


Theory: Detecting annotations

 13 minutes

0 / 3 problems solved

Skip this topic

Start practicing

413 users solved this topic. Latest completion was about 17 hours ago.

We’ve already learned what annotations are and when they are used. Do you remember any of them? The most used ones in the Java library are `@Deprecated` and `@Override`, but there are many other annotations in third-party libraries. For example, the Spring library has lots of them, and there’s a useful `@Test` annotation for testing Java classes. You can quickly go through [this page’s code examples](#) just to see how often they might be used in practice. Now let’s find out how they actually work and what magic happens when they are present.

§1. `getDeclaredAnnotations` method

Actually, annotations are just markers: they indicate that some class, method, or field has special properties. These annotations are further analyzed with the help of reflection when compiling or running the program.

Let’s look at the following class:

Current topic:

[Detecting annotations](#)

...

Topic depends on:

- ✓

[Annotations](#)

...
- ✗

[Manipulating fields and methods](#)

...

Table of contents:

- 1

[Detecting annotations](#)
- §1.

[getDeclaredAnnotations](#)

[method](#)
- §2.

[Code example](#)
- §3.

[Conclusion](#)
- [Feedback & Comments](#)

```

1  class Item {
2      @Deprecated
3      public static final int maxItems = 100;
4      public static int inStock = 19;
5
6      private String name;
7      protected int basePrice;
8
9      public Item(String name, int basePrice) {
10
11          this.name = name;
12
13          this.basePrice = basePrice;
14      }
15
16      public String getName() {
17
18          return name;
19      }
20
21      public int getPrice() {
22
23          return (int) (basePrice * getMarkUp());
24      }
25
26      public boolean haveSuchAmount(@Deprecated int amount) {
27
28          return inStock >= amount;
29      }
30
31      @Deprecated
32      protected double getMarkUp() {
33
34          double markUp = 0.1;
35
36          // ... connecting to the remote server
37
38          return 1 + markUp;
39      }
40
41      @Override
42      public String toString() {
43
44          return getName() + " " + getPrice();
45      }
46  }

```

It has three `@Deprecated` annotations and one `@Override` annotation. Let's try to analyze them by using reflection.

For working with annotations, `Class`, `Method`, `Field` and `Constructor` classes have the `getDeclaredAnnotations()` method, that returns an array of `Annotation` objects. This method returns an array, not an object because multiple annotations can be applied to a single field or method. If you follow the link above, the first example there will contain a huge pile of six annotations before the class!

\$2. Code example

Let's execute the following code. Here we go through all the names of methods and fields of our `ItemClass` and apply the `getDeclaredAnnotations()` method to get their annotations and print them next to the respective names:

```
1  Class itemClass = Item.class;
2
3  for (Field field : itemClass.getDeclaredFields()) {
4      for (Annotation annotation : field.getDeclaredAnnotations()) {
5          System.out.print(annotation + " - ");
6      }
7      System.out.println(field.getName());
8  }
9
10
11 for (Method method : itemClass.getDeclaredMethods()) {
12     for (Annotation annotation : method.getDeclaredAnnotations()) {
13         System.out.print(annotation + " - ");
14     }
15     System.out.println(method.getName());
16 }
17 }
```

Take a look at the output:

```
1  @java.lang.Deprecated(forRemoval=false, since="") - maxItems
2  inStock
3  name
4  basePrice
5  toString
6  getName
7  @java.lang.Deprecated(forRemoval=false, since="") - getMarkUp
8  getPrice
9  haveSuchAmount
```

You can see that though we had three instances of annotations in our class, we manage to output only two of them. `@Deprecated` appeared in line with `maxItems` field and `getMarkUp` method as we might have expected. However, for some reason, there is no `@Override toString` line in here. Let's try to find out why by ourselves and this way learn more about annotations and reflection. We may suppose that the absence of `@Override` has to do with its method: actually, `toString` is not declared in this class, but in the `Object`. To check our hypotheses let's use another method for getting class methods `getMethods`. As you probably remember, it returns all public methods of the class, including the inherited ones.

```
1  for (Method method : itemClass.getMethods()) {
2      for (Annotation annotation : method.getDeclaredAnnotations()) {
3          System.out.print(annotation + " - ");
4      }
5      System.out.println(method.getName());
6  }
```

If our suggestion is correct, this time we will get the `@Override` annotation for `toString` method. But look what we get by the above snippet:

```
1 toString
2 getName
3 getPrice
4 haveSuchAmount
5 wait
6 wait
7 wait
8 equals
9 @jdk.internal.HotSpotIntrinsicCandidate() - hashCode
1
0 @jdk.internal.HotSpotIntrinsicCandidate() - getClass
1
1 @jdk.internal.HotSpotIntrinsicCandidate() - notify
1
2 @jdk.internal.HotSpotIntrinsicCandidate() - notifyAll
```

Still, something is not right! As you see, we now got four methods with unfamiliar annotations, and the `toString` method still has no annotation. Let’s not focus on the ones containing the words *jdk* and *internal*, we have no intention to go there for now and just reproduce them for the sake of credibility. Note that this time we didn’t get `getMarkUp` method at all since it is *protected*.

Hence we may conclude that the reason lies in the nature of the annotation itself. Indeed, `@Override` annotation is a compile-time annotation and is not present when the program is running. While `@Deprecated` is present during the execution of the program, which means that it is a runtime annotation.

§3. Conclusion

Reflection package provides functionality to retrieve annotations from source code. `Class`, `Field` and `Method` classes have `getDeclaredAnnotations` method that returns a list of configured runtime annotations. Other annotations are not available at runtime, because the compiler erasures them.

 Report a typo

41 users liked this theory. 0 didn’t like it. What about you?



Start practicing

[Comments \(1\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)