# Compile Screencast

•••

Ramin Zare

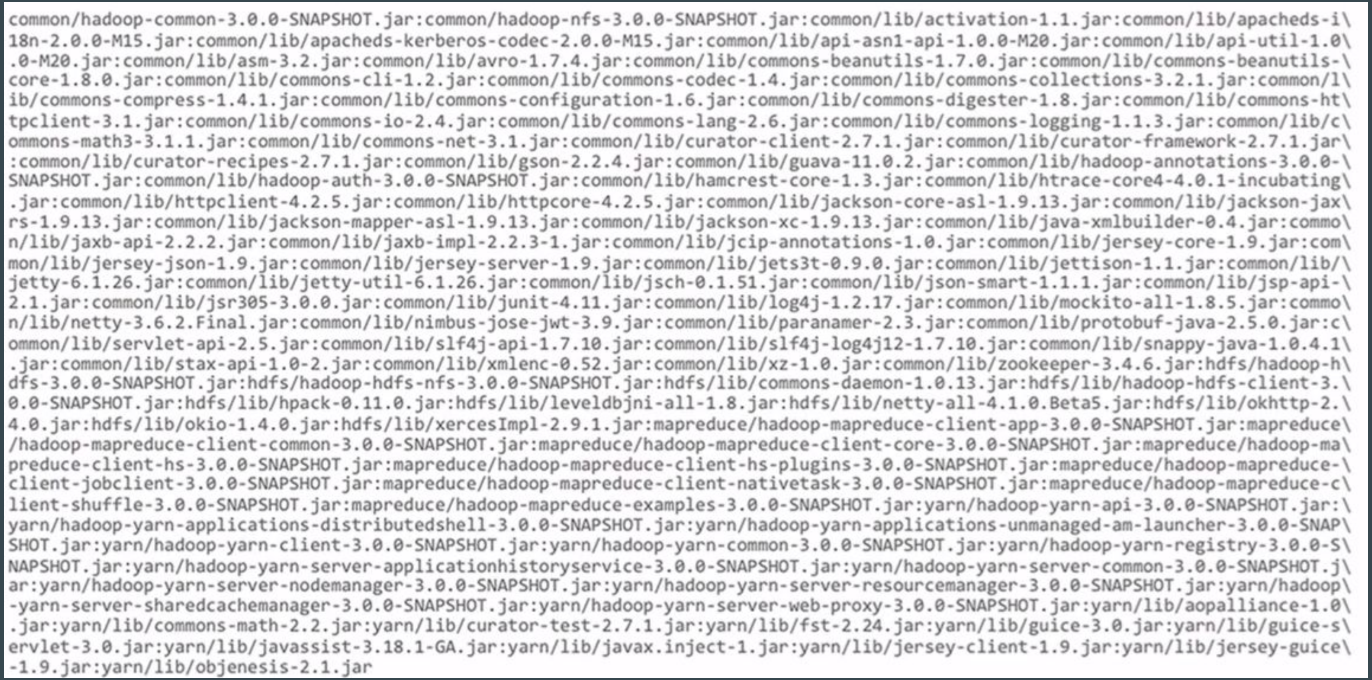
# JPMS (Java platform module system)

Java 9 feature

## Monolithic JDK

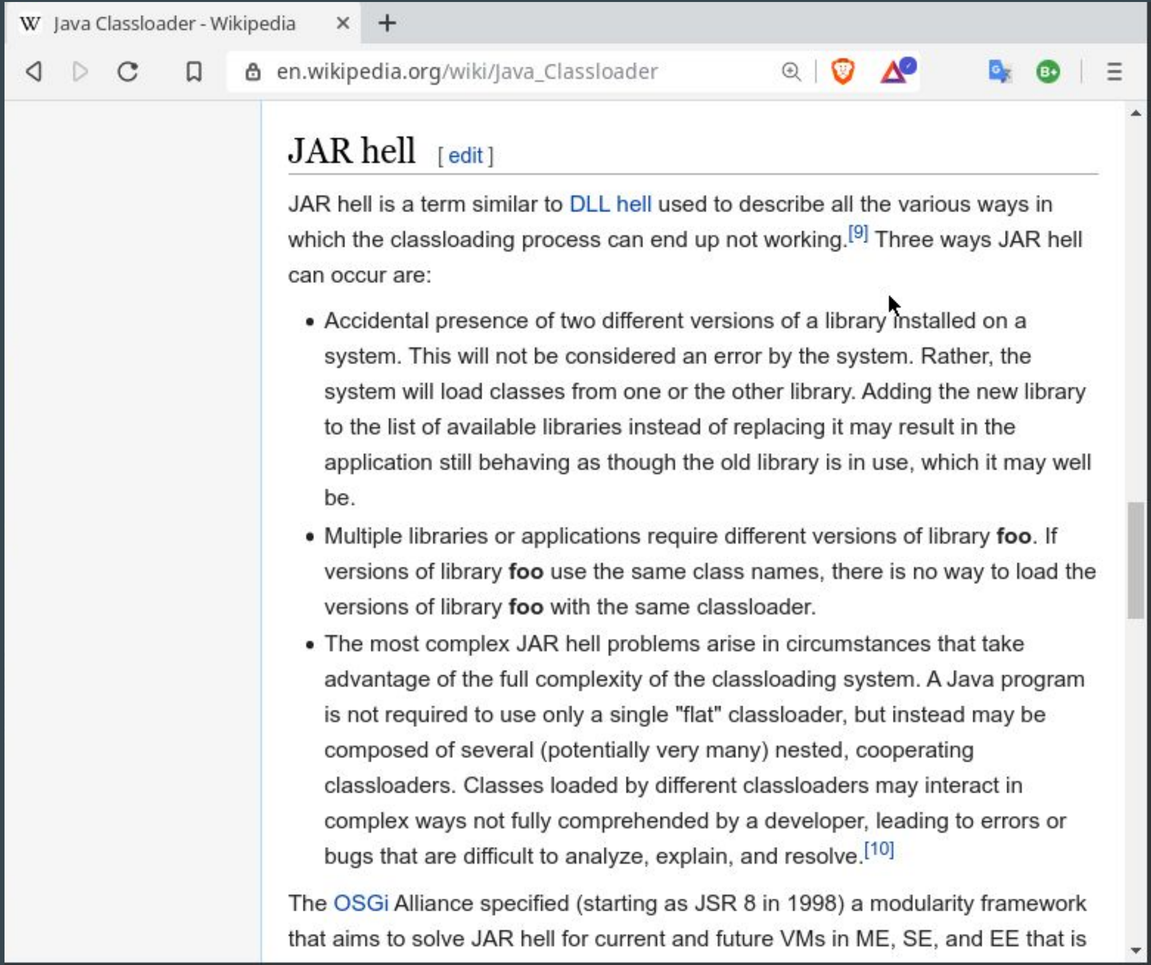
`-classpath`

## -classpath



```
common/hadoop-common-3.0.0-SNAPSHOT.jar:common/hadoop-nfs-3.0.0-SNAPSHOT.jar:common/lib/activation-1.1.jar:common/lib/apacheds-i\
18n-2.0.0-M15.jar:common/lib/apacheds-kerberos-codec-2.0.0-M15.jar:common/lib/api-asn1-api-1.0.0-M20.jar:common/lib/api-util-1.0\
.0-M20.jar:common/lib/asm-3.2.jar:common/lib/avro-1.7.4.jar:common/lib/commons-beanutils-1.7.0.jar:common/lib/commons-beanutils-\
core-1.8.0.jar:common/lib/commons-cli-1.2.jar:common/lib/commons-codec-1.4.jar:common/lib/commons-collections-3.2.1.jar:common/l\
ib/commons-compress-1.4.1.jar:common/lib/commons-configuration-1.6.jar:common/lib/commons-digester-1.8.jar:common/lib/commons-ht\
tpclient-3.1.jar:common/lib/commons-io-2.4.jar:common/lib/commons-lang-2.6.jar:common/lib/commons-logging-1.1.3.jar:common/lib/c\
ommons-math3-3.1.1.jar:common/lib/commons-net-3.1.jar:common/lib/curator-client-2.7.1.jar:common/lib/curator-framework-2.7.1.jar\
:common/lib/curator-recipes-2.7.1.jar:common/lib/gson-2.2.4.jar:common/lib/guava-11.0.2.jar:common/lib/hadoop-annotations-3.0.0-\
SNAPSHOT.jar:common/lib/hadoop-auth-3.0.0-SNAPSHOT.jar:common/lib/hamcrest-core-1.3.jar:common/lib/htrace-core4-4.0.1-incubating\
.jar:common/lib/httpclient-4.2.5.jar:common/lib/httpcore-4.2.5.jar:common/lib/jackson-core-asl-1.9.13.jar:common/lib/jackson-jax\
rs-1.9.13.jar:common/lib/jackson-mapper-asl-1.9.13.jar:common/lib/jackson-xc-1.9.13.jar:common/lib/java-xmlbuilder-0.4.jar:commo\
n/lib/jaxb-api-2.2.2.jar:common/lib/jaxb-impl-2.2.3-1.jar:common/lib/jcip-annotations-1.0.jar:common/lib/jersey-core-1.9.jar:com\
mon/lib/jersey-json-1.9.jar:common/lib/jersey-server-1.9.jar:common/lib/jets3t-0.9.0.jar:common/lib/jettison-1.1.jar:common/lib/\
jetty-6.1.26.jar:common/lib/jetty-util-6.1.26.jar:common/lib/jsch-0.1.51.jar:common/lib/json-smart-1.1.1.jar:common/lib/jsp-api-\
2.1.jar:common/lib/jsr305-3.0.0.jar:common/lib/junit-4.11.jar:common/lib/log4j-1.2.17.jar:common/lib/mockito-all-1.8.5.jar:commo\
n/lib/netty-3.6.2.Final.jar:common/lib/nimbus-jose-jwt-3.9.jar:common/lib/paranamer-2.3.jar:common/lib/protobuf-java-2.5.0.jar:c\
ommon/lib/servlet-api-2.5.jar:common/lib/slf4j-api-1.7.10.jar:common/lib/slf4j-log4j12-1.7.10.jar:common/lib/snappy-java-1.0.4.1\
.jar:common/lib/stax-api-1.0-2.jar:common/lib/xmlenc-0.52.jar:common/lib/xz-1.0.jar:common/lib/zookeeper-3.4.6.jar:hdfs/hadoop-h\
dfs-3.0.0-SNAPSHOT.jar:hdfs/hadoop-hdfs-nfs-3.0.0-SNAPSHOT.jar:hdfs/lib/commons-daemon-1.0.13.jar:hdfs/lib/hadoop-hdfs-client-3.\
0.0-SNAPSHOT.jar:hdfs/lib/hpack-0.11.0.jar:hdfs/lib/leveldbjni-all-1.8.jar:hdfs/lib/netty-all-4.1.0.Beta5.jar:hdfs/lib/okhttp-2.\
4.0.jar:hdfs/lib/okio-1.4.0.jar:hdfs/lib/xercesImpl-2.9.1.jar:mapreduce/hadoop-mapreduce-client-app-3.0.0-SNAPSHOT.jar:mapreduce\
/hadoop-mapreduce-client-common-3.0.0-SNAPSHOT.jar:mapreduce/hadoop-mapreduce-client-core-3.0.0-SNAPSHOT.jar:mapreduce/hadoop-ma\
preduce-client-hs-3.0.0-SNAPSHOT.jar:mapreduce/hadoop-mapreduce-client-hs-plugins-3.0.0-SNAPSHOT.jar:mapreduce/hadoop-mapreduce-\
client-jobclient-3.0.0-SNAPSHOT.jar:mapreduce/hadoop-mapreduce-client-nativetask-3.0.0-SNAPSHOT.jar:mapreduce/hadoop-mapreduce-c\
lient-shuffle-3.0.0-SNAPSHOT.jar:mapreduce/hadoop-mapreduce-examples-3.0.0-SNAPSHOT.jar:yarn/hadoop-yarn-api-3.0.0-SNAPSHOT.jar:\
yarn/hadoop-yarn-applications-distributedshell-3.0.0-SNAPSHOT.jar:yarn/hadoop-yarn-applications-unmanaged-am-launcher-3.0.0-SNAP\
SHOT.jar:yarn/hadoop-yarn-client-3.0.0-SNAPSHOT.jar:yarn/hadoop-yarn-common-3.0.0-SNAPSHOT.jar:yarn/hadoop-yarn-registry-3.0.0-S\
NAPSHOT.jar:yarn/hadoop-yarn-server-applicationhistoryservice-3.0.0-SNAPSHOT.jar:yarn/hadoop-yarn-server-common-3.0.0-SNAPSHOT.j\
ar:yarn/hadoop-yarn-server-nodemanager-3.0.0-SNAPSHOT.jar:yarn/hadoop-yarn-server-resourcemanager-3.0.0-SNAPSHOT.jar:yarn/hadoop\
-yarn-server-sharedcachemanager-3.0.0-SNAPSHOT.jar:yarn/hadoop-yarn-server-web-proxy-3.0.0-SNAPSHOT.jar:yarn/lib/aopalliance-1.0\
.jar:yarn/lib/commons-math-2.2.jar:yarn/lib/curator-test-2.7.1.jar:yarn/lib/fst-2.24.jar:yarn/lib/guice-3.0.jar:yarn/lib/guice-s\
ervlet-3.0.jar:yarn/lib/javassist-3.18.1-GA.jar:yarn/lib/javax.inject-1.jar:yarn/lib/jersey-client-1.9.jar:yarn/lib/jersey-guice\
-1.9.jar:yarn/lib/objenesis-2.1.jar
```

# JAR Hell



Java Classloader - Wikipedia

en.wikipedia.org/wiki/Java_Classloader

## JAR hell [ edit ]

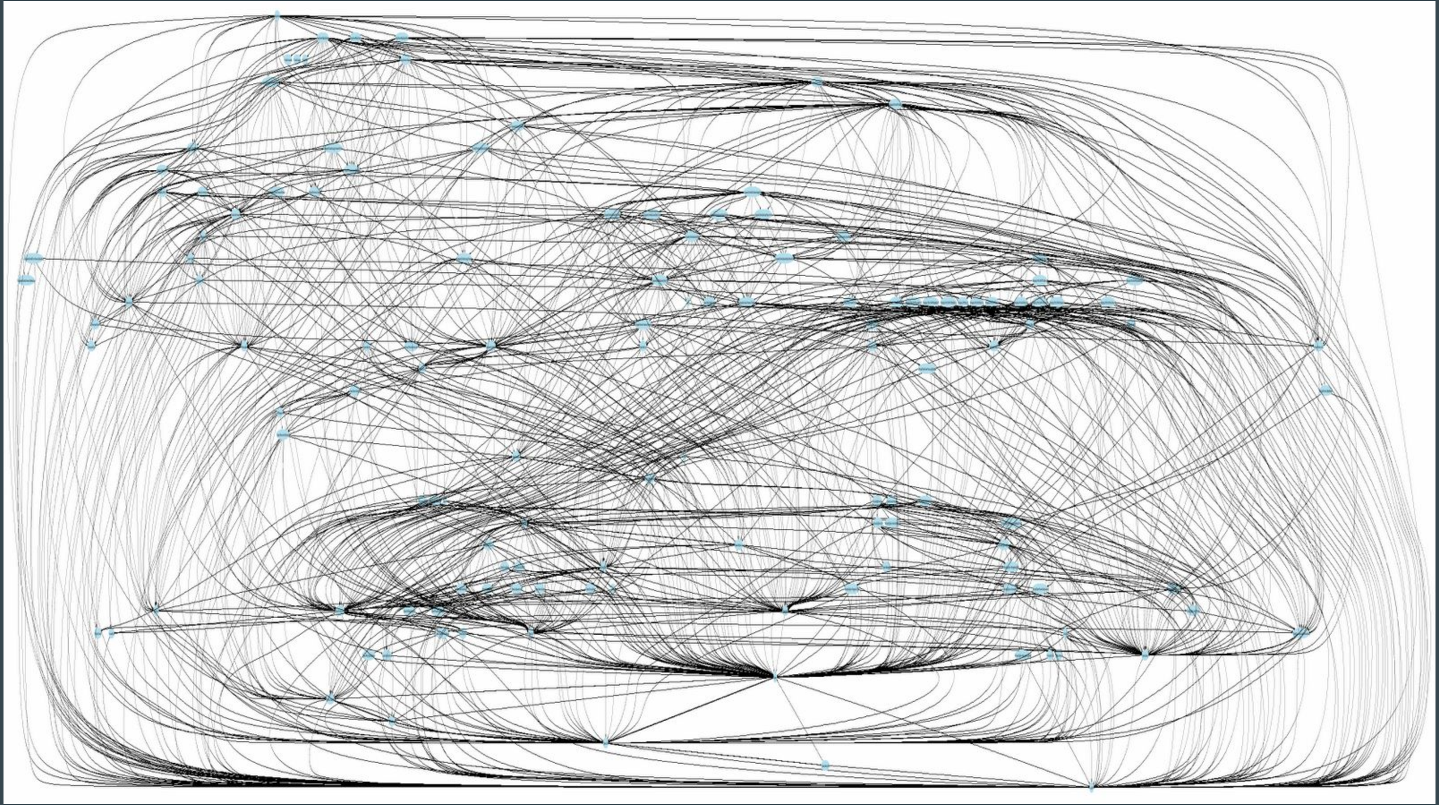
JAR hell is a term similar to DLL hell used to describe all the various ways in which the classloading process can end up not working.[9] Three ways JAR hell can occur are:

- Accidental presence of two different versions of a library installed on a system. This will not be considered an error by the system. Rather, the system will load classes from one or the other library. Adding the new library to the list of available libraries instead of replacing it may result in the application still behaving as though the old library is in use, which it may well be.
- Multiple libraries or applications require different versions of library **foo**. If versions of library **foo** use the same class names, there is no way to load the versions of library **foo** with the same classloader.
- The most complex JAR hell problems arise in circumstances that take advantage of the full complexity of the classloading system. A Java program is not required to use only a single "flat" classloader, but instead may be composed of several (potentially very many) nested, cooperating classloaders. Classes loaded by different classloaders may interact in complex ways not fully comprehended by a developer, leading to errors or bugs that are difficult to analyze, explain, and resolve.[10]

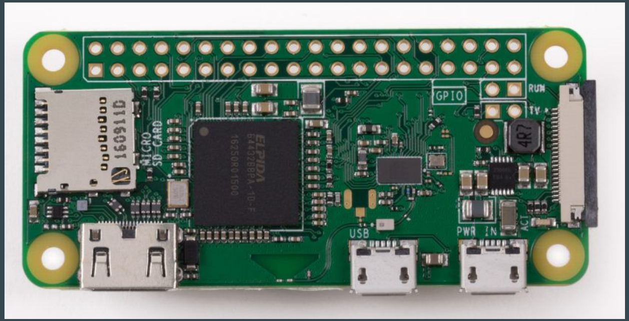
The OSGi Alliance specified (starting as JSR 8 in 1998) a modularity framework that aims to solve JAR hell for current and future VMs in ME, SE, and EE that is

## Monolithic JDK

## Monolithic JDK



## Monolithic JDK



## API های داخلی

بعضی از کلاسها داخل پکیج ها واقعا public نیستند

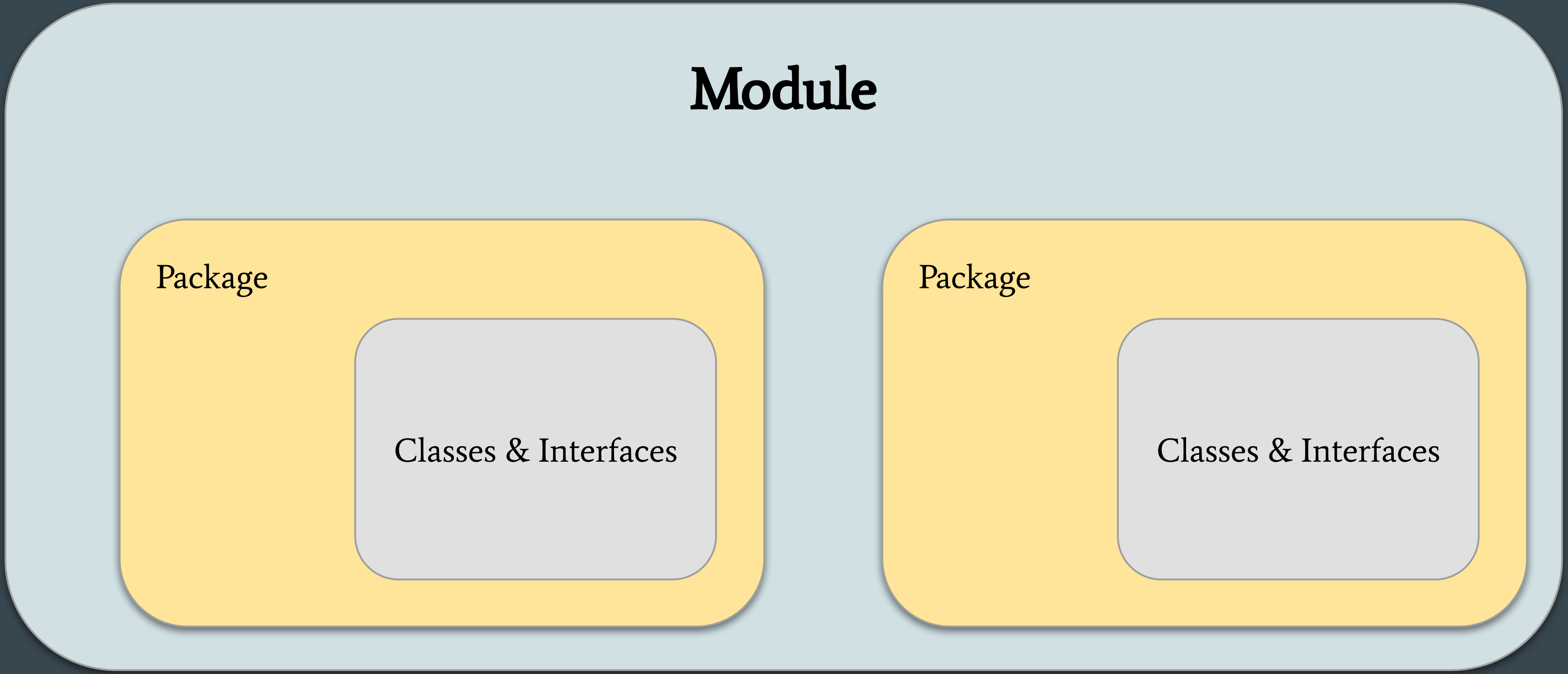
```
sun.*

*.internal.*
```

*java.lang.SecurityManager::checkPackageAccess*

## Modules

- پیکربندی قابل اطمینان
- Strong encapsulation

## Modules


Module
Package
Classes & Interfaces
Package
Classes & Interfaces

## Module چیست؟

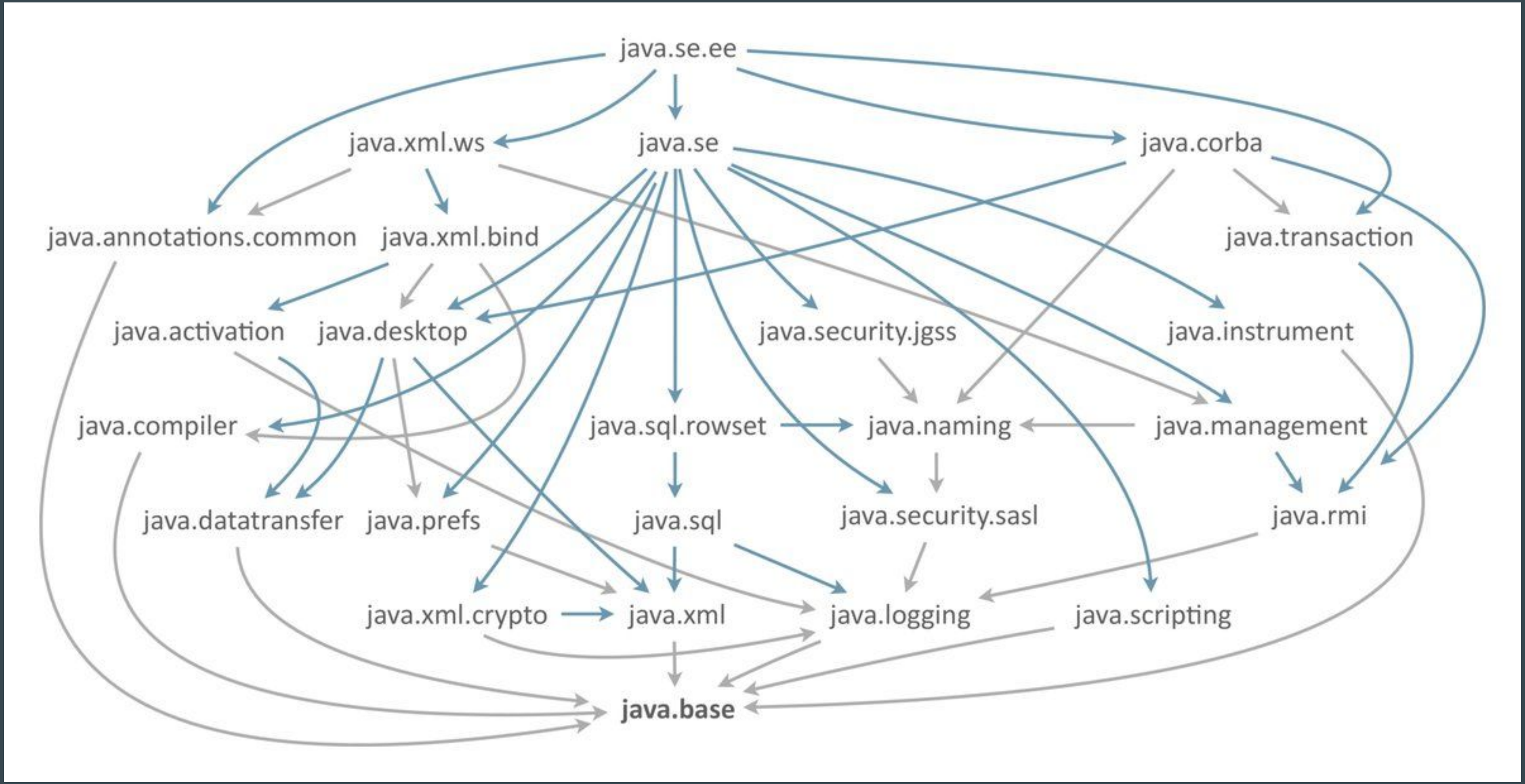
- یک ظرفی برای package هاست
- همیشه یه نام دارد
- یک فایل توصیفی دارد که در آن مشخص میکنیم
  - نام دیگر ماژول هایی که به آن وابسته هستیم
  - پکیج هایی که میخواهیم اجازه دهیم دیگران به آنها دسترسی داشته باشند
- نام فایل توصیفی ماژول ها `module-info.java` است.

## Modular Jar file

```
META-INF/MANIFEST.MF

module-info.class

com/.../DatabaseConnection.class

com/.../EmployeeDAO.class
```

## نوشتن یک نمونه کد

# Modular JDK


java.se.ee
java.xml.ws
java.se
java.corba
java.annotations.common
java.xml.bind
java.transaction
java.activation
java.desktop
java.security.jgss
java.instrument
java.compiler
java.sql.rowset
java.naming
java.management
java.datatransfer
java.prefs
java.sql
java.security.sasl
java.rmi
java.xml.crypto
java.xml
java.logging
java.scripting
java.base

# Modules

- Readable modules
- Reliable configuration

```
user/module-info.java:2: error: module not found: com.compilepodcast.car

   requires com.compilepodcast.engine;

                    ^

1 error
```

## Modules

/home/ramin/demo/api/module-info.java

```
module com.compilepodcast.engine{
    requires java.logging;
    exports com.compilepodcast.engine.main;
}
```

/home/ramin/demo/user/module-info.java

```
module com.compilepodcast.car {
    requires com.compilepodcast.engine;
}
```

## Strong encapsulation

-> Modular declaration

    -> Modular descriptor

        -> Modular jar file or Jmod file

            -> Observable module -> Readable modules

                -> Readable configuration

                    -> Accessible types

                        -> Strong encapsulation (Maintainability)

## Strong encapsulation

In Java 8

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

## Strong encapsulation

In Java 9 (public no longer means accessible)

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

## Q & A

- چطور میتونیم در یک ماژول یک پکیج را برای یک ماژول خاص فقط public کرد؟

```
module com.compilepodcast.engine{
    requires java.logging;
    exports com.compilepodcast.engine.main to com.compilepodcast.car;
}
```

## Q & A

- opens و to .. opens چیست؟

اجازه میده پکیج شما برای ماژول های دیگه توسط reflection قابل دسترس باشه

```
opens com.compilepodcast.api.main.model ;
opens com.compilepodcast.api.main.model to com.google.gson;
```

## Q & A

- requires transitive چیست؟

به اسم "implied readability" هم شناخته میشه

```
module com.compilepodcast.engine{
    requires java.logging;
    exports com.compilepodcast.engine.main;
}
module com.compilepodcast.car {
   requires transitive com.compilepodcast.engine; // also requires java.logging
}
```

## Q & A

- **provide … with** در ماژول چیست؟

In Java 9, we can develop **Services** and **Service Providers** as modules.
A service module declares that it uses one or more interfaces whose implementations will be provided at run-time by some provider modules.

## Q & A

- provide ... with در ماژول چیست؟

```
module com.compilepodcast.weather.logic{
   exports com.compilepodcast.weather.service;
   provides com.compilepodcast.weather.service.WeatherService with
com.compilepodcast.weather.service.internal.DarkskyAPIWeatherService;
}
```

## Q & A

- **provide ... with** در ماژول چیست؟

```
module com.compilepodcast.weather.ui {
     requires com.compilepodcast.weather.logic;
     uses com.compilepodcast.weather.service.WeatherService;
}
```

```
// Load the implementations of a service

ServiceLoader<WeatherService> services = ServiceLoader.load(WeatherService.class);

WeatherService service= services.findFirst().orElseThrow(IllegalStateException::new);
```

## Q & A

- What is JLink?

With jlink, Java applications can be distributed as native runtime images that do not require the JVM to be installed in the target system to run.

## Q & A

- Unnamed modules چیست؟

کلاسی که عضوی از یک 'named module' نباشد

```
java.lang.Class#getModule()

Module: unnamed module @7c75222b
Name: null
isNamed: false
Descriptor: null
```

## Q & A

- automatic modules چیست؟

هر Jar file که شامل module-info.class در دایرکتوری اصلی اش نداشته باشد به عنوان automatic module شناخته میشود

## Q & A

- ابزار jdeps چه فایده ای دارد؟

این ابزار به شما کمک میکند وابستگی کلاس ها یا پکیج های شما را تشخیص دهید

```
jdeps /home/user/hibernate-2.0-M8.jar
```

## آخرین نکته

سیستم ماژول یک ویژگی معمولی نیست
نمیتوان آن را به سرعت به پروژه اضافه و استفاده کرد

ماژول ها مثل یک کمربند ایمنی هستند
برنامه شما را ایمن تر میکنند
و از طرفی کار بیشتری میطلبد

## منابع

- https://openjdk.java.net/projects/jigsaw/
- https://www.youtube.com/watch?v=l1s7R85GF1A
- https://sites.google.com/a/athaydes.com/renato-athaydes/posts/guidetojava9-compilejarrun
- https://www.journaldev.com/13121/java-9-features-with-examples
- https://www.youtube.com/watch?v=QnMDsI2GbOc
- https://www.logicbig.com/tutorials/core-java-tutorial/modules/unnamed-modules.html

# Compile Screencast

Ramin Zare

Youtube : youtube.com/c/RaminZare
Github : @zare88