JAVA

Modules
Modules

• a module
  – explicitly defines what is provided but also what is required

• why?
  – the classpath concept is “fragile”
  – no encapsulation
Modules

- a module
  - explicitly defines what is provided but also what is required

why?
- the classpath concept is "fragile"
- no encapsulation

GRAPH THEORY FOR GEEKS

- this is a path
- and this is a classpath?
- yep
Modular apps – motivation

• why
  – applications get more complex
  – assembled from pieces
  – developed by distributed teams
  – complex dependencies
  – good architecture
    • know your dependencies
    • manage your dependencies
Module declaration

• module-info.java
  module com.foo.bar {
    requires com.foo.baz;
    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;
  }

• modular artifact
  - modular JAR – JAR with module-info.class
  - a new format JMOD
    • a ZIP with classes, native code, configuration,...
• JDK std library modularized too
  - java.base – always „required“

```java
module java.base {
    exports java.io;
    exports java.lang;
    exports java.lang.annotation;
    exports java.lang.invoke;
    exports java.lang.module;
    exports java.lang.ref;
    exports java.lang.reflect;
    exports java.math;
    exports java.net;
    ...
```
Module readability & module path

• When one module depends directly upon another

  Module reads another module (or, equivalently, second module is readable by first)

• Module path – equivalent to classpath
  – but for modules
    • -p, --module-path
module com.foo.app {
    requires com.foo.bar;
    requires java.sql;
}
Accessibility

- If two types S and T are defined in different modules, and T is public, then code in S can access T if:
  - S’s module reads T’s module, and
  - T’s module exports T’s package
Implied readability

• Readability is not transitive
  - example:

```java
java.sql.Driver {
    java.util.Logger getParrentLogger();
    ...
}
```

```text
module java.sql {
    requires public java.logging;
    requires public java.xml;
    exports java.sql;
    exports javax.sql;
    exports javax.transaction.xa;
}
```
module com.mysql.jdbc {
    requires java.sql;
    requires org.slf4j;
    exports com.mysql.jdbc;
    provides java.sql.Driver with com.mysql.jdbc.Driver;
}

module java.sql {
    requires public java.logging;
    requires public java.xml;
    exports java.sql;
    exports javax.sql;
    exports javax.transaction.xa;
    uses java.sql.Driver;
}
Qualified exports

• module java.base {
  ...
  exports sun.reflect to
  java.corba,
  java.logging,
  java.sql,
  java.sql.rowset,
  jdk.scripting.nashorn;
}

• not intended for common usage
requires static

- required at compile time, but is optional at runtime

```java
module com.foo.bar {
  requires static com.foo.baz;
}
```

- WARNING
  - code that uses required static package has to be prepared for unavailability
before Java 9, anything can be accessed via reflection
  - even private members
in Java 9+, reflection follows rules of modules
but – packages can be opened

```java
module com.foo.bar {
    opens com.foo.bar.alpha;
}
```

- types in opened package are accessible at runtime

```java
open module com.foo.bar {
}
```
- opens all its packages
• **opens** package **to** list-of-modules
  
  - opens to code in the listed modules only
package java.lang.reflect;

public final class Module {
    public String getName();
    public ModuleDescriptor getDescriptor();
    public ClassLoader getClassLoader();
    public boolean canRead(Module source);
    public boolean isExported(String packageName);

    ...
}

Layer

- layer – instantiation of module graph at runtime
- maps each module in the graph to the unique class loader

- layers can be stacked
  - a new layer can be built on top of another one
    - a layer’s module graph can be considered to include, by reference, the module graphs of every layer below it

- boot layer
  - created by VM at startup

- layers intended for app. servers, IDEs,...
Compatibility with “old” Java

• Classpath still supported
  – in fact – modules are “optional”

• Unnamed module
  – artefacts outside any module
    • “old” code
  – reads every other module
  – exports all of its packages to every other module
Automatic module

- a named module that is defined implicitly
  - it does not have a module declaration

- “regular” JAR placed on the module path rather than the class path
  - JAR without module-info.java