Java 8: 7 useful miscellaneous additions you must know

Posted on November 8, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

The much awaited addition in Java 8 are Lambda expressions to be used in **functional programming** and default and static methods in interfaces to support multiple behavior inheritance and helper methods. Here are a few other additional handy additions in Java 8.

Addition #1: ***String.join( )*** method, which is an opposite of String.split(…) was there from pre Java 8.

Example 1:

```
String[] javaTechnologies = {"Java", "JEE", "JDBC
String joined = String.join(",", javaTechnologies

```

Example 2:

```
 String joined = String.join("/", "var", "opt", "

```

Addition #2: The ***Comparator*** interfaces have a number of useful methods to sort objects with options to nest multiple fields with ***thenComparing****(…)*, and other handy methods like ***nullsFirst****( )*, ***nullsLast****( )*, ***naturalOrder****( )*, ***reverseOrder****( )*, etc. This was made possible as Java 8 interfaces can have default methods (which gives you multiple behavior inheritance) and static methods (a replacement for helper classes).

Example 1:

```
Arrays.sort(people, Comparator.comparing(Person::
                                .thenComparing(Per

```

Example 2:

```
Comparator<Person> multiFieldComparator =
    Comparator.comparing(Person::getGender, Compa
              .thenComparing(Person::getName, Com
              .thenComparing(Person::getAge,  Com

//parallel() processing using Fork/Join
List<Object> sortedPeople = people.stream().paral

```

Addition #3: In Java you cannot simply get rid of the null references that have historically existed, but before Java 8, you need to rely on your code and proper documentation to understand if a reference is **optional** (i.e can be null) as shown below,

```

```

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class PersonTest {
5
6   public static void main(String[] args) {
7
8    List<Person> people = new ArrayList<>();
9    people.add(new Person("John", 35, Person.Gende
10   people.add(new Person("Simone", 30, Person.Gen
11   people.add(new Person("Shawn", 30, Person.Gend
12
13   Person foundPerson = find("Sam", people);
14   System.out.println(foundPerson.getName()); //N
15
16  }
17
18  public static Person find(String name, List<Per
19   for (Person person : Persons) {
20    if (person.getName().equalsIgnoreCase(name))
21     return person;
22    }
23   }
24   return null;
25  }
26 }
27
```

In Java 8, java.util.**Optional** class has been added to deal with optional object references. The intention with the ***Optional*** class is not to replace every null-able reference, but to help in the creation of more robust APIs you could tell if you can expect an optional reference by reading the signature of a method.

```
1  import java.util.ArrayList;
2  import java.util.List;
3  import java.util.Optional;
4
5  public class PersonTest {
6
7   public static void main(String[] args) {
8
9    List<Person> people = new ArrayList<>();
10   people.add(new Person("John", 35, Person.Gende
11   people.add(new Person("Simone", 30, Person.Gen
12   people.add(new Person("Shawn", 30, Person.Gend
13
14   Optional<Person> foundPerson = find("Sam", peo
15   System.out.println(foundPerson.orElse(new Pers
16
17  }
18
```

```
19  public static Optional<Person> find(String name
20   for (Person Person : Persons) {
21    if (Person.getName().equals(name)) {
22     return Optional.of(Person);
23    }
24   }
25   return Optional.empty();
26  }
27 }
28
```

The find method signature tells you that it returns a person optionally.

Addition #4: In cases where you want to detect result overflow errors in int and long, the methods ***addExact***, ***subtractExact***, ***multiplyExact***, and ***toIntExact*** in Java 8, the *java.lang.**Math*** class throws an *ArithmeticException* when the results overflow. Generally, it is recommended to use *BigInteger* if you expect large values.

```
1  public class DataOverflowTest {
2
3   public static void main(String[] args) {
4
5    //for int -2^31 and a maximum value of 2^31-1,
6
7    int i = 2147483647 + 1;
8    System.out.println(i); //wrong -- prints -2147
9
10   //Java 8
11   Math.addExact(2147483647, 1);  //java.lang.Ari
12  }
13 }
14
```

Addition #5: The ability to lazily read lines from a file with **Files.lines(…)**. For example, let's read ***technologies.txt*** that has

Java
JEE
JDBC
Hibernate
Spring

Now the code that covers previous additions as well.

```
```

```
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Optional;
import java.util.stream.CloseableStream;

public class FileReadTest {

 public static void main(String[] args) {
  Path path = Paths.get("c:\\temp\\technologies.
  try(CloseableStream<String> lines = Files.line
   Optional<String> found = lines.filter(p -> p.
   System.out.println(found.orElse("NOT FOUND!!"
  } catch (IOException e) {
   e.printStackTrace();
  }//CloseableStream is AutoCloseable
 }
}

```

Addition #6: The **@Repeatable** annotation.

Pre Java 8:

```
public @interface Habit {
 String value();
}

```

```
public @interface Habits {
     Habit[] value();
}

```

```
@Habits({@Habit("habit1"), @Habit("habit2")})
public class Person  {
   //......
}

```

Post Java 8:

```
import java.lang.annotation.Repeatable;

@Repeatable(Habits.class)
public @interface Habit {
 String value();
}


```

```
1 @Habit("habit1")
2 @Habit("habit2")
3 public class Person  {
4    //...
5 }
6
```

In Pre Java 8, if you repeat the same annotation, you will get a compile-time error “Duplicate annotation”. In Java 8, you can repeat the same annotation with the @Repeatable annotation.

Addition #7: Java 8 introduces the “**Type Annotations**“, which are annotations that can be placed anywhere you use a type.

- new operator,
- type casts,
- implements clauses and
- throws clauses.

Handy for stronger type checking, but can clutter your code. Use them judiciously. For example,

```
1 @NotNull String str1 = ...
2 @Email String str2 = ...
3 Map.@NonNull Entry = ...
4 new @NonEmpty List<String>(notEmptySet)
5 List<@ReadOnly @Localized Message> messages = ...
6
```

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

400 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**

‹ Why does good API design matter?

Java 8: What is currying? Does Java 8 support currying? ›

Posted in Java 8, member-paid

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

 ↑