JAVA-8

java programming language

tutorialspoint

SIMPLYEASYLEARNING

About the Tutorial

Java 8 is the most awaited and is a major feature release of Java programming language.

This is an introductory tutorial that explains the basic-to-advanced features of Java 8 and their usage in a simple and intuitive way.

Audience

This tutorial will be useful for most Java developers, starting from beginners to experts.

After completing this tutorial, you will find yourself at a moderate level of expertise in Java 8, from where you can take yourself to next levels.

Prerequisites

Knowledge of basic Java programming language is the only prerequisite for learning the concepts explained in this tutorial.

Copyright & Disclaimer

Table of Contents

1. JAVA 8 – OVERVIEW

JAVA 8 is a major feature release of JAVA programming language development. Its initial version was released on 18 March 2014. With the Java 8 release, Java provided supports for functional programming, new JavaScript engine, new APIs for date time manipulation, new streaming API, etc.

New Features

- **Lambda expression** - Adds functional processing capability to Java.
- **Method references** - Referencing functions by their names instead of invoking them directly. Using functions as parameter.
- **Default method** - Interface to have default method implementation.
- **New tools** - New compiler tools and utilities are added like 'jdeps' to figure out dependencies.
- **Stream API** - New stream API to facilitate pipeline processing.
- **Date Time API** - Improved date time API.
- **Optional** - Emphasis on best practices to handle null values properly.
- **Nashorn, JavaScript Engine** - A Java-based engine to execute JavaScript code.

Consider the following code snippet.

```
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;
import java.util.Comparator;

public class Java8Tester {

   public static void main(String args[]){
      List<String> names1 = new ArrayList<String>();
      names1.add("Mahesh ");
      names1.add("Suresh ");
      names1.add("Ramesh ");
      names1.add("Naresh ");
      names1.add("Kalpesh ");

      List<String> names2 = new ArrayList<String>();
      names2.add("Mahesh ");
      names2.add("Suresh ");
      names2.add("Ramesh ");
      names2.add("Naresh ");
      names2.add("Kalpesh ");

      Java8Tester tester = new Java8Tester();

      System.out.println("Sort using Java 7 syntax: ");
      tester.sortUsingJava7(names1);
      System.out.println(names1);

      System.out.println("Sort using Java 8 syntax: ");
      tester.sortUsingJava8(names2);
      System.out.println(names2);
   }

   private void sortUsingJava7(List<String> names){
      //sort using java 7
      Collections.sort(names, new Comparator<String>() {
         @Override
         public int compare(String s1, String s2) {
            return s1.compareTo(s2);
         }
      });
   }
   private void sortUsingJava8(List<String> names){
      // sort using java 8
      Collections.sort(names, (s1, s2) ->  s1.compareTo(s2));
   }
```

```
}
```

Run the program to get the following result.

```
Sort using Java 7 syntax:
[ Kalpesh Mahesh Naresh Ramesh Suresh ]
Sort using Java 8 syntax:
[ Kalpesh Mahesh Naresh Ramesh Suresh ]
```

Here the **sortUsingJava8()** method uses sort function with a lambda expression as parameter to get the sorting criteria.

2. JAVA 8 – ENVIRONMENT SETUP

Try it Option Online

We have set up the Java Programming environment online, so that you can compile and execute all the available examples online. It gives you confidence in what you are reading and enables you to verify the programs with different options. Feel free to modify any example and execute it online.

Try the following example using our online compiler available at http://www.compileonline.com/

```
public class MyFirstJavaProgram {


    public static void main(String []args) {
       System.out.println("Hello World");
    }
}
```

For most of the examples given in this tutorial, you will find a **Try it** option in our website code sections at the top right corner that will take you to the online compiler. So just make use of it and enjoy your learning.

Local Environment Setup

If you want to set up your own environment for Java programming language, then this section guides you through the whole process. Please follow the steps given below to set up your Java environment.

Java SE can be downloaded for free from the following link:
http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

You download a version based on your operating system.

Follow the instructions to download Java, and run the **.exe** to install Java on your machine. Once you have installed Java on your machine, you would need to set environment variables to point to correct installation directories.

Setting Up the Path for Windows 2000/XP

Assuming you have installed Java in c:\Program Files\java\jdk directory:

1. Right-click on 'My Computer' and select 'Properties'.

2. Click on the 'Environment variables' button under the 'Advanced' tab.
3. Now, alter the 'Path' variable so that it also contains the path to the Java executable. For example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

Setting Up the Path for Windows 95/98/ME

Assuming you have installed Java in c:\Program Files\java\jdk directory:

- Edit the 'C:\autoexec.bat' file and add the following line at the end:
  'SET PATH=%PATH%;C:\Program Files\java\jdk\bin'

Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation if you have trouble doing this.

For example, if you use *bash* as your shell, then you would add the following line at the end of your '.bashrc: export PATH=/path/to/java:$PATH'

Popular Java Editors

To write Java programs, you need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following:

- **Notepad:** On Windows machine, you can use any simple text editor like Notepad (recommended for this tutorial) or TextPad.
- **Netbeans:** It is a Java IDE that is open-source and free. It can be downloaded from **http://www.netbeans.org/index.html**.
- **Eclipse:** It is also a Java IDE developed by the Eclipse open-source community and can be downloaded from **http://www.eclipse.org/**.

3. JAVA 8 – LAMBDA EXPRESSIONS

Lambda expressions are introduced in Java 8 and are touted to be the biggest feature of Java 8. Lambda expression facilitates functional programming, and simplifies the development a lot.

Syntax

A lambda expression is characterized by the following syntax.

```
parameter -> expression body
```

Following are the important characteristics of a lambda expression.

- **Optional type declaration** - No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
- **Optional parenthesis around parameter** - No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
- **Optional curly braces** - No need to use curly braces in expression body if the body contains a single statement.
- **Optional return keyword** – The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Lambda Expressions Example

Create the following Java program using any editor of your choice in, say, C:\> JAVA.

Java8Tester.java

```
public class Java8Tester {
   public static void main(String args[]){
      Java8Tester tester = new Java8Tester();


      //with type declaration
      MathOperation addition = (int a, int b) -> a + b;


      //with out type declaration
      MathOperation subtraction = (a, b) -> a - b;

      //with return statement along with curly braces
      MathOperation multiplication = (int a, int b) -> { return a * b; };
      //without return statement and without curly braces
      MathOperation division = (int a, int b) -> a / b;

      System.out.println("10 + 5 = " + tester.operate(10, 5, addition));
      System.out.println("10 - 5 = " + tester.operate(10, 5, subtraction));
      System.out.println("10 x 5 = " + tester.operate(10, 5, multiplication));
      System.out.println("10 / 5 = " + tester.operate(10, 5, division));

      //with parenthesis
      GreetingService greetService1 = message ->
      System.out.println("Hello " + message);

      //without parenthesis
      GreetingService greetService2 = (message) ->
      System.out.println("Hello " + message);

      greetService1.sayMessage("Mahesh");
      greetService2.sayMessage("Suresh");
   }

   interface MathOperation {
      int operation(int a, int b);
   }

   interface GreetingService {
      void sayMessage(String message);
   }

   private int operate(int a, int b, MathOperation mathOperation){
      return mathOperation.operation(a, b);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows:

```
C:\JAVA>javac Java8Tester.java
```

Now run the Java8Tester as follows:

```
C:\JAVA>java Java8Tester
```

It should produce the following output:

```
10 + 5 = 15
10 - 5 = 5
10 x 5 = 50
10 / 5 = 2
Hello Mahesh
Hello Suresh
```

Following are the important points to be considered in the above example.

- Lambda expressions are used primarily to define inline implementation of a functional interface, i.e., an interface with a single method only. In the above example, we've used various types of lambda expressions to define the operation method of MathOperation interface. Then we have defined the implementation of sayMessage of GreetingService.
- Lambda expression eliminates the need of anonymous class and gives a very simple yet powerful functional programming capability to Java.

Scope

Using lambda expression, you can refer to any final variable or effectively final variable (which is assigned only once). Lambda expression throws a compilation error, if a variable is assigned a value the second time.

Scope Example

Create the following Java program using any editor of your choice in, say, C:\> JAVA.

Java8Tester.java

```
public class Java8Tester {
   final static String salutation = "Hello! ";
   public static void main(String args[]){
      GreetingService greetService1 = message ->
      System.out.println(salutation + message);
      greetService1.sayMessage("Mahesh");
   }
   interface GreetingService {
      void sayMessage(String message);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows:

```
C:\JAVA>javac Java8Tester.java
```

Now run the Java8Tester as follows:

```
C:\JAVA>java Java8Tester
```

It should produce the following output:

```
Hello! Mahesh
```

4. JAVA 8 – METHOD REFERENCES

Method references help to point to methods by their names. A method reference is described using "::" symbol. A method reference can be used to point the following types of methods:

- Static methods
- Instance methods
- Constructors using new operator (TreeSet::new)

Method Reference Example

Create the following Java program using any editor of your choice in, say, C:\> JAVA.

Java8Tester.java

```
import java.util.List;
import java.util.ArrayList;
public class Java8Tester {

   public static void main(String args[]){

      List names = new ArrayList();
      names.add("Mahesh");
      names.add("Suresh");
      names.add("Ramesh");
      names.add("Naresh");
      names.add("Kalpesh");

      names.forEach(System.out::println);
   }
}
```

Here we have passed System.out::println method as a static method reference.

Verify the Result

Compile the class using **javac** compiler as follows:

```
C:\JAVA>javac Java8Tester.java
```

Now run the Java8Tester as follows:

```
C:\JAVA>java Java8Tester
```

It should produce the following output:

```
Mahesh
Suresh
Ramesh
Naresh
Kalpesh
```