

Language changes for Modules

Syntax

Example 1

```
module M1 @ 1.0 {
  requires M2 @ 2.0, M3 @ 3.0;
  provides M4 @ 4.0, M5 @ 5.0;
  permits M6;
  class com.foo.bar;
}
```

Example 2

```
import com.mycorp.annotations.*;
@Foo
module M @ 5.0u7:SPARC for osgi+springdm @ 4.2 {
  @Bar
  requires N;

  @Quux
  requires package P @ [1.0,2.0) ;

  @Ping
  requires X:5.0 vendor=S;

  @Pong
  classpath a.jar b.jar c.jar;
}
```

Grammar

x_{opt} indicates zero or one occurrences of x . $\{x\}$ denotes zero or more occurrences of x .

CompilationUnit:

```
ImportDeclarations_opt ModuleDeclaration_opt PackageDeclaration_opt
ImportDeclarations_opt TypeDeclarations_opt
```

ModuleDeclaration:

```
Annotations_opt 'module' Name ModuleSystem_opt '{' { Directive } '}'
```

ModuleSystem:

```
'for' Name
```

Name:

```
Word Version_opt
```

Word:

```
WordCharacter
Word WordCharacter
```

WordCharacter:

```
InputCharacter but not VersionSeparator or WhiteSpace or \ or } or ; or ,
// InputCharacter is any UnicodeInputCharacter except CR and LF
// Must manually exclude WhiteSpace, akin to how Identifier excludes it by allowing
only 'Java letters and digits'
// Would like to exclude Separator but ()[] must be permitted
```

```

EscapeSequence
// Module support this get-out clause

Version:
  VersionSeparator Word

VersionSeparator:
  '@'
  '='
  ':'

Directive:
  Annotations_opt Word { Name } ';'

Modifier:
  Annotation
  'public'
  'module'
  'protected'
  ...

```

Semantics

Module membership

A module is a set of types in one or more named packages. The membership of a type in a module is determined by the host system. The host system should determine the same module membership for all types declared in the same compilation unit.

If the host system does not determine the module membership of a type, then that type is in the *unnamed module*. A type in an unnamed package is a member of the unnamed module.

Module accessibility

`module` is a restricted keyword in the `ClassDeclaration` and `MethodOrFieldDecl` productions of a `CompilationUnit`.

(A restricted keyword is a character sequence that is an identifier except where it appears as a terminal in a specific production, whereupon the terminal itself is defined as a keyword. Restricted keywords permit character sequences that were formerly classified as identifiers to be introduced into the syntactic grammar of the Java programming language as if they are keywords in very limited contexts. This overcomes the problem that introducing keywords into the syntactic grammar otherwise makes identifiers with the same character sequences illegal. There must be no ambiguity between a restricted keyword and an identifier anywhere in the grammar.)

If a class or interface type is declared `public`, then it may be accessed by any code, provided that the compilation unit in which it is declared is observable.

If a class or interface type is declared `module`, then it may be accessed only from within the same module.

If a top level class or interface type is not declared `public` or `module`, then it may be accessed only from within the package in which it is declared.

A member (class, interface, field, or method) of a reference (class, interface, or array) type or a constructor of a class type is accessible only if the type is accessible and the member or constructor is declared to permit access:

- If the member or constructor is declared `public`, then access is permitted.
- If the member or constructor is declared `module`, then access is permitted only from within the module containing the class in which the module member or constructor is declared.



Module declaration

A module declaration specifies a new named module. The identity of a module is a name and optionally a version. There is no obscuring between module names and other names. This means there is no prohibition against a module having the same name as any package. The Java programming language assigns no structure to a version.

A module declaration may contain directives used by the host system to configure observability for types determined to be members of the declared module by the host system.

Discussion (non-normative): When packages are stored on a filesystem, the compilation unit that contains a module declaration is a file called `module-info.java`, stored in a directory corresponding to the module name.

The restricted keyword `module` in a module declaration may optionally be preceded by annotation modifiers. If an annotation `a` on a module declaration corresponds to an annotation type `T`, and `T` has a (meta-)annotation `m` that corresponds to `annotation.Target`, then `m` must have an element whose value is `java.lang.annotation.ElementType.MODULE`, or a compile-time error occurs.

A directive in the body of a module declaration may optionally be preceded by annotation modifiers. If an annotation `a` on a module declaration corresponds to an annotation type `T`, and `T` has a (meta-)annotation `m` that corresponds to `annotation.Target`, then `m` must have an element whose value is `java.lang.annotation.ElementType.MODULE_DIRECTIVE`, or a compile-time error occurs.