Project Jigsaw: Under The Hood

Alex Buckley
Java Platform Group, Oracle
August 2016
JDK 9 At A Glance

• Module System
• Modular JDK
• Language enhancements
• Library enhancements
• Tool enhancements
JDK 9 At A Glance

- Module System
- Modular JDK
- Language enhancements
- Library enhancements
- Tool enhancements

Project Jigsaw

The primary goals of this Project are to:

- Make the Java SE Platform, and the JDK, more easily scalable down to small computing devices;
- Improve the security and maintainability of Java SE Platform Implementations in general, and the JDK in particular;
- Enable improved application performance; and
- Make it easier for developers to construct and maintain libraries and large applications, for both the Java SE and EE Platforms.

To achieve these goals we propose to design and implement a standard module system for the Java SE Platform and to apply that system to the Platform itself, and to the JDK. The module system should be powerful enough to modularize the JDK and other large legacy code bases, yet still be approachable by all developers.
Project Jigsaw: Under The Hood

Part I: Accessibility and Readability
Part II: Different Kinds of Modules
Part III: Loaders and Layers
Part IV: The Road Ahead
Part I: Accessibility and Readability
Accessibility (JDK 1 – JDK 8)

• public
• protected
• <package>
• private
Accessibility (JDK 9 –)

- public to everyone
- public but only to specific modules
- public only within a module
- protected
- <package>
- private
‘public’ no longer means “accessible”.
The result:
Accessibility and Module Declarations

// src/java.sql/module-info.java
module java.sql {
    exports java.sql;
    exports javax.sql;
    exports javax.transaction.xa;
}

Accessibility and Class Loaders

Loader 1

class P.C

delegates

accesses

Loader 2

class Q.D

Loader 1

class P.C

no delegation

Loader 2

class Q.D

no access
Accessibility and Class Loaders

Loader 1

class P.C

Loader 1

no delegation

class P.C

Loader 2

accesses

class Q.D

Loader 2

delegates

class Q.D

no access
One Loader, Many Modules

Loader 1

Module X

class P.C

accesses?

class Q.D

Module Y
The Role of Readability

Module X

class P.C

reads

Loader 1

Module Y

class Q.D

accesses
The Role of Readability

module X {
    requires Y;
}

module Y {
    exports Q;
}
Readability in the JDK Module Graph

```java
module java.sql {
    requires java.logging;
    exports java.sql;
}

module java.logging {
    exports java.util.logging;
}

package java.sql;
import java.util.logging.Logger;
public class DriverManager {
    new Logger() { .. }
}

package java.util.logging;
public class Logger {
    ...
}
```
Readability in the JDK Module Graph

```java
module java.sql {
    requires java.logging;
    exports java.sql;
}

module java.logging {
    exports java.util.logging;
}

package java.sql;
import java.util.logging.Logger;
public interface Driver {
    Logger getParentLogger();
}

package java.util.logging;
public class Logger {
    ...
}
```
Readability in the JDK Module Graph

```java
module myapp {
    requires java.sql;
    requires java.logging;
}

module java.sql {
    requires java.logging;
    exports java.sql;
}

module java.logging {
    exports java.util.logging;
}
```
Readability in the JDK Module Graph

module myapp {
    requires java.sql;

    requires java.logging; 😊
}

module java.sql {
    requires public java.logging;
    exports java.sql;
}

module java.logging {
    exports java.util.logging;
}

Readability in the JDK Module Graph

module myapp {
    requires java.sql;
    requires java.logging;
}

module java.sql {
    requires public java.logging;
    exports java.sql;
}

module java.logging {
    requires public logextras;
    exports java.util.logging;
}

module logextras {
    ...
}
Direct and Implied Readability

• X reads Y if:
  – X requires Y
  or
  – X reads Q, and Q requires public Y
void doSomething(Class<?> c) {
    Method[] ms = c.getDeclaredMethods();
    ms[0].invoke(...);
}
Core Reflection

`setAccessible(true)`
Summary of Part I: Accessibility and Readability

• Accessibility used to be a simple check for ‘public’ or “same package”.
• In JDK 9, accessibility strongly encapsulates module internals.
• Accessibility relies on readability, which can be direct or implied.
• Accessibility is enforced by the compiler, VM, and Core Reflection.
Part II: Different Kinds of Modules
Named Modules

Named modules:
- java.base
- java.sql
- jdk.compiler
- jdk.javadoc

classpath:
- guava.jar
- junit.jar
- glassfish.jar
- hibernate.jar
The Unnamed Module

Named modules

- java.base
- java.sql
- jdk.compiler
- jdk.javadoc

Unnamed module

- guava.jar
- junit.jar
- glassfish.jar
- hibernate.jar
The Unnamed Module

Named modules

java.base  java.sql
jdk.compiler  jdk.javadoc

Unnamed module

guava.jar  junit.jar
glassfish.jar  hibernate.jar
Automatic Modules

Named modules
- java.base
- java.sql
- jdk.compiler
- jdk.javadoc
- guava

Unnamed module
- junit.jar
- glassfish.jar
- hibernate.jar
Automatic Modules

Named modules:
- java.base
- java.sql
- guava
- jdk.compiler
- jdk.javadoc

Unnamed module:
- junit.jar
- glassfish.jar
- hibernate.jar
Automatic Modules

Named modules
- java.base
- java.sql
- jdk.compiler
- jdk.javadoc
- myapp

Unnamed module
- guava
- junit.jar
- glassfish.jar
- hibernate.jar

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.
Summary of Part II: Different Kinds of Modules

• Explicit named modules (java.sql)
• Automatic named modules (guava)
• Unnamed module (a.k.a. classpath)
• Lots of readability “for free” to help with migration.
Part III: Loaders and Layers
Class loading doesn’t change.
Class Loading in JDK 9

- Bootstrap loader
- Platform loader
- Application loader

Java Virtual Machine
Class Loading in JDK 9

Java Virtual Machine

Bootstrap loader
- java.base
- java.logging

Platform loader
- java.sql
- java.corba

Application loader
- jdk.compiler
- guava
- junit.jar
- glassfish.jar
Modular Class Loading on JDK 9

Java Virtual Machine

Bootstrap loader
Platform loader
Application loader

Loader A
A.jar

Loader B
B.jar

Loader C
C.jar

Module System
Modular Class Loading in JDK 9

Java Platform Module System

Java Virtual Machine
Layers

Java Platform Module System

Java Virtual Machine
Layer Creation

(1) String moduleName -> {
    switch (moduleName) {
        case "java.base":
            return BOOTSTRAP_LDR;
        case "java.logging":
            return APP_LDR;
        default:
            return APP_LDR;
    }
}

(2) String moduleName -> {
    switch (moduleName) {
        case "java.base":
            return BOOTSTRAP_LDR;
        case "java.logging":
            return BOOTSTRAP_LDR;
        default:
            return APP_LDR;
    }
}
Layers and the VM

Boot layer

Bootstrap loader
- java.base
- java.logging

Platform loader
- java.sql
- java.corba

Application loader
- jdk.compiler
- guava

Java Platform Module System

Java Virtual Machine
- java.base
- java.logging
- java.sql
- java.corba
- jdk.compiler
- guava
Well-formed Graphs
Well-formed Graphs

• A module must read only one module that exports a package called P.
Well-formed Maps

• Different modules with the same package must map to different loaders.

```java
String moduleName -> {
    switch (moduleName) {
        case "java.base":
        case "java.logging":
            return BOOTSTRAP_LDR;
        default:
            return APP_LDR;
    }
}
```
Well-formed Maps

• Loader delegation must respect module readability.
Layers of Layers

- Boot layer
- Bootstrap loader
- Platform loader
- Application loader

Java Platform Module System

Java Virtual Machine

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.
Layers and Versions

Boot layer

Bootstrap loader
- java.base
- java.logging

Platform loader
- java.sql
- java.corba

Application loader
- myapp
- mylib

Hadoop layer

Loader 16
- hadoop
- guava@11

Loader 17
- jackson@1

Loader 23
- closure-compiler
- guava@18

Loader 24
- jackson@2

JavaScript layer

Java Platform Module System

Java Virtual Machine
Summary of Part III: Loaders and Layers

• Modules do a better job of encapsulation than class loaders, but class loaders are still necessary.

• Layers control the relationship between modules and class loaders.

• Assuming class loaders respect the module graph, the system is safe by construction – no cycles or split packages.
Summary of Summaries

• Strong encapsulation of modules by the compiler, VM, Core Reflection.
• Unnamed and automatic modules help with migration.
• The system is safe by construction – no cycles or split packages.
Part IV: The Road Ahead
Incompatible Changes in JDK 9

- java.util.{logging,jar}, java.awt [.dnd].peer
- org.omg.CORBA, javax.rmi, javax.xml.{bind,ws}, javax.annotation
- java[.vm][.specification].version

- sun.misc
- sun.net.www, sun.security.x509, com.sun.org.apache.xerces.internal.jaxp
- rt.jar, tools.jar, -Xbootclasspath/p
Incompatible Changes in JDK 9

• java.util.{logging,jar}, java.awt.[.dnd].peer
• org.omg.CORBA, javax.rmi, javax.xml.{bind,ws}, javax.annotation
• java[.vm][.specification].version

• sun.misc
• sun.net.www, sun.security.x509, com.sun.org.apache.xerces.internal.jaxp
• rt.jar, tools.jar, -Xbootclasspath/p
Advice for Frameworks

- Modularize nothing
- Modularize the framework
- Accept modular input
- Modularize the framework and the input
Preparing for JDK 9

• JDK 8: Run `jdeps –jdkinternals MyApp.jar`  
• JEP 261: Module System  
• JEP 260: Encapsulate Most Internal APIs  
• JEP 223: New Version String Scheme  
• JEP 220: Modular Run-Time Images  
• JEP 200: The Modular JDK
Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.