: remove white spaces

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

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

Test it Now

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

Java String split()

The **java string split()** method splits this string against given regular expression and returns a char array.

Internal implementation

```java
public String[] split(String regex, int limit) {
        /* fastpath if the regex is a
         (1)one-char String and this character is not one of the
         RegEx's meta characters ".$|()[{^?*+\\", or
      (2)two-char String and the first char is the backslash and
         the second is not the ascii digit or ascii letter.
      */
     char ch = 0;
     if (((regex.value.length == 1 &&
          ".$|()[{^?*+\\".indexOf(ch = regex.charAt(0)) == -1) ||
          (regex.length() == 2 &&
           regex.charAt(0) == '\\' &&
           (((ch = regex.charAt(1))-'0')|('9'-ch)) < 0 &&
           ((ch-'a')|('z'-ch)) < 0 &&
           ((ch-'A')|('Z'-ch)) < 0)) &&
         (ch < Character.MIN_HIGH_SURROGATE ||
          ch > Character.MAX_LOW_SURROGATE))
     {
         int off = 0;
         int next = 0;
         boolean limited = limit > 0;
         ArrayList<String> list = new ArrayList<>();
         while ((next = indexOf(ch, off)) != -1) {
             if (!limited || list.size() < limit - 1) {
                 list.add(substring(off, next));
                 off = next + 1;
             } else {    // last one
                 //assert (list.size() == limit - 1);
                 list.add(substring(off, value.length));
                 off = value.length;
                 break;
```

```
            }
        }
        // If no match was found, return this
        if (off == 0)
            return new String[]{this};

        // Add remaining segment
        if (!limited || list.size() < limit)
            list.add(substring(off, value.length));

        // Construct result
        int resultSize = list.size();
        if (limit == 0)
            while (resultSize > 0 && list.get(resultSize - 1).length() == 0)
                resultSize--;
        String[] result = new String[resultSize];
        return list.subList(0, resultSize).toArray(result);
    }
    return Pattern.compile(regex).split(this, limit);
}
```

Signature

There are two signature for split() method in java string.

```
public String split(String regex)
and,
public String split(String regex, int limit)
```

Parameter

regex : regular expression to be applied on string.

limit : limit for the number of strings in array. If it is zero, it will returns all the strings matching regex.

Returns

array of strings

Throws

PatternSyntaxException if pattern for regular expression is invalid

---

Since

1.4

Java String split() method example

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

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

Test it Now

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

Java String split() method with regex and length example

```
public class SplitExample2{
public static void main(String args[]){
String s1="welcome to split world";
System.out.println("returning words:");
for(String w:s1.split("\\s",0)){
System.out.println(w);
}
System.out.println("returning words:");
for(String w:s1.split("\\s",1)){
System.out.println(w);
}
System.out.println("returning words:");
for(String w:s1.split("\\s",2)){
System.out.println(w);
}

}}
```

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.

```java
public class SplitExample3 {
    public static void main(String[] args) {
        String str = "Javatpointtt";
        System.out.println("Returning words:");
        String[] arr = str.split("t", 0);
        for (String w : arr) {
            System.out.println(w);
        }
        System.out.println("Split array length: "+arr.length);
    }
}
```

```
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.

Internal implementation

```
public boolean startsWith(String prefix, int toffset) {
    char ta[] = value;
    int to = toffset;
    char pa[] = prefix.value;
    int po = 0;
    int pc = prefix.value.length;
    // Note: toffset might be near -1>>>1.
    if ((toffset < 0) || (toffset > value.length - pc)) {
        return false;
    }
    while (--pc >= 0) {
        if (ta[to++] != pa[po++]) {
            return false;
        }
    }
    return true;
}
```

Signature

The syntax or signature of startWith() method is given below.

```java
public boolean startsWith(String prefix)
public boolean startsWith(String prefix, int offset)
```

Parameter

prefix : Sequence of character

Returns

true or false

Java String startsWith() method example

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

Test it Now

Output:

```
true
true
```

Java String startsWith(String prefix, int offset) Method Example

This is overloaded method of startWith() method which is used to pass one extra argument (offset) to the function. This method works from the passed offset. Let's see an example.

```java
public class StartsWithExample2 {
    public static void main(String[] args) {
        String str = "Javatpoint";
        System.out.println(str.startsWith("J")); // True
        System.out.println(str.startsWith("a")); // False
        System.out.println(str.startsWith("a",1)); // True
    }
}
```

Output:

```
true
false
true
```

Next »« Prev

Java String substring()

The **java string substring()** method returns a part of the string.

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.

There are two types of substring methods in java string.

Internal implementation

```
public String substring(int beginIndex) {
        if (beginIndex < 0) {
            throw new StringIndexOutOfBoundsException(beginIndex);
        }
        int subLen = value.length - beginIndex;
        if (subLen < 0) {
            throw new StringIndexOutOfBoundsException(subLen);
        }
        return (beginIndex == 0) ? this : new String(value, beginIndex, subLen);
    }
```

Signature

```
public String substring(int startIndex)
```

```
and
public String substring(int startIndex, int endIndex)
```

If you don't specify endIndex, java substring() method will return all the characters from startIndex.

Parameters

startIndex : starting index is inclusive

endIndex : ending index is exclusive

Returns

specified string

Throws

StringIndexOutOfBoundsException if start index is negative value or end index is lower than starting index.

Java String substring() method example

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

Test it Now

```
va
vatpoint
```

Java String substring() Method Example 2

```
public class SubstringExample2 {
    public static void main(String[] args) {
        String s1="Javatpoint";
        String substr = s1.substring(0); // Starts with 0 and goes to end
        System.out.println(substr);
        String substr2 = s1.substring(5,10); // Starts from 5 and goes to 10
        System.out.println(substr2);
        String substr3 = s1.substring(5,15); // Returns Exception
    }
}
```

```
Javatpoint
point
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 5, end 15, length 10
```

Java String toCharArray()

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 and its contents are initialized with the characters of this string.

Internal implementation

```
public char[] toCharArray() {
      // Cannot use Arrays.copyOf because of class initialization order issues
      char result[] = new char[value.length];
      System.arraycopy(value, 0, result, 0, value.length);
      return result;
  }
```

Signature

The signature or syntax of string toCharArray() method is given below:

```
public char[] toCharArray()
```

Returns

character array

Java String toCharArray() method example

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

Test it Now

Output:

```
hello
```

Java String toCharArray() Method Example 2

Let's see one more example of char array. It is useful method which returns char array from the string without writing any custom code.

```java
public class StringToCharArrayExample2 {
    public static void main(String[] args) {
        String s1 = "Welcome to Javatpoint";
        char[] ch = s1.toCharArray();
        int len = ch.length;
        System.out.println("Char Array length: " + len);
        System.out.println("Char Array elements: ");
        for (int i = 0; i < len; i++) {
            System.out.println(ch[i]);
        }
    }
}
```

Output:

```
Char Array length: 21
Char Array elements:
W
e
l
c
o
m
```

Java String toLowerCase()

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

Internal implementation

```java
public String toLowerCase(Locale locale) {
        if (locale == null) {
            throw new NullPointerException();
        }

        int firstUpper;
        final int len = value.length;

        /* Now check if there are any characters that need to be changed. */
        scan: {
            for (firstUpper = 0 ; firstUpper < len; ) {
                char c = value[firstUpper];
                if ((c >= Character.MIN_HIGH_SURROGATE)
                        && (c <= Character.MAX_HIGH_SURROGATE)) {
                    int supplChar = codePointAt(firstUpper);
                    if (supplChar != Character.toLowerCase(supplChar)) {
                        break scan;
                    }
                    firstUpper += Character.charCount(supplChar);
                } else {
                    if (c != Character.toLowerCase(c)) {
                        break scan;
                    }
                    firstUpper++;
                }
            }
            return this;
        }

        char[] result = new char[len];
        int resultOffset = 0;  /* result may grow, so i+resultOffset
                                * is the write location in result */

        /* Just copy the first few lowerCase characters. */
        System.arraycopy(value, 0, result, 0, firstUpper);

        String lang = locale.getLanguage();
        boolean localeDependent =
                (lang == "tr" || lang == "az" || lang == "lt");
        char[] lowerCharArray;
        int lowerChar;
        int srcChar;
        int srcCount;
        for (int i = firstUpper; i < len; i += srcCount) {
            srcChar = (int)value[i];
            if ((char)srcChar >= Character.MIN_HIGH_SURROGATE
                    && (char)srcChar <= Character.MAX_HIGH_SURROGATE) {
                srcChar = codePointAt(i);
                srcCount = Character.charCount(srcChar);
            } else {
                srcCount = 1;
            }
            if (localeDependent || srcChar == '\u03A3') { // GREEK CAPITAL LETTER SIGMA
                lowerChar = ConditionalSpecialCasing.toLowerCaseEx(this, i, locale);
            } else if (srcChar == '\u0130') { // LATIN CAPITAL LETTER I DOT
                lowerChar = Character.ERROR;
            } else {
                lowerChar = Character.toLowerCase(srcChar);
            }
            if ((lowerChar == Character.ERROR)
                    || (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
                if (lowerChar == Character.ERROR) {
                    if (!localeDependent && srcChar == '\u0130') {
                        lowerCharArray =
                            ConditionalSpecialCasing.toLowerCaseCharArray(this, i, Locale.ENGLISH);
                } else {
                    lowerCharArray =
                            ConditionalSpecialCasing.toLowerCaseCharArray(this, i, locale);
                }
            } else if (srcCount == 2) {
                resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCount;
                continue;
            } else {
                lowerCharArray = Character.toChars(lowerChar);
            }

            /* Grow result if needed */
            int mapLen = lowerCharArray.length;
            if (mapLen > srcCount) {
                char[] result2 = new char[result.length + mapLen - srcCount];
                System.arraycopy(result, 0, result2, 0, i + resultOffset);
                result = result2;
            }
            for (int x = 0; x < mapLen; ++x) {
                result[i + resultOffset + x] = lowerCharArray[x];
            }
            resultOffset += (mapLen - srcCount);
        } else {
            result[i + resultOffset] = (char)lowerChar;
        }
    }
    return new String(result, 0, len + resultOffset);
}
```

Signature

There are two variant of toLowerCase() method. The signature or syntax of string toLowerCase() method is given below:

```
public String toLowerCase()
public String toLowerCase(Locale locale)
```

The second method variant of toLowerCase(), converts all the characters into lowercase using the rules of given Locale.

Returns

string in lowercase letter.

Java String toLowerCase() method example

```
public class StringLowerExample{
public static void main(String args[]){
String s1="JAVATPOINT HELLO stRIng";
String s1lower=s1.toLowerCase();
System.out.println(s1lower);
}}
```

Test it Now

Output:

```
javatpoint hello string
```

Java String toLowerCase(Locale locale) Method Example 2

This method allows us to pass locale too for the various langauges. Let's see an example below where we are getting string in english and turkish both.

```java
import java.util.Locale;
public class StringLowerExample2 {
    public static void main(String[] args) {
        String s = "JAVATPOINT HELLO stRIng";
        String eng = s.toLowerCase(Locale.ENGLISH);
        System.out.println(eng);
        String turkish = s.toLowerCase(Locale.forLanguageTag("tr")); // It shows i without dot
        System.out.println(turkish);
    }
}
```

Output:

```
javatpoint hello string
javatpo?nt hello str?ng
```

Java String toUpperCase()

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

Internal implementation

```java
public String toUpperCase(Locale locale) {
      if (locale == null) {
          throw new NullPointerException();
      }

      int firstLower;
      final int len = value.length;

      /* Now check if there are any characters that need to be changed. */
      scan: {
         for (firstLower = 0 ; firstLower < len; ) {
             int c = (int)value[firstLower];
             int srcCount;
             if ((c >= Character.MIN_HIGH_SURROGATE)
                     && (c <= Character.MAX_HIGH_SURROGATE)) {
                 c = codePointAt(firstLower);
                 srcCount = Character.charCount(c);
             } else {
                 srcCount = 1;
             }
             int upperCaseChar = Character.toUpperCaseEx(c);
             if ((upperCaseChar == Character.ERROR)
                     || (c != upperCaseChar)) {
                 break scan;
             }
             firstLower += srcCount;
         }
          return this;
```

```
}

char[] result = new char[len]; /* may grow */
int resultOffset = 0;  /* result may grow, so i+resultOffset
 * is the write location in result */

/* Just copy the first few upperCase characters. */
System.arraycopy(value, 0, result, 0, firstLower);

String lang = locale.getLanguage();
boolean localeDependent =
        (lang == "tr" || lang == "az" || lang == "lt");
char[] upperCharArray;
int upperChar;
int srcChar;
int srcCount;
for (int i = firstLower; i < len; i += srcCount) {
    srcChar = (int)value[i];
    if ((char)srcChar >= Character.MIN_HIGH_SURROGATE &&
        (char)srcChar <= Character.MAX_HIGH_SURROGATE) {
        srcChar = codePointAt(i);
        srcCount = Character.charCount(srcChar);
    } else {
        srcCount = 1;
    }
    if (localeDependent) {
        upperChar = ConditionalSpecialCasing.toUpperCaseEx(this, i, locale);
    } else {
```

```java
            upperChar = Character.toUpperCaseEx(srcChar);
        }
        if ((upperChar == Character.ERROR)
                || (upperChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
            if (upperChar == Character.ERROR) {
                if (localeDependent) {
                    upperCharArray =
                            ConditionalSpecialCasing.toUpperCaseCharArray(this, i, locale);
                } else {
                    upperCharArray = Character.toUpperCaseCharArray(srcChar);
                }
            } else if (srcCount == 2) {
                resultOffset += Character.toChars(upperChar, result, i + resultOffset) - srcCount;
                continue;
            } else {
                upperCharArray = Character.toChars(upperChar);
            }

            /* Grow result if needed */
            int mapLen = upperCharArray.length;
            if (mapLen > srcCount) {
                char[] result2 = new char[result.length + mapLen - srcCount];
                System.arraycopy(result, 0, result2, 0, i + resultOffset);
                result = result2;
            }
```

```
                for (int x = 0; x < mapLen; ++x) {
                    result[i + resultOffset + x] = upperCharArray[x];
                }
                resultOffset += (mapLen - srcCount);
            } else {
                result[i + resultOffset] = (char)upperChar;
            }
        }
        return new String(result, 0, len + resultOffset);
    }
```

Signature

There are two variant of toUpperCase() method. The signature or syntax of string toUpperCase() method is given below:

```
public String toUpperCase()
public String toUpperCase(Locale locale)
```

The second method variant of toUpperCase(), converts all the characters into uppercase using the rules of given Locale.

Returns

string in uppercase letter.

Java String toUpperCase() method example

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

```
String s1="hello string";
String s1upper=s1.toUpperCase();
System.out.println(s1upper);
}}
```

Test it Now

Output:

```
HELLO STRING
```

Java String toUpperCase(Locale locale) Method Example 2

```
import java.util.Locale;
public class StringUpperExample2 {
    public static void main(String[] args) {
        String s = "hello string";
        String turkish = s.toUpperCase(Locale.forLanguageTag("tr"));
        String english = s.toUpperCase(Locale.forLanguageTag("en"));
        System.out.println(turkish);//will print I with dot on upper side
        System.out.println(english);
    }
}
```

Output:

```
HELLO STR?NG
HELLO STRING
```

ava String trim()

The **java string trim()** method eliminates leading and trailing spaces. The unicode value of space character is '\u0020'. The trim() method in java string checks this unicode value before and after the string, if it exists then removes the spaces and returns the omitted string.

The string trim() method doesn't omits middle spaces.

Internal implementation

```java
public String trim() {
        int len = value.length;
        int st = 0;
        char[] val = value;    /* avoid getfield opcode */

        while ((st < len) && (val[st] <= ' ')) {
            st++;
        }
        while ((st < len) && (val[len - 1] <= ' ')) {
            len--;
        }
        return ((st > 0) || (len < value.length)) ? substring(st, len) : this;
    }
```

Signature

The signature or syntax of string trim method is given below:

```java
public String trim()
```

Returns

string with omitted leading and trailing spaces

Java String trim() method example

```java
public class StringTrimExample{
public static void main(String args[]){
String s1="  hello string   ";
System.out.println(s1+"javatpoint");//without trim()
System.out.println(s1.trim()+"javatpoint");//with trim()
}}
```

Test it Now

```
 hello string   javatpoint
hello stringjavatpoint
```

Java String trim() Method Example 2

This example demonstrate the use of trim method. This method removes all the trailing spaces so the length of string also reduces. Let's see an example.

```java
public class StringTrimExample {
    public static void main(String[] args) {
        String s1 ="  hello java string   ";
        System.out.println(s1.length());
        System.out.println(s1); //Without trim()
        String tr = s1.trim();
        System.out.println(tr.length());
        System.out.println(tr); //With trim()
    }
}
```

```
22
  hello java string
```

```
17
hello java string
```

Java String valueOf()

The **java string valueOf()** method converts different types of values into string. By the help of string valueOf() method, you can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to string and char array to string.

Internal implementation

```
public static String valueOf(Object obj) {
       return (obj == null) ? "null" : obj.toString();
   }
```

Signature

The signature or syntax of string valueOf() method is given below:

```
public static String valueOf(boolean b)
public static String valueOf(char c)
public static String valueOf(char[] c)
public static String valueOf(int i)
public static String valueOf(long l)
public static String valueOf(float f)
public static String valueOf(double d)
public static String valueOf(Object o)
```

Returns

string representation of given value

Java String valueOf() method example

```
public class StringValueOfExample{
public static void main(String args[]){
int value=30;
String s1=String.valueOf(value);
System.out.println(s1+10);//concatenating string with 10
}}
```

Test it Now

Output:

```
3010
```

Java String valueOf(boolean bol) Method Example

This is a boolean version of overloaded valueOf() method. It takes boolean value and returns a string. Let's see an example.

```
public class StringValueOfExample2 {
    public static void main(String[] args) {
        // Boolean to String
        boolean bol = true;
        boolean bol2 = false;
        String s1 = String.valueOf(bol);
        String s2 = String.valueOf(bol2);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Test it Now

Output:

```
true

false
```

Java String valueOf(char ch) Method Example

This is a char version of overloaded valueOf() method. It takes char value and returns a string. Let's see an example.

```java
public class StringValueOfExample3 {
    public static void main(String[] args) {
        // char to String
        char ch1 = 'A';
        char ch2 = 'B';
        String s1 = String.valueOf(ch1);
        String s2 = String.valueOf(ch2);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Test it Now

Output:

```
A

B
```

Java String valueOf(float f) and valueOf(double d)

This is a float version of overloaded valueOf() method. It takes float value and returns a string. Let's see an example.

```java
public class StringValueOfExample4 {
    public static void main(String[] args) {
        // Float and Double to String
        float f  = 10.05f;
        double d = 10.02;
        String s1 = String.valueOf(f);
        String s2 = String.valueOf(d);
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Test it Now

Output:

```
10.05
10.02
```

Java String valueOf() Complete Examples

Let's see an example where we are converting all primitives and objects into strings.

```java
public class StringValueOfExample5 {
    public static void main(String[] args) {
        boolean b1=true;
        byte b2=11;
        short sh = 12;
        int i = 13;
        long l = 14L;
        float f = 15.5f;
        double d = 16.5d;
        char chr[]={'j','a','v','a'};
        StringValueOfExample5 obj=new StringValueOfExample5();
        String s1 = String.valueOf(b1);
        String s2 = String.valueOf(b2);
        String s3 = String.valueOf(sh);
        String s4 = String.valueOf(i);
        String s5 = String.valueOf(l);
        String s6 = String.valueOf(f);
        String s7 = String.valueOf(d);
        String s8 = String.valueOf(chr);
        String s9 = String.valueOf(obj);
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s5);
        System.out.println(s6);
        System.out.println(s7);
        System.out.println(s8);
        System.out.println(s9);
    }
}
```

Test it Now

Output:

```
true
11
12
13
14
15.5
16.5
java
StringValueOfExample5@2a139a55
```