Introduction à Java

dernière modification : 04/01/2023 10:44

JAVA c'est quoi ?

- Une technologie développée par SUN Microsystems™ lancée en 1995 - rachetée par Oracle en 2009
  - Un langage de programmation
    - Dans un des premiers papiers* sur le langage JAVA, James Gosling et Henry Mc Gilton le décrivent comme suit :

« *Java : a simple,* ***object-oriented****, distributed,* ***robust, secure, architecture neutral, portable, high-performance****, multithreaded, and dynamic language* »

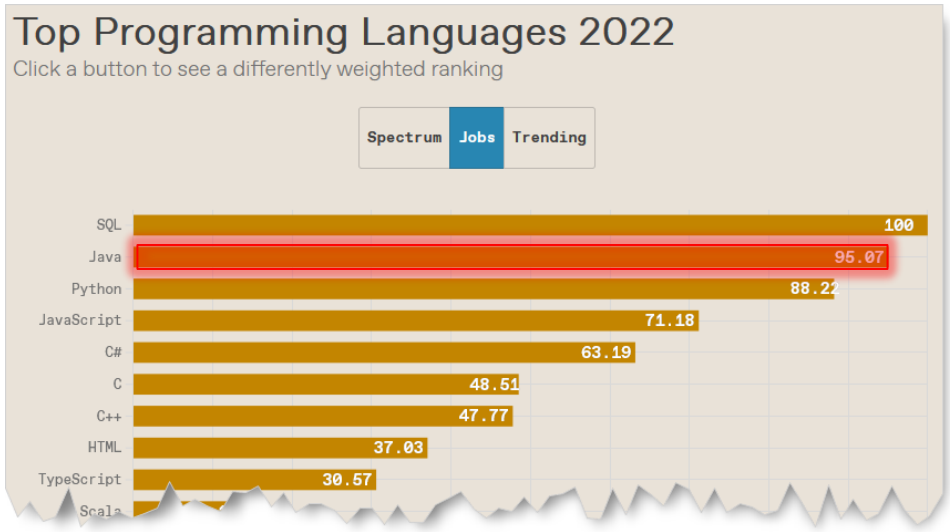
* White Paper :The Java Language Environment - James Gosling, Henry McGilton - May 1996
https://www.oracle.com/technetwork/java/langenv-140151.html

  - Une plateforme, environnement logiciel dans lequel les programmes java s'exécutent.
    - Machine Virtuelle Java (JVM)
    - Environnement d'exécution (Java Run Time Environment)

JAVA c'est quoi ?

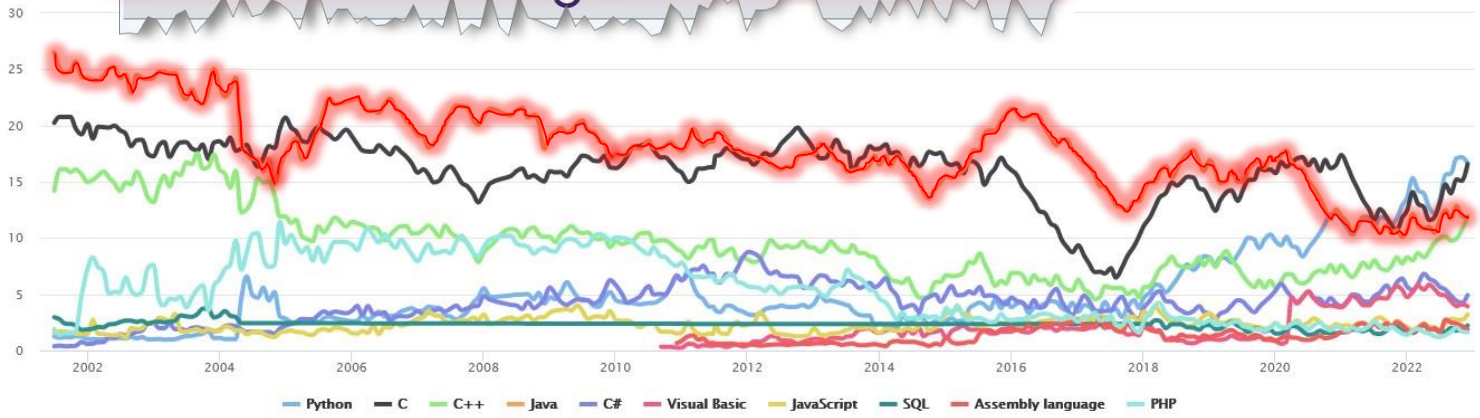
- Technologie présente dans de très nombreux domaines d'application : des serveurs d'applications aux téléphones portables et cartes à puces
- Dominante dans les année 2000, elle demeure toujours d'actualité

[The Top Programming Languages IEEE Spectrum's 2022 Ranking]

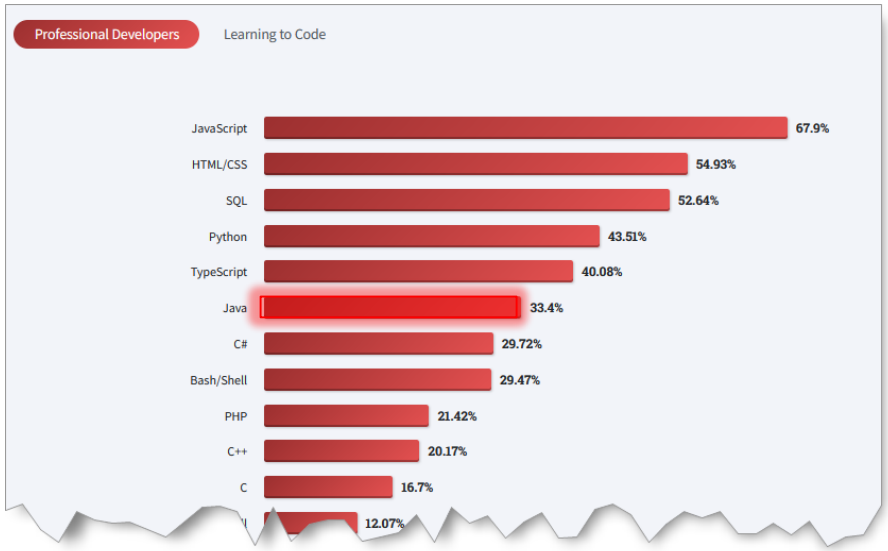


[TIOBE index Dec. 2022]

Dec 2022	Dec 2021	Change	Programming Language	Ratings	Change
1	1		Python	16.66%	+3.76%
2	2		C	16.56%	+4.77%
3	4	^	C++	11.94%	+4.21%
4	3	v	Java	11.82%	+1.70%
5	5		C#	4.92%	+1.48%



[Stackoverflow developer survey 2022]



Le langage Java

Mon premier programme Java (pas très objet…)

1 Tout code java doit être défini à l'intérieur d'une **classe**

Le code de la classe doit être enregistré dans un fichier de même nom (casse comprise) que la classe **`HelloWorld.java`**

3

```
public class HelloWorld {

    public static void main(String[] args) {
        int i = 0;

        while (i < 5) {
            System.out.println("Hello World !");
            i++;
        }

    }
}
```

Le point d'entrée pour l'exécution est la méthode `main()`

4

5

2 La description de la classe est effectuée à l'intérieur d'un bloc { }

Java est un langage **typé statiquement** (toute variable ou fonction doit être typée) contrairement à JavaScript (typage dynamique)

Compilation :
`javac HelloWorld.java`

`HelloWorld.java`

javac

`HelloWorld.class`

Exécution :
`java HelloWorld`

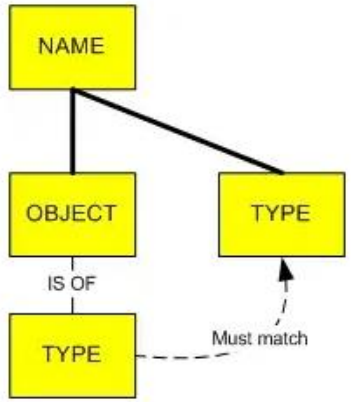
java

```
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
```

Typage statique/Typage dynamique

typage statique

- lorsqu'une variable est créée un type lui est associé
- le type de la variable ne peut être changé
- seules des valeurs compatibles avec le type de la variable peuvent lui être affectées



java

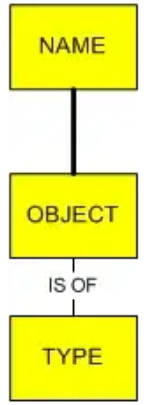
```
int number;
number = 10;
number = "hello";
number = false;
```

autres exemples :
C, C++, C#, FORTRAN…

- nécessite en général déclaration explicite des types
- vérification des types peut être faite avant l'exécution (compilation) → diminue risques d'erreurs d'exécution
- permet de produire du code plus optimisé (au moment de l'exécution car la vérification de type a déjà été effectuée)

typage dynamique

- le type d'une variable est associé à la valeur qui lui est affectée
- le type de la variable peut être changé au cours de l'exécution



JavaScript

```
let number;
number  = 10;
number = "hello";
number = false;
```

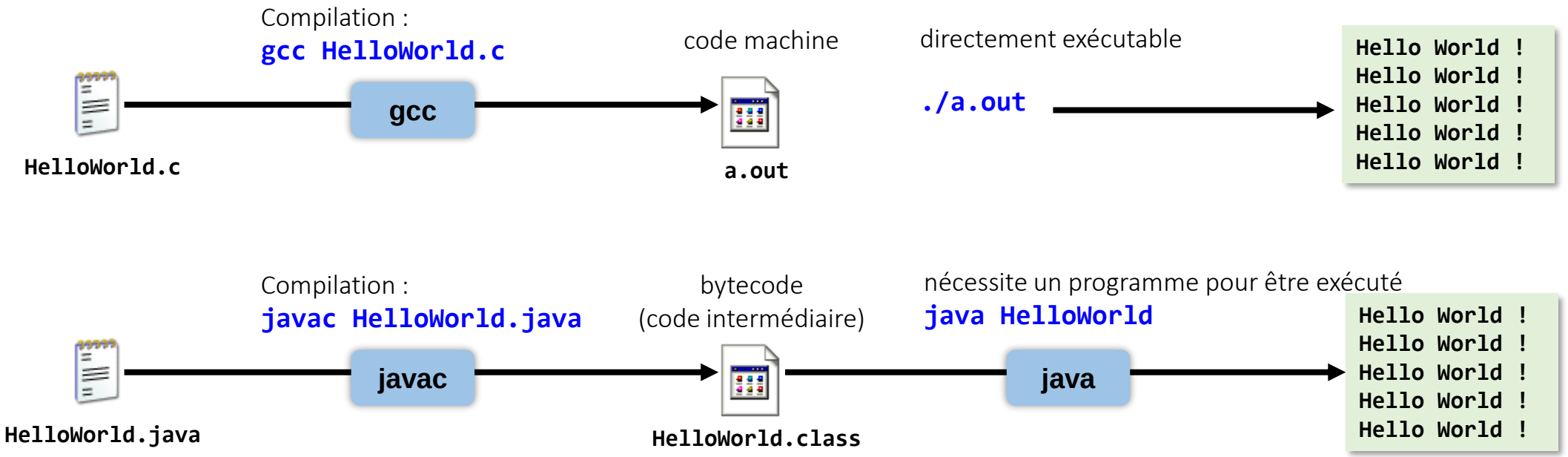
autres exemples :
Python, Ruby, PHP, Lisp…

- ne nécessite pas de déclaration explicite des types
- vérification des types faite à l'exécution → risque d'erreurs à l'exécution
- tendance à une exécution plus lente car les informations de type pour chaque variable doivent être récupérées au moment de l'exécution.

Le langage Java

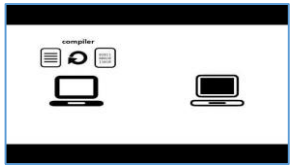
Un langage compilé / interprété

- Contrairement à d'autre langages compilé (C, FORTRAN, Pascal…) le code java compilé n'est pas directement exécutable, il est exécuté au travers d'un autre programme : `java`



https://www.youtube.com/watch?v=I1f45REi3k4

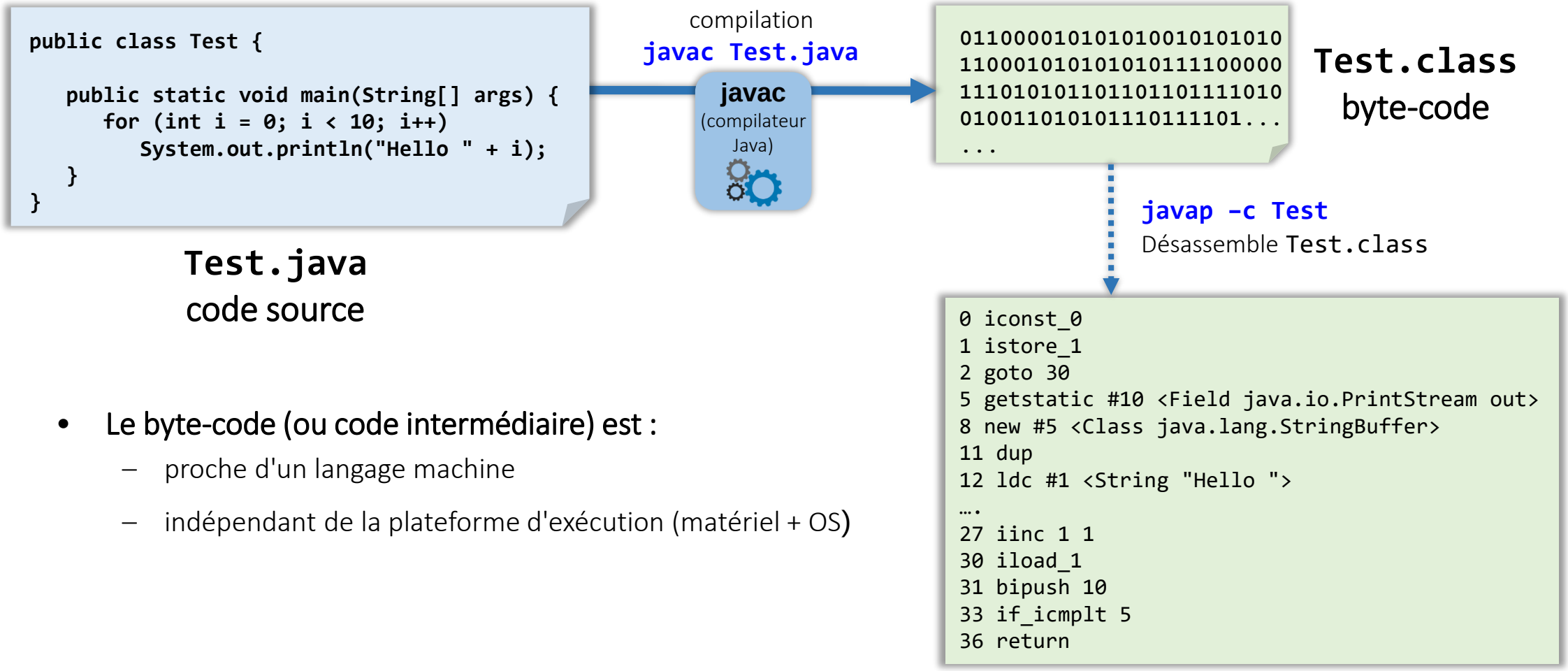
Compiler and Interpreter: Compiled Language vs Interpreted Programming Languages
durée 6 min



Le langage Java

Un langage compilé / interprété

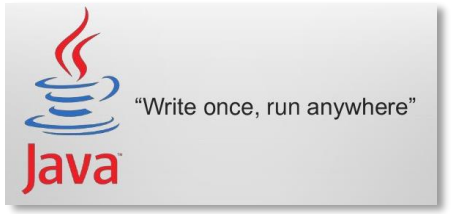
- Compilation d'un programme JAVA : génération de byte-code



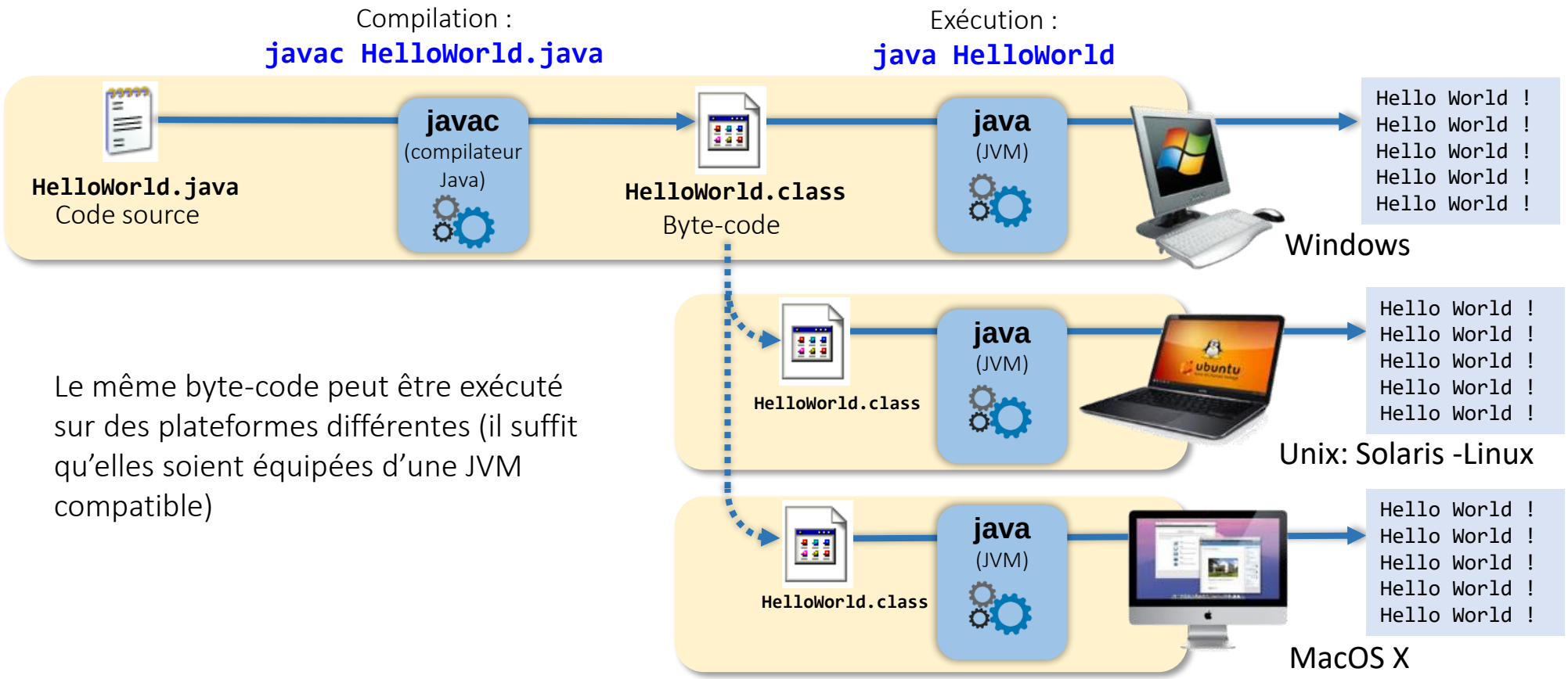
- Le byte-code (ou code intermédiaire) est :
  - proche d'un langage machine
  - indépendant de la plateforme d'exécution (matériel + OS)

La machine virtuelle Java

Exécution d'un programme Java compilé



- byte-code assure la portabilité des programmes Java
  - langage d'une Machine Virtuelle
  - à l'exécution un interpréteur (JVM) simule cette machine virtuelle



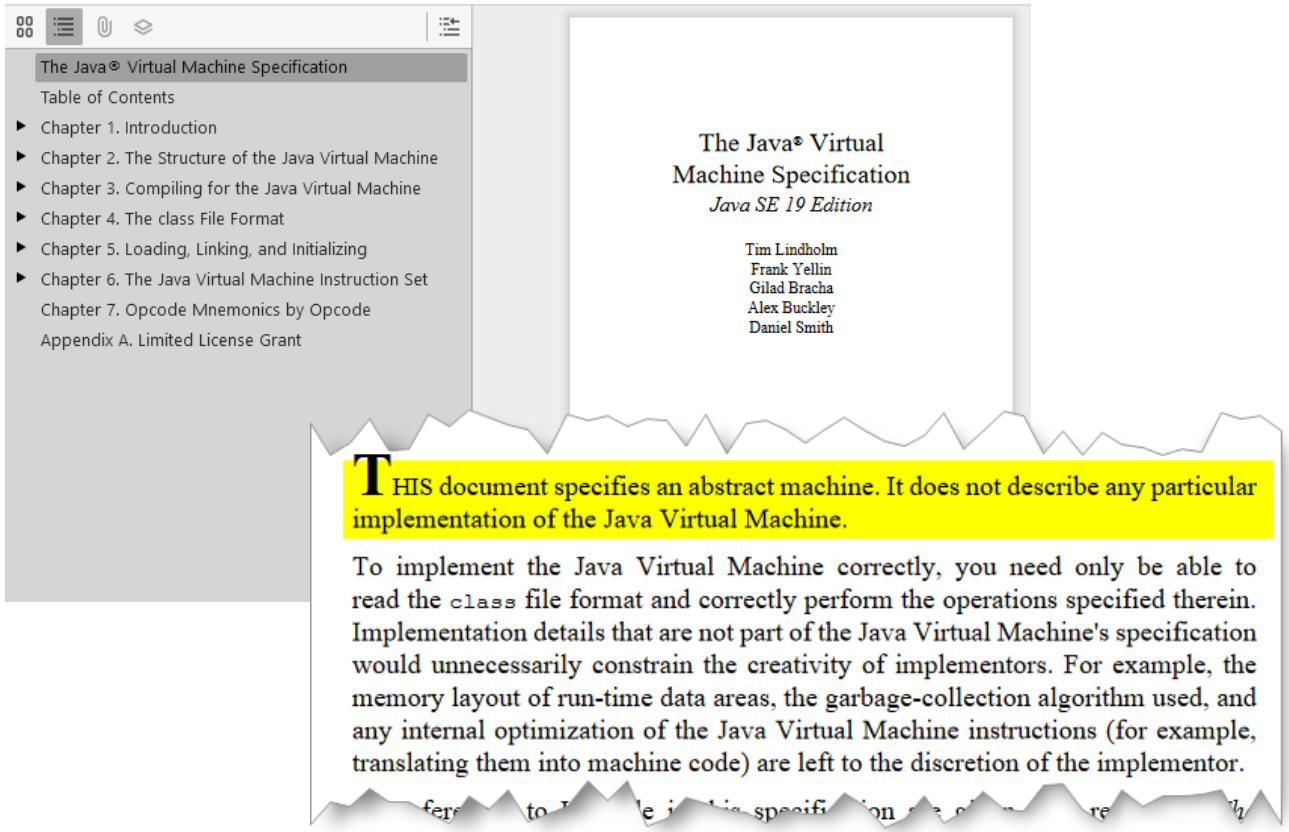
Le même byte-code peut être exécuté sur des plateformes différentes (il suffit qu'elles soient équipées d'une JVM compatible)

La machine virtuelle java

- fonctions
  - permettre d'exécuter des programmes java sur n'importe quel environnement (matériel + système d'exploitation)
  - gérer et optimiser l'utilisation de la mémoire

- une spécification de la JVM

https://docs.oracle.com/javase/specs/



- différentes implémentations (open source ou propriétaires)
  - HotStop (Open JDK) : implémentation de référence
  - GraalVM (Oracle)
  - Eclipse OpenJ9 (IBM)
  - Azul Zing
  - …

https://www.fineconstant.com/posts/comparing-jvm-performance/

La machine virtuelle java

Utilisation pour d'autres langages

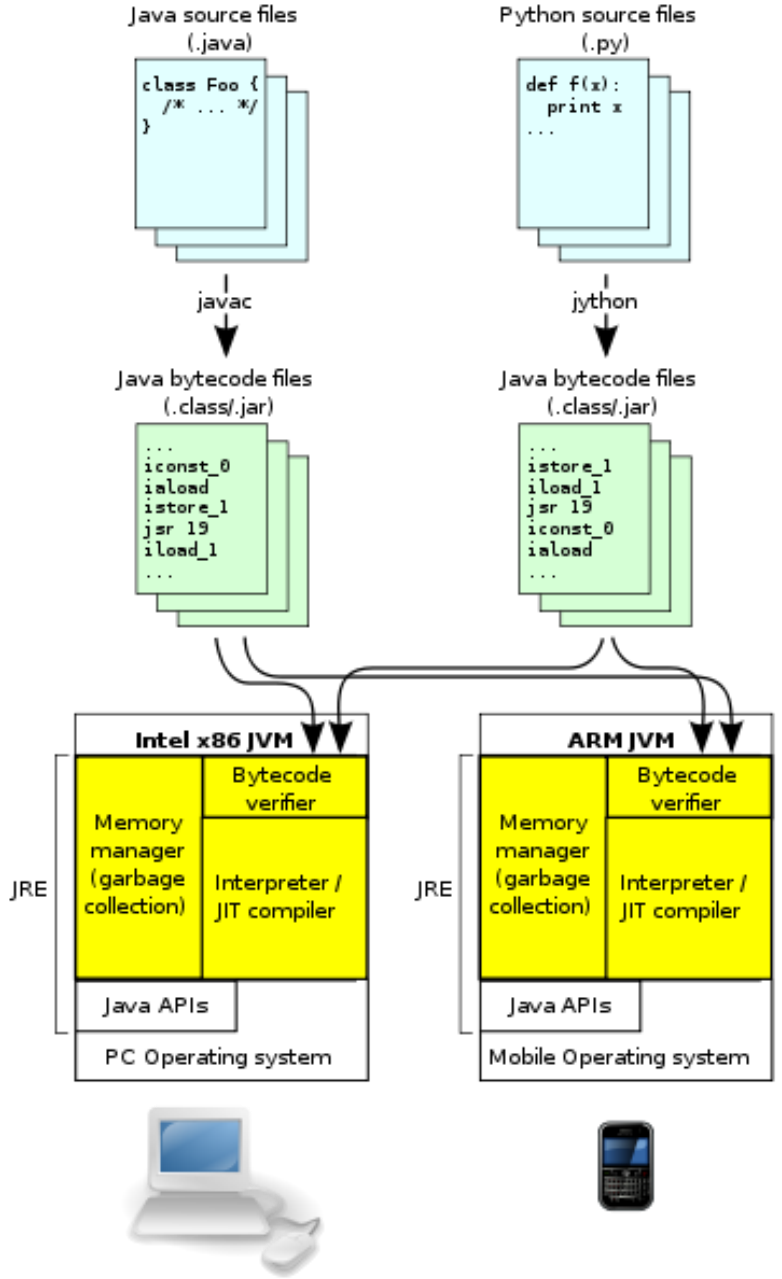
- Alors qu'elle était autrefois réservée à Java, la JVM est aujourd'hui suffisamment flexible et puissante pour prendre en charge de nombreux autres langages.

Versions of non-JVM languages

Language	On JVM
Erlang	Erjang
JavaScript	Rhino
Pascal	Free Pascal
PHP	Quercus
Python	Jython
REXX	NetRexx[3]
Ruby	JRuby
Tcl	Jacl

Languages designed expressly for JVM

Language
BBj
Clojure
Fantom
Groovy
Kotlin
MIDletPascal
Scala
Kawa



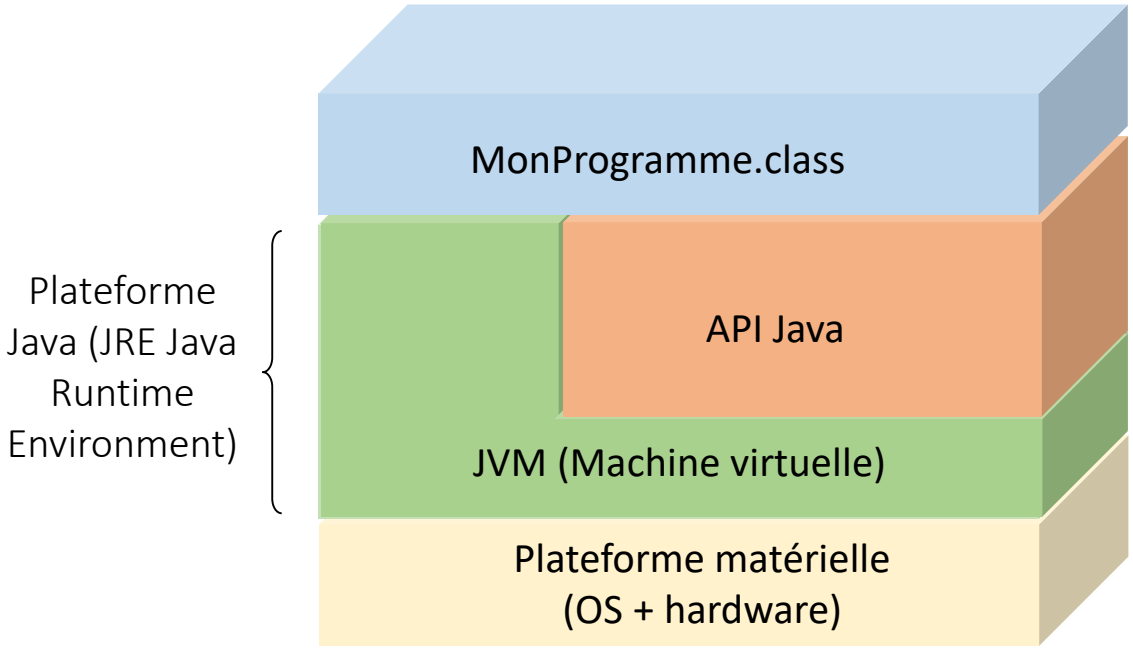
https://en.wikipedia.org/wiki/Da_Vinci_Machine

- même s'ils ne codent pas en Java, les programmeur conservent l'accès au très vaste écosystème de bibliothèques Java.

La plateforme Java

- Plateforme
  - Environnement matériel et/ou logiciel dans lequel un programme s'exécute.
    - La plupart des plateformes sont la combinaison d'un OS et du matériel sous-jacent (*MS Windows + Intel, Linux + Intel, Solaris + Sparc, Mac Os X + intel*)

- La plateforme Java est entièrement logicielle et s'exécute au dessus des plateformes matérielles

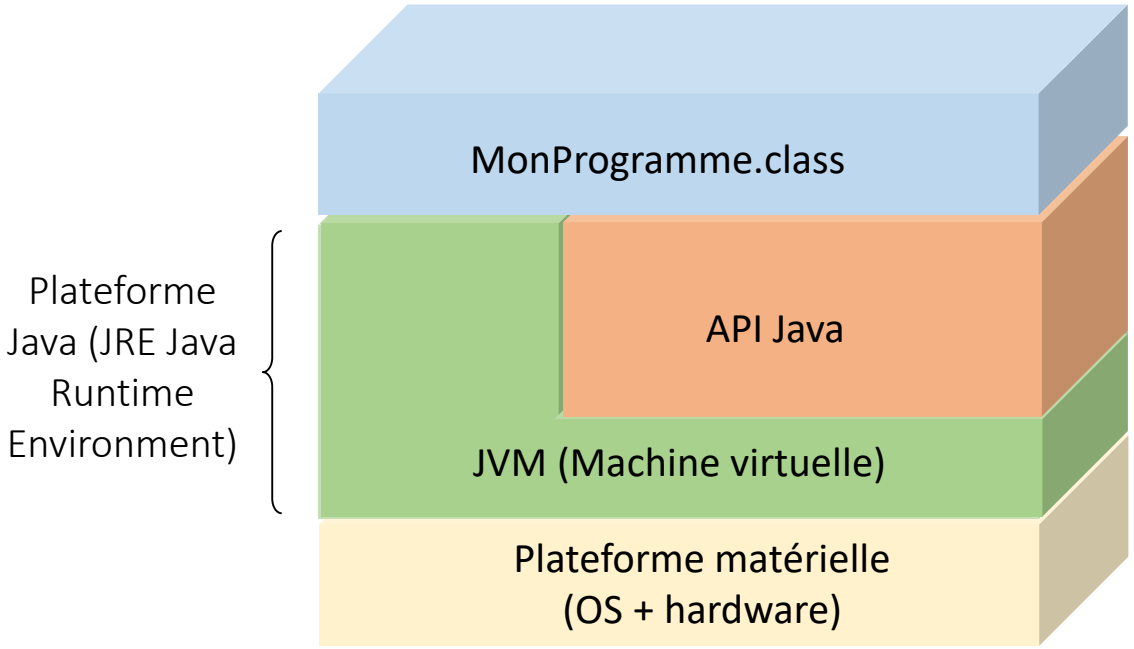


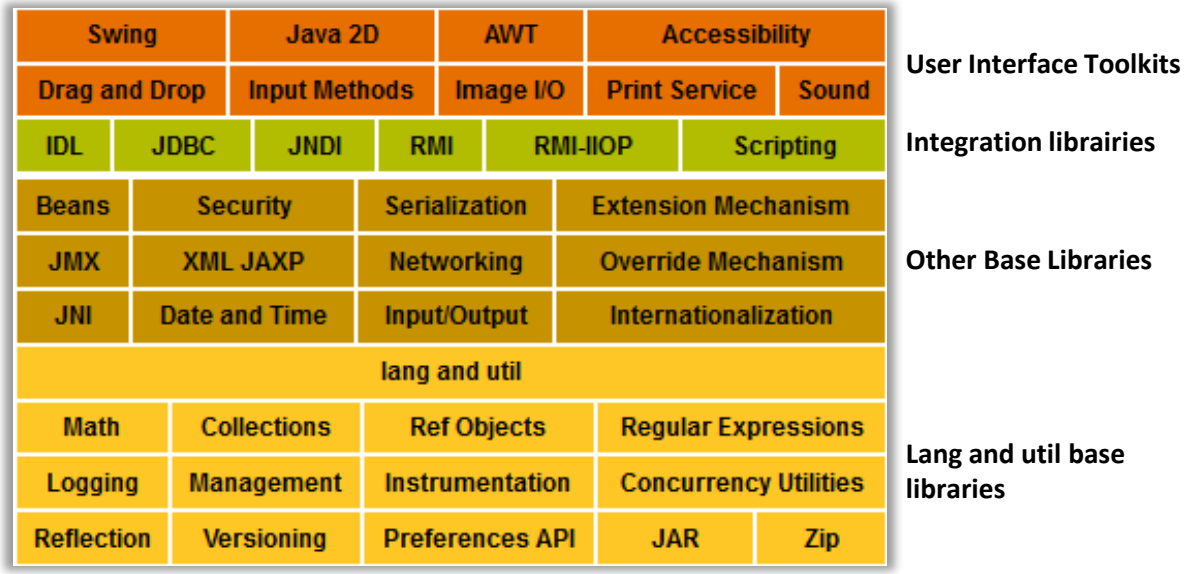
API (Application Programming Interface) Java :
Bibliothèques Java standards sur lesquelles le programmeur peut s'appuyer pour écrire son code.

La plateforme Java

API Java

- API Java
  - (très) vaste collection de composants logiciels (classes et interfaces)
  - organisée en bibliothèques (*packages*)
  - offre de nombreux services de manière standard (indépendamment de la plateforme matérielle)





Programmer en Java nécessite une bonne connaissance de l'API. Attention à la courbe d'apprentissage (learning curve) qui est peut être longue

La plateforme Java

Les différentes éditions de Java

- 3 éditions de Java

Standard Edition JSE

Fourni les compilateurs, outils, runtimes, et APIs standards pour écrire, déployer, et exécuter des applications dans la langage de programmation Java

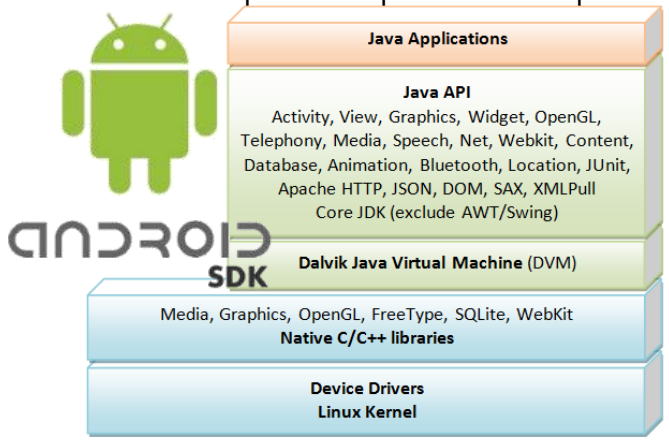
Entreprise Edition JEE

Destinée au développement d'applications « d'entreprise » *(«business applications»)* robustes et interopérables.
Construit au dessus de JSE, elle propose des APIs supplémentaires pour le développement et le déploiement d'applications distribuées et articulées autours du web.

Micro Edition JME

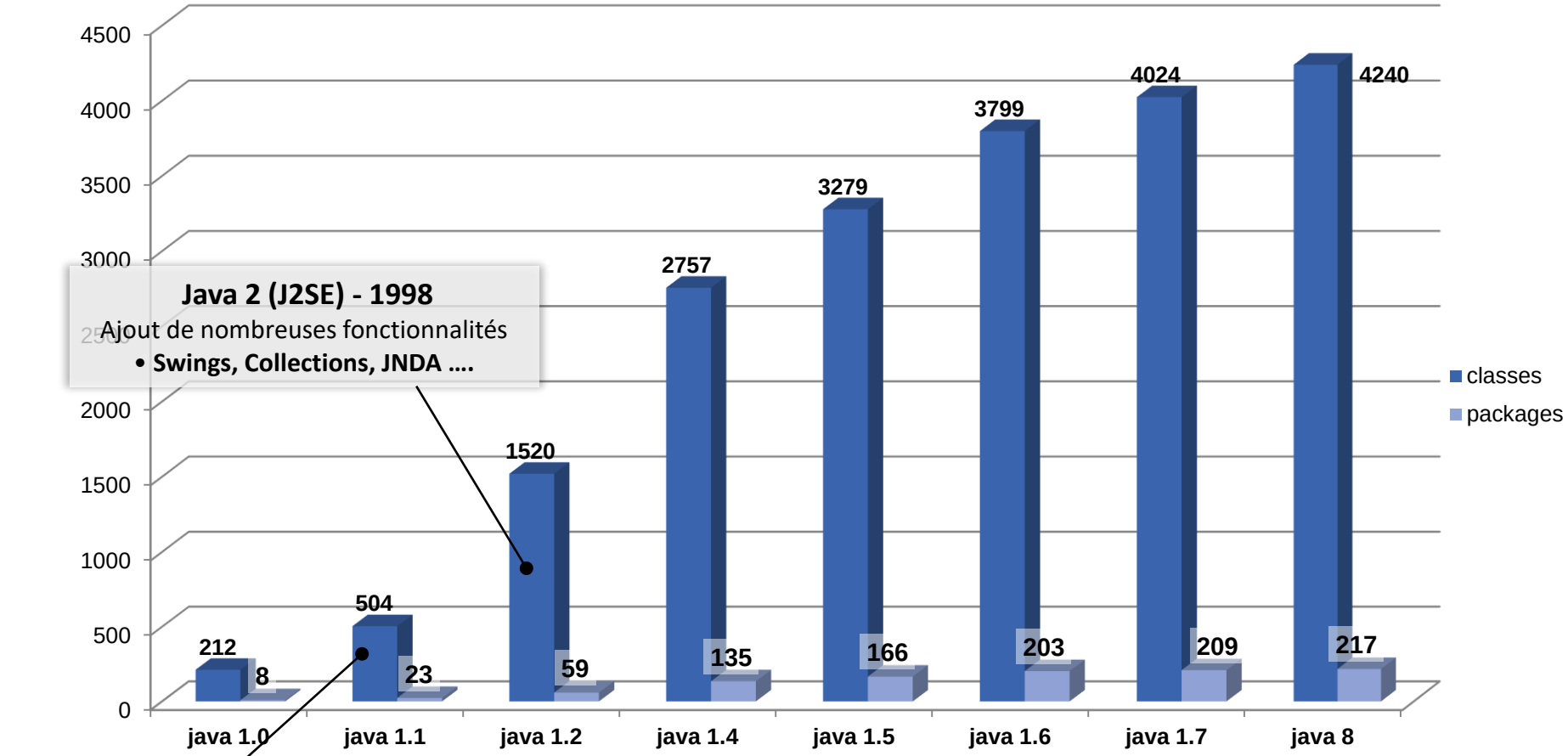
Environnement d'exécution basé sur un ensemble réduit d'API et optimisé pour les dispositifs « légers » :

- Carte à puce (smart cards)
- Téléphones mobiles
- Objets connectés



La plateforme Java JSE

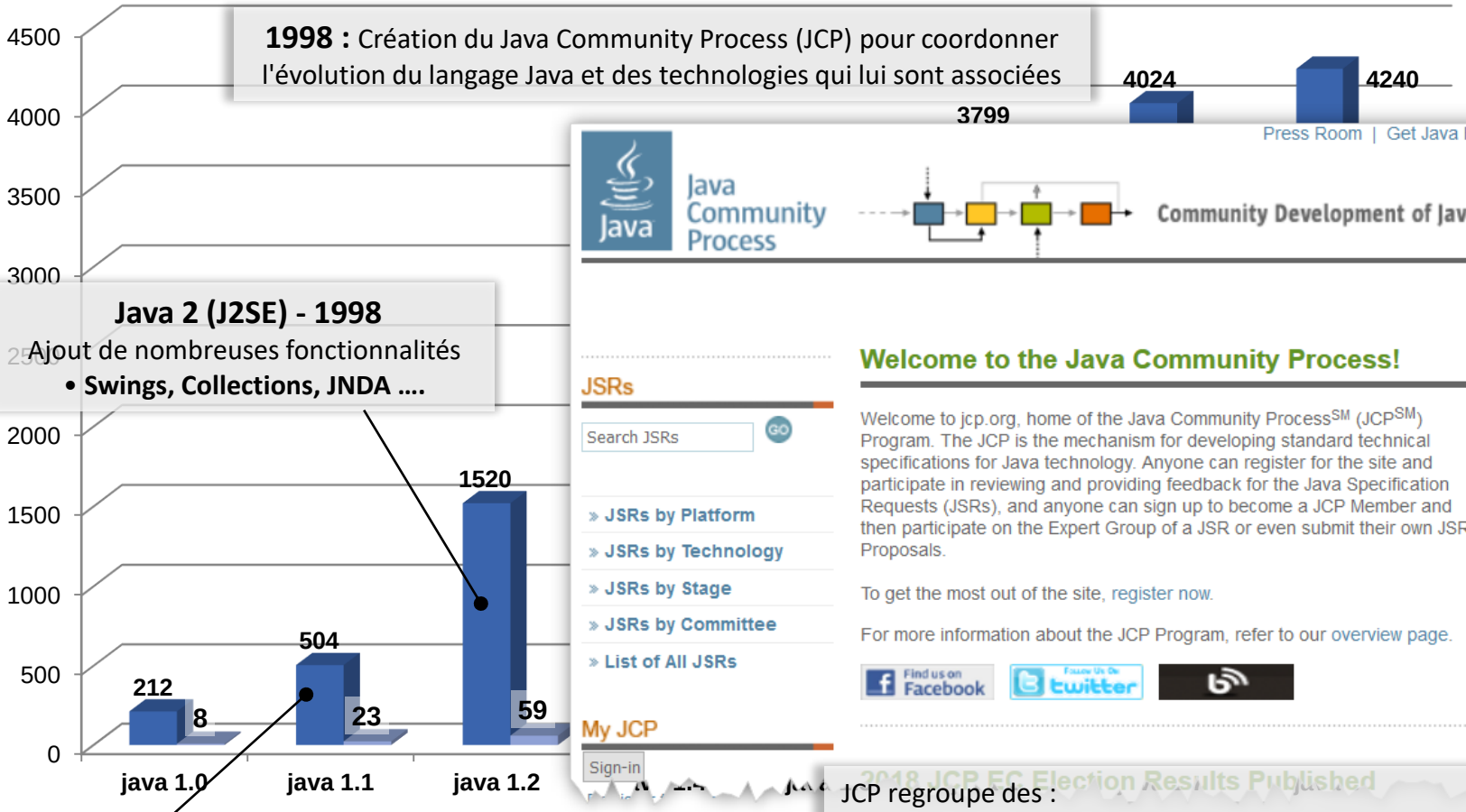
Evolution de l'API



La plateforme Java

Evolution de l'API JSE

1998 : Création du Java Community Process (JCP) pour coordonner l'évolution du langage Java et des technologies qui lui sont associées



Java 2 (J2SE) - 1998

Ajout de nombreuses fonctionnalités

- **Swings, Collections, JNDA ….**

Java 1.1 – 1997

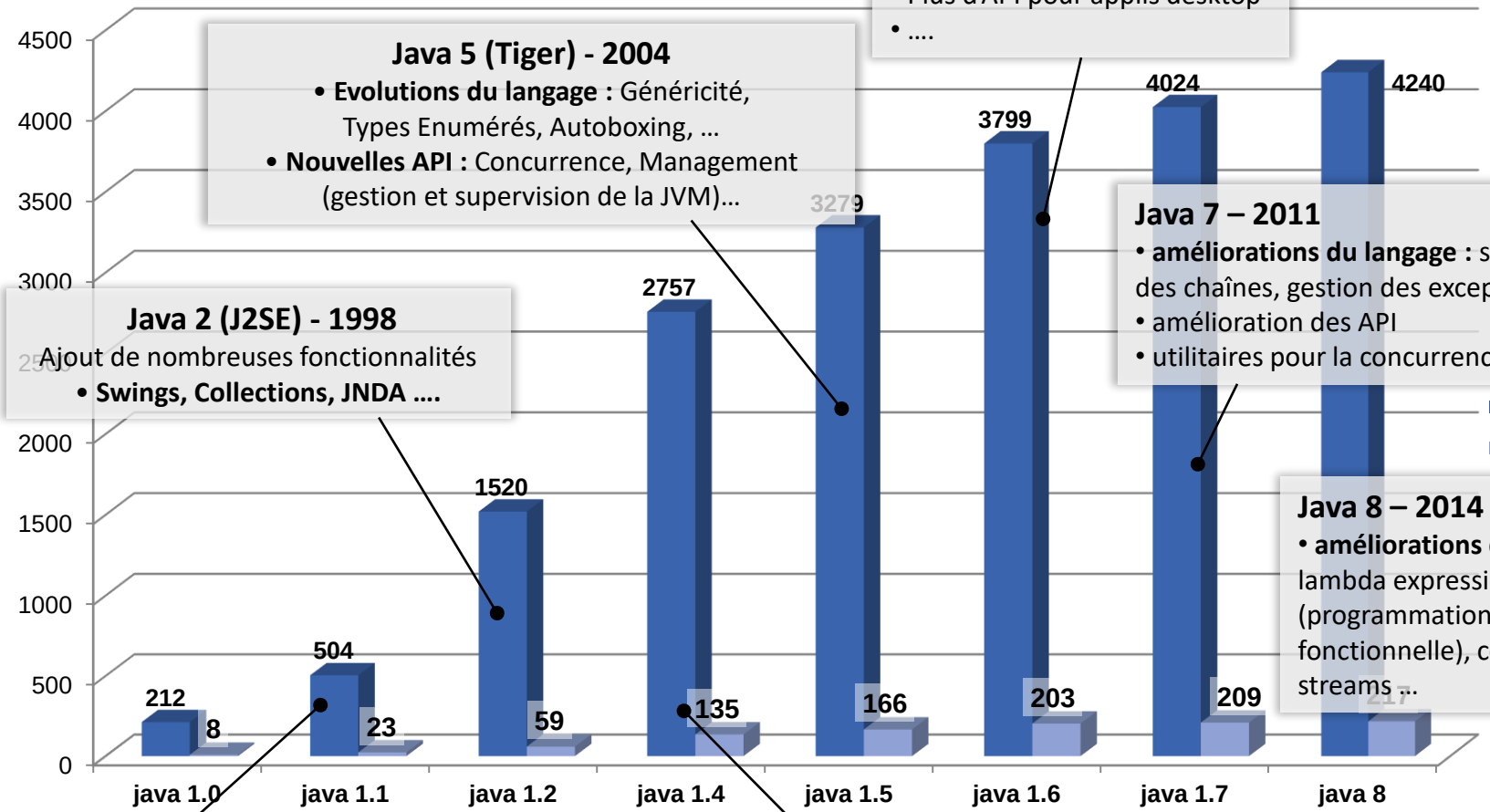
- Modification du langage : classes internes
- Modification du modèle événementiel pour awt Java Beans
- Amélioration des performances de la VM

JCP regroupe des :

- Entreprises (Oracle, Google, Apple ….)
- Acteurs du logiciel libre (Apache Software Foundation, Eclipse Foundaion …)
- Membres à à titre individuel (adhésion gratuite)

La plateforme Java

Evolution de l'API JSE



Java 6 (Mustang) - 2006
- Services Web
- Langages de scripts
- Accès au compilateur
- Plus d'API pour applis desktop
- ....

Java 5 (Tiger) - 2004
- **Evolutions du langage :** Généricité, Types Enumérés, Autoboxing, …
- **Nouvelles API :** Concurrence, Management (gestion et supervision de la JVM)…

Java 7 – 2011
- **améliorations du langage :** switch avec des chaînes, gestion des exceptions,
- amélioration des API
- utilitaires pour la concurrence.

Java 2 (J2SE) - 1998
Ajout de nombreuses fonctionnalités
- **Swings, Collections, JNDA ….**

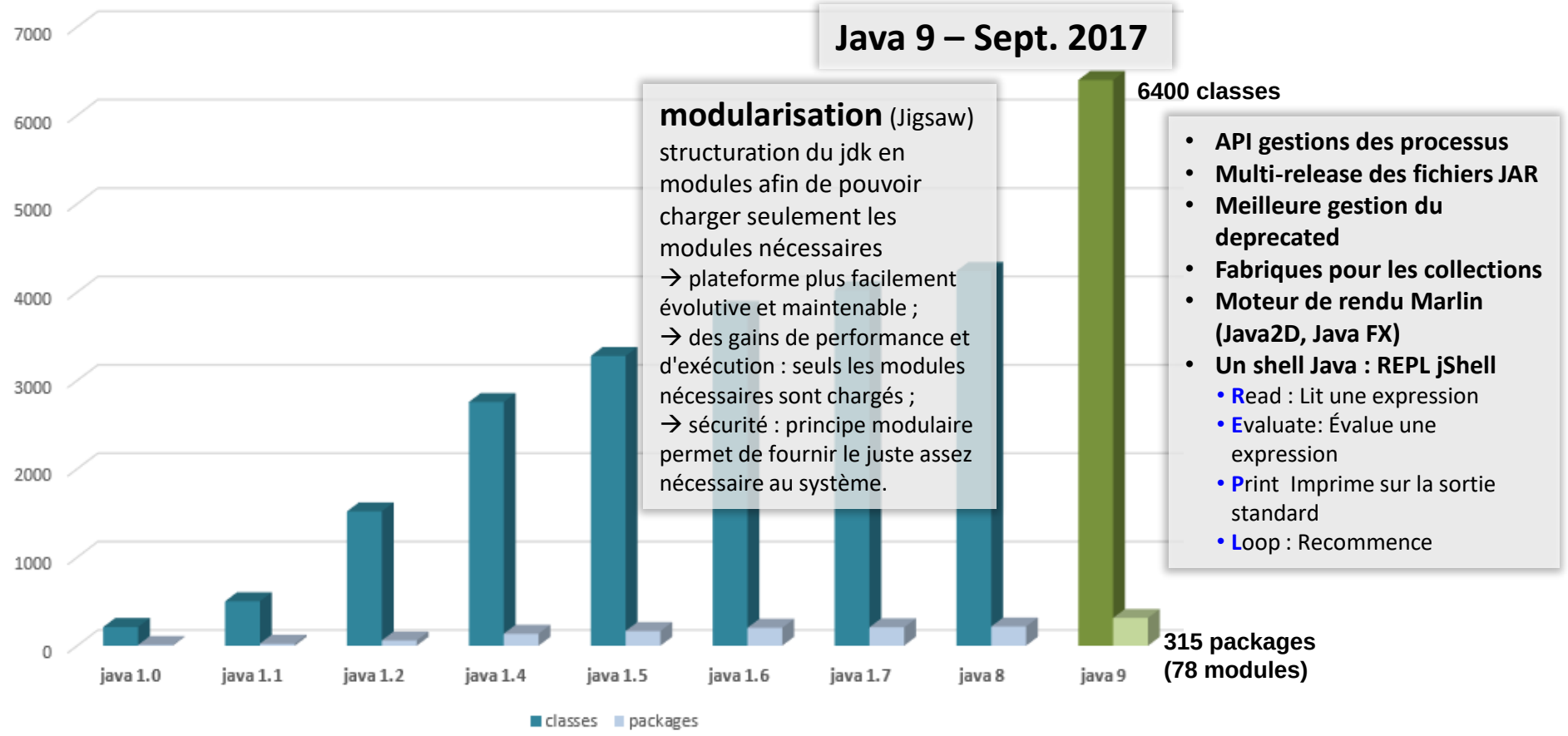
Java 8 – 2014
- **améliorations du langage :** lambda expressions (programmation fonctionnelle), collections, streams …

Java 1.1 – 1997
- Modification du langage : classes internes
- Modification du modèle événementiel pour awt Java Beans
- Amélioration des performances de la VM

J2SE 1.4 – 2002
- Amélioration des performances
- Assertions, Nouveau package pour i/o,
- Nouvelles classes pour collections…
- Nombreux nouveaux packages : XML, log des applications, préférences utilisateur …

La plateforme Java

Evolution de l'API JSE



Java 9 – Sept. 2017

6400 classes

315 packages (78 modules)

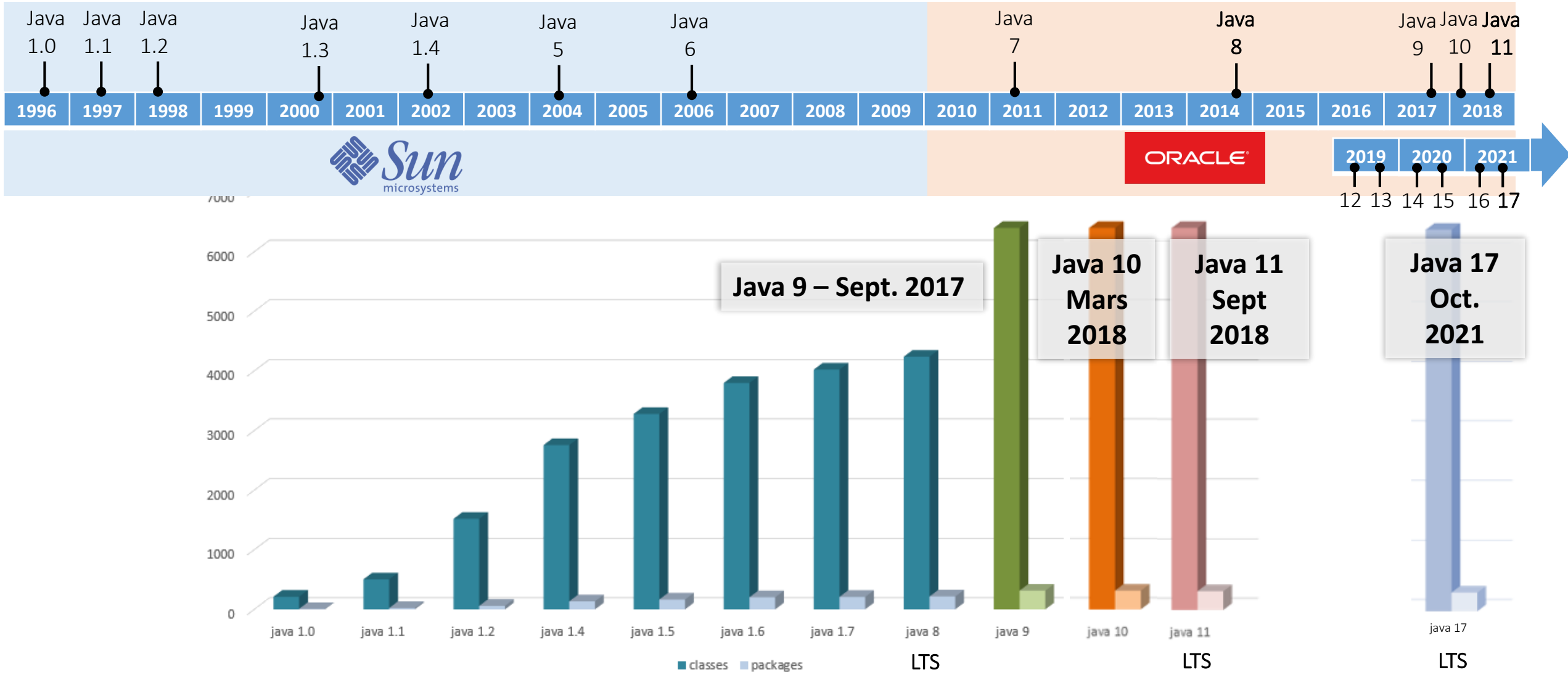
modularisation (Jigsaw)
structuration du jdk en modules afin de pouvoir charger seulement les modules nécessaires
→ plateforme plus facilement évolutive et maintenable ;
→ des gains de performance et d'exécution : seuls les modules nécessaires sont chargés ;
→ sécurité : principe modulaire permet de fournir le juste assez nécessaire au système.

- **API gestions des processus**
- **Multi-release des fichiers JAR**
- **Meilleure gestion du deprecated**
- **Fabriques pour les collections**
- **Moteur de rendu Marlin (Java2D, Java FX)**
- **Un shell Java : REPL jShell**
  - **R**ead : Lit une expression
  - **E**valuate: Évalue une expression
  - **P**rint Imprime sur la sortie standard
  - **L**oop : Recommence

La plateforme Java

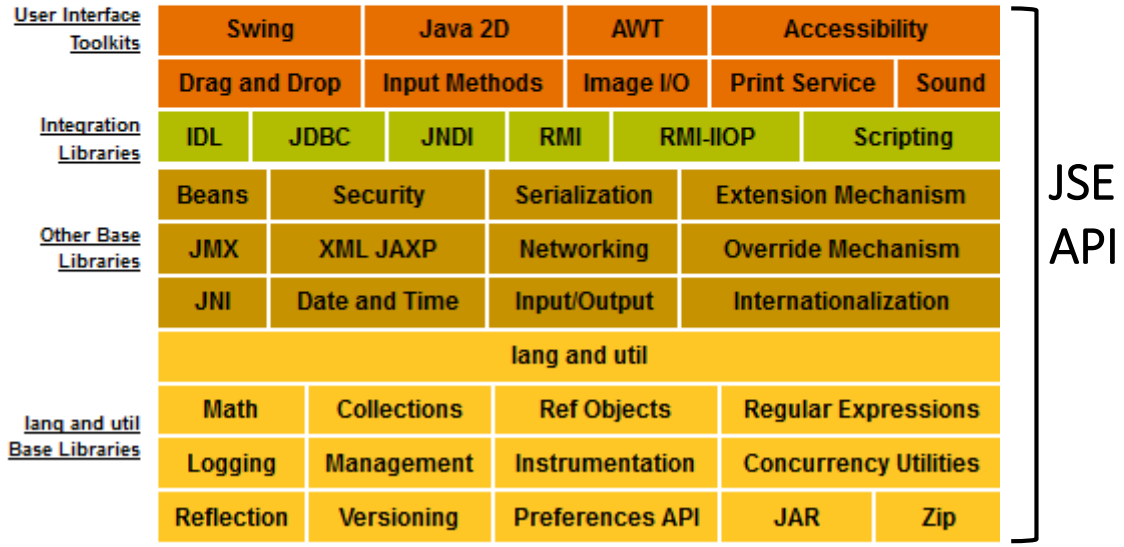
Evolution de l'API JSE

Depuis Java 9, Oracle JCP adopte une politique de release tous les 6 mois. Seules les releases LTS (Long Term Support) sont maintenues sur plusieurs années



JSE : Java Standard Edition

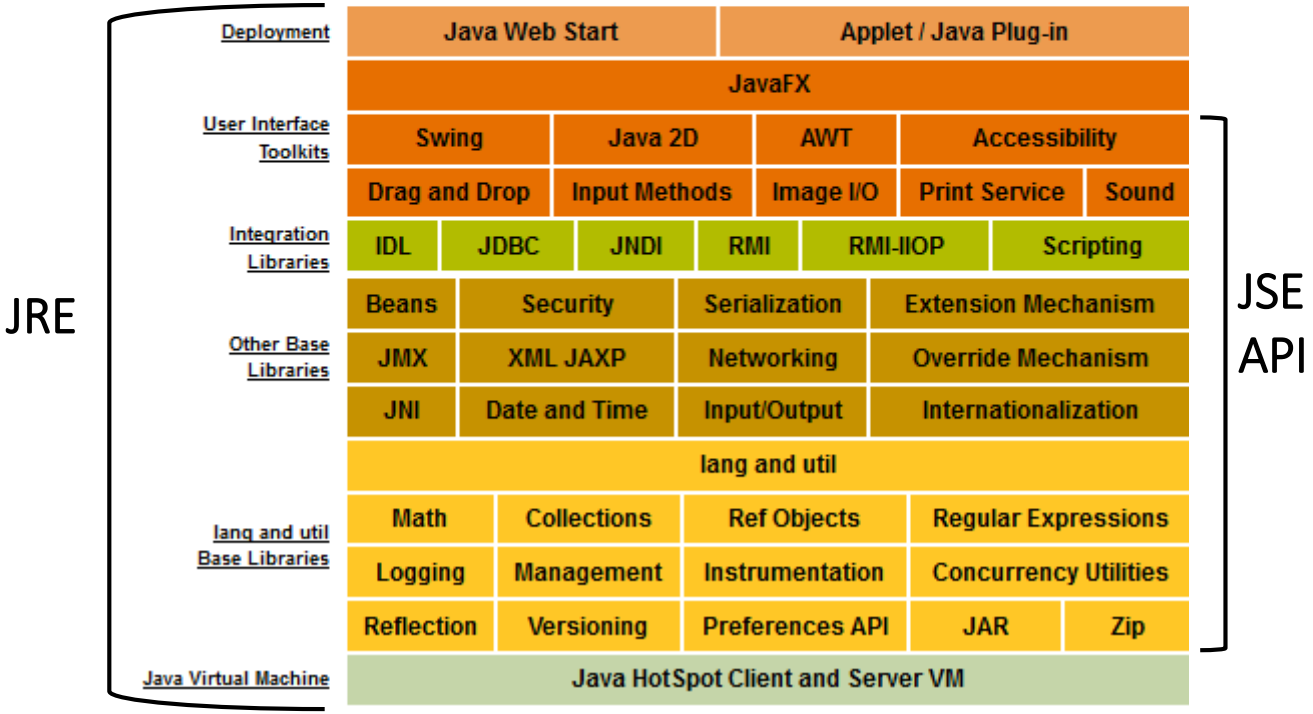
Organisation générale



JSE API : Java Application Programming interface

JSE : Java Standard Edition

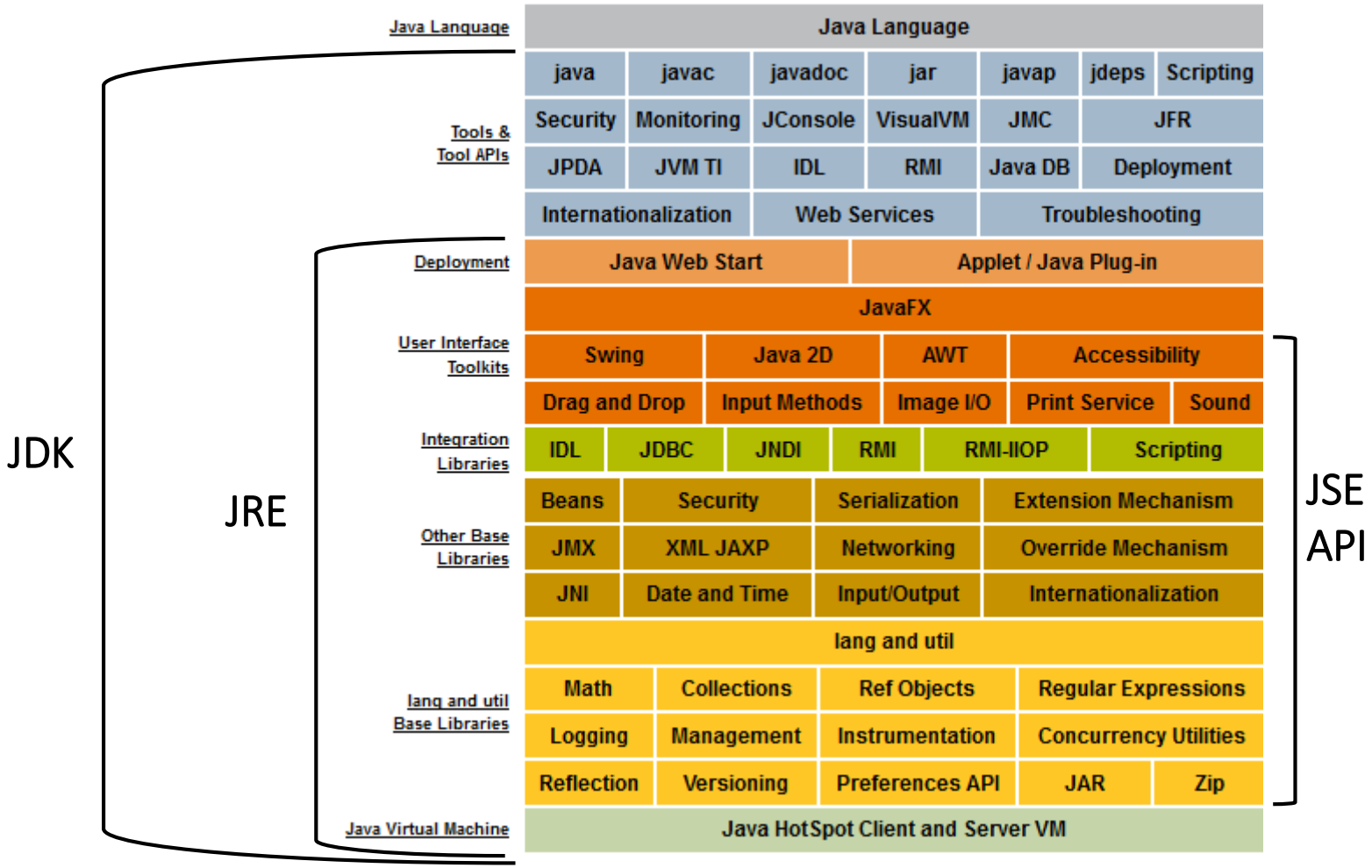
Organisation générale



JRE (Java Runtime Environnement) pour l'exécution de code java compilé

JSE : Java Standard Edition

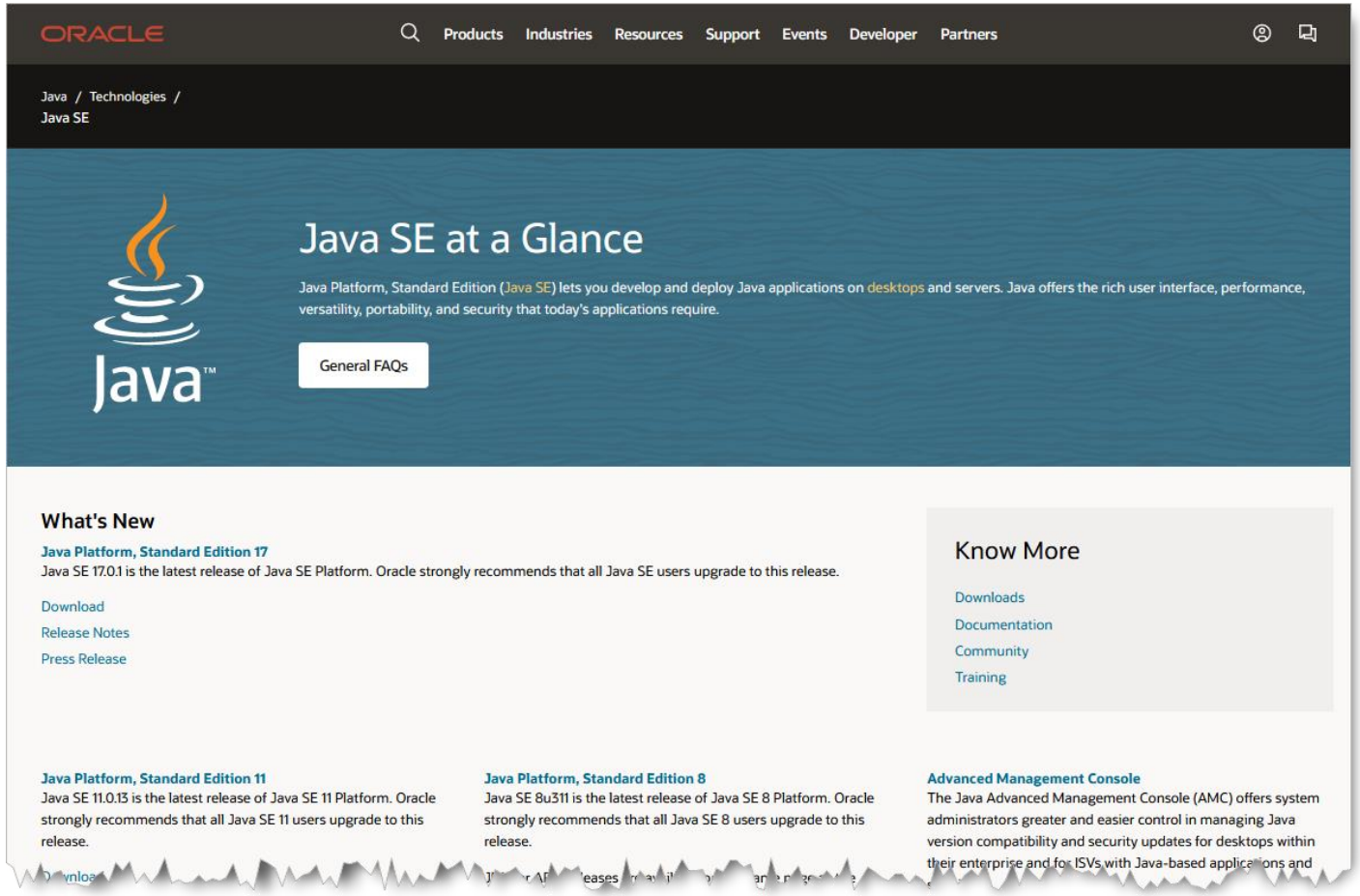
Organisation générale



JDK (Java Developper's Kit) outils de base pour le développement d'applications Java

Distributions du JDK

- Oracle JDK
  https://www.oracle.com/technetwork/java/javase/overview/index.html



récupéré le 4/01/2022

- Open JDK
  https://openjdk.java.net/

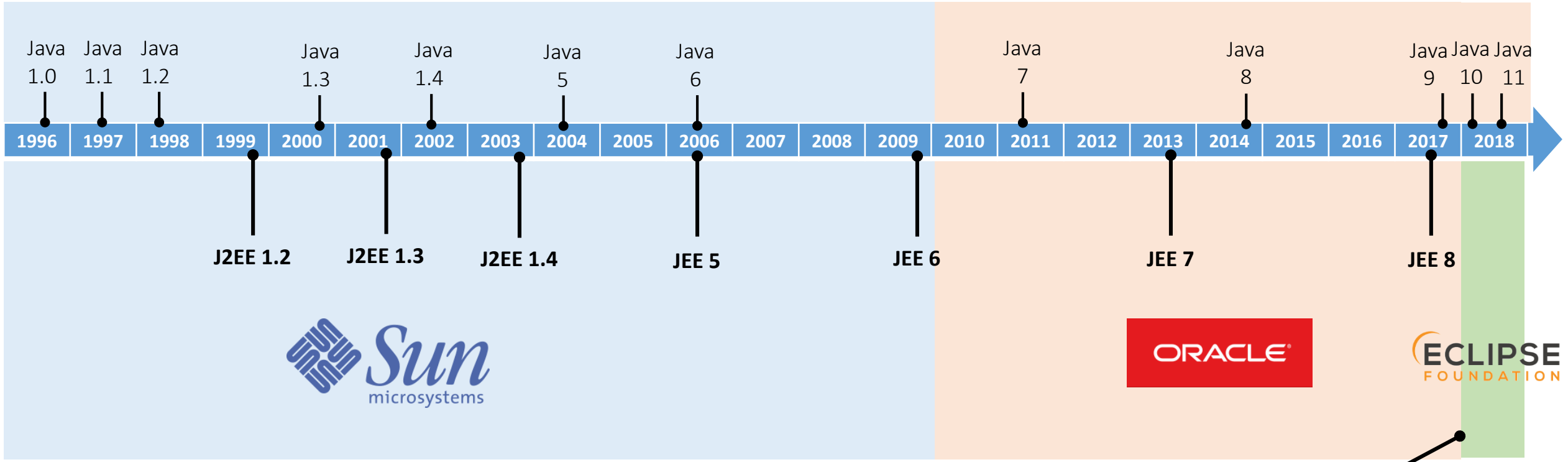
récupéré le 4/01/2022

JEE Java Entreprise Edition

- Java EE (Java Enterprise Edition), anciennement J2EE (Java 2 Entreprise Edition) et maintenant Jakarta EE est un ensemble de spécifications étendant JSE pour le développement d'applications d'entreprise.
  https://jakarta.ee/specifications/platform/
- La spécification JEE définit
  - Un ensemble d'API pour les applications d'entreprises (applications réparties, services web, mapping Objet/Relationnel pour la persistance des données…),
  - Un plate-forme (Java EE Platform) à base de composants pour héberger et exécuter les applications (serveurs Java EE)
  - Une suite de tests (*Java EE Compatibility Test Suite*) pour vérifier la compatibilité des implémentations
  - Une implémentation de référence (*Java EE Reference Implementation*): GlassFish ;

JEE Java Entreprise Edition - Historique

- Evolution de la spécification JEE par rapport à évolution du langage Java (JSE)



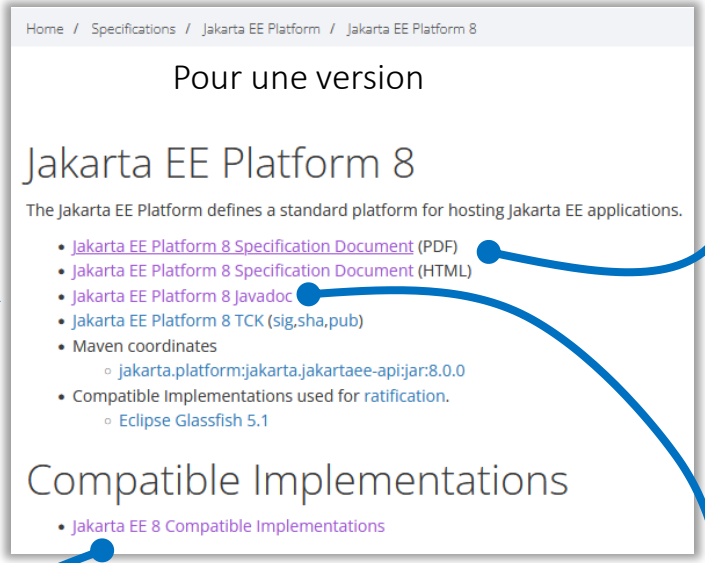
Fin 2017 Oracle cède JEE à la fondation Eclipse. Java étant une marque déposée, Java EE a été renommé EE4J (Eclipse Enterprise for Java) pour finalement devenir **Jakarta EE**

JEE Spécifications

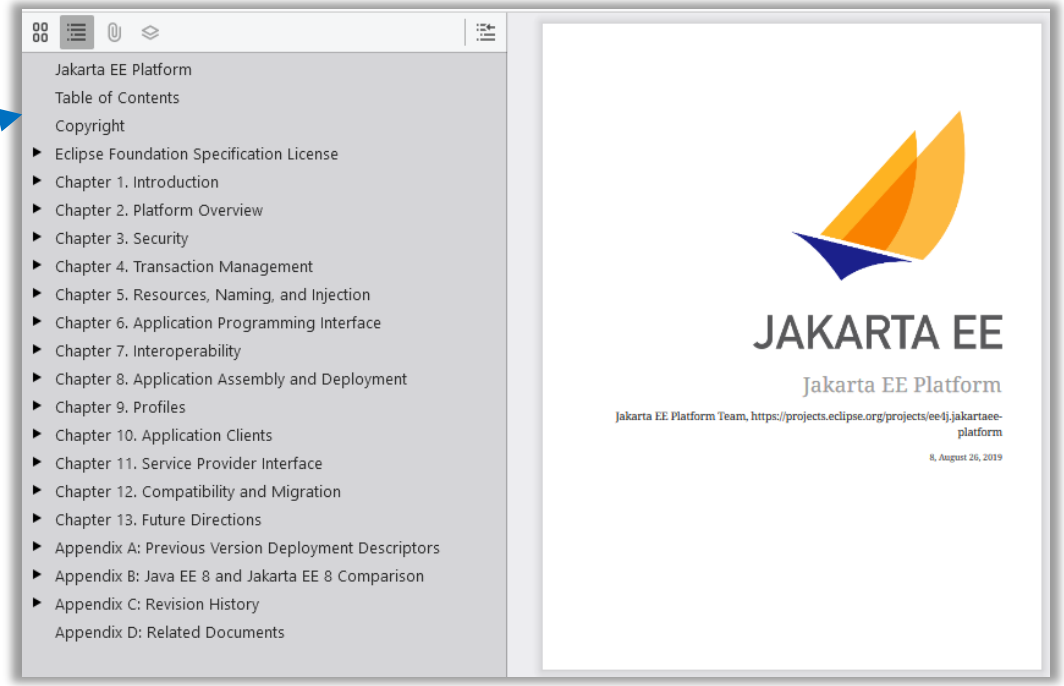
- https://jakarta.ee/specifications/platform/

Différentes versions

Pour une version



Spécifications



Implémentations (open source ou commerciales)

APIs (Javadoc)

Java EE - APIs

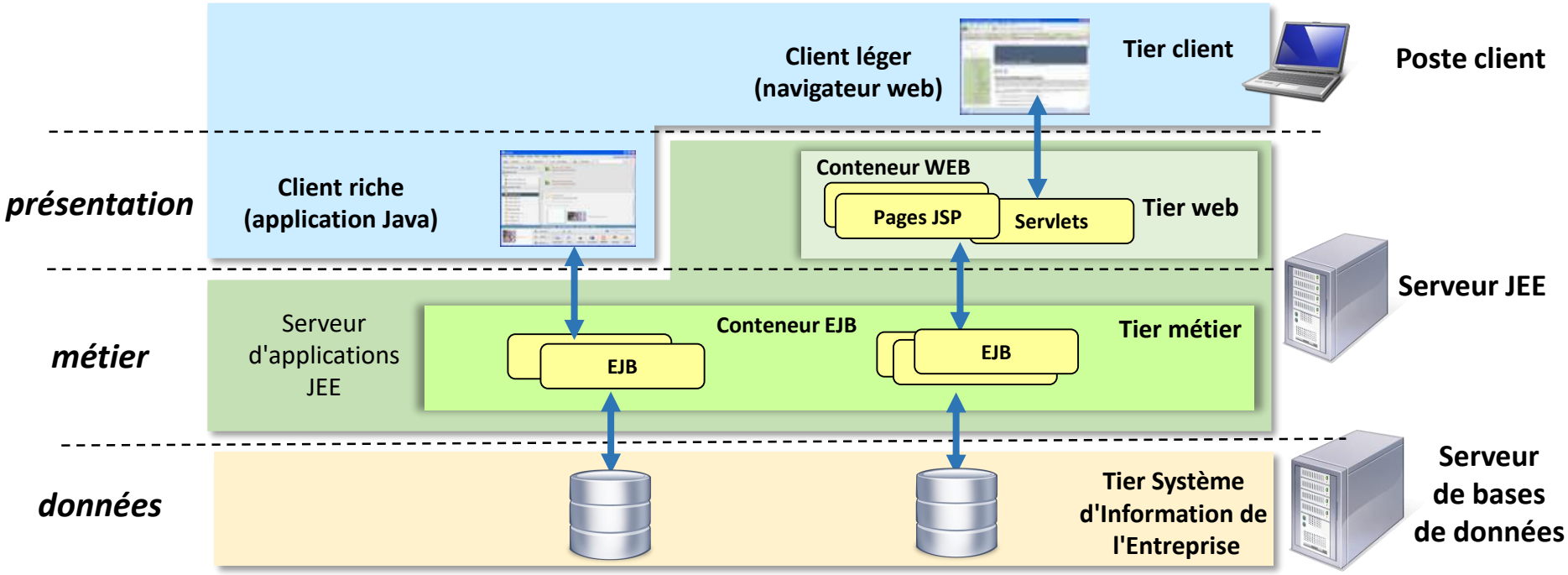
- JEE inclus de nombreuses API couvrant de nombreuses fonctionnalités
  - pages web dynamiques, lecture écriture transactionnelle dans des BD, files d'attentes distribuées…
  - Web
    - Servlet: gestion de requêtes HTTP. API de bas niveau, les autres spécifications JEE s'appuient sur elle,
    - JSP (Java Server Pages) : pages web dynamiques,
    - JSF (Java Server Faces): construction d'interfaces utilisateur à partir de composants;
    - EL (Unified Expression Language): langage simple pour lier des composants (JSP ou JSF) avec des objets Java,
    - WebSocket: gestion de connexions WebSocket.
  - Services Web
    - Java API for XML Web Services: pour la création de services web SOAP,
    - Java API for RESTful Web Services: support pour créer des services web conformes aux principes architecturaux REST (Representational State Transfer);
    - Java API for JSON Processing: gestion d'informations encodées au format JSON format;,
    - Java API for JSON Binding: conversion d'informations JSON en classes et objets Java et inversement,
    - Java Architecture for XML Binding: transformation de XML en objets Java,

Java EE - APIs

- Spécifications d'entreprise
  - EJB (Enterprise JavaBean) ensemble d'APIs légères (lightweigh) pour la définition d'objets métiers (Entreprise Beans) pris en charge par un container d'objets (EJB container)
    - transactions (en utilisant JTA), appels de procédures à distance (RMI ou RMI-IIOP), gestion de la concurrence, injection de dépendances, control d'accès.
  - JPA (Java Persistence API) : ORM (Object-Relational Mapping) entre tables d'un SGBDR et des classes Java.
  - JTA (Java Transaction API) : interaction avec le support transactionnel offert par JEE with the transaction support offert par Java EE.
  - JMS (Java Message Service) : création, envoi, réception de messages d'un système de messagerie d'entreprise.
- Autres spécifications
  - Validation (Bean Validation API) : fournit un moyen unifié pour définir des contraintes sur les objets métiers (beans, par exemple les classes de modèle JPA model) qui peuvent être appliquées sur plusieurs couches (persistance JPA, vues JSF).
  - Batch Applications: exécution de programmes en tâches de fond.

Architecture des applications Java EE

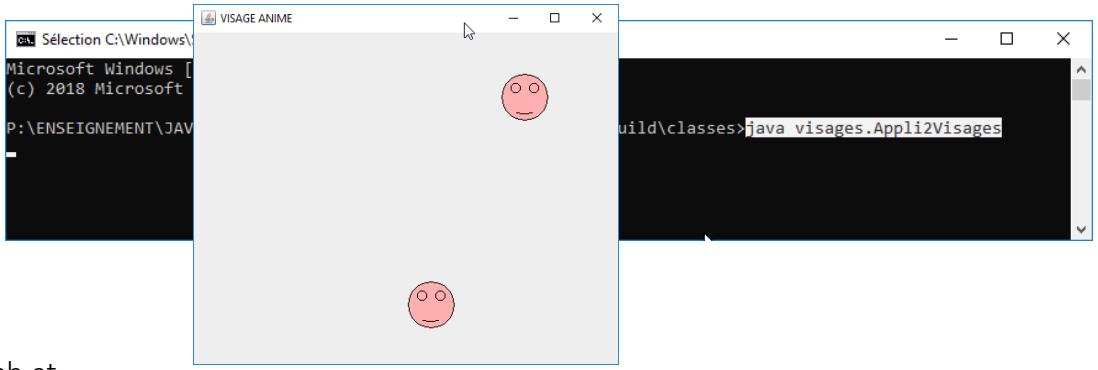
- S'appuie sur un modèle d'architecture multi-tiers (multi-couches)



- Logique de l'application :
  - Composants web (Servlet, JSP,JFS)
  - Composants métiers (EJB) : Client , Compte en Banques
- Services standards (cycle de vie des composants, multithreading, transactions, persistance…) pris en charge par les conteneurs Web et EJB du serveur d'application JEE

Exemples de programmes Java

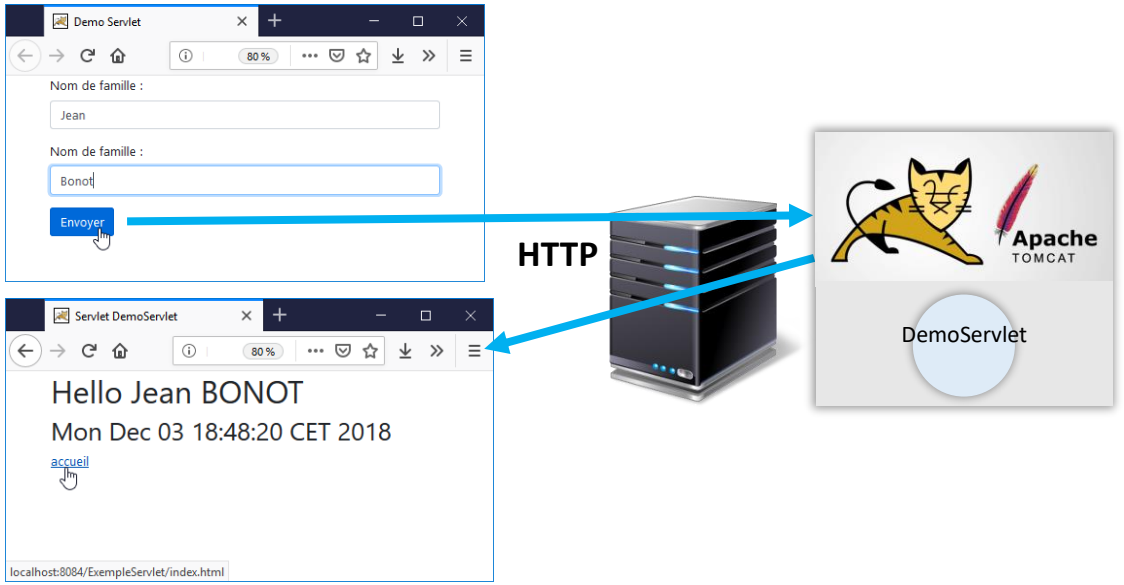
- Les Applications indépendantes
  - Programmes autonomes (stand-alone) lancé depuis une JVM



- Les Applets
  - Programmes exécutées dans l'environnement d'un navigateur Web et chargés au travers de pages HTML

http://lig-membres.imag.fr/genoud/ENSJAVA/cours/TicTacToe/example1.html

- Servlets
  - Programmes exécutés par un serveur d'application (exemple Apache Tomcat conteneur Web JEE)



Seuls diffèrent les contextes d'invocation et d'exécution

- Les droits des applications, applets servlets ne sont pas les mêmes

Exemples de programmes Java

Application indépendante

Application est définie par un ensemble de classes dont **une** jouera le rôle de **classe principale**

`Appli2Visages.java`
La classe "principale"

`VisageRond.java`
Défini un type d'objet Visage

`Dessin.java`
Gère la zone de dessin

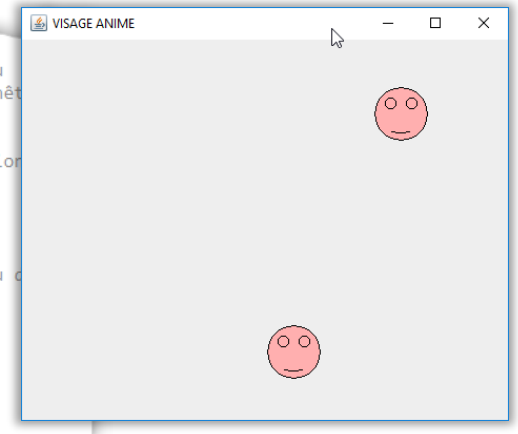
```java
package visages;

import javax.swing.JFrame;
import javax.swing.WindowConstants;

/**
 * Animation d'un visage dans une fenêtre graphique.
 * Un visage est dessiné à l'intérieur d'une fenêtre et se déplace. Chaque fois
 * que l'un des bords est atteint, le visage change de direction.
 * @author Philippe Genoud
 */
public class Appli2Visages  {

    public static void main(String[] args) {

        // la fenêtre graphique
        JFrame laFenetre = new JFrame("VISAGE ANIME");
        laFenetre.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        laFenetre.setSize(512,512);

        // créé la zone de dessin et la place dans la fenêtre
        Dessin d = new Dessin();
        laFenetre.add(d);

        //  affiche la fenêtre
        laFenetre.setVisible(true);

        // creation d'un objet VisageRond
        VisageRond  v1 = new VisageRond();
        // on rajoute cet objet la zône de dessin
        d.ajouterObjet(v1);
        // création d'un deuxième visage
        VisageRond  v2 = new VisageRond(d.getLargeur()/2,d.getHauteur()/2);
        v2.setDy(-5);
        // ajout de cet objet à la zone de dessin
        d.ajouterObjet(v2);
```

- Application doit posséder une classe principale
  - classe possédant une méthode de signature
    `public static void main(String[] args)`
  - cette méthode servira de point d'entrée pour l'exécution
    `java Appli2Visages`

Tableau de chaînes de caractères (équivalent à **`argc`**, **`argv`** du C)

Exécute le code défini dans la méthode main contenue dans le fichier **`Appli2Visages.class`**

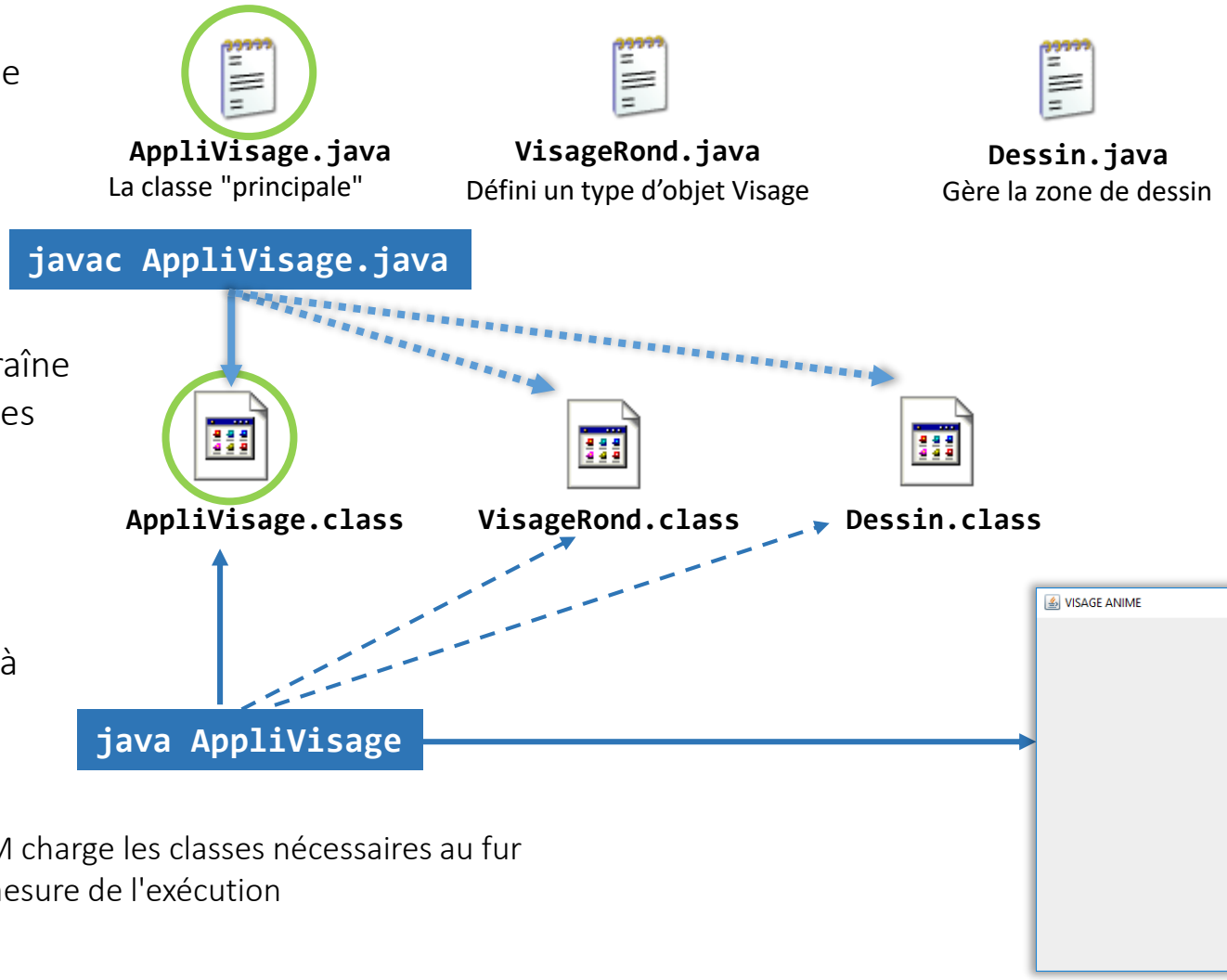
```java
        // la boucle d'animation
        // c'est une boucle infinie, le programme devra être interrompu
        // par CTRL-C ou en cliquant dans le cas de fermeture de la fenêt
        while (true) {
            // les visages effectuent un déplacement élémentaire
            // en rebondissant sur les bords et en changeant d'expressior
            v1.deplacerAvecRebond();
            v2.deplacerAvecRebond();
            // la zone de dessin se réaffiche
            d.repaint();
            // un temps de pause pour avoir le temps de voir le nouveau d
            d.pause(50);

        }
    }

} // AppliVisage1
```



Exemples de programmes Java

Application indépendante

Application est définie par un ensemble de classes dont **une** jouera le rôle de **classe principale**



La compilation de la classe principale entraîne la compilation de toutes les classes utilisées

`javac ≃ make`

Pour exécuter l'application on indique à l'interpréteur java le nom de la classe principale

La JVM charge les classes nécessaires au fur et à mesure de l'exécution

Exemples programmes Java

Applet

- Classe principale ne posséde pas de méthode `main()`
- Hérite de **`java.awt.Applet`** ou **`javax.swing.JApplet`**

- Son bytecode réside sur un serveur http
- Elle est véhiculée vers un client http (navigateur Web) via une page html qui contient son url

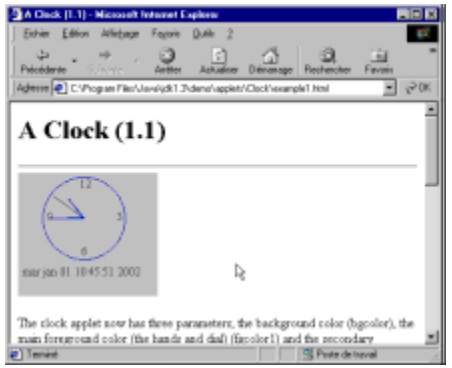
- Lorsqu'un navigateur compatible Java (avec sa propre machine virtuelle java (JVM) : Java Plugin) reçoit cette page HTML, il télécharge le code de la classe et l'exécute sur le poste client
  - l'applet doit posséder un certain nombre de méthodes pour permettre cette exécution
    - **`init(), start(), stop(), paint(), destroy()`**

Exemple de programmes Java

Applet : Principe de fonctionnement

DEPRECATED

```
<HTML>
  <HEAD>
    <TITLE>A Clock (1.1)</TITLE>
  </HEAD>
  <BODY>
    <h1>A Clock (1.1)</h1>
    <hr>
    <applet code="Clock2.class"
                width=170 height=150>
        <param bgcolor="C0C0C0">
    </applet>
    ....
  </BODY>
```



http://java.sun.com/j...applets/Clock/example1.html

1. Demande de chargement d 'une page web contenant une applet

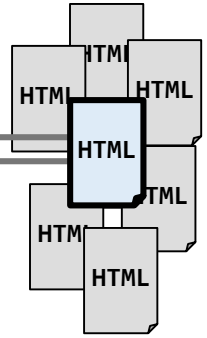
2. Chargement de la page html

example1.html

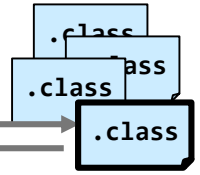
Client http

Navigateur intégrant une machine virtuelle Java

Serveur http



3. Demande de chargement de l'applet référencée dans la page



4. Chargement du bytecode des classes de l'applet

Clock2.class

5. Exécution de l'applet dans la JVM du client

1ère séance de TP

- http://lig-membres.imag.fr/genoud/teaching/PL2AI/tds/PL2/sujets/tp01_jdk/tp01_jdk.html
- utilisation depuis ligne de commande des outils de base du JDK pour le développement d'applications Java
  - Compilation :
    - **javac**
  - Exécution :
    - **java**
  - Documentation
    - **javadoc**
  - "Archivage"
    - **jar**
- Pourquoi ne pas utiliser tout de suite un IDE Java ?
  - Être capable de travailler avec Java quelque soit l'environnement,
  - Bien comprendre les étapes de compilation puis d'exécution,
  - Comprendre des mécanismes parfois masqués par des environnements intégrés (ex. la notion de **CLASSPATH**).

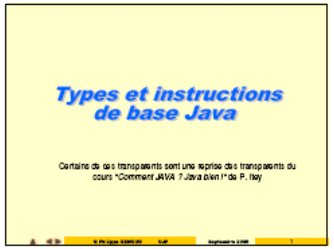
2ème séance de TP

http://lig-membres.imag.fr/genoud/teaching/PL2AIPL2AI/tds/PL2/sujets/tp02_typesSimples/tp02_typesSimples.html

- Objectif : expérimenter les constructions de base du langage Java (types simples, instructions de contrôle, itérations) qui sont très proches syntaxiquement de celles utilisées par le langage C

- Ecriture de programmes simples
  - Thème 1 : expressions
  - Thème 2 : instructions conditionnelles
  - Thème 3 : itérations
  - Thème 4 : instanciation d'objets - envois de messages

http://lig-membres.imag.fr/genoud/PL2AI/cours/

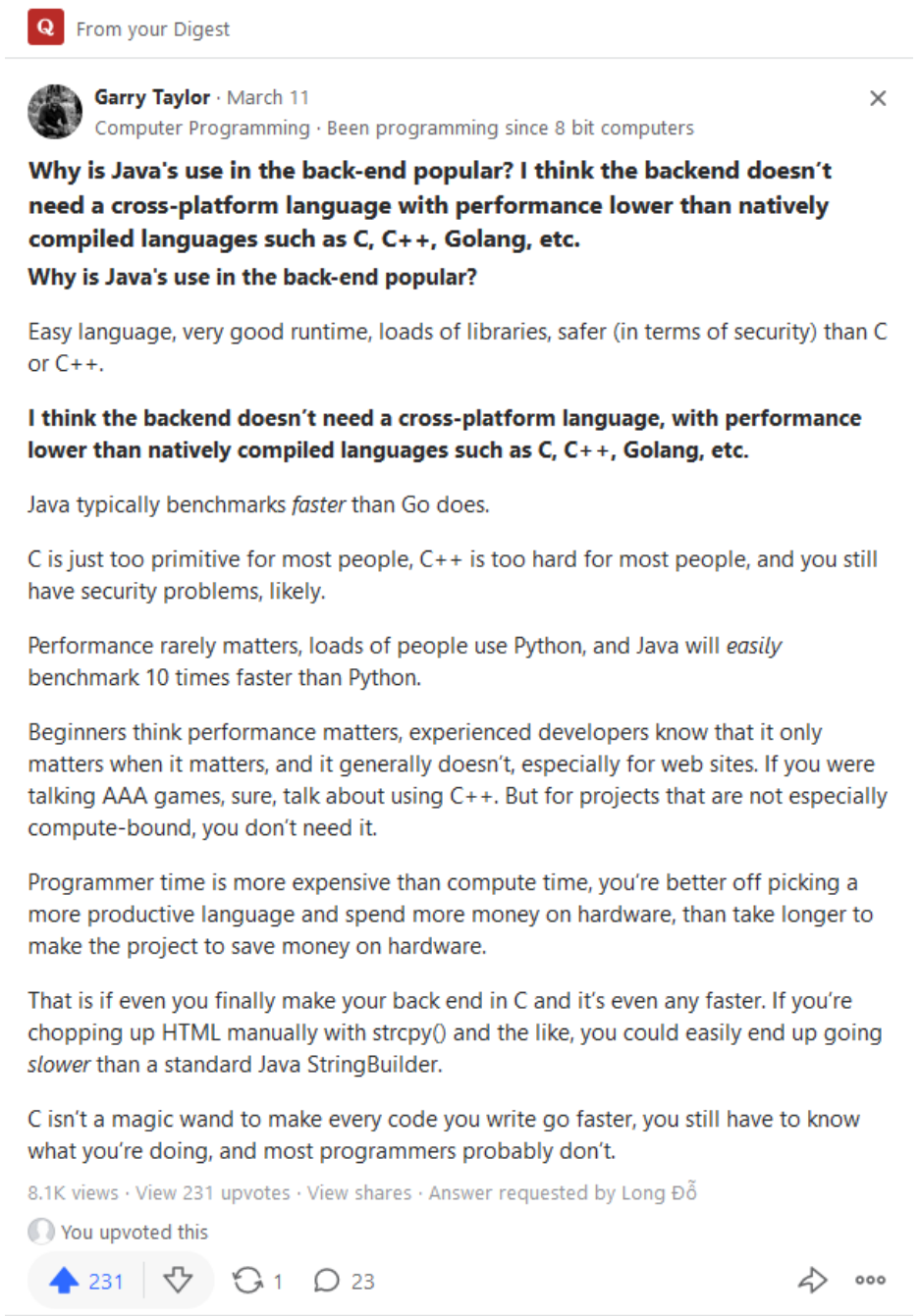


Performances

- Exécution d'un programme Java
  - le code Java est compact,
  - le chargement des classes nécessaires est sélectif et dynamique,
  - ... mais Java est interprété

- Palier aux lenteurs de l'interprétation
  - utilisation d'un JIT (compilateur « Just-in-Time »)
    - compilation à la volée du byte-code
    - réutilisation du code déjà compilé
    - intégration du JIT HotSpot dans JVM depuis version 1.3

https://www.quora.com/Why-is-Javas-use-in-the-back-end-popular-I-think-the-backend-doesn-t-need-a-cross-platform-language-with-performance-lower-than-natively-compiled-languages-such-as-C-C-Golang-etc



From your Digest

Garry Taylor · March 11
Computer Programming · Been programming since 8 bit computers

Why is Java's use in the back-end popular? I think the backend doesn't need a cross-platform language with performance lower than natively compiled languages such as C, C++, Golang, etc.

Why is Java's use in the back-end popular?

Easy language, very good runtime, loads of libraries, safer (in terms of security) than C or C++.

I think the backend doesn't need a cross-platform language, with performance lower than natively compiled languages such as C, C++, Golang, etc.

Java typically benchmarks *faster* than Go does.

C is just too primitive for most people, C++ is too hard for most people, and you still have security problems, likely.

Performance rarely matters, loads of people use Python, and Java will *easily* benchmark 10 times faster than Python.

Beginners think performance matters, experienced developers know that it only matters when it matters, and it generally doesn't, especially for web sites. If you were talking AAA games, sure, talk about using C++. But for projects that are not especially compute-bound, you don't need it.

Programmer time is more expensive than compute time, you're better off picking a more productive language and spend more money on hardware, than take longer to make the project to save money on hardware.

That is if even you finally make your back end in C and it's even any faster. If you're chopping up HTML manually with strcpy() and the like, you could easily end up going *slower* than a standard Java StringBuilder.

C isn't a magic wand to make every code you write go faster, you still have to know what you're doing, and most programmers probably don't.

8.1K views · View 231 upvotes · View shares · Answer requested by Long Đỗ

You upvoted this

231 1 23