Java - String charAt() Method example

The method charAt(int index) returns the character at the specified index. The index value should lie between 0 and length()-1. For e.g. s.charAt(0) would return the first character of the string “s”. It throws IndexOutOfBoundsException if the index is less than zero or greater than equal to the length of the string (index<0|| index>=length()).

Example:

In this example we are fetching few characters of the input string using charAt() method.

```
public class CharAtExample {
    public static void main(String args[]) {
        String str = "Welcome to string handling tutorial";
         char ch1 = str.charAt(0);
         char ch2 = str.charAt(5);
         char ch3 = str.charAt(11);
         char ch4 = str.charAt(20);
         System.out.println("Character at 0 index is: "+ch1);
         System.out.println("Character at 5th index is: "+ch2);
         System.out.println("Character at 11th index is: "+ch3);
         System.out.println("Character at 20th index is: "+ch4);
    }
}
```

Output:

```
Character at 0 index is: W
Character at 5th index is: m
Character at 11th index is: s
Character at 20th index is: n
```

Java - String equals() and equalsIgnoreCase() Methods example

In this tutorial we will discuss equals() and equalsIgnoreCase() methods. Both of these methods are used for comparing two strings. The only difference between them is that the equals() methods considers the case while equalsIgnoreCase() methods ignores the case during comparison. For e.g. The equals() method would return false if we compare the strings “TEXT” and “text” however equalsIgnoreCase() would return true.

boolean equals(String str): Case sensitive comparison
boolean equalsIgnoreCase(String str): Case in-sensitive comparison

Example 1: equals()

```
public class EqualsExample1{
   public static void main(String args[]){
       String str1= new String("Hello");
       String str2= new String("Hi");
       String str3= new String("Hello");
       System.out.println("str1 equals to str2:"+str1.equals(str2));
       System.out.println("str1 equals to str3:"+str1.equals(str3));
       System.out.println("str1 equals to Welcome:"+str1.equals("Welcome"));
       System.out.println("str1 equals to Hello:"+str1.equals("Hello"));
       System.out.println("str1 equals to hello:"+str1.equals("hello"));
   }
}
```

Output:

```
str1 equals to str2:false
str1 equals to str3:true
str1 equals to Welcome:false
str1 equals to Hello:true
str1 equals to hello:false
```

Example2: equalsIgnoreCase()

```
public class EqualsExample2{
   public static void main(String args[]){
       String str1= new String("Apple");
       String str2= new String("MANGO");
       String str3= new String("APPLE");
       System.out.println("str1 equals to str2:"+str1.equalsIgnoreCase(str2));
       System.out.println("str1 equals to str3:"+str1.equalsIgnoreCase(str3));
       System.out.println("str1                     equals                     to
Welcome:"+str1.equalsIgnoreCase("Welcome"));
       System.out.println("str1 equals to Apple:"+str1.equalsIgnoreCase("Apple"));
       System.out.println("str2 equals to mango:"+str2.equalsIgnoreCase("mango"));
   }
}
```

Output:

```
str1 equals to str2:false
str1 equals to str3:true
str1 equals to Welcome:false
str1 equals to Apple:true
str2 equals to mango:true
```

Java - String equals() and equalsIgnoreCase() Methods example

In this tutorial we will discuss `equals()` and `equalsIgnoreCase()` methods. Both of these methods are used for comparing two strings. The only difference between them is that the `equals()` methods considers the case while `equalsIgnoreCase()` methods ignores the case during comparison. For e.g.

The equals() method would return false if we compare the strings “TEXT” and “text” however equalsIgnoreCase() would return true.

boolean equals(String str): Case sensitive comparison
boolean equalsIgnoreCase(String str): Case in-sensitive comparison

Example 1: equals()

```
public class EqualsExample1{
   public static void main(String args[]){
       String str1= new String("Hello");
       String str2= new String("Hi");
       String str3= new String("Hello");
       System.out.println("str1 equals to str2:"+str1.equals(str2));
       System.out.println("str1 equals to str3:"+str1.equals(str3));
       System.out.println("str1 equals to Welcome:"+str1.equals("Welcome"));
       System.out.println("str1 equals to Hello:"+str1.equals("Hello"));
       System.out.println("str1 equals to hello:"+str1.equals("hello"));
   }
}
```

Output:

```
str1 equals to str2:false
str1 equals to str3:true
str1 equals to Welcome:false
str1 equals to Hello:true
str1 equals to hello:false
```

Example2: equalsIgnoreCase()

```
public class EqualsExample2{
   public static void main(String args[]){
       String str1= new String("Apple");
       String str2= new String("MANGO");
       String str3= new String("APPLE");
       System.out.println("str1 equals to str2:"+str1.equalsIgnoreCase(str2));
       System.out.println("str1 equals to str3:"+str1.equalsIgnoreCase(str3));
       System.out.println("str1                    equals                    to
Welcome:"+str1.equalsIgnoreCase("Welcome"));
       System.out.println("str1 equals to Apple:"+str1.equalsIgnoreCase("Apple"));
       System.out.println("str2 equals to mango:"+str2.equalsIgnoreCase("mango"));
   }
}
```

Output:

```
str1 equals to str2:false
str1 equals to str3:true
str1 equals to Welcome:false
str1 equals to Apple:true
str2 equals to mango:true
```

Java - String compareTo() Method example

The method compareTo() is used for comparing two strings lexicographically. Each character of both the strings is converted into a Unicode value for comparison. If both the strings are equal then this method returns 0 else it returns positive or negative value. The result is positive if the first string is lexicographically greater than the second string else the result would be negative.

This method can be used as follows:

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

Here the comparison is between string literals. For e.g. string1.compareTo(string2) where string1 and string2 are String literals.

```
int compareTo(Object obj)
```

This is a comparison between a string and an object. For e.g. string1.compareTo("Just a String object")where string1 is a literal whose value would be compared with the string specified in the method argument.

Example

Here we have three Strings and we are comparing them with each other using compareTo() method.

```
public class CompareToExample {
   public static void main(String args[]) {
       String str1 = "String method tutorial";
       String str2 = "compareTo method example";
       String str3 = "String method tutorial";

       int var1 = str1.compareTo( str2 );
       System.out.println("str1 & str2 comparison: "+var1);

       int var2 = str1.compareTo( str3 );
       System.out.println("str1 & str3 comparison: "+var2);

       int var3 = str2.compareTo("compareTo method example");
       System.out.println("str2 & string argument comparison: "+var3);
   }
}
```

Output:

```
str1 & str2 comparison: -16
str1 & str3 comparison: 0
str2 & string argument comparison: 0
```

Java - String compareToIgnoreCase() Method example

The method compareToIgnoreCase() is similar to the compareTo() method as it also compares two strings lexicographically. The only difference between them is that compareToIgnoreCase() ignores the case (uppercase or lowercase) while comparing two strings. Similar to compareTo() this method also compares the strings based on the Unicode value of their each character. It returns 0 when the strings are equal else it returns positive or negative value.

```
int compareToIgnoreCase(String str)
```

Example

In the below example we have three String literals which have same value but the case is different. string1 is in uppercase, string2 is in lowercase and string3 is a mix of uppercase and lowercase. We are using compareToIgnoreCase() method to compare these strings.

```
public class CompareExample {
   public static void main(String args[]) {
       String string1 = "HELLO";
       String string2 = "hello";
       String string3 = "Hello";

       int var1 = string1.compareToIgnoreCase(string2);
       System.out.println("string1 and string2 comparison: "+var1);

       int var2 = string1.compareToIgnoreCase(string3);
       System.out.println("string1 and string3 comparison: "+var2);

       int var3 = string1.compareToIgnoreCase("HeLLo");
       System.out.println("string1 and HeLLo comparison: "+var3);
   }
}
```

Output:

```
string1 and string2 comparison: 0
string1 and string3 comparison: 0
string1 and HeLLo comparison: 0
```

Java - String startsWith() Method example

The method startsWith() is used for checking prefix of a String. It returns a boolean value true or false based on whether the specified string is prefix of the particular String or not.
boolean startsWith(String str): It returns true if the str is a prefix of the String.

boolean startsWith(String str, index fromIndex): It returns true if the String begins with str, it starts looking from the specified index “fromIndex”.

Example

```
public class StartsWithExample{
   public static void main(String args[]) {
       String str= new String("quick brown fox jumps over the lazy dog");
       System.out.println("String       str       starts       with       quick: 
"+str.startsWith("quick"));
       System.out.println("String       str       starts       with       brown: 
"+str.startsWith("brown"));
       System.out.println("substring  of  str(starting  from  6th  index)  has  brown 
prefix: "
+str.startsWith("brown", 6));
       System.out.println("substring  of  str(starting  from  6th  index)  has  quick 
prefix: "
+str.startsWith("quick", 6));

   }
}
```

Output:

```
String str starts with quick: true
String str starts with brown: false
substring of str(starting from 6th index) havs brown prefix: true
substring of str(starting from 6th index) havs quick prefix: false
```

Java - String startsWith() Method example

The method startsWith() is used for checking prefix of a String. It returns a boolean value true or false based on whether the specified string is prefix of the particular String or not.
boolean startsWith(String str): It returns true if the str is a prefix of the String.
boolean startsWith(String str, index fromIndex): It returns true if the String begins with str, it starts looking from the specified index “fromIndex”.

Example

```
public class StartsWithExample{
   public static void main(String args[]) {
       String str= new String("quick brown fox jumps over the lazy dog");
       System.out.println("String       str       starts       with       quick: 
"+str.startsWith("quick"));
       System.out.println("String       str       starts       with       brown: 
"+str.startsWith("brown"));
       System.out.println("substring  of  str(starting  from  6th  index)  has  brown 
prefix: "
+str.startsWith("brown", 6));
       System.out.println("substring  of  str(starting  from  6th  index)  has  quick 
prefix: "
+str.startsWith("quick", 6));

   }
}
```
Output:

```
String str starts with quick: true
String str starts with brown: false
substring of str(starting from 6th index) havs brown prefix: true
substring of str(starting from 6th index) havs quick prefix: false
```

Java - String endsWith() Method example

The method `endsWith(String suffix)` checks whether the `String` ends with a specified suffix. This method returns a boolean value true or false. If the specified suffix is found at the end of the string then it returns true else false.

`public boolean endsWith(String suffix)`

Example

In this example we have two strings and we are checking whether the strings are ending with the specified suffixes.

```
public class EndsWithExample{
   public static void main(String args[]){
       String str1 = new String("This is a test String");
       String str2 = new String("Test ABC");
       boolean var1 = str1.endsWith("String");
       boolean var2 = str1.endsWith("ABC");
       boolean var3 = str2.endsWith("String");
       boolean var4 = str2.endsWith("ABC");
       System.out.println("str1 ends with String: "+ var1);
       System.out.println("str1 ends with ABC: "+ var2);
       System.out.println("str2 ends with String: "+ var3);
       System.out.println("str2 ends with ABC: "+ var4);
   }
}
```
Output:

```
str1 ends with String: true
str1 ends with ABC: false
str2 ends with String: false
str2 ends with ABC: true
```

Java - String trim() and hashCode() Methods

In this tutorial we will discuss about trim() and hashCode() methods. Lets discuss about trim() first: It returns a String after removing leading and trailing white spaces from the input String. For e.g. "   Hello".trim() would return the String "Hello".

```
public String trim()
```

Example: trim() method

```
public class TrimExample{
   public static void main(String args[]){
       String str = new String("    How are you??   ");
       System.out.println("String before trim: "+str);
       System.out.println("String after trim: "+str.trim());
   }
}
```

Output:

```
String before trim:     How are you??
String after trim: How are you??
```

hashCode() method

This method returns the hash code for the String. The computation is done like this:

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

Syntax of this method:

```
public int hashCode()
```

Example: hashCode() method

```
public class HashCodeExample{
   public static void main(String args[]){
       String str = new String("Welcome!!");
       System.out.println("Hash Code for String str: "+str.hashCode());
   }
}
```

Output:

```
Hash Code for String str: 1601728226
```

Java - String indexOf() Method example

The method indexOf() is used for finding out the index of the specified character or substring in a particular String. There are 4 variations of this method:
int indexOf(int ch): It returns the index of the first occurrence of character ch in a String.
int indexOf(int ch, int fromIndex): It returns the index of first occurrence if character ch, starting from the specified index "fromIndex".

int indexOf(String str): Returns the index of string str in a particular String. int indexOf(String str, int fromIndex): Returns the index of string str, starting from the specified index “fromIndex”.

All the above variations returns -1 if the specified char/substring is not found in the particular String.

Example

```
public class IndexOfExample{
   public static void main(String args[]) {
       String str1 = new String("This is a BeginnersBook tutorial");
       String str2 = new String("Beginners");
       String str3 = new String("Book");
       String str4 = new String("Books");
       System.out.println("Index of B in str1: "+str1.indexOf('B'));
       System.out.println("Index of B in str1 after 15th char:"+str1.indexOf('B',
15));
       System.out.println("Index of B in str1 after 30th char:"+str1.indexOf('B',
30));
       System.out.println("Index of string str2 in str1:"+str1.indexOf(str2));
       System.out.println("Index of str2 after 15th char"+str1.indexOf(str2, 15));
       System.out.println("Index of string str3:"+str1.indexOf(str3));
       System.out.println("Index of string str4"+str1.indexOf(str4));
       System.out.println("Index of harcoded string:"+str1.indexOf("is"));
       System.out.println("Index    of    hardcoded    string    after    4th
char:"+str1.indexOf("is", 4));
   }
}
```

Output:

```
Index of B in str1: 10
Index of B in str1 after 15th char:19
Index of B in str1 after 30th char:-1
Index of string str2 in str1:10
Index of str2 after 15th char-1
Index of string str3:19
Index of string str4-1
Index of harcoded string:2
Index of hardcoded string after 4th char:5
```

Java - String indexOf() Method example

The method indexOf() is used for finding out the index of the specified character or substring in a particular String. There are 4 variations of this method: int indexOf(int ch): It returns the index of the first occurrence of character ch in a String.
int indexOf(int ch, int fromIndex): It returns the index of first occurrence if character ch, starting from the specified index “fromIndex”. int indexOf(String str): Returns the index of string str in a particular String.

int indexOf(String str, int fromIndex): Returns the index of string str, starting from the specified index “fromIndex”.

All the above variations returns -1 if the specified char/substring is not found in the particular String.

Example

```
public class IndexOfExample{
   public static void main(String args[]) {
       String str1 = new String("This is a BeginnersBook tutorial");
       String str2 = new String("Beginners");
       String str3 = new String("Book");
       String str4 = new String("Books");
       System.out.println("Index of B in str1: "+str1.indexOf('B'));
       System.out.println("Index of B in str1 after 15th char:"+str1.indexOf('B',
15));
       System.out.println("Index of B in str1 after 30th char:"+str1.indexOf('B',
30));
       System.out.println("Index of string str2 in str1:"+str1.indexOf(str2));
       System.out.println("Index of str2 after 15th char"+str1.indexOf(str2, 15));
       System.out.println("Index of string str3:"+str1.indexOf(str3));
       System.out.println("Index of string str4"+str1.indexOf(str4));
       System.out.println("Index of harcoded string:"+str1.indexOf("is"));
       System.out.println("Index     of     hardcoded     string     after     4th
char:"+str1.indexOf("is", 4));
   }
}
```

Output:

```
Index of B in str1: 10
Index of B in str1 after 15th char:19
Index of B in str1 after 30th char:-1
Index of string str2 in str1:10
Index of str2 after 15th char-1
Index of string str3:19
Index of string str4-1
Index of harcoded string:2
Index of hardcoded string after 4th char:5
```

Java - String lastIndexOf() Method example

In the last tutorial we discussed indexOf() method. In this tutorial we will discuss lastIndexOf()method which is used for finding out the index of last occurrence of a char/sub-string in a particular String.
Variations:
int lastIndexOf(int ch): It returns the last occurrence of character ch in the particular String.
int lastIndexOf(int ch, int fromIndex): It returns the last occurrence of ch, starting searching backward from the specified index “fromIndex”.

int lastIndexOf(String str): Returns the last occurrence of str in a String.
int lastIndexOf(String str, int fromIndex): Returns the last occurrence of str, starting searching backward from the specified index “fromIndex”.

Example of lastIndexOf() method

```
public class LastIndexOfExample{
   public static void main(String args[]) {
       String str1 = new String("This is a BeginnersBook tutorial");
       String str2 = new String("Beginners");
       String str3 = new String("Book");
       String str4 = new String("Books");
       System.out.println("Last 'B' in str1: "+str1.lastIndexOf('B'));
       System.out.println("Last 'B' in str1 whose index<=15:"+str1.lastIndexOf('B',
15));
       System.out.println("Last 'B' in str1 whose index<=30:"+str1.lastIndexOf('B',
30));
       System.out.println("Last        occurrence        of        str2        in
str1:"+str1.lastIndexOf(str2));
       System.out.println("Last    occurrence    of    str2    in    str1    before
15:"+str1.lastIndexOf(str2, 15));
       System.out.println("Last        occurrence        of        str3        in
str1:"+str1.lastIndexOf(str3));
       System.out.println("Last        occurrence        of        str4        in
str1"+str1.lastIndexOf(str4));
       System.out.println("Last        occurrence        of        'is'        in
str1:"+str1.lastIndexOf("is"));
       System.out.println("Last    occurrence    of    'is'    in    str1    before
4:"+str1.lastIndexOf("is", 4));
   }
}
```

Output:

```
Last 'B' in str1: 19
Last 'B' in str1 whose index<=15:10
Last 'B' in str1 whose index<=30:19
Last occurrence of str2 in str1:10
Last occurrence of str2 in str1 before 15:10
Last occurrence of str3 in str1:19
Last occurrence of str4 in str1-1
Last occurrence of 'is' in str1:5
Last occurrence of 'is' in str1 before 4:2
```

Java - String lastIndexOf() Method example

BY CHAITANYA SINGH | FILED UNDER: STRING HANDLING

In the last tutorial we discussed indexOf() method. In this tutorial we will discuss lastIndexOf()method which is used for finding out the index of last occurrence of a char/sub-string in a particular String.
Variations:
int lastIndexOf(int ch): It returns the last occurrence of character ch in the particular

String.
int lastIndexOf(int ch, int fromIndex): It returns the last occurrence of ch, starting searching backward from the specified index “fromIndex”.
int lastIndexOf(String str): Returns the last occurrence of str in a String.
int lastIndexOf(String str, int fromIndex): Returns the last occurrence of str, starting searching backward from the specified index “fromIndex”.

Example of lastIndexOf() method

```
public class LastIndexOfExample{
   public static void main(String args[]) {
       String str1 = new String("This is a BeginnersBook tutorial");
       String str2 = new String("Beginners");
       String str3 = new String("Book");
       String str4 = new String("Books");
       System.out.println("Last 'B' in str1: "+str1.lastIndexOf('B'));
       System.out.println("Last 'B' in str1 whose index<=15:"+str1.lastIndexOf('B',
15));
       System.out.println("Last 'B' in str1 whose index<=30:"+str1.lastIndexOf('B',
30));
       System.out.println("Last        occurrence        of        str2        in
str1:"+str1.lastIndexOf(str2));
       System.out.println("Last    occurrence    of    str2    in    str1    before
15:"+str1.lastIndexOf(str2, 15));
       System.out.println("Last        occurrence        of        str3        in
str1:"+str1.lastIndexOf(str3));
       System.out.println("Last        occurrence        of        str4        in
str1"+str1.lastIndexOf(str4));
       System.out.println("Last        occurrence        of        'is'        in
str1:"+str1.lastIndexOf("is"));
       System.out.println("Last    occurrence    of    'is'    in    str1    before
4:"+str1.lastIndexOf("is", 4));
   }
}
```

Output:

```
Last 'B' in str1: 19
Last 'B' in str1 whose index<=15:10
Last 'B' in str1 whose index<=30:19
Last occurrence of str2 in str1:10
Last occurrence of str2 in str1 before 15:10
Last occurrence of str3 in str1:19
Last occurrence of str4 in str1-1
Last occurrence of 'is' in str1:5
Last occurrence of 'is' in str1 before 4:2
```

Java - String indexOf() Method example

The method indexOf() is used for finding out the index of the specified character or substring in a particular String. There are 4 variations of this method: int indexOf(int ch): It returns the index of the first occurrence of character ch in a String.
int indexOf(int ch, int fromIndex): It returns the index of first occurrence if character ch, starting from the specified index "fromIndex". int indexOf(String str): Returns the index of string str in a particular String. int indexOf(String str, int fromIndex): Returns the index of string str, starting from the specified index "fromIndex".

All the above variations returns -1 if the specified char/substring is not found in the particular String.

Example

```
public class IndexOfExample{
   public static void main(String args[]) {
       String str1 = new String("This is a BeginnersBook tutorial");
       String str2 = new String("Beginners");
       String str3 = new String("Book");
       String str4 = new String("Books");
       System.out.println("Index of B in str1: "+str1.indexOf('B'));
       System.out.println("Index of B in str1 after 15th char:"+str1.indexOf('B', 
15));
       System.out.println("Index of B in str1 after 30th char:"+str1.indexOf('B', 
30));
       System.out.println("Index of string str2 in str1:"+str1.indexOf(str2));
       System.out.println("Index of str2 after 15th char"+str1.indexOf(str2, 15));
       System.out.println("Index of string str3:"+str1.indexOf(str3));
       System.out.println("Index of string str4"+str1.indexOf(str4));
       System.out.println("Index of harcoded string:"+str1.indexOf("is"));
       System.out.println("Index of hardcoded string after 4th 
char:"+str1.indexOf("is", 4));
   }
}
```

Output:

```
Index of B in str1: 10
Index of B in str1 after 15th char:19
Index of B in str1 after 30th char:-1
Index of string str2 in str1:10
Index of str2 after 15th char-1
Index of string str3:19
Index of string str4-1
Index of harcoded string:2
Index of hardcoded string after 4th char:5
```

Java - String lastIndexOf() Method example

In the last tutorial we discussed indexOf() method. In this tutorial we will discuss lastIndexOf()method which is used for finding out the index of last occurrence of a char/sub-string in a particular String.
Variations:
int lastIndexOf(int ch): It returns the last occurrence of character ch in the particular String.
int lastIndexOf(int ch, int fromIndex): It returns the last occurrence of ch, starting searching backward from the specified index "fromIndex".
int lastIndexOf(String str): Returns the last occurrence of str in a String.
int lastIndexOf(String str, int fromIndex): Returns the last occurrence of str, starting searching backward from the specified index "fromIndex".

Example of lastIndexOf() method

```
public class LastIndexOfExample{
   public static void main(String args[]) {
       String str1 = new String("This is a BeginnersBook tutorial");
       String str2 = new String("Beginners");
       String str3 = new String("Book");
       String str4 = new String("Books");
       System.out.println("Last 'B' in str1: "+str1.lastIndexOf('B'));
       System.out.println("Last 'B' in str1 whose index<=15:"+str1.lastIndexOf('B',
15));
       System.out.println("Last 'B' in str1 whose index<=30:"+str1.lastIndexOf('B',
30));
       System.out.println("Last        occurrence        of        str2        in
str1:"+str1.lastIndexOf(str2));
       System.out.println("Last    occurrence    of    str2    in    str1    before
15:"+str1.lastIndexOf(str2, 15));
       System.out.println("Last        occurrence        of        str3        in
str1:"+str1.lastIndexOf(str3));
       System.out.println("Last        occurrence        of        str4        in
str1"+str1.lastIndexOf(str4));
       System.out.println("Last        occurrence        of        'is'        in
str1:"+str1.lastIndexOf("is"));
       System.out.println("Last    occurrence    of    'is'    in    str1    before
4:"+str1.lastIndexOf("is", 4));
   }
}
```

Output:

```
Last 'B' in str1: 19
Last 'B' in str1 whose index<=15:10
Last 'B' in str1 whose index<=30:19
Last occurrence of str2 in str1:10
Last occurrence of str2 in str1 before 15:10
Last occurrence of str3 in str1:19
Last occurrence of str4 in str1-1
Last occurrence of 'is' in str1:5
Last occurrence of 'is' in str1 before 4:2
```

Java - String substring() Method example

Method substring() is used for getting a substring of a particular String. There are two variants of this method:

String substring(int beginIndex): Returns the substring starting from the specified index(beginIndex) till the end of the string. For e.g. "Chaitanya".substring(2) would return "aitanya". This method throws IndexOutOfBoundsException If the beginIndex is less than zero or greater than the length of String (beginIndex<0||> length of String).

String substring(int beginIndex, int endIndex): Returns the substring starting from the given index(beginIndex) till the specified index(endIndex). For e.g. "Chaitanya".substring(2,5) would return "ait". It throws IndexOutOfBoundsException If the beginIndex is less than zero OR beginIndex > endIndex OR endIndex is greater than the length of String.

Example: substring()

```
public class SubStringExample{
   public static void main(String args[]) {
       String str= new String("quick brown fox jumps over the lazy dog");
       System.out.println("Substring starting from index 15:");
       System.out.println(str.substring(15));
       System.out.println("Substring starting from index 15 and ending at 20:");
       System.out.println(str.substring(15, 20));
   }
}
```

Output:

```
Substring starting from index 15:
 jumps over the lazy dog
Substring starting from index 15 and ending at 20:
 jump
```

Java - String substring() Method example

Method substring() is used for getting a substring of a particular String. There are two variants of this method:

String substring(int beginIndex): Returns the substring starting from the specified index(beginIndex) till the end of the string. For e.g. "Chaitanya".substring(2) would return "aitanya". This method throws IndexOutOfBoundsException If the beginIndex is less than zero or greater than the length of String (beginIndex<0||> length of String).

String substring(int beginIndex, int endIndex): Returns the substring starting from the given index(beginIndex) till the specified index(endIndex). For e.g. "Chaitanya".substring(2,5) would return "ait". It throws IndexOutOfBoundsException If the beginIndex is less than zero OR beginIndex > endIndex OR endIndex is greater than the length of String.

Example: substring()

```
public class SubStringExample{
   public static void main(String args[]) {
       String str= new String("quick brown fox jumps over the lazy dog");
       System.out.println("Substring starting from index 15:");
       System.out.println(str.substring(15));
       System.out.println("Substring starting from index 15 and ending at 20:");
       System.out.println(str.substring(15, 20));
   }
}
```

Output:

```
Substring starting from index 15:
 jumps over the lazy dog
Substring starting from index 15 and ending at 20:
 jump
```

Java - String concat() Method example

In this tutorial we will learn how to concatenate multiple strings using concat() method in Java.

```
public String concat(String str)
```

This method concatenates the string str at the end of the current string. For e.g. s1.concat("Hello");would concatenate the String “Hello” at the end of the String s1. This method can be called multiple times in a single statement like this

```
String s1="Beginners";
s1= s1.concat("Book").concat(".").concat("com");
```

The value of s1 would be BeginnersBook.com after the execution of above statement.

Complete Example

In this example we have shared two ways for String concatenation.

```
public class ConcatenationExample {
   public static void main(String args[]) {
       //One way of doing concatenation
       String str1 = "Welcome";
       str1 = str1.concat(" to ");
       str1 = str1.concat(" String handling ");
```

```
        System.out.println(str1);

        //Other way of doing concatenation in one line
        String str2 = "This";
        str2 = str2.concat(" is").concat(" just a").concat(" String");
        System.out.println(str2);
    }
}
```
Output:

```
Welcome to  String handling
This is just a String
```

Java - String replace(), replaceFirst() and replaceAll() method

In this tutorial we will discuss `replace()`, `replaceFirst()`and `replaceAll()` methods. All of these methods are mainly used for replacing a part of String with another String.
`String replace(char oldChar, char newChar)`: It replaces all the oldChar characters with newChar characters. For e.g. `"pog pance".replace('p', 'd')` would return `dog dance`.
`String replaceFirst(String regex, String replacement)`: It replaces the first substring that fits the specified regular expression with the replacement String. `PatternSyntaxException` if the specified regular expression(regex) is not valid.
`String replaceAll(String regex, String replacement)`: It replaces all the substrings that fits the given regular expression with the replacement String.

Example 1:replace() Method

```
public class Example1{
   public static void main(String args[]){
       String str = new String("Site is BeginnersBook.com");

       System.out.print("String after replacing all 'o' with 'p' :" );
       System.out.println(str.replace('o', 'p'));

       System.out.print("String after replacing all 'i' with 'K' :" );
       System.out.println(str.replace('i', 'K'));
   }
}
```
Output:

```
String after replacing all 'o' with 'p' :Site is BeginnersBppk.cpm
String after replacing all 'i' with 'K' :SKte Ks BegKnnersBook.com
```

Example 2:replaceFirst() Method

```
public class Example2{
   public static void main(String args[]){
       String str = new String("Site is BeginnersBook.com");
```

```java
        System.out.print("String after replacing com with net :" );
        System.out.println(str.replaceFirst("com", "net"));

        System.out.print("String after replacing Site name:" );
        System.out.println(str.replaceFirst("Beginners(.*)", "XYZ.com"));
   }
}
```

Output:

```
String after replacing com with net :Site is BeginnersBook.net
String after replacing Site name:Site is XYZ.com
```

Example 3:replaceAll() Method

```java
public class Example3{
   public static void main(String args[]){
        String str = new String("My .com site is BeginnersBook.com");
        System.out.print("String after replacing all com with net :" );
        System.out.println(str.replaceAll("com", "net"));
   }
}
```

Output:

```
String after replacing all com with net :My .net site is BeginnersBook.net
```

Java String contains() method explained with examples

Java String contains() method checks whether a particular string is a part of another string or not. This method returns true if a specified sequence of characters is present in a given string, else it returns false.

For example:

```java
String str = "Game of Thrones";

//This will print "true" because "Game" is present in the given String
System.out.println(str.contains("Game"));

/* This will print "false" because "aGme" is not present, the characters
 * must be present in the same sequence as specified in the contains method
 */
System.out.println(str.contains("aGme"));
```

Syntax of contains() method

```java
public boolean contains(CharSequence str)
```

The return type is boolean which means this method returns true or false. When the character sequence found in the given string then this method returns true else it returns false.

If the CharSequence is null then this method throws NullPointerException. For example: calling this method like this would throw NullPointerException.

```
str.contains(null);
```

Java String contains() method Example

The second print statement displayed false because the contains() method is case sensitive. **You can also use the contains() method for case insensitive check**, I have covered this **at the end of this tutorial**.

```
class Example{
   public static void main(String args[]){
        String str = "Do you like watching Game of Thrones";
        System.out.println(str.contains("like"));

        /* this will print false as the contains() method is
         * case sensitive. Here we have mentioned letter "l"
         * in upper case and in the actual string we have this
         * letter in the lower case.
         */
        System.out.println(str.contains("Like"));
        System.out.println(str.contains("Game"));
        System.out.println(str.contains("Game of"));
   }
}
```

Output:

```
true
false
true
true
```

Example 2: Using Java String contains() method in the if-else statement

```
class Example{
   public static void main(String args[]){
        String str = "This is an example of contains()";
        /* Using the contains() method in the if-else statements, since
         * this method returns the boolean value, it can be used
         * in the if else conditions whenever needed.
         */
        if(str.contains("example")){
           System.out.println("The word example is found in given string");
        }
        else{
           System.out.println("The word example is not found in the string");
        }
   }
}
```

Output:

```
The word example is found in given string
```

Java String contains() method for case insensitive check

We have seen above that the contains() method is case sensitive, however with a little trick, you can use this method for case insensitive checks. Lets take an example to understand this:

Here we are using the toLowerCase() method to convert both the strings to lowercase so that we can perform case insensitive check using contains() method. We can also use toUpperCase() method for this same purpose as shown in the example below.

```
class Example{
  public static void main(String args[]){
        String str = "Just a Simple STRING";
        String str2 = "string";
        //Converting both the strings to lower case for case insensitive checking
        System.out.println(str.toLowerCase().contains(str2.toLowerCase()));

        //You can also use the upper case method for the same purpose.
        System.out.println(str.toUpperCase().contains(str2.toUpperCase()));
  }
}
```

Output:

```
true
true
```

Java - String toLowerCase() and toUpperCase() Methods

The method `toLowerCase()` converts the characters of a `String` into lower case characters. It has two variants:

`String toLowerCase(Locale locale)`: It converts the string into Lowercase using the rules defined by specified Locale.

`String toLowerCase()`: It is equivalent to `toLowerCase(Locale.getDefault())`. `Locale.getDefault()` gets the current value of the default locale for this instance of the Java Virtual Machine. The Java Virtual Machine sets the default locale during startup based on the host environment. It is used by many locale-sensitive methods if no locale is explicitly specified. It can be changed using the `setDefault()` method.

Example: toLowerCase() method

```
import java.util.Locale;
public class LowerCaseExample{
   public static void main(String args[]){
       String str = new String("ABC IS NOT EQUAL TO XYZ");
       //Standard method of conversion
       System.out.println(str.toLowerCase());

       //By specifying Locale
       System.out.println(str.toLowerCase(Locale.FRANCE));
   }
}
```

Output:

```
abc is not equal to xyz
abc is not equal to xyz
```

Method: toUpperCase()

Like `toLowerCase()` method, `toUpperCase()` also has two variants:
`String toUpperCase(Locale locale)`: It converts the string into a UpperCase string using the rules defined by the specified Locale.
`String toUpperCase()`: It is equavalent to `toUpperCase(Locale.getDefault())`.

Example: toUpperCase() method

```
import java.util.Locale;
public class UpperCaseExample{
   public static void main(String args[]){
       String str = new String("this is a test string");
       //Standard method of conversion
       System.out.println(str.toUpperCase());

       //By specifying Locale
       System.out.println(str.toUpperCase(Locale.CHINA));
   }
}
```

Output:

```
THIS IS A TEST STRING
THIS IS A TEST STRING
```

Java - String toLowerCase() and toUpperCase() Methods

The method `toLowerCase()` converts the characters of a `String` into lower case characters. It has two variants:

String toLowerCase(Locale locale): It converts the string into Lowercase using the rules defined by specified Locale.

String toLowerCase(): It is equivalent to toLowerCase(Locale.getDefault()). Locale.getDefault() gets the current value of the default locale for this instance of the Java Virtual Machine. The Java Virtual Machine sets the default locale during startup based on the host environment. It is used by many locale-sensitive methods if no locale is explicitly specified. It can be changed using the setDefault() method.

Example: toLowerCase() method

```
import java.util.Locale;
public class LowerCaseExample{
   public static void main(String args[]){
       String str = new String("ABC IS NOT EQUAL TO XYZ");
       //Standard method of conversion
       System.out.println(str.toLowerCase());

       //By specifying Locale
       System.out.println(str.toLowerCase(Locale.FRANCE));
   }
}
```

Output:

```
abc is not equal to xyz
abc is not equal to xyz
```

Method: toUpperCase()

Like toLowerCase() method, toUpperCase() also has two variants:
String toUpperCase(Locale locale): It converts the string into a UpperCase string using the rules defined by the specified Locale.
String toUpperCase(): It is equavalent to toUpperCase(Locale.getDefault()).

Example: toUpperCase() method

```
import java.util.Locale;
public class UpperCaseExample{
   public static void main(String args[]){
       String str = new String("this is a test string");
       //Standard method of conversion
       System.out.println(str.toUpperCase());

       //By specifying Locale
       System.out.println(str.toUpperCase(Locale.CHINA));
   }
}
```

Output:

```
THIS IS A TEST STRING
THIS IS A TEST STRING
```

Java String intern() method explained with examples

Java String intern() method is used for getting the string from the memory if it is already present. This method ensures that all the **same strings** share the same memory. For example, creating a string “hello” 10 times using intern() method would ensure that there will be only one instance of “Hello” in the memory and all the 10 references point to the same instance.

A Simple Java String intern() method example

This example demonstrates the use of intern() method. This method searches the memory pool for the mentioned String, if the string is found then it returns the reference of it, else it allocates a new memory space for the string and assign a reference to it.

```
public class Example{
   public static void main(String args[]){
        String str1 = "beginnersbook";

        /* The Java String intern() method searches the string "beginnersbook"
         * in the memory pool and returns the reference of it.
         */
        String str2 = new String("beginnersbook").intern();
        //prints true
        System.out.println("str1==str2: "+(str1==str2));
   }
}
```

Output:

```
str1==str2: true
```

Java automatically interns String literals

When we create Strings using string literals instead of creating them using new keyword then the java automatically interns String them. Lets take an example to understand this:

```
public class Example{
   public static void main(String args[]){
        String str1 = "Hello";

        //Java automatically interns this
        String str2 = "Hello";
        //This is same as creating string using string literal
        String str3 = "Hello".intern();

        //This will create a new instance of "Hello" in memory
        String str4 = new String("Hello");



        if ( str1 == str2 ){
            System.out.println("String str1 and str2 are same");
        }

        if ( str2 == str3 ){
            System.out.println("String str2 and str3 are same" );
        }

        if ( str1 == str4 ){
            //This will not be printed as the condition is not true
            System.out.println("String str1 and str4 are same" );
        }

        if ( str3 == str5 ){
            System.out.println("String str3 and str5 are same" );
        }
   }
}
```
Output:

```
String str1 and str2 are same
String str2 and str3 are same
String str3 and str5 are same
```

Java String isEmpty() method with example

Java String isEmpty() method checks whether a String is empty or not. This method returns true if the given string is empty, else it returns false. In other words you can say that this method returns true if the length of the string is 0.
Signature of isEmpty() method:

```
public boolean isEmpty()
```

Java String isEmpty() method Example

```
public class Example{
   public static void main(String args[]){
        //empty string
        String str1="";
        //non-empty string
        String str2="hello";

        //prints true
        System.out.println(str1.isEmpty());
        //prints false
        System.out.println(str2.isEmpty());
   }
}
```
Output:

```
true
false
```

Java String isEmpty() method Example to check if string is null or empty

As we have seen in the above example that isEmpty() method only checks whether the String is empty or not. If you want to check whether the String is null or empty both then you can do it as shown in the following example.

```
public class Example{
   public static void main(String args[]){
        String str1 = null;
        String str2 = "beginnersbook";

        if(str1 == null || str1.isEmpty()){
           System.out.println("String str1 is empty or null");
        }
        else{
           System.out.println(str1);
        }
        if(str2 == null || str2.isEmpty()){
           System.out.println("String str2 is empty or null");
        }
        else{
           System.out.println(str2);
        }
   }
}
```
Output:

```
String str1 is empty or null
beginnersbook
```

Java String join() method explained with examples

In Java 8 we have a new Method join() in the Java String class. Java String join() method concatenates the given Strings and returns the concatenated String. Java 8 also introduced a new StringJoiner classfor the same purpose.

Java String Join() method Signature

```
public static String join(CharSequence delimiter,
                          CharSequence... elements)
```

Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
For example,

```
String message = String.join("-", "This", "is", "a", "String");
// message returned is: "This-is-a-String"
```

The first argument of this method specifies the delimiter that is used to join multiple strings.
Note that if an element is null, then “null” is added.

Java String join() example

```
public class Example{
   public static void main(String args[]){
        //The first argument to this method is the delimiter
        String str=String.join("^","You","are","Awesome");
        System.out.println(str);
   }
}
```

Output:

```
You^are^Awesome
```

Java String join() Example to join list elements by a delimiter

In this example we are joining the elements of a List by a delimiter using join() method.

```
import java.util.List;
import java.util.Arrays;
public class Example{
   public static void main(String args[]){
        //Converting an array of String to the list
        List list<String> = Arrays.asList("Steve", "Rick", "Peter", "Abbey");
        String names = String.join(" | ", list);
        System.out.println(names);
   }
}
```

Output:

```
Steve | Rick | Peter | Abbey
```

Java - String replace(), replaceFirst() and replaceAll() method

In this tutorial we will discuss replace(), replaceFirst()and replaceAll() methods. All of these methods are mainly used for replacing a part of String with another String. String replace(char oldChar, char newChar): It replaces all the oldChar characters with newChar characters. For e.g. "pog pance".replace('p', 'd') would return dog dance. String replaceFirst(String regex, String replacement): It replaces the first substring that fits the specified regular expression with the replacement String. PatternSyntaxException if the specified regular expression(regex) is not valid. String replaceAll(String regex, String replacement): It replaces all the substrings that fits the given regular expression with the replacement String.

Example 1:replace() Method

```
public class Example1{
   public static void main(String args[]){
       String str = new String("Site is BeginnersBook.com");

       System.out.print("String after replacing all 'o' with 'p' :" );
       System.out.println(str.replace('o', 'p'));

       System.out.print("String after replacing all 'i' with 'K' :" );
       System.out.println(str.replace('i', 'K'));
   }
}
```

Output:

```
String after replacing all 'o' with 'p' :Site is BeginnersBppk.cpm
String after replacing all 'i' with 'K' :SKte Ks BegKnnersBook.com
```

Example 2:replaceFirst() Method

```
public class Example2{
   public static void main(String args[]){
       String str = new String("Site is BeginnersBook.com");

       System.out.print("String after replacing com with net :" );
       System.out.println(str.replaceFirst("com", "net"));

       System.out.print("String after replacing Site name:" );
       System.out.println(str.replaceFirst("Beginners(.*)", "XYZ.com"));
   }
}
```

Output:

```
String after replacing com with net :Site is BeginnersBook.net
String after replacing Site name:Site is XYZ.com
```

Example 3:replaceAll() Method

```
public class Example3{
   public static void main(String args[]){
       String str = new String("My .com site is BeginnersBook.com");
       System.out.print("String after replacing all com with net :" );
       System.out.println(str.replaceAll("com", "net"));
   }
}
```

Output:

```
String after replacing all com with net :My .net site is BeginnersBook.net
```

Java - String replace(), replaceFirst() and replaceAll() method

In this tutorial we will discuss `replace()`, `replaceFirst()`and `replaceAll()` methods. All of these methods are mainly used for replacing a part of String with another String.
`String replace(char oldChar, char newChar)`: It replaces all the oldChar characters with newChar characters. For e.g. `"pog pance".replace('p', 'd')` would return `dog dance`.
`String replaceFirst(String regex, String replacement)`: It replaces the first substring that fits the specified regular expression with the replacement String. PatternSyntaxException if the specified regular expression(regex) is not valid.
`String replaceAll(String regex, String replacement)`: It replaces all the substrings that fits the given regular expression with the replacement String.

Example 1:replace() Method

```
public class Example1{
   public static void main(String args[]){
       String str = new String("Site is BeginnersBook.com");

       System.out.print("String after replacing all 'o' with 'p' :" );
       System.out.println(str.replace('o', 'p'));

       System.out.print("String after replacing all 'i' with 'K' :" );
       System.out.println(str.replace('i', 'K'));
   }
}
```

Output:

```
String after replacing all 'o' with 'p' :Site is BeginnersBppk.cpm
String after replacing all 'i' with 'K' :SKte Ks BegKnnersBook.com
```

Example 2:replaceFirst() Method

```
public class Example2{
   public static void main(String args[]){
       String str = new String("Site is BeginnersBook.com");

       System.out.print("String after replacing com with net :" );
       System.out.println(str.replaceFirst("com", "net"));
```

```
        System.out.print("String after replacing Site name:" );
        System.out.println(str.replaceFirst("Beginners(.*)", "XYZ.com"));
   }
}
```
Output:

```
String after replacing com with net :Site is BeginnersBook.net
String after replacing Site name:Site is XYZ.com
```

Example 3:replaceAll() Method

```
public class Example3{
   public static void main(String args[]){
        String str = new String("My .com site is BeginnersBook.com");
        System.out.print("String after replacing all com with net :" );
        System.out.println(str.replaceAll("com", "net"));
   }
}
```
Output:

```
String after replacing all com with net :My .net site is BeginnersBook.net
```

Java - String split() Method example

The method `split()` is used for splitting a String into its substrings based on the given delimiter/regular expression. This method has two variants:

`String[] split(String regex)`: It returns an array of strings after splitting an input String based on the delimiting regular expression. `String[] split(String regex, int limit)`: The only difference between above variation and this one is that it limits the number of strings returned after split up. For e.g. `split("anydelimiter", 3)` would return the array of only 3 strings even through the delimiter is present in the string more than 3 times. If the **limit is negative** then the returned array would be having as many substrings as possible however when the **limit is zero** then the returned array would be having all the substrings excluding the trailing empty Strings.

It throws `PatternSyntaxException` if the syntax of specified regular expression is not valid.

Example: split() method

```
public class SplitExample{
   public static void main(String args[]){
        String str = new String("28/12/2013");
        System.out.println("split(String regex):");
        String array1[]= str.split("/");
        for (String temp: array1){
           System.out.println(temp);
```

```
       }
       System.out.println("split(String regex, int limit) with limit=2:");
       String array2[]= str.split("/", 2);
       for (String temp: array2){
          System.out.println(temp);
       }
       System.out.println("split(String regex, int limit) with limit=0:");
       String array3[]= str.split("/", 0);
       for (String temp: array3){
          System.out.println(temp);
       }
       System.out.println("split(String regex, int limit) with limit=-5:");
       String array4[]= str.split("/", -5);
       for (String temp: array4){
          System.out.println(temp);
       }
   }
}
```

Output:

```
split(String regex):
28
12
2013
split(String regex, int limit) with limit=2:
28
12/2013
split(String regex, int limit) with limit=0:
28
12
2013
split(String regex, int limit) with limit=-5:
28
12
2013
```

In the above example split("/",0) and split("/",-5) returned same value however in some cases the result would be different. Lets see with the help of an example:

```
String s="bbaaccaa";
String arr1[]= s.split("a", -1);
String arr2[]= s.split("a", 0);
```

In this case arr1 would be having {"bb", " ", "cc", " ", " "} However arr2 would be having {"bb", " ", "cc"} because limit zero excludes trialing empty Strings.

Java - String split() Method example

The method `split()` is used for splitting a String into its substrings based on the given delimiter/regular expression. This method has two variants:

`String[] split(String regex)`: It returns an array of strings after splitting an input String based on the delimiting regular expression.

String[] split(String regex, int limit): The only difference between above variation and this one is that it limits the number of strings returned after split up. For e.g. split("anydelimiter", 3) would return the array of only 3 strings even through the delimiter is present in the string more than 3 times. If the **limit is negative** then the returned array would be having as many substrings as possible however when the **limit is zero** then the returned array would be having all the substrings excluding the trailing empty Strings.

It throws PatternSyntaxException if the syntax of specified regular expression is not valid.

Example: split() method

```
public class SplitExample{
   public static void main(String args[]){
       String str = new String("28/12/2013");
       System.out.println("split(String regex):");
       String array1[]= str.split("/");
       for (String temp: array1){
          System.out.println(temp);
       }
       System.out.println("split(String regex, int limit) with limit=2:");
       String array2[]= str.split("/", 2);
       for (String temp: array2){
          System.out.println(temp);
       }
       System.out.println("split(String regex, int limit) with limit=0:");
       String array3[]= str.split("/", 0);
       for (String temp: array3){
          System.out.println(temp);
       }
       System.out.println("split(String regex, int limit) with limit=-5:");
       String array4[]= str.split("/", -5);
       for (String temp: array4){
          System.out.println(temp);
       }
   }
}
```

Output:

```
split(String regex):
28
12
2013
split(String regex, int limit) with limit=2:
28
12/2013
split(String regex, int limit) with limit=0:
28
12
2013
split(String regex, int limit) with limit=-5:
28
12
```

```
2013
```
In the above example split(“/”,0) and split(“/”,-5) returned same value however in some cases the result would be different. Lets see with the help of an example:

```
String s="bbaaccaa";
String arr1[]= s.split("a", -1);
String arr2[]= s.split("a", 0);
```
In this case arr1 would be having {“bb”, ” “, “cc”, ” “, ” “} However arr2 would be having {“bb”, ” “, “cc”} because limit zero excludes trialing empty Strings.

Java - String toLowerCase() and toUpperCase() Methods

The method `toLowerCase()` converts the characters of a `String` into lower case characters. It has two variants:

`String toLowerCase(Locale locale)`: It converts the string into Lowercase using the rules defined by specified Locale.

`String toLowerCase()`: It is equivalent to `toLowerCase(Locale.getDefault())`. `Locale.getDefault()` gets the current value of the default locale for this instance of the Java Virtual Machine. The Java Virtual Machine sets the default locale during startup based on the host environment. It is used by many locale-sensitive methods if no locale is explicitly specified. It can be changed using the `setDefault()` method.

Example: toLowerCase() method

```
import java.util.Locale;
public class LowerCaseExample{
   public static void main(String args[]){
       String str = new String("ABC IS NOT EQUAL TO XYZ");
       //Standard method of conversion
       System.out.println(str.toLowerCase());

       //By specifying Locale
       System.out.println(str.toLowerCase(Locale.FRANCE));
   }
}
```
Output:

```
abc is not equal to xyz
abc is not equal to xyz
```

Method: toUpperCase()

Like toLowerCase() method, toUpperCase() also has two variants:
String toUpperCase(Locale locale): It converts the string into a UpperCase string using the rules defined by the specified Locale.
String toUpperCase(): It is equavalent to toUpperCase(Locale.getDefault()).

Example: toUpperCase() method

```
import java.util.Locale;
public class UpperCaseExample{
   public static void main(String args[]){
       String str = new String("this is a test string");
       //Standard method of conversion
       System.out.println(str.toUpperCase());

       //By specifying Locale
       System.out.println(str.toUpperCase(Locale.CHINA));
   }
}
```

Output:

```
THIS IS A TEST STRING
THIS IS A TEST STRING
```

Java String format() method explained with examples

BY CHAITANYA SINGH | FILED UNDER: STRING HANDLING

Java String format() method is used for formatting the String. There are so many things you can do with this method, for example we can even concatenate the strings using this method and at the same time, we can format the output concatenated string. In this tutorial, we will see several examples of Java String format() method.

Syntax of format() method

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

Returns a formatted string using the specified locale, format string, and arguments.

and

```
public static String format(String format,
            Object... args)
```
Returns a formatted string using the specified format string and arguments.

A Simple Example of Java String format() method

```
public class Example{
   public static void main(String args[]){
        String str = "just a string";

        //concatenating string using format
        String formattedString = String.format("My String is %s", str);

        /*formatting the  value passed and concatenating at the same time
         * %.6f is for having 6 digits in the fractional part
         */
        String formattedString2 = String.format("My String is %.6f",12.121);

        System.out.println(formattedString);
        System.out.println(formattedString2);
   }
}
```
Output:

```
My String is just a string
My String is 12.121000
```

Java String format() example of concatenating arguments to the string

We can specify the argument positions using %1$, %2$,..format specifiers. Here %1$ represents first argument, %2$ second argument and so on.

```
public class Example{
   public static void main(String args[]){
        String str1 = "cool string";
        String str2 = "88";
        /* Specifying argument positions. %1$ is for the first argument and
         * %2$ is for the second argument
         */
        String fstr = String.format("My String is: %1$s, %1$s and %2$s", str1, 
str2);
        System.out.println(fstr);
   }
}
```
Output:

```
My String is: cool string, cool string and 88
```
As you can see that how we have passed the string “cool string” twice in the format() method using the argument positions format specifiers.

Left padding the string using string format()

In this example, we are left padding a number wth 0's and converting the number to a formatted String. In the above example we have formatted the float numbers and Strings and in this example we are formatting an integer. The important point to remember is that the format specifiers for these are different.
%s – for strings
%f – for floats
%d – for integers

```
public class Example{
   public static void main(String args[]){
        int str = 88;
        /* Left padding an integer number with 0's and converting it
         * into a String using Java String format() method.
         */
        String formattedString = String.format("%05d", str);
        System.out.println(formattedString);
   }
}
```

Output:

```
00088
```

Java - String toLowerCase() and toUpperCase() Methods

BY CHAITANYA SINGH | FILED UNDER: STRING HANDLING

The method `toLowerCase()` converts the characters of a `String` into lower case characters. It has two variants:

`String toLowerCase(Locale locale)`: It converts the string into Lowercase using the rules defined by specified Locale.

`String toLowerCase()`: It is equivalent to `toLowerCase(Locale.getDefault())`. `Locale.getDefault()` gets the current value of the default locale for this instance of the Java Virtual Machine. The Java Virtual Machine sets the default locale during startup based on the host environment. It is used by many locale-sensitive methods if no locale is explicitly specified. It can be changed using the `setDefault()` method.

Example: toLowerCase() method

```
import java.util.Locale;
public class LowerCaseExample{
   public static void main(String args[]){
       String str = new String("ABC IS NOT EQUAL TO XYZ");
       //Standard method of conversion
       System.out.println(str.toLowerCase());

       //By specifying Locale
       System.out.println(str.toLowerCase(Locale.FRANCE));
   }
}
```
Output:

```
abc is not equal to xyz
abc is not equal to xyz
```

Method: toUpperCase()

Like toLowerCase() method, toUpperCase() also has two variants:
String toUpperCase(Locale locale): It converts the string into a UpperCase string using the rules defined by the specified Locale.
String toUpperCase(): It is equavalent to toUpperCase(Locale.getDefault()).

Example: toUpperCase() method

```
import java.util.Locale;
public class UpperCaseExample{
   public static void main(String args[]){
       String str = new String("this is a test string");
       //Standard method of conversion
       System.out.println(str.toUpperCase());

       //By specifying Locale
       System.out.println(str.toUpperCase(Locale.CHINA));
   }
}
```
Output:

```
THIS IS A TEST STRING
THIS IS A TEST STRING
```

Java - String trim() and hashCode() Methods

In this tutorial we will discuss about trim() and hashCode() methods. Lets discuss about trim() first: It returns a String after removing leading and trailing white spaces from the input String. For e.g. "   Hello".trim() would return the String "Hello".

```
public String trim()
```

Example: trim() method

```
public class TrimExample{
   public static void main(String args[]){
       String str = new String("    How are you??   ");
       System.out.println("String before trim: "+str);
       System.out.println("String after trim: "+str.trim());
   }
}
```

Output:

```
String before trim:     How are you??
String after trim: How are you??
```

hashCode() method

This method returns the hash code for the String. The computation is done like this:

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

Syntax of this method:

```
public int hashCode()
```

Example: hashCode() method

```
public class HashCodeExample{
   public static void main(String args[]){
       String str = new String("Welcome!!");
       System.out.println("Hash Code for String str: "+str.hashCode());
   }
}
```

Output:

```
Hash Code for String str: 1601728226
```

Java - String toCharArray() Method example

The method toCharArray() returns an Array of chars after converting a String into sequence of characters. The returned array length is equal to the length of the String and the sequence of chars in Array matches the sequence of characters in the String.

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

Example: toCharArray() method

In this example we are converting a String into array of chars using toCharArray() method.

```
public class CharArrayExample{
   public static void main(String args[]){
       String str = new String("Welcome to BeginnersBook.com");
       char[] array= str.toCharArray();
       System.out.print("Content of Array:");
       for(char c: array){
           System.out.print(c);
       }
   }
}
```

Output:

```
Content of Array:Welcome to BeginnersBook.com
```

Java - String copyValueOf() Method example

The method `copyValueOf()` is used for copying an array of characters to the `String`. The point to note here is that this method does not append the content in String, instead it replaces the existing string value with the sequence of characters of array.
It has two variations:
1) `static copyValueOf(char[] data)`: It copies the whole array (data) to the string.
2) `static String copyValueOf(char[] data, int offset, int count)`: It copies only specified characters to the string using the specified offset and count values. offset is the initial index from where characters needs to be copied and count is a number of characters to copy. For e.g. offset 2 and count 3 would be interpreted as: Only 3 characters of array starting from 2nd index(3rd position as index starts with 0) should be copied to the concerned String.

Example

In this example we have two strings str1 & str2 and an array of chars named data. We are copying the array to the strings using both the variations of method `copyValueOf()`.

```
public class CopyValueOfExample {
   public static void main(String args[]) {
       char[] data = {'a','b','c','d','e','f','g','h','i','j','k'};
       String str1 = "Text";
       String str2 = "String";
       //Variation 1:String copyValueOf(char[] data)
       str1 = str1.copyValueOf(data);
       System.out.println("str1 after copy: " + str1);

       //Variation 2:String copyValueOf(char[] data,int offset,int count)
       str2 = str2.copyValueOf(data, 5, 3 );
       System.out.println("str2 after copy: " + str2);
   }
}
```

Output:

```
str1 after copy: abcdefghijk
str2 after copy: fgh
```

Java - String copyValueOf() Method example

The method `copyValueOf()` is used for copying an array of characters to the `String`. The point to note here is that this method does not append the content in String, instead it replaces the existing string value with the sequence of characters of array.
It has two variations:
1) `static copyValueOf(char[] data)`: It copies the whole array (data) to the string.
2) `static String copyValueOf(char[] data, int offset, int count)`: It copies only specified characters to the string using the specified offset and count values. offset is the initial index from where characters needs to be copied and count is a number of characters to copy. For e.g. offset 2 and count 3 would be interpreted as: Only 3 characters of array starting from 2nd index(3rd position as index starts with 0) should be copied to the concerned String.

Example

In this example we have two strings str1 & str2 and an array of chars named data. We are copying the array to the strings using both the variations of method `copyValueOf()`.

```
public class CopyValueOfExample {
   public static void main(String args[]) {
       char[] data = {'a','b','c','d','e','f','g','h','i','j','k'};
       String str1 = "Text";
       String str2 = "String";
       //Variation 1:String copyValueOf(char[] data)
       str1 = str1.copyValueOf(data);
       System.out.println("str1 after copy: " + str1);

       //Variation 2:String copyValueOf(char[] data,int offset,int count)
       str2 = str2.copyValueOf(data, 5, 3 );
       System.out.println("str2 after copy: " + str2);
   }
}
```

Output:

```
str1 after copy: abcdefghijk
str2 after copy: fgh
```

Java - String getChars() Method example

The method getChars() is used for copying String characters to an Array of chars.

```
public void getChars(int srcBegin, int srcEnd, char[] dest, int destBegin)
```

Parameters **description:**
srcBegin – index of the first character in the string to copy.
srcEnd – index after the last character in the string to copy.
dest – Destination array of characters in which the characters from String gets copied.
destBegin – The index in Array starting from where the chars will be pushed into the Array.

It throws IndexOutOfBoundsException – If any of the following conditions occurs:
(srcBegin<0) srcBegin is less than zero. (srcBegin>srcEnd) srcBegin is greater than srcEnd.
(srcEnd > length of string) srcEnd is greater than the length of this string.
(destBegin<0) destBegin is negative.
dstBegin+(srcEnd-srcBegin) is larger than dest.length.

Example: getChars() method

```
public class GetCharsExample{
   public static void main(String args[]){
       String str = new String("This is a String Handling Tutorial");
       char[] array = new char[6];
       str.getChars(10, 16, array, 0);
       System.out.println("Array Content:" );
       for(char temp: array){
           System.out.print(temp);
       }

       char[] array2 = new char[]{'a','a','a','a','a','a','a','a'};
       str.getChars(10, 16, array2, 2);
       System.out.println("Second Array Content:" );
       for(char temp: array2){
           System.out.print(temp);
       }
   }
}
```

Output:

```
Array Content:
StringSecond Array Content:
aaString
```

Java String valueOf() method explained with examples

Java String valueOf() method returns the String representation of the boolean, char, char array, int, long, float and double arguments. We have different versions of this method for each type of arguments. The signature of this method is as follows:

Method valueOf() signature:

```
public static String valueOf(boolean b): Used for converting boolean value to a
String
public static String valueOf(char c): char to String
public static String valueOf(int i): int to String
public static String valueOf(long l): long to String
public static String valueOf(float f): float to String
public static String valueOf(double d): double to String
```

Java String valueOf() Example

In example we are using Java String valueOf() method to convert the integer, float, long, double, char and char array to the String.

```
public class Example{
   public static void main(String args[]){
        int i = 10; //int value
        float f = 10.10f; //float value
        long l = 111L; //long value
        double d = 2222.22; //double value
        char ch = 'A'; //char value
        char array[] = {'a', 'b', 'c'}; //char array

        //converting int to String
        String str1 = String.valueOf(i);

        //converting float to String
        String str2 = String.valueOf(f);

        //converting long to String
        String str3 = String.valueOf(l);

        //converting double to String
        String str4 = String.valueOf(d);

        //converting char to String
        String str5 = String.valueOf(ch);

        //converting char array to String
        String str6 = String.valueOf(array);
        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
        System.out.println(str4);
        System.out.println(str5);
        System.out.println(str6);
    }
}
```
Output:

```
10
10.1
111
2222.22
A
abc
```

ava - String contentEquals() Method example

The method `contentEquals()` compares the `String` with the `String Buffer` and returns a boolean value. It returns true if the String matches to the String buffer else it returns false.

`boolean contentEquals(StringBuffer sb)`

Example

In this example we have two Strings and two String Buffers. We are comparing the Strings with String Buffers using the `contentEquals()` method. Here we are displaying the result by directly calling the method in `System.out.println` statement. However you can also store the returned value in a boolean variable and use it further like this: `boolean var = str1.contentEquals(sb1);`

```
public class ContentEqualsExample {
   public static void main(String args[]) {
       String str1 = "First String";
       String str2 = "Second String";
       StringBuffer str3 = new StringBuffer( "Second String");
       StringBuffer str4 = new StringBuffer( "First String");
       System.out.println("str1 equals to str3:"+str1.contentEquals(str3));
       System.out.println("str2 equals to str3:"+str2.contentEquals(str3));
       System.out.println("str1 equals to str4:"+str1.contentEquals(str4));
       System.out.println("str2 equals to str4:"+str2.contentEquals(str4));
   }
}
```
Output:

```
str1 equals to str3:false
str2 equals to str3:true
str1 equals to str4:true
str2 equals to str4:false
```

Java - String regionMatches() Method example

The method regionMatches() tests if the two Strings are equal. Using this method we can compare the substring of input String with the substring of specified String.

Two variants:
public boolean regionMatches(int toffset, String other, int ooffset, int len): Case sensitive test.
public boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len): It has option to consider or ignore the case.

Parameters description:
ignoreCase– if true, ignore case when comparing characters.
toffset – the starting offset of the subregion in this string.
other – the string argument.
ooffset – the starting offset of the subregion in the string argument.
len – the number of characters to compare.

Example: regionMatches() method

```java
public class RegionMatchesExample{
   public static void main(String args[]){
       String str1 = new String("Hello, How are you");
       String str2 = new String("How");
       String str3 = new String("HOW");

       System.out.print("Result of Test1: " );
       System.out.println(str1.regionMatches(7, str2, 0, 3));

       System.out.print("Result of Test2: " );
       System.out.println(str1.regionMatches(7, str3, 0, 3));

       System.out.print("Result of Test3: " );
       System.out.println(str1.regionMatches(true, 7, str3, 0, 3));
   }
}
```

Output:

```
Result of Test1: true
Result of Test2: false
Result of Test3: true
```

Java - String regionMatches() Method example

The method regionMatches() tests if the two Strings are equal. Using this method we can compare the substring of input String with the substring of specified String.

Two variants:
public boolean regionMatches(int toffset, String other, int ooffset, int len): Case sensitive test.
public boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len): It has option to consider or ignore the case.

Parameters description:
ignoreCase– if true, ignore case when comparing characters.
toffset – the starting offset of the subregion in this string.
other – the string argument.
ooffset – the starting offset of the subregion in the string argument.
len – the number of characters to compare.

Example: regionMatches() method

```
public class RegionMatchesExample{
   public static void main(String args[]){
       String str1 = new String("Hello, How are you");
       String str2 = new String("How");
       String str3 = new String("HOW");

       System.out.print("Result of Test1: " );
       System.out.println(str1.regionMatches(7, str2, 0, 3));

       System.out.print("Result of Test2: " );
       System.out.println(str1.regionMatches(7, str3, 0, 3));

       System.out.print("Result of Test3: " );
       System.out.println(str1.regionMatches(true, 7, str3, 0, 3));
   }
}
```
Output:

```
Result of Test1: true
Result of Test2: false
Result of Test3: true
```

Java - String getBytes() Method example

The getBytes() method encodes a given String into a sequence of bytes and returns an array of bytes. The method can be used in below two ways:

public byte[] getBytes(String charsetName): It encodes the String into sequence of bytes using the specified charset and return the array of those bytes.

It throws UnsupportedEncodingException – If the specified charset is not supported. public byte[] getBytes(): It encodes the String using default charset method.

Example: getBytes() method

```
import java.io.*;
public class GetBytesExample{
   public static void main(String args[]){
       String str = new String("Hello");
       byte[] array1 = str.getBytes();
       System.out.print("Default Charset encoding:");
       for(byte b: array1){
           System.out.print(b);
       }
       System.out.print("\nUTF-16 Charset encoding:");
       try{
             byte [] array2 = str.getBytes("UTF-16");
             for(byte b1: array2){
                System.out.print(b1);
             }
             byte [] array3 = str.getBytes("UTF-16BE");
             System.out.print("\nUTF-16BE Charset encoding:");
             for(byte b2: array3){
                System.out.print(b2);
             }
        }catch(UnsupportedEncodingException ex){
             System.out.println("Unsupported character set"+ex);
        }
   }
}
```

Output:

```
Default Charset encoding:72101108108111
UTF-16 Charset encoding:-2-10720101010801080111
UTF-16BE Charset encoding:0720101010801080111
```

In the above example we have done encoding using charset UTF -16 and UTF -16BE, there are many other standard charset like:

Java - String getBytes() Method example

The getBytes() method encodes a given String into a sequence of bytes and returns an array of bytes. The method can be used in below two ways:

public byte[] getBytes(String charsetName): It encodes the String into sequence of bytes using the specified charset and return the array of those bytes. It throws UnsupportedEncodingException – If the specified charset is not supported. public byte[] getBytes(): It encodes the String using default charset method.

Example: getBytes() method

```
import java.io.*;
public class GetBytesExample{
   public static void main(String args[]){
       String str = new String("Hello");
       byte[] array1 = str.getBytes();
       System.out.print("Default Charset encoding:");
       for(byte b: array1){
           System.out.print(b);
       }
       System.out.print("\nUTF-16 Charset encoding:");
       try{
             byte [] array2 = str.getBytes("UTF-16");
             for(byte b1: array2){
                System.out.print(b1);
             }
             byte [] array3 = str.getBytes("UTF-16BE");
             System.out.print("\nUTF-16BE Charset encoding:");
             for(byte b2: array3){
                System.out.print(b2);
             }
        }catch(UnsupportedEncodingException ex){
             System.out.println("Unsupported character set"+ex);
        }
   }
}
```

Output:

```
Default Charset encoding:72101108108111
UTF-16 Charset encoding:-2-10720101010801080111
UTF-16BE Charset encoding:0720101010801080111
```

In the above example we have done encoding using charset `UTF -16` and `UTF -16BE`, there are many other standard charset like:

Java - String length() Method example

BY CHAITANYA SINGH | FILED UNDER: STRING HANDLING

Java String `length()` method is used for finding out the length of a String. This method counts the number of characters in a String including the white spaces and returns the count.

Method signature

```
int length()
```

String length limit: The maximum size a string can have is: 2^{31}-1.

Example1: length()

In this example we have three different Strings and we are finding out the length of them using `length()` method

```
public class LengthExample{
   public static void main(String args[]) {
       String str1= new String("Test String");
       String str2= new String("Chaitanya");
       String str3= new String("BeginnersBook");
       System.out.println("Length of str1:"+str1.length());
       System.out.println("Length of str2:"+str2.length());
       System.out.println("Length of str3:"+str3.length());
   }
}
```

Output:

```
Length of str1:11
Length of str2:9
Length of str3:13
```

Example 2: Java String length() method to calculate the length of String without spaces

As we have already seen in the above example that this method calculates the length including white spaces. In case if you only want to count the number of characters in a string excluding spaces then you can do so by using the string replace method as shown in the example below.

Here we are omitting the spaces in the given String using replace method and then using the **length() method** on it.

```
public class Example {
   public static void main(String[] args) {
        String str = "hi guys    this is a string";

        //length of the String
        System.out.println("Length of the String: "+str.length());

        //length of the String without white spaces
        System.out.println("Length of String without spaces: "+
        str.replace(" ", "").length());
   }
}
```

Output:

```
Length of the String: 27
Length of String without spaces: 19
```

Java – String matches() Method example

BY CHAITANYA SINGH | FILED UNDER: STRING HANDLING

Method matches() checks whether the String is matching with the specified regular expression. If the String fits in the specified regular expression then this method returns true else it returns false. Below is the syntax of the method:

```
public boolean matches(String regex)
```

It throws PatternSyntaxException – if the specified regular expression is not valid.

Example: matches() method

In this example we have a String and three regular expressions. We are matching the regular expressions(regex) with the input String using the matches() method.

```
public class MatchesExample{
   public static void main(String args[]){
       String str = new String("Java String Methods");

       System.out.print("Regex: (.*)String(.*) matches string? " );
       System.out.println(str.matches("(.*)String(.*)"));

       System.out.print("Regex: (.*)Strings(.*) matches string? " );
       System.out.println(str.matches("(.*)Strings(.*)"));

       System.out.print("Regex: (.*)Methods matches string? " );
       System.out.println(str.matches("(.*)Methods"));
   }
}
```

Output:

```
Regex: (.*)String(.*) matches string? true
Regex: (.*)Strings(.*) matches string? false
Regex: (.*)Methods matches string? true
```