# Java 7: Top 8 new features with examples

Posted on November 9, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓



There are several small new features and enhancements in Java 7. The major features and enhancements are in Java 8. Let's look at the Java 7 new features.

### #1: string in switch statement:

```
1  public class Java7Feature1 {
2
3   private static String color = "BLUE";
4
5   private enum Color {
6    RED, GREEN
7   };
8
9   public static void main(String[] args) {
```

```
10
11    // Pre Java 5
12    if (color.equals("RED")) {
13     System.out.println("Color is Red");
14    } else if (color.equals("GREEN")) {
15     System.out.println("Color is Green");
16    } else {
17     System.out.println("Color not found");
18    }
19
20    // Java 5 enum. try/catch is required for colo
21    try {
22     switch (Color.valueOf(color)) {
23     case RED:
24      System.out.println("Color is Red");
25      break;
26     case GREEN:
27      System.out.println("Color is Green");
28     }
29    } catch (IllegalArgumentException e) {
30     System.out.println("Color not found");
31    }
32
33    // Java 7 String in switch statement for simpl
34    //JDK 7 switch performs better than if-else
35    //using types with enums is only useful when i
36    //the value for color could come from database
37    switch (color) {
38    case "RED":
39     System.out.println("Color is Red");
40     break;
41    case "GREEN":
42     System.out.println("Color is Green");
43     break;
44    default:
45     System.out.println("Color not found");
46    }
47   }
48
49 }
50
51
52
```

**Output is:**

```
1 Color not found
2 Color not found
3 Color not found
4
```

## #2 Binary integral literals

```
1  public class Java7Feature2 {
2
3   public static void main(String[] args) {
4    // Pre Java 7
```



## 16 Technical Key Areas

```
5    int n = Integer.parseInt("10000000", 2);
6    System.out.println(n);
7
8    n = 1 << 7;
9    System.out.println(n);
10
11   // Java 7
12   n = 0b10000000; // 128 = 2^7
13   System.out.println(n);
14  }
15 }
16
```

**Output:**

```
1 128
2 128
3 128
4
```

### #3: Underscores for better readability in numeric literals

```
1  public class Java7Feature3 {
2
3   public static void main(String[] args) {
4           //pre Java 7
5     int million = 1000000;
6     System.out.println(million);
7
8     //Java 7. More readable
9     million = 1_000_000;
10    System.out.println(million);
11
12    //consecutive underscores are allowed
13    int ten_million = 10__000_000;
14    System.out.println(ten_million);
15
16    //underscores can be used in other numeric ty
17    double million_dollars_5_cents = 1_000_000.0_
18    System.out.println(million_dollars_5_cents);
19
20    //illegal to have underscores
21    //1. start or end a literal with an underscor
22    //2. have underscores before or after a decim
23  }
24 }
25
```

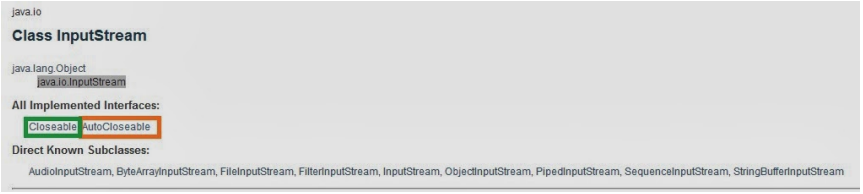
**Output**:

```
1 1000000
2 1000000
```

```
3 10000000
4 1000000.05
5
```

### **#4**: **AutoCloseable** interface.

Java 5 introduced the *Closeable* interface and Java 7 has introduced the *AutoCloseable* interface to avoid the unsightly try/catch/finally(within finally try/catch) blocks to close a resource. It also prevents potential resource leaks due to not properly closing a resource. The *java.io.InputStream* and java.io.OutputStream now implements the *AutoCloseable* interface.




try-with-resources is one of the most useful additions in Java 7.

```
1  import java.io.BufferedReader;
2  import java.io.File;
3  import java.io.FileInputStream;
4  import java.io.IOException;
5  import java.io.InputStream;
6  import java.io.InputStreamReader;
7
8  public class Java7Feature4 {
9
10  public static void main(String[] args) {
11   // pre Java 7
12   BufferedReader br = null;
13
14   try {
15    File f = new File("c://temp/simple.txt");
16    InputStream is = new FileInputStream(f);
17    InputStreamReader isr = new InputStreamReader
18    br = new BufferedReader(isr);
19
20    String read;
21
22    while ((read = br.readLine()) != null) {
23     System.out.println(read);
24    }
25   } catch (IOException ioe) {
26    ioe.printStackTrace();
27   } finally {
28    try {
29     if (br != null)
30      br.close();
31    } catch (IOException ex) {
```

```
32      ex.printStackTrace();
33     }
34    }
35
36
37    // Java 7 -- more concise 11 lines as opposed
38    try (InputStream is = new FileInputStream(new
39      InputStreamReader isr = new InputStreamReade
40      BufferedReader br2 = new BufferedReader(isr)
41
42     String read;
43
44     while ((read = br2.readLine()) != null) {
45      System.out.println(read);
46     }
47
48    }
49    catch (IOException ioe) {
50     ioe.printStackTrace();
51    }
52
53   }
54
55 }
56
```

The output is:

```
1 Big
2 brown fox
3 jumped over the fence
4 Big
5 brown fox
6 jumped over the fence
7
```

try can now have multiple statements in the parenthesis and each statement should create an object which implements the new java.lang.***AutoCloseable*** interface. The ***AutoCloseable*** interface consists of just one method.
void close () throws Exception {}. Each *AutoClosable* resource created in the try statement will be automatically closed! If an exception is thrown in the try block and another *Exception* is thrown while closing the resource, the first *Exception* is the one eventually thrown to the caller.

Think of the close( ) method as implicitly being called as the last line in the try block.

## #5 Multi-catch to avoid code duplication

```
1  public class Java7Feature4 {
2
3   public static void main(String[] args) {
4
5    //pre Java 7
6    try {
7     someMethod();
8    } catch (CustomException1 ex1) {
9     ex1.printStackTrace();
10   } catch (CustomException2 ex2) {
11    ex2.printStackTrace();
12   }
13
14
15   //Java 7 -- 5 lines as opposed to 7 lines.
16   //no code duplication
17   try {
18    someMethod();
19   } catch (CustomException1|CustomException2 ex)
20    ex.printStackTrace();
21   }
22  }
23
24  public static void someMethod() throws CustomEx
25
26  }
27
28  public static class CustomException1 extends Ex
29   private static final long serialVersionUID = 1
30  }
31
32  public static class CustomException2 extends Ex
33   private static final long serialVersionUID = 1
34  }
35
36 }
37
```

Note that the pipe '|' character is used as the delimiter.

### #6 Improved type inference for generic instance creation

This is only a small change that makes generics declaration a little less verbose. As shown below, you can just use empty diamond "<>" in Java 7 on the RHS.

```
1  import java.util.Collections;
2  import java.util.HashMap;
3  import java.util.List;
4  import java.util.Map;
5
6  public class Java7Feature4 {
7
8   public static void main(String[] args) {
```

```
9          //Pre Java 7
10    getEmployeesWithManagersOld("a102");
11    //Java 7
12    getEmployeesWithManagersNew("a102");
13   }
14
15
16   public static Map<String, List<Employee>>  getE
17     if(empCode == null){
18        return Collections.emptyMap();
19     }
20
21     //gives type safety warning. You need to add
22     Map<String, List<Employee>> mapEmployees = ne
23
24
25     return mapEmployees;
26   }
27
28
29   //Java 7
30   public static Map<String, List<Employee>>  getE
31     if(empCode == null){
32        return Collections.emptyMap();
33     }
34
35     //no duplication of generic inference
36     Map<String, List<Employee>> mapEmployees = ne
37     //do something with mapEmployees
38
39     return mapEmployees;
40   }
41
42
43   static class Employee {}
44  }
45
46
```

### #7: More new I/O APIs for the Java platform (NIO – 2.0)

Those who worked with Java IO may still remember the headaches that framework caused. It was never easy to work seamlessly across operating systems or multi-file systems. The NIO 2.0 has come forward with many enhancements. It's also introduced new classes to ease the life of a developer when working with multiple file systems with classes and interfaces such as ***Path***, ***Paths***, ***FileSystem***, ***FileSystems*** and others.

Another very handy feature is the *WatchService* for file change notifications. It can monitor a directory for changes as demonstrated below.

```
1  import java.io.IOException;
2  import java.nio.file.FileSystems;
3  import java.nio.file.Files;
4  import java.nio.file.Path;
5  import java.nio.file.Paths;
6  import java.nio.file.StandardWatchEventKinds;
7  import java.nio.file.WatchEvent;
8  import java.nio.file.WatchEvent.Kind;
9  import java.nio.file.WatchKey;
10 import java.nio.file.WatchService;
11
12 public class Java7Feature7 {
13
14  public static void main(String[] args) throws I
15
16   // Java 7
17   Path path = Paths.get("c:\\Temp\\simple.txt");
18   System.out.println(path.getFileName());
19   System.out.println(path.getRoot());
20   System.out.println(path.getParent());
21
22   // Java 7 file change watch service
23   WatchService watchService = FileSystems.getDef
24
25   //register Temp folder with the watch service
26   path.getParent().register(watchService, Standa
27     StandardWatchEventKinds.ENTRY_MODIFY, Standa
28
29   //wait for incoming events
30   while (true) {
31    final WatchKey key = watchService.take();
32    for (WatchEvent<?> watchEvent : key.pollEvent
33     final Kind<?> kind = watchEvent.kind();
34     // Overflow event
35     if (StandardWatchEventKinds.OVERFLOW == kind
36      continue; // loop
37     } else if (StandardWatchEventKinds.ENTRY_CRE
38       || StandardWatchEventKinds.ENTRY_DELETE ==
39      @SuppressWarnings("unchecked")
40      final WatchEvent<Path> watchEventPath = (Wa
41      final Path entry = watchEventPath.context()
42
43      System.out.println(kind + "-->" + entry);
44
45     }
46
47    }
48
49    if (!key.reset()) {
50     break;
51    }
52   }
53
54   // deleting a file is as easy as.
55   Files.deleteIfExists(path); // Java 7 feature
56  }
57
58 }
59
60
```

The output will be something like:

```
simple.txt
c:\
c:\Temp
ENTRY_CREATE-->New Text Document.txt
ENTRY_DELETE-->New Text Document.txt
ENTRY_CREATE-->File1.txt
ENTRY_MODIFY-->File1.txt


```

### #8: Fork and Join

Java 7 has incorporated the feature that would distribute the work across multiple cores and then join them to return the result set as a Fork and Join framework. he effective use of parallel cores in a Java program has always been a challenge. It's a divide-and-conquer algorithm where Fork-Join breaks the task at hand into mini-tasks until the mini-task is simple enough that it can be solved without further breakups. One important concept to note in this framework is that ideally no worker thread is idle. They implement a work-stealing algorithm in that idle workers "steal" the work from those workers who are busy.

The example below demonstrates this with a simple task of summing up 10 numbers. If the count of numbers to be added are greater than 5, it is forked into chunks of 5 to be processed by separate thread, and the forked sum are then joined to give the overall total of 10 numbers from 1 to 10, which is 55.  The total of numbers 1 to 5 is 15, and 6 to 10 is 40.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class Java7Feature8 {

 static int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8

 public static void main(String[] args) throws I
     int numberOfCpuCores = Runtime.getRuntime()
     ForkJoinPool forkJoinPool = new ForkJoinPoo
     int sum = forkJoinPool.invoke(new ChunkingT
```

```
16      System.out.println(sum);
17  }
18
19  //inner class
20  static class SumCalculatorTask extends Recursiv
21   int[] numbers;
22
23   SumCalculatorTask(int[] numbers) {
24    this.numbers = numbers;
25   }
26
27   @Override
28   protected Integer compute() {
29    int sum = 0;
30    for (int i : numbers) {
31     sum += i;
32    }
33
34    System.out.println(Thread.currentThread().get
35
36    return sum;
37   }
38  }
39
40
41  //inner class
42
43  /**
44   *
45   *chunking size is 5
46   */
47  static class ChunkingTask extends RecursiveTask
48
49   private static final int CHUNK_SIZE = 5;
50   int[] numbers;
51
52   ChunkingTask(int[] numbers) {
53    this.numbers = numbers;
54   }
55
56   @Override
57   protected Integer compute() {
58    int sum = 0;
59    List<RecursiveTask<Integer>> forks = new Arra
60
61    //if the numbers size is > CHUNK_SIZE fork th
62    if (numbers.length > CHUNK_SIZE) {
63     ChunkingTask chunk1 = new ChunkingTask(Array
64     ChunkingTask chunk2 = new ChunkingTask(Array
65     forks.add(chunk1);
66     forks.add(chunk2);
67     chunk1.fork();
68     chunk2.fork();
69    //size is less than or equal to CHUNK_SIZE st
70    } else {
71     SumCalculatorTask sumCalculatorTask = new Su
72     forks.add(sumCalculatorTask);
73     sumCalculatorTask.fork();
74    }
75
76    // Combine the result from all the tasks
77    //join
78    for (RecursiveTask<Integer> task : forks) {
79     sum += task.join();
80    }
81
```

```
82     return sum;
83    }
84
85   }
86
87 }
88
```

**Output is:**

```
1 ForkJoinPool-1-worker-2 sum = 15
2 ForkJoinPool-1-worker-2 sum = 40
3 55
4
```

Java 8's *Arrays.parallelSort( … )* make use of this fork and join feature to sort an array in parallel.

# Popular Posts

♦ 11 Spring boot interview questions & answers

**825 views**

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

**765 views**

18 Java scenarios based interview Questions and Answers

**399 views**

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

**388 views**

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

**295 views**

♦ 7 Java debugging interview questions & answers

**293 views**

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

**285 views**

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

**279 views**

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

**239 views**

001B: ♦ Java architecture & design concepts interview questions & answers

**201 views**

Bio | Latest Posts

## Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

**About** Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Java 8: Different ways to sort a collection of objects in pre and post Java 8

SSL and truststore vs keystore for Java developers ›

**Posted in** Java 7, member-paid

**Tags:** Novice FAQs, TopX

# Leave a Reply

Logged in as geethika. Log out?

**Comment**

Post Comment

# Empowers you to open more doors, and fast-track

### Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

### Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

# Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

## © Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.



↑