

Industrial strength Java/JEE Career Companion to open more doors


[Home](#)
[Java FAQs](#)
[600+ Java Q&As](#)
[Career](#)
[Tutorials](#)
[Member](#)
[Why?](#)
[Can u Debug?](#)
[Java 8 ready?](#)
[Top X](#)
[Productivity Tools](#)
[Judging Experience?](#)

[Home](#) › [Interview](#) › [Core Java Interview Q&A](#) › [Java 8](#) › Java 8: 7 useful miscellaneous additions you must know

## Java 8: 7 useful miscellaneous additions you must know

Posted on [November 8, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — No

[Comments](#) ↓

0  
Like  
Share

Tweet

0  
G+1  
Share

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:**

**600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs**

[open all](#) | [close all](#)

- [+ Ice Breaker Interview](#)
- [+ Core Java Interview C](#)
- [+ Java Overview \(4\)](#)
- [+ Data types \(6\)](#)
- [+ constructors-methc](#)
- [+ Reserved Key Wor](#)
- [+ Classes \(3\)](#)
- [+ Objects \(8\)](#)
- [+ OOP \(10\)](#)
- [+ GC \(2\)](#)
- [+ Generics \(5\)](#)
- [+ FP \(8\)](#)
- [+ IO \(7\)](#)
- [+ Multithreading \(12\)](#)
- [+ Algorithms \(5\)](#)
- [+ Annotations \(2\)](#)
- [+ Collection and Dat](#)
- [+ Differences Between](#)
- [+ Event Driven Progr](#)
- [+ Exceptions \(2\)](#)
- [+ Java 7 \(2\)](#)

```

1 String[] javaTechnologies = {"Java", "JEE", "JDBC"}
2 String joined = String.join(", ", javaTechnologies)
3

```

### Example 2:

```

1 String joined = String.join("/", "var", "opt", " ")
2

```

**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:

```

1 Arrays.sort(people, Comparator.comparing(Person::
2                                     .thenComparing(Per
3

```

### Example 2:

```

1 Comparator<Person> multiFieldComparator =
2     Comparator.comparing(Person::getGender, Compa
3                     .thenComparing(Person::getName, Com
4                     .thenComparing(Person::getAge, Com
5
6 //parallel() processing using Fork/Join
7 List<Object> sortedPeople = people.stream().paral
8

```

**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,

## Java 8 (24)

- 01: ♦ 19 Java 8 I
- 02: ♦ Java 8 Stre
- 03: ♦ Functional
- 04: ♥♦ Top 6 tips
- 04: Convert Lists
- 04: Understandir
- 05: ♥ 7 Java FP
- 05: ♦ Finding the
- 06: ♥ Java 8 way
- 07: ♦ Java 8 API
- 08: ♦ Write code
- 10: ♦ Executors
- Fibonacci numb
- Java 8 String str
- Java 8 using the
- Java 8: 7 useful
- Java 8: Different
- Java 8: Does "O
- Java 8: What is
- Learning to write
- Non-trivial Java 8
- Top 6 Java 8 fea
- Top 8 Java 8 fea
- Understanding J

## JVM (6)

## Reactive Programn

## Swing & AWT (2)

## JEE Interview Q&A (3)

## Pressed for time? Jav

## SQL, XML, UML, JSC

## Hadoop & BigData Int

## Java Architecture Inte

## Scala Interview Q&As

## Spring, Hibernate, & I

## Testing & Profiling/Sa

## Other Interview Q&A 1

## Free Java Interview

```

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.Gender.MALE));
10        people.add(new Person("Simone", 30, Person.Gender.FEMALE));
11        people.add(new Person("Shawn", 30, Person.Gender.MALE));
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<Person> people) {
19        for (Person person : people) {
20            if (person.getName().equalsIgnoreCase(name))
21                return person;
22        }
23    }
24    return null;
25 }
26 }
27

```

## 16 Technical Key Areas

[open all](#) | [close all](#)

- [Best Practice \(6\)](#)
- [Coding \(26\)](#)
- [Concurrency \(6\)](#)
- [Design Concepts \(7\)](#)
- [Design Patterns \(11\)](#)
- [Exception Handling \(3\)](#)
- [Java Debugging \(21\)](#)
- [Judging Experience Level \(1\)](#)
- [Low Latency \(7\)](#)
- [Memory Management \(1\)](#)
- [Performance \(13\)](#)
- [QoS \(8\)](#)
- [Scalability \(4\)](#)
- [SDLC \(6\)](#)
- [Security \(13\)](#)
- [Transaction Management \(1\)](#)

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.Gender.MALE));
11        people.add(new Person("Simone", 30, Person.Gender.FEMALE));
12        people.add(new Person("Shawn", 30, Person.Gender.MALE));
13
14        Optional<Person> foundPerson = find("Sam", people);
15        System.out.println(foundPerson.orElse(new Person("N/A", 0, Person.Gender.UNKNOWN)));
16
17    }
18

```

## 80+ step by step Java Tutorials

[open all](#) | [close all](#)

- [Setting up Tutorial \(6\)](#)
- [Tutorial - Diagnosis \(2\)](#)
- [Akka Tutorial \(9\)](#)
- [Core Java Tutorials \(2\)](#)
- [Hadoop & Spark Tutorial \(1\)](#)
- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate Tutorial \(1\)](#)
- [Tools Tutorials \(19\)](#)
- [Other Tutorials \(45\)](#)

## 100+ Java pre-interview

```

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.

## coding tests

[open all](#) | [close all](#)

- [Can you write code? \(](#)
- [♦ Complete the given](#)
- [Converting from A to I](#)
- [Designing your classe](#)
- [Java Data Structures](#)
- [Passing the unit tests](#)
- [What is wrong with th](#)
- [Writing Code Home A](#)
- [Written Test Core Jav](#)
- [Written Test JEE \(1\)](#)

## How good are your .....?

[open all](#) | [close all](#)

- [Career Making Know-](#)
- [Job Hunting & Resum](#)

```

1 import java.io.IOException;
2 import java.nio.charset.Charset;
3 import java.nio.file.Files;
4 import java.nio.file.Path;
5 import java.nio.file.Paths;
6 import java.util.Optional;
7 import java.util.stream.CloseableStream;
8
9 public class FileReadTest {
10
11     public static void main(String[] args) {
12         Path path = Paths.get("c:\\temp\\technologies.
13         try(CloseableStream<String> lines = Files.line
14             Optional<String> found = lines.filter(p -> p.
15             System.out.println(found.orElse("NOT FOUND!!"
16         } catch (IOException e) {
17             e.printStackTrace();
18         } //CloseableStream is AutoCloseable
19     }
20 }
21

```

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

### Pre Java 8:

```

1 public @interface Habit {
2     String value();
3 }
4

```

```

1 public @interface Habits {
2     Habit[] value();
3 }
4

```

```

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

```

### Post Java 8:

```

1 import java.lang.annotation.Repeatable;
2
3 @Repeatable(Habits.class)
4 public @interface Habit {
5     String value();
6 }
7
8

```

```
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 @NotEmpty 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](https://www.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](https://www.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

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.