JAVA SE

Quiz yourself: Module definitions and automatic modules

The quiz experts explore the Java modular system and how it uses JAR files.

by Mikalai Zaikin and Simon Roberts

August 9, 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.

Imagine you have the following nonmodular JAR file containing the compiled `Animal` class:

```
> jar -tvf animal-domain-1.0.jar
META-INF/MANIFEST.MF
Animal.class
```

While you were developing the modular application, you created a module called `client` containing the class `app.Main`. The `client` module declares variables of the `Animal` class type and instances of that class. The `META-INF/MANIFEST.MF` does not contain module-related attributes.

Which module definition (if any) allows you to compile the client module using the nonmodular JAR file as an automatic module? Choose one.

A.

```
module client {
    requires animal-domain;
    exports app;
}
```

The answer is A.

B.

```
module client {
    requires animal.domain;
    exports app;
}
```

The answer is B.

C.

```
module client {
    requires animal-domain-1.0;
    exports app;
}
```

The answer is C.

D.

```
module client {
    requires animal-domain-1.0.jar;
    exports app;
}
```

The answer is D.

E. The `animal-domain-1.0.jar` file may not be used as an automatic module.

The answer is E.

Answer. In general, placing a pre-Java 9 JAR file—that is, a JAR file that does *not* contain a `module-info` file—on the module path creates an *automatic module* if another regular module refers to it, such as by using a `requires` directive referring to that automatic module.

One question that must be answered is how an automatic module gets a name. This is documented in the ModuleFinder class's documentation. The rules are quite detailed, but for this quiz's purposes, a simplified (and therefore not fully correct) view of this is that the filename is stripped of the extension (`.jar`) as well as any parts of the name that look like a version identity. Any nonalphanumeric characters are also changed to dots.

Therefore, the filename `animal-domain-1.0.jar` would be stripped down to create an automatic module named `animal.domain`. This gives the impression that option B is correct. However, there is another problem.

If you look at the JAR file structure, notice that the `Animal` class is in the default (that is, unnamed) package. *Java Language Specification*

section 7.4.2 for unnamed packages notes the following:

> Unnamed packages are provided by the Java SE Platform principally for convenience when developing small or temporary applications or when just beginning development.

And it further states the following (emphasis added):

> The host system must associate ordinary compilation units in an unnamed package with an unnamed module, *not a named module*.

In other words, a Java module cannot contain classes from the unnamed package, and all the classes in a module must belong to a unique, named package. This tells you that the `animal-domain` JAR file cannot be used as an automatic module at all. Therefore, option E is correct.

By the way, the `animal-domain-1.0.jar` file can only possibly be used in a nonmodular environment. It also turns out that classes in the unnamed package can be used only by other classes that are also in the unnamed package. Thus, the usefulness of the reference to the `Animal` class is severely limited.

Conclusion. The correct answer is option E.

Related quizzes

- Quiz yourself: Declaring and accessing modules
- Quiz yourself: Migrating to the Java Platform Module System
- Quiz yourself: Describe the modular SDK

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

© Oracle | Site Map | Terms of Use & Privacy | Cookie Preferences | Ad Choices