# Theory: Annotations



13 minutes 1 / 5 problems solved



In Java, **annotations** are a kind of metadata that provide information about a program. They can mark classes, methods, fields, variables, and other elements of a program.

Annotations can be used for different purposes:

- to provide information for the compiler;
- to provide information for development tools to generate code, XML files and so forth;
- to provide information for frameworks and libraries at runtime.

## §1. Marking elements with annotations

To mark an element (class/method/field/etc) with an annotation we need to write the symbol `@` followed by the annotation name. For example:

```
@Deprecated
```

The symbol `@` signals to the compiler that this is an annotation.

## §2. Built-in annotations

Java has three built-in general-purpose annotations:

- `@Deprecated` indicates that the marked element (class, method, field and so on) is deprecated and should no longer be used. This annotation causes a compile warning if the element is used.
- `@SuppressWarnings` instructs the compiler to disable the compile-time warnings specified in the annotation parameters. This annotation can be applied to classes, methods, fields, local variables and other elements.
- `@Override` marks a method that overrides a superclass method. This annotation can only be applied to methods. We will consider it in a separated topic about overriding methods.

Let's annotate a method as an example:

```
@Deprecated
public static void method() {
    System.out.println("an old method");
}
```

This annotation indicates that `method` should not be used.

When it comes to `@Deprecated`, you can even deprecate a whole class:

```
@Deprecated
class MyClass {
    // fields and methods
}
```

We can also mark a class/method/field/etc by two or more annotations!

## §3. Annotation elements

Some annotations include elements and those elements have a type. For example, the annotation `@SupressWarnings` has an element named `value` to specify what type of warning we'd like to disable.

```
@SuppressWarnings(value = "unused")
public static void method() {
    int a = 0;
}
```

Current topic:
Annotations
Topic depends on:
Stage 6
Instance methods
Topic is required for:
Detecting annotations
Rest controller
Introduction to JPA
JUnit and Mockito

Now, you should remember only two possible values: `"unused"` and `"deprecation"`. The first one instructs the compiler to suppress warnings about unused local variables, the second one — to suppress warnings about using deprecated elements.

If an annotation has just one element and it is named `value`, then the name of this element can be omitted (this is a so-called unnamed element):

```
@SuppressWarnings("unused")
public static void method() {
    int a = 0;
}
```

An annotation element can also be an array. Here, for instance, `value` is an array:

```
@SuppressWarnings({"unused", "deprecation"})
public static void method() { ... }
```

The code above is correct because the single element of `@SuppressWarnings` is an array.

If an element of annotation has a default value, we can skip it. The element `value` of `@SuppressWarnings`, for example, has no default value, therefore it cannot be skipped. The following syntax is not correct:

```
@SuppressWarnings // it's not a correct syntax for this annotation
public static void method() { ... }
```

## §4. A custom annotation example

In this topic, we won't create any new annotations. Let's assume, we have an external library that includes two annotations `@NotNull` and `@Range`. Both annotations mark classes, fields, methods, and parameters.

The `@NotNull` annotation indicates that:

- a method should not return `null`;
- a variable cannot be `null`.

The `@Range` annotation indicates that:

- a method returns an integer number that belongs to the specified range;
- a variable always belongs to the specified range.

To demonstrate how these annotations function, here is a class that represents a computer game character.

```
class GameCharacter {

    @NotNull
    private String login;

    @Range(min = 1, max = 100)
    private int level = 1;

    public GameCharacter(

            @NotNull String login,

            @Range(min = 1, max = 100) int level) {



        this.login = login;

        this.level = level;

    }



    @NotNull

    public String getLogin() {

        return login;

    }



    @Range(min = 1, max = 100)

    public int getLevel() {

        return level;

    }

}
```

These annotations may be used by a static code analyzer to check your code and show some warnings.

Of course, the provided class and annotations are just an example. But in large applications, classes and their members may have a lot of annotations.

**Note:** the annotation `@NotNull` is taken from JetBrains. The annotation `@Range` is written just as an example, but some frameworks and libraries have an annotation with the same name but a slightly different meaning.

Report a typo

**249** users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (5) Hints (0) Useful links (1) Show discussion