JAVA SE

Quiz yourself: Working with pluggable service provider modules in Java

It's essential to know when—and where—to use the exports, requires, and provides directives in module declarations.

by Mikalai Zaikin and Simon Roberts

August 30, 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 that a modular Java application you are developing for linguistics uses pluggable service providers. The following interface has been defined for a service:

```
package com.lang;
public interface TranslatorFactory {
    Translator getTranslator(String sourceLang, St
}
```

The interface is declared in a module that has the following declaration:

```
module com.lang.service {
    exports com.lang;
}
```

One implementation of the service is defined in this class

```
package com.lang.online;
import com.lang.Translator;
```

```
import com.lang.TranslatorFactory;
public class OnlineTranslatorFactory implements Tr
    ... // code here
}
```

which is located in a module that carries this declaration

```
module com.lang.online {
    requires com.lang.service;
    provides com.lang.online.OnlineTranslatorFacto
        com.lang.TranslatorFactory;
}
```

Finally, the client application module is declared as follows:

```
module com.lang.app {
    requires com.lang.service;
    uses com.lang.TranslatorFactory;
}
```

Attempting to compile the application fails on one of the modules. What is the reason? Choose one.

A. There is a problem with the `exports` directive in the `com.lang.service` module declaration.

The answer is A.

B. There is a problem with the `requires` directive in the `com.lang.online` module declaration.

The answer is B.

C. There is a problem with the `provides` directive in the `com.lang.online` module declaration.

The answer is C.

D. There is a problem with the `requires` directive in the `com.lang.app` module declaration.

The answer is D.

Answer. The public types contained in a package that one module exports are available for use by other modules that declare they require the *module* that exported the package. Note particularly that the `exports` directive refers to a *package* name. When publishing the description of a service, in this case `TranslatorFactory`, all that's needed is for the describing type to be accessible in the modules that will provide implementations and in those that will use the service. In this case, these aspects are correctly coded; therefore, option A is incorrect.

In a module declaration, each occurrence of the `requires` directive must be followed by a single *module* name, and this is correct in both

the `com.lang.app` and `com.lang.online` module declarations. This tells you that option B is incorrect.

When a module wants to *use* a service, the module must be coded against the service description (again, `TranslatorFactory` in this case). So it needs a `requires` directive that refers to the module describing that (`com.lang.service` in this case) along with the `uses` directive. However, a key concept in a service-based system is that the module using the service should not be dependent upon the module that implements the service. Because of this, in the Java Platform Module System (JPMS) the implementing modules are simply placed on the module path. They are not mentioned in the using module's declaration. From this and the previous paragraph, you can see that the module declaration of `com.lang.app` is correct, and option D is incorrect.

A module that provides a service implementation declares this using the `provides` directive. The syntax for this is as follows:

```
provides <Service Describing Type> with <Service I
```

Notice that the directive should specify the service description type first and the implementing class second. However, in this question these two types are in the wrong positions. This is the cause of the compilation failure; therefore, option C is correct.

Conclusion. The correct answer is option C.

Related quizzes

- Quiz yourself: Declaring and accessing modules
- Quiz yourself: Migrating to the Java Platform Module System
- Quiz yourself: Module definitions and automatic modules

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