Core Java Modifiers, Generics, and Annotations: differences between X and Y interview Q&A

Posted on May 5, 2015 by Arulkumaran Kumaraswamipillai — No Comments ↓

This post is for quick brush-up. These Q&A are discussed in detailed elsewhere in posts like Java modifiers interview Q&A/a> | Java generics interview Q&A | Java annotations interview Q&A

Q1. What are the differences among final, finally, and finalize?
A1. final makes a variable reference not changeable, makes a method not overridable, and makes a class not inheritable.

finally is used in a try/catch statement to almost always execute the code. Even when an exception is thrown, the finally block is executed. This is used to close non-memory resources like file handles, sockets, database connections, etc till Java 7. This is no longer true in Java 7 with the introduction of **AutoCloseable** interface.

finalize is called when an object is garbage collected. This method should not be used to release non-memory resources like file handles, sockets, database connections, etc because Java has only a finite number of these resources and you do not know when the garbage collection is going to kick in to release these non-memory resources through the finalize( ) method.

Q2. What is the difference between 'final' and 'const' modifiers on a variable?
Q2. This is a bit of a tricky question because the 'const' is a reserved keyword in Java, but not used.

A **final** modifier on a reference variable just means that the reference cannot be changed to reference a different object once assigned. This does not mean that the variable is a constant because the values of the object it refers to can be modified unless the values themselves are marked as final.

A **const** is a reserved keyword in Java to make a variable constant, so that the referenced object values also cannot be modified, but it is currently not used in Java. The compiler will complain if you use it.

Q3. What is the difference between 'volatile' and 'synchronized' modifiers?
A3. The **volatile** modifier is applied to variables of both primitives and objects, whereas the synchronized modifier is applied to only objects.

16 Technical Key Areas

The **volatile** modifier only guarantees visibility and ordering, but not atomicity, whereas the **synchronized** modifier guarantees both visibility and atomicity if done properly.

So, the volatile variable has a limited use, and cannot be used in compound operations like incrementing a variable.

Q4. What is the difference between 'transient' modifier and '@Transient' annotation?
A4. A **transient** modifier is used with serialization. It tells a member variable not to be serialized when it is persisted to streams of bytes. It cannot be used with a static variable as a static variable belongs to a class, not to an object. You can only serialize an object.

@Transient annotation suggests that an annoted variable in an object should not be persisted to a database table column in JPA, Hibernate, etc.

Q5. What is the difference between 'marker interfaces' like Serializable and 'annotations' like @Transient?
A5. Marker interfaces were introduced since the inception of Java, whilst the annotations were only introduced with Java 5. Everything that can be done with marker interfaces can be done instead with annotations. In fact, annotations can do a lot more. So, it's now recommended to favor annotations over marker interfaces. Annotations can have parameters of various kinds, hence they're much more flexible.

Q6. What are the differences among List<Number>,List<? extends Number>, and List<? super Number>
A6. Many developers struggle with the wild cards. Here is the guide:

1. Use the ? extends wildcard if you need to retrieve object from a data structure. That is read only. You can't add elements to the collection.
2. Use the ? super wildcard if you need to put objects in a data structure.
3. If you need to do both read and add elements, don't use any wildcard.

⊞ Transaction Managen

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 Tu
⊞ 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 .....?

Q7. What are the differences among List, List<?>, and List<Object>
A7.

1. Just “List” is a **heterogeneous** mixture or a mixed bag that contains elements of all types like Integer, String, Fruit, Vegetable, etc.
2. The “List<?>” is a **homogenous** collection that represents a family of generic instantiations of List like List<String>, List<Integer>, List<Fruit>, etc.
3. The “List<Object>” is also a **heterogeneous** mixture like the “List” (i.e. without any generics), but not the same and can be more restrictive than a “List”.

So, unlike an “Object” class is the super class for all the other Java objects, List<Object> is not the super class for all the different types of list objects. A List<?> is the superclass for all the different types of list objects.

Q8. What are the differences between compile-time and runtime annotations?
A8. “compile-time” annotation is read when the program is compiled with the “javac” command, and the “runtime” annotation is read when the program is executed with the “java” command.

@Override is a simple compile-time annotation to catch little mistakes like typing tostring( ) instead of toString( ) in a subclass.

@Test is an annotation that JUnit framework uses at runtime with the help of reflection to determine which method(s) to execute within a test class.

Q9. What are differences among the terms annotation types, annotations, and annotation processors?
A9. An **annotation** type is used for defining an annotation.

```
1 @Documented
2 @Inherited
3 @Retention(RetentionPolicy.RUNTIME)
4 @Target({ElementType.METHOD,ElementType.TYPE})
```

```
5 public @interface ToDo {
6       String comments();
7 }
```

An **annotation** is the meta tag that you use in your applications.

```
1 @ToDo(comments="Not yet complete")
2 public class MyClass {
3
4     @Deprecated
5     public void doSomething() {
6         // some logic
7     }
8 }
9
```

Finally, annotating your code alone is not going to give you any functionality apart from some form of documentation unless you have **annotation processors** that process the annotations in special way to add behavior. The processors can be Java compiler itself, tools that are shipped with Java itself like Javadoc, apt (Annotation Processing Tool), Java Runtime, Java IDEs like eclipse, Net Beans, etc and frameworks like Hibernate 3.0 and Spring 3.0, JEE CDI, etc.

Q10. What are differences among the access modifiers private, protected, and public?
A10. Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. Without access modifiers you can't implemt encapsulation. The following table shows the access to members permitted by each modifier.

Access Modifiers	*Same Class*	*Same Package*	*Subclass*	*Other packages*
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no access modifier	Y	Y	N	N
private	Y	N	N	N

Java access modifiers

Q11. What are differences between 'autoboxing' and 'casting'?

A11. Auto-boxing is when you **wrap** a primitive type to an object reference type like Integer, etc and **auto un-boxing** is the reverse.

Casting is when you want one type to be treated as another type among primitive types and reference types. There are two types of castings — **"implicit and explicit"** with regards to object references and **"narrowing and widening"** with regards to primitive types. If it were not for implcit casting, polymorphism would not be possible. Explicit casting requires casting construct as shown below.

```
1 int myNumber = (int) 240l; // casting long to int
2 long myOtherNumber = 240;  // casting int to long
3
```

Popular Posts

♦ 11 Spring boot interview questions & answers

824 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**

‹ 04: ♦ How to go about designing a medium size JEE application?

Single Pointer: Partitioning around a pivotal number ›

Posted in Differences Between X & Y, 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

 ↑