Day 5
Thu 22 Aug 2017
Java Tutorial
Instructor: Hameed Mahmoud

Day 5

Basics of Java Programming Language

Objectives

1. Practice
2. String Methods
3. Functions

Sr.No.	Method & Description
1	**char charAt(int index)** Returns the character at the specified index.
2	**int compareTo(Object o)** Compares this String to another Object.
3	**int compareTo(String anotherString)** Compares two strings lexicographically.
4	**int compareToIgnoreCase(String str)**

	Compares two strings lexicographically, ignoring case differences.
5	**String concat(String str)** Concatenates the specified string to the end of this string.
6	**boolean contentEquals(StringBuffer sb)** Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
7	**static String copyValueOf(char[] data)** Returns a String that represents the character sequence in the array specified.
8	**static String copyValueOf(char[] data, int offset, int count)** Returns a String that represents the character sequence in the array specified.
9	**boolean endsWith(String suffix)** Tests if this string ends with the specified suffix.
10	**boolean equals(Object anObject)**

	Compares this string to the specified object.
11	**boolean equalsIgnoreCase(String anotherString)** Compares this String to another String, ignoring case considerations.
12	**byte getBytes()** Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
13	**byte[] getBytes(String charsetName)** Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
14	**void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** Copies characters from this string into the destination character array.
15	**int hashCode()** Returns a hash code for this string.
16	**int indexOf(int ch)**

	Returns the index within this string of the first occurrence of the specified character.
17	**int indexOf(int ch, int fromIndex)** Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
18	**int indexOf(String str)** Returns the index within this string of the first occurrence of the specified substring.
19	**int indexOf(String str, int fromIndex)** Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
20	**String intern()** Returns a canonical representation for the string object.
21	**int lastIndexOf(int ch)** Returns the index within this string of the last occurrence of the specified character.

22	**int lastIndexOf(int ch, int fromIndex)** Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
23	**int lastIndexOf(String str)** Returns the index within this string of the rightmost occurrence of the specified substring.
24	**int lastIndexOf(String str, int fromIndex)** Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
25	**int length()** Returns the length of this string.
26	**boolean matches(String regex)** Tells whether or not this string matches the given regular expression.
27	**boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)** Tests if two string regions are equal.

28	**boolean regionMatches(int toffset, String other, int ooffset, int len)** Tests if two string regions are equal.
29	**String replace(char oldChar, char newChar)** Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
30	**String replaceAll(String regex, String replacement** Replaces each substring of this string that matches the given regular expression with the given replacement.
31	**String replaceFirst(String regex, String replacement)** Replaces the first substring of this string that matches the given regular expression with the given replacement.
32	**String[] split(String regex)** Splits this string around matches of the given regular expression.
33	**String[] split(String regex, int limit)** Splits this string around matches of the given regular expression.

34	**boolean startsWith(String prefix)** Tests if this string starts with the specified prefix.
35	**boolean startsWith(String prefix, int toffset)** Tests if this string starts with the specified prefix beginning a specified index.
36	**CharSequence subSequence(int beginIndex, int endIndex)** Returns a new character sequence that is a subsequence of this sequence.
37	**String substring(int beginIndex)** Returns a new string that is a substring of this string.
38	**String substring(int beginIndex, int endIndex)** Returns a new string that is a substring of this string.
39	**char[] toCharArray()** Converts this string to a new character array.

40	**String toLowerCase()** Converts all of the characters in this String to lower case using the rules of the default locale.
41	**String toLowerCase(Locale locale)** Converts all of the characters in this String to lower case using the rules of the given Locale.
42	**String toString()** This object (which is already a string!) is itself returned.
43	**String toUpperCase()** Converts all of the characters in this String to upper case using the rules of the default locale.
44	**String toUpperCase(Locale locale)** Converts all of the characters in this String to upper case using the rules of the given Locale.
45	**String trim()** Returns a copy of the string, with leading and trailing whitespace omitted.

46	**static String valueOf(primitive data type x)** Returns the string representation of the passed data type argument.

Java Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.**println()** method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

Creating Method

Considering the following example to explain the syntax of a method –

Syntax

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

- **public static** – modifier
- **int** – return type
- **methodName** – name of the method
- **a, b** – formal parameters

- **int a, int b** – list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax –

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes –

- **modifier** – It defines the access type of the method and it is optional to use.
- **returnType** – Method may return a value.
- **nameOfMethod** – This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List** – The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body** – The method body defines what the method does with the statements.

Example

Here is the source code of the above defined method called **max()**. This method takes two parameters num1 and num2 and returns the maximum between the two –

```
/** the snippet returns the minimum between two numbers */

public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;
```

```
   return min;
}
```

Method Calling

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when −

- the return statement is executed.
- it reaches the method ending closing brace.

The methods returning void is considered as call to a statement. Lets consider an example −

```
System.out.println("This is tutorialspoint.com!");
```

The method returning value can be understood by the following example −

```
int result = sum(6, 9);
```

Following is the example to demonstrate how to define a method and how to call it −

Example

```
public class ExampleMinNumber {

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("Minimum Value = " + c);
   }

   /** returns the minimum of two numbers */
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
```

```
         min = n1;

      return min;
   }
}
```

This will produce the following result −

Output

```
Minimum value = 6
```

The void Keyword

The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering a void method *methodRankPoints*. This method is a void method, which does not return any value. Call to a void method must be a statement i.e. *methodRankPoints(255.7);*. It is a Java statement which ends with a semicolon as shown in the following example.

Example

```
public class ExampleVoid {

   public static void main(String[] args) {
      methodRankPoints(255.7);
   }

   public static void methodRankPoints(double points) {
      if (points >= 202.5) {
         System.out.println("Rank:A1");
      }else if (points >= 122.4) {
         System.out.println("Rank:A2");
      }else {
         System.out.println("Rank:A3");
      }
   }
}
```

This will produce the following result −

Output

Rank:A1

Passing Parameters by Value

While working under calling process, arguments is to be passed. These should be in the same order as their respective parameters in the method specification. Parameters can be passed by value or by reference.

Passing Parameters by Value means calling a method with a parameter. Through this, the argument value is passed to the parameter.

Example

The following program shows an example of passing parameter by value. The values of the arguments remains the same even after the method invocation.

```
public class swappingExample {

   public static void main(String[] args) {
      int a = 30;
      int b = 45;
      System.out.println("Before swapping, a = " + a + " and b = " + b);

      // Invoke the swap method
      swapFunction(a, b);
      System.out.println("\n**Now, Before and After swapping values will be
same here**:");
      System.out.println("After swapping, a = " + a + " and b is " + b);
   }

   public static void swapFunction(int a, int b) {
      System.out.println("Before swapping(Inside), a = " + a + " b = " + b);

      // Swap n1 with n2
      int c = a;
      a = b;
      b = c;
      System.out.println("After swapping(Inside), a = " + a + " b = " + b);
   }
}
```

This will produce the following result –

Output

```
Before swapping, a = 30 and b = 45
Before swapping(Inside), a = 30 b = 45
After swapping(Inside), a = 45 b = 30

**Now, Before and After swapping values will be same here**:
After swapping, a = 30 and b is 45
```