(/)

Java Annotations Interview Questions (+ Answers)

Last modified: November 11, 2018

by baeldung (https://www.baeldung.com/author/baeldung/)

Java (https://www.baeldung.com/category/java/) **+**

Interview (https://www.baeldung.com/tag/interview/)

This article is part of a series:

1. Introduction

Annotations have been around since Java 5, and nowadays, they are ubiquitous programming constructs that allow enriching the code.

In this article, we'll review some of the questions, regarding annotations; that are often asked on technical interviews and, where appropriate; we'll implement examples to understand their answers better.

2. Questions

Q1. What are annotations? What are their typical use cases?

Annotations are metadata bound to elements of the source code of a program and have no effect on the operation of the code they operate.

Their typical uses cases are:

- **Information for the compiler** – with annotations, the compiler can detect errors or suppress warnings
- **Compile-time and deployment-time processing** – software tools can process annotations and generate code, configuration files, etc.
- **Runtime processing** – annotations can be examined at runtime to customize the behavior of a program

Q2. Describe some useful annotations from the standard library.

There are several annotations in the *java.lang* and *java.lang.annotation* packages, the more common ones include but not limited to:

- *@Override* – marks that a method is meant to override an element declared in a superclass. If it fails to override the method correctly, the compiler will issue an error
- *@Deprecated* – indicates that element is deprecated and should not be used. The compiler will issue a warning if the program uses a method, class, or field marked with this annotation
- *@SuppressWarnings* – tells the compiler to suppress specific warnings. Most commonly used when interfacing with legacy code written before generics appeared
- *@FunctionalInterface* – introduced in Java 8, indicates that the type declaration is a functional interface and whose implementation can be provided using a Lambda Expression

Q3. How can you create an annotation?

Annotations are a form of an interface where the keyword *interface* is preceded by *@,* and whose body contains *annotation type element* declarations that look very similar to methods:

```
public @interface SimpleAnnotation {
    String value();

    int[] types();
}
```

After the annotation is defined, yon can start using it in through your code:

```
@SimpleAnnotation(value = "an element", types = 1)
public class Element {
    @SimpleAnnotation(value = "an attribute", types = { 1, 2 })
    public Element nextElement;
}
```

Note that, when providing multiple values for array elements, you must enclose them in brackets.

Optionally, default values can be provided as long as they are constant expressions to the compiler:

```java
public @interface SimpleAnnotation {
    String value() default "This is an element";

    int[] types() default { 1, 2, 3 };
}
```

Now, you can use the annotation without those elements:

```java
@SimpleAnnotation
public class Element {
    // ...
}
```

Or only some of them:

```java
@SimpleAnnotation(value = "an attribute")
public Element nextElement;
```

Q4. What object types can be returned from an annotation method declaration?

The return type must be a primitive, *String*, *Class*, *Enum*, or an array of one of the previous types. Otherwise, the compiler will throw an error.

Here's an example code that successfully follows this principle:

```
enum Complexity {
    LOW, HIGH
}

public @interface ComplexAnnotation {
    Class<? extends Object> value();

    int[] types();

    Complexity complexity();
}
```

The next example will fail to compile since *Object* is not a valid return type:

```
public @interface FailingAnnotation {
    Object complexity();
}
```

Q5. Which program elements can be annotated?

Annotations can be applied in several places throughout the source code. They can be applied to declarations of classes, constructors, and fields:

```
@SimpleAnnotation
public class Apply {
    @SimpleAnnotation
    private String aField;

    @SimpleAnnotation
    public Apply() {
        // ...
    }
}
```

Methods and their parameters:

```java
@SimpleAnnotation
public void aMethod(@SimpleAnnotation String param) {
    // ...
}
```

Local variables, including a loop and resource variables:

```java
@SimpleAnnotation
int i = 10;

for (@SimpleAnnotation int j = 0; j < i; j++) {
    // ...
}

try (@SimpleAnnotation FileWriter writer = getWriter()) {
    // ...
} catch (Exception ex) {
    // ...
}
```

Other annotation types:

```java
@SimpleAnnotation
public @interface ComplexAnnotation {
    // ...
}
```

And even packages, through the *package-info.java* file:

```
@PackageAnnotation
package com.baeldung.interview.annotations;
```

As of Java 8, they can also be applied to the *use* of types. For this to work, the annotation must specify an *@Target* annotation with a value of *ElementType.USE*:

```
@Target(ElementType.TYPE_USE)
public @interface SimpleAnnotation {
    // ...
}
```

Now, the annotation can be applied to class instance creation:

```
new @SimpleAnnotation Apply();
```

Type casts:

```
aString = (@SimpleAnnotation String) something;
```

Implements clause:

```
public class SimpleList<T>
  implements @SimpleAnnotation List<@SimpleAnnotation T> {
    // ...
}
```

And *throws* clause:

```
void aMethod() throws @SimpleAnnotation Exception {
    // ...
}
```

Q6. Is there a way to limit the elements in which an annotation can be applied?

Yes, the *@Target* annotation can be used for this purpose. If we try to use an annotation in a context where it is not applicable, the compiler will issue an error.

Here's an example to limit the usage of the *@SimpleAnnotation* annotation to field declarations only:

```
@Target(ElementType.FIELD)
public @interface SimpleAnnotation {
    // ...
}
```

We can pass multiple constants if we want to make it applicable in more contexts:

```
@Target({ ElementType.FIELD, ElementType.METHOD, ElementType.PACKAGE })
```

We can even make an annotation so it cannot be used to annotate anything. This may come in handy when the declared types are intended solely for use as a member type in complex annotations:

```
@Target({})
public @interface NoTargetAnnotation {
    // ...
}
```

Q7. What are meta-annotations?

Are annotations that apply to other annotations.

All annotations that aren't marked with *@Target,* or are marked with it but include *ANNOTATION_TYPE* constant are also meta-annotations:

```java
@Target(ElementType.ANNOTATION_TYPE)
public @interface SimpleAnnotation {
    // ...
}
```

Q8. What are repeating annotations?

These are annotations that can be applied more than once to the same element declaration.

For compatibility reasons, since this feature was introduced in Java 8, repeating annotations are stored in a *container annotation* that is automatically generated by the Java compiler. For the compiler to do this, there are two steps to declared them.

First, we need to declare a repeatable annotation:

```java
@Repeatable(Schedules.class)
public @interface Schedule {
    String time() default "morning";
}
```

Then, we define the containing annotation with a mandatory *value* element, and whose type must be an array of the repeatable annotation type:

```java
public @interface Schedules {
    Schedule[] value();
}
```

Now, we can use @Schedule multiple times:

```
@Schedule
@Schedule(time = "afternoon")
@Schedule(time = "night")
void scheduledMethod() {
    // ...
}
```

Q9. How can you retrieve annotations? How does this relate to its retention policy?

You can use the Reflection API or an annotation processor to retrieve annotations.

The *@Retention* annotation and its *RetentionPolicy* parameter affect how you can retrieve them. There are three constants in *RetentionPolicy* enum:

- *RetentionPolicy.SOURCE* – makes the annotation to be discarded by the compiler but annotation processors can read them
- *RetentionPolicy.CLASS* – indicates that the annotation is added to the class file but not accessible through reflection
- *RetentionPolicy.RUNTIME* –Annotations are recorded in the class file by the compiler and retained by the JVM at runtime so that they can be read reflectively

Here's an example code to create an annotation that can be read at runtime:

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Description {
    String value();
}
```

Now, annotations can be retrieved through reflection:

```
Description description
  = AnnotatedClass.class.getAnnotation(Description.class);
System.out.println(description.value());
```

An annotation processor can work with *RetentionPolicy.SOURCE*, this is described in the article Java Annotation Processing and Creating a Builder (/java-annotation-processing-builder).

RetentionPolicy.CLASS is usable when you're writing a Java bytecode parser.

Q10. Will the following code compile?

```
@Target({ ElementType.FIELD, ElementType.TYPE, ElementType.FIELD })
public @interface TestAnnotation {
    int[] value() default {};
}
```

No. It's a compile-time error if the same enum constant appears more than once in an *@Target* annotation.

Removing the duplicate constant will make the code to compile successfully:

```
@Target({ ElementType.FIELD, ElementType.TYPE})
```

Q11. Is it possible to extend annotations?

No. Annotations always extend *java.lang.annotation.Annotation*, as stated in the Java Language Specification (http://docs.oracle.com/javase/specs/jls/se7/html/jls-9.html#jls-9.6).

If we try to use the *extends* clause in an annotation declaration, we'll get a compilation error:

```
public @interface AnAnnotation extends OtherAnnotation {
    // Compilation error
}
```

Conclusion

In this article, we covered some of the frequently asked questions appearing in technical interviews for Java developers, regarding annotations. This is by no means an exhaustive list, and should only be considered as the start of further research.

We, at Baeldung, wish you success in any upcoming interviews.

Next »
Top Spring Framework Interview Questions (https://www.baeldung.com/spring-interview-questions)

« Previous
Java Exceptions Interview Questions (+ Answers) (https://www.baeldung.com/java-exceptions-interview-questions)

>> Get the eBook

newest oldest most voted

Harshavardhan Musanalli (http://www.harshavmb.com)

Guest

Nice article. Crisp and clear. Thanks

0

1 year ago

Javin Paul (http://javarevisited.blogspot.com/)

Guest

Nice list, Annotations are a topic which many developer not prepare so well, these questions will surely help to bridge that gap. Btw, I have also compiled core Java interview questions from last 5 years @ https://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html (https://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html), let me know how do find it.

0

1 year ago

Eugen Paraschiv (http://www.baeldung.com/)

Guest

Looks quite useful Paul.
Cheers,
Eugen.

0

1 year ago

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)

SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)