

WHAT'S NEW IN JAVA 9

BILLY KORANDO @BILLYKORANDO

SOFTWARE CONSULTANT - KEYHOLE SOFTWARE

<http://bit.ly/2woEb1S>

JAVA 9 IS IN DEVELOPMENT



A close-up photograph of a white cat with dark grey or black stripes on its ears and face. The cat is looking slightly to the right. It is positioned behind a white, textured curtain, with a brown surface visible to the right.

SOON.

SEPTEMBER 21ST

<http://bit.ly/2woEb1S>

MISCONCEPTIONS ABOUT JAVA 9

~~1. MAVEN WILL NOT WORK~~

~~2. I'LL HAVE TO MODULARIZE MY CODE~~

~~3. MY LIBRARIES WILL NOT WORK *~~

SOME LIBRARIES WILL NEED TO
UPGRADED

PROVE IT!

<http://bit.ly/2woEb1S>

AN UPGRADE WITH BENEFITS

<http://bit.ly/2woEb1S>

BENEFITS OF RUNNING IN JAVA 9

Better Use of Memory

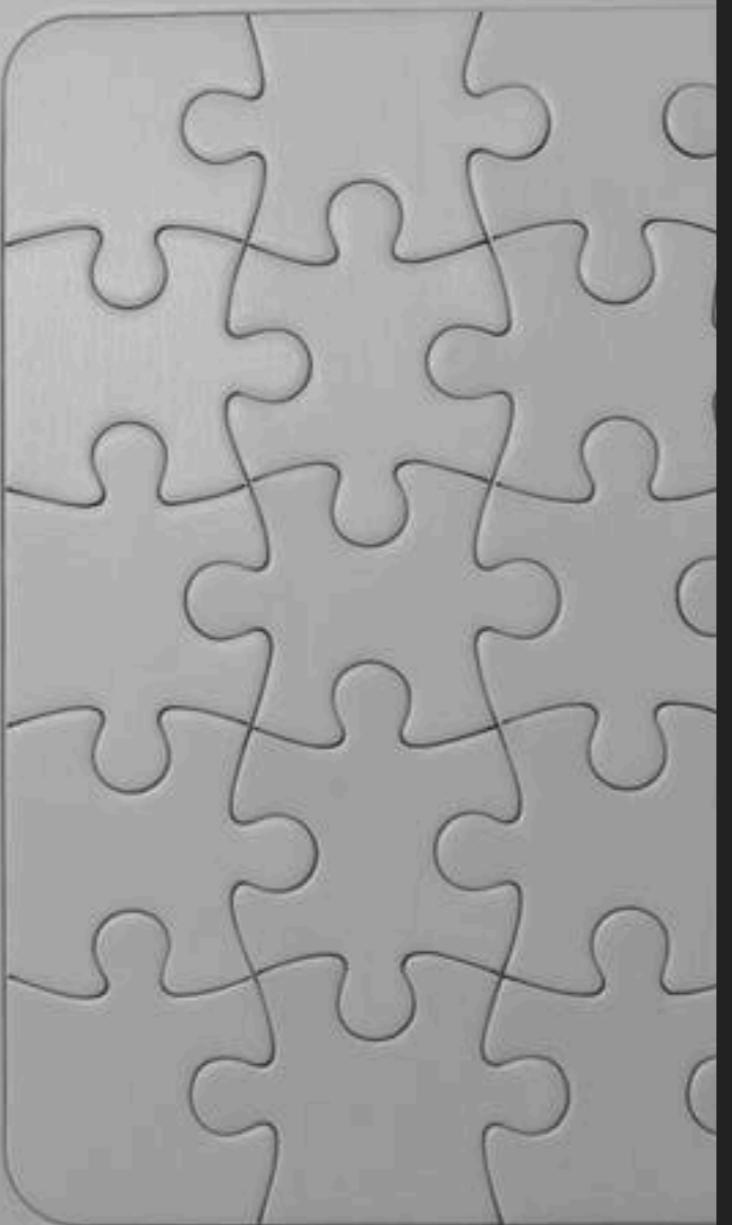
Better Performance

Better Use of Hardware

Better Documentation

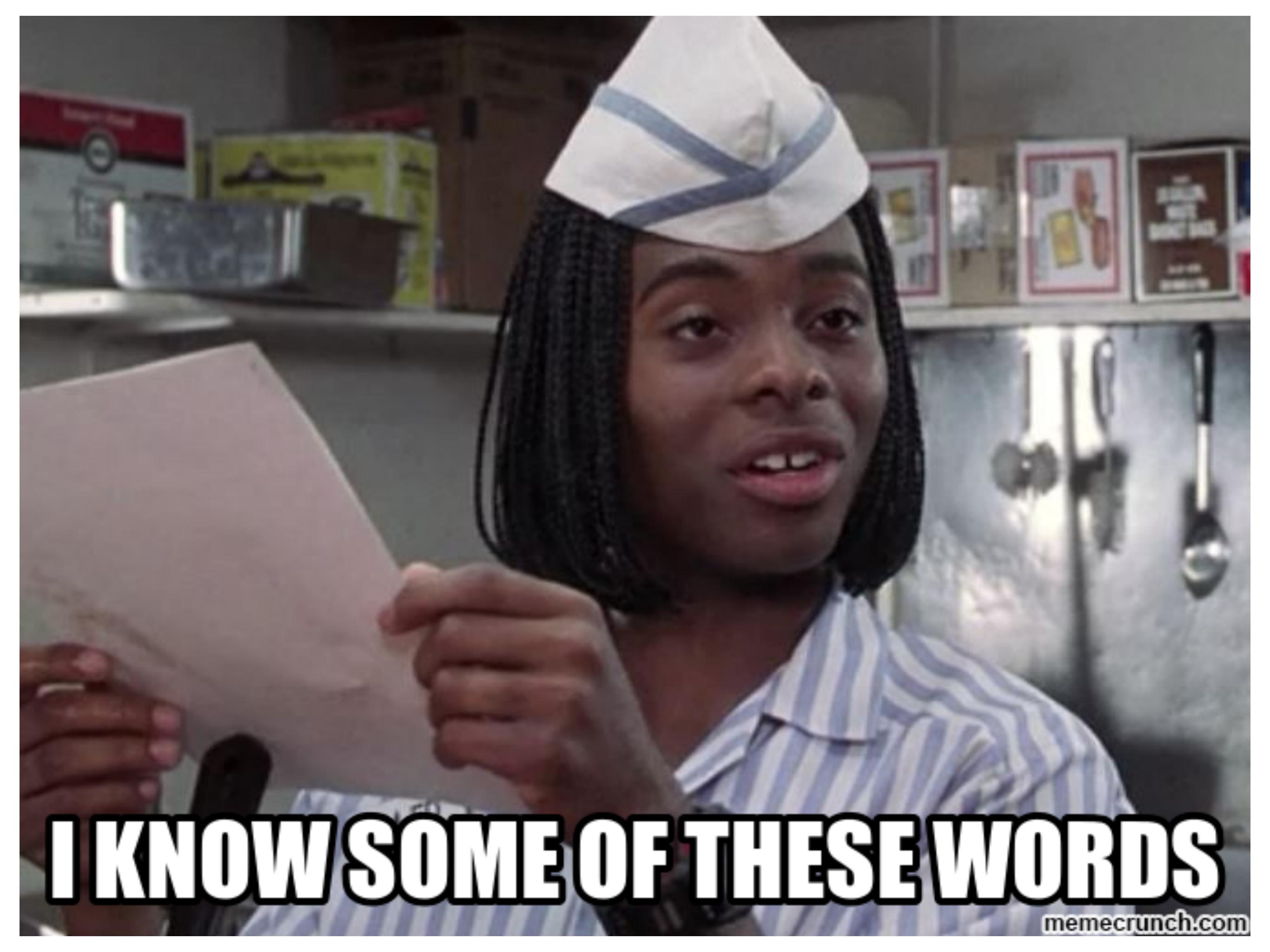
Prettier Graphics

Faster Compilation



JIGSAW

<http://bit.ly/2woEb1S>

A black and white photograph of a woman with dark hair, wearing a white sailor-style hat with blue stripes and a striped shirt. She is looking upwards and pointing her right index finger towards the top left corner of the frame. In her left hand, she holds a large, light-colored menu or piece of paper. The background shows shelves with various items, possibly in a shop or restaurant setting.

I KNOW SOME OF THESE WORDS

JIGSAW GOALS

- ▶ Stronger encapsulation
- ▶ More reliable configuration
- ▶ Create customized JDK images (jlink tool)

Module Types

UNNAMED

- Jars on classpath become unnamed modules
- Exports all packages
- Requires all modules
- Cannot be declared as a dependency

AUTOMATIC

- Jars placed on module path
- Name derived from jar name or optional
Automatic-Module-Name in MANIFEST.MF
- Exports all packages
- Requires all modules
- Can be declared as a dependency

EXPLICIT/NAMED

- Defined by module-info.java
- Can be declared as a dependency

Current file structure

```
src/  
  com.foo.bar/  
    com.foo.bar.model/  
      Foo.java  
  com.foo.bar.service/  
    FooService.java  
  com.foo.bar.service.impl/  
    FooServiceImpl.java
```

Modularized file structure

```
src/
  com.foo.bar/
    module-info.java
    com.foo.bar/
      com.foo.bar.model/
        Foo.java
      com.foo.bar.service/
        FooSvc.java
      com.foo.bar.service.impl/
        FooSvclmpl.java
```

Dissecting module-info

Note:

New reserved words ONLY
reserved in module-info

```
module com.foo.bar {  
    exports com.foo.bar.service;  
    exports com.foo.bar.model;  
  
    provides com.foo.bar.service.FooSvc  
        with  
        com.foo.bar.service.impl.FooSvclmpl;  
  
    uses org.evil.service.EvilService;  
  
    requires org.evil;  
    requires java.sql;  
}
```

<http://bit.ly/2woEb1S>

Module Declaration

Defines the name of the module.

Additional modifiers:

“open”

exports all packages and
allows deep reflective access

```
module com.foo.bar {
```

```
    exports com.foo.bar.service;
```

```
    exports com.foo.bar.model;
```

```
    provides com.foo.bar.service.FooSvc  
        with
```

```
        com.foo.bar.service.impl.FooSvclmpl;
```

```
    uses org.evil.service.EvilService;
```

```
    requires org.evil;
```

```
    requires java.sql;
```

```
}
```

<http://bit.ly/2woEb1S>

Export Declaration

Defines a module's public API

Note:

Each exported package must
be defined individually

Additional modifiers:

“opens” (in lieu of exports)

allows deep reflective access to
all client modules

“to” [module name]

allows only named modules to
have deep reflective access

```
module com.foo.bar {
```

```
    exports com.foo.bar.service;
```

```
    exports com.foo.bar.model;
```

```
    provides com.foo.bar.service.FooSvc  
        with
```

```
        com.foo.bar.service.impl.FooSvclmpl;
```

```
    uses org.evil.service.EvilService;
```

```
    requires org.evil;
```

```
    requires java.sql;
```

```
}
```

<http://bit.ly/2woEb1S>

Provides

Declares the service this module supports and the implementation it supports it with.

```
module com.foo.bar {
```

```
    exports com.foo.bar.service;  
    exports com.foo.bar.model;
```

```
    provides com.foo.bar.service.FooSvc  
        with  
        com.foo.bar.service.impl.FooSvclmpl;
```

```
    uses org.evil.service.EvilService;
```

```
    requires org.evil;  
    requires java.sql;
```

```
}
```

<http://bit.ly/2woEb1S>

Uses

Declares a service this module depends upon

```
module com.foo.bar {  
    exports com.foo.bar.service;  
    exports com.foo.bar.model;  
  
    provides com.foo.bar.service.FooSvc  
        with  
        com.foo.bar.service.impl.FooSvclmpl;  
  
    uses org.evil.service.EvilService;  
  
    requires org.evil;  
    requires java.sql;  
}
```

Requires

Modules this module depends upon.

Additional modifiers:

“transitive” dependent modules don’t have to additionally require this module to make use of it

Note:

`java.base` implicitly required by all modules

```
module com.foo.bar {
```

```
    exports com.foo.bar.service;
```

```
    exports com.foo.bar.model;
```

```
    provides com.foo.bar.service.FooSvc  
        with
```

```
        com.foo.bar.service.impl.FooSvclmpl;
```

```
    uses org.evil.service.EvilService;
```

```
    requires org.evil;
```

```
    requires java.sql;
```

```
}
```

<http://bit.ly/2woEb1S>



JHELL

<http://bit.ly/2woEb1S>

Read

Evaluate

Print

Loop

WHY JSHELL IS PART OF JAVA 9

- ▶ Explore APIs
- ▶ Tool to Teach Java

WHY JSHELL IS PART OF JAVA 9

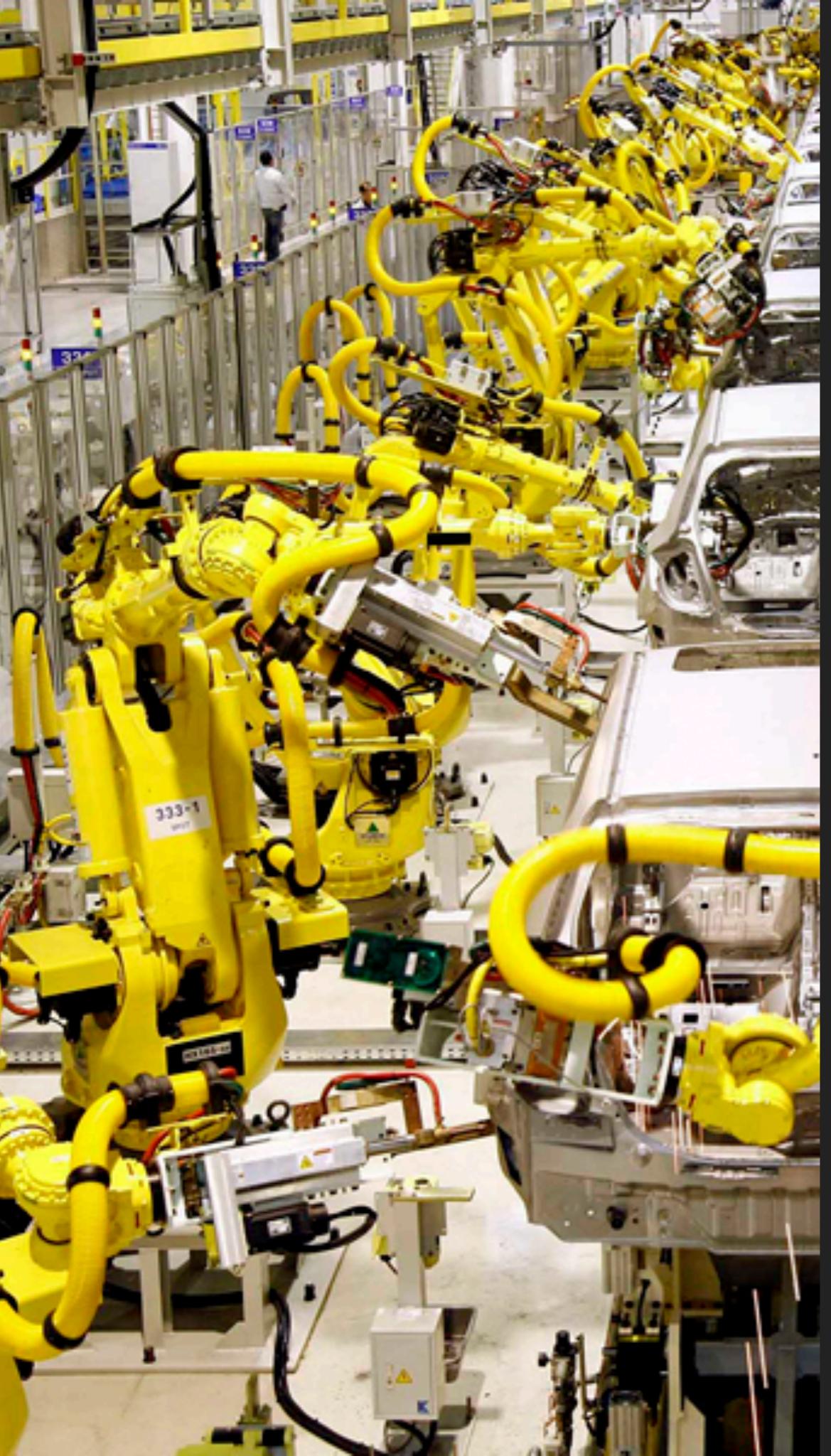
- ▶ Explore APIs
- ▶ Tool to Teach Java

```
public class HelloWorld{  
    public static void main(String args[]) {  
        System.out.println("Hello World!");  
    }  
}
```

```
public class HelloWorld{  
    public static void main(String args[]) {  
        System.out.println("Hello World!");  
    }  
}
```

DEMO

<http://bit.ly/2woEb1S>



COLLECTION FACTORY METHODS

<http://bit.ly/2woEb1S>

```
public class TestFoo{  
  
    @Test  
    public void test() {  
        Foo newFoo = new Foo();  
        List<String> list = new ArrayList();  
        list.add("Hello World!");  
        list.add("Goodbye World!");  
        newFoo.takeListOfStrings(list);  
    }  
  
}
```

```
public class TestFoo{  
  
    @Test  
    public void test() {  
        Foo newFoo = new Foo();  
        List<String> list = List.of(  
            "Hello World!", "Goodbye World!");  
        newFoo.takeListOfStrings(list);  
    }  
}
```

```
public class TestFoo{  
  
    @Test  
    public void test() {  
        Foo newFoo = new Foo();  
        Map<Integer, String> map = Map.of(  
            1, "Hello World!", 2, "Goodbye  
            World!"  
        );  
        newFoo.readAMap(map);  
    }  
}
```

```
public class TestFoo{  
  
    @Test  
    public void test() {  
        Foo newFoo = new Foo();  
        Map<Integer, String> map =  
        Map.ofEntries(  
            Map.entry( 1, "Hello World!" ),  
            Map.entry( 2, "Goodbye World!" )  
        );  
        newFoo.readAMap( map );  
    }  
}
```



STREAMS UPDATES

<http://bit.ly/2woEb1S>

```
public class TakeKansasCitySportsTeams {  
  
    public void takeTeams () {  
        Stream.of("Chiefs", "Royals",  
                 "Sporting", "Mavericks", "", "Monarchs")  
            .takeWhile(s -> !String.isEmpty(s))  
            .forEach(String.out::print)  
    }  
  
}
```

Output: ChiefsRoyalsSportingMavericks

```
public class DropKansasCitySportsTeams {  
  
    public void dropTeams () {  
        Stream.of("", "Chiefs", "Royals",  
                "Sporting", "Mavericks")  
            .dropWhile(s -> !String.isEmpty(s))  
            .forEach(String.out::print)  
    }  
  
}
```

Output: ChiefsRoyalsSportingMavericks



PRIVATE INTERFACE METHODS

<http://bit.ly/2woEb1S>

```
public interface Foo{  
  
    default void bar(String arg) {  
        System.out.println("Hello World!");  
    }  
  
    private default String lorem() {  
        ...  
    }  
}
```

TRY-WITH-RESOURCES

UPDATES

<http://bit.ly/2woEb1S>

```
public class TryWithFoo {  
  
    public void TryWithBar() {  
        try(Resource bar = new Resource()) {  
            //use bar  
        }  
    }  
}
```

```
public class TryWithFoo{  
  
    public void TryWithBar() {  
        Resource bar = new Resource();  
        try (bar) {  
            //use bar  
        }  
    }  
}  
}
```

```
public class TryWithFoo {  
  
    final Resource bar = new Resource();  
    public void TryWithBar() {  
        try (bar) {  
            //use bar  
        }  
    }  
}  
}
```



MULTI-RELEASE JARS

<http://bit.ly/2woEb1S>

EXAMPLE MULTI-RELEASE FILE STRUCTURE

```
jar/  
  foo.class  
  bar.class  
  evil.class  
  META-INF/  
    MANIFEST.MF  
    Multi-Release: true  
    versions/  
      9/  
        foo.class  
        bar.class  
      10/  
        bar.class
```

The diagram illustrates a multi-release file structure for Java 9 and Java 10. It features a dashed circle representing the manifest's visibility boundary. Inside the circle, the `MANIFEST.MF` file contains the entry `Multi-Release: true`. Below it, the `versions` directory is shown, containing two sub-directories: `9/` and `10/`. In the `9/` directory, there are files `foo.class` and `bar.class`. In the `10/` directory, there is a file `bar.class`. Two blue arrows point from the text labels to their respective file locations: one arrow points to the `bar.class` file in the `9/` directory with the label "Code with Java 9 artifacts", and another arrow points to the `bar.class` file in the `10/` directory with the label "Code with Java 10 artifacts".

Code with Java 9 artifacts

Code with Java 10 artifacts



JAVADOC & DEPRECATION ENHANCEMENTS

<http://bit.ly/2woEb1S>

► Javadoc Enhancements

- searchable
- html5 compliant

► @Deprecation enhancements

- addition of forRemoval(boolean)
- addition of since(String)

ADDITIONAL CHANGES

<http://bit.ly/2woEb1S>

ADDITIONAL CHANGES

- ▶ Reactive updates
- ▶ G1 now default garbage collector
- ▶ Compact strings
- ▶ New version format
 - ▶ \$MAJOR.\$MINOR.\$SECURITY.\$PATCH
- ▶ HTTP/2 support (experimental)
- ▶ Unified GC Logging
- ▶ Unified JVM Logging
- ▶ Stackwalking API

REMOVALS

<http://bit.ly/2woEb1S>

REMOVALS

- ▶ jhat
- ▶ applet API
- ▶ Internal APIs (e.g. com.sun.Unsafe)
- ▶ Deprecated GC combinations
- ▶ JVM TI hprof Agent

DEVELOPER TOOLS

<http://bit.ly/2woEb1S>

DEVELOPER TOOL JAVA 9 COMPATIBILITY

- ▶ IntelliJ
 - ▶ IDEA 2017.1+
- ▶ Eclipse
 - ▶ Requires Oxygen, currently in (BETA)
- ▶ Netbeans
 - ▶ Currently available in nightly builds
- ▶ Maven
 - ▶ 3.0+
- ▶ Gradle
 - ▶ 4.1 (limited Java 9 compatibility)
 - ▶ Full modules support coming in 4.2

SPRING FRAMEWORK JAVA 9 SUPPORT



- ▶ SPRING 4.X (CURRENT)
 - LIMITED AND ONLY FOR 4.3.10
- ▶ SPRING 5.X (SEPTEMBER 2017)
 - STABLE AUTOMATIC MODULE NAMES
- ▶ SPRING 6.X (2019+)
 - FULL JIGSAW SUPPORT

SPRING LIBRARIES JAVA 9 SUPPORT



- ▶ SPRING BOOT 1.X (CURRENT)
 - NO SUPPORT
- ▶ SPRING BOOT 2.X (NOVEMBER 2017)
 - SUPPORTS RUNNING IN JAVA 9
- ▶ OTHER LIBRARIES WILL VARY

WHO TO FOLLOW FOR JAVA 9 NEWS



@PBAKKER
PAUL BAKKER



@SANDER_MAK
SANDER MAK



@NIPAFX
NICOLAI PARLOG

RESOURCES

Source code: <https://github.com/wkorando/java-9-microservices-demo>

Real World Java 9 by Trisha Gee: <https://www.youtube.com/watch?v=GkP83hvdeMk>

AskArch DevoxxUK: https://www.youtube.com/watch?v=ac1v5kF_FGs

Java 9 Modularity in Action: <https://www.youtube.com/watch?v=oy3202OFPpM>

Full Java 9 feature list: <http://openjdk.java.net/projects/jdk9/>

What's in Java 9: <https://dzone.com/articles/java-9-besides-modules>

OPPORTUNITIES TO CONNECT AND TALK JAVA & SPRING



1ST WEDNESDAY OF THE MONTH
**[HTTPS://WWW.MEETUP.COM/
KC-SPRING](https://www.meetup.com/KC-SPRING)**



3RD WEDNESDAY OF THE MONTH
**[HTTPS://WWW.MEETUP.COM/
KANSASCITYJUG](https://www.meetup.com/KANSASCITYJUG)**

GET STARTED!

[HTTP://JDK.JAVA.NET/9/](http://JDK.JAVA.NET/9/)

[HTTP://JDK.JAVA.NET/JIGSAW/](http://JDK.JAVA.NET/JIGSAW/)