JAVA SE

Quiz yourself: Define the structure of a Java class (intermediate)

Test your knowledge of Java classes, such as their valid names, the use of variables inside a method, and the number of import statements.

by Simon Roberts and Mikalai Zaikin

September 15, 2020 | Download a PDF of this article
More quiz questions available here

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. The “intermediate” and “advanced” designations refer to the exams rather than to the questions, although in almost all cases, “advanced” questions will be harder. 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.

The objective here is to test your knowledge of Java classes.

Which of the following statements are correct about a Java class? Choose two.

A. A Java class must have a name shown in the source code.

The answer is A.

B. A Java class may have several local variables with the same name inside the same method.

The answer is B.

C. A Java class may have several `import` statements.

The answer is C.

D. An underscore character ("_") is a valid Java class name.

The answer is D.

Answer. Let's start by saying that option A is incorrect. Not all classes have an explicit name shown in the source code. Although this is not a topic of the first-level exam, Java provides anonymous classes, such as this:

```
Runnable r = new Runnable(){ public void run(
    System.out.print("Do nothing!");
}};
r.run();
```

The `run` method of the `Runnable` interface is abstract, and yet you can see that there is a real object because you instantiated it and can invoke the `run` method. This shows that some concrete class, which implements the `Runnable` interface, exists. The variable `r` is a reference to an instance of that class, but the class name is not known in the source code.

Variables are visible only within the *scope* in which they were defined. But since a block bounded by curly braces defines a scope, you can create two sibling scopes inside one method. If you do this, two variables with the same name can coexist without a problem, like this:

```
void twoVars() {
    { int i = 0; }
    { int i = 1; } // OK
}
```

In view of this, option B is correct.

Option C discusses multiple `import` statements. Having multiple `import` statements is not merely permitted; in most cases, it's necessary to have many `import` statements to provide access to classes in different packages. It's also typical to have multiple imports providing access to each of several classes in the same package, rather than using wildcards.

Even repeating imports for the same class or package is syntactically valid, though it would probably trigger a request during code review to tidy up the code, so the following is completely legal:

```
import java.util.*;
import java.util.*;

public class MyClass { // OK
}
```

By the way, if a class defines more than one `package` statement —whether specifying the same package name or a different package name—compilation would fail. Thus, the following would not compile:

```
package a.b.c;
package a.b.c; // FAILS

public class MyClass {
}
```

But since option C asks if only multiple `import` statements are permitted, option C is correct.

As for option D, through Java 8, the single underscore character was a valid identifier and could be used as a class name, a method name, or a variable name. However, starting in Java 8, a warning during compilation indicated that this character was reserved for future language changes. However, the warning did not prevent its use.

Beginning with Java 9, the single underscore character was defined as a keyword and is, therefore, no longer valid as an identifier. As of Java 14, the single underscore keyword still does nothing and is simply being reserved for future use.

This change was described in the Java 9 summary of changes: “The underscore character is not a legal name. If you use the underscore character ("_") an identifier, your source code can no longer be compiled.”

By the way, it is still legal to use a double underscore as an identifier, such as for a class name, a method name, or a variable name. You can also start a variable name with a single underscore.

```
public class MyClass {
    int __; // Double underscore identifier
}
```

Because the single underscore is a keyword and is not a legal identifier on its own any longer, option D is incorrect.

Therefore, the correct answers are options B and C.

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