

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](#) › [Tech Key Areas](#) › [13 Technical Key Areas Interview Q&A](#) › [Best Practice](#) ›

Top 5 Core Java Exceptions and best practices

Top 5 Core Java Exceptions and best practices

Posted on [December 18, 2014](#) by [Arulkumaran Kumaraswamipillai](#) — 2

[Comments](#) ↓

0
Like
Share

Tweet

0
G+1
Share

#1: ConcurrentModificationException

Getting a **ConcurrentModificationException** when trying to modify (i.e. adding or removing an item) a collection while iterating. The following code throws a **ConcurrentModificationException**.

```
1 List<T> list = getListOfItems();
2 for (Iterator<T> iter = list.iterator(); iter.hasNext(); iter.next()) {
3     T obj = iter.next();
4     if (obj.someCondition()) {
5         list.remove(0); // ConcurrentModificationException
6     }
7 }
```

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

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

- ✚ Ice Breaker Interview
- ✚ Core Java Interview Questions
- ✚ Java Overview (4)
- ✚ Data types (6)
- ✚ constructors-methods
- ✚ Reserved Key Words
- ✚ Classes (3)
- ✚ Objects (8)
- ✚ OOP (10)
- ✚ GC (2)
- ✚ Generics (5)
- ✚ FP (8)
- ✚ IO (7)
- ✚ Multithreading (12)
- ✚ Algorithms (5)
- ✚ Annotations (2)
- ✚ Collection and Data Structures
- ✚ Differences Between
- ✚ Event Driven Programming
- ✚ Exceptions (2)
- ✚ Java exception

To avoid `ConcurrentModificationException` in a **single-threaded environment**, you can remove the object that you are working on with the iterator.

```

1 List<T> list = getListOfItems();
2 for (Iterator<T> iter = list.iterator(); iter.hasNext(); ) {
3     T obj = iter.next();
4     if (obj.someCondition()) {
5         iter.remove(); //OK to use the iterator
6     }
7 }

```

To Avoid `ConcurrentModificationException` in a **multi-threaded environment**:

If you are using JDK1.5 or higher then you can use **`ConcurrentHashMap`** and **`CopyOnWriteArrayList`** classes. This is the recommended approach compared to other approaches like locking the list with `synchronized(list)` while iterating, but this approach defeats the purpose of using multi-threading.

It is a **best practice** to assume that your code is going to be executed in a multi-threaded environment, hence favor using the **`java.util.concurrent`** package.

Note: Iterators returned by most of pre JDK1.5 collection classes like `Vector`, `ArrayList`, `HashSet`, etc are **fail-fast iterators**. Iterators returned by JDK 1.5+ `ConcurrentHashMap` and `CopyOnWriteArrayList` classes are **fail-safe iterators**.

#2. `NoClassDefFoundError`, `ClassNotFoundException`, `NoSuchMethodError` and `IllegalArgumentException`

Static class loading throws “`NoClassDefFoundError`” if the class is not found and the dynamic class loading throws “`ClassNotFoundException`” if the class is not found. “`ClassNotFoundException`” could be quite tricky to troubleshoot. When you get a `ClassNotFoundException`, it

Top 5 Core Java

- [Java 7 \(2\)](#)
- [Java 8 \(24\)](#)
- [JVM \(6\)](#)
- [Reactive Programming \(2\)](#)
- [Swing & AWT \(2\)](#)
- [JEE Interview Q&A \(3\)](#)
- [Pressed for time? Java Interview Q&A \(1\)](#)
- [SQL, XML, UML, JSC](#)
- [Hadoop & BigData Interview Q&A \(1\)](#)
- [Java Architecture Interview Q&A \(1\)](#)
- [Scala Interview Q&As \(1\)](#)
- [Spring, Hibernate, & JPA Interview Q&A \(1\)](#)
- [Testing & Profiling/Selenium Interview Q&A \(1\)](#)
- [Other Interview Q&A \(1\)](#)
- [Free Java Interview Questions \(1\)](#)

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 Interview Questions \(1\)](#)
- [Low Latency \(7\)](#)
- [Memory Management \(1\)](#)
- [Performance \(13\)](#)
- [QoS \(8\)](#)
- [Scalability \(4\)](#)
- [SDLC \(6\)](#)
- [Security \(13\)](#)
- [Transaction Management \(1\)](#)

means the JVM has traversed the entire classpath and not found the class you've attempted to reference. The JAR hell issues also lead to exceptions like `NoSuchMethodError` or `IllegalArgumentException`. These are discussed in detail in the links shown below.

- [Java class loading interview Q&A to ascertain your depth of Java knowledge](#) .
- [Debugging JAR hell issues in Java](#).

#3. NullPointerException

`NullPointerException` is thrown when an application attempts to use an object reference, having the null value. Careless use of null can cause a variety of bugs. null is very ambiguous as it's rarely obvious what a null return value is supposed to mean. For example, `Map.get(key)` can return null either because the value in the map is null, or the value is not in the map. So, Null can mean failure, can mean success, can mean optional value, can mean broken code, etc.

a) Many of the cases where programmers use null is to indicate some sort of optional value.

If you are using Java 8, then start using the "Optional" class. [Java 8: Does "Optional" class alleviate the pain of ubiquitous NullPointerException?](#)

b) It also means broken code.

In an entry to a public method that takes parameters, perform pre condition check and fail fast by throwing exceptions early. Best practice is to throw exceptions at the point when the errors occur so that you have the most detail about the cause of the exception. For example, you want to know the line that throws the `NullPointerException` so that you can figure out which variable is null. `NullPointerException` means you have some broken code, and you need to fix your code. A good example for throwing early would be to throw `IllegalArgumentException`.

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 Tuto](#)
- [JEE Tutorials \(19\)](#)
- [Scala Tutorials \(1\)](#)
- [Spring & Hibernate Ti](#)
- [Tools Tutorials \(19\)](#)
- [Other Tutorials \(45\)](#)

100+ Java pre-interview 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 public void someMethod(String input){
2     if(StringUtils.isEmpty(input)){
3         throw new IllegalArgumentException("input c
4     }
5
6     //.....do something
7 }
8
```

c) There are times when null is the right and correct thing to use

Especially, if you are using pre Java 8, and want to indicate something is Optional or not set yet.

Null best practices

- 1) If Using Java 8, use the [Optional](#) class.
- 2) Return an empty collection (e.g. `Collections.emptyList()`) from a method as opposed to null.
- 3) When using an enum, add a constant to mean whatever you're expecting null to mean. For example, `CashForecastType.NOTREQUIRED`. So, define a reasonable default value.
- 4) Avoid using null values in a Set or as a key in a Map.
- 5) If you want to use null as a value in a Map then leave out that entry and keep a separate Set of null keys.
- 6) Prefer using null friendly API methods.
 - a) Favor `valueOf()` over `toString()` where passing null to `valueOf()` returns null instead of `NullPointerException`.
 - b) There are open source libraries out there like Apache Commons `StringUtils.isBlank()`, Google `Guava` libraries, etc.
 - c) Use some methods to assure that null value not exist like `contains()`, `indexOf()`, `isEmpty()`, `containsKey()`, `containsValue()`, etc.
- 7) Changing the way you code.

Example 1:

Instead of: (throws `NullPointerException` if emp is null when trying to evaluate `emp.getFirstname()`)

```
1  if(emp != null & emp.getFirstname() != null) {  
2  
3  }
```

Use, short circuit &&. emp.getFirstname() is only executed if emp != null.

```
1  if(emp != null && emp.getFirstname() != null) {  
2  
3  }
```

Example 2:

Instead of: (throws NullPointerException if status is null)

```
1  private Boolean isFinished(String status) {  
2      if (status.equalsIgnoreCase("Complete")) {  
3          return Boolean.TRUE;  
4      } else {  
5          return Boolean.FALSE;  
6      }  
7  }  
8
```

code defensively as shown below:

```
1  private Boolean isFinished(String status) {  
2      if ("Complete".equalsIgnoreCase(status))  
3          return Boolean.TRUE;  
4      } else {  
5          return Boolean.FALSE;  
6      }  
7  }  
8
```

8) Use of annotation @NotNull and @Nullable to define the contracts about null values, and modern IDEs like IntelliJ, and code quality tools like Sonar can read this annotation and assist you to put a missing null check, or inform you about an redundant null check.

Example 3: Autoboxing or Autounboxing can cause NullPointerException

```
1  boolean status = false;
```

```
2 Double val1 = 0d;  
3 Double val2 = null;  
4 Double result = status ? val1.doubleValue() : val2;
```

The return type of the conditional expression “status ? val1.doubleValue() : val2;” is double. So, val2 will need to unbox to double by invoking val2.doubleValue(). Since, val2 is null, it throws a NullPointerException.

#4. ArrayIndexOutOfBoundsException

The most common case is to declare an array with size “n” and access the nth element instead of n-1. The array indices are 0 based.

```
1 char[] characters = new char[5];  
2 characters[5] = '\n'; // wrong  
3 characters[4] = '\n'; //right
```

```
1 for(int i = 0; i <= characters.length; ++i) {  
2 for(int i = 0; i < characters.length; ++i) {  
3 }
```

#5. NumberFormatException and ClassCastException

1. NumberFormatException is thrown when a method that converts a String to a number receives a String that it cannot convert.

So, to prevent NumberFormatException “check before parsing” or handle Exception properly.

```
1 try{  
2     int i = Integer.parseInt(input);  
3 }catch(NumberFormatException ex){ // handle your  
4     //.....  
5 }  
6
```

or validate before parsing

```
1 String input="string";
2 String pattern = "-?\\d+";
3 if(input.matches("-?\\d+")){ // any positive or n
4     ...
5 }
```

2. ClassCastException is thrown when attempting to cast a reference variable to a type that fails the IS-A test. Generics let you avoid casting in many cases, and if you don't cast, you can't get a ClassCastException. The main aim of Generics was to reduce the ClassCastException at runtime.

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](#)

◀ Part-3: Java Tree structure interview and coding questions

Why Java-Success.com training? ▶

Posted in Best Practice, Exception Handling, Exceptions, member-paid

Tags: TopX

2 comments on “Top 5 Core Java Exceptions and best practices”



Arulkumaran Kumaraswamipillai says:

November 5, 2015 at 9:01 am

Well spotted. Correcting the "<=" to "<" to avoid `java.lang.ArrayIndexOutOfBoundsException`.

[Reply](#)



sen says:

November 5, 2015 at 5:25 am

Guess there is a typo in the second line.

```
for(int i = 0; i <= characters.length; ++i) { // wrong
for(int i = 0; i < characters.length; ++i) { // right, "<="
```

[Reply](#)

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.