

Optez pour une transformation digitale intelligente

L'INTELLIGENCE ARTIFICIELLE AU SERVICE DE VOTRE BUSINESS.



JAVA



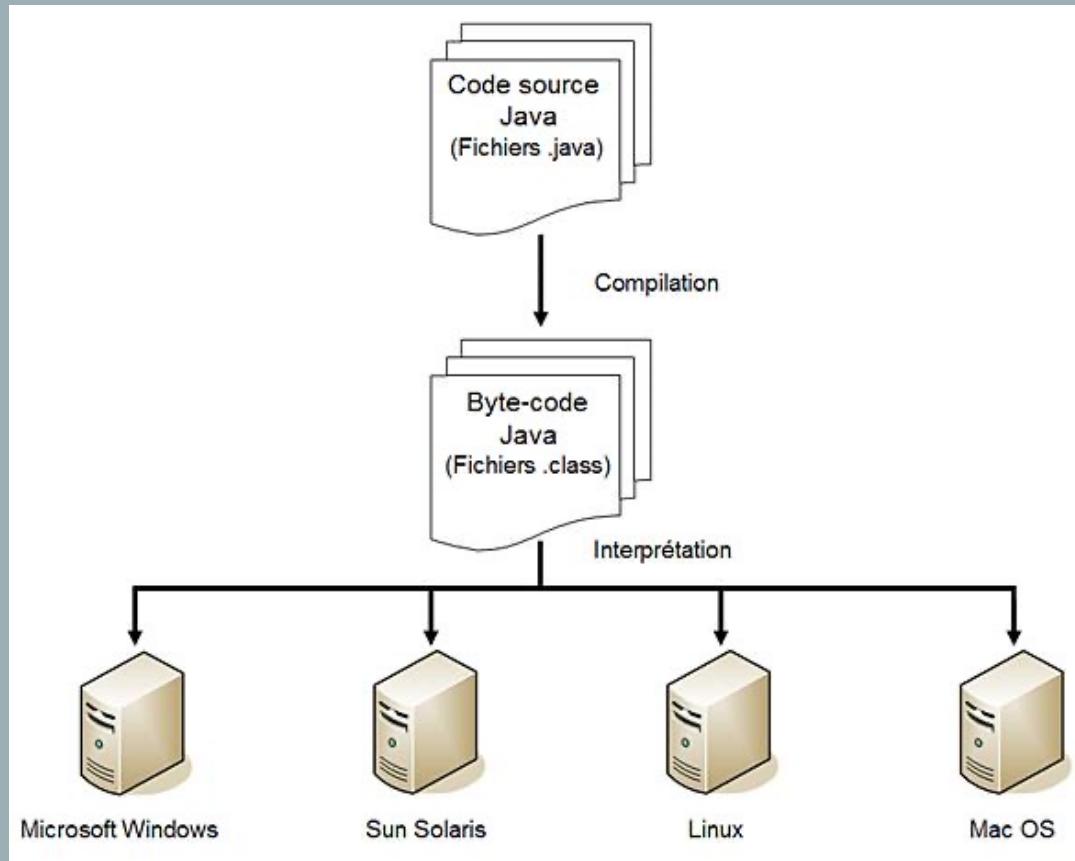
SOMMAIRE

I. L'écosystème Java

1. La plateforme Java.
2. L'environnement d'exécution : la JVM.
3. Le socle technique (Java SE) complété par des librairies.
4. La spécification pour les applications d'entreprise (Java EE – Jakarta EE)
5. Les langages : Java, Kotlin, Scala, Groovy, Clojure, etc.
6. Les outils de conception : Maven, Graddle.
7. Les IDE : Eclipse, IntelliJ.

I - L'écosystème Java

I – 2 - La plateforme java



WORA (*Write Once, Run Anywhere : écrire une fois, exécuter partout*).

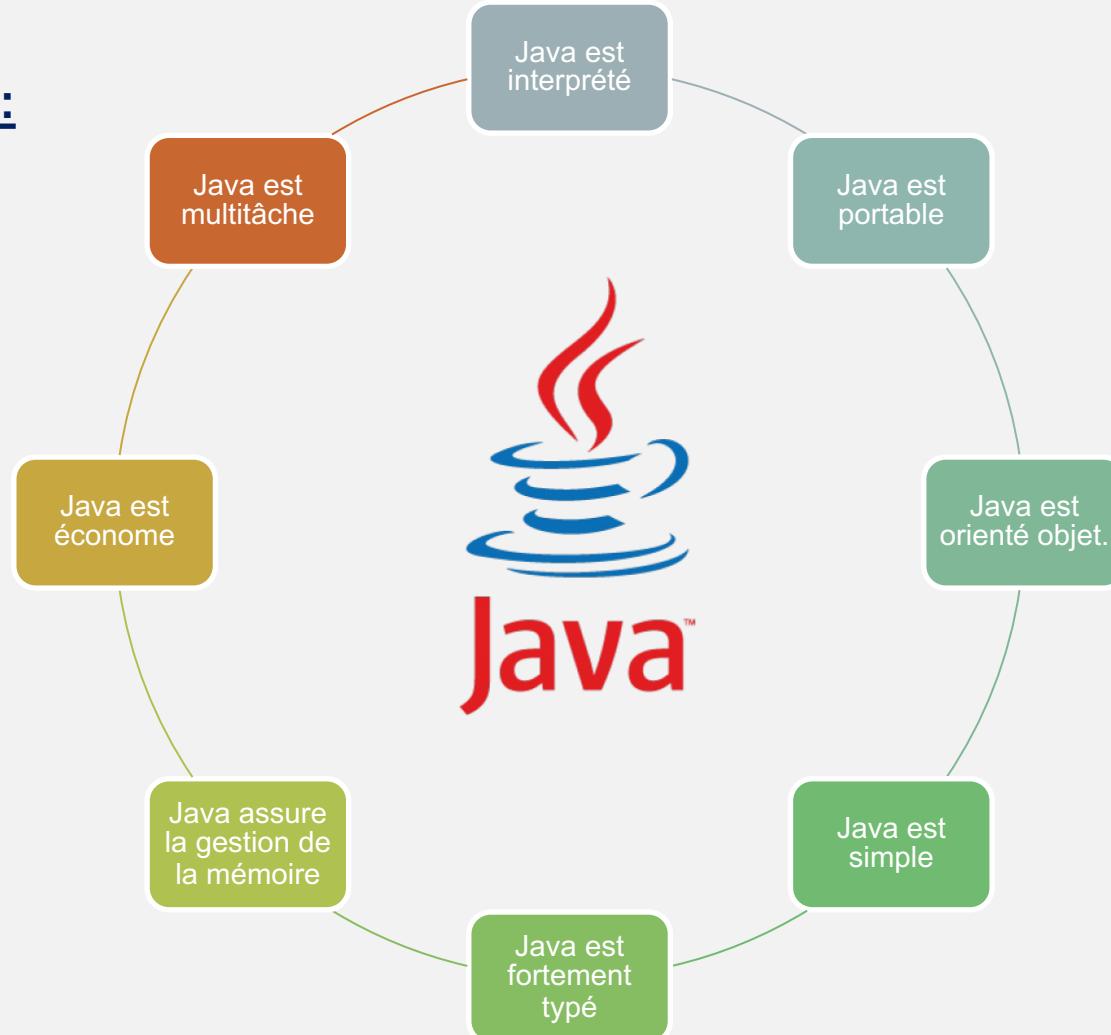
Historique

- En 1991, la société Sun Microsystems démarre un projet d'informatique embarquée.
- Le langage utilisé par les ingénieurs de Sun est le **C++**.
- Ce langage est inadapté au développement de leur projet (*problème gestion mémoire, sécurité, etc.*).
- Ils ont souhaités créer un nouveau langage plus adapté, orienté objet, inspiré du C++.
- Appellation c++--, puis Oak et enfin **Java**.
- En 1995, la première version du kit de développement logiciel en Java (JDK).
- La compilation du code source Java ne donne pas, comme c'est le cas avec beaucoup d'autres langages, un exécutable natif, mais un format de fichier spécifique, appelé **bytecode**.
- Il faut que chaque machine qui souhaite travailler avec du code java, installe une **JVM** (Java Virtuel Machine).
- La **JVM** va se charger d'interpréter le bytes-code et lancer le programme.



I – 2 - La langage Java

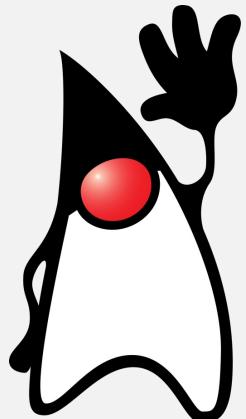
Les caractéristiques :



I – 2 - Le langage Java

Les dates importantes :

1995	Mai : premier lancement commercial du JDK 1.0
1996	Janvier : JDK 1.0.1 - Septembre : Lancement du JDC
1997	Java Card 2.0 - Février : JDK 1.1
1998	Décembre : lancement de J2SE 1.2 et du JCP - Personal Java 1.0
1999	Décembre : lancement J2EE 1.2
2000	Mai : J2SE 1.3
2001	J2EE 1.3
2002	Février : J2SE 1.4
2003	J2EE 1.4
2004	Septembre : J2SE 5.0
2005	Lancement du programme Java Champion
2006	Mai : Java EE 5 - Décembre : Java SE 6.0
2008	Décembre : JavaFX 1.0
2009	Février : JavaFX 1.1 - Juin : JavaFX 1.2 - Décembre : Java EE 6
2010	Janvier : rachat de Sun Microsystems par Oracle - Avril : JavaFX 1.3
2011	Juillet : Java SE 7 - Octobre : JavaFX 2.0
2012	Août : JavaFX 2.2
2013	Juin : Java EE 7
2014	Mars : Java SE 8, JavaFX 8
2017	Septembre Java SE 9, Java EE 8
2018	Mars : Java SE 10 - Septembre : Java SE 11



I – 2 - La langage Java

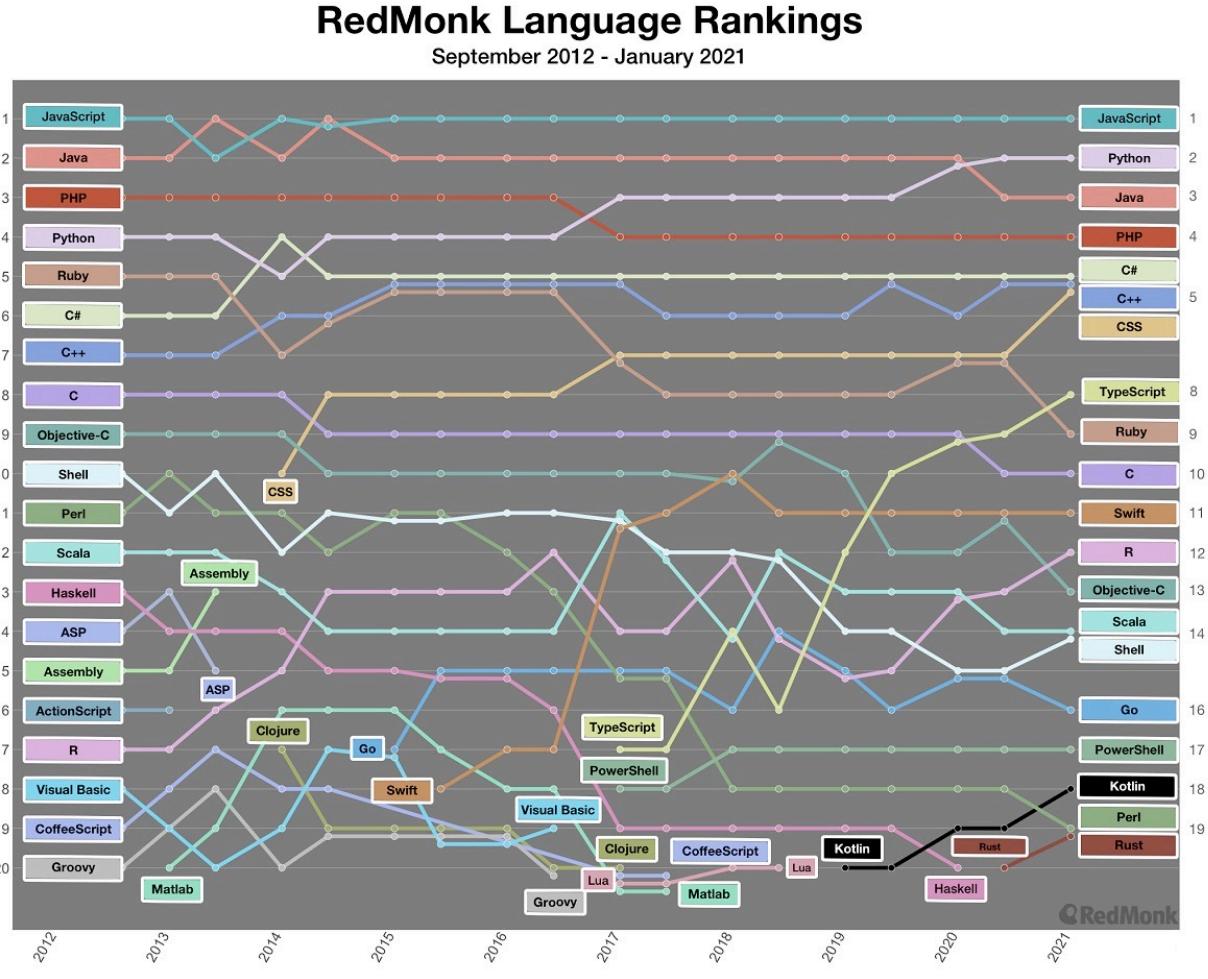
Java est compilé	Le source est compilé en pseudo code ou bytecode puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout). En effet, le bytecode , s'il ne contient pas de code spécifique à une plate-forme particulière peut être exécuté et obtenir quasiment les mêmes résultats sur toutes les machines disposant d'une JVM.
Java est portable : il est indépendant de toute plate-forme	Il n'y a pas de compilation spécifique pour chaque plate forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du bytecode .
Java est orienté objet.	Comme la plupart des langages récents, Java est orienté objet. Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primaires (entier, caractère, flottant, booléen,...).
Java assure la gestion de la mémoire	L'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au Garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.
Java est fortement typé	Toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données. Si une telle conversion doit être réalisée, le développeur doit obligatoirement utiliser un cast ou une méthode statique fournie en standard pour la réaliser.
Java est multitâche	Il permet l'utilisation de threads qui sont des unités d'exécutions isolées. La JVM, elle même, utilise plusieurs threads.
Java est sûr	La sécurité fait partie intégrante du système d'exécution et du compilateur. Un programme Java planté ne menace pas le système d'exploitation. Il ne peut pas y avoir d'accès direct à la mémoire. L'accès au disque dur est réglementé dans une applet. Les applets fonctionnant sur le Web sont soumises aux restrictions suivantes dans la version 1.0 de Java : <ul style="list-style-type: none">• Aucun programme ne peut ouvrir, lire, écrire ou effacer un fichier sur le système de l'utilisateur• Aucun programme ne peut lancer un autre programme sur le système de l'utilisateur• Toute fenêtre créée par le programme est clairement identifiée comme étant une fenêtre Java, ce qui interdit par exemple la création d'une fausse fenêtre demandant un mot de passe• Les programmes ne peuvent pas se connecter à d'autres sites Web que celui dont ils proviennent.
Java est économique	Le pseudo code a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.

Environnement de développement

- L'environnement de développement Java est :
 - Multiplateformes (Linux, Windows, MacOS)
 - Open Source
 - Gratuit
 - Librement **redistribuable** dans vos applications
- Vous pouvez télécharger l'environnement de développement (**JDK**) à cette adresse :
 - <http://jdk.java.net/>
- Vous pouvez trouver l'ensemble des documentations pour les différentes version de JAVA (SE, EE...) sur le site d'oracle : <https://docs.oracle.com/en/java/> (*Attention le JDK proposé par ce site est soumis à licence commerciale.*)
 - [Java SE](#)
 - [Java EE](#)
 - [Java ME](#)

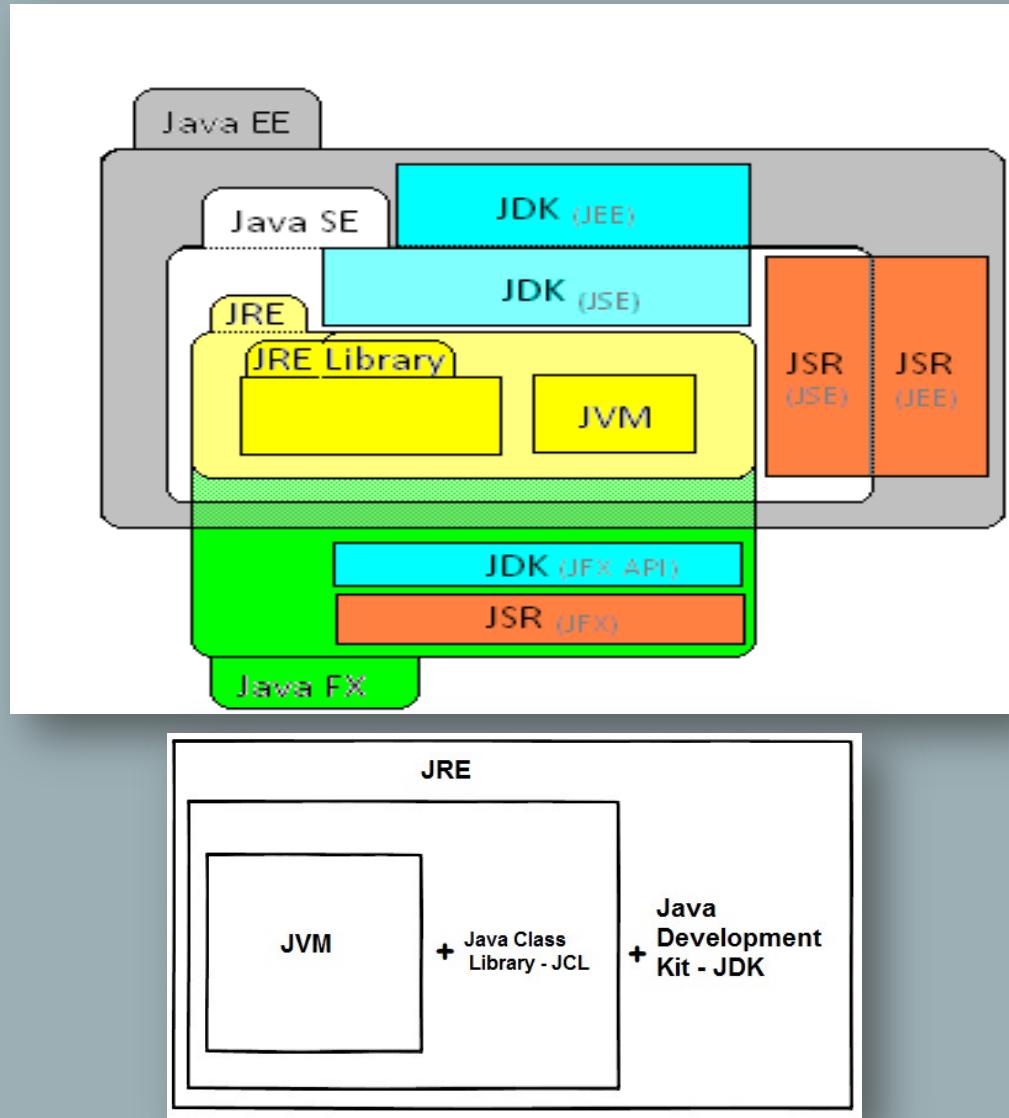
I – 2 - La plateforme java

Java de nos jours



- La plateforme Java est une des plateformes de développement logiciel les plus adoptées par les entreprises.
- Plusieurs avantages :
 - ✓ Multithreading
 - ✓ Gestion de la mémoire
 - ✓ Évolutivité
 - ✓ Développement multiplateforme
 - ✓ Haute sécurité
- La plateforme Java se décompose en 3 plateformes :
 - ❖ **JSE** (Java Standard Edition) : **destinée aux ordinateurs de bureau**.
 - ❖ **JEE** (Jakarta Entreprise Edition) : **extension de JSE, destinée aux serveurs web**.
 - ❖ **JME** (Java Micro Edition) : **destinée aux appareils portables comme les smartphones**, partageant un noyau commun avec Java SE.

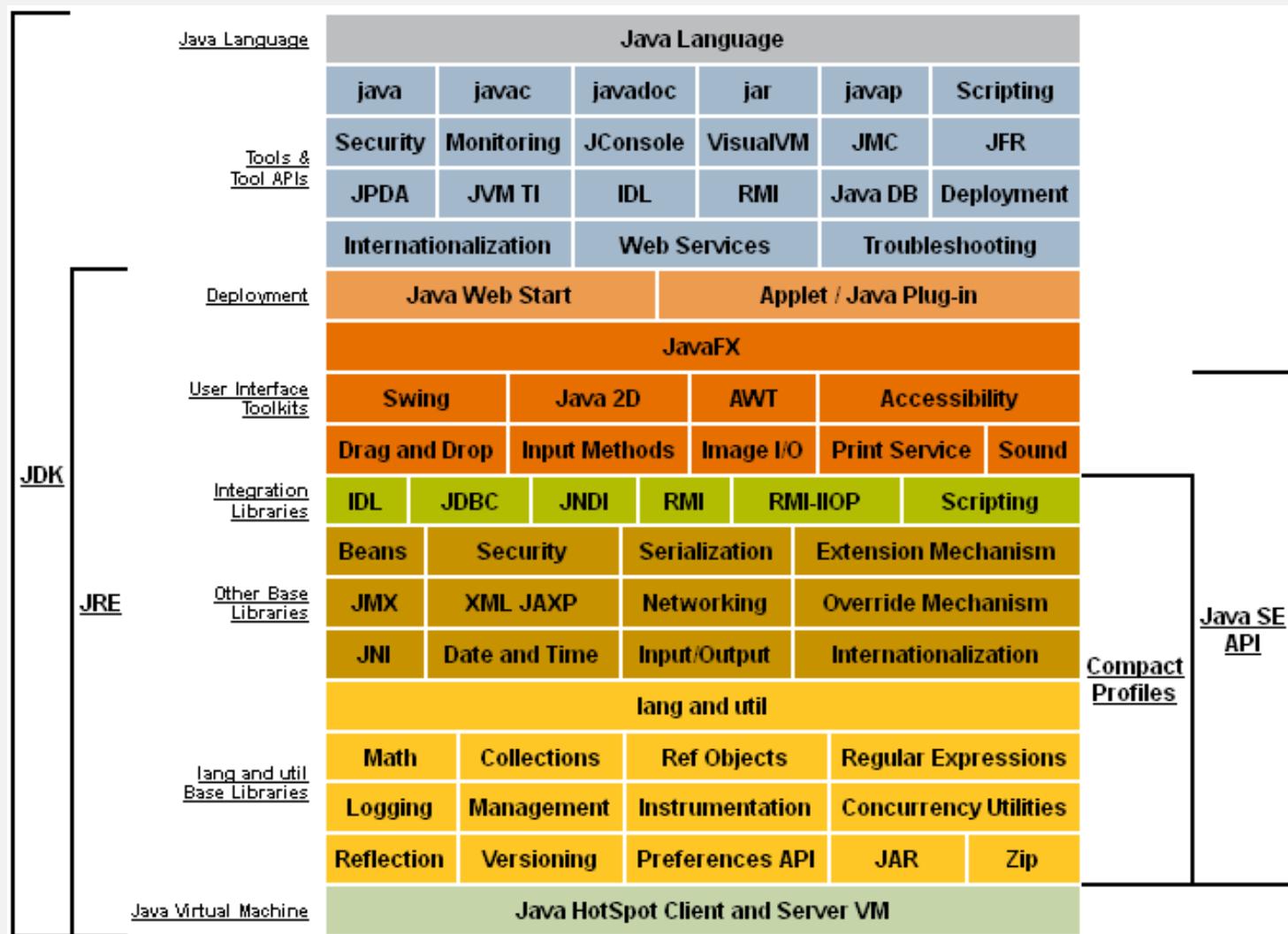
I – 2 - La plateforme java



Java de nos jours

- **JDK (Java Développement Kit)**: Le JDK désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la machine virtuelle Java. Il fournit les outils nécessaire au développement d'application java. Il comprend l'environnement d'exécution Java (JRE), un interpréteur / chargeur (Java), un compilateur (javac), un archiveur (jar), un générateur de documentation (Javadoc) et d'autres outils nécessaires au développement Java.
- **JRE (Java Runtime Environnement)** : Le JRE est l'environnement d'exécution Java. Il est intégré au JDK. Java Runtime Environment fournit la configuration minimale requise pour l'exécution d'une application Java. Il comprend la machine virtuelle Java (JVM), les classes principales et les fichiers de support.
- **JVM (Java Virtual Machine)**: La JVM est une partie très importante du JDK et du JRE car elle est contenue ou intégrée dans les deux. Quel que soit le programme Java que vous exécutez à l'aide de JRE ou de JDK, il est intégré à la machine virtuelle Java et celle-ci est chargée de l'exécution ligne par ligne du programme Java. Il est donc également appelé interpréteur.

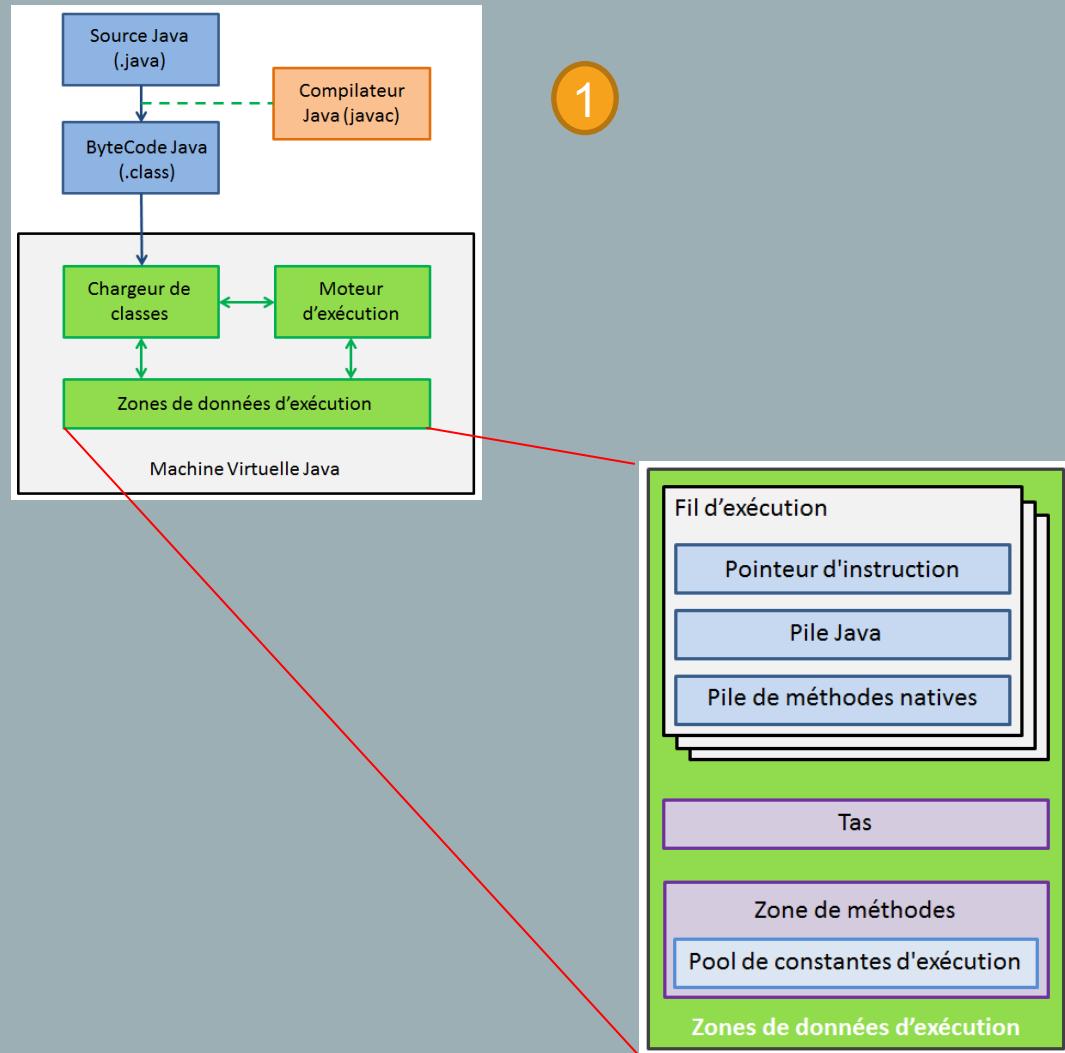
I - 2 La plateforme java



I – 3 - JVM

- La **machine virtuelle Java** ou JVM (Java Virtual Machine) est un environnement d'exécution pour applications Java.
- La **machine virtuelle** permet notamment :
 - L'interprétation du bytecode
 - L'interaction avec le système d'exploitation
 - La gestion de sa mémoire grâce au ramasse-miettes
- **La machine virtuelle ne connaît pas le langage Java** : elle ne connaît que le bytecode qui est issu de la compilation de codes sources écrits en Java.
- Il existe de nombreuses implémentations de JVM dont les plus connues sont :
 - [HotSpot](#) de Sun Microsystem (la plus utilisée).
 - [Apache Harmony](#) par la fondation Apache
 - OpenJ9 initialement IBM (puis sous licence Eclipse).
 - MRJ licence propriétaire Apple.
 - Etc...

I – 3 - JVM

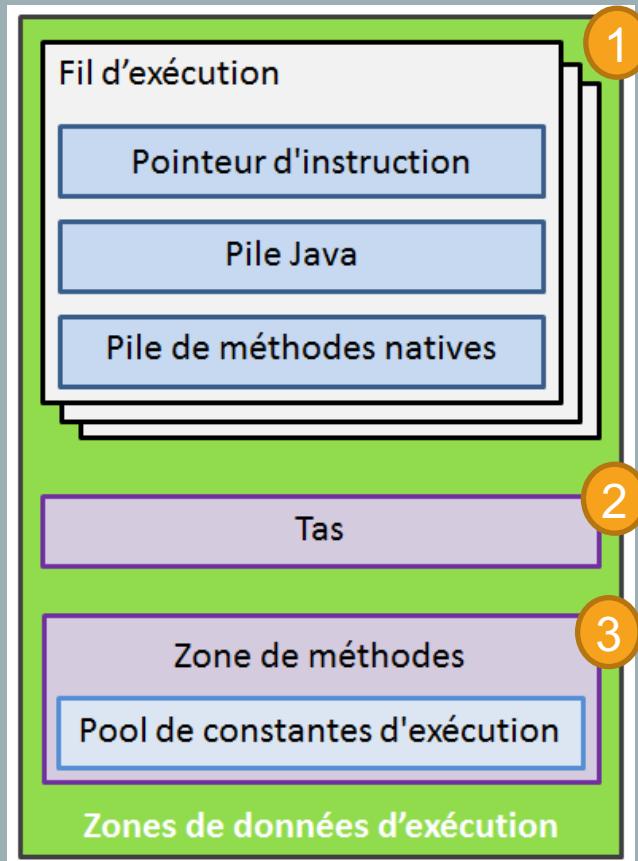


Lors de l'exécution de notre code Java :

- 1 • Le fichier .java va être compilé par le compilateur pour le transformer en byte Code avec une extension .class.
- Le **chargeur de classe (ClassLoader)** se charge de retrouver l'emplacement des bibliothèques, de lire leur contenu et de charger les classes qu'elles contiennent.
- Le **moteur d'exécution** est le Composant Central de la JVM. Il communique avec différentes zones mémoire de la JVM. Il exécute les fichiers .class.
- Le **moteur d'exécution** exécute le code d'octet qui est affecté aux zones de données d'exécution dans JVM via le chargeur de classe.
- Le **moteur d'exécution** lit le byte code et interprète (convertit) en code machine (code natif) et les exécute de manière séquentielle.

- 2 • La **JVM** définit différentes **zones de données d'exécution** qui sont utilisées durant l'exécution d'un programme.
- Certaines de **ces zones de données** sont créées lorsque la **JVM** est lancée et sont détruites lorsqu'elle s'arrête.

I – 3 - JVM

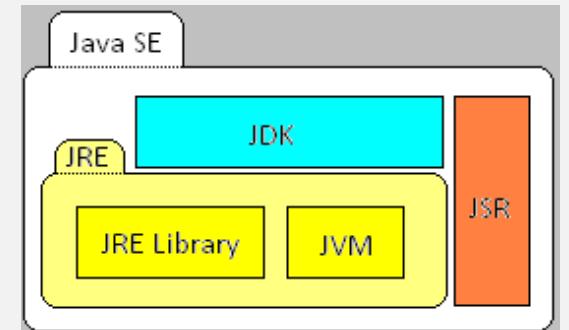


Lors de l'exécution de notre code Java :

- Les autres **zones de données d'exécution** sont propres à chaque fil d'exécution (thread).
 - Les **zones de données par fils d'exécution** sont créées lorsque le fil d'exécution est créé et sont détruites lorsqu'il se termine.
-
- La **JVM** a un **tas** partagé par tous les **fils d'exécution**.
 - Le **tas** est une zone de données d'exécution dans laquelle toutes les instances de classes et les tableaux sont stockés.
-
- Lorsque le **chargeur de classes** charge une classe, il stocke sa structure dans la **Zone de Méthodes**.
 - La **Zone de Méthodes** d'un espace mémoire partagé par tous les fils d'exécution

I – 4 - JSE

- **Java Platform, Standard Edition**, ou **Java SE** (anciennement **Java 2 Platform, Standard Edition**, ou **J2SE**), est une spécification de la plate-forme Java d'Oracle, destinée typiquement aux applications pour poste de travail.
- La plate-forme est composée, outre les [API](#) de base :
 - Des API spécialisées dans le poste client ([JFC](#) et donc [Swing](#), [AWT](#) et [Java2D](#)) ;
 - Des API d'usage général comme [JAXP](#) (pour le *parsing XML*) ;
 - De [JDBC](#) (pour la gestion des bases de données).



I-5 - La plateforme Java Enterprise Edition (Java EE)

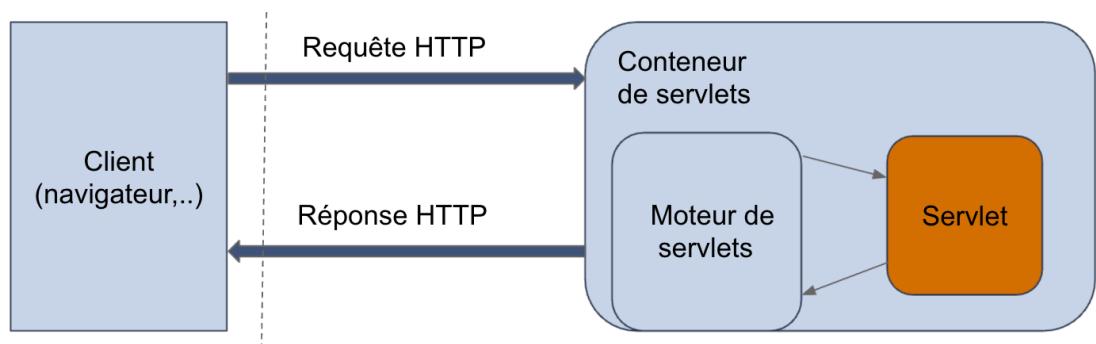
- JEE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées.
- Elle est composée de 3 parties essentielles :
 - Des modèles de composants pour développer les applications, livrés sous forme de bibliothèques de programmation.
 - Une plateforme de service intégrée par les infrastructures d'exécution, et utilisée par les composants.
 - Une infrastructure d'exécution pour héberger les applications.

I-5 Les composants Java EE

- Le modèle de développement d'applications JEE préconisé par Sun Microsystems fait intervenir trois types de composants logiciels :
 - **Servlet**
 - **JSP**
 - **EJB**
- L'objectif est de mieux séparer les traitements et donc les responsabilités de chacun de ces composants dans l'application.

I-5 Les servlets

Echange HTTP Classique

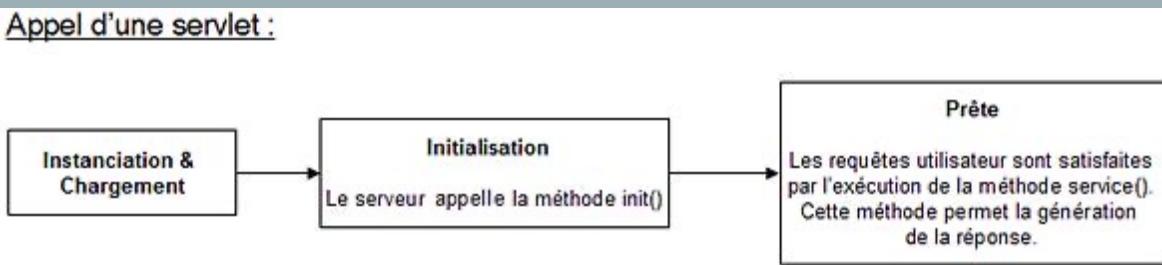


- Une servlet est une classe de l'API JEE/Servlet. Elle est conçue pour être déployée et exécutée dans un conteneur de servlet (type **Apache Tomcat**).
- Les servlets sont des composants logiciels entièrement écrits en Java.
- Son rôle est d'écouter les requêtes HTTP entrantes et d'y répondre. Elle peut répondre par une page HTML, du JSON, du XML, etc.
- Une servlet ne doit jamais être instanciée à la main.
- Le serveur est seul le maître du cycle de vie d'une servlet. Il la crée lui-même et se charge de la manipuler.

I-5 Les servlets

Cycle de vie d'une servlet

Appel d'une servlet :



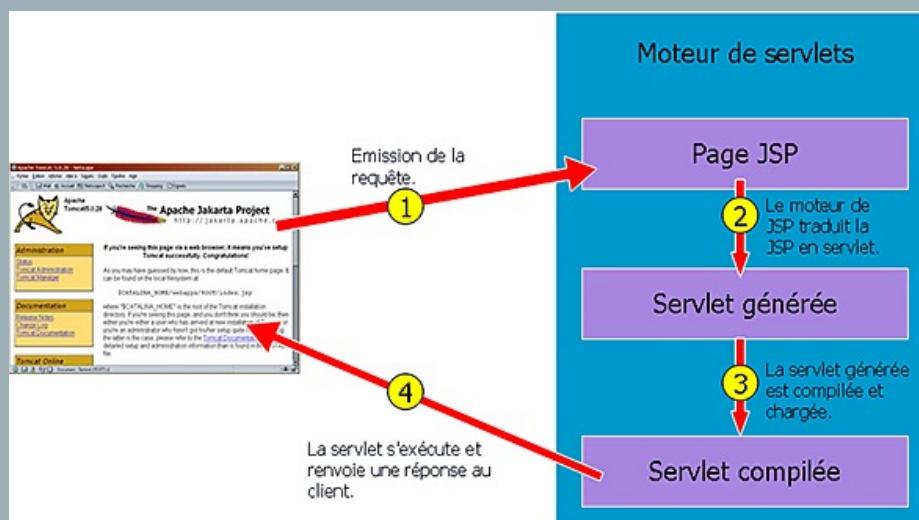
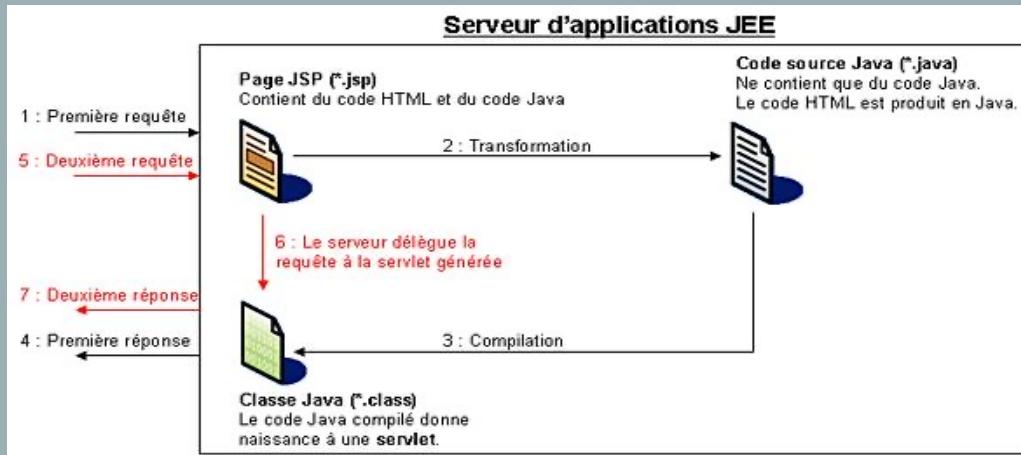
Arrêt du serveur ou de l'application :



- Une servlet étant avant toute chose une classe Java,
- Elle doit être instanciée.
- Puis interprétée par votre JVM.
- Le serveur va initialiser la servlet pour lui faire charger des informations de configuration.
- La servlet est maintenant prête à recevoir des requêtes et à renvoyer des réponses
- Lorsque l'application ou le serveur s'arrête :
 - *La servlet est détruite.*
 - *Son instance est nettoyée par la machine virtuelle Java.*

Détails JSP

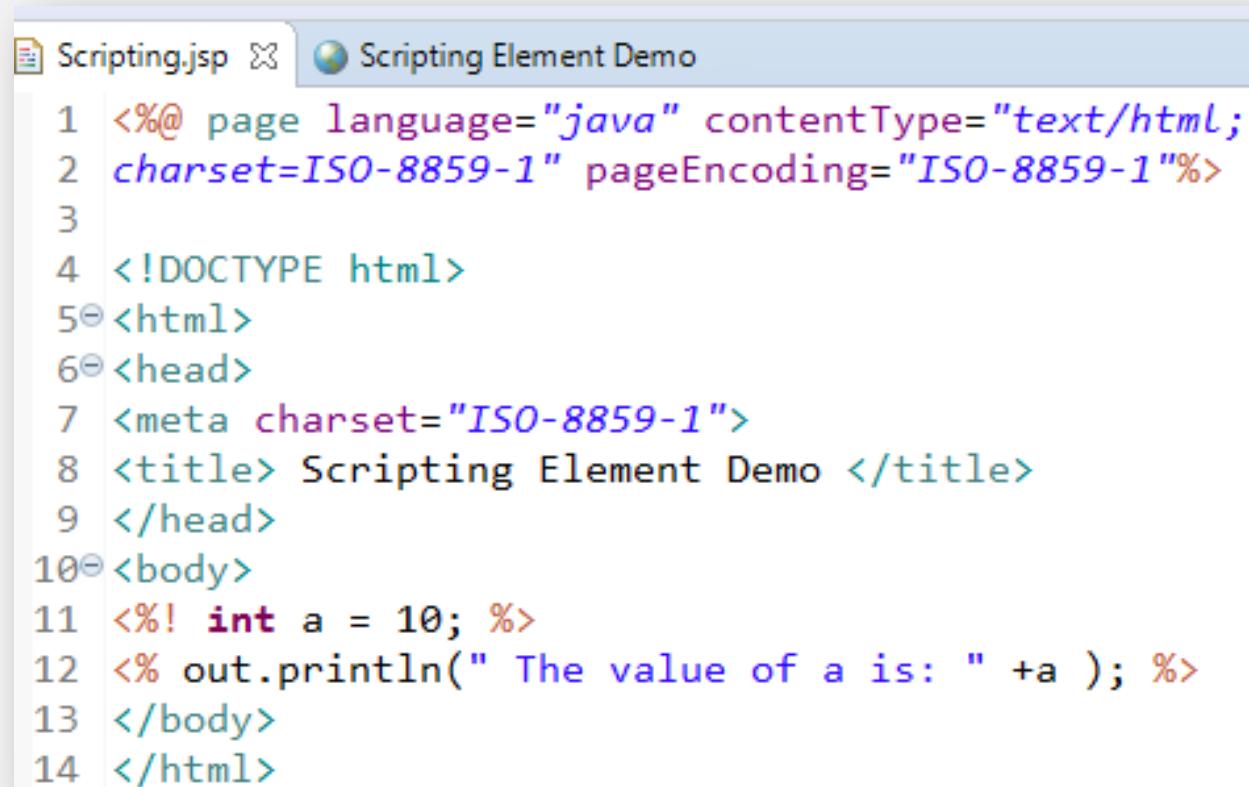
I-5 Java Server Pages : JSP



- La technologie Java Server Pages permet le développement de pages Web dynamiques.
- Une JSP est habituellement constituée :
 - De données et de tags HTML.
 - De tags JSP.
 - De scriptlets (code Java intégré à la JSP).
- Les fichiers JSP possèdent par convention l'extension .jsp.
- Le rôle d'une JSP dans une application JEE est de prendre en charge la partie visuelle de cette application en présentant les données au client.
- Les étapes :
 1. Appel page JSP via requête http par client web.
 2. Le conteneur qui la gère (moteur de servlets) capture et analyse la requête.
 3. Si la page n'a jamais été appelé ou si code modifié, le conteneur la traduit sous forme d'une servlet.
 4. Cette servlet est compilée et exécutée pour fournir une réponse à la requête web.
 5. Si la page a déjà été utilisé et si son code n'a pas été modifié, la servlet existe déjà, le conteneur va juste exécuter cette servlet pour répondre au client.

I-5 Java Server Pages : JSP

Exemple page JSP



```
Scripting.jsp ✘ Scripting Element Demo
1 <%@ page language="java" contentType="text/html;
2 charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
3
4 <!DOCTYPE html>
5<html>
6<head>
7 <meta charset="ISO-8859-1">
8 <title> Scripting Element Demo </title>
9 </head>
10<body>
11 <%! int a = 10; %>
12 <% out.println(" The value of a is: " +a ); %>
13 </body>
14 </html>
```

I-5 Entreprise javabeans: ejb

- Avec les EJB, nous abordons les composants qui ne sont plus liés à la présentation Web.
- Ils encapsulent la logique de traitement de l'application (la logique « métier »).
- Le but des EJB est de faciliter la création d'applications distribuées pour les entreprises.
- Les composants Enterprise JavaBeans sont des composants métier distribués, c'est à dire qu'ils sont invocables par le réseau.
- Comme pour la servlet, un EJB ne s'instancie pas. Il dispose d'un cycle de vie.
- Les EJB s'exécutent dans un environnement particulier : le serveur d'EJB.
- Ils jouent plusieurs rôles : accès aux services Java EE par injection, gestion des transactions, etc.

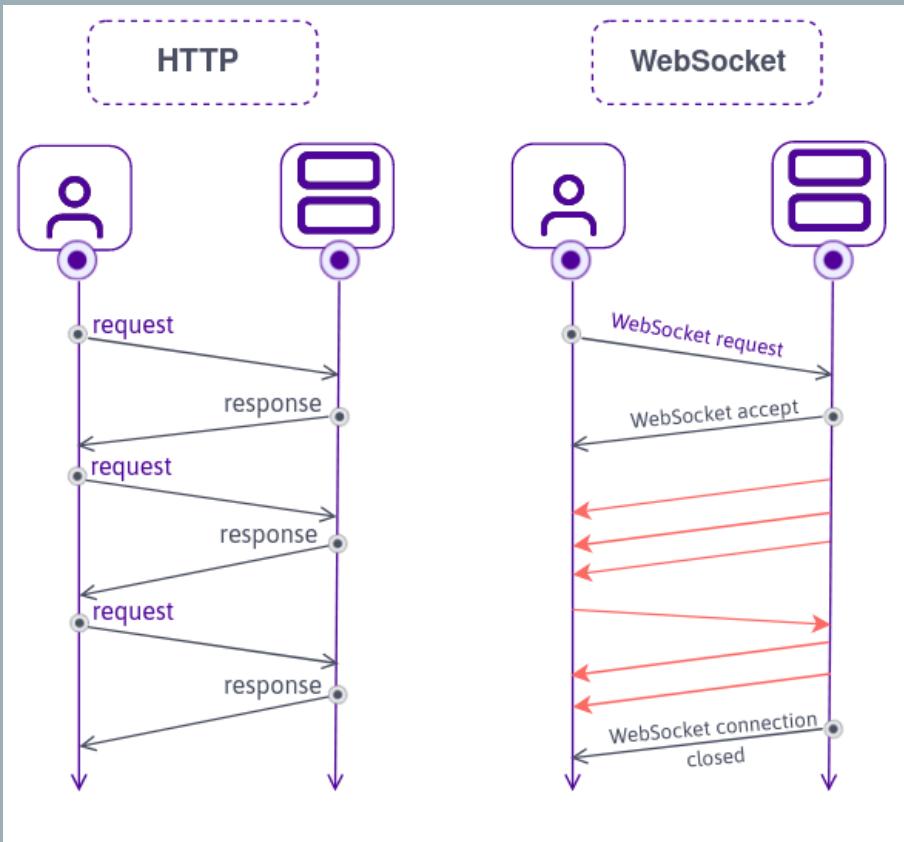
I-5 Entreprise javabeans: ejb

- Il existe plusieurs types d'EJB :
 - ✓ **Les EJB session** : *ils peuvent, selon leurs configurations, maintenir ou non des informations sur les clients et les traitements qu'ils réalisent (ils sont dit avec ou sans état : stateless ou stateful).*
 - ✓ **Les EJB orientés message (ou MDB : Message Driven-Bean)** : ils permettent de faire communiquer entre eux les différents composants d'applications clientes par envoi de données directement interprétables et utilisables.
 - ✓ **Les EJB Beans** : Les EJB pour gérer la persistance. Il s'agit des composants Java EE qui représentent les données d'un modèle relationnel de base de données. Depuis EJB 3 et JEE 5, **l'API JPA** définit les entity beans.

I-5 Les entités Java

- Les entités Java sont des objets persistants, c'est à dire que leur état est sauvegardé dans une base de données relationnelle.
- Les entités Java sont créées avec l'API de persistance Java **JPA**.
- L'API JPA utilise les annotations pour indiquer les caractéristiques de persistance des objets.
- Mapping ou mapping Objet/Relationnel :
 - On associe une classe (@entity) avec une table en base de donnée.
 - Chaque objet ou instance de cette classe est représentée par une ligne dans la table en base de donnée.
- Des éditeurs d'applications peuvent en se basant sur la spécification JPA créer des ORM comme :
 - ✓ Hibernate.
 - ✓ OpenJPA.
 - ✓ EclipseLink

I-5 les web socket



- La technologie **WebSocket** permet d'ouvrir un canal de communication interactif et bidirectionnel entre un navigateur (Client) et un serveur.
- Le protocole http fonctionne avec un système où **il est nécessaire d'avoir effectué une requête** pour obtenir une réponse.
- **WebSocket** permet de lever cette limitation.
- Il permet donc :
 - La notification au client d'un changement d'état du serveur,
 - L'envoi de données en mode « pousser » (méthode Push) du serveur vers le client, sans que ce dernier ait à effectuer une requête.

I-5 La plate-forme de service

- Il existe différents services au sein de la plate-forme JEE qui vont faciliter et permettre la communication entre les composants ou d'accéder à une base de donnée :
 - ❑ **JDBC (Java DataBase Connectivity)** : JDBC permet aux programmes Java d'accéder aux bases de données. Cette API de programmation possède la particularité de permettre un développement sans se soucier du type de la base de données utilisée : elle utilise pour ça, un pilote d'accès à la base de données.
 - ❑ **JNDI** : Elle a un double rôle :
 - Elle permet l'accès aux services d'annuaire utilisé par une entreprise.
 - Elle permet également d'implémenter un service de nommage. Cela va permettre d'identifier les services qui sont fournis ou accessibles par le serveur d'application. Un nom logique devra être attribué à ces services.

I-5 La plate-forme de service

- **JMS : Java Message Service** : Elle permet la communication entre composant de manière asynchrone. Un système de file d'attente est mis en place quand un message est envoyé à un composant et que la réponse n'est pas immédiate.
- **JavaMail** : Elle permet la création et l'envoi de message électronique via Java. Il permet l'utilisation des protocoles de messagerie comme POP3, IMAP4, SMTP, etc.
- **JAAS : Java Authentication and Authorization Service** : L'accès aux applications peut être géré de manière standard par le serveur d'applications et par simple déclarations dans les applications. Ceci permet d'éviter un développement spécifique de la partie sécurité pour chaque application, et de se reposer sur un mécanisme fourni par le serveur.
- **JTA : Java Transaction API** : Le principe de la transaction consiste à considérer un ensemble d'opérations comme une seule. une transaction est une unité de travail, en général liée à des changements de données dans une base. Exemple une opération bancaire (Débit/crédit). Cette unité est indivisible. Elle est réalisée que si elle va jusqu'au bout. JAT permet ce traitement au sein d'un serveur d'application.

I-5 - La plate-forme de service

- **RMI/IOP : (Remote Method Invocation/Internet InterORB Protocol)** : RMI fait partie de la plateforme JSE. Elle est une API Java qui permet l'appel de fonctionnalité à distance, en utilisant une communication réseau.
- **JCA : JEE Connector Architecture** : Elle va faciliter et permettre l'accès à des ressources qui ne sont pas interfacées pour interagir avec un environnement JEE. JCA est la solution de J2EE pour résoudre le problème d'intégration entre le monde J2EE et le système d'information d'entreprise (EIS).

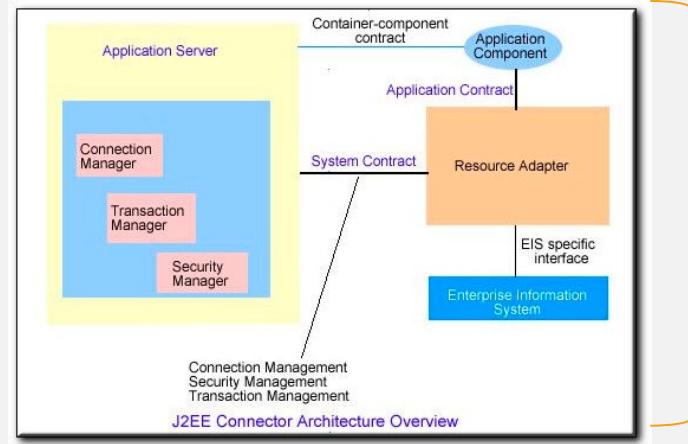
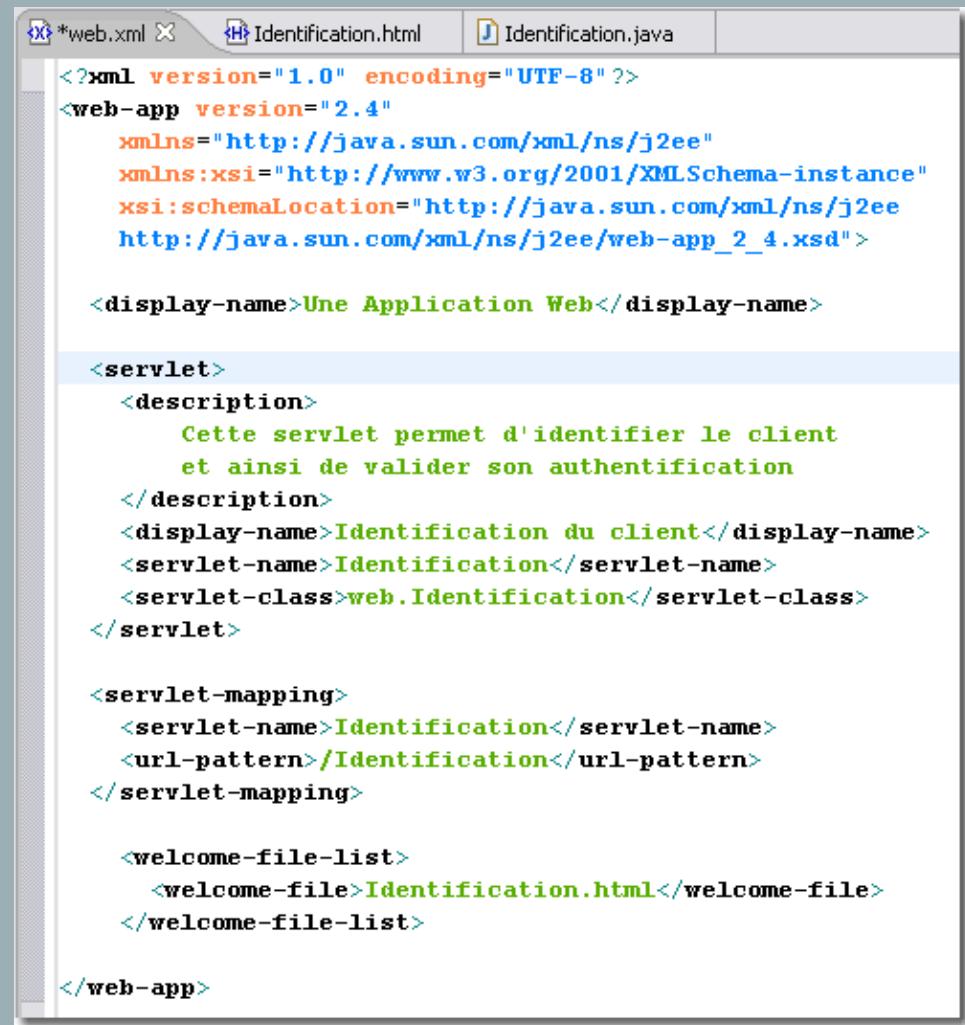


Schéma
JCA

I-5 - La plate-forme de service



The screenshot shows a Java IDE interface with three tabs at the top: 'Identification.html' (selected), 'Identification.java', and 'web.xml'. The 'Identification.html' tab contains a simple HTML page with a single line of text: 'Identification du client'. The 'Identification.java' tab shows a Java class with a constructor and a method named 'Identification'. The 'web.xml' tab displays the XML configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Une Application Web</display-name>

  <servlet>
    <description>
      Cette servlet permet d'identifier le client
      et ainsi de valider son authentification
    </description>
    <display-name>Identification du client</display-name>
    <servlet-name>Identification</servlet-name>
    <servlet-class>web.Identification</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Identification</servlet-name>
    <url-pattern>/Identification</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>Identification.html</welcome-file>
  </welcome-file-list>

</web-app>
```

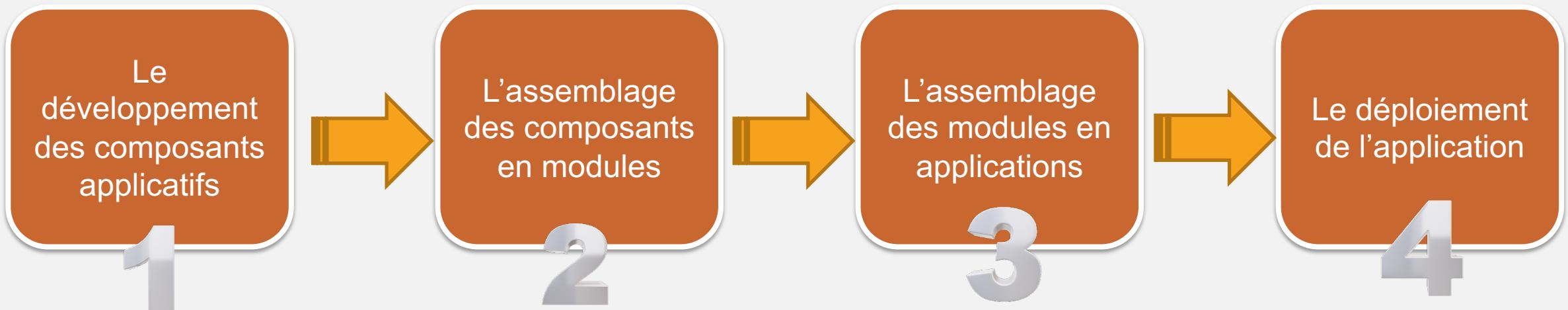
XML :

- **Ce n'est pas une API.**
- **Le XML**, pour Extensible Markup Language, désigne un métalangage informatique.
- Il est d'abord utilisé pour écrire les différents fichiers de configuration.
- Il est aussi utilisé pour la communication entre application sur le principe des web services.
- Il existe plusieurs API (JAXB, JAXP) qui vont permettre la traduction des informations contenues dans ces fichiers côté Java.
- Le langage est basé sur une arborescence de balises « ouvrantes » et « fermantes ».

I-5 - Les APPLICATIONS JEE

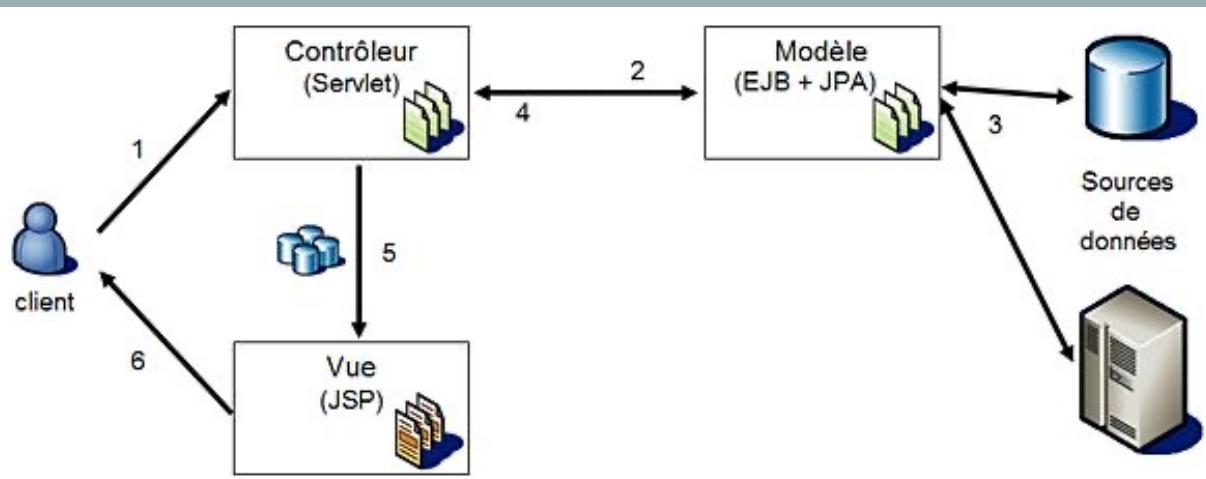
En plus de fournir un modèle de composants, **JEE** standardise également la manière dont ces composants doivent être assemblés avant de pouvoir être installés dans un **serveur JEE**.

Cycle de conception application avec
JEE :

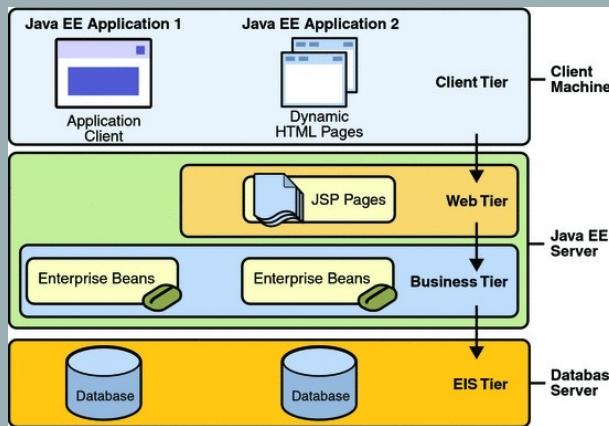
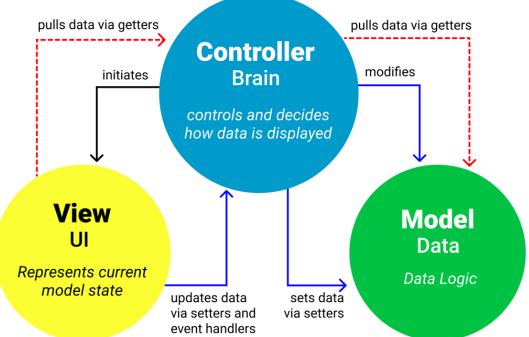


Modèle Vue Contrôleur

I-5 - Le modèle mvc



MVC Architecture Pattern



- Son objectif principal est d'apporter une séparation de la logique métier et de la logique d'affichage.
 - Elle divise l'application en trois parties distinctes : le **modèle**, la **vue** et enfin le **contrôleur**.
1. *Le client émet une requête HTTP à destination de l'application. La servlet intercepte la requête et en traite les informations.*
 2. *Les informations sont utilisées pour appeler les traitements métier.*
 3. *Les composants du modèle manipulent les données du système d'information*
 4. *Les traitements métier retournent les données résultats à la servlet, qui stocke ces données pour les rendre accessible aux JSP.*
 5. *La servlet appelle la JSP adéquate.*
 6. *La JSP s'exécute, inclut les données transmises par la servlet, et génère la réponse au client.*

I-5 - Les différents modules

- Le regroupement ou le découpage est effectué à ce niveau en fonction des rôles des différents éléments de code de l'application.
- Le regroupement de ces ressources se fait dans des modules appelés **modules de déploiement JEE**.
- Un module n'est ni plus ni moins qu'une archive au format ZIP incluant les différentes ressources, mais avec une extension particulière.

Modules Web :

- Contient les éléments nécessaires à son utilisation dans un navigateur web.
- JSP, Servlet, éléments statiques (images, html).
- Bibliothèques java sous forme de jar.
- Web.xml => **config déploiement**.
- Extension .war (webARchive).

Modules EJB :

- L'objectif des modules EJB est de fournir une archive homogène pour la livraison et le déploiement de ces composants.
- Pour être exploitable, il est nécessaire d'utiliser des fichiers spécifiques contenus dans le serveur d'application.
- Fichier configuration => ejb-jar.xml.
- Extension .jar (JavaARchive).

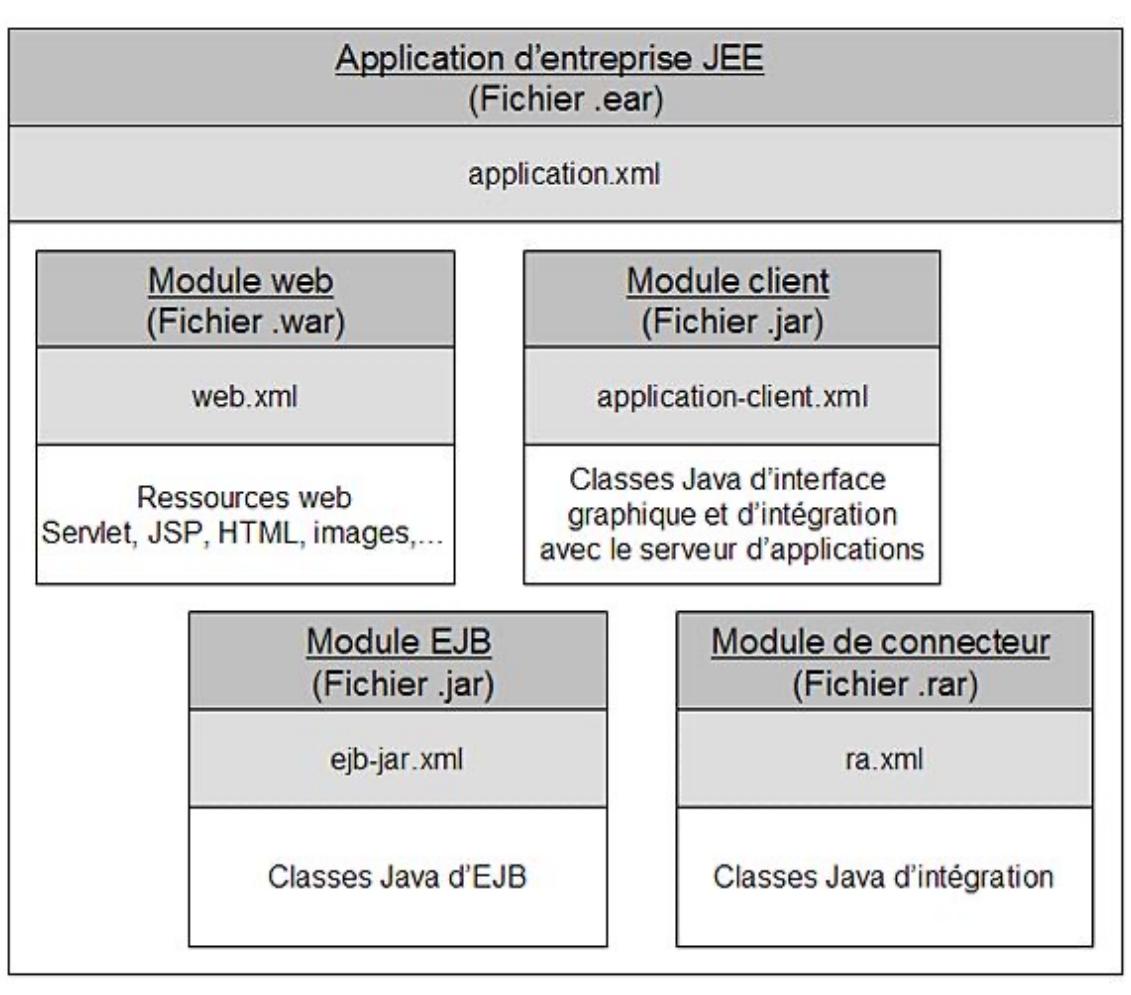
Modules Client :

- Il s'agit des EJB utilisés au travers d'une interface graphique client lourd développée en utilisant les API de programmation Java AWT, SWING JavaFX.
- Fichier configuration => **application-client.xml**.
- Extension .jar (JavaARchive).

Modules de connecteurs :

- Ce module regroupe les services de connexion comme JNDI, JDBC ou JCA.
- Fichier configuration => **ra.xml**.
- Extension .rar.

I-5 - Structure et packaging des applications



- Chaque module dispose de son fichier de configuration (exemple `web.xml` pour un module web avec une extension `.war`).
- Les outils d'installation contenus dans le serveur d'application lisent en priorité ces fichiers pour pouvoir installer correctement l'application.
- On appelle ces fichiers des descripteurs.
- L'application dans son ensemble dispose d'un descripteur => `application.xml`.
- Chacun des modules de l'application est installable seul (comme le module web).
- L'archive d'une application a une extension de type `.ear`.

I – 6 - Les langages

- À l'instar de la plateforme .NET ou de l'environnement Eclipse qui ciblent plusieurs langages, la plateforme Java vise à **supporter d'autres langages** de programmation que son langage natif Java.
- Pour langages qui utilisent la plateforme Java, on peut citer :
 - **Kotlin.**
 - **Scala.**
 - **Groovy.**
 - **Clojure.**
 - **Ceylon**
 - **Etc.**

I – 6 - Les langages

```
public class HelloWorld {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String digaHello(){  
        return "Hello " + nome + ".";  
    }  
  
    public static void main(String[] args) {  
        HelloWorld hw = new HelloWorld();  
        hw.setNome("Bruno");  
        System.out.println(hw.digaHello());  
    }  
}  
  
class HelloWorld {  
    def digaHello = {nome-> "Hello ${nome}"}  
}  
print new HelloWorld().digaHello.call("Bruno")
```

Java

Groovy

```
# Avec Java  
String maChaine = "Mon texte";  
int[] tab = {1,2,3};  
  
# Avec Groovy  
maChaine = "Mon texte"  
int[] tab = [1,2,3]
```

Groovy:



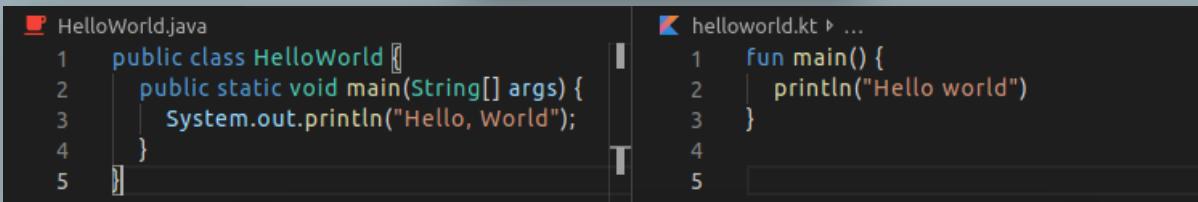
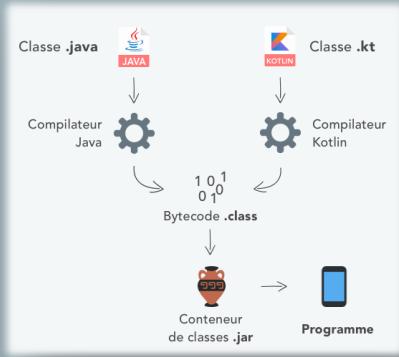
- Lancé en 2003 par James Strachan et *Bob McWhirter*, le projet Open Source Groovy est un langage de script dont les caractéristiques principales sont :
 - Un langage orienté objet pour la **JVM Java** qui s'inspire entre autres de **Python**, **Java**, **Ruby** et **Smalltalk**.
 - Un langage dynamique et agile (ex. : typage dynamique, codage facilité par le point-virgule facultatif en fin de ligne).
 - Une syntaxe proche de **Java**.
 - Prise en charge native des expressions régulières.
 - Prise en charge native de divers langages de balisage tels que XML et HTML.
 - Le **bytecode** qui est généré directement.
 - La réutilisation des librairies **Java**.
 - Permet d'écrire facilement des DSL (Domain specific Langage) **Gradle**

I – 6 - Les langages

Kotlin: 

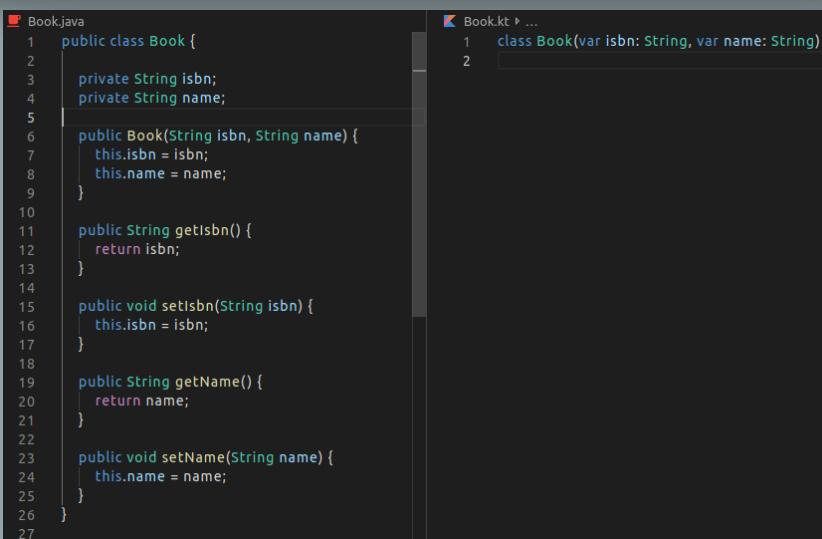
- Kotlin, développé et mis au point pendant plus de 5 ans par l'entreprise JetBrains (l'éditeur de l'IDE IntelliJ) :

- **Kotlin** est un langage de programmation orienté objet et fonctionnel.
- Première version officielle date de 2011.
- Le mot "**Kotlin**" fait référence à une petite île russe près de Saint-Pétersbourg.
- Il est statiquement typé.
- Kotlin est devenu le langage officiel pour le développement Android
- **Kotlin se veut**
 - Concis.
 - Sûr.
 - Pragmatique.
 - 100 % interopérable avec Java.



```
HelloWorld.java
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World");
4     }
5 }
```

```
helloworld.kt > ...
1 fun main() {
2     println("Hello world")
3 }
```



```
Book.java
1 public class Book {
2     private String isbn;
3     private String name;
4
5     public Book(String isbn, String name) {
6         this.isbn = isbn;
7         this.name = name;
8     }
9
10    public String getisbn() {
11        return isbn;
12    }
13
14    public void setisbn(String isbn) {
15        this.isbn = isbn;
16    }
17
18    public String getName() {
19        return name;
20    }
21
22    public void setName(String name) {
23        this.name = name;
24    }
25
26 }
```

```
Book.kt > ...
1 class Book(var isbn: String, var name: String)
```

I – 6 - Les langages

```
public class Car {  
    private final int year;  
    private int miles;  
  
    public int getYear() { return year; }  
    public int getMiles() { return miles; }  
    public void setMiles(int theMiles) { miles = theMiles; }  
  
    public Car(int theYear, int theMiles) {  
        year = theYear;  
        miles = theMiles;  
    }  
}
```



```
class Car(val year: Int, var miles: Int)
```



Scala :



- **Scala** est un langage multi-paradigme conçu à l'Ecole polytechnique fédérale de Lausanne (Martin Odersky, en 2001) :
 - Le projet a été rendu public en 2003.
 - Son nom vient de l'anglais *Scalable language* (mise à l'échelle).
 - **Scala** intègre les paradigmes de programmation orientée objet et de programmation fonctionnelle, avec un typage statique.
 - **Scala** fonctionne sur la JVM.
 - Avec Scala, les fonctions sont des objets.
 - Avec Scala, les fonctions imbriquées sont possibles.
 - Scala est utilisé pour le développement d'Apache Spark

I – 6 - Les langages

Caractéristique	Java	Groovy	Scala	Kotlin
Tout n'est pas objet (primitive)	✓	✓	✗	✗
Inférence de type	✓ Depuis Java 10	✓	✓	✓
Interface avec les méthodes abstraites	✓ Depuis Java 8	✓	✓	✓
Closure	✓ Depuis Java 8	✓	✓	✓
Duck typing (canard dactylographie)	✗	✓	✗	✗
Sécurité nulle (null safety)	✗	✓	✗	✓
Fonctions d'ordre supérieur	✓ Depuis Java 8	✓	✓	✓
Fonction en ligne	✗	✗	✗	✓
Surcharge de l'opérateur	✗	✓	✓	✓

I – 7 - Les outils de conception

- Pour faciliter le travail des développeurs, des outils d'automatisation et de conception de projet Java ont été créés comme :

-  **Maven™**

-  **Gradle**

I – 7 - Les outils de conception

MAVEN

- **Apache Maven** (couramment appelé **Maven**) est un outil de gestion et d'automatisation de production des projets logiciels Java en général et Java JEE en particulier.
- **Maven** est géré par l'organisation [*Apache Software Foundation*](#).
- Il est utilisé pour automatiser l'intégration continue lors d'un développement de logiciel.



I – 7 - Les outils de conception

MAVEN

- **L'objectif recherché est de :**
 - Produire un logiciel à partir de ses sources,
 - En optimisant les tâches réalisées à cette fin,
 - Et en garantissant le bon ordre de fabrication :
 - *Compiler, tester, contrôler, production des packages livrables*
 - *Publier la documentation et les rapports sur la qualité.*



I – 7 - Les outils de conception

MAVEN

- **L'apport de Maven est :**
 - La simplification du processus de construction d'une application.
 - De fournir les bonnes pratiques de développement.
 - Uniformiser le processus de construction d'un logiciel.
 - Vérifier la qualité du code.
 - Faciliter la maintenance d'un projet.



I – 7 - Les outils de conception

MAVEN

- **Maven** utilise un paradigme connu sous le nom de **Project Object Model (POM)** afin de :
 - Décrire un projet logiciel.
 - Décrire Ses dépendances avec des modules externes.
 - Décrire l'ordre à suivre pour sa production.



I – 7 - Les outils de conception

MAVEN

Fonctionnement général :

- C'est une application externe à installer indépendamment. Elles nécessite la présence d'un JDK et la configuration de certaines variables d'environnement.
- Dernière version à date : 3.6.0
- Elle s'exécute en ligne de commande: **mvn <option>** dans un répertoire contenant un fichier **pom.xml** décrivant les opérations.
- Elle peut aussi être intégrée à l'**IDE**.

The Maven logo consists of the word "Maven" in a bold, black, sans-serif font. The letter "a" is stylized with three colorful feather-like shapes extending from its top: one purple, one orange, and one yellow.

I – 7 - Les outils de conception

MAVEN

Terminologie :

- **Pom ou pom.xml** : Fichier décrivant le comportement que maven doit suivre lors de son exécution. Par défaut à la racine du projet
- **Repository** : répertoire contenant les dépendances déjà téléchargées dans le passé par maven.
- **Settings ou settings.xml** : Fichier contenant les configuration spécifiques d'un utilisateur (authentification, localisation du repository, etc). Par défaut dans le répertoire personnel de l'utilisateur
- **Lifecycle** : Cycle de vie maven. Ensemble des opérations effectuées par l'outil.

 Maven™

The logo for Maven consists of the word "Maven" in a bold, black, sans-serif font. The letter "A" has two colorful feather-like shapes extending from its top right corner, one purple and one orange.

I – 7 - Les outils de conception

MAVEN

Build lifecycle en détail :

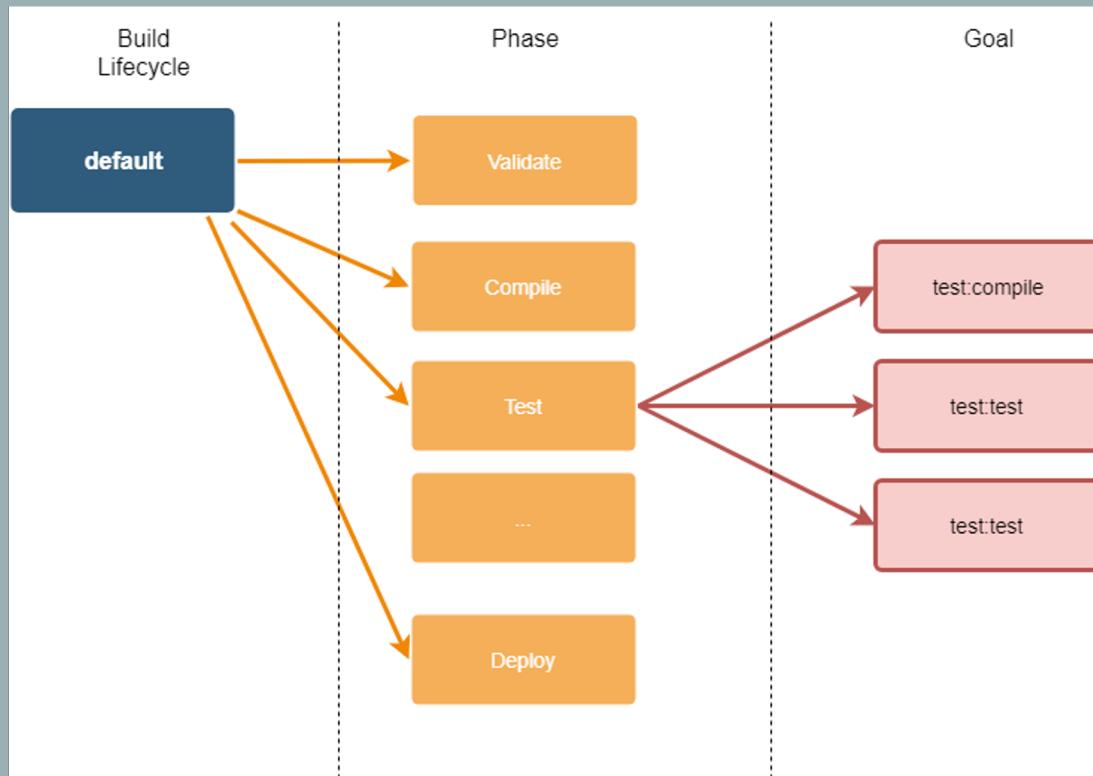
3 Builds lifecycle existants :

- **clean** : nettoie les sources de tous les éléments générés par maven, principalement créés dans un répertoire **target** à la racine
- **site** : génère un site utilisable par maven pour le projet (peu utilisé)
- **default** : permet la construction et le déploiement de l'application et de ses binaires

Ces lifecycles sont eux mêmes divisés en phases, variable d'un build à l'autre.



I – 7 - Les outils de conception



Phases du build default :

- **validate** : Contrôle structurel du pom et du projet
- **compile** : Compilation des sources en binaires
- **test** : Exécution des tests embarqués dans l'application
- **package** : Création du livrable a partir des éléments compilés
- **install** : Installation du livrable dans le repository local
- **deploy** : Envoi du livrable sur un repository distant



Par défaut, chacune des phases est bloquante pour la suivante. Ainsi, si le projet ne compile pas, ou que les tests échouent, le livrable ne sera pas créé.

I – 7 - Les outils de conception

MAVEN

Le fichier pom.xml :

- Fichier .xml régi par une xsd : un fichier de contrôle de structure. Sa rédaction doit être très rigoureuse! Syntaxe HTML pour les commentaires.
- En cas d'erreur, la commande mvn ne peut être exécutée.
- Sa structure est découpée en plusieurs blocs/balises xml, chacun d'eux étant chargé d'apporter des éléments sur le comportement de maven :
 - nom du projet
 - dépendances à ajouter
 - tests à ignorer
 - scripts à jouer, etc.



I – 7 - Les outils de conception

MAVEN

Principales balises du fichier pom.xml:

- `<groupId>` : Identifiant de regroupement du projet, servant, entre autre, à son rangement dans un repository maven. Utiliser la syntaxe de type “package”
- `<artifactId>` : Nom du projet, espaces impossibles. Préférer le kebab ou le kamel case.
- `<packaging>` : Format du livrable : pom (pour des hiérarchies de projets), jar, war, etc.
- `<version>` : Version du projet. Le projet évoluant dans le temps, ce chiffre augmentera au fil de son développement
- `<name>` : Nom du projet plus explicite, sans limite de saisie
- `<description>` : Bref descriptif du projet



I – 7 - Les outils de conception

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.2.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.ya.performance</groupId>
  <artifactId>ya-performance-server</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ya-performance-server</name>
  <description>ya-performance backend</description>
  <properties>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
    </dependency>
  </dependencies>
</project>
```

```
        <build>
          <plugins>
            <plugin>
              <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
          </plugins>
        </build>
      </project>
```

- Properties :

- <properties> : Permet la définition d'un ensemble de properties et de variables accessible dans le pom.xml. Les properties s'utilisent avec la syntaxe \${ma.properties}

- Builds :

- La balise <build> sert principalement à la déclaration des plugins qui seront utilisés lors la compilation du projet et de la création de son livrable.
- Elle contient elle même une balise <plugins>, elle même contenant une liste de balises <plugin>.
- Chacune de ces dernières servira à la description du comportement d'un plugin.
- Chaque balise <plugin> va posséder une partie commune, la balise <artifactId>, ainsi qu'une balise <configuration> qui lui est propre.

- Dependencies :

- Les dépendances peuvent être gérées au sein de la balise <dependencies>, contenant elle même une liste de <dependency>.
- Comme pour notre projet, une dépendance est identifiée par son <groupId> et son <artifactId>.
- Il est également nécessaire de préciser la <version> nécessaire, cette dernière pouvant avoir un impact direct sur les éléments que contient la librairie.
- Pour qu'une dépendance soit téléchargée, et effectivement intégrée au projet, il est nécessaire d'atteindre **au moins la phase compile du lifecycle default**.
- Un <scope> peut être ajouté à une dépendance. Il conditionne l'usage de cette dernière, ainsi que sa présence dans le packaging. Il existe plusieurs valeurs possibles.

I – 7 - Les outils de conception

Gradle

- **Gradle** est un moteur de production fonctionnant sur la plateforme Java.
- Il permet de construire des projets en Java, Scala, Groovy voire C++.
- **Gradle** est un logiciel libre distribué sous la licence Licence Apache 2.0.
- Son fonctionnement est basé sur une série de tâches de compilation qui sont exécutées de manière sérielle ou en parallèle.
- **Gradle** allie les atouts de Apache Maven et Apache Ant.
- **Gradle** permet d'écrire des tâches de construction dans un fichier de construction en utilisant le langage **Groovy**.



I – 7 - Les outils de conception

```
apply plugin: "java"

sourceCompatibility = config.java.source
targetCompatibility = config.java.target

compileJava.options.encoding = "UTF-8"

test {
    ignoreFailures = false
    // exclusion of the test suites prevents double execution of test cases.
    exclude '**/*Suite.class'
}

javadoc {
    failOnError = false
    exclude "**/gen/**"

    if(System.properties['java.home'].contains('1.8')) {
        options.addStringOption('Xdoclint:none', '-quiet') // works only if using JDK 8
    }
}
```

Gradle

- **Gradle** reprend certaines des idées fortes de **Maven** :
 - cycle de vie
 - gestion des dépendances.
- **Gradle** présente les avantages suivants :
 - Possibilité de scripter la construction en Groovy dans le fichier de construction.
 - Possibilité de changer le comportement par défaut de certaines tâches.
 - Une notation compacte pour décrire les dépendances.
 - Un moteur de production pensé pour produire des projets multi-langages.

*Migrer de **Maven** vers **Gradle** se fait très facilement pour un projet respectant les conventions **Maven**.*



I – 7 - Les outils de conception

Gradle	Maven
C'est un système de construction d'automatisation développé à Groovy	C'est un outil de gestion de build et de projet.
Gradle n'est piloté par aucun fichier xml pour la construction du projet au lieu de cela, il utilise le langage Groovy qui est spécifique au domaine. Les informations du projet sont conservées dans l'outil Gradle.	Maven est géré par un fichier xml qui contient des informations sur les dépendances, les plugins et les profils, etc.
Gradle fonctionne de manière incrémentielle et accélère la construction.	Maven n'adopte pas une approche incrémentielle et est plus lent en termes de temps de construction que Gradle.
Le script Gradle est simple, peu long et peut être compris facilement.	Maven a le fichier xml qui est descriptif, long et difficile à comprendre.
Gradle peut être personnalisé facilement car il a beaucoup de flexibilité en termes de grand nombre d'options disponibles dans l'outil avec le support IDE.	La personnalisation de Maven n'est pas facile et parfois impossible car il ne prend pas en charge IDE.
Le but d'un outil Gradle est d'avoir de nouvelles fonctionnalités dans le projet.	Le but d'un outil Maven est de terminer un projet dans un temps fixe.
En termes de performances, Gradle est meilleur car il ne traite que de la tâche en cours d'exécution et non de l'entrée ou de la sortie fournie.	Maven n'utilise pas les artefacts de construction précédents ni le cache pour créer le projet, de sorte que le temps requis pour générer un nouveau projet est plus long.
Dans Gradle, la compilation Java n'est pas une étape obligatoire.	Dans Maven, la compilation est une étape obligatoire.
Gradle est un outil relativement moderne et ses utilisateurs sont limités en nombre.	Maven est un outil familier et populaire parmi les développeurs Java.
De nombreuses dépendances pour le projet peuvent être ajoutées dans Gradle sans l'utilisation de xml.	De nombreuses dépendances peuvent être ajoutées au projet en les ajoutant au fichier xml (pom), ce qui le rend plus complexe et difficile à gérer que Gradle.
Le fichier Build.gradle contient les éléments tels que group, baseUrl et version.	Le fichier Pom.xml contient les éléments tels que, et.

I – 8 - Les IDE

- **Un environnement de développement intégré** (*IDE : Integrated Development Environment*) est un ensemble d'outils qui permet d'augmenter la productivité des programmeurs qui développent des logiciels.
- Les outils de développement incluent souvent
 - ✓ des éditeurs de texte.
 - ✓ des bibliothèques de code.
 - ✓ des compilateurs.
 - ✓ un debugger.
 - ✓ des plates-formes de test.
- Certains IDE sont open source, tandis que d'autres sont des offres commerciales.
- L'objectif d'un IDE est d'augmenter la productivité des programmeurs en automatisant une partie des activités et en simplifiant les opérations.

I – 8 - Les IDE

- **Il existe presque autant d'IDE que de langage, ci-dessous quelques uns d'entre :**
 - Visual studio code.
 - Intelij Idea.
 - Eclipse.
 - NetBeans.
 - Xcode, etc.
- **Les 2 principaux pour Java :**

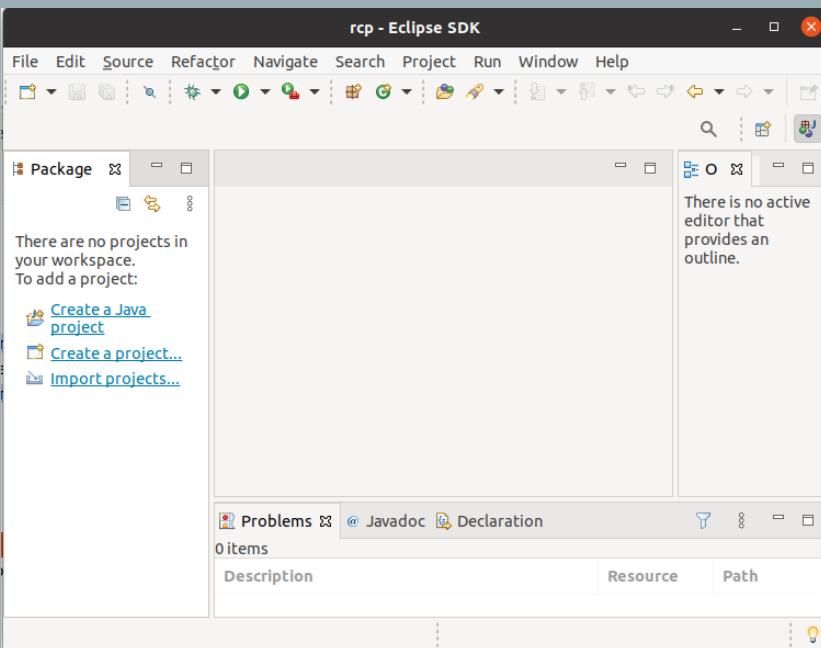
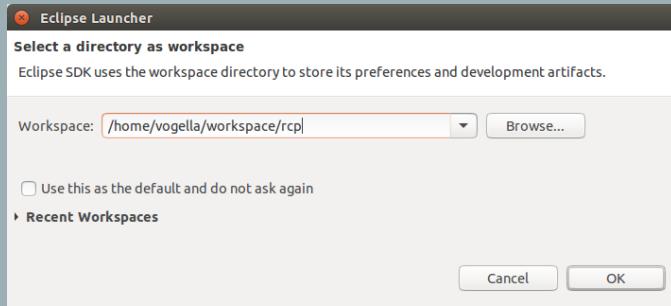


I – 8 - Les IDE



- En 2021, **l'IDE Eclipse** compte environ un million de téléchargements par mois, ce qui en fait l'un des principaux IDE pour le développement Java.
- **Eclipse** peut être étendu avec des composants logiciels supplémentaires appelés plugins.
- La distribution **Eclipse IDE for Java Developers** est conçue pour prendre en charge le développement Java standard.
 - *Il inclut la prise en charge du système de construction Maven et Gradle et la prise en charge du système de contrôle de version Git.*

I – 8 - Les IDE



- Téléchargez le programme d'installation via <https://eclipse.org/downloads/eclipse-packages/>.

• **Les éléments importants :**

1. Espace de travail et projets :

L' *espace de travail* est l'emplacement physique (chemin d'accès au fichier) pour stocker les métadonnées et (facultatif) vos artefacts de développement.

2. L'interface utilisateur :

Eclipse fournit des *vues* et des *éditeurs* pour naviguer et modifier le contenu. La vue et les éditeurs peuvent être regroupés en *perspectives* .

3. Projets Eclipse :

Un projet Eclipse contient des fichiers source, de configuration et binaires liés à une certaine tâche. Il les regroupe en unités constructibles et réutilisables. Un projet Eclipse peut avoir des *natures* qui lui sont assignées qui décrivent le but de ce projet.

I – 8 - Les IDE



- **IntelliJ IDEA** également appelé « IntelliJ », « IDEA » ou « IDJ » est un environnement de développement intégré destiné au développement de logiciels informatiques reposant sur la technologie [Java](#).
- Il est développé par [JetBrains](#) (anciennement « [IntelliJ](#) ») avec 2 versions :
 - L'une communautaire, open source, sous [licence Apache 2](#).
 - L'autre propriétaire, protégée par une [licence commerciale](#).

I – 8 - Les IDE



- **IntelliJ IDEA** apporte une étroite intégration avec quelques-uns des outils de développement libres les plus répandus, tels que [Git](#), [CVS](#), [Subversion](#), [Ant](#) et [Maven](#), [JUnit](#) et [TestNG](#).
- **Les autres langages pris en charge :**
 - [Dart](#) – [CoffeeScript](#) - [Go](#) - [Groovy](#)
 - [HTML/XHTML/CSS](#)
 - [Java](#) - [JavaScript](#) - [JSON](#) - [Kotlin](#)
 - [Markdown](#) – [PHP](#) – [Python](#) - [Ruby/Jruby](#) – [Rust](#) - [Scala](#)
 - [SQL](#) - [XML/XSL](#) - [YAML](#)

2 – Les variables

Variables

- Une variable Java est un morceau de mémoire qui peut contenir une valeur de données
- Java est un langage fortement typé : chaque variable a un type
- Une variable est définie par la combinaison d'un identifiant et d'un type
- Vous pouvez déclarer une variable avec le mot-clé 'var' - dans ce cas, le type de la variable sera défini par le compilateur
- Toutes les variables ont une portée, qui définit leur visibilité
- Vous ne pouvez pas donner un nom à la variable du mot-clé java (https://en.wikipedia.org/wiki/List_of_Java_keywords)

Définition

- Les variables sont des symboles qui associent un nom (l'identifiant) à un type et une valeur :
 - « *Type* » « *nomVariable* » = « *valeur* »;
- Il existe **deux** grandes **familles** de **variables** :
 - **Les variables primitives**
 - Ces variables contiennent une valeur en mémoire une fois initialisée
 - **Les variables de référence aux objets**
 - Ces variables ne contiennent pas de valeur mais la référence mémoire de l'objet (pointeur mémoire)
- Il existe une **infinité** de **type** de variables allant des plus simple (entiers) aux plus complexes (objets)
 - `int solde = 123; // Déclare une variable de type entier nommé solde de valeur 123`
 - `String nomFamille = « Durand » ; // Variable de type chaîne de caractères (String)`
 - `Maison grandModel = new Maison(«individuelle» , 95 , 4 , true); // Objet`

Les Variables Primitives

- La famille des variables **primitives** est composée de **quatre sous-famille** :
 - Les nombres entiers
 - *Intéger (`int`)*
 - *Le Byte (`byte`)*
 - *Le Short (`short`)*
 - *Le Long (`long`)*
 - Les caractères
 - *Le Caractère (`char`)*
 - Les nombres décimaux
 - *Le Float (`float`)*
 - *Le Double (`double`)*
 - Les variables booléennes
 - *Le Booléen (`boolean`)*
- Depuis la **version 9** de Java, il est possible d'utiliser l'**auto-typage** pour déclarer les **variables** (`var`) :
 - Important, bien que le type ne soit plus explicite, la variable reste bien typé, elle prend le type de sa valeur une fois initialisée
 - Ex: `var age = 15;` // Auto-typage de la variable en int car la valeur associée est de ce type

Les Nombres Entiers

- Détails des différents **types d'entiers** et leurs plages d'utilisation:

	Taille (bits)	Etendue
byte	8	- 128 à 127
short	16	- 32768 à 32767
int	32	- 214748648 à 214748647
long	64	- 9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

- En Java, les **type numériques** sont **OBLIGATOIUREMENT** signés:
 - C'est pourquoi, à l'inverse du langage C, le **Java** emploie uniquement le **char** pour les caractères et le **type numérique** est confié au **byte**.

Les Nombres Entiers

- Il existe plusieurs bases numériques pouvant être employées en java :
 - *Base binaire* (valeurs préfixées par `0b` – à partir de Java SE 7) :
 - `int valeur = 0b111; // 7 en base binaire`
 - *Base octale* (valeurs préfixées par `0`) :
 - `int valeur = 0o20; // 16 en base octale`
 - *Base décimale* :
 - `int valeur = 10;`
 - *Base Hexadécimale* (valeur préfixées pas `0x`) :
 - `int valeur = 0x20; // 32 en base Hexadécimale`
- Depuis Java SE 7, il est possible d'employer un séparateur de groupe de chiffre (le caractère `_`) :
 - Fonctionne avec *toutes les bases numériques* (*le nombre de chiffres regroupés est libre*)
 - *Le caractère `_` ne doit pas être positionné ni avant le premier, ni après le dernier chiffre.*
 - `int count = 1_222_333_444;`
 - `int color = 0xff_ff_00_ff;`

Les Nombres Entiers

- Déclaration d'une variable de type entier:
 - *byte* :
 - `byte variableByte; // Déclare une variable de type byte nommé variableByte`
 - *short* :
 - `short variableShort; // Déclare une variable de type short nommé variableShort`
 - *int* :
 - `int variableInt; // Déclare une variable de type Int nommé variableInt`
 - *long* :
 - `long variableLong; // Déclare une variable de type long nommé variableLong`
- Initialisation / assignation de valeur à la variable :
 - *byte* : // B à la fin d'un byte
 - `variableByte = 7B;`
 - *short* :
 - `variableShort = 138;`
 - *int* : // Déclaration & initialisation de sa valeur
 - `int variableInt = 54321;`
 - *long* : // Ne pas oublier le L à la fin d'un long
 - `long variableLong = 9876543210L;`

Les Nombres Décimaux

- Détails des différents types de nombres décimaux :

	Taille (bits)	Exemple
float	32	3,25f
double	64	0,0000325

- Tout comme le langage C, il existe seulement deux types de variables décimales :
 - La seule différence entre ces deux types est leur précision (Grâce au nombre d'octets utilisés pour stocker une valeur flottante)
 - Par défaut, une variable décimale est typée **double**. Ne pas oublier le **f** à la fin pour les **float** sinon il sera pas compilé:
 - **Compile : float f = 10.75F;**
 - **Ne compile pas: float f = 10.75;**

Les Nombres Décimaux

- Déclaration d'une variable de type décimal:
 - **float** :
 - `float variableFloat; // Déclare une variable de type float nommé variableFloat`
 - **double** :
 - `double variableDouble; // Déclare une variable de type double nommé variableDouble`
- Initialisation / modification de la valeur de la variable :
 - **float** :
 - `variableFloat = 10.75f; // Assigne à variableFloat la valeur de 10,75`
 - **double** :
 - `variableDouble = 999999.999; // Assigne à variableDouble la valeur de 999999.999`
- Déclaration de la variable et initialisation de sa valeur à la création :
 - **float** :
 - `float variableFloat = 10.75f;`
 - **double** :
 - `double variableDouble = 999999.999`

Les Nombres Décimaux

Utiliser flottant ou double ?

- La précision d'une valeur à virgule flottante indique le nombre de chiffres que la valeur peut avoir après la virgule décimale.
- La précision de float n'est que de six ou sept chiffres décimaux, tandis que les variables doubles ont une précision d'environ 15 chiffres.
- Par conséquent, il est plus sûr d'utiliser double pour la plupart des calculs.

Les Booléens (**bool**)

- Les variables de **types** Booléenne (**boolean**) ne peuvent avoir que deux états:

	Taille (bits)	Etat
boolean	1	True (vrai)
boolean	1	false (faux)

- Important, à la différence du langage c et c# , une valeur **numérique** n'est pas considérée comme une valeur **booléenne** en Java

- Il est donc **impossible** d'écrire ce genre de vérification :

➤ **if (verif == 1) { ... };** // Ne compile pas

- Il faudra plutôt écrire ce genre de vérification d'égalité:

➤ **if (verif == true) { ... };** // Compile parfaitement. Si verif est égal à true alors le code contenu dans la boucle sera exécuté

Les Booléens (**bool**)

- Déclaration d'une variable de type Booléenne (**bool**):

➤ **boolean** variableBool; // Déclare une variable de type boolean nommé variableBool

- Initialisation / modification de la valeur de la variable :

➤ variableBool = true; // Assigne à variableBool la valeur true (vrai)

- Déclaration de la variable et initialisation de sa valeur à la création :

➤ **boolean** variableBool = false; // Déclare et assigne à variableBool la valeur false

- L'état par défaut d'une variable booléenne est **false**

Les Caractères (**char**)

- Les variables de **types** Caractère (**char**) ne peuvent avoir de valeur numérique:

	Taille (Octets)	Etendue
char	2	Unicode (UTF-16)

- La valeur d'un char doit être contenue **OBLIGATOIREMENT** entre deux **simples quotes** (' ')
 - 'm' // *La lettre m en minuscule*
 - '\t' // *Une tabulation*
 - '\n' // *Un retour à la ligne*
 - '\\' // *Un caractère \ (car il a un sens particulier en Java)*
 - '\u03c0' // *Un caractère Unicode (π) à partir de son code UTF-16*

Les Caractères (**char**)

- Déclaration d'une variable de type Caractère (**char**) :
 - **char** variableChar; // Déclare une variable de type boolean nommé variableChar
- Initialisation / modification de la valeur de la variable :
 - variableChar = 'a'; // Assigne à variableChar la valeur 'a'
- Déclaration de la variable et initialisation de sa valeur à la création :
 - **char** variableChar = '\n'; // Déclare et assigne à variableChar la valeur \n (retour à la ligne)
- L'état par défaut d'une variable caractère (**char**) est '\0'

Les variables de référence aux objets

- Les variables de référence aux objets ont des types particulier faisant appel à des classes pour les définir :
 - Leur type est introduit par une majuscule
 - Ex: **String** phrase = « Ceci est une chaîne de caractères, elle est de type String »;
 - Elle peuvent exposer un certain nombre de méthodes propres à la classe d'origine
 - Ex: **String** phraseTitre = « Voici mon titre ».toUpperCase(); // Méthode Mise en Majuscule
- Les variables de référence aux objets sont immutable :
 - Une fois instanciée, les objets issus d'une classe ne peuvent plus être modifiés
 - Il faut obligatoirement que la classe à l'origine de l'instance en possède la méthode
 - Une méthode permettant la transformation d'un objet renvoie forcément à une nouvelle instance de celui-ci.

Les Chaînes de Caractères (**String**)

- Parmi les **variables de référence aux objets** un des types les plus usités est le type **String**. Il est employé afin de manipuler les chaînes de caractères :
 - Il s'agit bien d'une classe et non d'un type primitif, donc introduite par une majuscule
 - **String** phrase = « *Ceci est une chaîne de caractères, elle est de type String* »;
- A l'inverse de beaucoup de langage, en Java, un **String** n'est pas un tableau de (**char**) mais un objet
 - Comme tout objet, il hérite des méthodes de sa classe d'origine pour le manipuler
- Parmi les principales méthodes de la classe **String**, observons notamment :

Méthode	Etendue
charAt(pos)	Renvoie le caractère à la position indiquée (pos).
Length()	Renvoie la taille de la chaîne de caractères.
equals()	Compare le contenus de deux chaînes de caractères.
toLowerCase()	Produit une chaîne entièrement en minuscule à partir de l'objet courant.
toUpperCase()	Produit une chaîne entièrement en majuscule à partir de l'objet courant.

Les Chaînes de Caractères (**String**)

- Déclaration d'une variable de type chaîne de caractères (**String**) :

➤ **String** variableString ; // Déclare une variable de type String nommé variableString

- Initialisation / modification de la valeur de la variable :

➤ variableString = 'a'; // Assigne à variableString la valeur 'a'

- Déclaration de la variable et initialisation de sa valeur à la création :

➤ **String** variableString = '\n'; // Déclare et assigne à variableString la valeur \n (retour à la ligne)

- L'état par défaut d'une variable chaine de caractères (**String**) est 'null'

Les types énumérés (`enum`)

- Les types énumérés (`enum`) sont supportés depuis la version Java SE 5.0 :
 - *Un type énuméré propose plusieurs état prédefinis*
 - *Une variable basée sur un type énuméré ne pourra prendre comme valeur qu'un des états supportés.*
 - *On définit souvent un type énuméré dans un fichier du même nom (comme pour une classe) et son nom commence donc par un majuscule.*
- Exemple de définition d'un type énuméré:
- ```
- $ Public enum SystemExploitation {
 - $ Windows, Linux, MacOs
 - $ }
 - $ // Utilisation du type énuméré
 - $ SystemExploitation monOs = SystemExploitation.Linux;
 - $ if (monOs == SystemExploitation.Windows) {
 - $ System.out.println(« Votre système est compatible !»)
 - $ }
```

# Les variables

| Type                          | Size                      | Min Value                                                                                                    | Max Value                              | Wrapper Type |
|-------------------------------|---------------------------|--------------------------------------------------------------------------------------------------------------|----------------------------------------|--------------|
| <b>INTEGERS</b>               |                           |                                                                                                              |                                        |              |
| byte                          | 1 bytes                   | -128                                                                                                         | 127                                    | Byte         |
| short                         | 2 bytes                   | $-2^{15}$ (-32,768)                                                                                          | $-2^{15} - 1$ (-32,767)                | Short        |
| int                           | 4 bytes                   | $-2^{31}$ (-2,147,483,648)                                                                                   | $-2^{31}$ (-2,147,483,647)             | Integer      |
| long                          | 8 bytes                   | $-2^{63}$ (-9,223,372,036,854,755,808)                                                                       | $-2^{63}$ (-9,223,372,036,854,755,807) | Long         |
| <b>FLOATING-POINT NUMBERS</b> |                           |                                                                                                              |                                        |              |
| float                         | 4 bytes                   | approximately +-3.40282347E+38F (6- 7 significant decimal digits)<br>Java implements IEEE 754 standard Float |                                        | Float        |
| double                        | 8 bytes                   | approximately +-1.79769313486231570E+308 (15 significant decimal digits)                                     |                                        | Double       |
| <b>BOOLEAN</b>                |                           |                                                                                                              |                                        |              |
| boolean                       | virtual machine dependent | True OR false                                                                                                |                                        | Boolean      |
| <b>CHARACTERS</b>             |                           |                                                                                                              |                                        |              |
| char                          | 2 bytes                   | 0                                                                                                            | 65,535                                 | Character    |

## 2 – Les opérateurs

# Les Opérateurs de Valeur Numérique Entières

- Voici la liste des principaux opérateurs utilisable avec des valeurs numériques entières:

| Opérateurs Arithmétiques |                                                       |
|--------------------------|-------------------------------------------------------|
| +                        | Addition                                              |
| -                        | Soustraction                                          |
| *                        | Multiplication                                        |
| /                        | Division                                              |
| %                        | Modulo ( Reste de la division entière )               |
| Opérateurs Binaires      |                                                       |
| &                        | « Et » bit à bit : $0b010101 \& 0b101011 == 0b000001$ |
|                          | « Ou » bit à bit : $0b010101   0b101010 = 0b111111$   |
| ^                        | « Xor » bit à bit : $0b010101 ^ 0b111111 = 0b101010$  |

# Les Opérateurs de Valeur Numérique Décimale

- Voici la liste des principaux opérateurs utilisable avec des valeurs numériques décimale:

| Opérateurs Arithmétiques   |                                             |
|----------------------------|---------------------------------------------|
| +                          | Addition                                    |
| -                          | Soustraction                                |
| *                          | Multiplication                              |
| /                          | Division                                    |
| Opérateurs de comparaisons |                                             |
| ==                         | Egalité (Renvoie un booléen)                |
| !=                         | Différence (Renvoie un booléen)             |
| <                          | Infériorité stricte (Renvoie un booléen)    |
| <=                         | Infériorité ou égalité (Renvoie un booléen) |
| >                          | Supériorité stricte (Renvoie un booléen)    |
| >=                         | Supériorité ou égalité (Renvoie un booléen) |

# Les Opérateurs de Valeur Numérique

| Opérateurs d'affectation |                                  |                             |                                 |
|--------------------------|----------------------------------|-----------------------------|---------------------------------|
| =                        | Ex : <code>a = 2 ;</code>        |                             |                                 |
| <code>+=</code>          | Ex : <code>a+= 2 ;</code>        | <i>Equivaut quasiment à</i> | <code>a = a + 2 ;</code>        |
| <code>-=</code>          | Ex : <code>a-= 2 ;</code>        | <i>Equivaut quasiment à</i> | <code>a = a - 2 ;</code>        |
| <code>*=</code>          | Ex : <code>a*= 2 ;</code>        | <i>Equivaut quasiment à</i> | <code>a = a * 2 ;</code>        |
| <code>/=</code>          | Ex : <code>a/= 2 ;</code>        | <i>Equivaut quasiment à</i> | <code>a = a / 2 ;</code>        |
| <code>%=</code>          | Ex : <code>a%= 2 ;</code>        | <i>Equivaut quasiment à</i> | <code>a = a % 2 ;</code>        |
| <code>&amp;=</code>      | Ex : <code>a&amp;= 2 ;</code>    | <i>Equivaut quasiment à</i> | <code>a = a &amp; 2 ;</code>    |
| <code> =</code>          | Ex : <code>a = 2 ;</code>        | <i>Equivaut quasiment à</i> | <code>a = a   2 ;</code>        |
| <code>^=</code>          | Ex : <code>a^= 2 ;</code>        | <i>Equivaut quasiment à</i> | <code>a = a ^ 2 ;</code>        |
| <code>&lt;&lt;=</code>   | Ex : <code>a&lt;&lt;= 2 ;</code> | <i>Equivaut quasiment à</i> | <code>a = a &gt;&gt; 2 ;</code> |
| <code>&gt;&gt;=</code>   | Ex : <code>a&gt;&gt;= 2 ;</code> | <i>Equivaut quasiment à</i> | <code>a = a &gt;&gt; 2 ;</code> |

# Les Opérateurs de Valeur Booléenne

- Voici la liste des principaux opérateurs utilisable avec des valeurs Booléennes:

| Opérateurs Logiques     |                                                                     |
|-------------------------|---------------------------------------------------------------------|
| <code>&amp;&amp;</code> | « Et » logique ( Ex : <code>true &amp;&amp; false == false</code> ) |
| <code>  </code>         | « Ou » logique ( Ex : <code>true    false == true</code> )          |
| <code>!</code>          | « Not » logique                                                     |

# Les Opérateurs de Chaînes de Caractères (**String**)

- L'opérateur de comparaison `==` entre deux **String** requière une attention particulière:
  - Il compare les **valeurs** contenu dans les **variables**. Dans le cas d'un **String**, sa **valeur** est l'adresses (**pointeur mémoire**) de la chaîne de caractère (donc de l'objet).
    - L'opérateur `==` entre deux **String** reviens donc à demander s'il s'agit bien de la **même instance**
  - La méthodes `equals()` compare les contenus des deux chaînes de caractères
    - `-$ String nom = « toto»; // Déclaration d'une variable de type String nommé nom de valeur toto`
    - `-$ String part = « to»; // Déclaration d'une variable de type String nommé part de valeur to`
    - `-$ String autreNom = part + part; // Déclaration String nommé autreNom de valeur « to » + « to »`
  - `-$ System.out.println( nom == autreNom ); // false`
  - `-$ System.out.println( nom.equals( autreNom ) ); // true`
- `-$ String notANewObject = nom; // nom prend donc comme valeur le pointeur mémoire`
- `-$ System.out.println( nom == notANewObject ); // true`
- `-$ System.out.println( nom.equals( notANewObject ) ); // true`

# Les Opérateurs ++ et --

- Les **opérateurs d'incrémentation** ( `++` ) et de **décrémentation** ( `--` ) permettent **d'ajouter ou retirer** la valeur **1** à une variable numérique.

- Utilisation **préfixée** des **opérateurs** `++` et `--` :

```
-$ Int value = 10; // On initialise value à 10
-$ ++value; // On incrémente de 1
-$ System.out.println(value); // Affiche 11
-$ --value; // On décrémente de 1
-$ System.out.println(value); // Affiche 10
```

- Cas d'une **somme** de deux variables **préfixée** (incrémentée avant l'emploi) :

```
-$ Int premier = 10; // On initialise premier à 10
-$ Int deuxieme = 20; // On initialise deuxieme à 20
-$ int result = ++premier + ++deuxieme; // Somme des deux variables préfixée
-$ System.out.println(result); // Affiche 32 car incrémantation avant la somme
```

# Les Opérateurs ++ et --

- Utilisation **postfixée** des opérateurs ++ et -- :

```
-$ Int value = 10; // On initialise value à 10
-$ value++;
-$ System.out.println(value); // Affiche 11
-$ value--;
-$ System.out.println(value); // Affiche 10
```

- Cas d'une somme de deux variables **postfixée** (incrémentée après l'emploi) :

```
-$ Int premier = 10; // On initialise premier à 10
-$ Int deuxieme = 20; // On initialise deuxieme à 20
-$ int result = premier++ + deuxieme++; // Somme des deux variables postfixée
-$ System.out.println(result); // Affiche 30 car incrémantation après la somme
```

- La variable **préfixée** est d'abord incrémentée/décrémentée avant d'être **employée** tandis que la **postfixée** est d'abord **employée** avant d'être incrémenté/décrémenté

# Les autres opérateurs Java

- Quelques autres opérateurs du langage Java:

| Opérateurs ( Divers) |                                                                        |
|----------------------|------------------------------------------------------------------------|
| new                  | Opérateur d'allocation mémoire                                         |
| []                   | Opérateur permettant l'accès aux éléments d'un tableau                 |
| ?:                   | Opérateur conditionnel (Seul opérateur Java acceptant trois opérandes) |
| .                    | Opérateur de traversé                                                  |
| instanceof           | Permet de savoir si un objet est compatible avec un type de donné      |

## 3 – Les Instructions

# Vue d'ensembles des instructions applicables en Java

- Définition d'une instruction :

- Une instruction permet de lancer des actions (un test, une boucle, une assertion ... )
- Toute les instructions se termine par le caractère « ; »
  - Ex : `int age = 10;` // Instruction de type déclaration de variable
  - Ex : `System.out.println(" Age = " + age);` // Instruction faisant un appel de méthode
  - Ex : `if (age >= 18) System.out.println(" Vous êtes majeur ");` // Instruction de condition
- Tout oublier d'un « ; » entraîne systématiquement une erreur à la compilation

- Le bloc d'instructions :

- Un **bloc d'instructions** commence par « { » et se termine par « } »
- Un **bloc d'instructions** peut **contenir une ou plusieurs instruction(s)**

- Ex : `if (age >= 18){` // Si condition remplie on rentre dans le bloc d'instructions  
`System.out.println(" Vous êtes majeur ");` // Exécution de la 1ere instruction  
`System.out.println(" Vous pouvez voter ");` // Exécution de la 2nd instruction  
`}` // Fermeture et sortie du bloc d'instructions

# Les blocs d'instructions

- Une variable déclarée dans un bloc d'instruction voit sa durée de vie limité à la durée du bloc qui la contiens :
    - Quand on déclare une variable locale à un bloc d'instructions, un espace de mémoire est réservé sur la pile d'exécution afin d'y stocker la valeur de la variable
    - A la fin de l'exécution de ce bloc, la variable est supprimé de la pile d'exécution : On ne peux plus y accéder
- Ex :
- ```
{  
    int age = 10;  
    float taille = 1,80;  
    System.out.println(" Vous avez : " + age + " ans ");  
    System.out.println(" Vous mesurez : " + taille + " m");  
} // Une fois le bloc terminé, l'ensembles des instructions sont supprimées. Ici taille et age sont supprimés
```

Les instructions applicables en Java : if (else)

- Définition de l'instruction de condition **if** (Si) **else** (Sinon) :
 - La condition Si (**if**) est nécessaire afin de comparer une/des variable(s) et appliquer une/des instruction(s) si les conditions sont remplies
 - Voici la syntaxe d'une instruction (**if**) en Java :

```
if ( «Condition» ) {                                     // Si la condition est remplie
    InstructionAppliquee();                            // Instruction appliquée si condition remplie
}
                                         // Sortie de la condition Si
else {
    AutreInstructionAppliquee();                      // Instruction appliquée si condition non remplie
}
                                         // Sortie de la condition Sinon
```
- Notez que le bloc d'instruction **else** est facultatif mais s'il est présent, il est de bonne pratique qu'il contienne des instructions

Les instructions applicables en Java : **else if**

- Définition de l'instruction de condition **else if** (Sinon Si) :
 - La condition **else if** est nécessaire afin de comparer une/des variable(s) à plusieurs conditions et appliquer une/des instruction(s) différentes pour chaque condition
 - Voici la syntaxe d'une instruction **else if** en Java :

```
if ( « Condition1») { // Si la condition1 est remplie  
    Instruction1Appliquee(); // Instruction appliquée si condition1 remplie  
}  
  
else if ( « Condition2») { // Sinon Si (établissement d'une autre condition)  
    Instruction2Appliquee(); // Instruction appliquée si Condition2 remplie  
}  
  
else { // Sinon (Aucune condition n'est remplie, cas par défaut)  
    AutreInstructionAppliquee(); // Instruction appliquée si cas par défaut  
}
```

Les instructions applicables en Java : Boucle **for**

- Définition de l'instruction de boucle **for** (pour) :
 - Les boucles **for** testent une condition et exécutent le bout de code attaché à la boucle tant que la condition est remplie.
 - Elles sont le plus souvent utilisées lorsque l'on sait combien de fois on souhaite exécuter le bout de code attaché à la boucle.
 - Voici la syntaxe d'une boucle **for** en Java :

```
for ("InitialisationVariableBoucle" ; "ConditionDeRebouclage" ; "IncremetationDeLaBoucle" ) {  
    "Action à appliquer à chaque boucle"();
```

}

➤ Ex: `for (int i = 0 ; i < 3 ; i++) {`

```
System.out.println(i + 1 + " Instruction(s) dans la boucle for ");
```

➤ Résultat dans la console : - \$ 1 Instruction(s) dans la boucle for

- \$ 2 Instruction(s) dans la boucle for

- \$ 3 Instruction(s) dans la boucle for

Les instructions applicables en Java : Boucle **for (each)**

- Définition de l'instruction de boucle **for (each)** (Pour chaque) :
 - Les boucles **for (each)** permettent de traiter les éléments d'une collection sans avoir à gérer l'index de boucle.
 - Voici la syntaxe d'une boucle **for (each)** en Java :

```
« Type » [] « nomTableau » = { « valeur1 », « valeur2 », « valeur3 », ... } ;  
for (« Type » « nomObjet » : « nomTableauObjet » ) {  
    System.out.println("nomObjet");  
}
```
- Cette variante de l'instruction **for** fonctionne avec:
 - Les tableaux traditionnels (*exemple ci-dessus*)
 - Les collections basées sur l'interface `java.util.List(ArrayList, Vector, ...)`
 - Les collections basées sur l'interface `java.util.set(HashSet,...)`
 - T plus généralement tout type implémentant l'interface `java.util.Iterable`

Les instructions applicables en Java : Boucle **while**

- Définition de l'instruction de boucle **while** (tant que) :
 - Le code est exécuté tant que le booléen est vrai. Si avant l'instruction **while**, le booléen est faux, alors le code de la boucle ne sera jamais exécuté. Voici la syntaxe :

```
"ExpressionInitialisationBoucle"  
while ("ConditionDeRebouclage")  
{  
    Action à appliquer à chaque boucle() ;  
    IncremetationDeLaBoucle() ; // Si besoin  
}
```

Exemple: **int** i = 0 ;

```
while ( i < 5 ){  
    System.out.println( i + " Instruction(s) dans la boucle while" );  
    i++ ;  
}
```

Les instructions applicables en Java : **switch**

- Définition de l'instruction **switch** (Switchcase) :

- L'instruction **switch** est utile quand vous devez gérer beaucoup de **if** / **else if** / **else**.
- Elle a une syntaxe plus courte et plus appropriée pour ce type de cas.
Voici sa syntaxe ci-contre :
- Ici dans notre exemple de fonctionnement de cette instruction, on remarque la ligne **case 'valeur3' : 'valeur4'** :
 - Cette ligne veut dire que si la variable 'variable' vaut soit 'valeur3' soit 'valeur4', alors on exécutera « Instruction3 ».
 - Vous pouvez mettre autant de valeurs que vous souhaitez.
 - Si aucune valeur ne correspond, les instructions contenues dans le bloc **default** s'exécutent)

```
switch (variable)
{
    case 'valeur1':
        Instruction1;
        break;
    case 'valeur2':
        Instruction2;
        break;
    case 'valeur3' : 'valeur4'
        Instruction3;
        break;
    default :
        DefaultInstruction1;
        break;
}
```

4 – Les Tableaux

Comprendre la création et la gestion des tableaux

- Définition d'un tableau en Java :
 - Un **tableau** est une **structure de données** qui contient plusieurs **éléments** du même type
- Allocation d'un tableau en Java :
 - Un tableau doit être alloué dans la mémoire avec : "**new**"
 - Allocation d'un tableau de "**n**" éléments ayant pour type "**type**":
==> new "type"["nb"];
Exemple: `new int[8]` // Alloue un tableau de 8 entiers
- L'opérateur "**new**" renvoie une **référence** vers un objet "**tableau**", (une référence est un identifiant unique d'une structure de données)
L'exemple : `new int[8]` renvoie une référence vers ce tableau:
`[0,0,0,0,0,0,0,0]` // Java initialise à 0 chaque élément d'une allocation)

Comprendre la création et la gestion des tableaux

- Déclaration d'un tableau en Java :
 - Il n'y a pas de variable de type "tableau" en Java car ce sont des objets.
 - En revanche, on peut déclarer une variable de type référence vers un objet tableau :
`==> "type"[] var;`
"var" est une variable de type référence vers un tableau contenant des éléments de type "**type**"
 - Exemple: `int[] tab = new int[8]` // *tab est la variable référence d'un tableau de 8 entiers*
 - Avec cet exemple: `tab = [0,0,0,0,0,0,0,0];`
 - Java a référencé dans la variable "tab" un tableau de 8 entiers dont la valeur a été initialisée à 0.
- Déclaration avec allocation de valeur
 - On peut aussi allouer un tableau et l'initialiser en même temps:
`==> "type"[] var = {X1, X2, X3 ...Xn};`
 - Dans ce cas, la longueur du tableau est déduite du nombre "n" d'éléments initialisations.

Comprendre la création et la gestion des tableaux

- Déclaration d'un tableau avec allocation de valeur (suite)
 - Exemple de déclaration avec initialisation:

```
==> float[] tab = { 3.6, 2.89, 3.26, 23.42, 3.1416 };
```

 - Dans ce cas, Java alloue le tableau (fait un new) en déduisant la taille du nombre d'éléments [5]
 - Initialise la valeur des éléments du tableau à leurs valeurs respectives.
 - Donne à "tab" la valeur de référence vers le tableau
- Accéder et manipuler les valeurs d'un tableau
 - Connaitre la taille du tableau avec:

```
==> « monTableau ».length ; // retourne le nombre d'éléments dans le tableau
```
 - Accéder à un élément du tableau avec:

```
==> « monTableau »[indice] ; // retourne l'élément à cet indice dans le tableau
```

Comprendre la création et la gestion des tableaux

- Accéder et manipuler les valeurs d'un tableau (suite)
 - Exemple d'accès à une valeur dans un tableau:
==> `int[] tab = { 1, 2, 3, 4, 5 };`
Dans ce cas, `tab[3] = 4` ;
 - Les éléments sont indexés à partir de 0, dans notre cas de 0 ,1 , 2 , 3 , 4.
 - Un tableau possède des éléments allant de 0 à `tab.length-1`
 - Un accès en dehors des limites du tableau provoque systématiquement une erreur à l'exécution de type (`ArrayOutOfBoundsException`)