Modules

Exercises and solutions

1. What is a module?

**Answer:**

A module is a group of packages. It specifies the accessibility for its packages to other modules and its dependence on other modules.

1. What keyword do you use to declare a module?

**Answer:**

The module keyword is used to declare a module.

1. What are the rules to specify a module name? Which of the following module names are valid?  
     
   jdojo.dashboard  
   $jdojo.$dashboard  
   jdojo.policy.1.0  
   java9Fundamentals

**Answer:**

A module name is a qualified Java identifier which is a sequence of one or more Java identifiers separated by a dot. A java identifier is sequence of one or more Unicode letters and digits and must start with a letter.

jdojo.dashboard // ok  
$jdojo.$dashboard // ok  
jdojo.policy.1.0 // error for 1.0  
java9Fundamentals // ok

1. List all restricted keywords that are treated as keywords only when used in specific positions in a module declaration.

**Answer:**

open, module, requires, transitive, exports, opens, to, uses, provides, and with

1. What module statement do you use to export a package to all other modules or to a set of named modules?

**Answer:**

You use the exports statement to export a package of a module to all other modules or to a set of named modules.

An unqualified export is used to exports a package of a module to all other modules.

exports <package>;

A qualified or module-friendly export is used to export a package of a module to a set of one or more named modules.

exports <package> to <friend-module> [, <friend-module>...] ;

1. Consider the following declaration for a module named jdojo.core:  
     
   module jdojo.core {  
    exports com.jdojo.core to jdojo.ext, jdojo.util;  
   }  
     
   Explain the effect of the exports statement in this module declaration. Do these two modules, jdojo.ext and jdojo.util, have to exist when the jdojo.core module is compiled?

**Answer:**

The jdojo.core module uses a qualified export to export the com.jdojo.core package to only two modules named jdojo.ext and jdojo.util.

No. The jdojo.ext and jdojo.util modules do not have to exist for the jdojo.core module to compile. The jdojo.core module compiles with warnings that the jdojo.ext and jdojo.util are not found.

1. What module statement do you use to express dependence of a module to another module? What is a transitive dependence and what are the benefits of using a transitive dependence?

**Answer:**

The requires statement is used to specify a module’s dependence on another module. If module M declares a transitive dependence on module N, any module declaring a dependence on module M declares an implicit dependence on module N. Module N can be refactored easily without affecting module M and all modules that depend on M.

1. Consider the following declaration for a module named jdojo.ext:  
     
   module jdojo.ext {  
    requires jdojo.core;  
   }  
     
   What are the two modules the jdojo.ext reads?

**Answer:**

The jdojo.ext module reads two modules: jdojo.core and java.base. It reads the jdojo.core module explicitly and the java.base module implicitly.

1. How do you express a dependence of a module on another module that is mandatory at compile-time, but optional at runtime?

**Answer:**

Using the static modifier in the requires statement.

module jdojo.claim {

requires **static** jdojo.anotation;

}

1. What is an open module? When do you use an open module?

**Answer:**

An open module allows deep reflection on all types in all packages in that module at runtime. Open module is useful when deep reflective access is needed.

1. What is the difference between an open module and opening the packages of a module selectively? Why can't you use the opens statement inside an open module?

**Answer:**

Deep reflection is allowed on all types on all packages in an open module, whereas only the selected packages are open for deep reflection (using opens statement). Because an open module opens all packages for deep reflection, an opens statement is not allowed inside an open module.

1. Consider the following declaration for a module named jdojo.misc:  
     
   module jdojo.misc {  
    opens com.jdojo.misc;  
    exports com.jdojo.misc;  
   }  
     
   Is this module declaration valid? If it is valid, explain the effects of opening and exporting the same package of the module.

**Answer:**

Yes. This module declaration is valid. All public types and their public members of the com.jdojo.misc package are accessible at compile-time and the package allows deep reflection at runtime.

1. Can you have two modules that contain the same package? Describe the exact rule that prohibits two modules having the same package?

**Answer:**

No. Same package in multiple modules must not be readable to a module at the same time. If two modules named M and N define the same package named P, there must not exist a module Q such that the package P in both M and N modules is accessible to Q.

1. What is an automatic module? Describe two ways that the name of an automatic module can be specified or derived?

**Answer:**

A JAR, not a modular JAR, placed on the module path is treated as a module, which is called an automatic module. The name of an automatic module can be derived from two sources:

* From the attribute named Automatic-Module-Name in the MANIFEST.MF file
* From the name of JAR file if the MANIFEST.MF file does not contain the automatic module name.

1. What is an unnamed module? If you place a modular JAR on the class path, will all types from the modular JAR be members of an unnamed module?

**Answer:**

Every class loader defines an unnamed module whose members are all types the class loader loads from the class path. Yes. The types loaded from a modular JAR placed on the class path are loaded into unnamed module of the class loader.

1. What is an aggregator module? Name an aggregator module in the JDK 9.

**Answer:**

An aggregator module collects and re-exports the contents of other modules. It contains no code of its own. One such JDK 9 module is java.se.ee.

1. What is the fully qualified class name of the class that represents a module at runtime?

**Answer:**

java.lang.Module

1. How do you get the reference of the module that a class belongs to at runtime?

**Answer:**

Using the getModule() method of Class class. For example, the following snippet of code gets the reference of the Module object of which the Person class is a member:

Module m = Person.class.getModule();

1. Consider the following snippet of code assuming that a Person class exists:  
     
   Person john = new Person();  
   String moduleName = john./\* Complete the code \*/;  
   System.out.println("Module name of Person class is " + moduleName);  
     
   Complete this snippet of code by replacing the comment in the second line with your code. This snippet of code is supposed to print the name of the module the Person class is a member of or null if it is a member of an unnamed module.

**Solution:**

Person john = new Person();  
String moduleName = john.**getClass().getModule().getName();**  
System.out.println("Module name of Person class is " + moduleName);

1. What option do you use with the jar tool and java tools to describe a module?

**Answer:**

--describe-module

1. If you are given a module-info.class file that contains the compiled code for a module declaration, how will you get the source code of the module? In other words, what tool do you use to disassemble a class file, which can also be a module-info.class file?

**Answer:**

Use the javap tool, located in the JDK\_HOME\bin directory, to disassemble a module-info.class file.

javap <path-to-the-module-info.class-file>

1. JDK modules are stored in an internal format called JIMAGE. What is the name of the new scheme that JDK 9 has introduced to access the class files and resources of JDK module?

**Answer:**

The name of the scheme is jrt, which is short for Java runtime. The syntax to use the jrt scheme is as follows:

jrt:/<module>/<path-to-a-file>

1. Use the javap command to print the declaration of the java.sql module, which is a JDK module.

**Answer:**

javap jrt:/java.sql/module-info.class

Compiled from "module-info.java"

module java.sql@9.0.1 {

requires transitive java.xml;

requires transitive java.logging;

requires java.base;

exports javax.transaction.xa;

exports javax.sql;

exports java.sql;

uses java.sql.Driver;

}