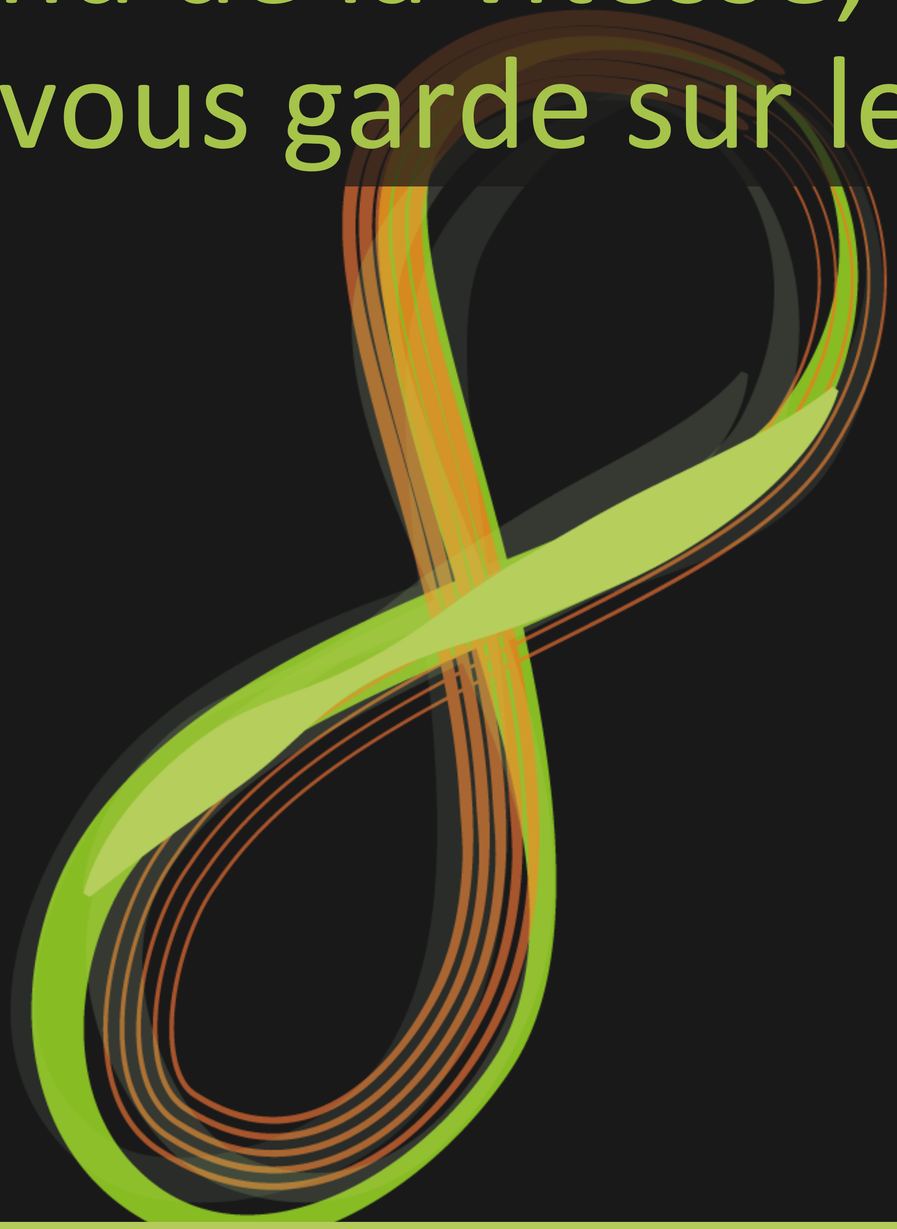
Quand Java prend de la vitesse, Apache Maven vous garde sur les rails

DEVOXX
FRANCE

Hervé Boutemy - @hboutemy
Arnaud Héritier - @aheritier



Objectif

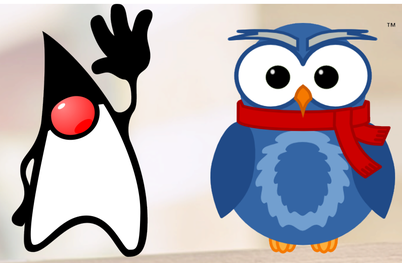
Avec Apache Maven,
appréhendez en toute sérénité les évolutions de Java

DEVOXX France

Au menu de ce Tools in Action

Apéritif

Cocktail de java duke et maven owl



Entrée

Une soupe de bytecode relevée d'une pointe d'enforcer

Plat

Une pièce d'Animal Sniffer et ses petits légumes

Dessert

Le toolchain & sa diligence de JDKs

Qui sommes nous ?

Hervé Boutemy

Committer Maven depuis 2007,

Membre du PMC Maven depuis 2009,

Membre de la Fondation Apache depuis 2011,

Maven PMC Chair depuis juillet 2014,

Touche à tout sur l'ensemble du code Apache Maven...

- Encoding, Maven Ant Tasks, Modello, maven-site-plugin, Doxia,
- Archetype, Plugin Tools,
- maven-checkstyle-plugin,Toolchains, …

DEVOXX France

Arnaud Héritier

Committer Maven depuis 2004,

Membre du PMC Maven depuis 2005,

Membre de la Fondation Apache depuis 2011,

Ne touche surtout plus au code... mais en parle beaucoup !

http://git.io/jEFs

Entendu sur *Les CastCodeurs*

www.lescastcodeurs.com @lescastcodeurs

eXo SOFTWARE FACTORY MANAGER

De quoi parle-t'on ?

Apache Maven et vous

Qui utilise Maven ?

Quelle version utilisez-vous majoritairement ?

- 3.3 ?
- 3.2 ?
- 3.1?
- 3.0 ?
- < 3.0 ?

On va vous apprendre à jongler

Java a 20 ans

Version	Première publication
JDK Alpha and Beta	1995
JDK 1.0	January, 1996
JDK 1.1	February, 1997
J2SE 1.2 (playground)	December, 1998
J2SE 1.3 (kestrel)	May, 2000
J2SE 1.4 (merlin)	February, 2002
J2SE 5.0 (tiger)	September, 2004
Java SE 6 (mustang)	December, **2006**
Java SE 7 (dolphin)	July, **2011**
Java SE 8	March, **2014**
Java SE 9	Early **2016**

Jongler entre les différentes versions de Java

Pour tirer profit des **nouveautés** offertes par Java

Pour assurer la **compatibilité** par rapport à la cible de production

Java et vous

Qui utilise **majoritairement**

La version 9 en développement ?	En production ?
La version 8 en développement ?	En production ?
La version 7 en développement ?	En production ?
La version 6 en développement ?	En production ?
La version <6 en développement ?	En production ?

Java et vous

Qui utilise sur son **poste de développement plusieurs versions** de Java et en change régulièrement ?

Qui utilise une **version de Java** en **développement différente** de celle de **production**?

Le dilemne du choix du JDK

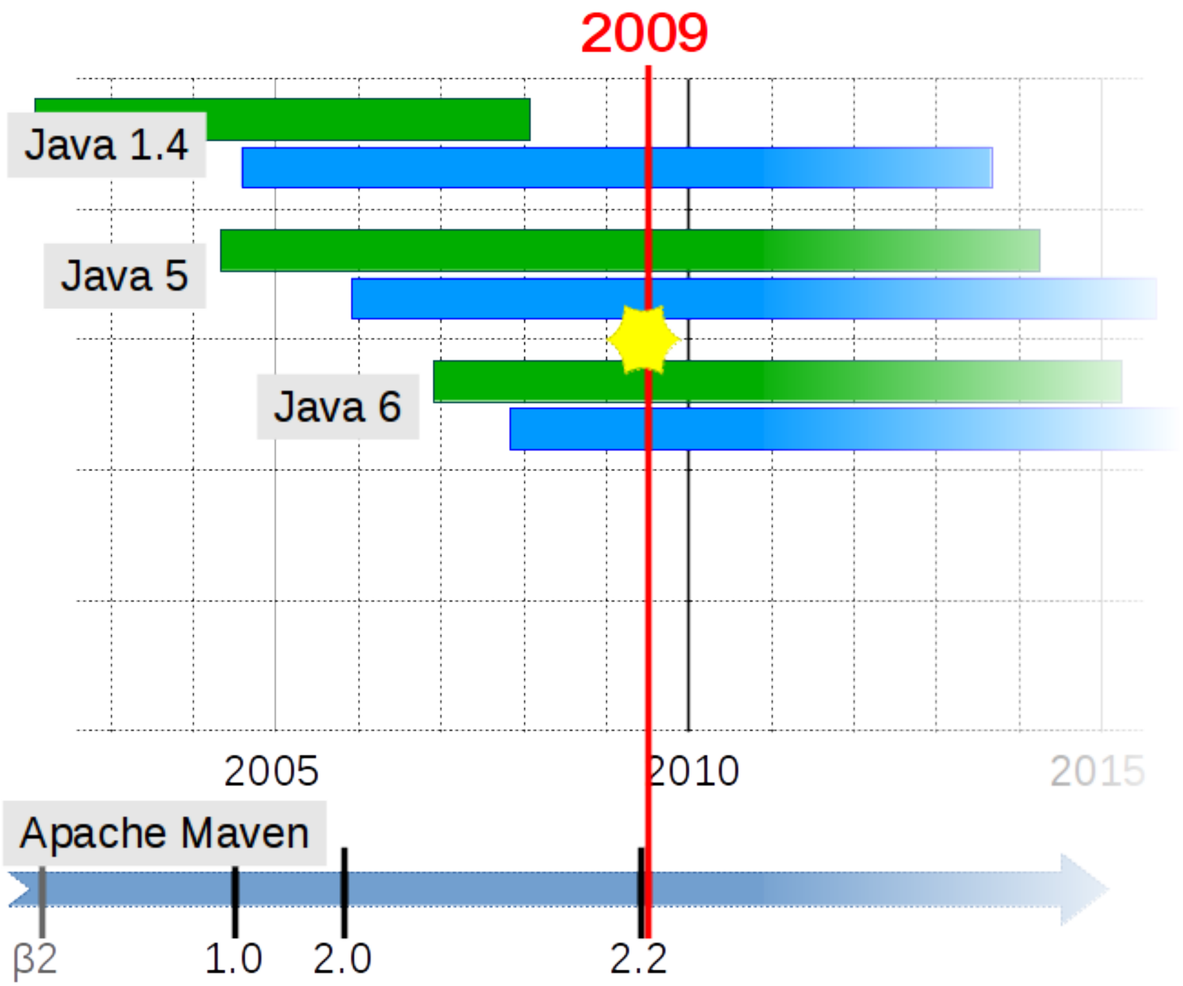
Tentation du développeur : outils de build récents, avec plus de features, nécessitant un JDK récent

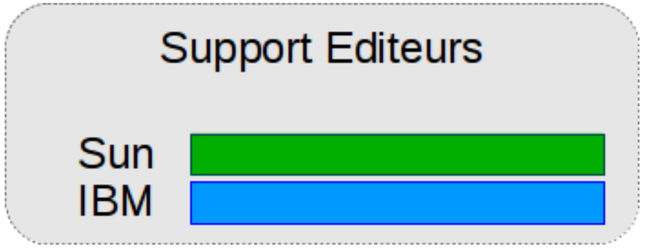
Exigence du manager : **garantir** la compatibilité avec l'environnement d'exécution cible : JRE souvent ancien (et parc homogène ?)

Plusieurs stratégies :

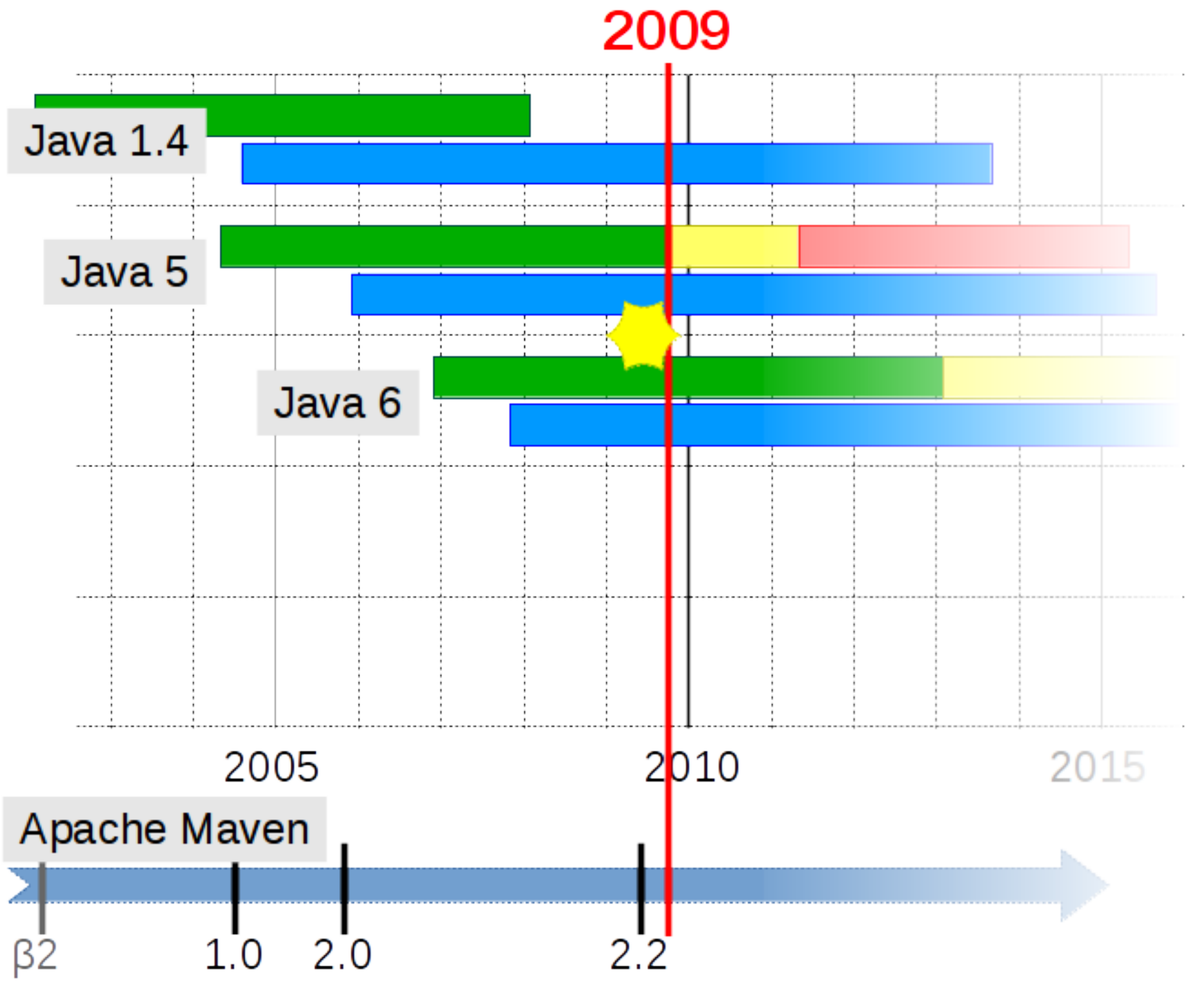
- **Conservateur** : JDK = min(JRE de toutes les applis) + vieux outils associés…
- **Courageux** : switch de JDK et outils associés à chaque appli
- **Joueur (inconscient ?)** : JDK récent
- **Sérieux** : JDK récent + CI et tests approfondis, avec bonne couverture
- **Malin** : Maven + quelques configurations

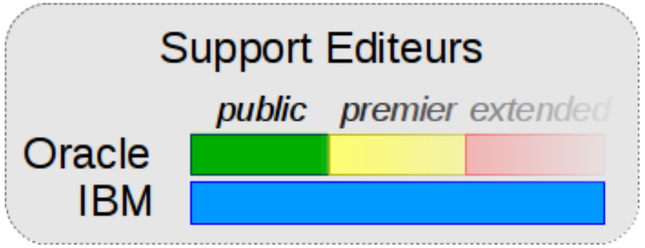
Roadmap Java & Apache Maven en 2009



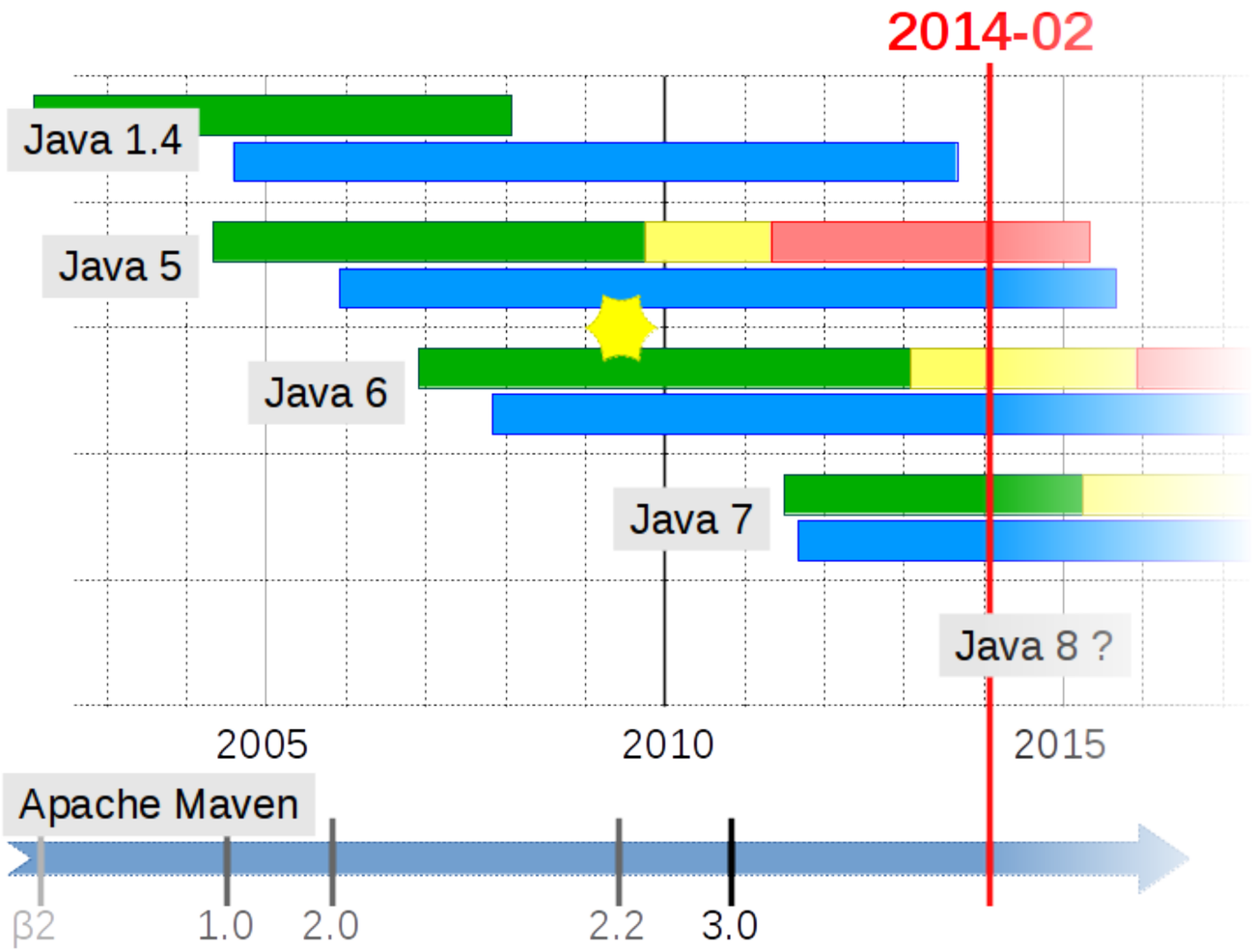


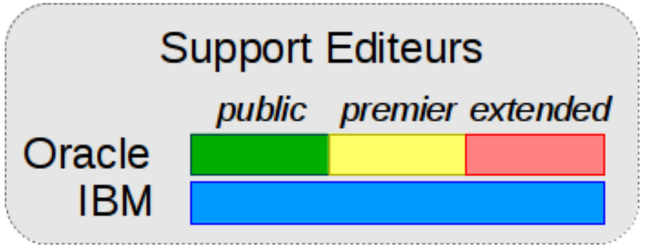
Roadmap Java ... en 2009 ...un rien plus tard...



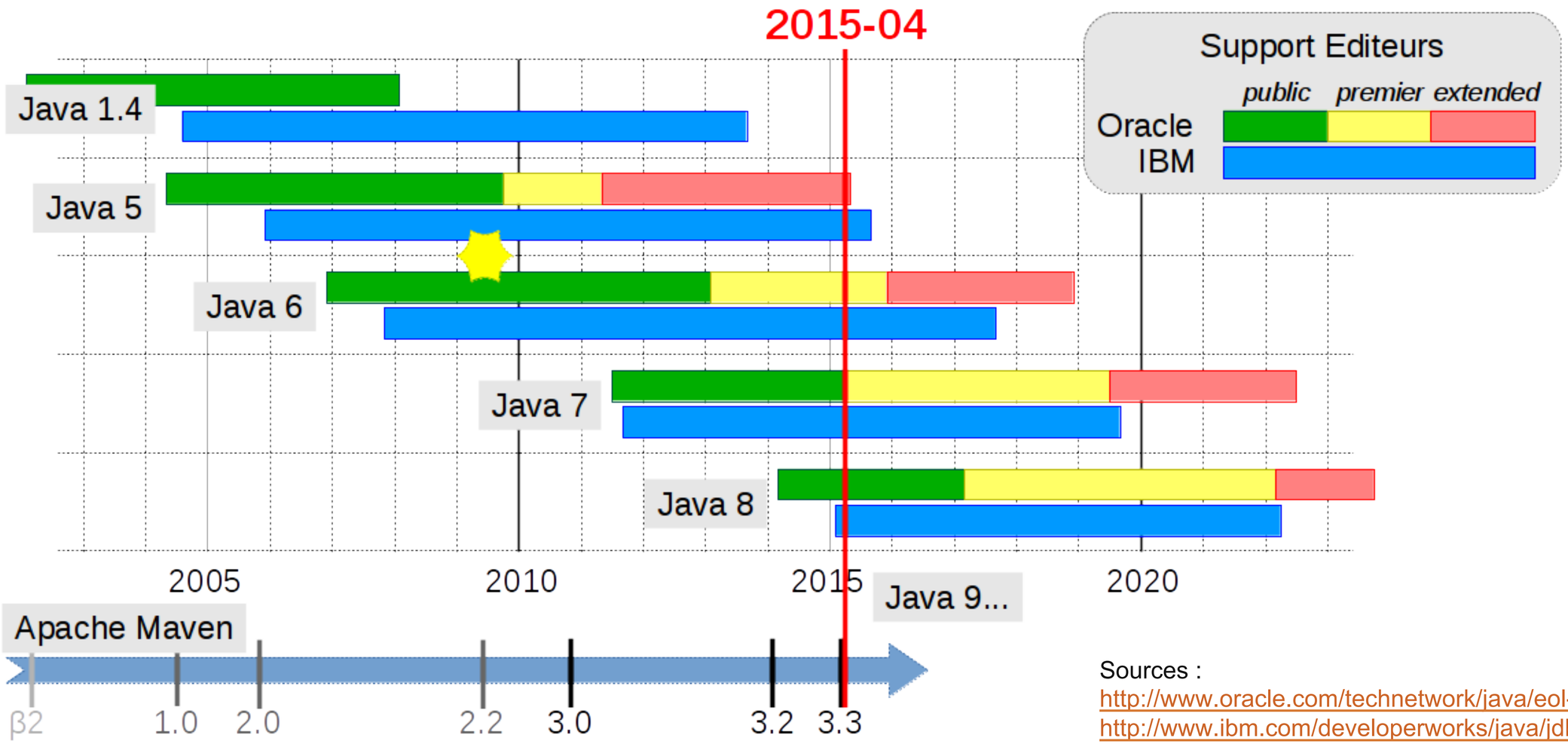


Roadmap Java & Apache Maven début 2014





Roadmap Java & Apache Maven aujourd'hui



Sources :
http://www.oracle.com/technetwork/java/eol-135779.html
http://www.ibm.com/developerworks/java/jdk/lifecycle/
http://maven.apache.org/docs/history.html

Version du bytecode

Version du bytecode

Format fichier .class
- 4 octets : magic number
- 2 octets : version mineure
- 2 octets : version majeure
- …

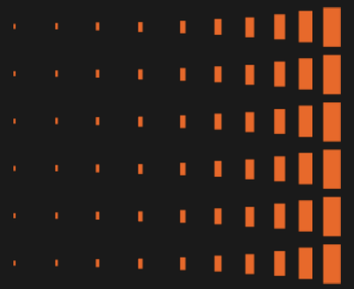
Java 8 = 52 (0x34)
Java 7 = 51 (0x33)
Java 6 = 50 (0x32)
Java 5 = 49 (0x31)
Java 1.4 = 48 (0x30)
Java 1.3 = 47 (0x2F)
Java 1.2 = 46 (0x2E)
Java 1.1 = 45 (0x2D)

Compatibilité binaire ascendante

- JVM exécute du bytecode plus ancien
- mais pas du bytecode plus récent, sinon…

```
java.lang.UnsupportedClassVersionError
```

Demo



Javac & version du bytecode

Javac

- par défaut, version bytecode = version du JDK utilisé
- `-target` : fixe version du bytecode

Maven & version du bytecode

Maven permet de contrôler facilement la version de bytecode du **build du projet**

- par défaut, maven-compiler-plugin fixe `-target` à `1.5`

=> indépendant du JDK utilisé

- paramètre `target` du maven-compiler-plugin
  - configuration `plugin` ou `pluginManagement`
  - plus compact : property `maven.compiler.target`

Maven & version du bytecode

Maven permet de contrôler facilement la version de bytecode des **dépendances du projet**

- règle `enforceBytecodeVersion` du maven-enforcer-plugin

Animal Sniffer

Demo

Animal Sniffer – Pourquoi ?

Pour vérifier qu'un code donné **respecte les signatures d'une API**

Même si l'usage le plus connu est le contrôle par rapport aux APIs du JDK, Animal Sniffer est **générique** et peut être utilisé pour n'importe quelle API, à condition d'en générer une signature

Animal Sniffer

Exemple

- Même en compilant avec un JDK 8, notre code ne doit utiliser que les APIs de Java 7

Comment ?

- Existe sous 3 formes : plugin Maven, règle enforcer, tâche Ant.
- Doit être executé sur les classes compilées.

Limitation

- Il ne s'agit que d'un contrôle sur la signature des APIs.
- Cela ne couvre pas leur sémantique.

Toolchains & JDK

Maven Toolchains : l'arme ultime

Permet d'utiliser un JDK pour compiler indépendant du JRE avec lequel Maven et ses plugins s'exécutent

JDK exécution build (javac…)

JRE exécution application

JRE exécution Maven & plugins

JDK de build = cible application != JRE d'exécution Maven

- un JRE récent pour exécuter Maven et ses plugins
- à chaque application buildée, le `pom.xml` fait sélection automatique du JDK de la version exacte requise par l'application courante

Permet de jongler facilement entre les JDKs pour garantir qu'il n'y a aucun risque pour l'application ciblée (même sans CI ni tests intensifs)

Demo

Maven Toolchains

Disponible depuis Maven 2.0.9 (4/2008)

1. outils disponibles paramétrés (path…) dans un fichier `~/.m2/toolchains.xml`
2. `pom.xml` configure le `maven-toolchains-plugin` pour sélectionner les outils requis, avec les contraintes éventuelles (version, autre…)
3. les plugins « toolchain aware » utilisent les toolchains sélectionnés sans recoder les contraintes

Les outils utilisés sont paramétrés en fonction de l'environnement d'exécution, et homogènes entre plugins

Maven Toolchains & JDK

- Générique : permet de gérer n'importe quel type de toolchain
- Toolchain `<type>jdk</type>` intégré dans Maven
- Plugins « jdk-toolchain aware » :
  - m-compiler-p, m-javadoc-p, m-surefire-p, m-webstart-p, m-jarsigner-p, exec-maven-plugin, keytool-maven-plugin, …
- Des toolchains custom existent déjà :
  - protobuf, netbeans, …

Actualités récentes Toolchains

maven-toolchains-plugin 1.1 (2014/11)

- messages d'exécution plus clairs
- documentation pour écrire des types custom

Maven 3.3 (2015/3)

- `${maven.home}/conf/toolchains.xml`
- possibilité pour un plugin d'utiliser un autre toolchain que celui sélectionné par maven-toolchains-plugin

maven-jdeps-plugin (développement en cours)

- utilise `jdeps` du JDK le plus récent (JDK 8 ou 9)

Résumé

Résumé

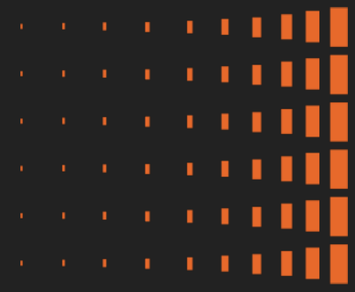
Avec la roadmap Java qui s'étoffe ces derniers temps, le **besoin de mix de versions de Java va redevenir une nécessité**, avec des risques accrus d'incompatibilités

Avec **Maven**, vous êtes équipés pour :

- **vérifier automatiquement la compatibilité** grâce à Animal Sniffer et l'Enforcer, simplement en configurant vos builds
- **utiliser le JDK adapté à chaque build** en configurant les Maven Toolchains dans vos environnements

Demo - Bonus

- Quand les optimisations du compilateur JDK 8 créent une incompatibilité binaire
- maven-jdeps-plugin



Resources

Animal Sniffer

- http://mojo.codehaus.org/animal-sniffer/

Enforcer plugin

- http://maven.apache.org/enforcer/maven-enforcer-plugin/

Toolchains

- https://maven.apache.org/guides/mini/guide-using-toolchains.html

Démos

- https://github.com/MavenDevoxxFR2015/demos

Q & R

DEVOXX™ France

Pour discuter plus longuement

Rendez vous au **BOF**

“Apache Maven, quel avenir?”

Quand ?

Demain, **jeudi 9 avril 2015, de 21h30 à 22h30**

Où ?

Salle **Neuilly 253**