Quiz yourself: AllowList usage and shortcuts for annotations

JAVA SE

Quiz yourself: AllowList usage and shortcuts for annotations

Some acceptable shortcuts for annotations are inconsistent with Java's usual syntax.

by Mikalai Zaikin and Simon Roberts

January 25, 2021

If you have worked on our quiz questions in the past, you know none of them is easy. They model the difficult questions from certification examinations. We write questions for the certification exams, and we intend that the same rules apply: Take words at their face value and trust that the questions are not intended to deceive you but to straightforwardly test your knowledge of the ins and outs of the language.

In this Java SE 11 quiz, assume that you are writing a source code parsing framework. You want to allow the framework's user to declare a list of programming languages supported by a parser, and to support this you created the following annotation:

```
@interface AllowList {
    String[] value();
}
```

Your colleague wants to use that framework to implement a Java parser and started writing the following code:

```
class JavaParser extends GenericParser {
    public void setLanguage(@AllowList("Java"
        … // method code here
    }
}
```

Which statement is correct? Choose one.

A. The class is valid and compiles as it is.

The answer is A.

B. The annotation usage is incorrect and should be `@AllowList({"Java"})`.

The answer is B.

C. The annotation usage is incorrect and should be `@AllowList(value={"Java"})`.

The answer is C.

D. The annotation usage is incorrect and should be `@AllowList(value="Java")`.

The answer is D.

Answer. Annotations allow the attachment of incidental information to program constructs (such as the method argument declaration in the question). The information takes the form of key-value pairs. *Java Language Specification* describes annotations in detail. However, that's more information than you need for this question.

The declaration of the `@interface AllowList` (which is the basis of creating an annotation) tells you that this annotation supports one key, called `value`, and that the type of the associated value will be `String[]` (that is, an array of `String`).

A very common form for using an annotation is to place the `@`, followed by the annotation's name, followed by parentheses containing a comma-separated list of key-value pairs specified in the form `key-name=literal-value`. In this example, the key name is `value` and the value would be specified using an array of `String` in a literal form. This means that the conventional format for this would be

```
@AllowList(value={"Java"})
```

This gives some credence to the idea that option C looks tempting. However, annotations allow some syntactic shortcuts.

It's quite common for an annotation to define a default value for an element, and in that case, there's no requirement to include a specification of a value for that element when the annotation is used. In this case, the annotation has one element but no default value, so that doesn't seem to affect this question.

Sometimes annotation usage needs to explicitly provide a value for only one key. This can happen either because there is only one key, as in this case, or because all the other keys have default values that are acceptable in this usage. If this situation

arises, it turns out that the key name `value` is a special case. Instead of specifying `key-name=literal-value`, you can simply specify `literal-value`. This tells you that it's fine to express the annotation as

```
@AllowList({"Java"})
```

So, at this point, option B looks tempting, too, and that creates some ambiguity. Well, you're not done with shortcuts yet, so let's continue the discussion.

Sometimes you want to provide only a single value to an annotation's element of array type, as is the case in this example. In that case, you can omit the curly braces (notice that this feature is quite inconsistent with the rest of Java's syntax, but it happens often enough that the shortcut is convenient). This means that you can equally well express the annotation as either

```
@AllowList("Java")
```

or

```
@AllowList(value="Java")
```

Notice that in these forms, there's no syntactic indication in the code that an array is permitted here. Of course, if multiple values are to be given, the shorthand no longer applies, and you would have to be more explicit, for example:

```
@AllowList({"Java", "Scala"})
```

From these discussions it's clear that the syntax used in the presentation of the question is valid, as are all the syntax variants proposed in options B, C, and D.

However, in the tricky way that options B, C, and D are written, each statement asserts that the code in the question is *invalid*. As a result, options B, C, and D are all incorrect, and option A is correct.

Conclusion: The correct answer is option A.

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.

Share this Page