

Java-Success.com

Industrial strength Java/JEE Career Companion for those who want to go places

[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 Overview](#) › 04: ♦ Top 10 most common Core Java beginner mistakes

04: ♦ Top 10 most common Core Java beginner mistakes

Posted on November 28, 2014 by Arulkumaran Kumaraswamipillai — 2

[Comments](#) ↓

Mistake #1: Using floating point data types like float or double for monetary calculations. This can lead to rounding issues.

```
1 package com.monetary;
2 import java.math.BigDecimal;
3
4 public class MonetaryCalc {
5
6     public static void main(String[] args) {
7         System.out.println(1.05 - 0.42); //1: $0.
8         //2: $0.63
9         System.out.println(new BigDecimal("1.05")
10         //3: $0.6300000000000000599520433297584531
11         System.out.println(new BigDecimal(1.05) .s
12         //4: $0.63
13         System.out.println(BigDecimal.valueOf(1.05
14         System.out.println(105 - 42); //5: 63 cent
```

[9 tips to earn more](#) | [What can u do to go places?](#) | **945+** members. [LinkedIn Group](#). [Reviews](#)

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

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

 [Ice Breaker Interview](#)

└─ 01: ♦ 15 Ice breake

└─ 02: ♥♦ 8 real life ex

└─ 03: ♦10+ Know you

└─ 04: Can you think c

└─ 05: ♥ What job inte

└─ 06: ► Tell me abou

└─ 07: ♥ 20+ Pre inter

 [Core Java Interview C](#)

 [Java Overview \(4\)](#)

└─ 01: ♦ ♥ 17 Java c

```

15     }
16 }
17

```

In the above code, 2, 4, and 5 are correct and 1 and 3 are incorrect usage leading to rounding issues. So, either use **BigDecimal** properly as shown in 2 and 4, or use the smallest monetary value like cents with data type long. The cents approach performs better, and handy in applications that have heavy monetary calculations.

Mistake #2: Using floating point variables like float or double in loops to compare for **equality**. This can lead to infinite loops.

```

1 public static void main(String[] args) {
2     float sum = 0;
3     while (sum != 1.0) { //causes infinite loop
4         sum += 0.1;
5     }
6     System.out.print("The sum is: "+sum);
7 }
8

```

Fix is to change while (sum != 1.0) to while (sum < 1.0). Even then, due to mistake #1, you will get rounding issues (e.g. The sum is: 1.0000001). So, the better fix is

```

1 public static void main(String[] args) {
2     BigDecimal sum = BigDecimal.ZERO;
3     while (sum.compareTo(BigDecimal.ONE) != 0) {
4         sum = sum.add(BigDecimal.valueOf(0.1));
5     }
6     System.out.print("The sum is: "+sum); //
7 }
8

```

Mistake #3: Not properly implementing the **equals(..)** and **hashCode()** methods. Can you pick if anything wrong with the following *Person* class.

```

1 public class Person {
2
3     private String name;
4     private Integer age;
5

```

02: ♥♦ Java Con

03: ♦ 9 Core Jav

04: ♦ Top 10 mos

☐ Data types (6)

01: Java data typ

02: ♥♦ 10 Java S

03: ♦ ♥ Java aut

04: Understandir

05: Java primitiv

Working with Da

☐ constructors-methc

Java initializers,

☐ Reserved Key Wor

♥♦ 6 Java Modifi

Java identifiers

☐ Classes (3)

♦ Java abstract c

♦ Java class load

♦ Java classes a

☐ Objects (8)

► Beginner Jav

♥♦ HashMap & H

♦ 5 Java Object

♦ Java enum inte

♦ Java immutabl

♥♦ Object equals

Java serialization

Mocks, stubs, dc

☐ OOP (10)

♥ Design princip

♦ 30+ FAQ Java

♦ Why favor cor

08: ♦ Write code

Explain abstracti

How to create a

Top 5 OOPs tips

Top 6 tips to go a

Understanding C

What are good r

☐ GC (2)

♦ Java Garbage

```

6  public Person(String name, Integer age) {
7      this.name = name;
8      this.age = age;
9  }
10
11 //getters and setters
12
13 @Override
14 public String toString() {
15     return "Person [name=" + name + ", age=" + age
16 }
17 }
18

```

Now, the main class that creates a collection of **Person**, and then searches for a person.

```

1  import java.util.Collections;
2  import java.util.Set;
3  import java.util.TreeSet;
4  import java.util.concurrent.CopyOnWriteArraySet;
5
6  public class PersonTest {
7
8      public static void main(String[] args) {
9
10         Set<Person> people = new CopyOnWriteArraySet<>
11         people.add(new Person("John", 35));
12         people.add(new Person("John", 32));
13         people.add(new Person("Simon", 30));
14         people.add(new Person("Shawn", 30));
15
16         System.out.println( people);
17
18         Person search = new Person("John", 35);
19
20         if(people.contains(search)){
21             System.out.println("found: " + search);
22         }
23         else {
24             System.out.println("not found: " + search);
25         }
26     }
27 }
28
29 }
30

```

The output

```

[Person [name=John, age=35], Person [name=John,
age=32], Person [name=Simon, age=30], Person
[name=Shawn, age=30]]
not found: Person [name=John, age=35]

```

03: Java GC tun

Generics (5)

♥ Java Generics

♥ Overloaded m

♦ 12 Java Gener

♦ 7 rules to reme

3 scenarios to ge

FP (8)

01: ♦ 19 Java 8 I

02: ♦ Java 8 Stre

03: ♦ Functional

04: ♥♦ Top 6 tips

05: ♥ 7 Java FP

Fibonacci numb

Java 8 String str

Java 8: What is c

IO (7)

♥ Reading a text

♦ 15 Java old I/C

06: ♥ Java 8 way

Processing large

Processing large

Read a text file f

Reloading config

Multithreading (12)

01: ♥♦ 15 Beginr

02: ♥♦ 10+ Java

03: ♦ More Java

04: ♦ 6 popular J

05: ♦ How a thre

06: ♦ 10+ Atomic

07: 5 Basic multi

08: ♦ ThreadLoc

09: Java FutureT

10: ♦ ExecutorSe

Java ExecutorSe

Producer and Co

Algorithms (5)

♦ Splitting input t

♦ Tree traversal :

♥ ♦ Java coding

Q. Why is the person not found even though there in the collection?

A. Every Java object implicitly extends the *Object* class, which has the default implementation of ***equals(...)*** and ***hashCode()*** methods. The *equals(...)* method is implicitly invoked by the *Set* class's *contains(...)* method. In this case the default implementation in the *Object* class performs a shallow comparison of the references, and not the actual values like ***name*** and ***age***. The *equals* and *hashCode* methods in Java are meant for overridden for POJOs. So, this can be fixed as shown below by overriding the *equals* and *hashCode* methods in ***Person***.

```

1 import org.apache.commons.lang.builder.EqualsBuilder;
2 import org.apache.commons.lang.builder.HashCodeBuilder;
3
4 public class Person {
5
6     private String name;
7     private Integer age;
8
9     public Person(String name, Integer age) {
10         this.name = name;
11         this.age = age;
12     }
13
14     //getters and setters
15
16     @Override
17     public int hashCode() {
18         return new HashCodeBuilder().append(name).append(age).toHashCode();
19     }
20
21     @Override
22     public boolean equals(Object obj) {
23         Person rhs = (Person) obj;
24         return new EqualsBuilder().append(this.name, rhs.name).append(this.age, rhs.age).isEquals();
25     }
26
27     @Override
28     public String toString() {
29         return "Person [name=" + name + ", age=" + age + "]";
30     }
31 }
32
33

```

Now, output will be

[Person [name=John, age=35], Person [name=John, age=32], Person [name=Simon, age=30], Person

[Searching algorithm](#)

[Swapping, partitioning](#)

[Annotations \(2\)](#)

[8 Java Annotations](#)

[More Java annotations](#)

[Collection and Data Structures](#)

[♦ Find the first non-repeating character in a string](#)

[♦ Java Collections Framework](#)

[♥ Java Iterable & Iterator](#)

[♥♦ HashMap & HashSet](#)

[♦ Sorting objects](#)

[02: ♦ Java 8 Stream API](#)

[04: Understanding the Java Collections Framework](#)

[4 Java Collections Framework](#)

[If Java did not have Collections Framework](#)

[Java 8: Different ways to iterate over a collection](#)

[Part-3: Java Tree Traversal](#)

[Sorting a Map by value](#)

[When to use which Collection](#)

[Differences Between ArrayList and LinkedList](#)

[♥ Java Iterable & Iterator](#)

[♦ Multithreading](#)

[♦ Why do Proxy, Decorator, Adapter, Composite, Strategy, Singleton, Factory, Builder, etc. exist?](#)

[Core Java Modifications](#)

[Differences between ArrayList and Vector](#)

[Java Collections Framework](#)

[Event Driven Programming](#)

[Event Driven Programming](#)

[Event Driven Programming](#)

[Exceptions \(2\)](#)

[♦ Java exceptions](#)

[Top 5 Core Java Interview Questions](#)

[Java 7 \(2\)](#)

[Java 7 fork and join](#)

[Java 7: Top 8 new features](#)

[Java 8 \(24\)](#)

[01: ♦ 19 Java 8 Interview Questions](#)

[02: ♦ Java 8 Stream API](#)

[03: ♦ Functional Programming](#)

[04: ♥♦ Top 6 tips for Java 8](#)

[04: Convert Lists to Maps](#)

```
[name=Shawn, age=30]]
```

```
found: Person [name=John, age=35]
```

There are other subtle issues you can face if the *hashCode* and *equals* contracts are not properly adhered to. Refer to the **Object** class API for the contract details.

When sorting objects in a collection with *compareTo*, it is strongly recommended, but not strictly required that **(x.compareTo(y)==0) == (x.equals(y))**. Generally speaking, any class that implements the *Comparable* interface and violates this condition should clearly indicate this fact.

Mistake #4: 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(); ) {
3     T obj = iter.next();
4     if (obj.someCondition()) {
5         list.remove(0); // ConcurrentModificationException
6     }
7 }
8
```

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 }
8
```

[04: Understanding](#)
[05: ♥ 7 Java FP](#)
[05: ♦ Finding the](#)
[06: ♥ Java 8 way](#)
[07: ♦ Java 8 API](#)
[08: ♦ Write code](#)
[10: ♦ ExecutorSe](#)
[Fibonacci numbe](#)
[Java 8 String str](#)
[Java 8 using the](#)
[Java 8: 7 useful](#)
[Java 8: Different](#)
[Java 8: Does "O](#)
[Java 8: What is c](#)
[Learning to write](#)
[Non-trivial Java 8](#)
[Top 6 Java 8 fea](#)
[Top 8 Java 8 fea](#)
[Understanding J](#)
[JVM \(6\)](#)
[♦ Java Garbage](#)
[01: jvisualvm to](#)
[02: jvisualvm to](#)
[05: Java primitiv](#)
[06: ♦ 10+ Atomic](#)
[5 JMX and MBes](#)
[Reactive Program](#)
[07: Reactive Pro](#)
[10: ♦ ExecutorSe](#)
[3. Multi-Threadir](#)
[Swing & AWT \(2\)](#)
[5 Swing & AWT i](#)
[Q6 – Q11 Swing](#)
[JEE Interview Q&A \(3](#)
[JEE Overview \(2\)](#)
[♦ 8 Java EE \(aka](#)
[Java EE interview](#)
[Web basics \(8\)](#)
[01: ♦ 12 Web ba](#)
[02: HTTP basics](#)
[03: Servlet inter](#)

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.

Q. Difference between **fail-fast** and **fail-safe** iterators

A. 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.

Mistake #5: Common exception handling mistakes.

a) Sweeping exceptions under the carpet by doing nothing.

```

1 try{
2
3 }catch(SQLException sqe){
4     // do nothing
5 }
6

```

In few rare scenarios, it is desired to do nothing with an exception, e.g. in finally block, we try to close database connection, and some exception occurs. In this case, exception can be ignored.

```

1 try{
2
3 }catch(SQLException sqe){
4     ...
5     ...
6 }finally{
7     try{
8         conn.close();
9     }catch(Exception e){
10         //leave it.
11     }

```

04: JSP overview

05: Web patterns:

06: ♦ MVC0, MV

07: When to use

08: Web.xml inte

WebService (11)

01: ♥♦ 40+ Java

02: ♦ 6 Java RE

03: ♥ JAX-RS hc

04: 5 JAXB inter

05: RESTful We

06: RESTful Wel

07: HATEOAS R

08: REST constr

09: 11 SOAP We

10: SOAP Web

11: ♥ JAX-WS hc

JPA (2)

10: Spring, Java

8 JPA interview

JTA (1)

JTA interview Q&

JDBC (4)

♦ 12 FAQ JDBC

JDBC Overview

NamedParamete

Spring, JavaCon

JMS (5)

♦ 16 FAQ JMS ir

Configuring JMS

JMS versus AM

Spring JMS with

Spring JMS with

JMX (3)

5 JMX and MBe

Event Driven Pr

Yammer metrics

JNDI and LDAP (1)

JNDI and LDAP

Pressed for time? Jav

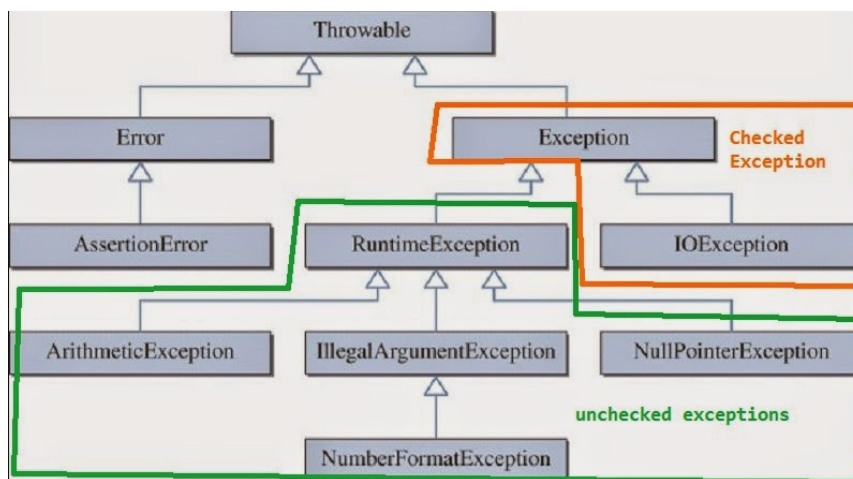
Job Interview Ice B


```
12 }
13 }
```

But in general, this is a very bad practice and can hide issues.

b) Inconsistent use of checked and unchecked (i.e. Runtime) exceptions.

Document a consistent exception handling strategy. In general favor unchecked (i.e. Runtime) exceptions, which you don't have to handle with catch and throw clauses. Use checked exceptions in a rare scenarios where you can recover from the exception like deadlock or service retries. In this scenario, you will catch a checked exception like *java.io.IOException* and wait a few seconds as configured with the retry interval, and retry the service configured by the retry counts.



c) Exceptions are polymorphic in nature and more specific exceptions need to be caught before the generic exceptions.

So, it is wrong to catch *Exception* before *IOException*.

```

1 try{
2     //.....
3 } catch(Exception ex){
4     log.error("Error:" + ex)
5 } catch(IOException ex){
6     log.error("Connectivity issue:" + ex); //never
7 }
```

01: ♦ 15 Ice brea
 02: ♥♦ 8 real life
 03: ♦10+ Know y
 FAQ Core Java Jot
 ♥♦ Q1-Q10: Top
 ♦ Q11-Q23: Top
 ♦ Q24-Q36: Top
 ♦ Q37-Q42: Top
 ♦ Q43-Q54: Top
 01: ♥♦ 15 Beginr
 02: ♥♦ 10+ Java
 FAQ JEE Job Inter
 ♦ 12 FAQ JDBC
 ♦ 16 FAQ JMS ir
 ♦ 8 Java EE (aka
 ♦ Q01-Q28: Top
 ♦ Q29-Q53: Top
 01: ♦ 12 Web ba
 06: ♦ MVC0, MV
 JavaScript mista
 JavaScript Vs Ja
 JNDI and LDAP
 JSF interview Q
 JSON interview
 FAQ Java Web Ser
 01: ♥♦ 40+ Java
 02: ♦ 6 Java RE
 05: RESTFul We
 06: RESTful Wel
 09: 11 SOAP We
 Java Application Ar
 001A: ♦ 7+ Java
 001B: ♦ Java arc
 04: ♦ How to go
 Hibernate Job Inter
 01: ♥♦ 15+ Hiber
 01b: ♦ 15+ Hiber
 06: Hibernate Fil
 8 JPA interview c
 Spring Job Intervie
 ♦ 11 Spring boot

8

Fix this by catching the more specific *IOException* first

```

1 try{
2     //....
3 } catch(IOException ex){
4     log.error("Connectivity issue:" + ex);
5 } catch(Exception ex){
6     log.error("Error:" + ex)
7 }
8

```

In Java 7 onwards, you can catch multiple exceptions like

```

1 try{
2     //....
3 }
4 catch (ParseException | IOException exception) {
5     // handle I/O problems.
6 } catch (Exception ex) {
7     //handle all other exceptions
8 }
9

```

d) Wiping out the stack trace

```

1 try{
2     //....
3 } catch(IOException ioe){
4     throw new MyException("Problem in data reading")
5 }
6

```

e) Unnecessary Exception Transformation.

In a layered application, many times each layer catches exception and throws new type of exception. Sometimes it is absolutely unnecessary to transform an exception. An unchecked exception can be automatically bubbled all the way upto the GUI Layer, and then handled at the GUI layer with a **log.error(ex)** for the log file and a generic info like “An expected error has occurred, and please contact support on xxxxx” to the user. Internal details like stack trace with host

01: ♥♦ 13 Spring

01b: ♦ 13 Spring

04 ♦ 17 Spring b

05: ♦ 9 Spring B

Java Key Area Ess

♦ Design pattern

♥ Top 10 causes

♥♦ 01: 30+ Writir

♦ 12 Java desigr

♦ 18 Agile Develo

♦ 5 Ways to debi

♦ 9 Java Transac

♦ Monitoring/Pro

02: ♥♦ 13 Tips to

15 Security key :

4 FAQ Performa

4 JEE Design Pa

5 Java Concurr

6 Scaling your J

8 Java memory i

OOP & FP Essenti

♦ 30+ FAQ Java

01: ♦ 19 Java 8 I

04: ♥♦ Top 6 tips

Code Quality Job I

♦ Ensuring code

♦ 5 Java unit tes

SQL, XML, UML, JSC

ERD (1)

♦ 10 ERD (Entity

NoSQL (2)

♦ 9 Java Transac

3. Understanding

Regex (2)

♥♦ Regular Expr

Regular Express

SQL (7)

♦ 15 Database d

♦ 14+ SQL interv

♦ 9 SQL scenari

Auditing databas

names, database table names, etc should not be shown to the user as it can be exploited to cause security threats.

Data Access Layer → Business Service Layer → GUI Layer

Logging

Mistake #6: Using `System.out.println(...)` statements for debugging without making use of the debugging capabilities provided in IDE tools. If you need to log, use log4j library instead.

```
1 if(log.isDebugEnabled()) {
2     log.debug(".....");
3 }
4 //....
5 log.info(".....");
6 //....
7 log.error(ex);
8
```

The log4j library will not only perform better than `System.out.println` statements, but also provides lots of additional features like writing to a file, console, queue, etc, archiving and rolling the log files, controlling the log levels with debug, warn, info, error, etc and many more.

Mistake #7: Reinventing the wheel by writing your own logic when there are already well written and proven APIs and libraries are available. When coding, always have Core Java APIs, Apache APIs, Spring framework APIs, Google Guava library APIs, and relevant reference documentations handy to reuse them instead of writing your own half baked solutions.

Instead of:

```
1 if(str != null && str.trim().length() > 0) {
2     //.....
3 }
4
```

You can use the **StringUtils** library from the Apache

Deleting records	
SQL Subquery in	
Transaction man	
UML (1)	
12 UML intervi	
JSON (2)	
JSON interview	
JSON, Jackson,	
XML (2)	
XML basics inter	
XML Processing	
XSD (2)	
11 FAQ XSD inte	
XSD reuse inter	
YAML (2)	
YAML with Java	
YAML with Sprin	
Hadoop & BigData Int	
01: Q1 – Q6 Had	
02: Q7 – Q15 Hada	
03: Q16 – Q25 Hada	
04: Q27 – Q36 Apa	
05: Q37 – Q50 Apa	
05: Q37-Q41 – Dat	
06: Q51 – Q61 HB	
07: Q62 – Q70 HDI	
Java Architecture Inte	
01: 30+ Writing	
001A: 7+ Java int	
001B: Java archi	
01: 40+ Java W	
02: 13 Tips to w	
03: What should l	
04: How to go ab	
05: ETL architectur	
1. Asynchronous pi	
2. Asynchronous pi	
Scala Interview Q&As	
01: Q1 – Q6 Scal	
02: Q6 – Q12 Scal	
03: Q13 – Q18 Sca	

commons API to simplify your code

```
1 if(!StringUtils.isEmpty(str){
2     //.....
3 }
4
```

EqualsBuilder, *HashCodeBuilder*, *StringBuilder*, etc are Apache commons utility methods.

Mistake #8: Resource leak issues are reported when resources are allocated but not properly disposed (i.e. closed) after use.

The system resources like file handles, sockets, database connections, etc are limited resources that need to be closed once done with them otherwise your applications run the risk of leaking resources and then running out of sockets, file handles, or database connections.

Bad code: if an exception is thrown before `in.close()` is reached, the Scanner that is holding on to the `System.in` resource will never get closed.

```
1 public void readShapeData() throws IOException {
2     Scanner in = new Scanner(System.in);
3     log.info("Enter salary: ");
4     salary = in.nextDouble();
5     in.close();
6 }
7
```

Good code: The finally block is reached even if an exception is thrown. So, the scanner will be closed.

```
1 public void readShapeData() throws IOException {
2     Scanner in = new Scanner(System.in);
3     try {
4         log.info("Enter salary: ");
5         salary = in.nextDouble();
6     } finally {
7         in.close(); //reached even when an exce
8     }
9 }
```

04: Q19 – Q26 Sca

05: Q27 – Q32 Sca

06: Q33 – Q40 Sca

07: Q41 – Q48 Sca

08: Q49 – Q58 Sca

09: Q59 – Q65 Hig

10: Q66 – Q70 Pat

11: Q71 – Q77 – S

12: Q78 – Q80 Rec

Spring, Hibernate, & I

Spring (18)

Spring boot (4)

◆ 11 Spring bc

01: Simple Sp

02: Simple Sp

03: Spring bo

Spring IO (1)

Spring IO tuto

Spring JavaConl

10: Spring, Ja

Spring, JavaC

Spring, JavaC

Spring, JavaC

01: ♥♦ 13 Spring

01b: ♦ 13 Spring

02: ► Spring DI

03: ♥♦ Spring DI

04 ♦ 17 Spring b

05: ♦ 9 Spring B

06: ♥ Debugging

07: Debugging S

Spring loading p

Hibernate (13)

01: ♥♦ 15+ Hiber

01b: ♦ 15+ Hiber

02: Understandir

03: Identifying ar

04: Identifying ar

05: Debugging H

06: Hibernate Fil

07: Hibernate mi

```

9  }
10

```

Mistake #9: Comparing two objects (== instead of .equals)

When you use the == operator, you are actually comparing two object references, to see if they point to the same object. You cannot compare, for example, two strings or objects for equality, using the == operator. You must instead use the .equals method, which is a method inherited by all classes from java.lang.Object.

Mistake #10: Confusion over passing by value, and passing by reference as Java has both primitives like int, float, etc and objects.

When you pass a primitive data type, such as a char, int, float, or double, to a function then you are passing by value, which means a copy of the data type is duplicated, and passed to the function. If the function chooses to modify that value, it will be modifying the copy only.

When you pass a Java object, such as an array or an Employee object, to a function then you are passing by reference, which means the reference is copied, but both the original and copied references point to the same object. Any changes you make to the object's member variables will be permanent – which can be either good or bad, depending on whether this was what you intended.

5 bonus mistakes:

1. Forgetting that Java is zero-indexed.
2. Not writing thread-safe code with proper synchronization or thread-local objects.
3. Not favoring immutable objects. Immutable objects are inherently thread-safe.
4. Not properly handling null references, and causing the ubiquitous *NullPointerException*. This is a run time exception and the compiler can't warn you.
5. Capitalization errors where the class names and Java file names should start with capital letters and method

08: Hibernate au
09: Hibernate en
10: Spring, Java
11: Hibernate de
12: Hibernate cu

AngularJS (2)

♥ 8 AngularJS in
More Angular JS

Git & SVN (6)

♥ Git & Maven fc
♥ Merging Vs rel
♥ Understanding
6 more Git interv
8 Git Source cor
Setting up Cygw

JMeter (2)

♥ JMeter for test
♦ JMeter perform

JSF (2)

JSF interview Q&
More JSF intervi

Maven (3)

♥ Git & Maven fc
12 Maven intervi
7 More Maven ir

Testing & Profiling/Sa

Automation Testing

♥ Selenium and

Code Coverage (2)

Jacoco for unit te
Maven and Cobr

Code Quality (2)

♥ 30+ Java Code
♦ Ensuring code

jvisualvm profiling (

01: jvisualvm to :
02: jvisualvm to :
03: jvisualvm to :

Performance Testir

♥ JMeter for test
♦ JMeter perform

and variable names should start with lowercase letters.

Popular Posts

♦ 11 Spring boot interview questions & answers

852 views

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

825 views

18 Java scenarios based interview Questions and Answers

447 views

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

400 views

♦ 7 Java debugging interview questions & answers

311 views

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

301 views

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

292 views

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

286 views

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

263 views

8 Git Source control system interview questions & answers

215 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

Unit Testing Q&A (2

BDD Testing (4)

Java BDD (Be

jBehave and E

jBehave and j

jBehave with t

Data Access Uni

♥ Unit Testing

Part #3: JPA H

Unit Test Hibe

Unit Test Hibe

JUnit Mockito Sp

JUnit Mockito

Spring Con

Unit Testing

Part 1: Unit te

Part 2: Mockit

Part 3: Mockit

Part 4: Mockit

Part 5: Mockit

Testing Spring T.

Integration Un

Unit testing Sp

♦ 5 Java unit tes

JUnit with Hamc

Spring Boot in u

Other Interview Q&A 1

Finance Domain In

12+ FX or Forex

15 Banking & fin

FIX Interview Q&A

20+ FIX basics in

Finding your way

Groovy Interview C

Groovy Coding C

Cash balance

Sum grades C

♥ Q1 – Q5 Groov

♦ 20 Groovy clos

♦ 9 Groovy meta

Groovy method c



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.



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.

◀ Top 5 OOPs tips for Java developers

▶ Java Beginner Interview Questions and Answers on the Object class

Video ▶

Posted in Java Overview, member-paid

Tags: Core Java FAQs, Novice FAQs, TopX

2 comments on "04: ♦ Top 10 most common Core Java beginner mistakes"



Nikky says:

July 2, 2015 at 2:21 am

What is the #2 way of calculation in Mistake #1

[Reply](#)



akumaras says:

July 2, 2015 at 10:48 am

- Q6 – Q10 Groov
- JavaScript Interview
- JavaScript Top I
- ♥ Q1 – Q10 J
- ♦ Q11 – Q20
- ♦ Q21 – Q30
- ♦ Q31 – Q37
- JavaScript mis
- JavaScript Vs Ja
- JavaScript Vs
- Unix Interview Q&A
- ♥ 14 Unix intervi
- ♥ Hidden Unix, C
- sed and awk to v
- Shell script inter
- Unix history com
- Unix remoting in
- Unix Sed comm
- Free Java Interview
- ▶ Java Integration
- ▶ Java Beginner I
- 02: ▶ Spring DIP, I
- 06: ▶ Tell me abou

As a Java Architect

[Java architecture & design concepts interview Q&As with diagrams](#) | [What should be a typical Java EE architecture?](#)

Senior Java developers must have a good handle on

Not sure if I understand your question. #2 way is to use the BigDecimal class for monetary values.

```
//2: $0.63
```

```
System.out.println(new BigDecimal("1.05")  
.subtract(new BigDecimal("0.42")));
```

[Reply](#)

Leave a Reply

Logged in as geethika. [Log out?](#)

Comment

[Post Comment](#)

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

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 \(45\)](#)

Preparing for Java written & coding tests

open all | close all

- ✚ ♦ Complete the given
- ✚ Can you write code? |
- ✚ 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...to go places?

open all | close all

- ✚ Career Making Know-
- ✚ Job Hunting & Resum

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.