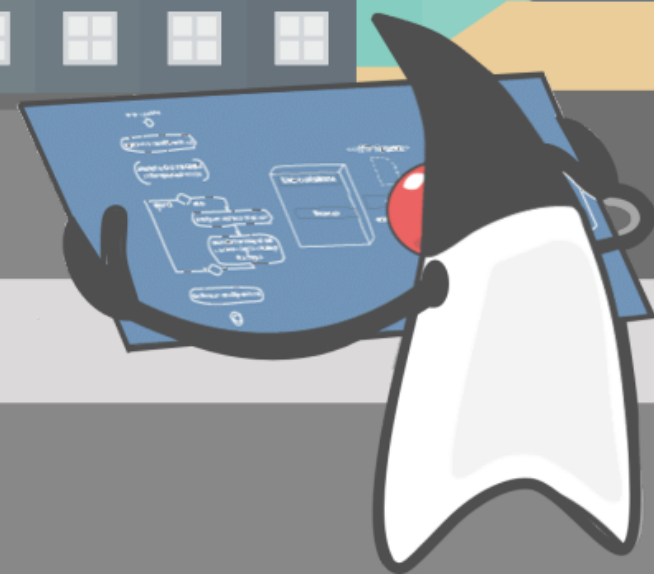
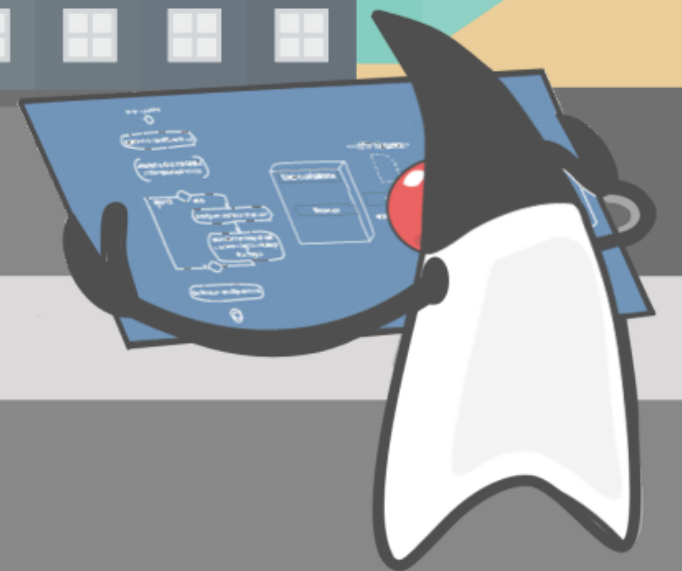


# En route vers Java 17



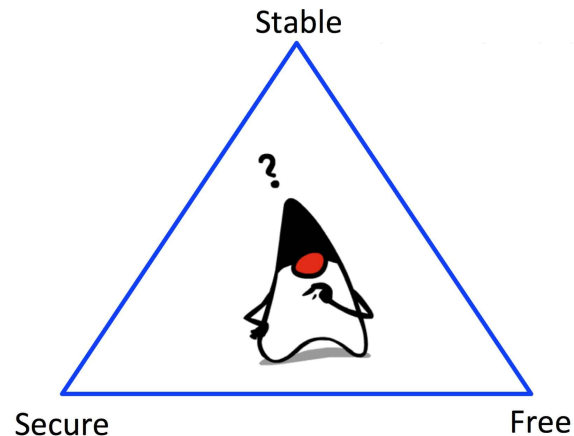
# Java 9



# Changement de cycle de release

## Jusqu'à Java 8

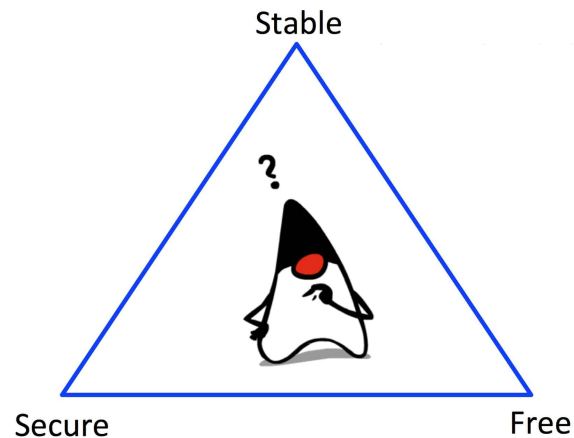
- **Stable** : release espacées dans le temps avec un chevauchement entre les diffusions public de version
- **Sécurisé** : mise à jour gratuite et déployées rapidement
- **Gratuit** : plateforme gratuite à l'usage et open-source depuis 2006



# Changement de cycle de release

## Ce qui ne change pas

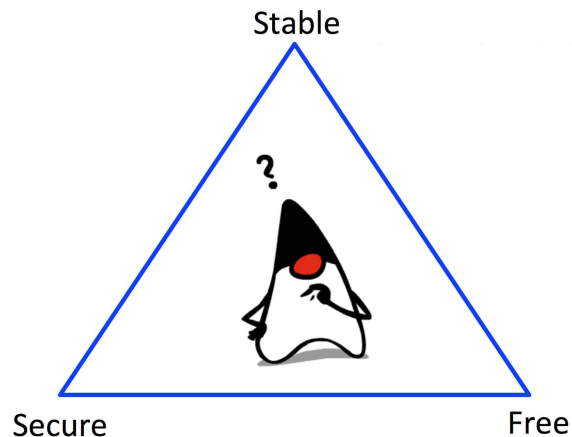
- Java reste une plateforme gratuite
- Java sera toujours mis à jour gratuitement et rapidement



# Changement de cycle de release

## Ce qui change

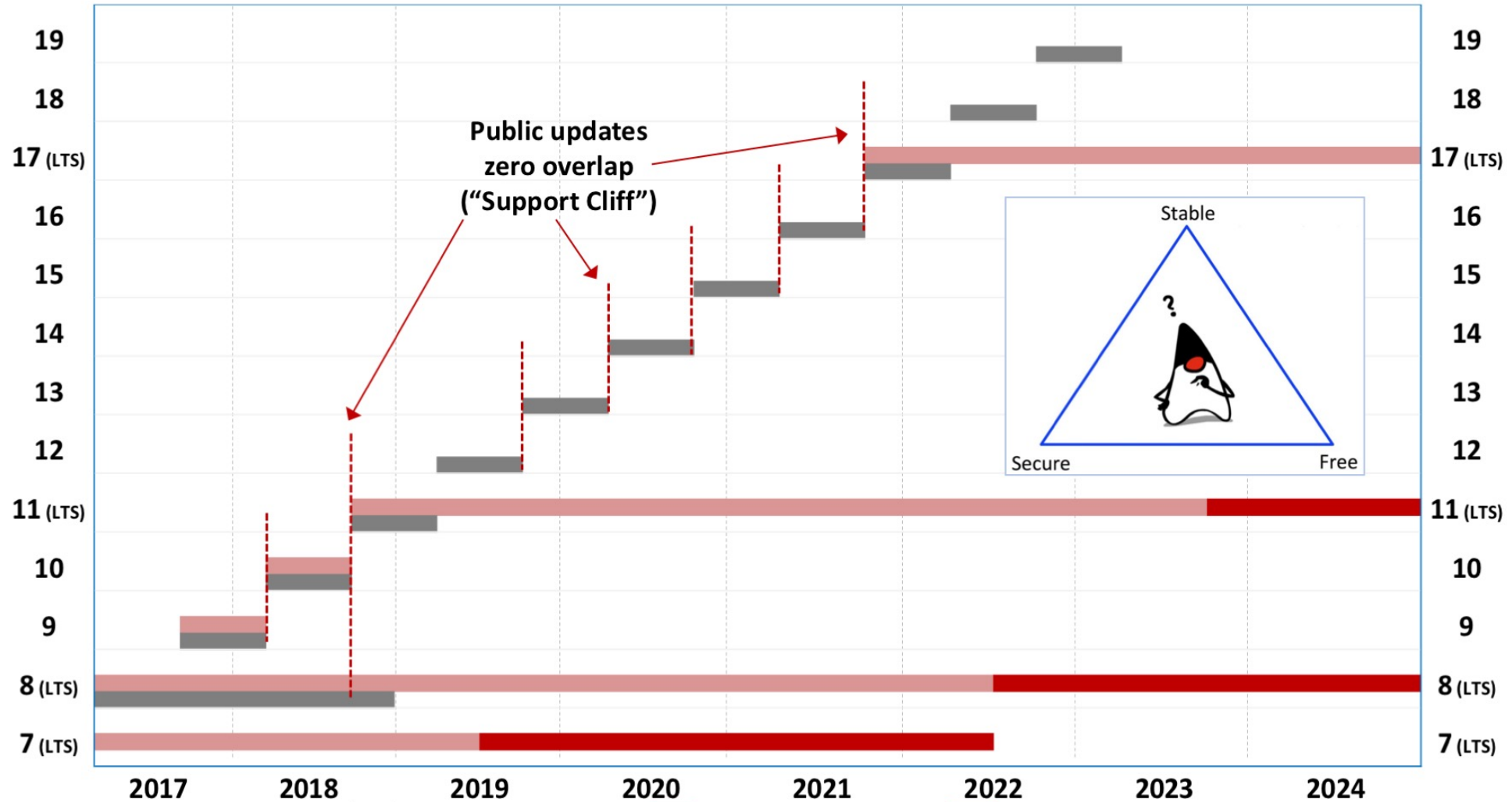
- 2 versions majeures par ans (mars et septembre)
- 1 version LTS tous les 3 ans à partir de java 11 (25 septembre 2018)
- plus de chevauchement dans les diffusions de versions publiques
- support commercial seulement pour les LTS



# Changement de cycle de release

## Java SE Lifecycle – 5+ Year Timeline

Java SE Version



Pourquoi parle-t-on autant de Java 9+ ?

Oracle Publicly available binaries (unsupported)

Oracle Commercial Support

Oracle Extended Commercial Support

# Les principales évolutions langage et API

## Ajout de l'instruction var pour l'inférence de type

```
// public ArrayList<Entity<Person>> getPersons();  
  
// En java 8  
List<Entity<Person>> listEntityPerson = getPersons();  
// peut être remplacé en Java 10 par  
var listEntityPerson = getPersons();
```

*Le type de listEntityPerson sera ArrayList avec le var*

```
for (var entityPerson : getPersons()) {  
    System.out.println(entityPerson);  
}
```



## Builder pour collections immuables

```
var immutableList = List.of("a", "b", "c");
```

```
var immutableMap = Map.of(  
    "a", 1,  
    "b", 2,  
    "c", 3  
);
```

*Map.of* est limité à une map de 10 entrée, après il faut passer par *ofEntries*





## Builder pour collections immuables

```
// Méthode à créer dans une classe utilitaire
public static Map.Entry <K,V> e(K key, V value) {
    return new AbstractMap.SimpleEntry<K,V>(key, value);
}
```

```
var immutableMap = Map.ofEntries(
    e("a", 1),
    e("b", 2),
    e("c", 3)
);
```

Plus de limitation à 10 entrées ici *<i class="em em-slightly\_smiling\_face"> </i>*



## Process API : API standard pour manipuler les processus de la machine hôte

Lister tous les processus de la machine :

```
ProcessHandle.allProcess()  
    .map(p -> p.info().command())  
    .collect(Collectors.toList());
```

Créer un nouveau processus :

```
ProcessBuilder processBuilder = new ProcessBuilder("notepad.exe");  
Process process = processBuilder.start();  
ProcessHandle processHandle = process.toHandle();
```



## Flow API : 4 interfaces standard pour la programmation reactive

```
public static interface Flow.Publisher<T> {  
    public void      subscribe(Flow.Subscriber<? super T> subscriber);  
}  
public static interface Flow.Subscriber<T> {  
    public void      onSubscribe(Flow.Subscription subscription);  
    public void      onNext(T item) ;  
    public void      onError(Throwable throwable) ;  
    public void      onComplete() ;  
}  
public static interface Flow.Subscription {  
    public void      request(long n);  
    public void      cancel() ;  
}  
public static interface Flow.Processor<T,R>  
    extends Flow.Subscriber<T>, Flow.Publisher<R> {  
}
```



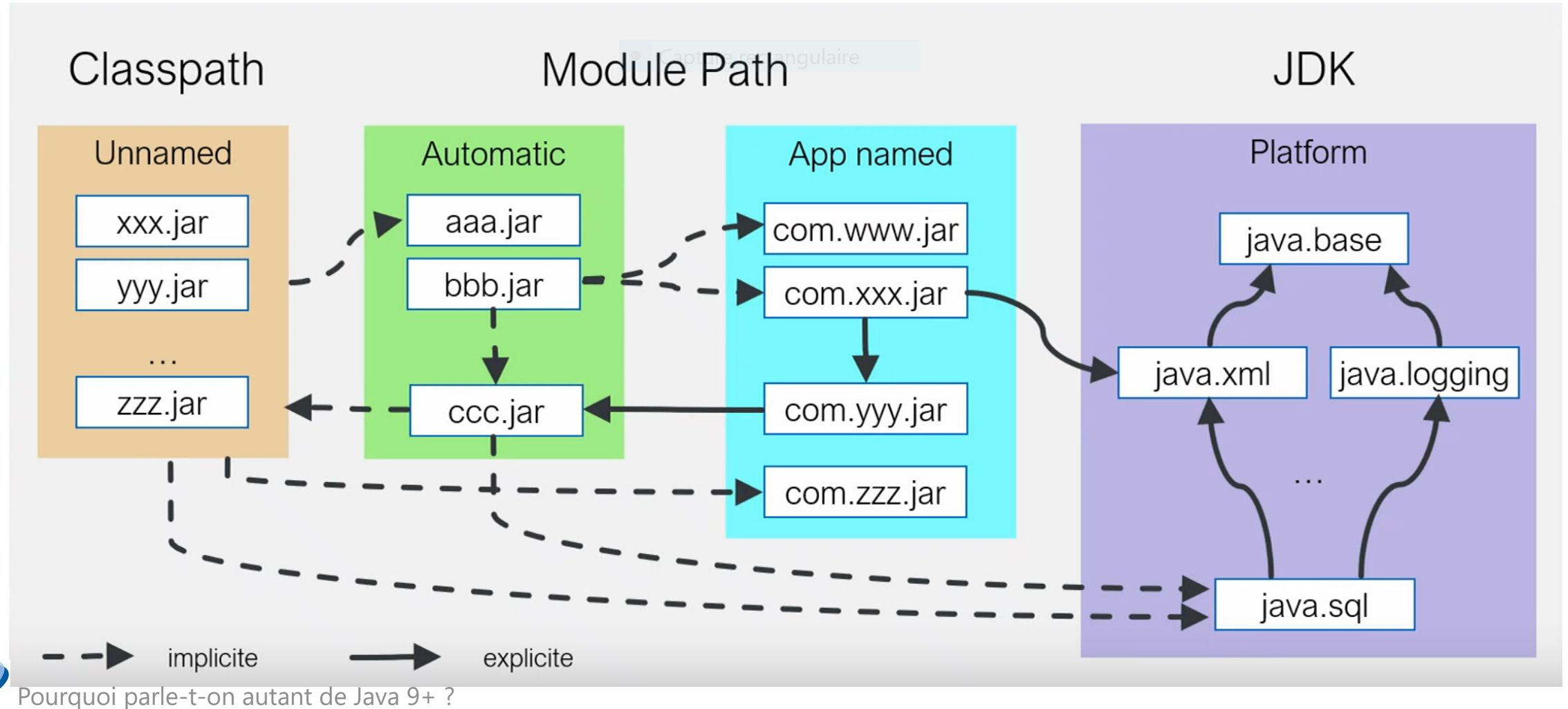
## Introduction des Modules

- Nouveau niveau d'encapsulation dans Java
- un module c'est un jar qui définit :
  - ce dont il a besoin pour fonctionner (ses dépendances)
  - ce qu'il expose comme package



# Les principales évolutions langage et API

## Introduction des Modules

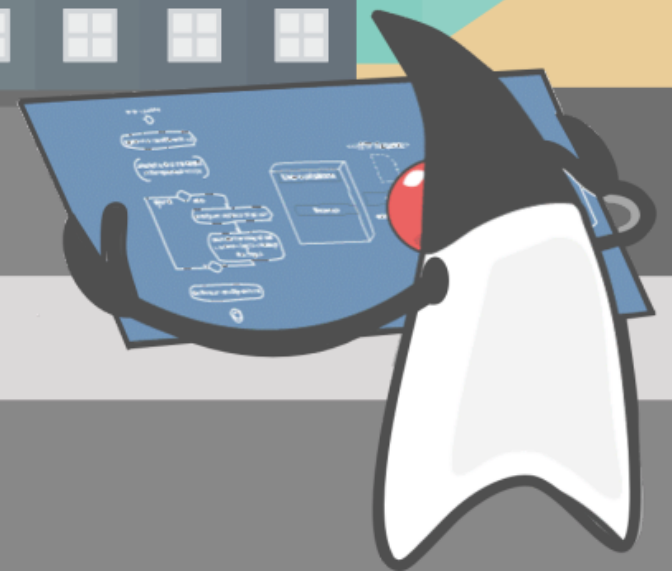


# Les principales évolutions dans les outils

- JVM :
  - CompactString : passage de String char[] à String byte[]
  - meilleur support de Docker (utilisation des CGroup sur Linux)
- JShell : interpréteur REPL (Read-Evaluate-Print Loop)
- JLink : permet de créer une JRE minimale personnalisée pour une application  
*toutes les dépendances doivent être des modules*
- JDeps : permet d'obtenir des infos sur les jar/modules pour avoir plus de visibilité sur les dépendances
- Multi-Release JAR : Jar contenant plusieurs variantes d'une même classe en fonction de la JVM qui exécutera le jar
- JDeprScan : analyse statique des .class pour détecter l'usage d'API @Deprecated



Et si on regarde plus loin ?  
Java 11, 12, 13, 14 ...



# Les Switch expressions

```
switch (day) {  
    case MONDAY, FRIDAY, SUNDAY -> System.out.println(6);  
    case TUESDAY                 -> System.out.println(7);  
    case THURSDAY, SATURDAY      -> System.out.println(8);  
    case WEDNESDAY              -> System.out.println(9);  
}
```

```
int numLetters = switch (day) {  
    case MONDAY, FRIDAY, SUNDAY -> 6;  
    case TUESDAY                 -> 7;  
    case THURSDAY, SATURDAY      -> 8;  
    case WEDNESDAY              -> 9;  
};
```





# Text Blocks

```
String html = ""  
    <html>  
        <body>  
            <p>Hello, world</p>  
        </body>  
    </html>  
    "";
```



# Pattern Matching

```
if (obj instanceof String) {  
    String s = (String) obj;  
    // use s  
}
```

```
if (obj instanceof String s) {  
    // can use s here  
}
```



# NullPointerException

```
Exception in thread "main" java.lang.NullPointerException:  
    Cannot assign field "i" because "a" is null  
    at Prog.main(Prog.java:5)
```

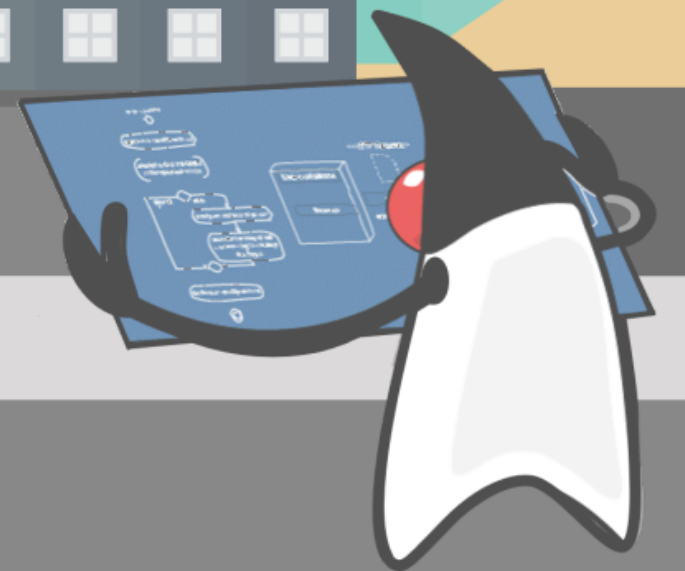


# Records

```
package examples;  
  
record Person (String firstName, String lastName) {}
```



# Les projets



# Projet Amber - Concise Method Bodies

```
ToIntFunction<String> lenFn = (String s) -> { return s.length(); };  
ToIntFunction<String> lenFn = (String s) -> s.length();  
ToIntFunction<String> lenFn = String::length;  
int length(String s) = String::length
```



# Projet Amber - Enhanced Enums

```
enum Primitive<X> {  
    INT<Integer>(Integer.class, 0) {  
        int mod(int x, int y) { return x % y; }  
        int add(int x, int y) { return x + y; }  
    },  
    FLOAT<Float>(Float.class, 0f) {  
        long add(long x, long y) { return x + y; }  
    }, ... ;  
  
    final Class<X> boxClass;  
    final X defaultValue;  
  
    Primitive(Class<X> boxClass, X defaultValue) {  
        this.boxClass = boxClass;  
        this.defaultValue = defaultValue;  
    }  
}
```



# Projet Amber - Enhanced Enums

```
// File system path  
"C:\\Dev\\file.txt"  
`C:\\Dev\\file.txt`  
  
// Regex  
"\\d+\\.\\d\\d"  
`\\d+\\.\\d\\d`  
  
// Multi-Line  
"Hello\\nWorld"  
`Hello  
World`
```





# Projet Valhalla - Value Object

```
package examples;  
  
value class Point {  
    int x;  
    int y  
}
```



# Projet Valhalla - Génériques spécialisés

```
List<int>
```



# Projet Loom - Fiber et Continuation

Le Projet Loom permet de prendre en charge un modèle d'accès concurrentiel léger à haut débit en Java.

Une **Continuations** est une séquence d'instructions qui peut céder et être reprise.

Une **Fiber** permet à une tâche de suspendre et de reprendre dans le runtime Java au lieu du noyau.



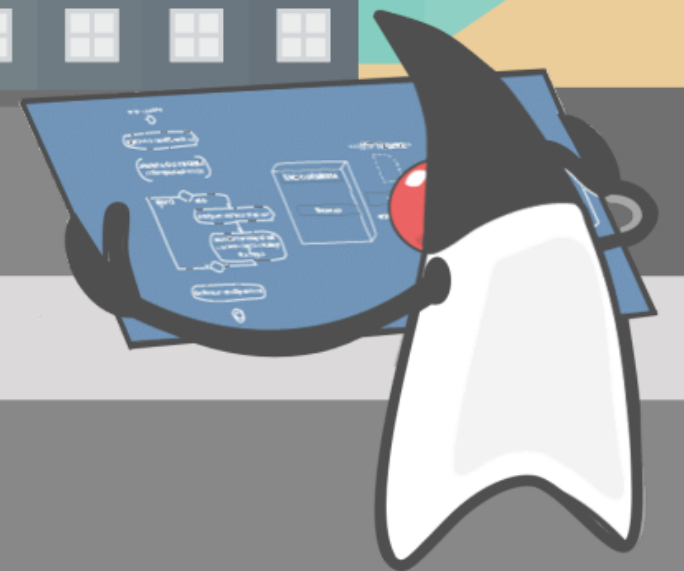
## JDK 11

- sorti de Java FX Java EE and CORBA Modules
- Running Java File with single command
- Java String Methods Files
- Flight Recorder
- HTTP Client

avec docker...



# REX : Migration de FuSIIon vers Java 11



# L'application FuSillon

## Frontend

RECHERCHER


REFERENTIEL

STATISTIQUES

ADMINISTRER

MON PROFIL

gaming



admin admin  
adminTest@sii.fr

Objectifs +

Missions +

Objectifs finis

Missions terminées

admin admin | adminTest@sii.fr

Mes évaluations

Angular 5 Junior 4	C++ Junior 3	cobol Junior 1	CSS 3 Junior 3	Delphi Junior 1
HTML5 Junior 5	Java 6 Junior 3	Java 7 Junior 4	Java 8 Junior 4	PHP Junior 1

Echelle notation

0 : Ne connaît pas

1 : Connait un peu

2 : Connait les bases

3 : Autonome

4 : Connait bien

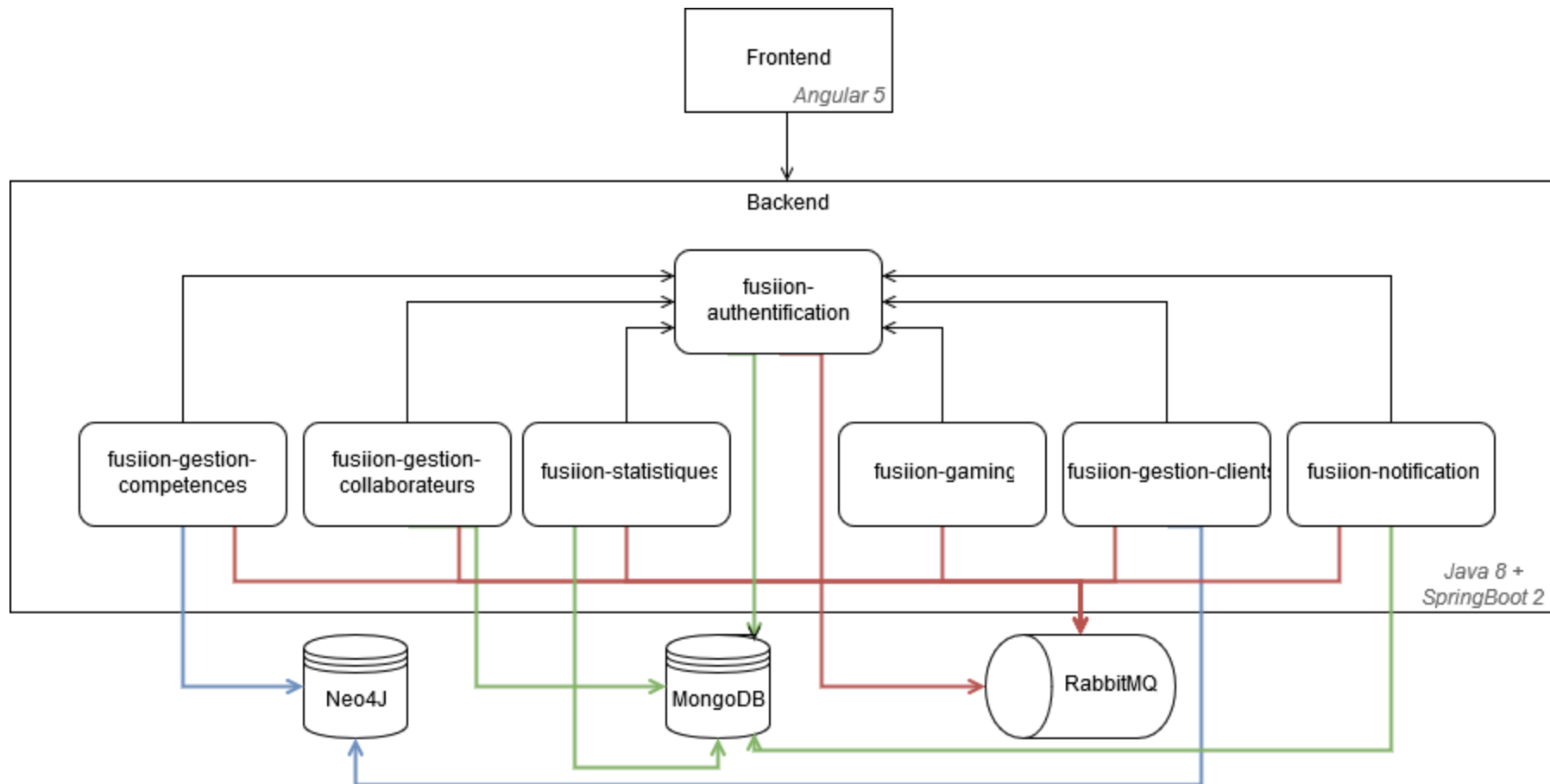
5 : Expert

# Planning de la migration

- On suit la recommandation oracle : run → build → modularize
  - **Run** : on ne recompile pas notre code, on lance juste nos jar en version Java 8 sur une JVM 11
  - **Build** : on ne modifie pas le code mais on compile via un JDK 11
  - **Modularize** : On bascule vers des modules nos différents jar
- La modularisation est indiquée comme une étape optionnelle mais recommandée

# L'application FuSIlon

## Backend





# FuSllon : Run on JVM 11

## Les étapes :

- Détection des modules Java nécessaires au fonctionnement avec JDeps
- Changement de JVM dans les Dockerfile
- Mise à jour de la commande de lancement pour indiquer les modules nécessaires à l'application

# FuSIlon : Run on JVM 11

## Dans la pratique : la commande JDeps

```
$ jdeps target/fusiion-authentication-1.0-SNAPSHOT-exec.jar
+fusiion-authentication-1.0-SNAPSHOT-exec.jar -> java.base
+fusiion-authentication-1.0-SNAPSHOT-exec.jar -> java.logging
+fusiion-authentication-1.0-SNAPSHOT-exec.jar -> java.xml.bind
+fusiion-authentication-1.0-SNAPSHOT-exec.jar -> not found
  fr.sii.atlantique.fusiion.fusiion_authentication ->
↳      java.lang                                java.base
  fr.sii.atlantique.fusiion.fusiion_authentication ->
↳      org.springframework.boot                  not found
  fr.sii.atlantique.fusiion.fusiion_authentication ->
↳      org.springframework.boot.autoconfigure    not found
  fr.sii.atlantique.fusiion.fusiion_authentication ->
↳      org.springframework.cloud.openfeign        not found
  ...
```



*Toutes les dépendances ne sont pas toujours détectées*

# FuSIlon : Run on JVM 11

## Dans la pratique : le Dockerfile

```
-FROM openjdk:8-jre-alpine
+FROM openjdk:11-jre-alpine

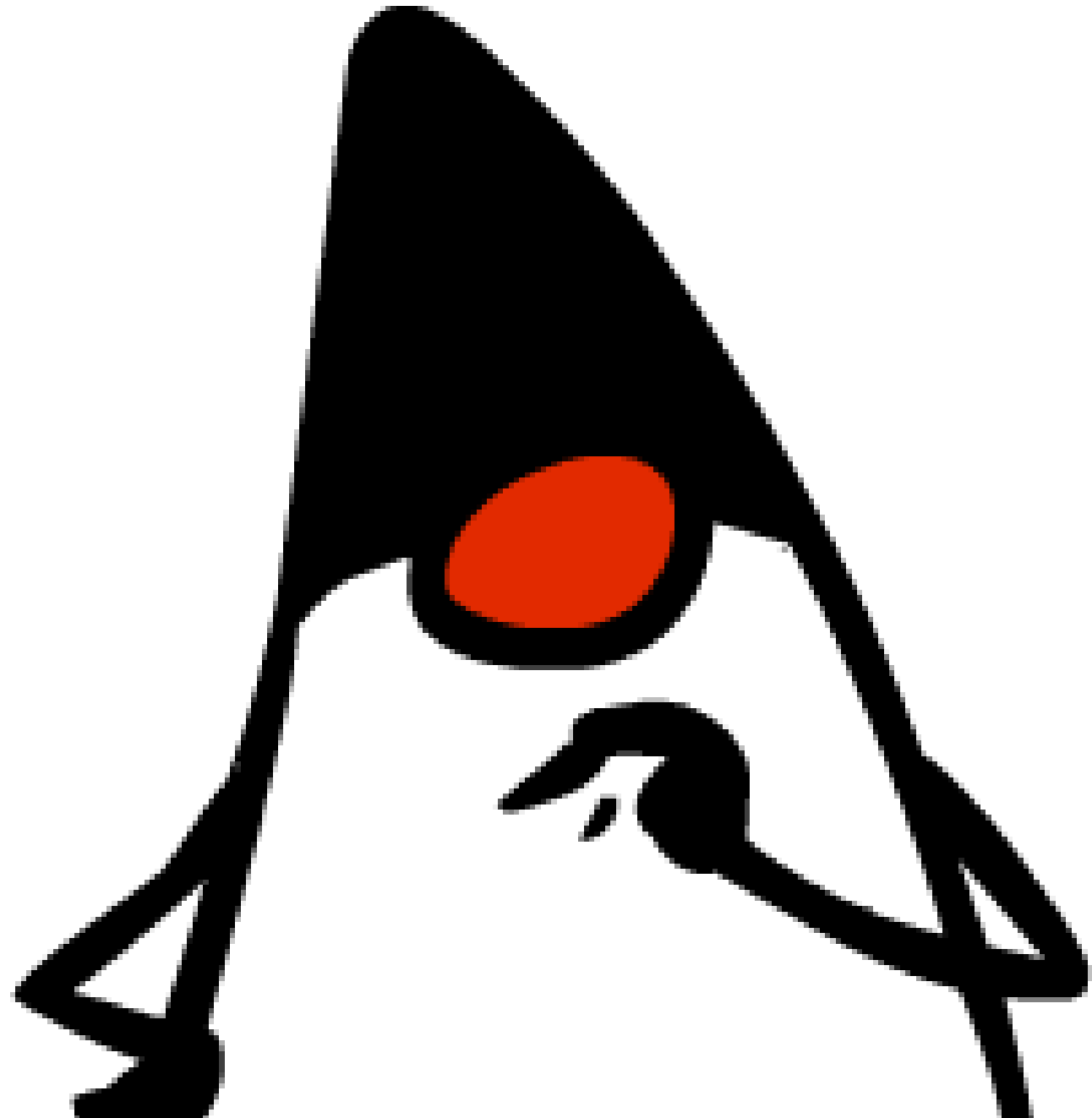
ADD /opt/${project.build.finalName}-exec.jar /opt/
EXPOSE 8080

ENTRYPOINT [ "java", "-XX:+UnlockExperimentalVMOptions",
-           "-XX:+UseCGroupMemoryLimitForHeap",
            "-XX:+UseG1GC",
+           "--add-modules", "java.base", \
+           "--add-modules", "java.logging", \
+           "--add-modules", "java.xml.bind", \
            "-jar", "/opt/${project.build.finalName}-exec.jar",
            "--logging.path=/var/logs" ]
```

# FuSllon : Run on JVM 11

## Conclusion

- Environ 1 journée de travail
- Facile à mettre en place
- Les outils sont présents pour nous aider



# FuSllon : Compile with Javac 11

## Les étapes :

- Installer une JDK 10
- De préférence OpenJDK 10 pour prendre l'habitude de l'OpenJDK
- Mettre à jour Maven sur la dernière version (mini : 3.5.0)
- Mettre à jour les plugins qu'on utilise, en particulier :
  - maven-compiler-plugin
  - org.ow2.asm:asm
- Mettre à jour les dépendances des plugins qu'on utilise
- ajouter en dépendance dans le pom.xml les modules nécessaires

# FuSllon : Compile with Javac 11

## Dans la pratique : pom.xml

```
<properties>  
-   <maven.compiler.source>1.8</maven.compiler.source>  
-   <maven.compiler.target>1.8</maven.compiler.target>  
+   <maven.compiler.release>11</maven.compiler.release>  
</properties>
```

# FuSllon : Compile with Javac 11

## Dans la pratique : pom.xml

```
<dependencies>
+       <dependency>
+           <groupId>com.sun.activation</groupId>
+           <artifactId>javax.activation</artifactId>
+           <version>1.2.0</version>
+       </dependency>
+       <dependency>
+           <groupId>javax.xml.bind</groupId>
+           <artifactId>jaxb-api</artifactId>
+           <version>2.3.0</version>
+       </dependency>
+       <dependency>
+           <groupId>org.glassfish.jaxb</groupId>
+           <artifactId>jaxb-runtime</artifactId>
+           <version>2.3.0.1</version>
+       </dependency>
</dependencies>
```

# FuSIlon : Compile with Javac 11

## Dans la pratique : pom.xml

```
<plugins>
+   <plugin>
+       <groupId>org.apache.maven.plugins</groupId>
+       <artifactId>maven-compiler-plugin</artifactId>
+       <version>3.7.0</version>
+       <configuration>
+           <release>11</release>
+       </configuration>
+       <dependencies>
+           <dependency>
+               <groupId>org.ow2.asm</groupId>
+               <artifactId>asm</artifactId>
+               <version>6.2</version>
+           </dependency>
+       </dependencies>
+   </plugin>
```



# FuSllon : Compile with Javac 11

## Dans la pratique : pom.xml

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>docker-maven-plugin</artifactId>
...
+   <dependencies>
+     <dependency>
+       <groupId>javax.activation</groupId>
+       <artifactId>activation</artifactId>
+       <version>1.1.1</version>
+     </dependency>
+   </dependencies>
</plugin>
</plugins>
```

# FuSllon : Compile with Javac 11

## Conclusion

- Environ 2 journées de travail
- Assez facile à mettre en place
- Les outils nous aide toujours
- On doit faire attention à des éléments qui étaient jusque transparent pour nous



# FuSllon : Modularize

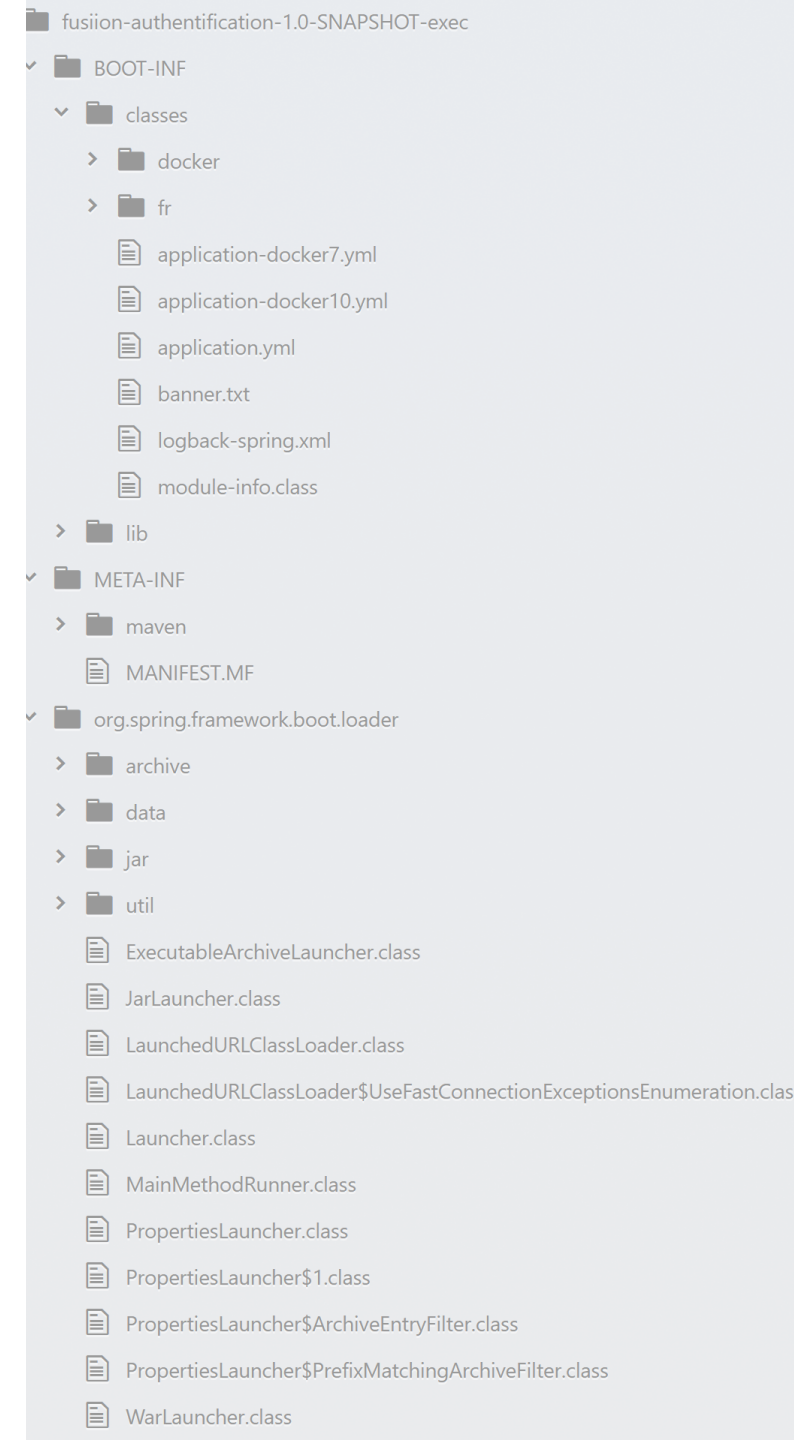
## Les étapes :

- Créer le fichier module-info.java
- Déterminer tous les modules en dépendance via le plugin Maven
- lancer l'application et l'utiliser pour déterminer si on a tout ou non
- itérer sur les 2 étapes précédentes jusqu'à ne plus avoir d'erreur

# FuSIlon : Modularize

## Le jar exécutable

- SpringBoot génère un jar contenant les dépendances en utilisant sa propre convention
- Le fichier module-info de l'application n'est pas à la racine du jar
- SpringBoot n'ajoute pas de module-info pour son loader
- Les jar SpringBoot ne sont pas des modules
- On ne peut pas utiliser JLink directement sur le jar



# FuSllon : Modularize

## Dans la pratique : maven-dependency-plugin

```
mvn compile org.apache.maven.plugins:maven-dependency-plugin:3.1.1:resolve
```

```
...  
[ERROR] /C:/dev/migration-java-10/Fusion-back/fusion-authentication/src/main/  
↳ java/fr/sii/atlantique/fusion/fusion_authentication/client/sii/  
↳ AuthentificationResultat.java:[3,22] package javax.xml.bind.annotation is not  
↳ visible  
  (package javax.xml.bind.annotation is declared in the unnamed module, but module  
  ↳ javax.xml.bind.annotation does not read it)  
...
```



# FuSllon : Modularize

## Dans la pratique : module-info

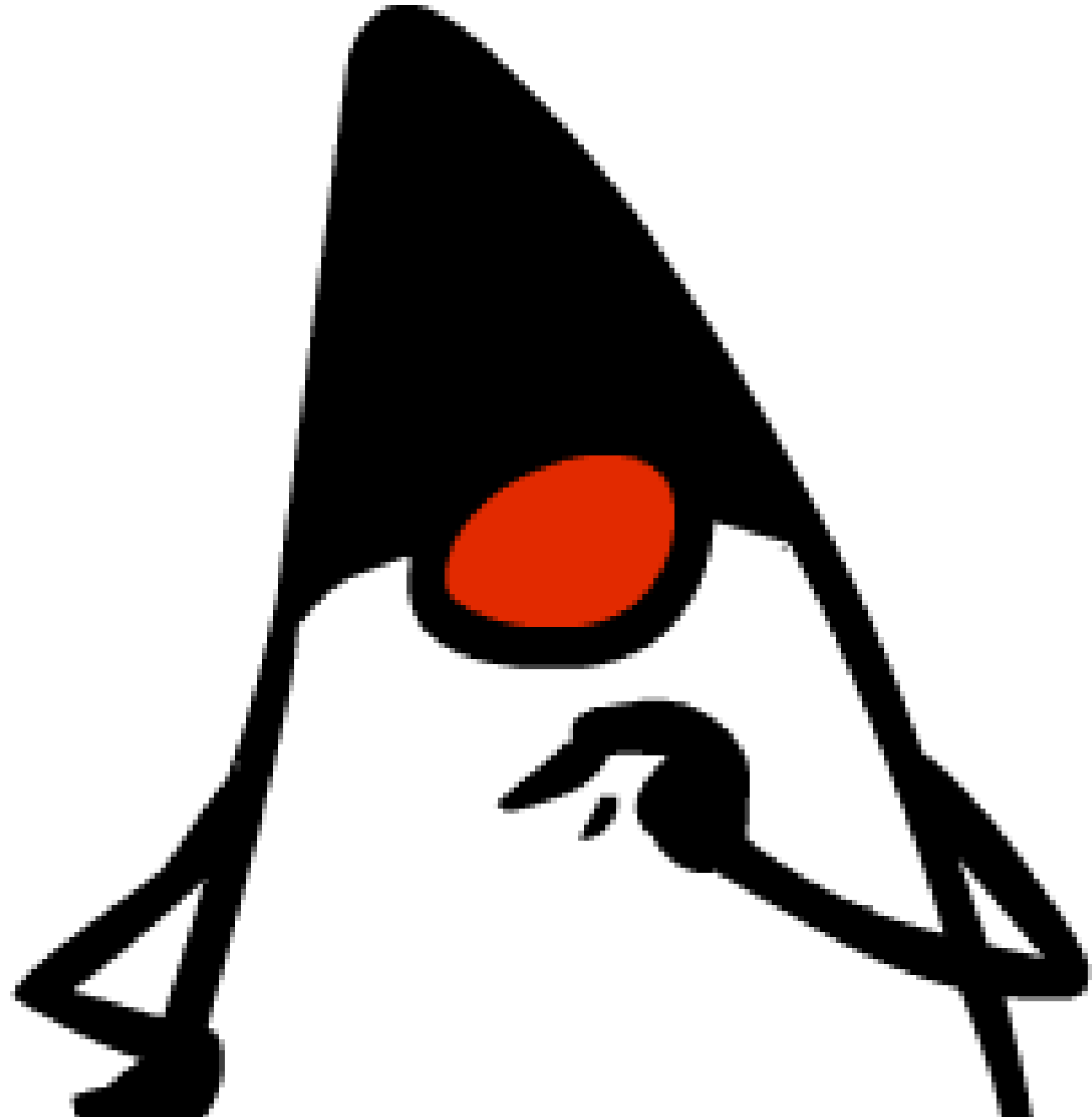
```
3  requires java.logging;
4  requires java.xml.bind;
5  requires java.validation;
6  requires java.mail;
7
8  requires gson;
9  requires jjwt;
10 requires slf4j.api;
11 requires com.fasterxml.jackson.databind;
12 requires com.fasterxml.jackson.core;
13
14 requires PBKDF2;
15
16 requires spring.amqp;
17 requires spring.beans;
18 requires spring.boot;
19 requires spring.boot.autoconfigure;
20 requires spring.boot.starter.mail;
21 requires spring.boot.starter.tomcat;
22 requires spring.cloud.openfeign.core;
23 requires spring.context;
24 requires spring.context.support; // org.springframework.mail.javamail
25 requires spring.core;
26 requires spring.data.commons;
27 requires spring.data.mongodb;
28 requires spring.rabbit;
29 requires spring.security.config;
30 requires spring.security.core;
31 requires spring.security.web;
32 requires spring.web;
33 requires spring.webmvc;
34 requires tomcat.embed.core;
35
```



# FuSllon : Modularize

## Conclusion

- Environ 7 journées de travail
- Pas de vrai difficulté mais c'est un travail laborieux
- Les outils nous aide assez peu
- Pas de gain visible dans des projets Spring, Hibernate, etc.



## FuSllon : Bonus - Custom JRE

### Pourquoi ajouter cette étape ?

- Permet de n'avoir que les modules de Java dont on a besoin
- Permet d'avoir un JRE plus léger
- Permet d'avoir des images Docker moins volumineuses



## FuSIlon : Bonus - Custom JRE

Image	openjdk:8-jre-alpine	openjdk:10-jre
fusiion/fusiion-authentification	128MB	666MB
fusiion/fusiion-gaming	109MB	647MB
fusiion/fusiion-gestion-clients	110MB	647MB
fusiion/fusiion-gestion-collaborateurs	109MB	647MB
fusiion/fusiion-gestion-competences	110MB	647MB
	...	

## FuSllon : Bonus - Custom JRE

### Les étapes :

- Utiliser la liste des modules java utilisés par l'application et toutes ses dépendances
- Ajouter une étape à la création de l'image Docker pour générer la JRE

# FuSIlon : Bonus - Custom JRE

```
FROM debian:9-slim AS builder
RUN set -ex && \
    apt-get update && apt-get install -y wget unzip && \
+   wget https://download.java.net/java/GA/jdk11/28/GPL/openjdk-11+28_linux-x64_
+↳ bin.tar.gz -O jdk.tar.gz -nv && \
    mkdir -p /opt/jdk && tar zxvf jdk.tar.gz -C /opt/jdk --strip-components=1 && \
    rm jdk.tar.gz && rm /opt/jdk/lib/src.zip
+RUN /opt/jdk/bin/jlink \
    --module-path /opt/jdk/jmods \
    --verbose \
+   --add-modules java.base,java.logging,java.xml,java.sql,java.naming,
+↳ java.desktop,java.management,java.instrument,java.security.jgss,jdk.unsigned
    --output /opt/jdk-minimal \
    --compress 2 --no-header-files
# Second stage, add only our custom jdk distro and our app
+FROM debian:9-slim
COPY --from=builder /opt/jdk-minimal /opt/jdk-minimal
...
```

# FuSIlon : Bonus - Custom JRE

## Comparatif de la taille des images java 8 et java 11

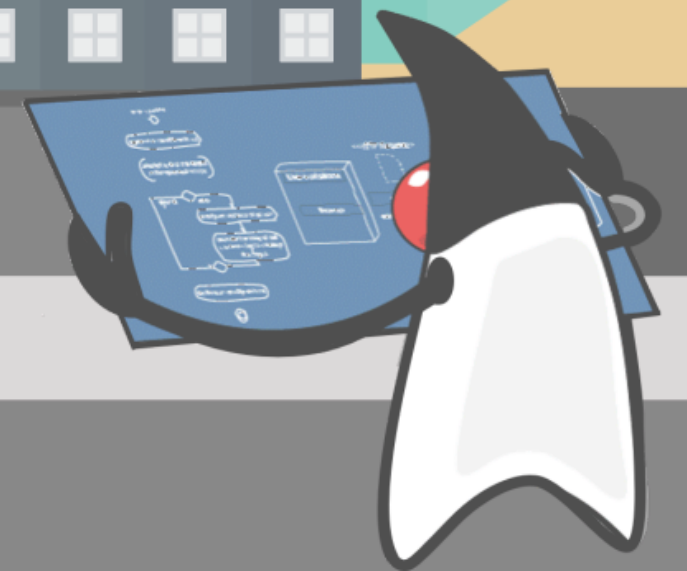
<div class="center-table">

Image	openjdk:8-jre-alpine	openjdk:10-jre	Custom JRE11
authentification	128MB	666MB	159MB
gaming	109MB	647MB	139MB
gestion-clients	110MB	647MB	140MB
gestion-	100MB	647MB	120MB

</div>



Conclusion : Java 11, on y va ?



# Conclusion : Java 11, on y va ?

## Oui... mais pas trop

- run sur une JVM 11 → oui
- build avec un JDK 11 → oui
- modulariser l'application
  - oui si votre application n'a que des dépendances modulaires
- Création d'un JRE personnalisé → oui si vous avez un vrai besoin



# Conclusion : Java 11, on y va ?

## En attendant de migrer on peut déjà

- Utiliser les outils pour éliminer les méthodes dépréciées
- Faire du tri dans nos dépendances
- Mettre à jour nos dépendances
- Fixer le nom du module dans le MANIFEST.MF
- S'assurer qu'on respecte les bonnes pratiques de développement Java



# Merci

