Quiz yourself: Understanding valid annotation declarations

JAVA SE

Quiz yourself: Understanding valid annotation declarations

Explore the allowed method return types to see where the code succeeds… or fails.

by Simon Roberts and Mikalai Zaikin

January 11, 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, which code fragments are valid annotation declarations? Choose two.

A.

```
@interface Lock {
    String resource();
    LocalDateTime start();
    LocalDateTime end();
}
```

The answer is A.

B.

```
@interface ValidNumber {
    int value() = 3;
}
```

The answer is B.

C.

```
@interface Area {
    int[][] points();
}
```

The answer is C.

D.

```
@interface Allowlist {
    String VER = "1.0.0";
    String[] words();
}
```

The answer is D.

E.

```
enum Val {V1,V2,V3,V4,V5,V6};
@interface RandomValues {
    LocalDate lastUpdated = LocalDate.of(2020
    public abstract Val[] choices() default {
}
```

The answer is E.

Answer. This question explores two issues related to the declaration of annotations. One is the valid type of an annotation's elements, and the other is the ability to declare constants in an annotation.

Annotations are used in code to attach additional information to program constructs, that is, to parts of the source code. In a sense, they're like those yellow sticky notes that you might write a few words on and then stick on a page in a book. The following shows one representative example of their usage:

```
@SomeAnnotation(key=aLiteralValue)
<annotated program construct>
```

In this example, the annotated program construct might be any one of many things, such as a class, a method, a field, or something else. In this example, the annotation name is shown followed by parentheses and a key-value pair. The annotation's element mentioned above is essentially this key-value pair.

The elements of an annotation are declared in the annotation type itself as methods. That method's name defines the key, and calling that method returns the value of the key/value pair. (By the way, for this quiz, it's unnecessary to discuss *how* to obtain the runtime instance of an annotation.)

When the annotation is used, the value must be provided as a compile-time constant (commonly, but not exclusively, as a literal) in the source code. This constant value limits the types that can be used and, therefore, the types that the methods are permitted to return.

Java Language Specification section 9.6 documents the declaration of annotation types.

The allowed method return types, documented in the Java specification section noted above, are

- A primitive type
- `String`
- `Class` or an invocation of `Class`
- An `enum` type
- An annotation type
- An array type whose component type is one of the preceding types

In view of these rules, let's investigate this quiz question's options.

Option A attempts to use `LocalDateTime` with the type of the elements being `start` and `end`. This isn't on the list of acceptable types and is not permitted. A moment's reflection will tell you that there's no literal format defined in the Java programming language for `LocalDateTime` objects! So, you can tell that option A is incorrect.

Option B appears to be an attempt to define a default value for the `value` element of the annotation. However, this syntax is simply wrong. Default values are supported for annotation elements, but they are created using the keyword `default`, not an equal sign. From this you know that option B is incorrect.

Option C tries to declare an element that is a two-dimensional array of primitives. Although it's possible to represent this as a compile-time constant (again, often a literal), it's not a permitted form in an annotation. The last item in the allowed-return-types list states "an array type whose component type is one of the *preceding* types," yet the previous items do not list an array type. Remember that Java creates two-dimensional arrays as an array of arrays. This tells you that option C is incorrect.

Option D declares an element that's a one-dimensional array of `String`, which is fine. It also declares a constant: `String VER`. This is a normal behavior for an interface and is actually permitted for an annotation too. It's no different from any other interface constant, and in the same way, this one is `public`, `static`, and `final` even though this is not explicitly stated. (This, too, is the same behavior as if this were a regular interface.) Therefore, the annotation is valid, and option D is correct.

Option E declares an element called `choices`. The method that declares this has a return type of array of `enum`. Enum types are the fourth item in the list of valid annotation element types, and the fourth item is also included in the last item as a valid base type for an array. So, this aspect is valid.

The declaration also includes a default value using the keyword `default` and an array literal. This declaration demonstrates the correct form, unlike the one shown in option B. The method that declares all this is defined explicitly as `public` and `abstract`, which is redundant in any interface but nevertheless valid.

Finally, this option declares a constant of type `LocalDate`. While `LocalDate` is not permitted as an element type, this is simply a constant and not subject to the same constraints. From these observations it’s clear that option E is correct.

Conclusion: The correct answers are options D and E.

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.

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.

Share this Page