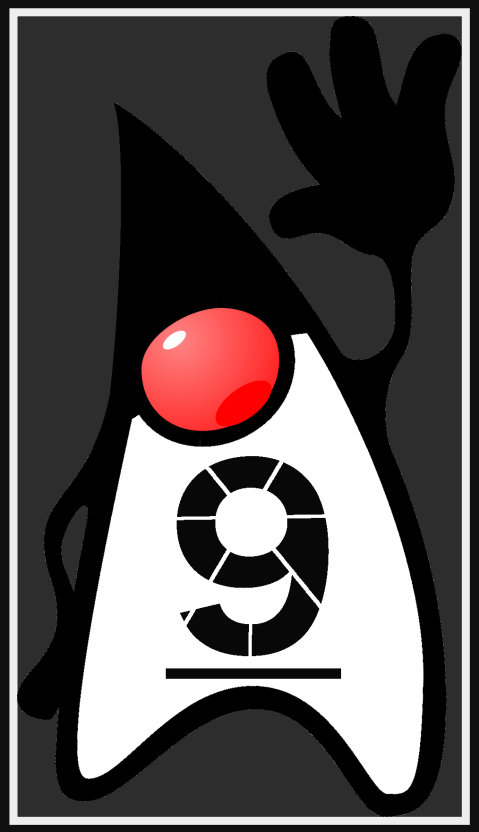
JAVA 9

SOME NEW FEATURES



JAVA 9 INTRODUCTION

Java 9 was released on 2017-09-21, with a patch 9.0.1 released on 2017-10-17.

Biggest upgrade to Java platform standard edition, coming after 3 and a half years. The last update was Java 8, released in March 2014.

Java 9 introduces over 150 new features and functionalities.

DEFINITELY NOT POSSIBLE TO COVER ALL OF THEM, HERE.

LANGUAGE

LANGUAGE CHANGES

The next few slides cover language changes.

LANGUAGE

PRIVATE METHODS IN INTERFACES

- Aimed to allow sharing of common code between `default` and `static` methods.
- `private abstract` and `private default` method signatures result in compile-time errors.

Java support in interfaces

	Java 7	Java 8	Java 9
constants			
abstract methods			
default methods			
public static methods			
private methods			
private static methods			

- Links:
  - Original Bug/Feature Request: https://bugs.openjdk.java.net/browse/JDK-8071453
  - Writeup: https://cguntur.me/2017/09/02/java-9-features-private-interface-methods/

LANGUAGE

COLLECTIONS - FACTORY METHODS

- Factory methods on collections allow ease of declaration/instantiation.
- Factory methods clean up existing verbose means of construction:
  - Most developers first instantiate a collection
  - Then, they invoke `add(...)` or `put(...)` methods.
  - Alternatively, some use `Collections` factory methods to initialize from an inline `Array`.

- `List.of("one", "two");`
- `Map.of("key1", "val1", "key2, "val2");`

- Links:
  - JEP 269 - Convenience Collection Factory Methods: http://openjdk.java.net/jeps/269
  - Collections re-fuelled: https://blogs.oracle.com/java/collections-refueled

LANGUAGE

OPTIONAL - ENHANCEMENTS

- `ifPresentOrElse()` – Checks the presence of a non-null value, else, invokes the Runnable.
- `or()` – Always return an optional.
  - Wrap and return an Optional of the current value if not null.
  - If current value is null, return Optional by invoking the specified Supplier.
- `stream()` – Returns a sequential stream containing only the value, if the value is non-null.

- Links:
  - `java.util.Optional` API: https://docs.oracle.com/javase/9/docs/api/java/util/Optional.html
  - Writeup: https://cguntur.me/2017/09/04/java-9-features-changes-to-optional/

LANGUAGE

STREAM - ENHANCEMENTS

- `takeWhile()` and `dropWhile()` to proces a stream until a condition is met.
- `ofNullable()` returns a stream of 0 or 1 elements based on the value being null.
- `iterate()` overloaded to a 3 param method similar to an old-fashioned `for` loop.

- Links:
  - `java.util.stream.Stream` API: https://docs.oracle.com/javase/9/docs/api/java/util/stream/Stream.html

LANGUAGE

CONCURRENCY - ENHANCEMENTS

- Support for a reactive stream Pub-Sub framework introduced.
  - Fully complies with the reactive streams manifesto. (http://www.reactive-streams.org/):
    1. process a potentially unbounded number of elements
    2. in sequence
    3. asynchronously passing elements between components
    4. with mandatory non-blocking backpressure.
  - Requires a `Publisher`, `Subscription`, `Subscriber` and a `Processor`.
  - No N/W- or I/O-based `java.util.concurrent` components for distributed messaging, in JDK9.
- Enhancements to the `CompletableFuture` API introduced.
  - Time-based enhancements enable a future to complete with a value or exceptionally after a certain duration.
  - Subclass enhancements make it easier to extend from `CompletableFuture`.
- Links:
  - JEP 266 - Concurrency Enhancements: http://openjdk.java.net/jeps/266
  - `java.uril.concurrent.Flow` API: https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.html
  - Oracle docs on Reactive Streams: https://community.oracle.com/docs/DOC-1006738

LANGUAGE

DEPRECATION - ENHANCEMENTS

- History:
  - Deprecation was a javadoc feature via `@deprecated` annotation in the comment.
  - The javadoc comment was limited in visibility since it was in documentation.
  - New **code** annotation `@Deprecated` introduced in Java 5.

- The annotation has two (optional) parameters since Java 9:
  - `forRemoval`: Boolean indicating a hint to remove from future releases, if true.
  - `since`: String representing version when deprecation was set.
- New deprecation parameters clarify if an API may be removed in the near future.
- New `jdeprscan` scans jars or any other aggregation of classes deprecated API usage.

- Links:
  - JEP 277 - Enhanced Deprecation: http://openjdk.java.net/jeps/277
  - Oracle docs on Deprecation: https://docs.oracle.com/javase/9/core/enhanced-deprecation1.htm
  - Oracle docs on `jdeprscan`: https://docs.oracle.com/javase/9/tools/jdeprscan.htm

LANGUAGE

NEW STACK-WALKING API

- Changed from `StackTraceElement` to `StackWalker` instance.
- `StackTraceElement` required an all-or-nothing fetch, and was expensive.

- `StackWalker` is thread-safe, filter-enabled and skip-enabled.
- Stack-walking API enables lazy access and easy filtering in stack traces.
- Supports both long and short walks.
  - **long walks** - traverse the entire stack.
  - **short walks** - filter to - *or* - skip to specific frame.
- The short walks reduce the cost of analyzing all frames.
  - Caller has an option now to explore only the top frames.

- Links:
  - JEP 259 - Stack-Walking: http://openjdk.java.net/jeps/259
  - `java.lang.StackWalker` API: https://docs.oracle.com/javase/9/docs/api/java/lang/StackWalker.html

LANGUAGE

COMPACT STRINGS

- Initiated in Java 6 as "compressed strings", deprecated in Java 7 and removed in Java 8.
  - Flag needed to be enabled (default was off), see the + below.
  - Old flag: `-XX:+UseCompressedStrings`
  - *"experimental feature, ... ultimately limited by design, error-prone, ... hard to maintain"* - A. Shipilev.
  - Pointed to a `byte[]` *if* string only had 7-bit ASCII *else* to `char[]` .
  - String operations depended on `char[]` for operations, hence was non-optimal.

- Replaced above with **Compact Strings**.
  - Every string now a `byte[]` representation with an **encoding field** flag.
  - Encoding field can determine if conversion from `byte[]` to `char[]` is needed.
  - New flag is **enabled by default**.
  - `-XX:-CompactStrings` to disable.

- Links:
  - JEP 254 - Compact Strings: http://openjdk.java.net/jeps/254

LANGUAGE

FEW OTHER LANGUAGE ENHANCEMENTS

There are way too many enhancements to list !

- JEP 213 - Milling Project Coin: http://openjdk.java.net/jeps/213:
  - Allows `@SafeVarargs` on private instance methods (typo in JEP, there is no `@SafeVargs`).
  - Allows final variables as a resource in try-with-resource statement.
  - Allows diamond operator with anonymous classes if the argument type is denotable.

- JEP 197 - Segmented Code Cache: http://openjdk.java.net/jeps/197:
  Segemented code-cache for profiled, non-profiled and non-method code.
  - **Non-method code** heap cache (compiler buffers, bytecode interprter etc.)- **long lifetime cache**.
  - **Non-profiled** or more static code (fully optimized code) - **long lifetime cache**.
  - **Profiled** code (lightly optimized) - **short lifetime cache**.

- Process API improvements - https://docs.oracle.com/javase/9/core/process-api1.htm

- Comprehensive list of JDK 9 features.
  - http://openjdk.java.net/projects/jdk9/

ASSORTED CHANGES

UNGROUPED IMPORTANT FEATURES

The next few slides cover a few important features added to Java 9.

GARBAGE COLLECTION

NEW DEFAULT GC - G1GC

Java 9 makes Garbage-first Garbage Collector (G1GC) as the default.

- Originally introduced in Java 7 and targeted for Java 8.
- No more PermGen !!!
- JVM now uses a metaspace (curtailing usage of available memory).
- Heap divided into ~2048 equal sized **regions**.
- Regions get marked as Eden, Survivor or Old as needed.
- GC focusses on garbage-heavy regions == STW (Stop-The-World) cycles are shorter.
- With very minor or no tweaks, ** most ** code will see performance gains.

- Links:
  - JEP 248 - Make G1 the default collector: http://openjdk.java.net/jeps/248.
  - Oracle docs on G1GC: https://docs.oracle.com/javase/9/gctuning/garbage-first-garbage-collector.htm

VERSIONS

VERSION STRING SCHEME

New version scheme for JDK and JRE releases.

Caveat Emptor: Version strings in- and post-Java 9 may vary!

- Version string scheme in Java 9 is **$MAJOR.$MINOR.$SECURITY_PATCH.**
- Replaces the confusing collision of version and build numbers in prior versions.
- This change may be short-lived due to a newer proposal from Oracle.
- Proposal for Long-Term-Support (LTS) versioning based on **$YEAR.$MONTH**.
- Implies a release every six months.
- Java 10 may be known as Java 18.3 or Java 18.9 (March or September release).

- Links:
  - JEP 223 - Version-String Scheme: http://openjdk.java.net/jeps/223
  - LTS Versioning Proposal: https://mreinhold.org/blog/forward-faster
  - Plea against LTS: http://mail.openjdk.java.net/pipermail/discuss/2017-September/004352.html

COMPILER ENHANCEMENTS

Compile For Previous Platforms.

- A **--release** option added as a new compiler directive.
  - Was not sufficient to just set the -source and -target options to the older value.
  - The bootclasspath had to also be set to correspond to the older release.
  - Forgetting the bootclasspath may have resulted in use of unsupported APIs on the target.
  - The new --release option will prevent any random use of unsupported API.
  - Single flag (`--release`) to cross compile.
  - Supported releases follow the "one plus three back" policy same as prior versions.
  - Supported release targets are 6, 7, 8, and 9.

- Restrictions apply when using the --release for module support (JDK 9, for instance).

- Links:
  - JEP 247 - Compile for Older Platforms: http://openjdk.java.net/jeps/247
  - Original Bug: https://bugs.openjdk.java.net/browse/JDK-8058150

LANGUAGE

FEW OTHER ENHANCEMENTS

New compiler control to issue compiler directives. New TLS, DTLS Support.

- New Compiler Control supersedes and is backward compatible with CompileCommand.
  - Compiler Control is useful for creating workarounds for bugs in the JVM compilers.
  - JEP 165 - Compiler Control: http://openjdk.java.net/jeps/165
  - Documentation: https://docs.oracle.com/javase/9/vm/compiler-control1.htm

- New **DTLS** (Datagram Transport Layer Security) “TLS over Datagram” protocol support.
  - Oracle docs on JSSE: https://docs.oracle.com/javase/9/security/java-secure-socket-extension-jsse-reference-guide.htm
  - Oracle docs on Security: https://docs.oracle.com/javase/9/security/toc.htm

MODULARITY

MODULARITY - PROJECT JIGSAW

The next few slides cover the delivery of project Jigsaw.
Link: http://openjdk.java.net/projects/jigsaw/

MODULARITY

JAVA PLAFORM MODULE SYSTEM

Java 9 introduces a module system with its distribution specified as Java Platform Module System.

- Prior dependency image: https://bugs.openjdk.java.net/secure/attachment/72525/jdk.png
- Aims to reduce the large and growing size of the java package ecosystem.
- Allows splitting the JDK in smaller units (modules).
- Root module called `java.base`.
- Leads to subsequent deprecation/removal of some vestigial packages.
- Dependencies packaged as `.jmod` files.
- New tools such as `jlink` to create custom runtime bundles.
- Module path to replace classpath.

- Links:
  - Modules and Javac: http://openjdk.java.net/projects/jigsaw/doc/ModulesAndJavac.pdf
  - State of the Module System (slightly out-of-date): http://openjdk.java.net/projects/jigsaw/spec/sotms/

JDK 9 MODULARITY

Java 9 itself is a modular system

- Restructures the JDK and JRE runtime images - as modules.
- New images improve performance, security, and maintainability.
- Modularity allows creation of custom configurations.
  - (see Compact Profiles: http://openjdk.java.net/jeps/161).
- New URI scheme for naming modules, classes/resources stored in a runtime image.
- No more `rt.jar` and `tools.jar` in libs.

- Links:
  - JPMS (JSR 376): http://openjdk.java.net/projects/jigsaw/spec/
  - JEP 261 - Module System: http://openjdk.java.net/jeps/261
  - JEP 200 - The Modular JDK: http://openjdk.java.net/jeps/200
  - JEP 220 - Modular Run-Time Images: http://openjdk.java.net/jeps/220
  - JEP 260 - Encapsulate Most Internal APIs: http://openjdk.java.net/jeps/260

MODULARITY

JAVA LINKER: JLINK

- `jlink` is responsible for assembling and optimizing modules.
- `jlink` also links transitive dependencies.
- Is a descendant of `jrecreate` used to build EJDKs.
- It produces a custom runtime image that reduces the size and complexity of the deployment.
- Java always had dynamic linking, jlink introduces *optional* static linking.
- This optional static linking phase is called **link time** that occurs between compile and run time.

- Links:
  - JEP 282 - jlink - The Java Linker: http://openjdk.java.net/jeps/282
  - Oracle docs on jlink: https://docs.oracle.com/javase/9/tools/jlink.htm

TOOLING

TOOLING CHANGES

The next few slides cover tooling changes.

TOOLING

JSHELL

A read-eval-print-loop (REPL) for a transcript evaluation on a command prompt.

- Originally targeted for Java 7.
- Accepts "snippets" of Java code, evaluates them and immediately displays the results.
- Built on JShell API, evaluation of snippets of Java code available to any Java program.
- In-depth coverage: http://cr.openjdk.java.net/~rfield/tutorial/JShellTutorial.html

- Links:
  - JEP 222 - jshell - The Java Shell: http://openjdk.java.net/jeps/222.
  - Oracle docs on jshell: https://docs.oracle.com/javase/9/tools/jshell.htm

TOOLING

ENHANCED JAVADOC

Upgraded the user experience in API documentation.

- Addition of a search for quick access - http://openjdk.java.net/jeps/225.
- New Doclet API, to meet and use existing standards - http://openjdk.java.net/jeps/221.
- HTML5 output for Javadoc - http://openjdk.java.net/jeps/224.

TOOLING

BETTER JAVA CONTROL PANEL (ORACLE JDK)

Upgraded the user experience in the control panel.

- Improved presentation and grouping of options in the control panel.
- Information easier to locate, no modal dialog boxes, search field is available.
- Less useful APIs deprecated - web browsers removed Java browser plugin support.

- Links:
  - Oracle docs on the Control Panel: https://docs.oracle.com/javase/9/deploy/java-control-panel.htm

CHALLENGES

MIGRATION CHALLENGES

The next few slides list challenges adopting JDK 9 due to deprecation or fixing of prior "*features*".

Migrating is non-trivial if you use reflection or private internal APIs (e.g. `sun.misc packages`).

Other challenges could usage of experimental features in prior JDKs.

Module System Risks and Assumptions is a good read:
http://openjdk.java.net/jeps/261#Risks-and-Assumptions

Excellent resource that helped understand these challenges is from Nicolai Parlog's blog:
https://blog.codefx.org/java/java-9-migration-guide/

CHALLENGES

ILLEGAL ACCESS TO INTERNAL APIS

When you see the dreaded:

```
WARNING: An illegal reflective access operation has occurred
```

- Most obvious fix is to stop depending on internal APIs.
- If depending cannot be avoided, at least acknowledge and set a flag to:
  - `--add-exports $module/$package=$mymodule`
    This will allow all public types of the module to be accessed by mymodule.
  - `--add-opens $module/$package=$mymodule`
    This will allow all public (and otherwise, *deep reflection*) types of the module to be accessed by mymodule.

- Use `jdeps` to track down dependencies on internal APIs.
- Special search is needed for reflective access: `--illegal-access=$parameter`
  - `--illegal-access=permit` default for JDK 9. Access to all unnamed modules.
  - `--illegal-access=warn` warning message for each illegal reflective-access operation.
  - `--illegal-access=debug` warning and stack trace for each illegal reflective-access operation.
  - `--illegal-access=deny` disable all illegal reflective-access operations except for those enabled by above command-line options. Future default.

CHALLENGES

JAVA EE MODULE SEPARATION

- Java EE modules are not included in the module path.
  - `java.activation` with `javax.activation` package.
  - `java.corba` with `javax.activity`, `javax.rmi`, `javax.rmi.CORBA`, and `org.omg.*` packages.
  - `java.transaction` with `javax.transaction` package.
  - `java.xml.bind` with all `javax.xml.bind.*` packages.
  - `java.xml.ws` with `javax.jws`, `javax.jws.soap`, `javax.xml.soap` and all `javax.xml.ws.*` packages.
  - `java.xml.ws.annotation` with `javax.annotation` package.
- Work towards declaring a regular dependency in the module's declaration.
- Until you have that, `--add-modules $module` is the tactical fix.
  - Either add individual modules as you need them.
  - Or add `java.se.ee` to get access to all Java EE modules.

CHALLENGES

SPLIT PACKAGE HANDLING

- A module is not allowed to read the same dependency package from more than one module.
- In fact, no two modules are allowed to contain same package (think duplicate dependencies).
- The module system complains if it finds split packages. Two pathways exist here:
  - Rather than a few classes, the code needs a replacement for an *entire existing* module.
    - Use the `--upgrade-module-path $dir` to replace the module.
    - See endorsed standards for upgrades: https://docs.oracle.com/javase/8/docs/technotes/guides/standards/
  - Tactical fix for a few classes, is to ***patch*** the artifact classes into current module.
    - Use the `--patch-module $module=$artifact` to patch the module.
    - An example is the `@Nonnull` (JSR 305) and `java.xml.ws.annotation` module.
    - Use `--patch-module java.xml.ws.annotation=path/to/jsr305-3.0.2.jar`.
    - As a strategic fix, determine how to remove such split.

CHALLENGES

DEPRECATION OF INTERNAL API

- Divided into two broad categories:
  - **Non-critical internal APIs**: Functionality available either in public JDK or via third-party libs.
  - **Critical internal APIs**: Functionality cannot be available outside of the JDK.
- Critical APIs with replacements in JDK9 are deprecated in JDK9.
  - Such replaced API will either be encapsulated or removed in a future version of JDK.

- Marked for deprecation (with no replacements in Java 9):
  - `sun.misc.{Signal, SignalHolder}`
  - `sun.misc.Unsafe` - Some replacements via *variable handles* - http://openjdk.java.net/jeps/193
  - `sun.reflect.Reflection::getCallerClass(int)`
    Some functionality available in StackWalking API. http://openjdk.java.net/jeps/259
  - `sun.reflect.ReflectionFactory`
  - Few classes in `com.sun.nio.file`

- Links:
  - JEP 260 - Encapsulate Most Internal APIs: http://openjdk.java.net/jeps/260
  - Oracle docs on jshell: https://docs.oracle.com/javase/9/tools/jshell.htm

CHALLENGES

NEW VERSION STRINGS

- Several tools both open source and bespoke depend on java versions.
- More than likely one of the following properties is used to check versions:
  - `java.version`
  - `java.vm.version`
  - `java.runtime.version`
  - `java.specification.version`
  - `java.vm.specification.version`

- Replace above with the new `Runtime.Version` class.
  - https://docs.oracle.com/javase/9/docs/api/java/lang/Runtime.Version.html
- Compatibility to older versions of java can be retained by aiming to build a multi-release jar.

- Links:
  - JEP 238 - Multi-release Jar: http://openjdk.java.net/jeps/238
  - Another excellent blog from Nicolai Parlog: https://www.sitepoint.com/inside-java-9-part-i/#multireleasejars

Java 9
That's all we have time for!
HAPPY CODING !