# What's New in Java 9

It's more than just another version

## So, what is new?

- "THE" list
- Drill-down on selected features
- Demo

## Key Changes

- Simplified Version-String Schema
    - Now, with security added:
        - MAJOR . MINOR . SECURITY . PATCH
    - A long time coming, no more 1.x
- Java Platform Module System
    - A new kind of programming component
    - A new kind of JRE/JDK (It's modular)

## New tools, improved tools

- JShell
  - Java goes REPL
  - Of course, there's a demo
- jlink
  - Assemble module sets into custom runtime images
- jar
  - Create multiple Java-version-specific class files in a single JAR
- jcmd
  - New commands to print class and method info, and UTF8 strings
- java and javac
  - Validation of numeric JVM command-line flags such as memory settings
  - --release to avoid accidental use of APIs (enhances -source and -target)

## JVM performance

- Garbage First (G1) is default GC
  - Optimized for reduced latency over higher throughput
- Deprecated Concurrent Mark and Sweep GC
  - Replaced by G1
- Removes Java 8 garbage collection combinations
  - DefNew + CMS, ParNew + SerialOld, and Incremental CMS
- Unified JVM logging for all components including GC

## Core libraries goodness

- Process API offers better control of Process IDs
- Compacted Strings now use byte arrays instead of char arrays
- Added XML Catelogs
- Convenience factories for collections: List.of(), Map.of(), Set.of()
- Variable Handles replace Unsafe memory ordering fencing
- Enhanced `@Deprecation`, now with `forRemoval` and `since` flags
- Spin-Wait hints
- Stack Walking API: easy filtering and lazy access of stack traces

## There's always a misc

- New security features
- Small language improvements, private interface methods
- JavaDocs module aware, simplified Doclet API
- Java 9 installer enhancements
- Nashorn supports EMCAScript 6 (selected features)
- Client-side: Multi-resolution images, HiDPI on Windows and Linux, GTK 3
- Supports Unicode 8.0, internationalization tweaks for XML and property files

## Modules

- What is modularity?
  - Managing and reducing complexity, especially at scale
    - Millions of lines of code
    - Dependencies across several dozen shared libraries
  - Strong encapsulation / Well-defined Interfaces / Explicit dependencies
- Pre-Java 9 Modularity
  - JAR: grouping classes
    - Public gone wild
    - No explicit dependencies, you learn you made a mistake at runtime
  - Classpath
    - Destroys the grouping of JARs, all classes in the same flat list
    - No explicit version control, first loaded is the winner

## Modular JRE/JDK

- Time for an upgrade
    - Twenty year old, gigantic, monolithic runtime
    - Everything is present regardless of need. (When was the last time you used AWT, or CORBA?)
    - Unencapsulated internal APIs (`sun.*` package, `Unsafe()`)
- Java 8: introduced compact profiles. A start, but ….
- Java 9: Project Jigsaw
    - Runtime source reorganized to support modularization
    - 90+ modules with clearly defined interfaces and dependencies beginning with `java.base`
    - Encapsulated internal APIs
    - Custom runtime images (Think small, and IoT.)

# Modularity building block

- **`module-folder/`**
  ```
      module classes and resources
      module-info.java
          module module.name {
              exports  package [to target-package];
              requires [transitive] module.name;
          }
  ```
- Strong encapsulation: All packages in a module are private to the module
- Exported interface: Packages must be explicitly exported to be public
- Declared dependencies: Specifies required modules
- It's all about ***readability***

## Goodbye! classpath. Hello! module path.

- The classpath has been replaced with the module path.
- Compiler ***and*** runtime use the module path to resolve exports and requires
- "Computing the transitive closure of the dependency graph"* (a.k.a. module resolution)
  1. Start with a single root module, add to the dependency graph
  2. Add each new, non-duplicating *`requires`* module to the dependency graph
  3. Repeat Step 2 for each module added in Step 2
- Old code in the land of modules: a pre-migration story
  - The classpath hasn't totally disappeared, just ignored unless needed
  - Any classes on the classpath are loaded into the unnamed module
  - The unnamed module automatically reads all other modules

* Java 9 Modularity, Bakker and Mak

## Demo

## References

- Official feature list from Oracle
  - https://docs.oracle.com/javase/9/migrate/toc.htm#JSMIG-GUID-7744EF96-5899-4FB2-B34E-86D49B2E89B6
- Java 9 Modularity, 1st ed.
  - Sander Mak, Paul Bakker, O'Reilly Media, Inc., 2017.
- Java 9 for Programmers, 4th ed.
  - Paul Deitel, Harvey Deitel, Prentice Hall, 2017.
- Modular Programming in Java 9
  - Koushik Kothagal, Packt Publishing, 2017.
- Java 9 with JShell
  - Gastón C. Hillar, Packt Publishing, 2017.