More Java annotations interview questions & answers

Posted on September 7, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. What do you understand by the terms annotation types, annotations, and annotation processors?

A1. An **annotation type** is used for defining an annotation. Java 5 defines a number of annotation types like @Override, @SuppressWarnings, @Deprecated, etc and meta annotation types that are used by annotation (i.e. a meta meta data) type like @Target, @RetntionPolicy, @Inherited, and @Documented. For example,

```
1 @Documented
2 @Inherited
3 @Retention(RetentionPolicy.RUNTIME)
```

```
@Target({ElementType.METHOD,ElementType.TYPE})
public @interface ToDo {
      String comments();
}
```

As you can see the difference between an interface definition and that of an annotation is the presence of **@** before the interface keyword. Here are some of the rules for defining an annotation:

– Method declarations inside an annotation should not have any parameters.
– Method declarations should not have any throws clauses.
– Return types of the methods should be primitives, String, Class, enum, or array of the above types.

An annotation is the meta tag that you use in your applications. For example, you can use the annotation types @Override, @Suppresswarnings, @Inherited, etc included in Java and the custom annotation type @ToDo that was defined above as shown below:

```
package annotation.example;

import java.util.ArrayList;
import java.util.List;

@ToDo(comments="Not yet complete")
public class MyClass {

      @Deprecated
      public void doSomething() {
          // some logic
      }

      @SuppressWarnings(value = "unchecked")
      @ToDo(comments="Need to confirm with legacy
      public void doSomethingBetter() {
          List vocabulary = new ArrayList();
          vocabulary.add("deliberate");
      }

      @Override
      public String toString() {
          return super.toString();
      }

      @Override
      public int hashCode() {
          return super.hashCode();
      }
}
```


- Exceptions (2)
- Java 7 (2)
- Java 8 (24)
- 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, & N
Testing & Profiling/Sa
Other Interview Q&A f
Free Java Interviev

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 Ir
- Low Latency (7)
- Memory Managemen
- Performance (13)
- QoS (8)
- Scalability (4)
- SDLC (6)
- Security (13)
- Transaction Managen

Finally, annotating your code alone is not going to give you any functionality apart from some form of documentation unless you have **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.

1) **@Override** and **@SuppressWarnings** are used by the Java compiler.
2) The annotation **@Deprecated** is used by the Java compiler and the IDEs like eclipse.
3) The custom annotation **@ToDo** can be used at runtime to produce a summary report as a to do list by querying the annotations at runtime using the Java reflection API.

```
1  package annotation.example;
2
3  import java.lang.annotation.Annotation;
4
5  public class QueryAnnotation {
6
7      public static void main(String[] args) {
8          Annotation[] typeAnnotations = MyClass.c
9          for (Annotation annotation : typeAnnotat
10             if (annotation.annotationType().getS
11                     ToDo.class.getSimpleName()))
12                 System.out.println(" --> Comment
13                         + ((ToDo) annotation).co
14             }
15         }
16     }
17 }
```

Q2. Can you describe the ways in which annotations can be used at pre-compile time, compile time, post-compile time and runtime?

A2. **Pre compile-time:** You can generate additional boiler plate source code and descriptor files using tools like apt (Annotation Processing Tool) during the build process. For example, a service framework can be developed using annotations where a developer provides a delegate class say UserDelegate with the required business logic and relevant annotations. The service framework will make use of the apt tool to read this delegate class and produce additional

artifacts required to expose the business functions via RMI (using EJBs) and Web Services. The apt tool can be used to generate the required source files like local interfaces, remote interfaces, wrapper implementation to expose the service as RMI and Web Service during build time (i.e. prior to compiling). The annotation processing tool is very powerful.

Compile-time: By the Java compiler and IDEs to raise errors and warnings during compiling source code into byte code (i.e. .class files) as discussed earlier.

Post Compile-time: Annotations can be scanned on byte code files (i.e. .class files) using byte code processing libraries like Javaassist or ASM. Javaassist does have reflection like API that allows you iterate over methods and fields of a class file. You can read your class files from the InputStreams from your classpath or .jar. Based on annotations, additional code can be injected or existing code can be modified. Any form of byte code manipulation can make your code harder to read or understand. Don't favor this approach unless you have a compelling reason to do so. For example, performance considerations associated with scanning for all the annotations by loading each and every class using your Class loader and Java reflection.

Runtime: By the application itself and other frameworks to check the validity of the input passed by the clients and extract program behaviors at runtime using Java reflection. The Java reflection API has been updated with facility to work with annotations since JDK 5.0.

Q3. What do you understand by annotation of annotation? How do you control when an annotation is need and where an annotation should go?
A3. JDK 5.0 provides four annotations in the java.lang.annotation package that are used only when writing annotations.

@Documented → Should the annotation be in Javadoc? Annotations on a class or method don't appear in the Javadocs by default. The @Documented is a marker

annotation (i.e. accepts no parameters) that changes this behavior.

@Retention → When the annotation is needed? There are three options as listed in the RetentionPolicy enumeration.

Policy	Description
RetentionPolicy. SOURCE	Annotations are discarded during compile-time. Annotations are not written to the byte code as they would not make any sense. For example, @Override and @SuppressWarnings.
RetentionPolicy. CLASS	Annotations are written to byte code, but discarded when they are loaded into the JVM. This is useful if you want to do any byte code level manipulation.
RetentionPolicy. RUNTIME	Do not discard. The annotation should be available for reflection at runtime. For example, @Deprecated and @Documented. You might be wondering why @Documented retention policy is runtime. This is because Javadoc loads its information from class files, using a JVM.

@Target → Where the annotation can go? You have eight options listed in the ElementType enumeration to tell where a particular annotation can be applied.

ElementType.TYPE (class, interface, enum)
ElementType.FIELD (instance variable)
ElementType.METHOD
ElementType.PARAMETER
ElementType.CONSTRUCTOR
ElementType.LOCAL_VARIABLE
ElementType.ANNOTATION_TYPE (on another annotation)
ElementType.PACKAGE

@Inherited → Should subclasses get the annotation? This controls if an annotation should affect subclasses. If you look at the earlier examples MyClass base class and @ToDo annotation, the @ToDo annotation is annotated with @Inherited.

Q4. What are the different annotation types?
A4.

1) Marker annotation.
2) Single element annotation.
3) Full value or multi-value annotation.

Marker type annotations have no elements, except the annotation name itself.

```
1 @Target(ElementType.METHOD)
2 @Retention(RetentionPolicy.SOURCE)
3 public @interface Override {
4 }
```

Single Element or single value type annotations provide a single piece of data only.

```
1 @Documented
2 @Inherited
3 @Retention(RetentionPolicy.SOURCE)
4 @Target({ElementType.METHOD,ElementType.TYPE})
5 public @interface ThreadSafe {
6      //default makes it optional
7     String value() default "";
8 }
```

Usage:

```
1 @ThreadSafe("not using any instance variables")
2 public void method1(){
3    //...
4 }
```

Full-value or multi-value type annotations have multiple data members.

```
1  package javax.ejb;
2
3  import java.lang.annotation.Retention ;
4  import java.lang.annotation.Target ;
5
6  @Target (ElementType.TYPE)
7  @Retention (RetentionPolicy.RUNTIME)
8  public @interface Stateless {
9     String  name() default "";
10    String  mappedName() default "";
11    String  description() default "";
12 }
```

Usage:

```
1 @Stateless(name="Charging", description="Charging
2 @TransactionManagement
3                   (TransactionManagementType.CONTAI
4 @TransactionAttribute(TransactionAttributeType.NE
5 public class ChargingDAOImpl extends MediationDAO
6                                     ChargingDAO {
7         // fields and methods
8 }
```

Popular Posts

♦ 11 Spring boot interview questions & answers

823 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**

‹ 02: ♥♦ 8 real life examples of SAR technique for Java developers

12 Maven interview Questions & Answers ›

Posted in Annotations, 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

 ↑