Tweak Your Modules!

Java Platform Module System

Stephan Herrmann
JDT embraces Java™ 9 / Part II

We've all migrated to Java 11

From here
adapting new Java versions
is a breeze

Compatibility for all!

... and live happily ever after ...
JDT embraces Java™ 9 / Part II

We've all migrated to Java 11

```
6 import javax.xml.parsers.ParserConfigurationException;
7 import javax.xml.parsers.SAXParser;
8 import javax.xml.parsers.SAXParserFactory;
9
10 import org.xml.sax;
11 import org.xml.sax;
12```

The package javax.xml.parsers is accessible from more than one module: `<unnamed>`, java.xml

Compatibility for all!

... and live happily ever after ...
Some Modules are More Equal than Others

Named modules

- System modules
  - part of JRE, implicitly available (not all!)
- Regular modules
  - user defined
  - have module-info
- Automatic modules
  - user defined
  - no module-info

Unnamed module

- All the rest – “legacy”

Wiring à la JPMS:

Module 1

```
package a

C0    C1    C2

package a.b

C1    C3
```

Module 2

```
package a.c

C1    C2

package a.b

C1    C4
```
Some Modules are More Equal than Others

Named modules

- System modules
  - part of JRE, implicitly available (not all!)

- Regular modules
  - user defined
  - have module-info

- Automatic modules
  - user defined
  - no module-info

Unnamed module

- All the rest – “legacy”

All Java9+ programs use modules, perhaps unknowingly.
Demo java.xml: Conflicting Modules
What if your CORBA application should run on Java 9 and Java 11?

JEP 320
Remove the Java EE and CORBA Modules
"In Java SE 9, the Java SE modules that contain **Java EE** and **CORBA** technologies are annotated as *deprecated for removal*, indicating the intent to remove them in a future release:

- java.xml.ws (JAX-WS, [...] SAAJ and Web Services Metadata)
- java.xml.bind (JAXB)
- java.activation (JAF)
- java.xml.ws.annotation (Common Annotations)
- java.corba (CORBA)
- java.transaction (JTA)"

"Related modules in Java SE 9 are also deprecated for removal:

- java.se.ee (Aggregator module for the six modules above)
- jdk.xml.ws (Tools for JAX-WS)
- jdk.xml.bind (Tools for JAXB)"

"This JEP will remove the nine modules listed above" – Fix Version/s: 11
Options for using JAXB

- **Java 9, named module**
  - Just require `java.xml.bind`
  - Put `jaxb-api.jar` on the modulepath
    1) Remove system module
    2) Add `jaxb-api.jar` to modulepath

- **Java 9, unnamed module**
  - Put `jaxb-api.jar` on the classpath
  - Add system module `java.xml.bind`

- **Java 11+**
  - Put `jaxb-api.jar` on the classpath / modulepath
  - There is no option 2

---

**java.xml.bind**
- exists as a system module
- deprecated for removal
- not visible by default to unnamed module

**java.xml.bind**
- no longer a system module
Options for using JAXB

**JEP 320:**
“Since deprecating modules for removal merely causes compile-time warnings, JDK 9 took a more robust step to prepare developers for the actual removal of these modules in a future release:

The modules are not resolved in JDK 9 when code on the class path is compiled or run. This allows developers on JDK 9 to deploy standalone versions of the Java EE and CORBA technologies on the class path, just like on JDK 8.”

“Alternatively, developers on JDK 9 can use the --add-modules flag on the command line to resolve the modules in the JDK runtime image.”

---

**java.xml.bind**
- exists as a system module
- deprecated for removal
- not visible by default to unnamed module

**java.xml.bind**
- no longer a system module
Defining a Modular Application

Language: module-info.java

- module
- Directives: requires, exports, opens, provides, uses
  - to, with
- Modifiers: open, transitive

Command line options

- --add-modules
- ...
  
Stephan Herrmann @EclipseCon Europe 2019 – published under the EPL
Demo java.xml.bind: Module Goes Missing
Eclipse UI ↔ Command Line

- **Eclipse:** Java Build Path > Module Dependencies > All Modules
  - Add System Module → additionally adds required modules
  - Remove → additionally removes requiring modules

- **Command Line**
  - `--add-modules` → need to **completely** enumerate all additional modules
  - `--limit-modules` → supports **minimal** form – will keep the transitive closure of listed modules

- To observe the correspondence: **Show JPMS Options** ...
The Case of Tests

- Modular project
- Main & test sources separated
  - test sources marked (Thanks to Till Brychcy)
- JUnit on the classpath
- Eclipse implicitly adds these tweaks for running tests:
  - --add-opens my.mod/test.pack=ALL-UNNAMED
  - --add-reads my.mod=ALL-UNNAMED
  - --add-modules=ALL-MODULE-PATH
  - --patch-module my.mod=/path/to/MyProject/bin-test
Undeclared Dependencies

- **JUnit framework** can access **SUT / Test**
- **add-reads**
- **add-opens sut**
- **reflective access to package of Test**
- **patch-module**
- **can share packages & access internals of SUT**
- **combined module SUT** can access **JUnit**
- **JUnit framework** can access **SUT / Test**
- **ensure SUT is in the module graph when launching JUnit**
- **add-modules**

Stephan Herrmann @EclipseCon Europe 2019 – published under the EPL
Demo:
Modular Tests
Knobs & Dials

› module-info.java
   › intrinsic properties

› Java Build Path
   › determine set of **observable** modules
   › superimpose more **edges** onto the **module graph**
   › just during building of this project

› Launch Configuration
   › initialized from Java Build Path
   › superimpose more **edges** onto the **module graph**
   › determine the set of **root** modules

› Special modules names
   › add-modules: ALL-SYSTEM, ALL-DEFAULT, ALL-MODULE-PATH
   › add-reads, add-exports, add-opens: ALL-UNNAMED