Java 9 Modules

What's the Hype All About?

By: Brent W. Wilkins

Java 9

- Was released in September, 2017

Java 9

- Was released in September, 2017
- Has already hit its EOL due to Java 10 being released in March 2018

Java 9

- Was released in September, 2017
- Has already hit its EOL due to Java 10 being released in March 2018
- Gave use several new features

Java 9

- Was released in September, 2017
- Has already hit its EOL due to Java 10 being released in March 2018
- Gave use several new features
  - jshell – REPL (Read, Evaluate, Print, Loop)

Java 9

- Was released in September, 2017
- Has already hit its EOL due to Java 10 being released in March 2018
- Gave use several new features
  - jshell – REPL (Read, Evaluate, Print, Loop)
  - Collection factory methods: of() to quickly initialize lists, sets and maps.

Java 9

- Was released in September, 2017
- Has already hit its EOL due to Java 10 being released in March 2018
- Gave use several new features
  - jshell – REPL (Read, Evaluate, Print, Loop)
  - Collection factory methods: of() to quickly initialize lists, sets and maps.
  - Stream API improvements – dropWhile and TakeWhile

Java 9

- Was released in September, 2017
- Has already hit its EOL due to Java 10 being released in March 2018
- Gave use several new features
  - jshell – REPL (Read, Evaluate, Print, Loop)
  - Collection factory methods: of() to quickly initialize lists, sets and maps.
  - Stream API improvements – dropWhile and TakeWhile
  - Private interface methods

Java 9

- Was released in September, 2017
- Has already hit its EOL due to Java 10 being released in March 2018
- Gave use several new features
  - jshell – REPL (Read, Evaluate, Print, Loop)
  - Collection factory methods: of() to quickly initialize lists, sets and maps.
  - Stream API improvements – dropWhile and TakeWhile
  - Private interface methods
  - And the Holy Grail of features: The **Project Module Platform System**

Java Platform Module System (JPMS)

aka: Project Jigsaw

- Initiated in August of 2008 and was slated for Java 7

Java Platform Module System (JPMS)

aka: Project Jigsaw

- Initiated in August of 2008 and was slated for Java 7
- Pushed to Java 8 and eventually to Java 9

Java Platform Module System (JPMS)

Goals

- Scale Java SE to small devices

Java Platform Module System (JPMS)

Goals

- Scale Java SE to small devices
  - Java 8 – rt.jar is ~61MB in size

Java Platform Module System (JPMS)

Goals

- Scale Java SE to small devices
  - Java 8 – rt.jar is ~61MB in size
  - rt.jar now broken into 95+ modules

Java Platform Module System (JPMS)

Goals

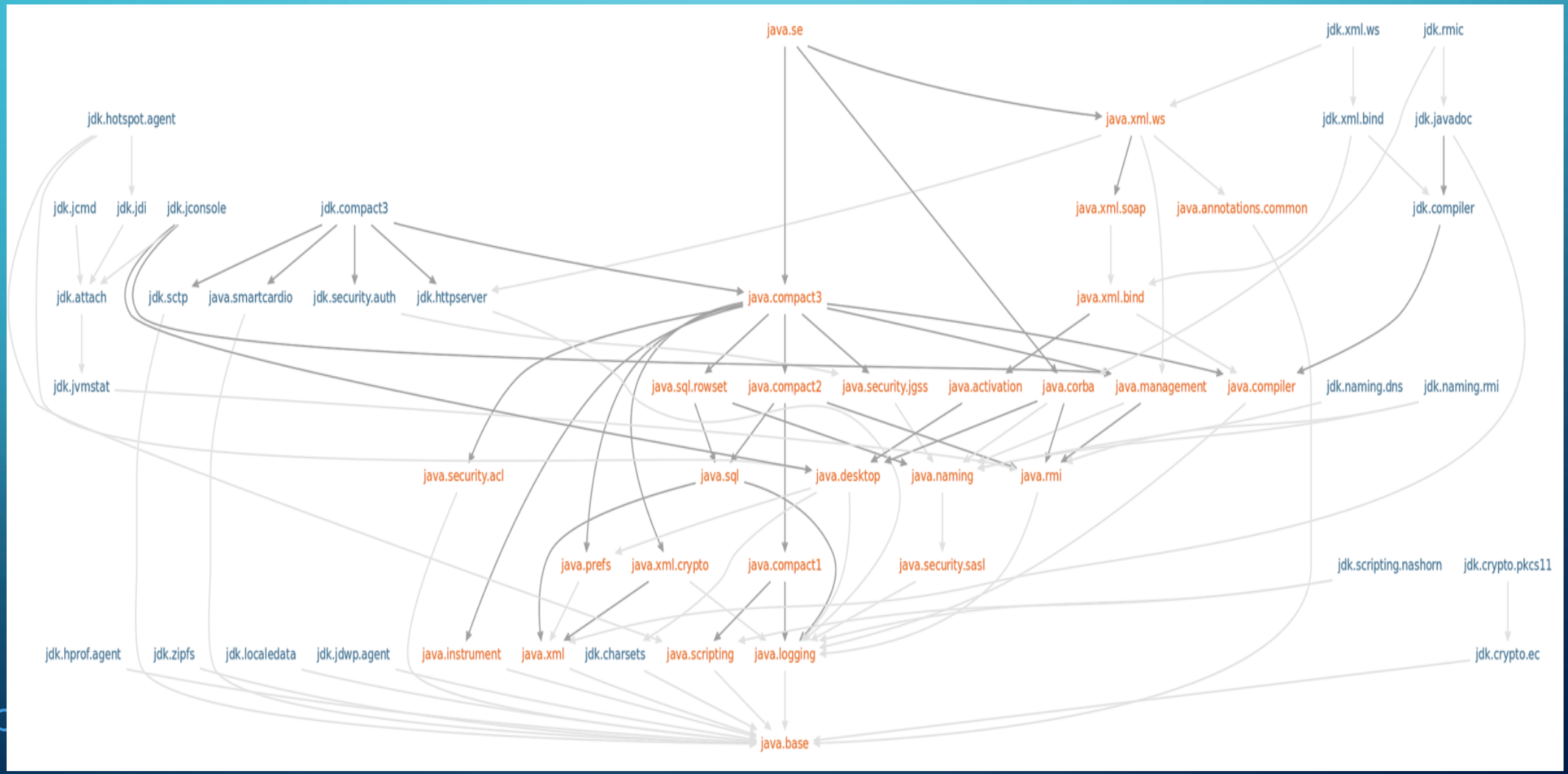
- Scale Java SE to small devices
  - Java 8 – rt.jar is ~61MB in size
  - rt.jar now broken into 95+ modules

```
java.activation
java.base
java.compiler
java.corba
java.datatransfer
java.desktop
java.instrument
java.jnlp
java.logging
```

```
java.management
java.management.rmi
java.naming
java.prefs
java.rmi
java.scripting
java.se
java.se.ee
java.security.jgss
```

```
java.security.sasl
java.smartcardio
java.sql
java.sql.rowset
java.transaction
java.xml
java.xml.bind
java.xml.crypto
java.xml.ws
```

Java Platform Module System (JPMS)



Java Platform Module System (JPMS)

Goals

- Scale Java SE to small devices
- Improve security and maintainability

Java Platform Module System (JPMS)

Goals

- Scale Java SE to small devices
- Improve security and maintainability
- Better application performance

Java Platform Module System (JPMS)

Goals

- Scale Java SE to small devices
- Improve security and maintainability
- Better application performance
- Easier to construct & maintain libraries/applications

Java Platform Module System (JPMS)

What is a module?

Java Platform Module System (JPMS)

What is a module?

- A **class** is a container of fields and methods

Java Platform Module System (JPMS)

What is a module?

- A **class** is a container of fields and methods
- A **package** is a container of classes and interfaces

Java Platform Module System (JPMS)

What is a module?

- A **class** is a container of fields and methods
- A **package** is a container of classes and interfaces
- A **module** is a container of packages

Java Platform Module System (JPMS)

What is a module?

- A **class** is a container of fields and methods
- A **package** is a container of classes and interfaces
- A **module** is a container of packages
- module-info.java – Is a java file that is a first class citizen of the java world. It compiles to a .class file and its existence in a jar file makes that jar a module.

Java Platform Module System (JPMS)

What is the purpose of module-info.java?

Java Platform Module System (JPMS)

What is the purpose of module-info.java?

- Define the dependencies that this module requires

Java Platform Module System (JPMS)

What is the purpose of module-info.java?

- Define the dependencies that this module requires
- Define what packages from this module will be exposed

Java Platform Module System (JPMS)

What is the purpose of module-info.java?

- Define the dependencies that this module requires
- Define what packages from this module will be exposed
- **`java.base`** is required by default and does not have to be specifically named.

Java Platform Module System (JPMS)

What is the purpose of module-info.java?

- Define the dependencies that this module requires
- Define what packages from this module will be exposed
- **`java.base`** is required by default and does not have to be specifically named.
- Must begin with the keyword **`module`**.

Java Platform Module System (JPMS)

What is the purpose of module-info.java?

- Define the dependencies that this module requires
- Define what packages from this module will be exposed
- **`java.base`** is required by default and does not have to be specifically named.
- Must begin with the keyword **`module`**.
- Module name must be unique.

Java Platform Module System (JPMS)

Requires – If the packages in a module have dependencies then they must be declared using the **`requires`** keyword.

Java Platform Module System (JPMS)

Requires – If the packages in a module have dependencies then they must be declared using the **`requires`** keyword.

- Must be another module. If requiring a jar that is not modularized then use the jar name minus the version number.

Java Platform Module System (JPMS)

Requires – If the packages in a module have dependencies then they must be declared using the **`requires`** keyword.

- Must be another module. If requiring a jar that is not modularized then use the jar name minus the version number.
- Versions are still handled by Gradle, Maven, or whatever build system you use.

Java Platform Module System (JPMS)

Requires – If the packages in a module have dependencies then they must be declared using the **`requires`** keyword.

- **`transitive`** – Allows modules accessing this module to also access the required module without having to expressly require it in its own module definition.

Java Platform Module System (JPMS)

Requires – If the packages in a module have dependencies then they must be declared using the **`requires`** keyword.

- **`transitive`** – Allows modules accessing this module to also access the required module without having to expressly require it in its own module definition.
- **`static`** – The module is required at compile time, but is optional at runtime. Ex. Junit, Mockito, etc.

Java Platform Module System (JPMS)

Exports – To expose packages that you want to make public, you use the **`exports`** keyword.

Java Platform Module System (JPMS)

Exports – To expose packages that you want to make public, you use the **`exports`** keyword.

- Only packages can be exported.

Java Platform Module System (JPMS)

Exports – To expose packages that you want to make public, you use the **`exports`** keyword.

- Only packages can be exported.
- All public and protected classes in the package will be exposed.

Java Platform Module System (JPMS)

Exports – To expose packages that you want to make public, you use the **`exports`** keyword.

- Only packages can be exported.
- All public and protected classes in the package will be exposed.
- Sub packages are not exposed and have to be exported separately.

Java Platform Module System (JPMS)

Exports – To expose packages that you want to make public, you use the **`exports`** keyword.

- Only packages can be exported.
- All public and protected classes in the package will be exposed.
- Sub packages are not exposed and have to be exported separately.
- Package names being exported must have a unique name across all modules. ie. No 2 modules can export the same package name.

Java Platform Module System (JPMS)

Open(s) – Allows classes to be discovered via reflection.

Java Platform Module System (JPMS)

Open(s) – Allows classes to be accessed via reflection.

- Before Java 9 you could not control who accessed the private methods in your jar file due to reflection.

Java Platform Module System (JPMS)

Open(s) – Allows classes to be accessed via reflection.

- Before Java 9 you could not control who accessed the private methods in your jar file due to reflection.
- **`open module <module-name>`** - Opens the entire module to reflection.

Java Platform Module System (JPMS)

Open(s) – Allows classes to be accessed via reflection.

- Before Java 9 you could not control who accessed the private methods in your jar file due to reflection.
- **`open module <module-name>`** - Opens the entire module to reflection.
- **`opens <package-name>`** - Only opens that package to reflection.

Java Platform Module System (JPMS)

javac – New parameters have been added for modules.

Java Platform Module System (JPMS)

javac – New parameters have been added for modules.

- `--module-path` – Specify where to find application modules.

Java Platform Module System (JPMS)

javac – New parameters have been added for modules.

- **`--module-path`** – Specify where to find application modules.
- **`--module-source-path`** – Specify where to find input source files for the module.

Java Platform Module System (JPMS)

java – New parameters have been added for modules.

Java Platform Module System (JPMS)

java – New parameters have been added for modules.

- `--module-path` – Like the classpath, the module-path specifies which modules to make available at run time.

Java Platform Module System (JPMS)

java – New parameters have been added for modules.

- **`--module-path`** – Like the classpath, the module-path specifies which modules to make available at run time.
- **`--module`** – Like the **`-jar`** parameter to execute a jar file but defines a module to execute.

Java Platform Module System (JPMS)

java – New parameters have been added for modules.

- **`--module-path`** – Like the classpath, the module-path specifies which modules to make available at run time.
- **`--module`** – Like the **`-jar`** parameter to execute a jar file but defines a module to execute.
- **`--describe-module`** – Describes a module by showing what the module requires, uses and exports.

Java Platform Module System (JPMS)

java – New parameters have been added for modules.

- **`--module-path`** – Like the classpath, the module-path specifies which modules to make available at run time.
- **`--module`** – Like the **`-jar`** parameter to execute a jar file but defines a module to execute.
- **`--describe-module`** – Describes a module by showing what the module requires, uses and exports.
- **`--list-modules`** – Lists the observable modules and exists.

Java Platform Module System (JPMS)

jlink – New command line tool to create your own JRE that includes only the modules you need for your project.

Java Platform Module System (JPMS)

jlink – New command line tool to create your own JRE that includes only the modules you need for your project.

- `--module-path` – Like the classpath, the module-path specifies path to the actual module files.

Java Platform Module System (JPMS)

jlink – New command line tool to create your own JRE that includes only the modules you need for your project.

- **`--module-path`** – Like the classpath, the module-path specifies path to the actual module files.
- **`--add-modules`** – The list of modules to include in the new JRE.

Java Platform Module System (JPMS)

jlink – New command line tool to create your own JRE that includes only the modules you need for your project.

- **`--module-path`** – Like the classpath, the module-path specifies path to the actual module files.
- **`--add-modules`** – The list of modules to include in the new JRE.
- **`--output`** – The folder to put the new JRE files into.

Java Platform Module System (JPMS)

DEMO

Java Platform Module System (JPMS)

Resources

- The State of the Module System – Mark Reinhold

http://openjdk.java.net/projects/jigsaw/spec/sotms/

- The Java 9 Modules Cheat Sheet

https://zeroturnaround.com/rebellabs/java-9-modules-cheat-sheet/

- Dzone – Java 9 Modules (Part 1) – Introduction

https://dzone.com/articles/java-9-modules-introduction-part-1