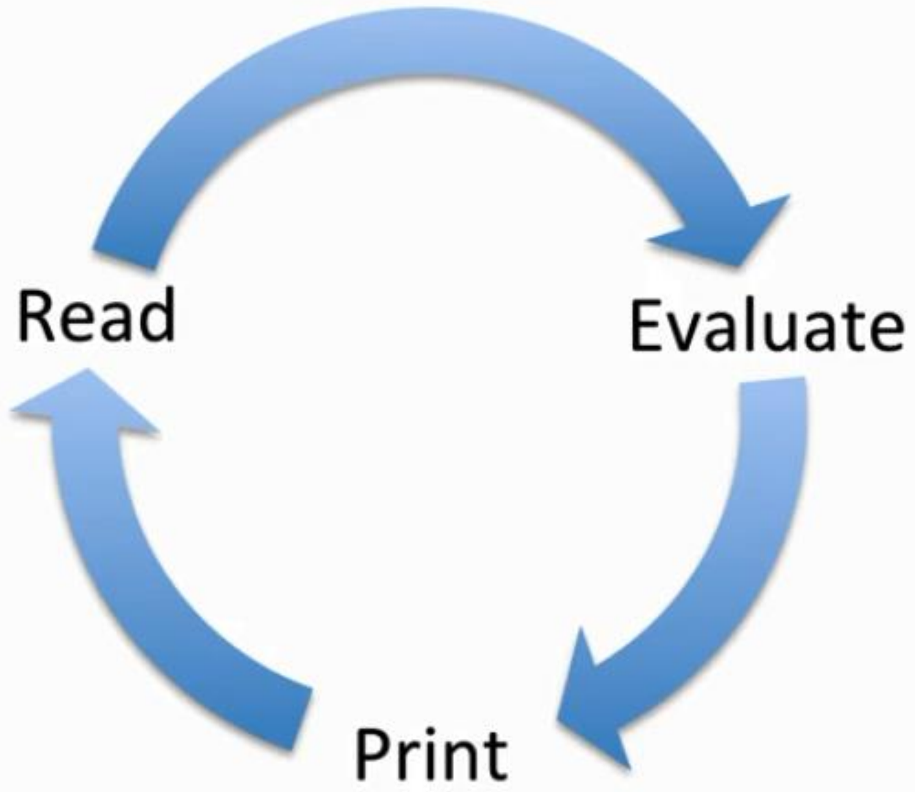
New Features – Java 9

The Java Shell (REPL) – JEP 222

"Aims to provide an interactive tool to evaluate declarations, statements, and expressions of the Java programming language, together with an API so that other applications can leverage this functionality"

http://openjdk.java.net/jeps/222

What’s a REPL



The Java Shell (REPL) – Motivation

“Immediate feedback is important when learning a programming language and its APIs.

The number one reason schools cite for moving away from Java as a teaching language is that other languages have a "REPL" and have far lower bars to an initial "Hello, world!" program”

The Java Shell (REPL) – First day of class

```
public class Main {

    public static void main(String[]  args) {

        System.out.println("Hello world");

    }
}
```

The Java Shell (REPL) – First day of class with a REPL

```
System.out.println("Hello world");
```

```
print("Hello world”)
```

The Java Platform Module System - Section Overview

- Similar REPLs exist in languages such as Python, Lisp, Scala, Swift, Ruby, JavaScript and many others.
- Useful for testing small code snippets
- Supports the expressions, statements, class declarations, interface declarations, method declarations, field declarations and import declarations

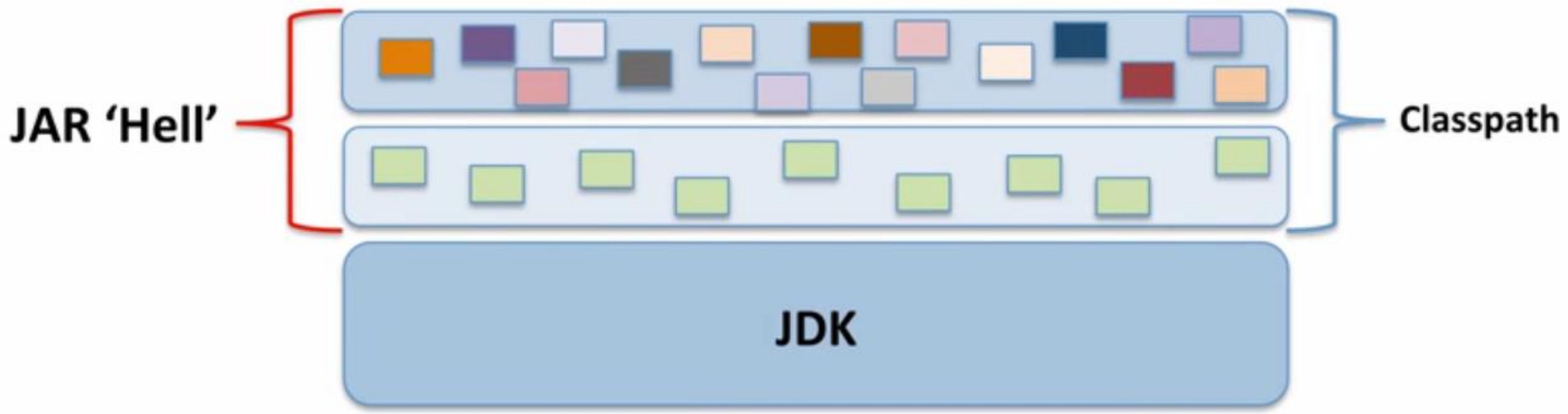
The Java Platform Module System - Section Overview

- Motivation and goals
- What is a module?
- Module dependencies
- Module directives (requires, exports, opens, uses,…)
- Module graphs
- Create a simple Joke Server application from no modules, to modules, to services, to allowing reflection using JavaFX
- Command line and IntelliJ IDEA
- Create modular JAR files
- Process of migrating existing Java code
- Creating a custom runtime

The Java Platform Module System - Motivation and Goals

- Java since 1995
- Millions of developers using Java
- JSR 277 – 2005 targeted for Java 7
- JSR 376 – targeted for Java 8

The Java Platform Module System - Motivation



The Java Platform Module System - Motivation and Goals

Why Classpath / JAR “Hell”?

- JARs can’t define dependencies
- Transitive dependencies
- Shadowing and version conflicts
- NoClassDefFoundError – at runtime!!

The Java Platform Module System - Motivation and Goals

JDK/JRE monolithic and very large

- And getting bigger with every Java release
- Very large apps
- JDK8 compact profiles

The Java Platform Module System - Goals

- Reliable Configuration
- Strong Encapsulation
- Scalable Java Platform
- Greater Platform Integrity
- Improved Performance and Security

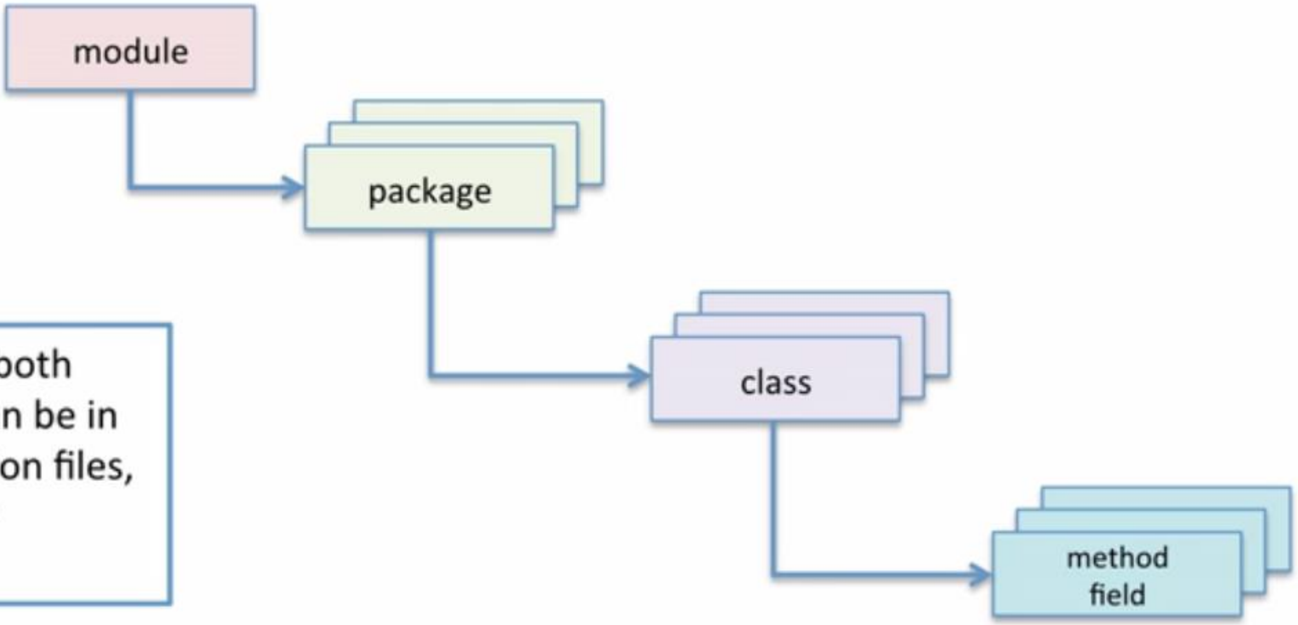
The Java Platform Module System - Modularity Specifications

- JEP 200 – The Modular JDK
- JEP 201 – Modular Source Code
- JEP 260 – Encapsulate most internal APIs
- JEP 261 – The Module System

- JEP 282 – jlink: The Java Linker
- JSR 376 – Java Platform Module System

- JEP 238 – Multi-Release JAR Files
- JEP 253 – Prepare JavaFX UI Controls and CSS APIs for Modularization
- JEP 260 – Encapsulate most internal APIs
- JEP 275 – Modular Java Application Packaging

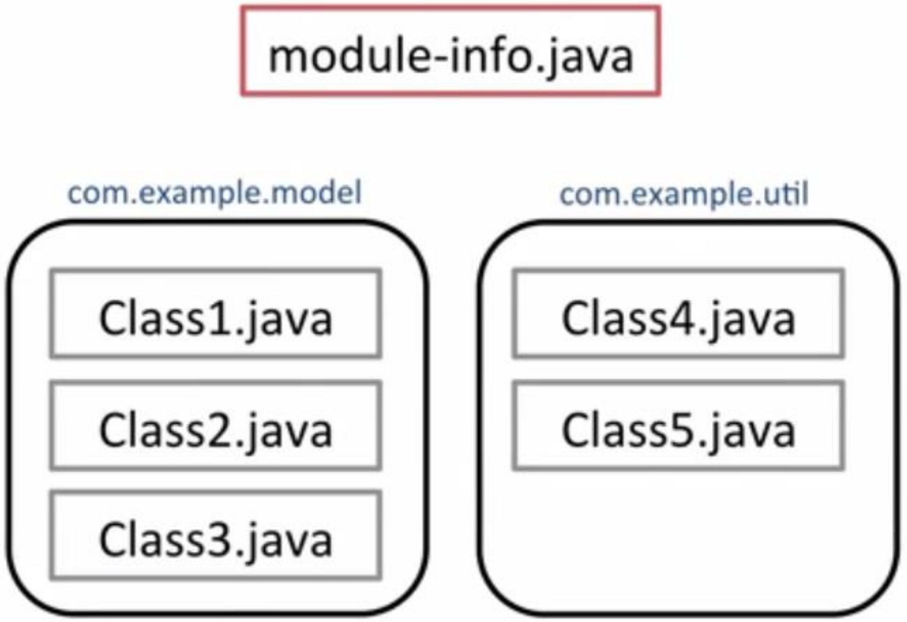
What is a Module?



A module can contain both code and data. Data can be in the form of configuration files, native code, and other resources.

What is a Module?

How do we declare a module?



What is a Module?

Directory Structure

Without Modules

```
src
└── com
    └── example
        ├── model
        │   ├── Class1.java
        │   ├── Class2.java
        │   └── Class3.java
        └── util
            ├── Class4.java
            └── Class5.java
```

With Modules

```
src
└── com.example.model
    ├── com
    │   └── example
    │       ├── model
    │       │   ├── Class1.java
    │       │   ├── Class2.java
    │       │   └── Class3.java
    │       └── util
    │           ├── Class4.java
    │           └── Class5.java
    └── module-info.java
```

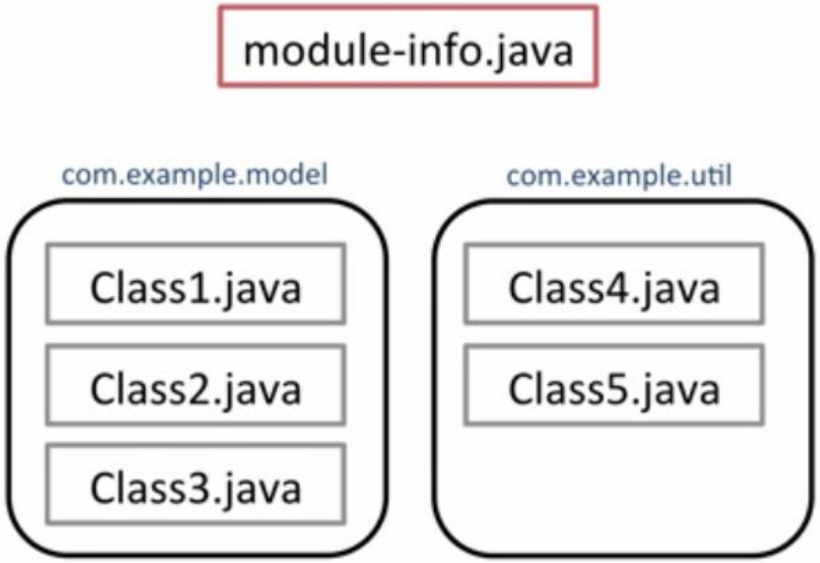
What is a Module?

Module Declaration

```
module com.example.model {

// module directives go here

}
```



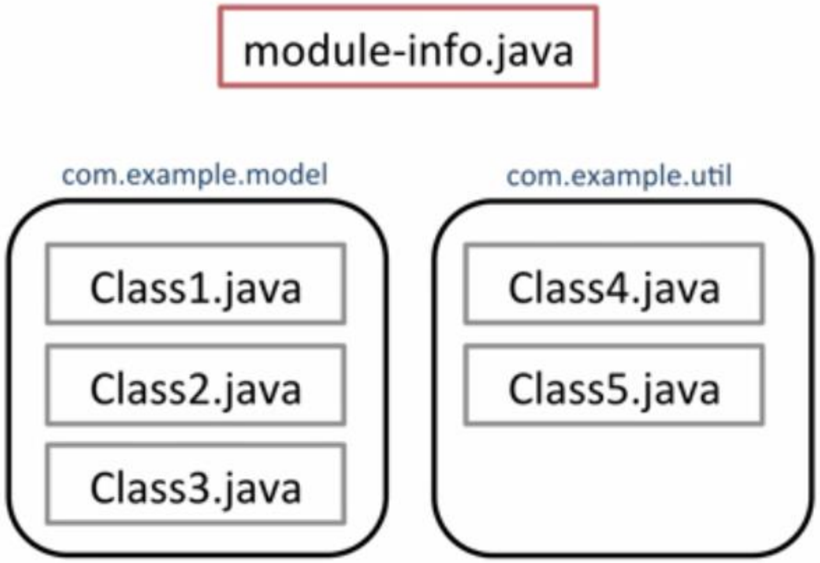
What is a Module?

Module Directives

```
module com.example.model {
```

- What other **modules** do I **require**?
- What **packages** will I **export** to other **modules**?
- What services do I **provide** to to other **modules**?
- What services do I **use** from other **modules** ?
- Will my **packages** be **open** to reflection?

```
}
```



Module Directives

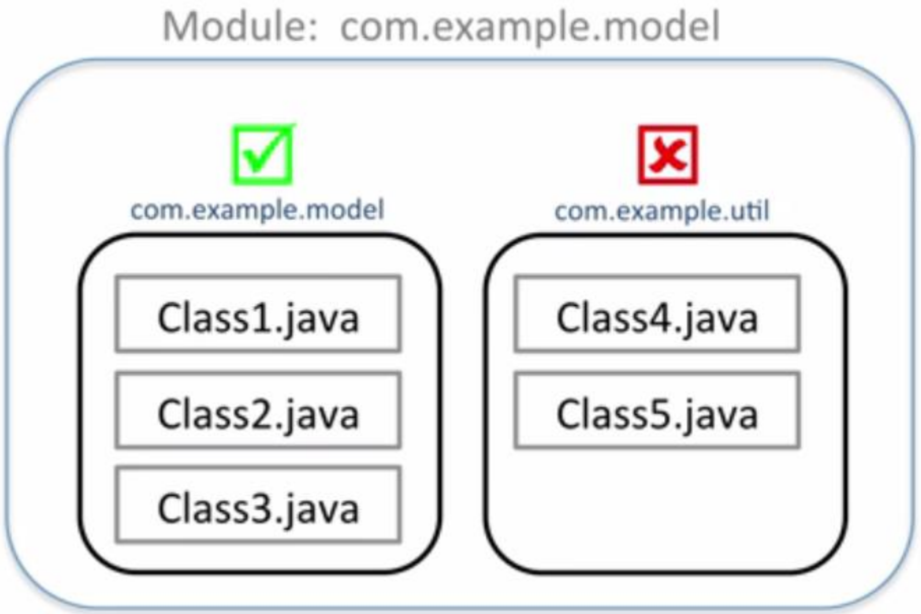
requires and exports

```
module com.example.cli {
  requires com.example.model;

}


module com.example.model {

  exports com.example.model;
}
```



Module Directives

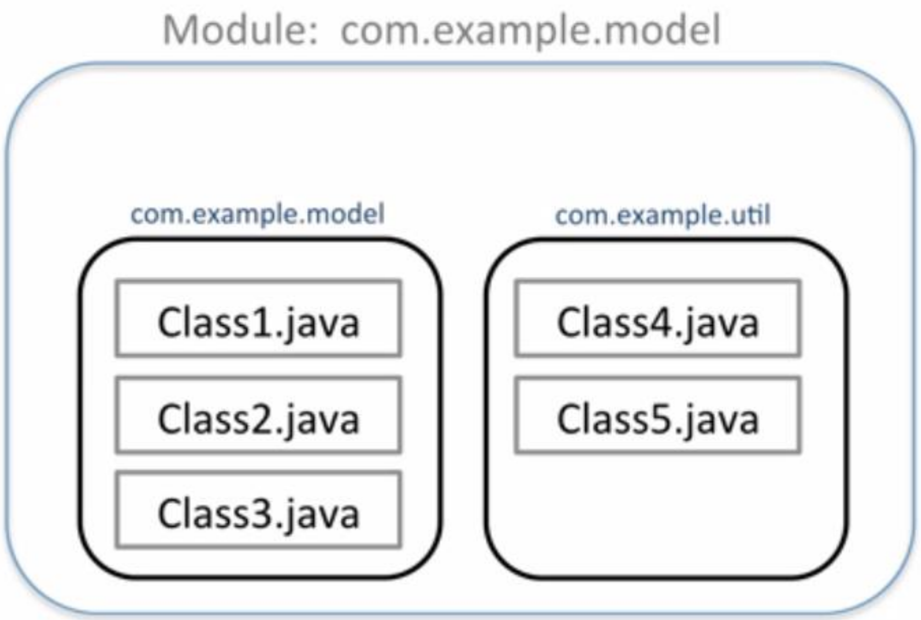
exports to

```
module com.example.cli {
 requires com.example.model;

}


module com.example.model {

 exports com.example.model
  to com.example.cli;
}
```



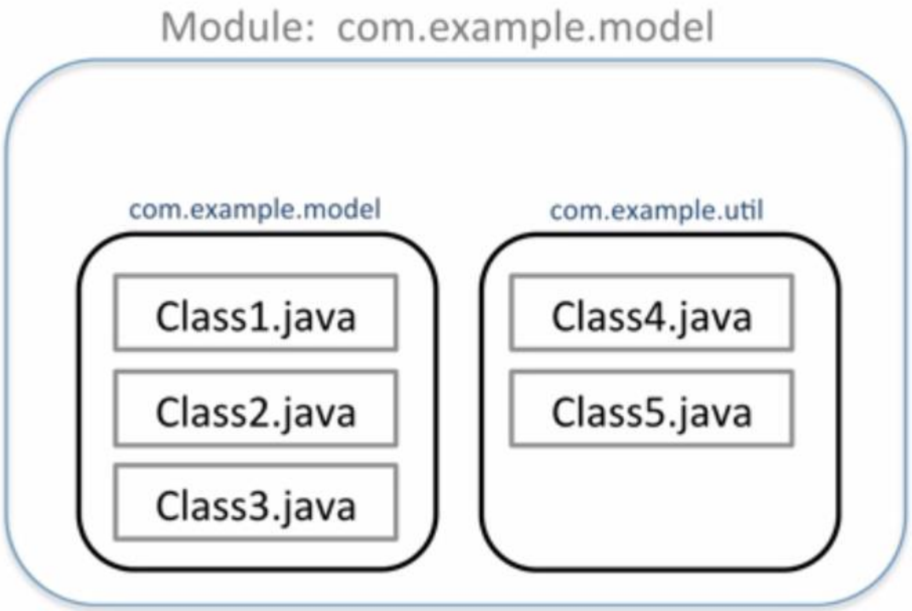
Module Directives

requires transitive (implied readability)

```
module com.example.cli {
 requires transitive
      com.example.model;

}

module com.example.model {
 requires com.example.another;
 exports com.example.model;
}
```



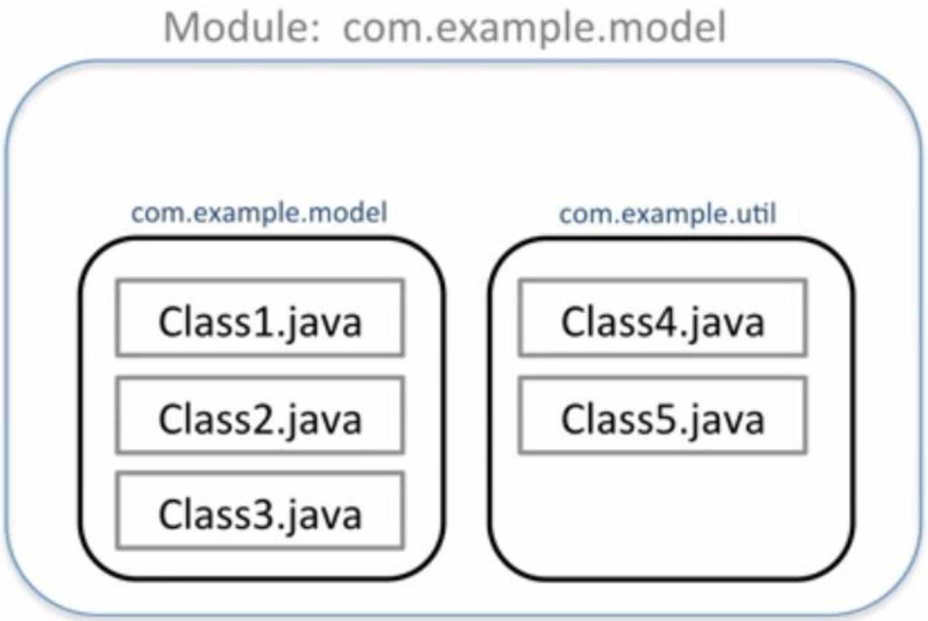
Module Directives

requires static

```
module com.example.cli {
   requires com.example.model;

}


module com.example.model {
   requires static
            com.example.another;
     exports com.example.model;
}
```

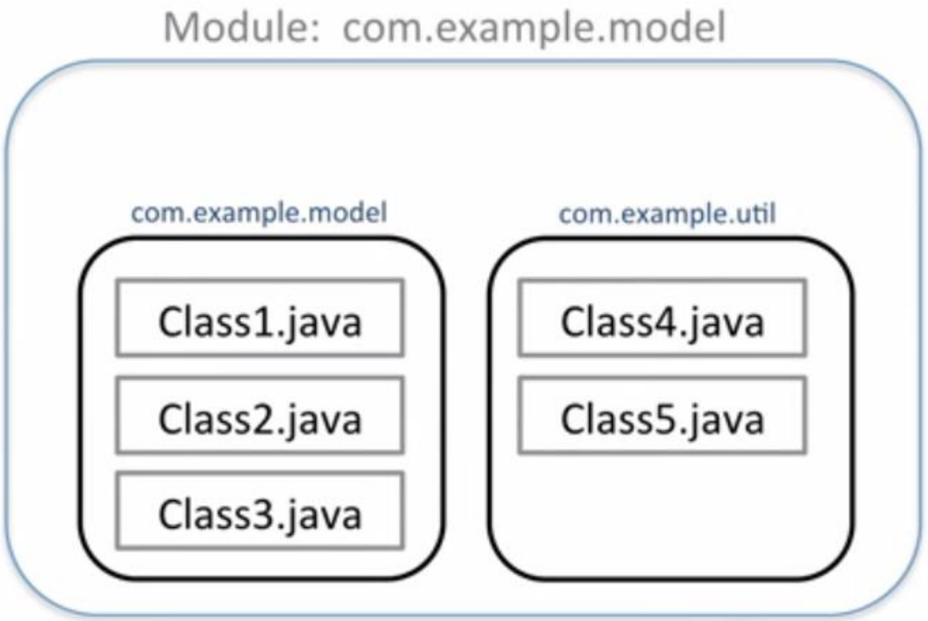


Module Directives

What about the JDK modules?

```
module com.example.cli {
   requires com.example.model;
   requires java.sql;
   requires java.base;
}

module com.example.model {
   requires java.logging;
   requires java.base;
   exports com.example.model;
}
```



The Modular JDK

- JDK 9 is completely modularized
- Java SE 9 – 94 modules

```
java --list-modules
```

- JEP 200

The Modular JDK

- JDK was monolithic
- Huge endeavor!!

Java 8

  http://openjdk.java.net/projects/jigsaw/doc/jdk-modularization.html

Java 9 SE Module Graph

  http://download.java.net/java/jdk9/docs/api/java.se.ee-graph.png

The Modular JDK

Major Changes

- Modules
- Change in directory structure
- Removal of some APIs
- New version string format

The Modular JDK

Modules – API Classification

- Classify and separate APIs
  - supported:

    `java.* javax.* jdk.* com.sun.*`
  - unsupported:

    `sun.*  sun.misc.Unsafe`
- Encapsulate most internal JDK APIs
- Remove some supported APIs

Change in Directory Structure

Lets compare the directory structure of Java 8 and Java 9

The Modular JDK

Change in Version String Format

MAJOR.MINOR.SECURITY

- MAJOR - Major version of Java (9)
- MINOR – Bug fix release (0)
- SECURITY – Critical security fixes (will not reset to zero on minor releases)

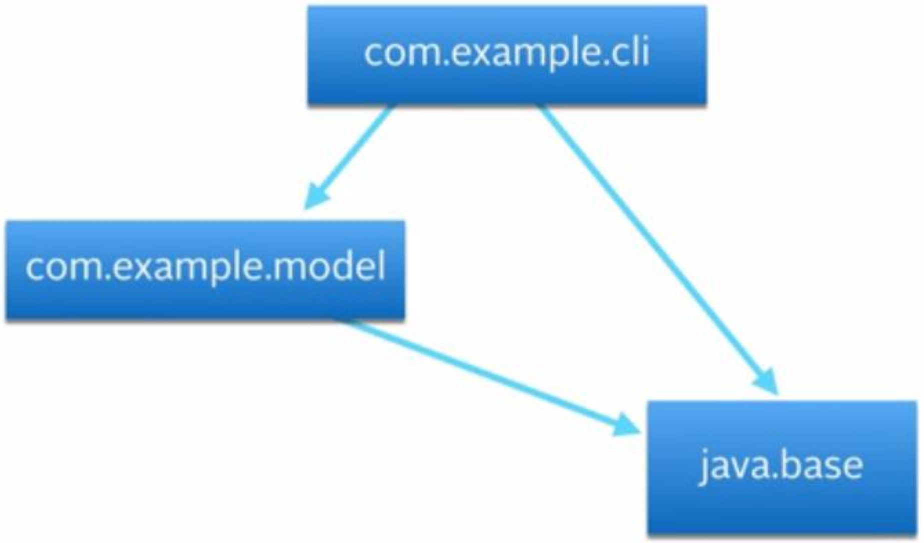
```
Runtime.Version version = Runtime.version();
version.major();      // 9
version.minor();      // 0
version.security();   // 0
version.build();      // 179
```

Module Graph

Module dependencies

```
module com.example.cli {
   requires com.example.model;
}

module com.example.model {
   exports com.example.model;
}
```

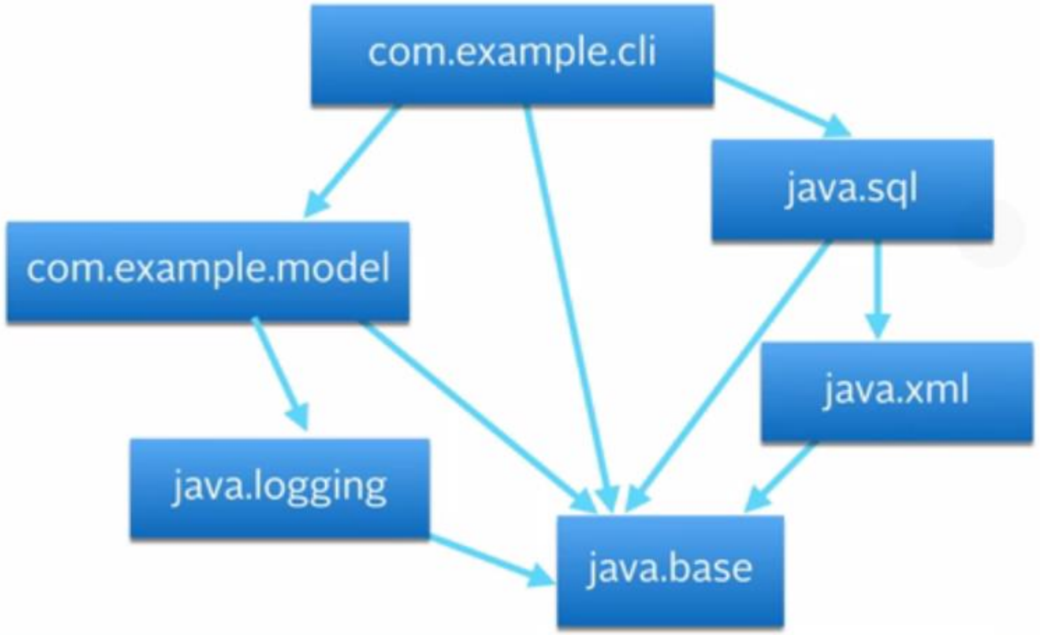


Module Graph

Module dependencies

```
module com.example.cli {
   requires com.example.model;
   requires java.sql;
}

module com.example.model {
   requires java.logging;
   exports com.example.model;
}
```


$$\begin{tikzcd}
 & \text{com.example.cli} \arrow[dl] \arrow[ddr] \arrow[r] & \text{java.sql} \arrow[d] \arrow[ddl] \\
\text{com.example.model} \arrow[d] \arrow[dr] & & \text{java.xml} \arrow[dl] \\
\text{java.logging} \arrow[r] & \text{java.base} &
\end{tikzcd}$$

Module Graph

Resolution Process

- Module path contains modules
- One module is the root module
- Other modules are resolved from the root module

Module Graph

Enforced Rules

- All modules are available
- Graph has no cycles
- No duplicate modules
- Packages in modules are unique

Compile-time
Link-time
Run-time

Module Graph

How are Modules Found?

`--module-path`

series of path names containing modular JAR, JMOD, or directory

`java --list-modules`

lists observable modules

`--add-modules`

add modules to the default set of root modules

`--module-source-path`

series of path names containing your modular source code

Allowing Reflection in Java 9

- Modules provide strong encapsulation
- Reflection allows access to private elements
- Exports allows access to public types at compile and runtime
- So, what's the problem?

Allowing Reflection in Java 9

What's the problem?

- There are many, many frameworks that access non-public elements via reflection
- Hibernate, JUnit, Spring, JavaFX,

  Eclipse, Tomcat, Struts, ...
- Very useful and used by millions of developer
- So, what's the solution?

Allowing Reflection in Java 9

The solution: `open`, `opens`, and `opens-to` module directives

`open` module directive – opens the all the module's packages for reflective access

public, protected, default, and private elements are available at runtime only

```
open module academy.learnprogramming.module {

}
```

Allowing Reflection in Java 9

The solution: `open`, `opens`, and `opens-to` module directives

`opens` module directive – opens specific module packages for reflective access

public, protected, default, and private elements are available at runtime only

```
module academy.learnprogramming.module {

  opens academy.learnprogramming.module.package1,
}
```

`jlink`: The Java Linker

- New command-line tool
- Allows creation of custom runtime images
- Make you own JRE!
- Only necessary modules are included

http://openjdk.java.net/jeps/282

`jlink`: The Java Linker

Command-line

```
jlink  --module-path <modulepath> \
   --add-modules <modules>    \
   --limit-modules <modules>   \
   --output <path>
```

`--module-path`	where to find the modules
`--add-modules`	add the modules we need
`--limit-modules`	limit the observable modules
`--output`	directory where the runtime image will be located

`jlink`: The Java Linker

Command-line

```
out
├── academy.learnprogramming.jokeapp
│   ├── academy
│   │   └── learnprogramming
│   │       └── jokeapp
│   │           └── Main.class
│   └── module-info.class
└── academy.learnprogramming.jokeserver
    ├── academy
    │   └── learnprogramming
    │       └── jokeserver
    │           ├── JokeServer.class
    │           └── internal
    │               └── Filter.class
    └── module-info.class
```

jlink: The Java Linker

Command-line

```
$JAVA_HOME/bin/jlink
 --module-path $JAVA_HOME/jmods:out \
 --add-modules academy.learnprogramming.jokeapp \
 --launcher JOKER=academy.learnprogramming.jokeapp/
academy.learnprogramming.jokeapp.Main \
 --output jre
```

jlink: The Java Linker

Custom JRE

```
jre
├── bin
├── conf
├── include
├── legal
├── lib
└── release
```

```
jre/bin
├── JOKER
├── java
└── keytool
```

35 MB

jlink: The Java Linker

Jlink Plugins

```
$JAVA_HOME/bin/jlink
 --module-path $JAVA_HOME/jmods:out \
 --add-modules academy.learnprogramming.jokeapp \
 --launcher JOKER=academy.learnprogramming.jokeapp/
academy.learnprogramming.jokeapp.Main  \
 --compress 2  \
 --no-header-files \
 --no-man-pages \
 --strip-debug  \
 --output jre
```

21 MB

`jlink`: The Java Linker

Running your application

```
jre/bin/JOKER
```

Support for private methods in interfaces

- Before JDK 8, all interface methods were public and abstract
- JDK 8 introduced default interface methods – these are methods that can have a default implementation
- JDK 9 allows private interface methods as well as private static interface methods

Support for private methods in interfaces

```
public interface PreJDK8Interface {

    void method1();
    String method2(String s);

}
```

Support for private methods in interfaces

```
public interface JDK8Interface {

    void method1();
    String method2(String s);

    default void method3 () {
        System.out.println("Default behavior setup");
    }

    default void method4 () {
        System.out.println("Default behavior setup");
    }
}
```

Support for private methods in interfaces

```
public interface JDK9Interface{

    void method1();
    String method2(String s);

    default void method3 () {
        setup();
    }

    default void method4 () {
        setup();
    }

    private void setup() {
        System.out.println("Default behavior setup");
    }
}
```

Support for private methods in interfaces

```
public interface JDK9Interface {

    void method1();
    String method2(String s);

    default void method3 () {  setup(); }

    default void method4 () { setup(); }

    private void setup() {
        System.out.println("Default behavior setup");
    }

    static void method5(String s) {
        staticSetup(s);
    }

    private static void staticSetup(String s) {
    }
}
```

Convenience Factory Methods for Collections – JEP 269

- Create unmodifiable collections (List, Set, Map)
- The current way is not convenient
- `List<Integer> list = #[10, 20 ,30];` ✖

http://openjdk.java.net/jeps/269

Creating an unmodifiable list Pre JDK 9

```
List<String> trees = new ArrayList<>();

trees.add(″Oak");
trees.add(″Cedar");
trees.add(″Pine");

trees = Collections.unmodifiableList(trees);
```

Creating an unmodifiable list JDK 9

```
List<String> trees = List.of("Oak", "Cedar", "Pine");
```

Creating an unmodifiable Set JDK 9

```
Set<String> set = Set.of("Oak", "Cedar", "Pine");
```

Creating an unmodifiable Map JDK 9

```
Map<String, String> states = Map.of(
        "FL", "Florida",
        "CA", "California",
        "GA", "Georgia"
);
```

For more than 10 items in Map use Map.ofEntries

Exceptions

NullPointerException

You provide null element

```
List<String> list = List.of("Oak", null);
```

Exceptions

`IllegalArgumentException`

You provide a duplicate element to a Set or Map

```
Set<Integer> set = Set(1, 2, 2);
```

Exceptions

UnsupportedOperationException

You try to modify the immutable collection object

```
List<String> trees = List.of("Oak", "Teak");
trees.add("Cherry");
```

Thank You…