JDK 9 Deep Dive

Simon Ritter
Deputy CTO, Azul Systems

@speakjava

JDK 9: Big And Small Changes

Process API Updates
HTTP 2 Client
Improve Contended Locking
Unified JVM Logging
Compiler Control
Variable Handles
Segmented Code Cache
Smart Java Compilation, Phase T
The Modular JDK
Modular Source Code
Elide Deprecation Warnings on I tatement
Resolve Lint and Doclint Warnin
Milling Project Coin
Remove GC Combinations Depre JDK 8
Tiered Attribution for javac
Process Import Statements Corr
Annotations Pipeline 2.0
Datagram Transport Layer Secur S)
Modular Run-Time Imag
Simplified Doclet API
jshell: The Java Shell (Rea nt Loop)
New Version-String Scheme
HTML5 Javadoc
Javadoc Search
UTF-8 Property Files
Unicode 7.0
Add More Diagnostic Commands
Create PKCS12 Keystores by Default
Remove Launch-Time JRE Version Selection

Improve Secure Application Performance
Generate Run-Time Compiler Tests Automatically
Test Class-File Attributes Generated by javac
Parser API for Nashorn
Linux/AArch64 Port
Multi-Release JAR Files
Remove the JVM TI hprof Agent
ve the jhat Tool
JVM Compil ace
ion-Layer Negotia nsion
Valid Comman ag Ar
Levera nstructi GHA KSA
Compil er Platt
Make G efault G
OCSP S or TLS
Store I Strings rch
Multi on Imag
Use ale Data lt
aFX UI Con SS APIs fo rization
t Strings
Merge Selected Xerces Fixes into JAX
BeanInfo Annotations
Update JavaFX/Media to Newer Version of GStreamer
HarfBuzz Font-Layout Engine
Stack-Walking API
Encapsulate Most Internal APIs
Module System
TIFF Image I/O
HiDPI Graphics on Windows and Linux

Platform Logging API and Service
Marlin Graphics Renderer
More Concurrency Updates
Unicode 8.0
XML Catalogs
Convenience Factory Methods for Collections
Reserved Stack Areas for Critical Sections
Unified
Platf eatures
DRB Secure mplementations
En Method Ha
Mo va Applicat aging
Dy king of Lar efined Object Models
Enh
Addit s Objects in G1
Improve Test Failure T hooting
Indify String Concaten
HotSpot C++ Unit-Te ework
jlink The Java Link
Enal
New H System
Spin-Wait Hints
SHA-3 Hash Algorithms
Disable SHA-1 Certificates
Deprecate the Applet API
Filter Incoming Serialization Data
Implement Selected ECMAScript 6 Features in Nashorn
Linux/s390x Port

JDK 9

Agenda

- Java Platform Module System
- Developer features
- Other things
- Migrating applications to JDK 9
  - Real world example
- Summary

JPMS: API Structural Changes

API Classification

- Supported, intended for public use
  - JCP specified: java.*, javax.*
  - JDK specific: some com.sun.*, some jdk.*
- Unsupported, not intended for public use
  - Mostly sun.*
  - Most infamous is sun.misc.Unsafe

General Java Compatability Policy

- If an application uses only supported APIs on version N of Java it *should* work on version N+1, even without recompilation
- Supported APIs can be removed
  - But only with advanced notice
- JDK 8 has 18 interfaces, 23 classes, 64 fields, 358 methods and 20 constructors that have been deprecated
  - None have been removed
  - Until now

Most Popular Unsupported APIs

1. sun.misc.BASE64Encoder
2. sun.misc.Unsafe
3. sun.misc.BASE64Decoder

Oracle dataset based on internal application code

JDK Internal API Classification

- Non-critical
  - Little or no use outside the JDK
  - Used only for convenience (alternatives exist)
- Critical
  - Functionality that would be difficult, if not impossible to implement outside the JDK

JEP 260 Proposal

- Encapsulate all non-critical JDK-internal APIs
- Encapsulate all critical JDK-internal APIs, for which supported replacements exist in JDK 8
- Do *not* encapsulate other critical JDK-internal APIs
  - Deprecate these in JDK 9
  - Plan to encapsulate or remove them in JDK 10
  - Command-line option to access encapsulated critical APIs
    - `--add-exports`
    - `--add-opens`

JEP 260 Accessible Critical APIs

- sun.misc.Unsafe
- sun.misc.Signal
- sun.misc.SignalHandler
- sun.misc.Cleaner
- sun.reflect.Reflection.getCallerClass
- sun.reflect.ReflectionFactory

Project Jigsaw And Modules

Goals For Project Jigsaw

- Make Java SE more scalable and flexible
  - Internet of Things
  - Microservices
- Improve security, maintainability and performance

- Simplify construction, deployment and maintenance of large scale applications
- Eliminate classpath hell

Module Fundamentals

- Module is a grouping of code
  - For Java this is a collection of packages
- Modules can contain other things
  - Native code
  - Resources
  - Configuration data

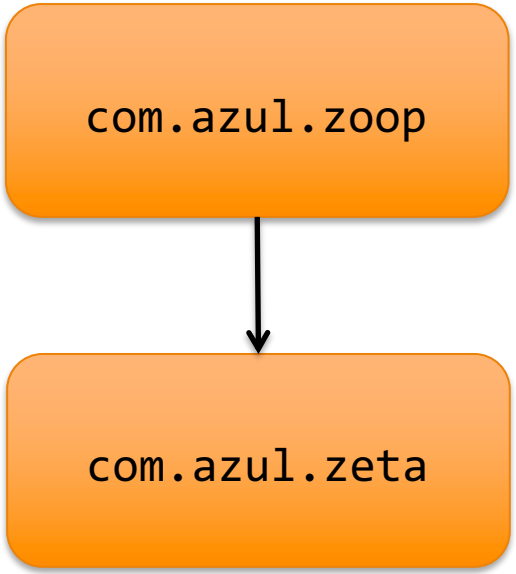
Module Declaration

```
module com.azul.zoop {
}
```

module-info.java
com/azul/zoop/alpha/Name.java
com/azul/zoop/alpha/Position.java
com/azul/zoop/beta/Animal.java
com/azul/zoop/beta/Reptile.java
com/azul/zoop/theta/Zoo.java
com/azul/zoop/theta/Lake.java

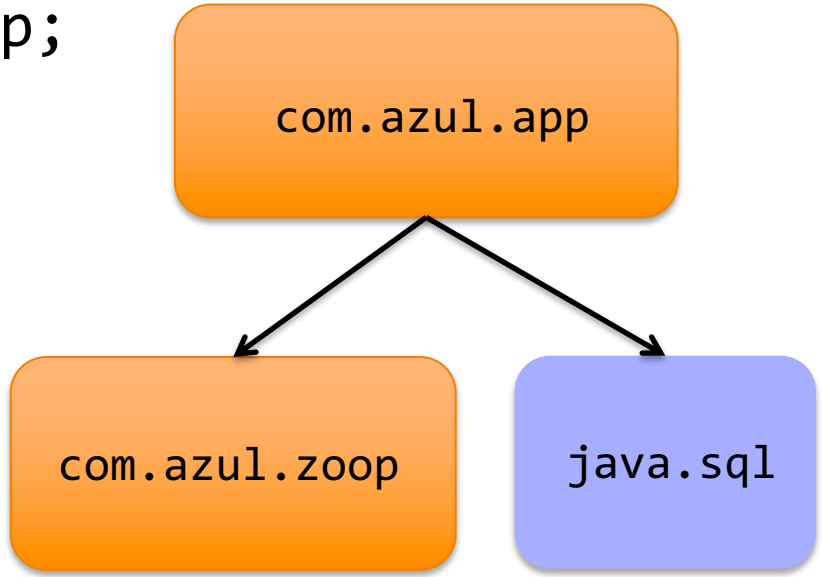
Module Dependencies

```
module com.azul.zoop {
  requires com.azul.zeta;
}
```

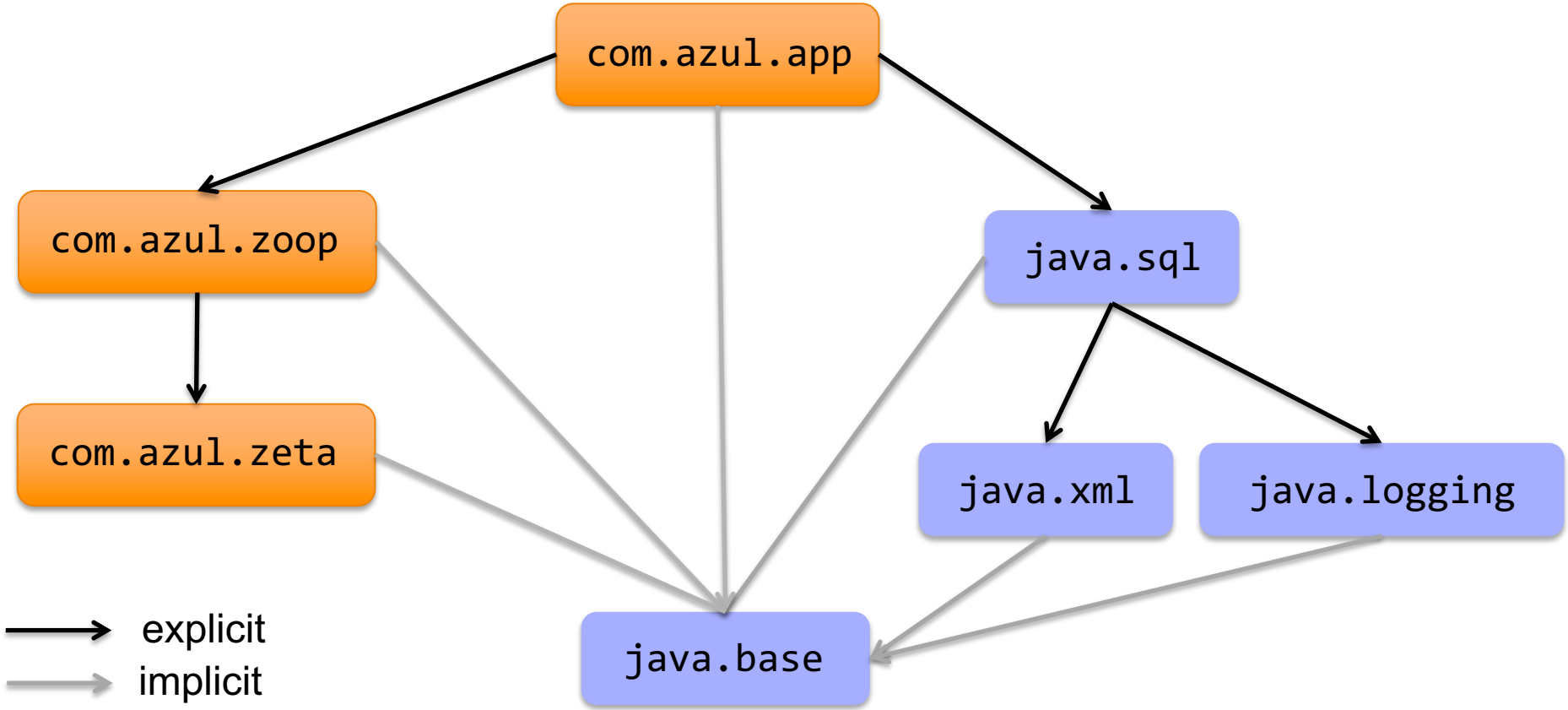


Module Dependencies

```
module com.azul.app {
  requires com.azul.zoop;
  requires java.sql;
}
```

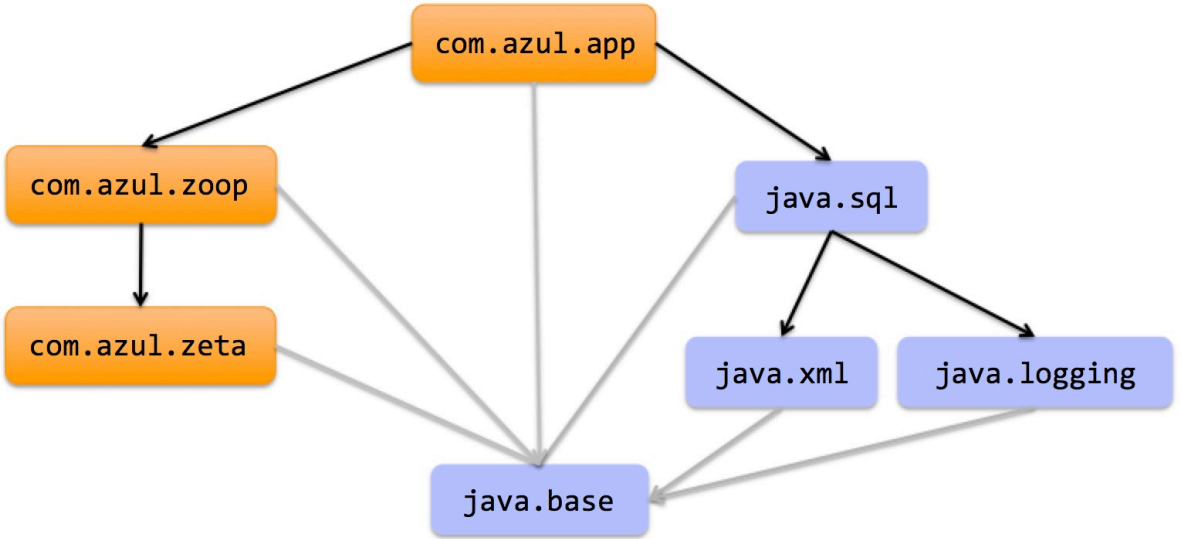


Module Dependency Graph

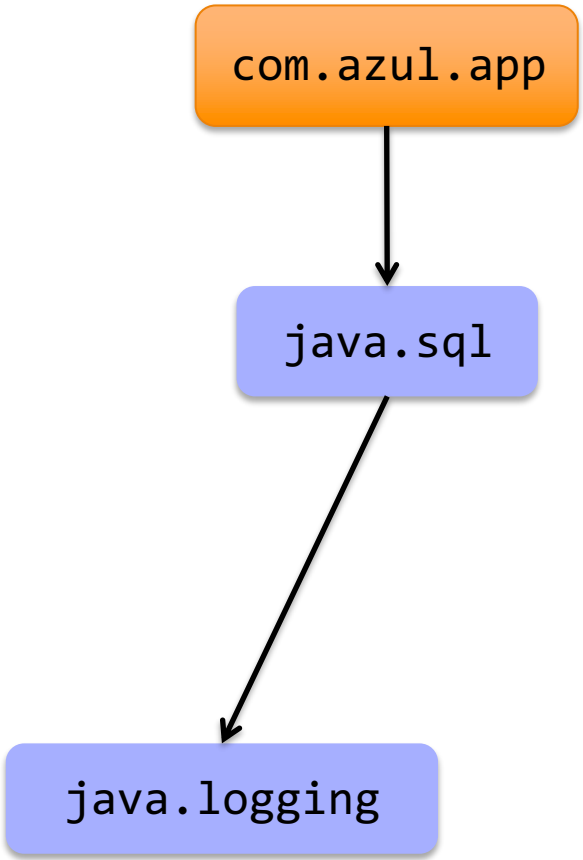


Module Dependency Graph

- No missing dependencies
- No cyclic dependencies
- No split packages



Readability v. Dependency

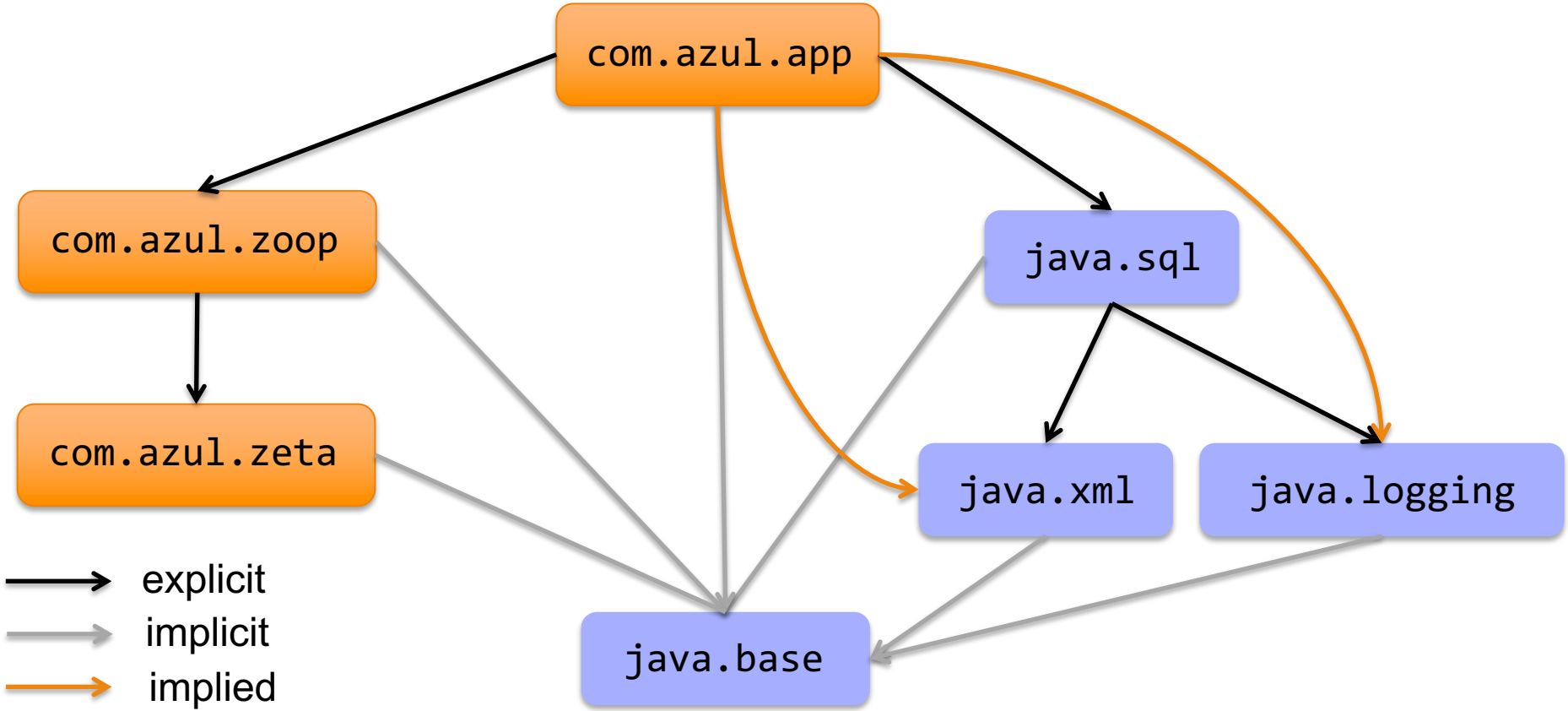



```
Driver d = …
Logger l = d.getParentLogger();
l.log(“azul’);
```

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

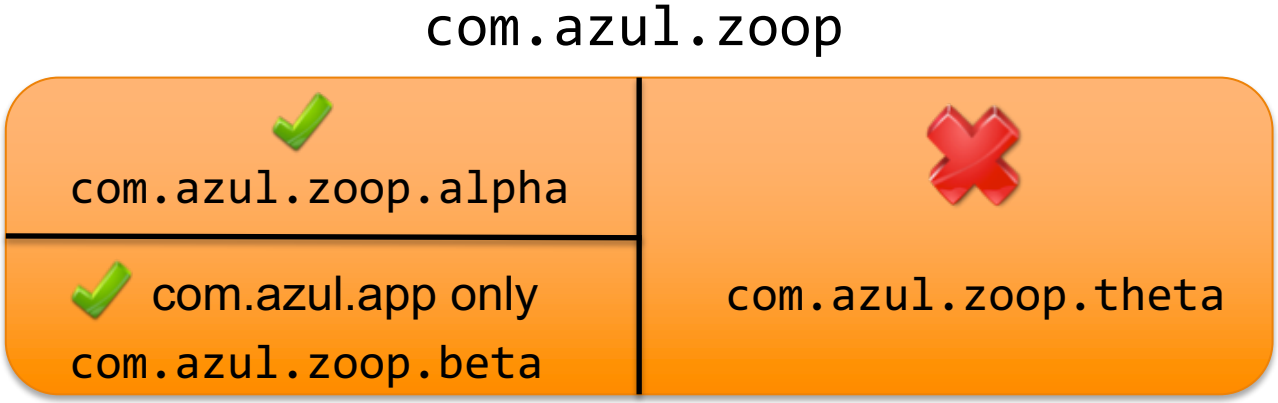
Implied readability

Module Implied Readability Graph



Package Visibility

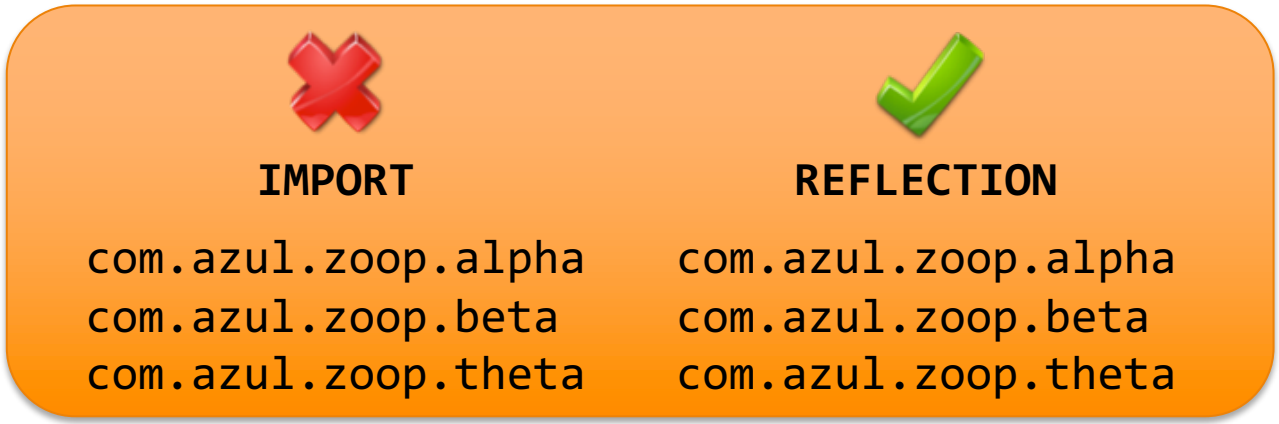
```
module com.azul.zoop {
   requires com.azul.zeta;
   exports com.azul.zoop.alpha;
   exports com.azul.zoop.beta to com.azul.app;
}
```



More Package Visibility

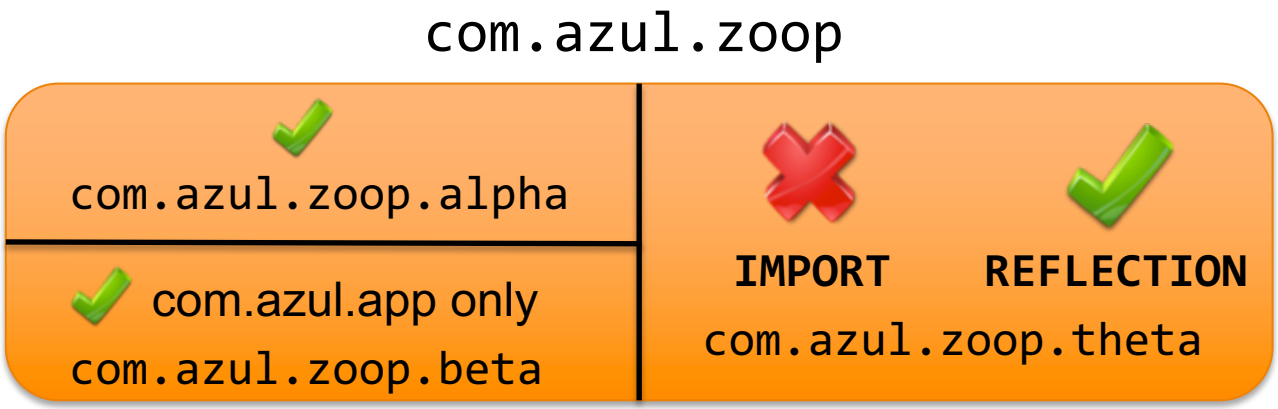
```
open module com.azul.zoop {
  requires com.azul.zeta;
}
```

com.azul.zoop



Even More Package Visibility

```
module com.azul.zoop {
   requires com.azul.zeta;
   exports com.azul.zoop.alpha;
   exports com.azul.zoop.beta to com.azul.app;
   opens com.azul.theta;
}
```



Restricted Keywords

```
module module {
  requires requires;
  exports exports to to;
  opens opens;
}
```

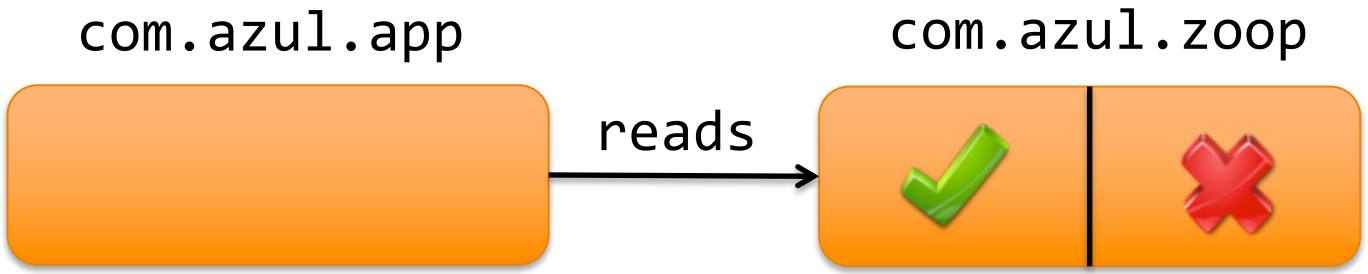
Optional Dependencies

- Required at compile time
- Possibly not needed at runtime
  - If it is generates a `NoClassDefFoundError`
- Use static keyword
  - Oddly this used to be optional, which is more logical

```
module mine.lib {
  requires static com.google.guava
}
```

Accessibility

- For a package to be visible
  - The package must be exported by the containing module
  - The containing module must be read by the using module
- Public types from those packages can then be used



Java Accessibility (pre-JDK 9)

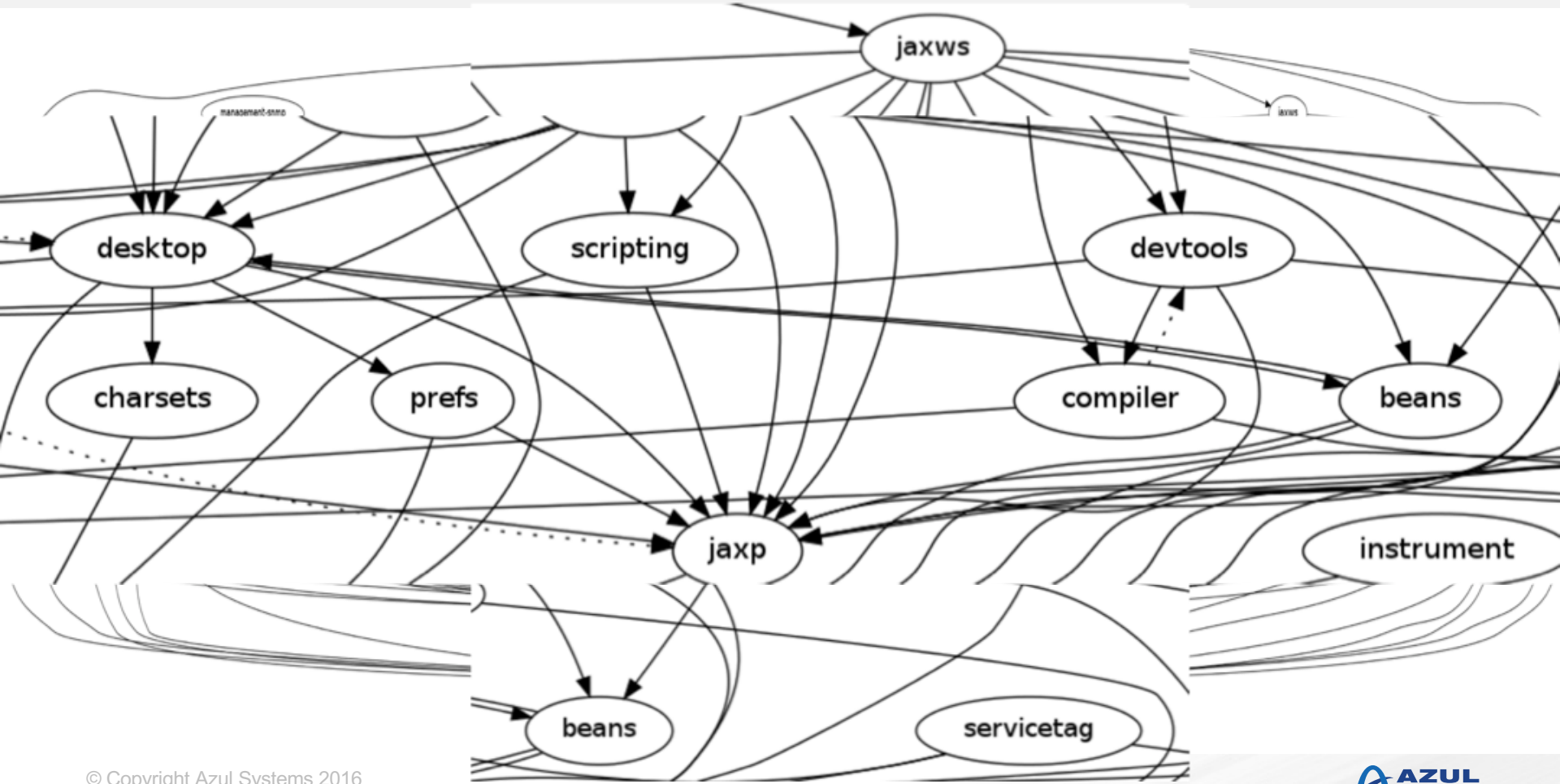
```
public
protected
<package>
private
```

Java Accessibility (JDK 9)

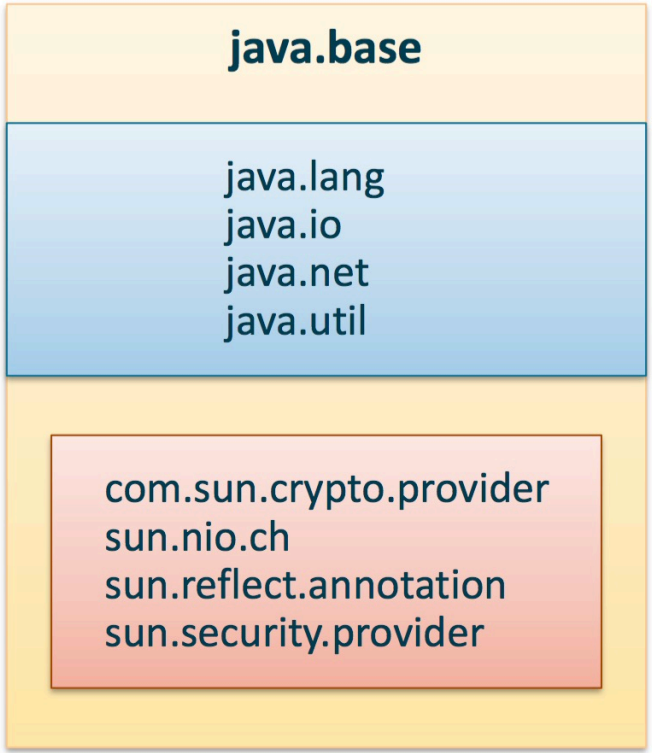
public to everyone
public, but only to specific modules
public only within a module
protected
<package>
private

public ≠ accessible (fundamental change to Java)

JDK 8 Dependencies

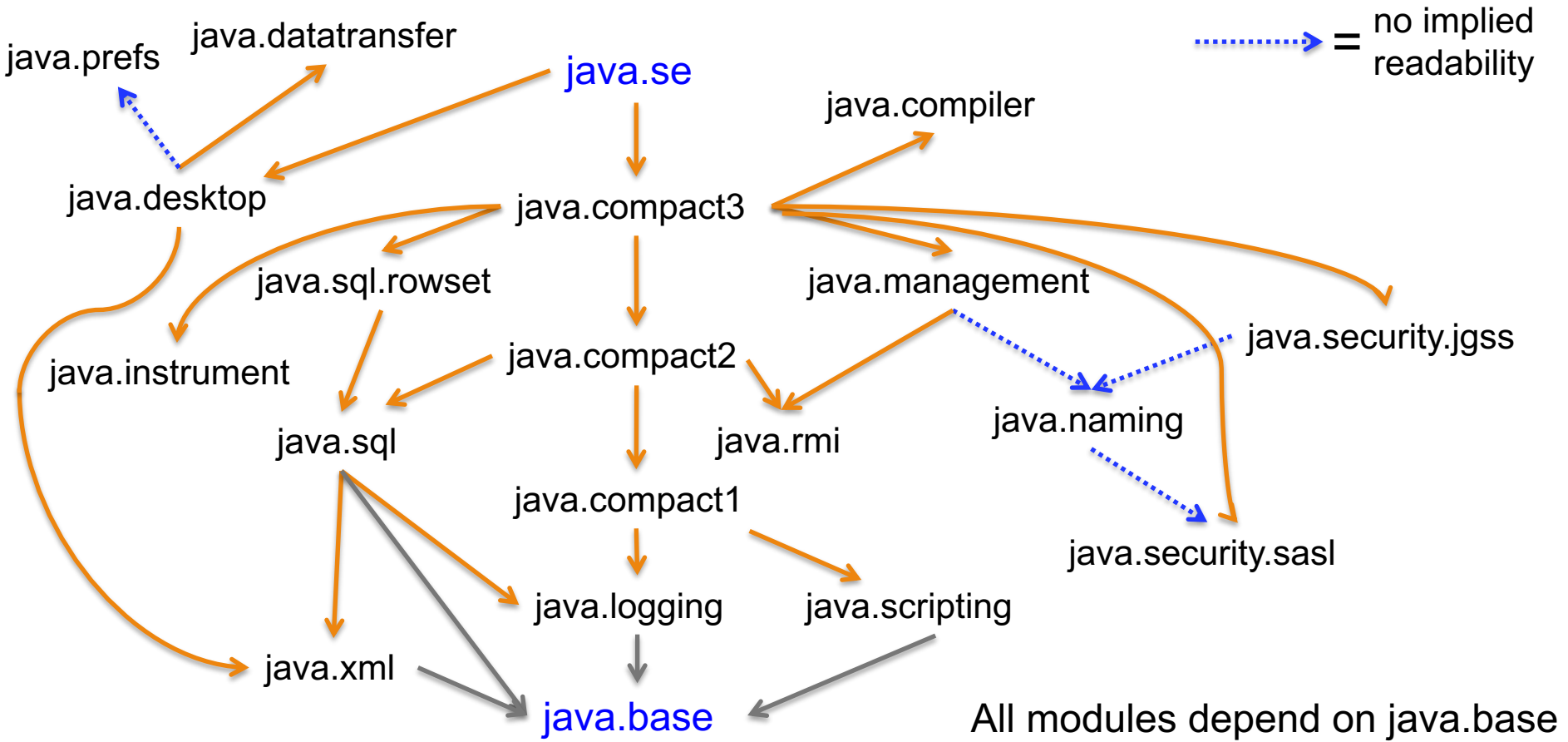


The java.base Module

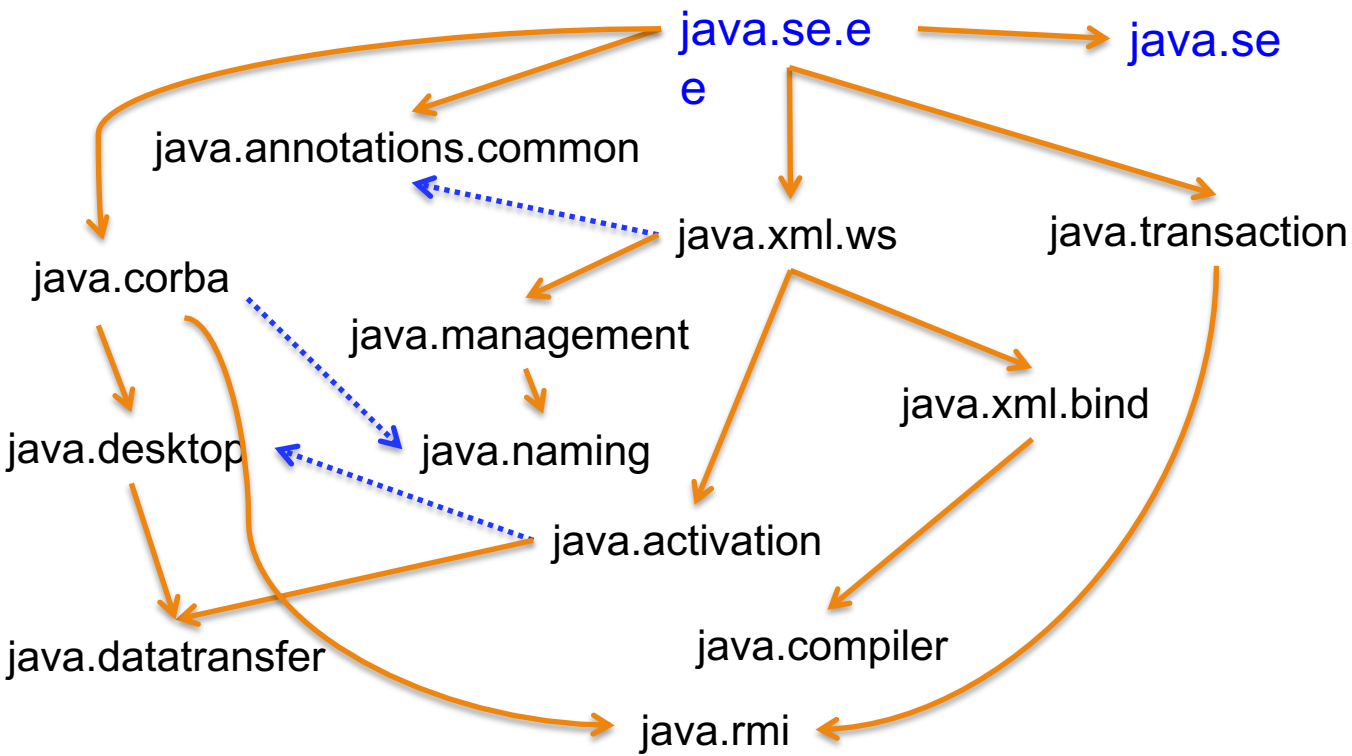



```
module java.base {
  exports java.lang;
  exports java.io;
  exports java.net;
  exports java.util;
}
```

JDK 9 Platform Modules



JDK 9 Platform Modules

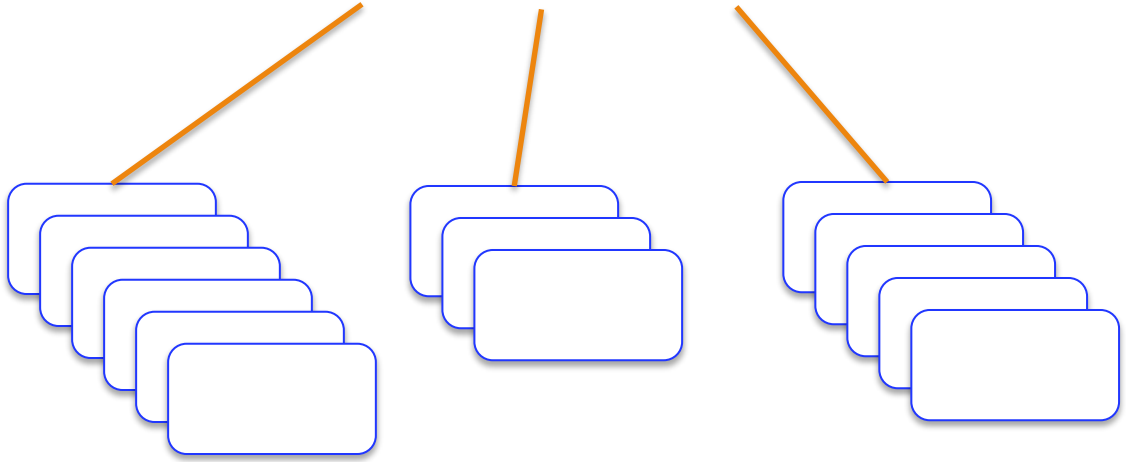


All modules depend on java.base

Using Modules

Module Path

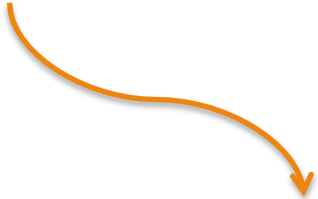
```
$ javac –modulepath dir1:dir2:dir3
```



Compilation With Module Path

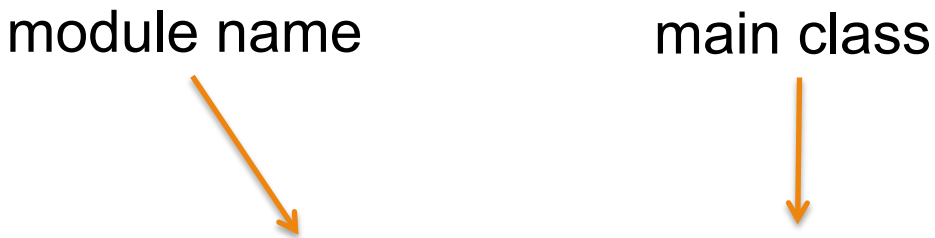
```
$ javac --module-path mods -d mods \
  src/module-info.java \
  src/com/azul/zoop/alpha/Name.java
```

```
src/module-info.java
src/com/azul/zoop/alpha/Name.java
```



```
mods/module-info.class
mods/com/azul/zoop/alpha/Name.class
```

Application Execution




```
$ java -p mods -m com.azul.app/com.azul.app.Main

Azul application initialised!
```

- `--module-path` can be abbreviated to `-p`

Packaging With Modular JAR Files

```
mods/module-info.class
mods/com/azul/app/Main.class
```

app.jar

```
module-info.class
com/azul/app/Main.class
```

```
$ jar --create --file=mylib/app.jar \
    --main-class=com.azul.app.Main \
    -C mods .
```

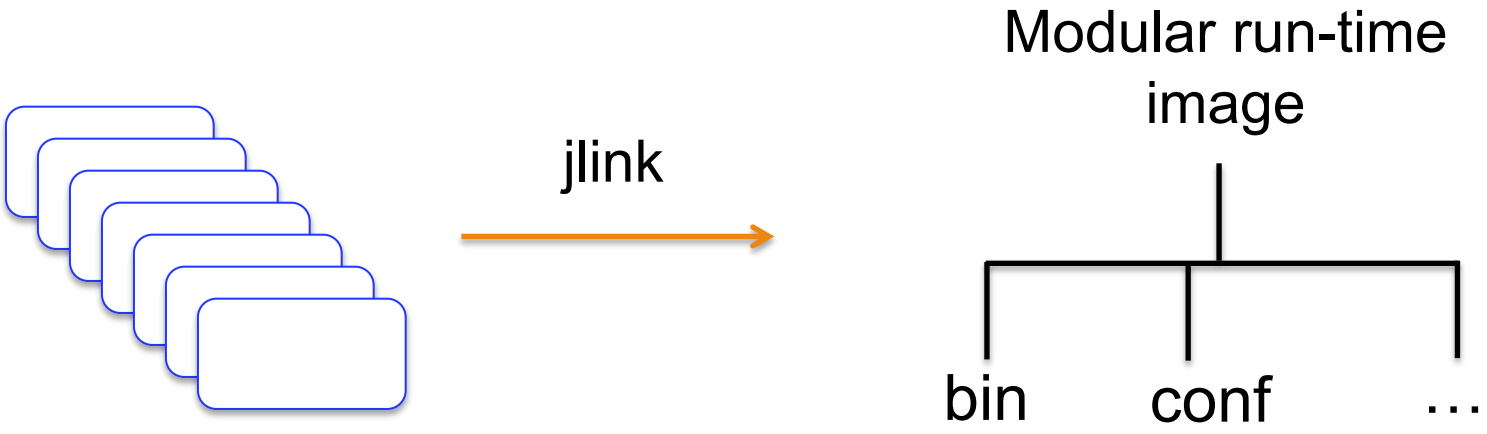
JAR Files & Module Information

```
$ jar --file=mylib/app.jar --print-module-descriptor
$ jar --file=mylib/app.jar -d

com.azul.app
  requires com.azul.zoop
  requires com.azul.zeta
  requires mandated java.base
  requires java.sql
  main-class com.azul.zoop.Main
```

Application Execution (JAR)

```
$ java -p mylib:mods -m com.azul.app

Azul application initialised!
```

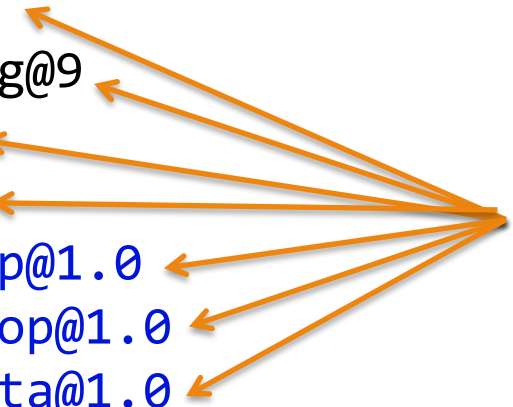
Linking




```
$ jlink --module-path $JDKMODS \
  --add-modules java.base –-output myimage

$ myimage/bin/java –-list-modules
java.base@9.0
```

Linking An Application

```
$ jlink --module-path $JDKMODS:$MYMODS \
  --add-modules com.azul.app --output myimage

$ myimage/bin/java --list-modules
java.base@9
java.logging@9
java.sql@9
java.xml@9
com.azul.app@1.0
com.azul.zoop@1.0
com.azul.zeta@1.0
```



Version numbering for information purposes only

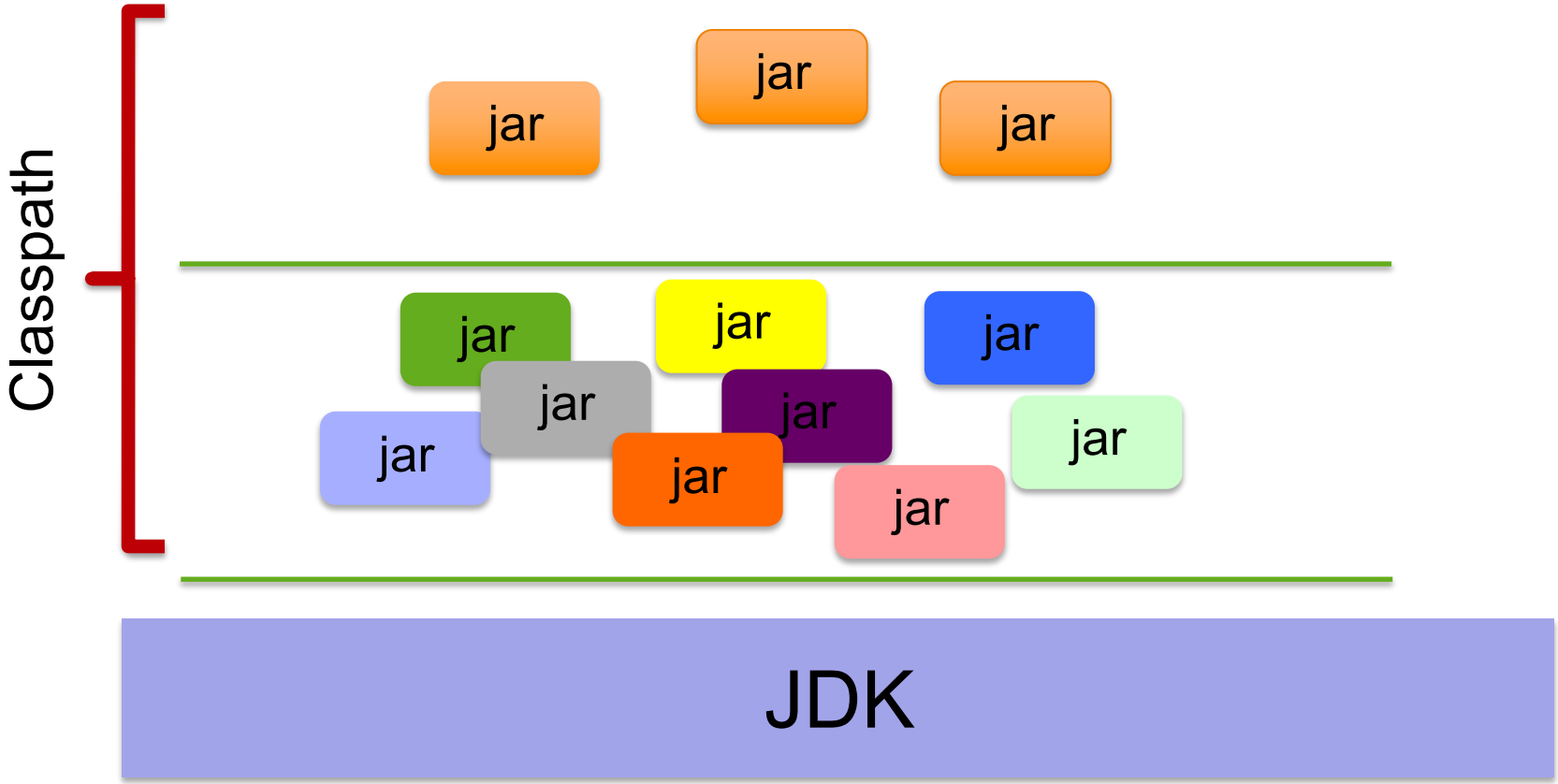
"It is not a goal of the module system to solve the version-selection problem"

The Implications Of `jlink`

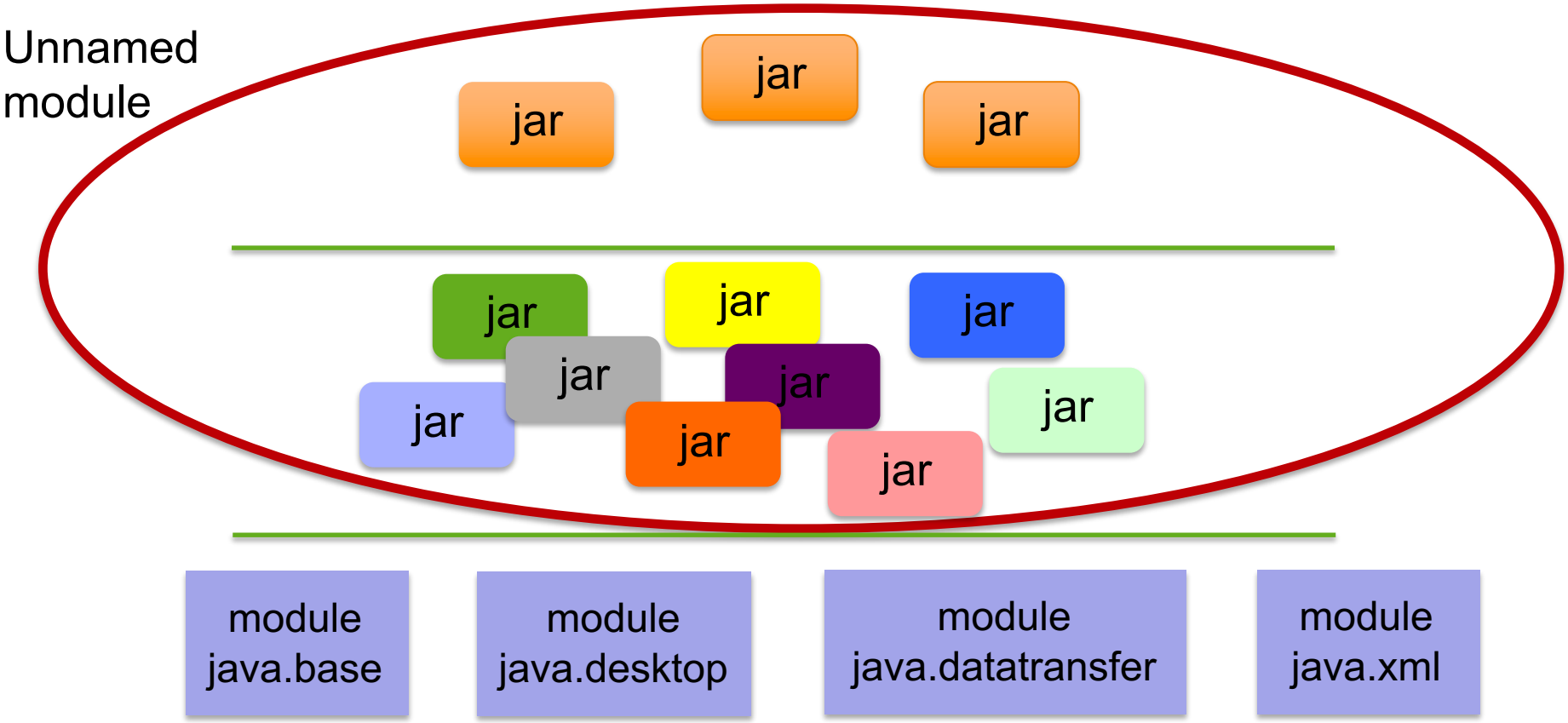
- "Write once, run anywhere"
  - Long term Java slogan, mainly held true
- `jlink` generated runtime may not include all Java SE modules
  - But is still a conforming Java implementation
  - To conform to the specification, the runtime:
    - Must include the `java.base` module
    - If other modules are included, all transitive module dependencies must also be included
      - Defined as a closed implementation

Application Migration

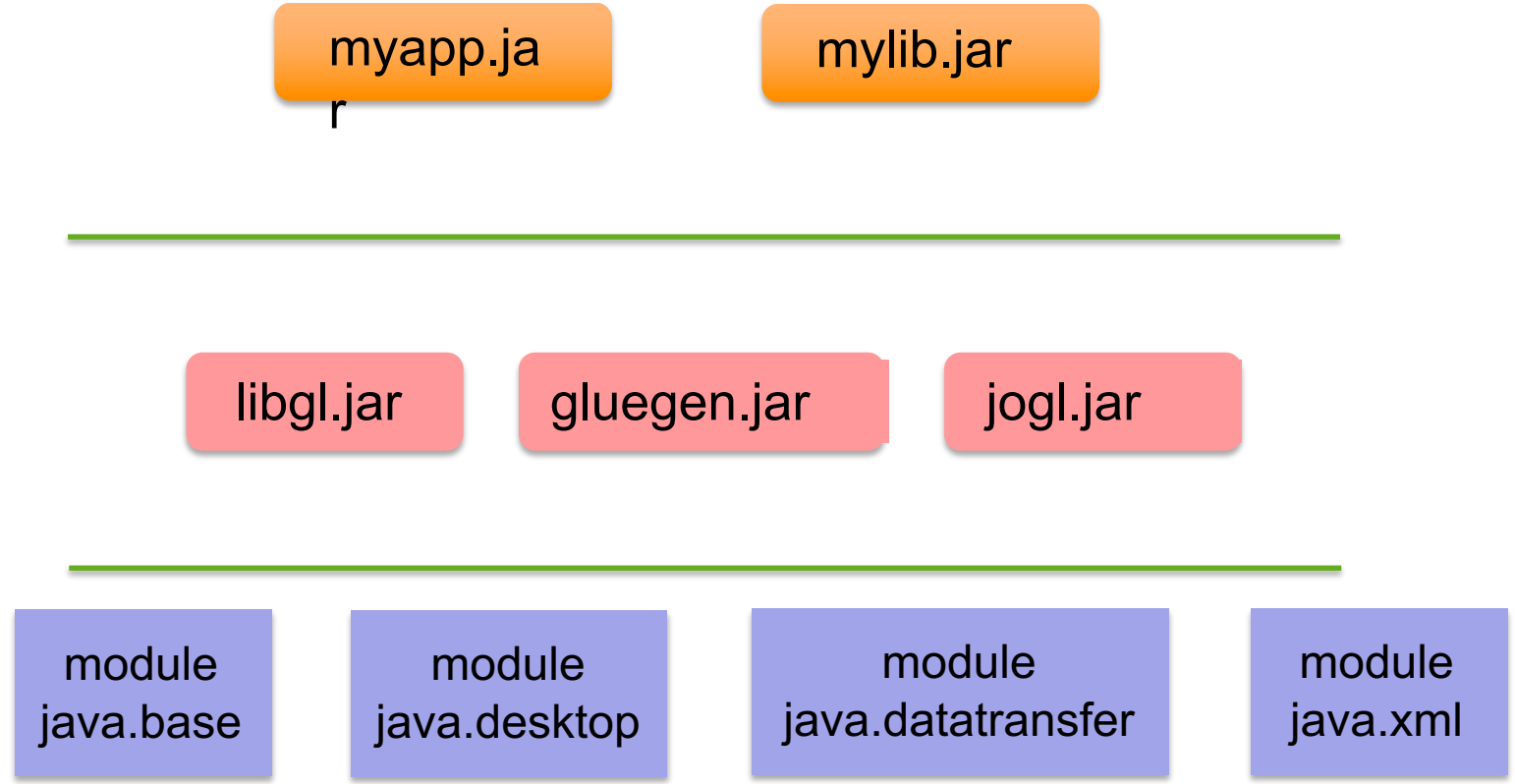
Typical Application (JDK 8)



Typical Application (JDK 9)



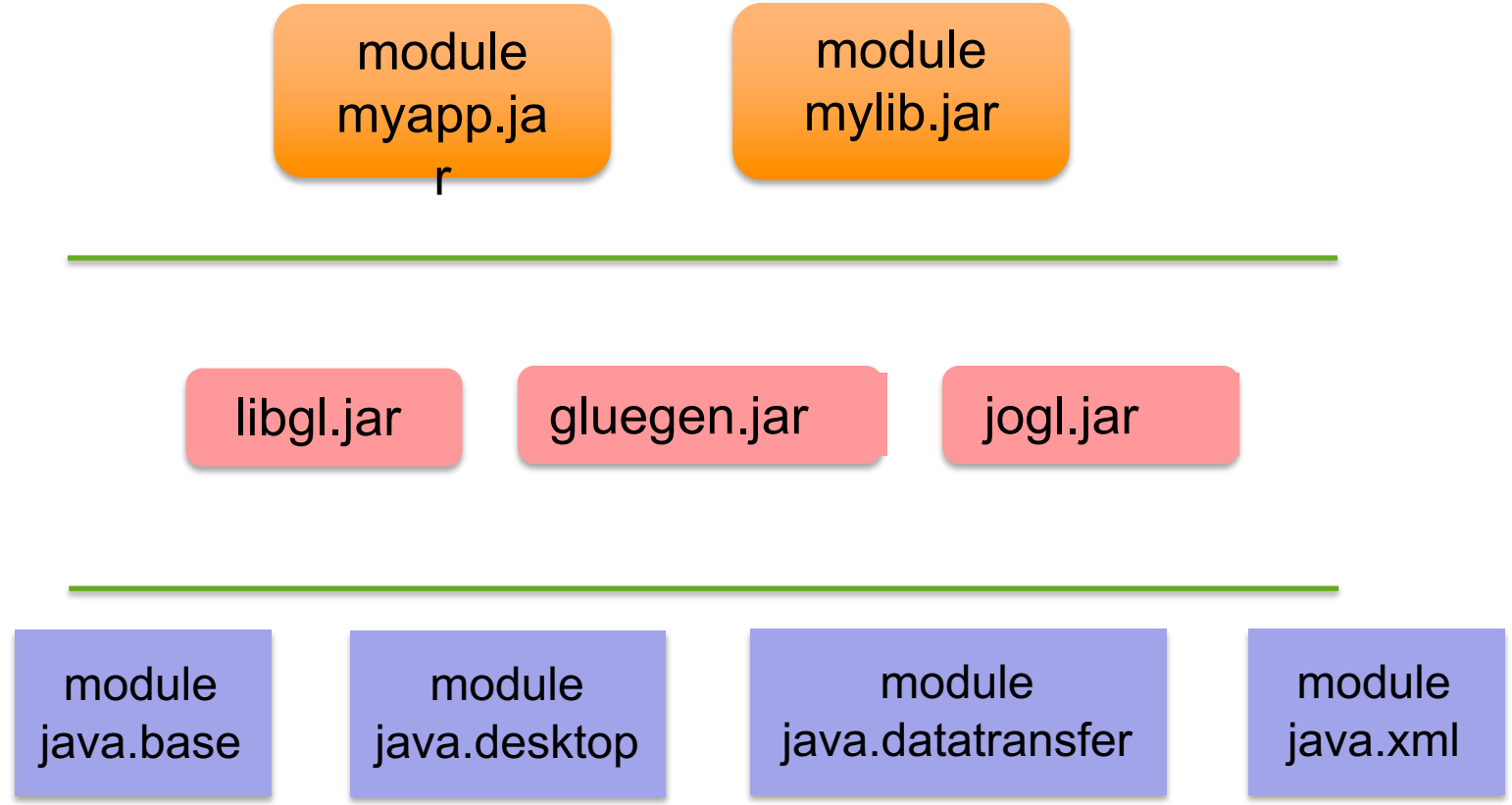
Sample Application



Run Application With Classpath

```
$ java –classpath \
  lib/myapp.jar: \
  lib/mylib.jar: \
  lib/libgl.jar: \
  lib/gluegen.jar: \
  lib/jogl.jar: \
  myapp.Main
```

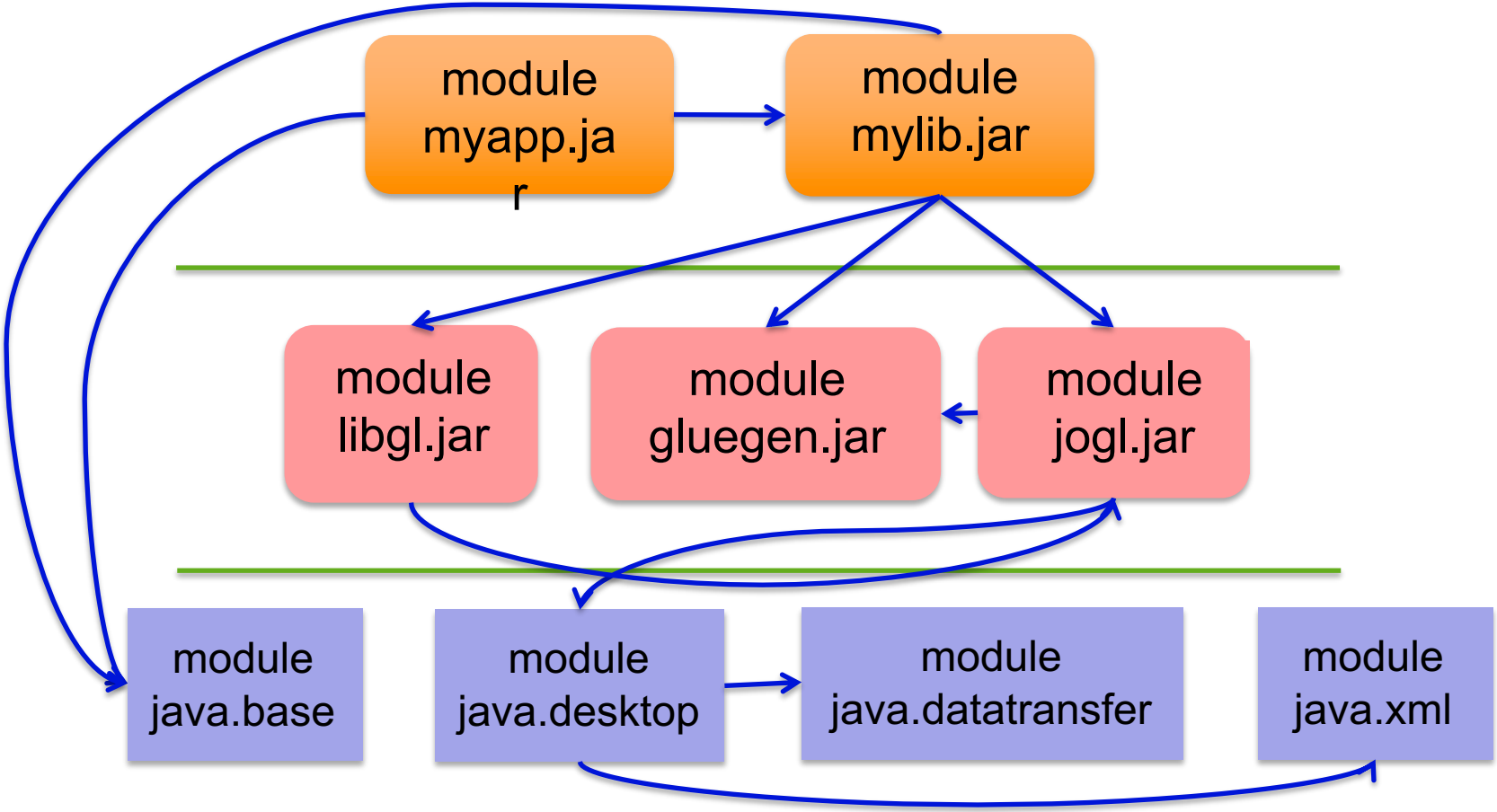
Sample Application



Application module-info.java

```
module myapp {
  requires mylib;
  requires java.base;
  requires java.sql;
  requires libgl;          ????
  requires gluegen;        ????
  requires jogl;           ????
}
```

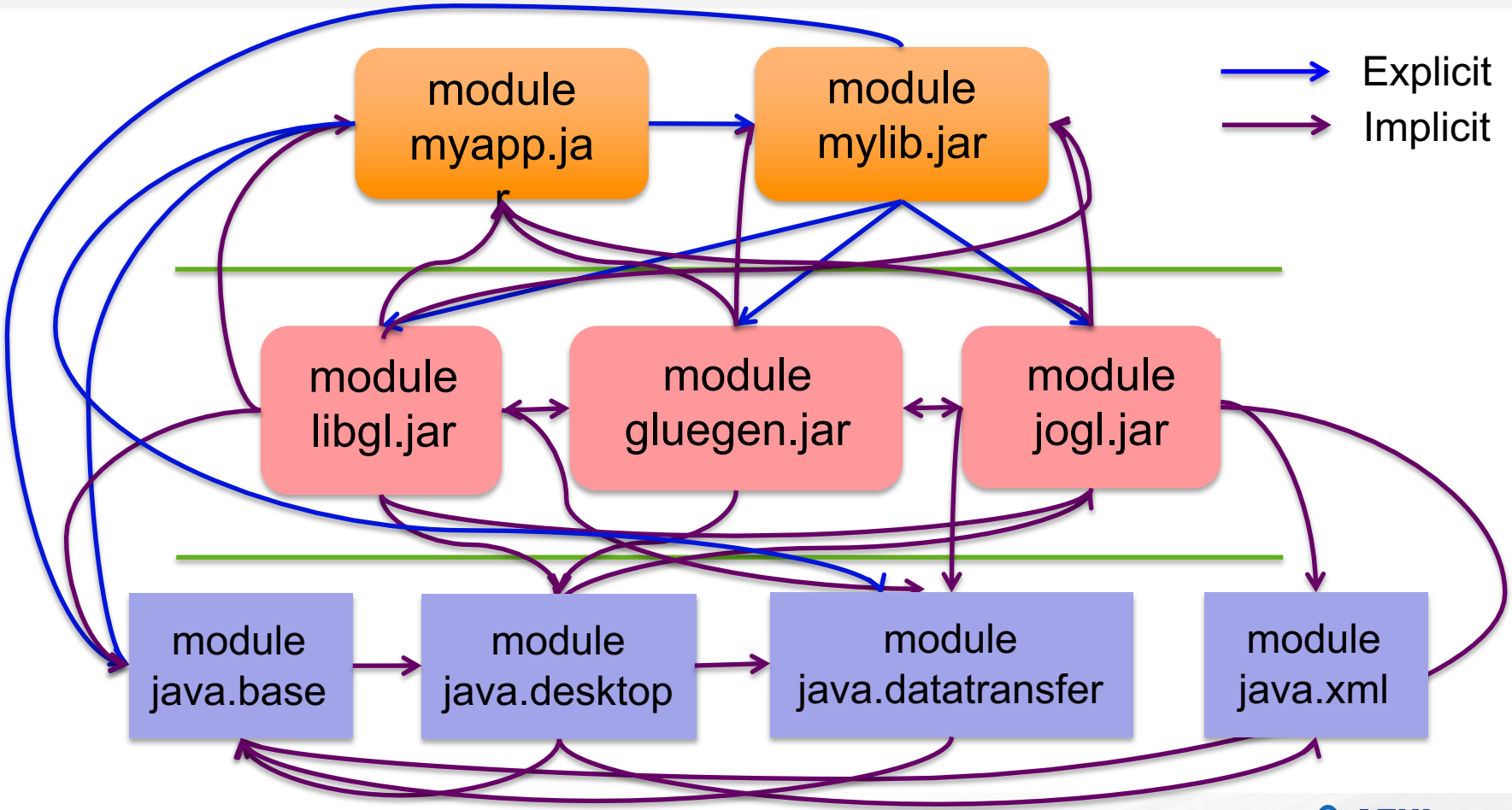
Sample Application



Automatic Modules

- Real modules
- Simply place unmodified jar file on module path
  - Rather than classpath
- No changes to JAR file
- Module name derived from JAR file name
- Exports all its packages
  - No selectivity
- Automatically requires all modules on the module path

Application Module Dependencies



Run Application With Modules

```
$ java –classpath \
  lib/myapp.jar: \
  lib/mylib.jar: \
  lib/libgl.jar: \
  lib/gluegen.jar: \
  lib/jogl.jar: \
  myapp.Main

$ java –p mylib:lib –m myapp
```

Advanced Details

Modular Jar Files And JMODs

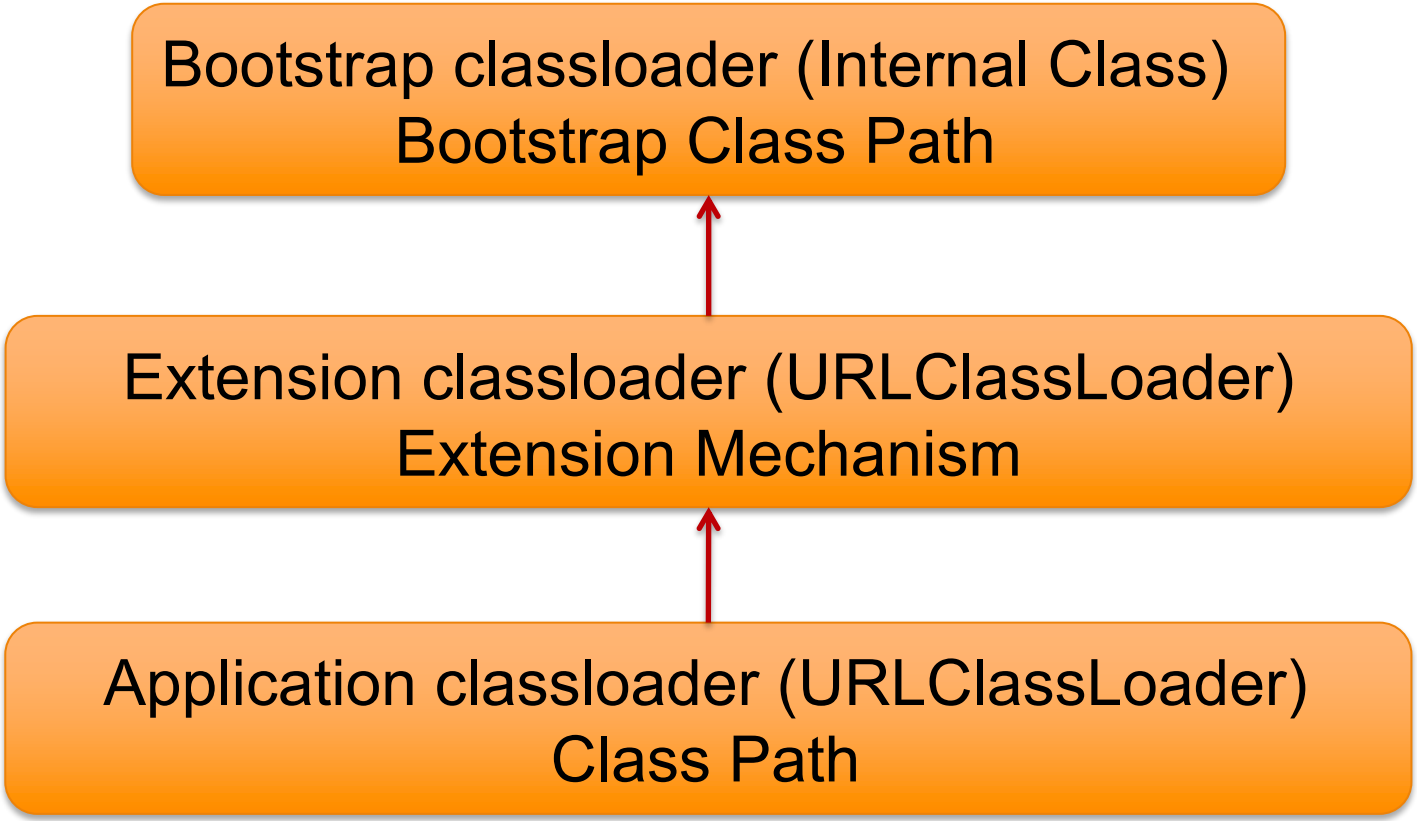
- Modular jar files are simple
  - Standard jar file possibly with module-info.class file
  - Can use existing (unmodified) jar files
- JMOD files
  - More complex module files
  - Used for modules in the JDK
  - Can include native files (JNI), configuration files and other data
  - Based on zip file format (pending final details - JEP 261)

`jmod` Command

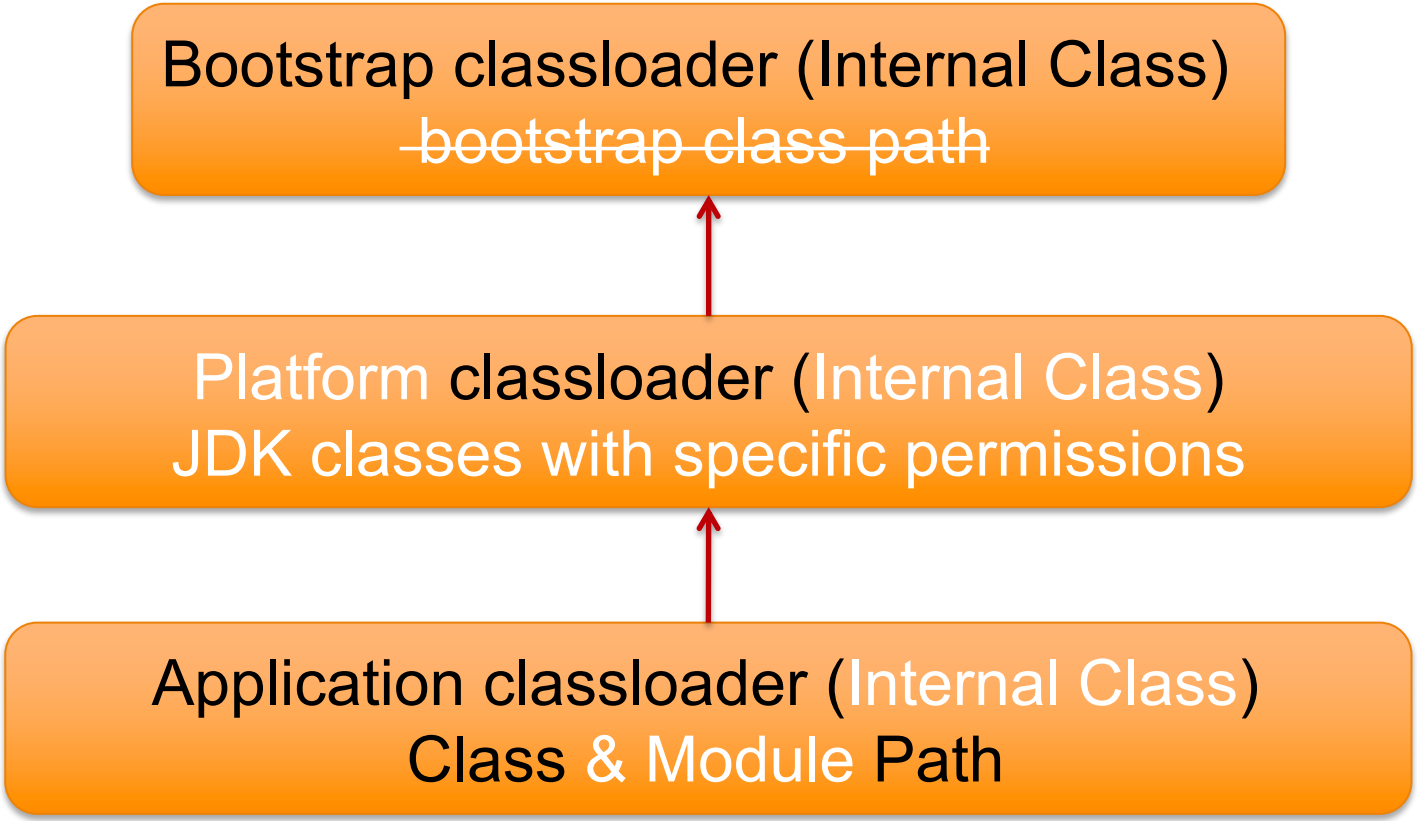
```
jmod (create | list | describe) <options> <jmod-file>
```

- Create can specify several details:
  - Main class
  - Native libraries and commands
  - Configuration data
  - OS name, version and machine architecture
- Details included in `module-info.class` file

Classloaders (Since JDK 1.2)



Classloaders (JDK 9)



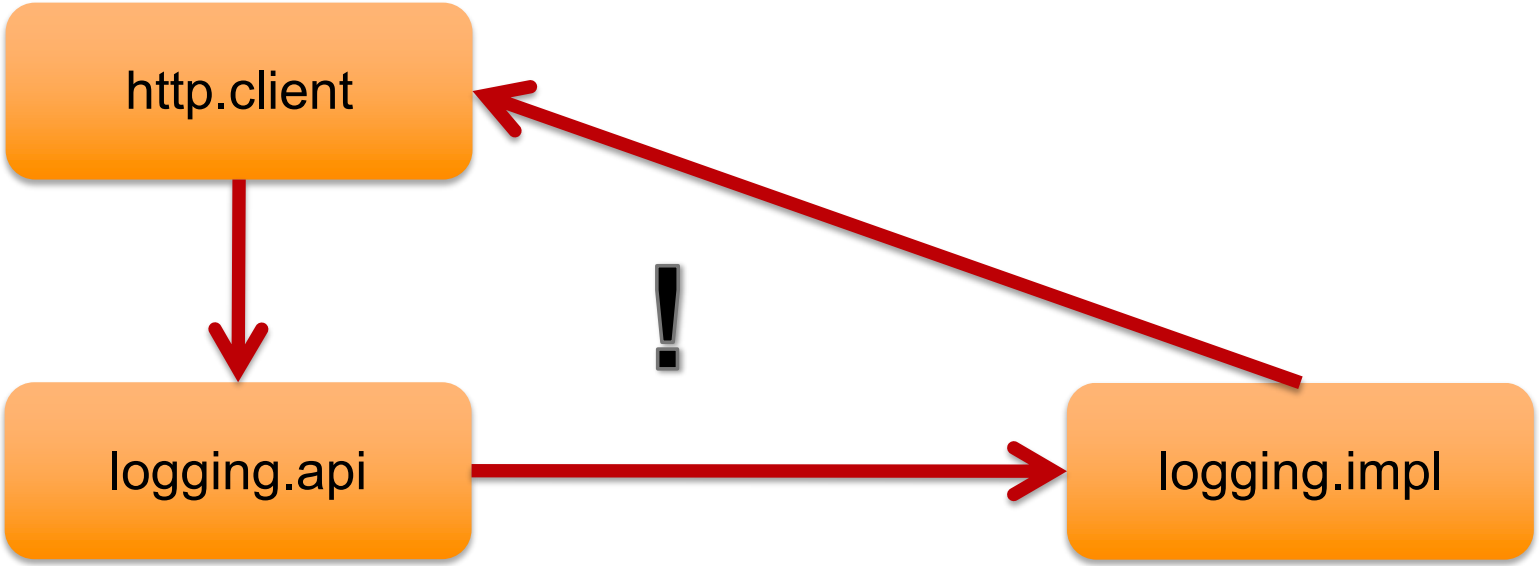
Services

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

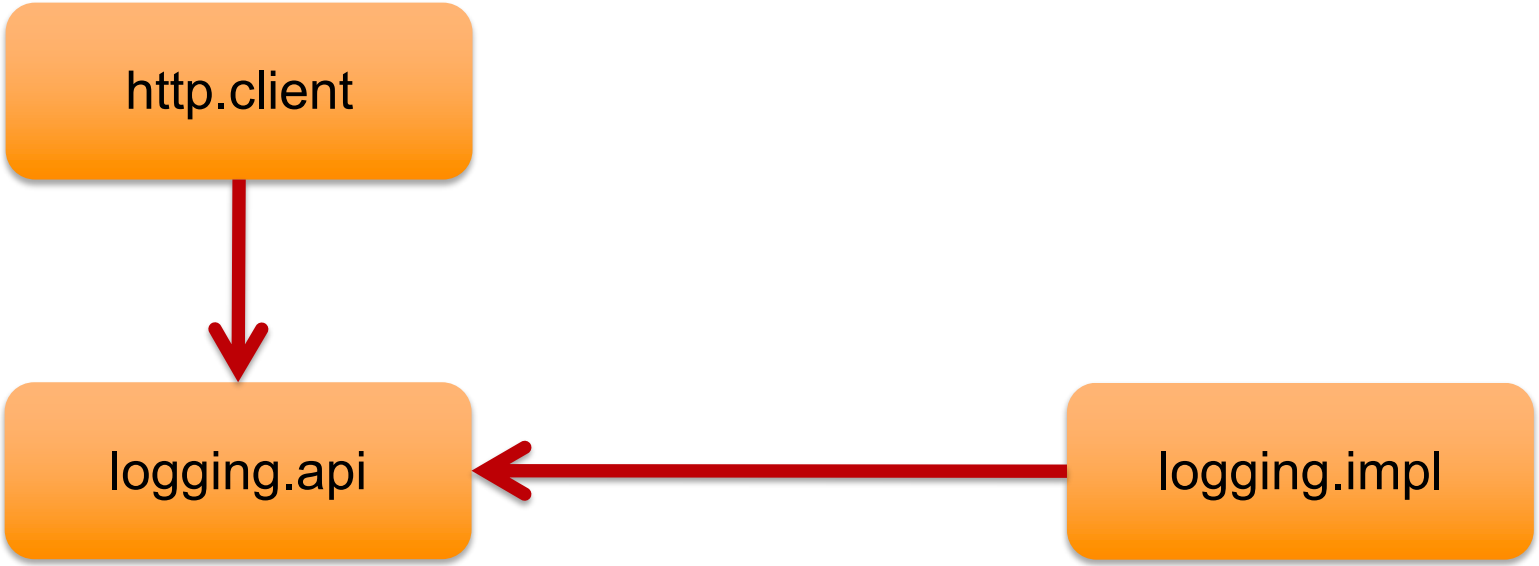
Services

```
module com.mysql.jdbc {
  requires java.sql;
  exports com.mysql.jdbc;
  provides java.sql.Driver with com.mysql.jdbc.Driver;
}
```

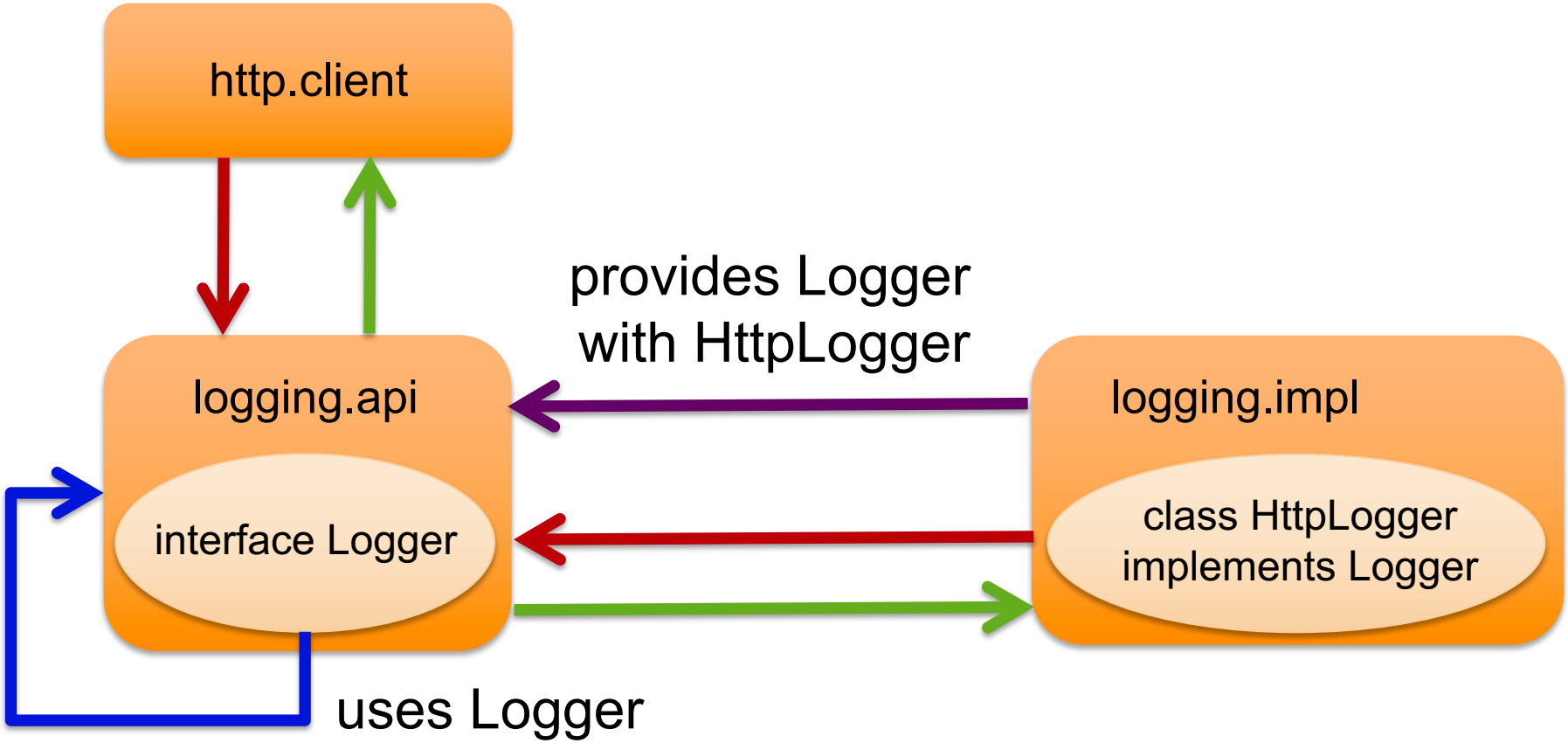
Avoiding Cyclic Dependencies



Avoiding Cyclic Dependencies



Avoiding Cyclic Dependencies



Incubator Modules (JEP 11)

- Develop APIs without making them part of the standard
  - At least not straight away
- Allow developers to “kick the tyres”
  - Not always possible to get a new API right first time
- Move from incubator to full module
  - Becomes part of the standard
- JDK 9 only has one incubator: HTTP/2 (JEP 110)
- Some concerns about fragmentation

```
--do-not-resolve-by-default
```

JDK 9 Developer Features

Java Language

Milling Project Coin (JEP 213)

- Single underscore is now a reserved word
  - No longer usable as an identifier
  - Two or more underscores still works
- Allow `@SafeVarargs` on private instance methods
  - In addition to constructors, final and static methods

Milling Project Coin

- Private methods are now permitted in interfaces
  - Separate common code for default and static methods

```
public interface MyInterface {
  default int getX() {
    return getNum() * 2;
  };

  static int getY() {
    return getNum() * 4;
  }

  private static int getNum() {
    return 12;
  }
}
```

Milling Project Coin

- Effectively final variables in try-with-resources

```
BufferedReader reader =
  new BufferedReader(new FileReader("/tmp/foo.txt"));

try (BufferedReader myReader = reader) {
  myReader.lines()
    .forEach(System.out::println);
}
```

JDK 8

Milling Project Coin

- Diamond operator with anonymous classes
  - Type inference can infer a non-denotable type
  - If the type inferred can be denotable you can use <>

```
public abstract class Processor<T> {
  abstract void use(T value);
}

Processor<String> processor = new Processor<>() {
  @Override
  void use(String value) {
    // Some stuff
  }
};
```

Core Libraries

Factory Methods For Collections (JEP 269)

- The problem:
  - Creating a small unmodifiable collection is verbose

```
Set<String> set = new HashSet<>();
set.add("a");
set.add("b");
set.add("c");
set = Collections.unmodifiableSet(set);
```

Factory Methods For Collections (JEP 269)

- Static methods added to `List`, `Set` and `Map` interfaces
  - Create compact, immutable instances
  - 0 to 10 element overloaded versions
  - Varargs version for arbitary number of elements

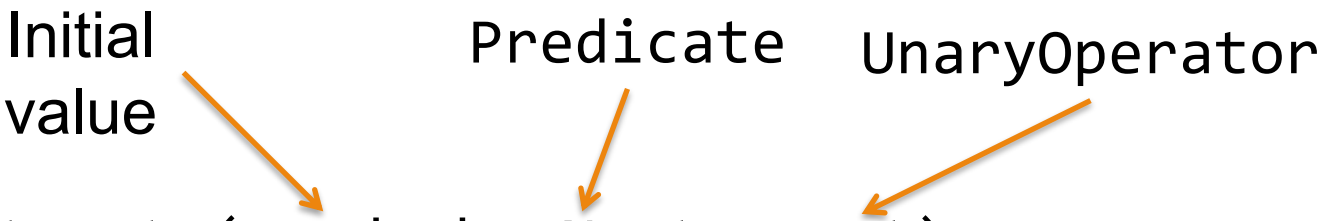
```
Set<String> set = Set.of("a", "b", "c");

List<String> list = List.of("a", "b", "c");

Map<String, String> map = Map.of(“k1”, “v1”, “k2”, “v2”);

Map<String, String> map = Map.ofEntries(
  entry(“k1”, “v1”), entry(“k2”, “v2”));
```

Stream Enhancements

- `Optional` now has a `stream()` method
  - Returns a stream of one element or an empty stream
- `Collectors.flatMapping()`
  - Returns a `Collector` that converts a stream from one type to another by applying a flat mapping function

```
Map<String, Set<LineItem>> items = orders.stream()
  .collect(groupingBy(Order::getCustomerName,
     flatMapping(order -> order.getLineItems().stream(), toSet())));
```

Improved Iteration



- `iterate(seed, hasNext, next)`
  - Can be used like the classic for loop

```
IntStream.iterate(0, i -> i < 10, i -> ++i)
  .forEach(System.out::println);

IntStream.iterate(0, i -> i < 10, i -> i++)
  .forEach(System.out::println);
```

Infinite stream

Stream takeWhile

- `Stream<T> takeWhile(Predicate<? super T> p)`
- Select elements from stream while `Predicate` matches
- Unordered stream needs consideration

```
thermalReader.lines()
  .mapToInt(i -> Integer.parseInt(i))
  .takeWhile(i -> i < 56)
  .forEach(System.out::println);
```

Stream dropWhile

- `Stream<T> dropWhile(Predicate<? super T> p)`
- Ignore elements from stream while `Predicate` matches
- Unordered stream still needs consideration

```
thermalReader.lines()
  .mapToInt(i -> Integer.parseInt(i))
  .dropWhile(i -> i < 56)
  .forEach(System.out::println);
```

dropWhile/takeWhile

- Be careful of unordered streams missing values

```
List<String> list = List.of("alpha", "bravo", "charlie",
  "delta", "echo", "foxtrot");
```

```
list.stream()
  .dropWhile(s -> s.charAt(0) < 'e')
  .forEach(System.out::println);
```

echo
foxtrot

```
list.stream()
  .takeWhile(s -> s.length() == 5)
  .forEach(System.out::println);
```

alpha
bravo

Additional Stream Sources

- `Matcher` stream support
  - `Stream<MatchResult> results()`
- `Scanner` stream support
  - `Stream<MatchResult> findAll(String pattern)`
  - `Stream<MatchResult> findAll(Pattern pattern)`
  - `Stream<String> tokens()`

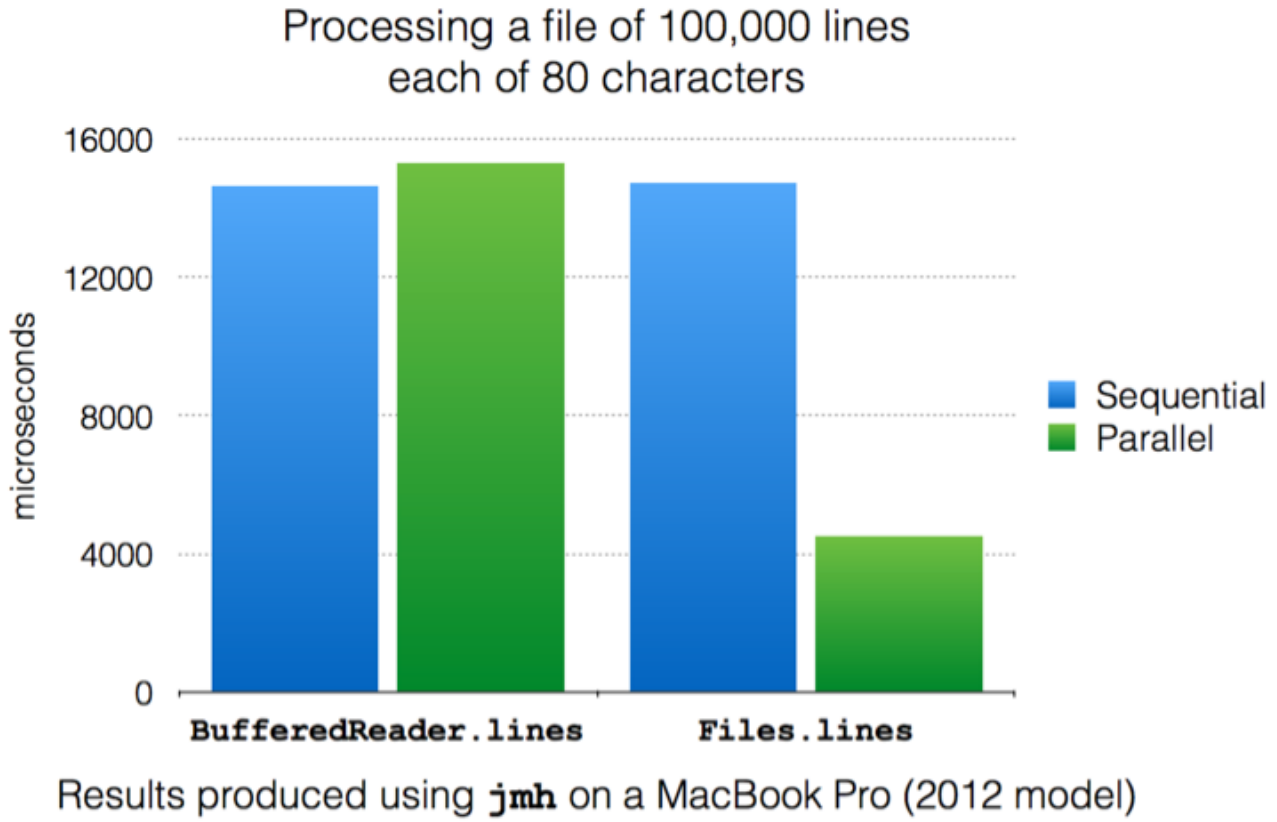
Additional Stream Sources

- java.net.NetworkInterface
  - Stream<InetAddress> inetAddresses()
  - Stream<NetworkInterface> subInterfaces()
  - Stream<NetworkInterface> networkInterfaces()
    - static
- java.security.PermissionCollection
  - Stream<Permission> elementsAsStream()

Parallel Support For Files.lines()

- Memory map file for UTF-8, ISO 8859-1, US-ASCII
  - Character sets where line feeds easily identifiable
- Efficient splitting of mapped memory region
- Divides approximately in half
  - To nearest line feed

Parallel Lines Performance



Optional: New Methods

- `ifPresentOrElse(Consumer action, Runnable emptyAction)`
  - If a value is present call accept() on action with the value
  - Otherwise, run the emptyAction
    - Not in a separate thread
- `or(Supplier<? extends Optional<? extends T>> supplier)`
  - Return the `Optional` if there is a value
  - Otherwise, return a new `Optional` created by the `Supplier`
- `stream()`
  - Returns a stream of zero or one element

Smaller Features

- Process API updates (JEP 102)
  - Native process
  - `Process`/`ProcessHandle`/`ProcessHandle.Info`
    - `Process.toHandle()`
  - More information about process
    - Command. command line, arguments, CPU duration, user
  - Control subject to security manager permissions

Multi-Release Jar Files (JEP 238)

- Multiple Java release-specific class files in a single archive
- Enhance `jar` tool to create multi-release files
- Support multi-release jar files in the JRE
  - Classloaders
  - `JarFile` API
- Enhance other tools
  - `javac`, `javap`, `jdeps`, etc.
- Also, modular jar files

jshell
Read-Eval-Print Loop
(REPL)
Demo

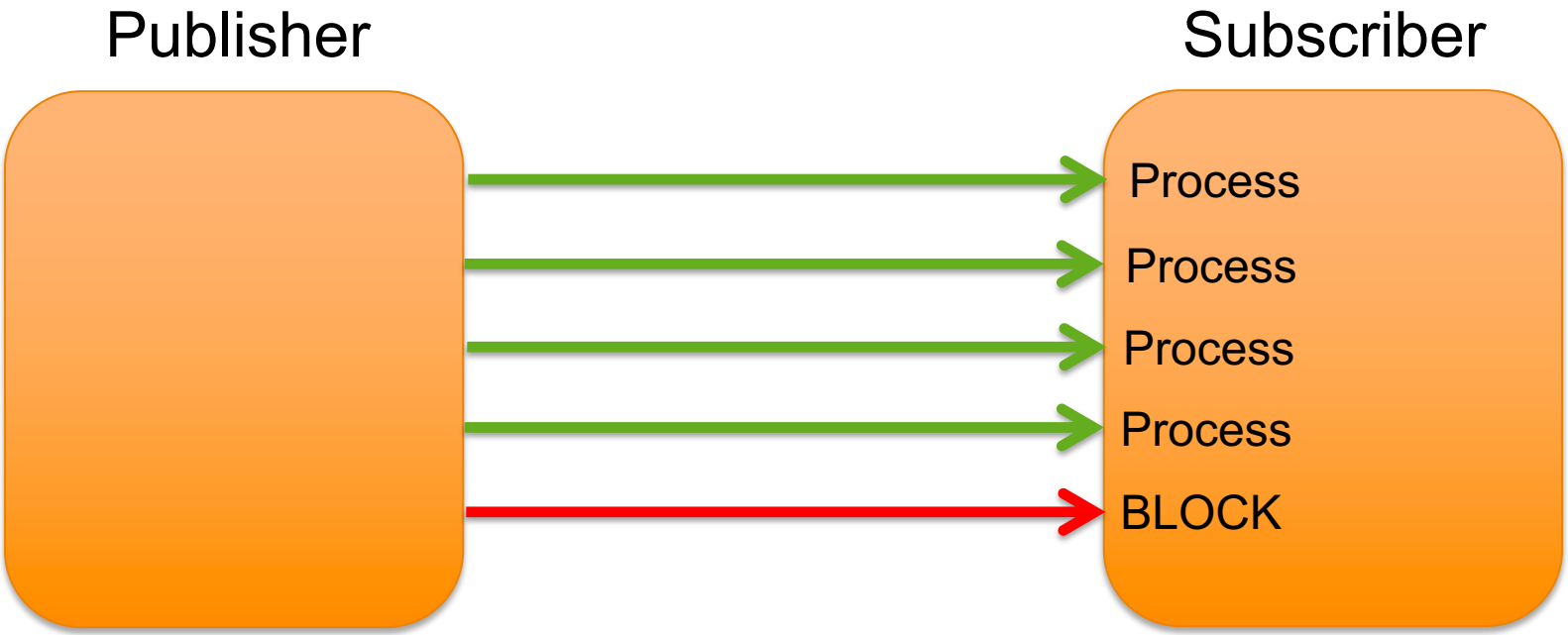
Spin-Wait Hints (JEP 285)

- Proposed by Azul
  - We rock!
- A new method for `Thread` class
  - `onSpinWait()`
- Enables the `x86 PAUSE` instruction to be used from Java code
  - If available
  - Ignored otherwise
  - Improved performance for things like Disruptor

Reactive Streams (JEP 266)

- Reactive streams publish-subscribe framework
- Asynchronous, non-blocking
- `Flow`
  - `Publisher`, `Subscriber`, `Processor`, `Subscription`
- `SubmissionPublisher` utility class
  - Asynchronously issues items to current subscribers
  - Implements `Flow.Processor`

Reactive Streams: The Problem



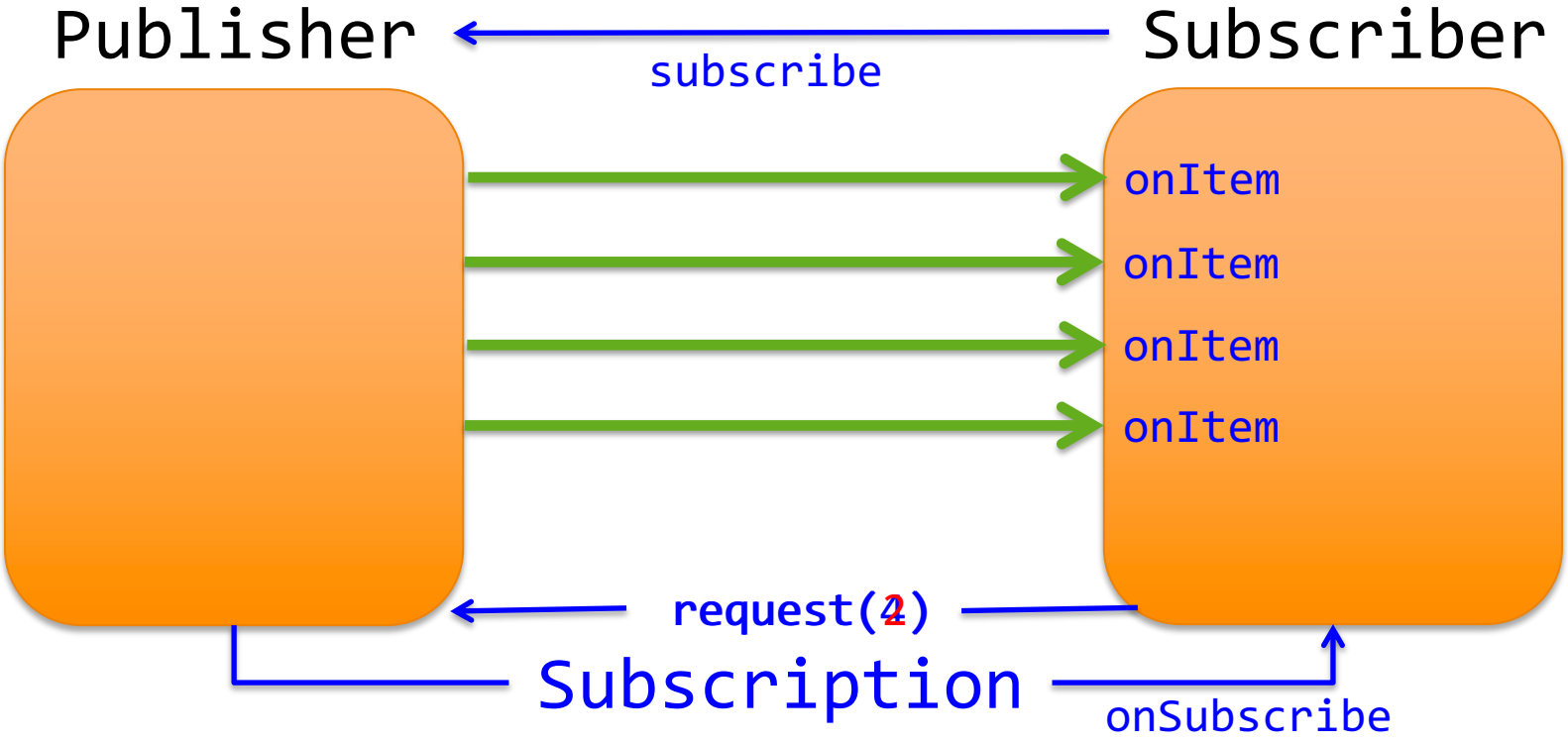
Reactive Streams: Flow API

- Set of interfaces
- `Publisher`
  - Producer of items to be received by `Subscribers`
- `Subscriber`
  - Receiver of messages
- `Processor`
  - Both a `Publisher` and `Subscriber` (chaining)
- `Subscription`
  - Message control linking a `Publisher` and `Subscriber`

Reactive Streams: Flow API

- Publisher
  - subscribe(Subscriber)
- Subscriber
  - OnSubscribe(Subscription)
  - onNext(T item)
  - onComplete() / onError(Throwable)
- Subscription
  - request(long)
  - cancel()

Solution



Concurrency Updates (JEP 266)

- `CompletableFuture` additions
  - Delays and timeouts
  - Better support for sub-classing
  - New static utility methods
    - `minimalCompletionStage`
    - `failedStage`
    - `newIncompleteFuture`
    - `failedFuture`

Variable Handles (JEP 193)

- Replacement for parts of `sun.misc.Unsafe`
- Fence operations
  - Fine grained memory control
  - Atomic operations on object fields and array elements
- `VarHandle`
  - `compareAndExchange()`, `compareAndSet()`
  - `getAndAdd()`, `getAndSet()`
  - `acquireFence()`, `releaseFence()`

JDK 9 Other Features

Enhanced Deprecation (JEP 277)

- We have `@deprecated` and `@Deprecated`
  - Used to cover too many situations
- New methods in `Deprecated` annotation
  - `boolean forRemoval()`
    - Will this ever be removed?
  - `String since()`
    - JDK Version when this was deprecated
- Several `@Deprecated` tags added
  - `java.awt.Component.{show(),hide()}` removed
- `jdeprscan` command to produce report

Default Collector: G1 (JEP 248)

- G1 now mature in development
- Designed as low-pause collector
- Concurrent class unloading (JEP 156) JDK8u40
  - Useful enhancement to improve G1
- Still falls back to full compacting collection
  - Pause times proportional to heap size
  - Use Zing from Azul for truly pauseless

Better String Performance

- Compact strings (JEP 254)
  - Improve the space efficiency of the `String` class
  - Not using alternative encodings
- Store interned strings in CDS archive (JEP 250)
  - Share `String` and `char[]` objects between JVMs
- Indify `String` concatenation (JEP 280)
  - Change from static `String`-concatenation bytecode sequence to `invokedynamic`
  - Allow future performance improvements

JIT Compiler Control (JEP 165)

- Control of C1/C2 JIT
  - Directive file
  - Runtime changes via `jcmd`

Compiler Directive Example

```
[ //Array of directives
  { //Directive Block
    //Directive 1
    match: ["java*.*","oracle*.*"],
    c1: { Enable: true, Exclude: true,
      BreakAtExecute: true, },
    c2: { Enable:  false, MaxNodeLimit: 1000, },
      BreakAtCompile: true, DumpReplay: true, },
  { //Directive  Block
    //Directive 2 match: ["*Concurrent.*"],
    c2: { Exclude:true, },
  },
]
```

AOT Compilation (JEP 295)

- Experimental feature in JDK 9
  - `-XX:+UnlockExperimentalVMOptions`
  - Only the java.base module has AOT version
- `jaotc` command
  - `jaotc --output libMyStuff.so MyStuff.jar`
- JVM uses AOT code to replace interpreted
  - Can recompile with C1 to collect further profiling data
  - Recompile with C2 for optimum performance

AOT Compilation

- Future use
- Project Metropolis
  - Rewrite the JVM in Java
  - Start with AOT code, profile and recompile with JIT

Migrating Applications To JDK 9

Migration Guidance From Oracle

"Clean applications that just depend on java.se *should* just work"

Java Platform Module System

- The core Java libraries are now a set of modules
  - 97 modules for JDK, 28 of which are Java SE
  - No more rt.jar or tools.jar files
- Most internal APIs are now encapsulated
  - `sun.misc.Unsafe`, etc.
  - Numerous libraries and frameworks use internal APIs
- Module path used to locate modules
  - Separate and distinct from classpath

Migrating Applications to JPMS

- Initially, leave everything on the classpath
- Anything on the classpath is in the unnamed module
  - All packages are exported
  - The unnamed module depends on all modules
- Migrate to modules as required
  - Try automatic modules
  - Move existing jar files from classpath to modulepath

Reversing Encapsulation

- "The Big Kill Switch"

```
--illegal-access
```

- Four options:
  - `permit` (current default)
  - `warn`
  - `debug`
  - `deny` (future default)

Reversing Encapsulation

- Big kill switch overrides encapsulation
  - From the unnamed module (i.e. classpath)
  - Allows unlimited refective access to named modules
    - Not from named modules
- Warning messages written to error stream
- Useful to understand use of `--add-exports` and `--add-opens` when migrating to named modules

Reversing Encapsulation

- Allowing direct access to encapsulated APIs
  - `--add-exports`

```
--add-exports java.management/com.sun.jmx.remote.internal=mytest
--add-exports java.management/sun.management=ALL-UNNAMED
```

- Allowing reflective access to encapsulated APIs
  - `--add-opens`

```
--add-opens java.base/java.util=ALL-UNNAMED
```

Reversing Encapsulation

- Using the JAR file manifest

```
Add-Exports: java.base/sun.security.provider
```

Finding Encapsulated API Use

- `jdeps`
  - Analyses dependencies on APIs
- Example: Minecraft

```
jdeps --list-deps 1.8.jar
   java.base
   java.datatransfer
   java.desktop
   java.management
   java.naming
   not found
   unnamed module: 1.8.jar
```

"Missing" Modules

- Remember, "Clean applications that only use java.se..."
- The `java.se.ee` module not included by default
  - Compilation and runtime
- Affected modules
  - `java.corba`
  - `java.transaction`
  - `java.activation`
  - `java.xml.bind`
  - `java.xml.ws`
  - `java.xml.ws.annotation`

Using "Missing" Modules

- Use the command line option
  - `--add-modules java.corba`
- All modules (except CORBA) have standalone versions
  - Maven central
  - Relevant JSR RI
- Deploy standalone version on the upgrade module path
  - `--upgrade-module-path <path>`
- Deploy standalone version on the classpath

Small Incompatibilities

Milling Project Coin

- A single underscore is now a keyword in Java

```
error: as of release 9, '_' is a keyword, and may not be used as
an identifier
```

- Fear not, two or more underscores can still be used

Deleted Deprecated Methods

- Classes
  - `java.util.jar.Pack200`
  - `java.util.jar.Unpack200`
  - `java.util.logging.LogManager`
- Methods
  - `addPropertyChangeListener()`
  - `removePropertyChangeListener()`

- Removal required for clean modularisation

Deleted Deprecated Class

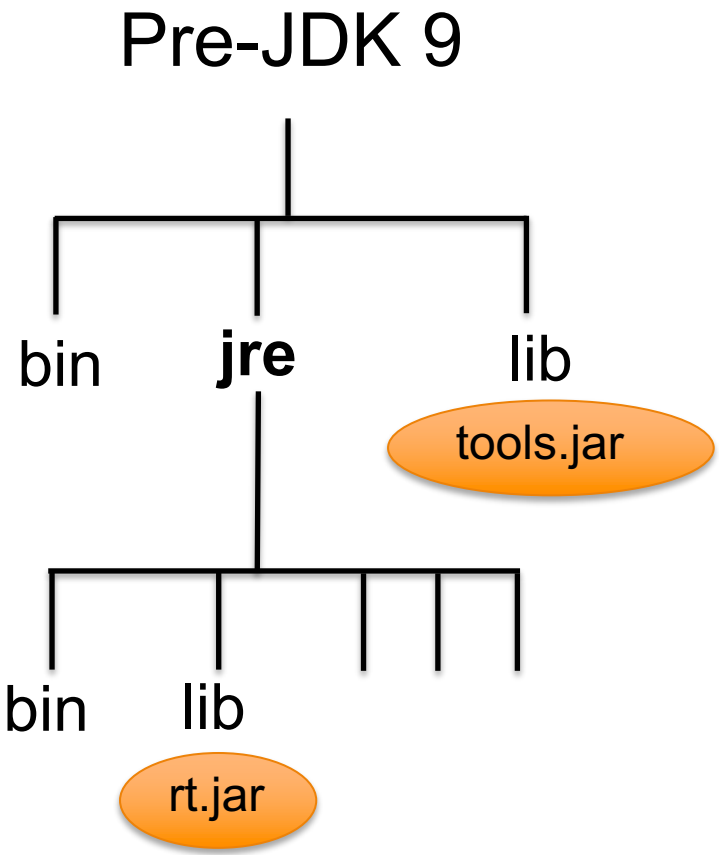
- `com.sun.security.auth.callback.DialogCallbackHandler`
- Part of the Java Authentication and Authorisation Service
  - JAAS
  - Deprecated in JDK 7

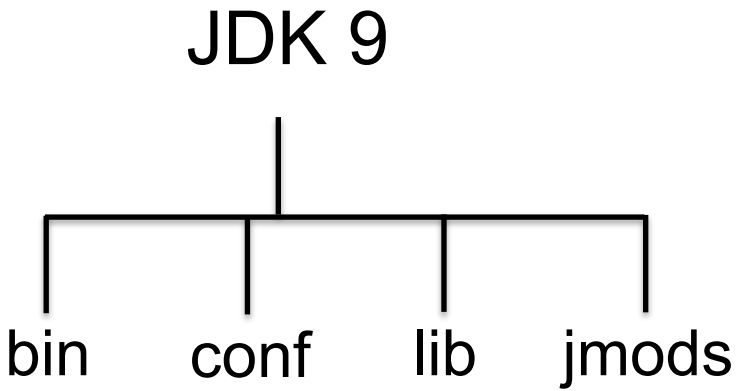
Finding Deprecated API Use

- `jdeprscan`
  - New tool in JDK 9
  - Statically analyses class files and jar files against Java SE APIs
  - Looks for and reports usage of deprecated Java SE APIs

```
$ jdeprscan --class-path classes example.Deprecations
class example/Deprecations uses type java/rmi/RMISecurityManager deprecated
class example/Deprecations uses method javax/swing/JList getSelectedValues ()[Ljava/
lang/Object; deprecated
class example/Deprecations uses method in type java/rmi/RMISecurityManager deprecated
class example/Deprecations uses method java/lang/Boolean <init> (Z)V deprecated
```

JDK/JRE File Structure (JEP 220)





~~jre directory~~
~~tools.jar~~
~~rt.jar~~

New Version String Format (JEP 223)

- Old
  - Limited update release/Critical patch update (CPU)
  - Download: Java SE 8u131, `java -version`: jdk1.8.0_131
  - Which has more patches, JDK 7u55 or JDK 7u60?
- New
  - `JDK $MAJOR.$MINOR.$SECURITY`
  - Easy to understand by humans and apps
  - Semantic versioning

Non-Programmatic Issues

- Java Network Launch Protocol (JNLP) [JSR 52]
  - Now uses strict parsing of configuration files
  - Some files that did parse may now fail
- Extension mechanism/Endorsed Standards Override mechanisms removed
  - Directories removed
    - `$JAVA_HOME/lib/ext`
    - `$JAVA_HOME/lib/endorsed`

```
<JAVA_HOME>/lib/ext exists, extensions mechanism no longer
supported; Use -classpath instead.
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
```

Removed GC Options (JEP 214)

- Deprecated in JDK 8 (JEP 173)

```
DefNew + CMS        : -XX:-UseParNewGC -XX:+UseConcMarkSweepGC
ParNew + SerialOld  : -XX:+UseParNewGC
ParNew + iCMS       : -Xincgc
ParNew + iCMS       : -XX:+CMSIncrementalMode -XX:+UseConcMarkSweepGC
DefNew + iCMS       : -XX:+CMSIncrementalMode -XX:+UseConcMarkSweepGC
                      -XX:-UseParNewGC
CMS foreground      : -XX:+UseCMSCompactAtFullCollection
CMS foreground      : -XX:+CMSFullGCsBeforeCompaction
CMS foreground      : -XX:+UseCMSCollectionPassing
```

JVM Logging

- Unified JVM logging (JEP 158)
  - Common logging system for all components of JVM
- Unified GC logging (JEP 271)
  - Re-implement GC logging using unified JVM logging
  - Many command line options changed

Removed JVM Flags: Ignored

- AdaptiveSizePausePolicy
- CodeCacheMinimumFreeSpace
- DefaultThreadPriority
- JNIDetachReleasesMonitors
- LazyBootClassLoader
- NmethodSweepCheckInterval
- NmethodSweepFraction
- PrintOopAddress
- ReflectionWrapResolutionErrors
- StarvationMonitorInterval
- ThreadSafetyMargin
- UseAltSigs
- UseBoundThreads
- UseCompilerSafepoints
- UseFastAccessorMethods
- UseFastEmptyMethods
- BackEdgeThreshold
- PreInflateSpin

```
Java HotSpot(TM) 64-Bit Server VM warning: Ignoring option
<Option>; support was removed in 9.0
```

Deprecated JVM Flags

JDK 8 Option	JDK 9 Replacement
PrintGC	-Xlog:gc
PrintGCDetails	-Xlog:gc*
CreateMinidumpOnCrash	CreateCoredumpOnCrash (suggestion)
DefaultMaxRAMFraction	MaxRAMFraction (suggestion)
TraceBiasedLocking	-Xlog:biasedlocking=info
TraceClassLoadingPreorder	-Xlog:class+preorder=debug
TraceClassResolution	-Xlog:class+resolve=debug
TraceMonitorInflation	-Xlog:monitorinflation=debug
TraceRedfineClasses	-Xlog:redefine+class*=info
TraceSafepointCleanupTime	-Xlog:safepoint+cleanup=info
TraceClassLoading	-Xlog:class+load=info
TraceClassUnloading	-Xlog:class+unload=info
TraceLoaderConstraints	-Xlog:class+loader+constraints=info
ConvertSleepToYield	NONE

Deprecated JVM Flags

- Two forms of warning message

```
warning[gc] -XX:+PrintGC is deprecated. Will use -Xlog:gc instead.

Java HotSpot(TM) 64-Bit Server VM warning: Option
CreateMinidumpOnCrash was deprecated in version 9.0 and will likely
be removed in a future release. Use option CreateCoredumpOnCrash
instead.
```

JVM Flags: Non-Starters (1)

- AdjustConcurrency
- CMSCompactWhenClearAllSoftRefs
- CMSDumpAtPromotionFailure
- CMSFullGCsBeforeCompaction
- CMSIncrementalDutyCycle
- CMSIncrementalDutyCycleMin
- CMSIncrementalMode
- CMSIncrementalOffset
- CMSIncrementalPacing
- CMSParPromoteBlocksToClaim
- CMSPrintEdenSurvivorChunks
- CollectGen0First
- GCLogFileSize
- NumberOfGCLogFiles
- ParallelGCVerbose
- PrintAdaptiveSizePolicy
- PrintCMSInitiationStatistics
- PrintCMSStatistics
- PrintClassHistogramAfterFullGC
- PrintClassHistogramBeforeFullGC
- PrintFLSCensus
- PrintFLSStatistics
- PrintGCApplicationConcurrentTime
- PrintGCApplicationStoppedTime
- PrintGCCause
- PrintGCDateStamps

JVM Flags: Non-Starters (2)

- PrintGCTaskTimeStamps
- PrintGCTimeStamps
- PrintHeapAtGC
- PrintHeapAtGCExtended
- PrintJNIGCStalls
- PrintOldPLAB
- PrintPLAB
- PrintParallelOldGCPhaseTimes
- PrintPromotionFailure
- PrintReferenceGC
- PrintTLAB
- PrintTenuringDistribution
- TraceDynamicGCThreads
- TraceGen0Time
- TraceGen1Time
- TraceMetadataHumongousAllocation
- TraceParallelOldGCTasks
- UseCMSCollectionPassing
- UseCMSCompactAtFullCollection
- UseGCLogFileRotation
- UseMemSetInBOT
- UsePPCLWSYNC
- UseVMInterruptibleIO
- WorkAroundNPTLTimedWaitHang

JVM Flags: Non-Starters

- 50 command line flags from JDK 8
- Use will cause the JVM to abort at start
  - It won't run your application

```
Unrecognized VM option '<Option>'
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
```

Real World Example

HdrHistogram Library

- Open source library (created by Gil Tene, Azul CTO)
- *Ideally* this should work with JDK 6, 7, 8 and 9 code
  - Unchanged and with no special command line flags
- Problem:
  - Needs to encode and decode using Base64
- Solutions:
  - `sun.misc.BASE64Encoder/Decoder` (JDK 6, 7, 8)
  - `javax.xml.bind.DatatypeConverter` (JDK 6, 7, 8)
  - `java.util.Base64.{Encoder,Decoder}` (JDK 8, 9)

Solution

- Helper class with same named methods as from `xml.bind`
  - `printBase64Binary(byte[] array)`
  - `parseBase64Binary(String input)`
- Static code to determine which class is available
  - Initialise `Method` object reference
- Helper methods invoke through method reference to available implementation

Solution (1)

- Code in static block

```
try {
  Class<?> javaUtilClass = Class.forName("java.util.Base64");

  Method getDecoderMethod =  javaUtilClass.getMethod("getDecoder");
  decoderObj = getDecoderMethod.invoke(null);
  decodeMethod = decoderObj.getClass().getMethod("decode", String.class);
  Method getEncoderMethod =  javaUtilClass.getMethod("getEncoder");
  encoderObj = getEncoderMethod.invoke(null);
  encodeMethod = encoderObj.getClass()
    .getMethod("encodeToString", byte[].class);
} catch (Throwable e) {  // ClassNotFoundException, NoSuchMethodException
  decodeMethod = null;
  encodeMethod = null;
}
```

Solution (2)

```
if (encodeMethod == null) {
  decoderObj = null;
  encoderObj = null;

  try {
    Class<?> xmlBindClass = Class.forName("javax.xml.bind.DatatypeConverter");
    decodeMethod = xmlBindClass.getMethod("parseBase64Binary", String.class);
    encodeMethod = xmlBindClass.getMethod("printBase64Binary", byte[].class);
  } catch (Throwable e) {
    decodeMethod = null;
    encodeMethod = null;
  }
}
```

Summary

Summary

- JDK 9 has many changes
- JPMS is the big one
  - Encapsulation of internal APIs
- Smaller developer features
  - Stream enhancements, jshell, reactive streams
- Many changes to the platform
  - JVM option changes, better string performance
- Ignore JDK 9 (and probably JDK 10) for deployment
  - But not the features

Zulu Java

- Azul's binary distribution of OpenJDK
  - Passes all TCK tests
  - Multi-platform (Windows, Linux, Mac)
  - Free, with paid support options
  - No licensing issues
  - Verified as built from 100% open-source
- JDK 6, 7, 8 and 9

www.zulu.org/download

Thank you

Simon Ritter
Deputy CTO, Azul Systems

@speakjava