

Optez pour une transformation digitale intelligente

L'INTELLIGENCE ARTIFICIELLE AU SERVICE DE VOTRE BUSINESS.



JAVA



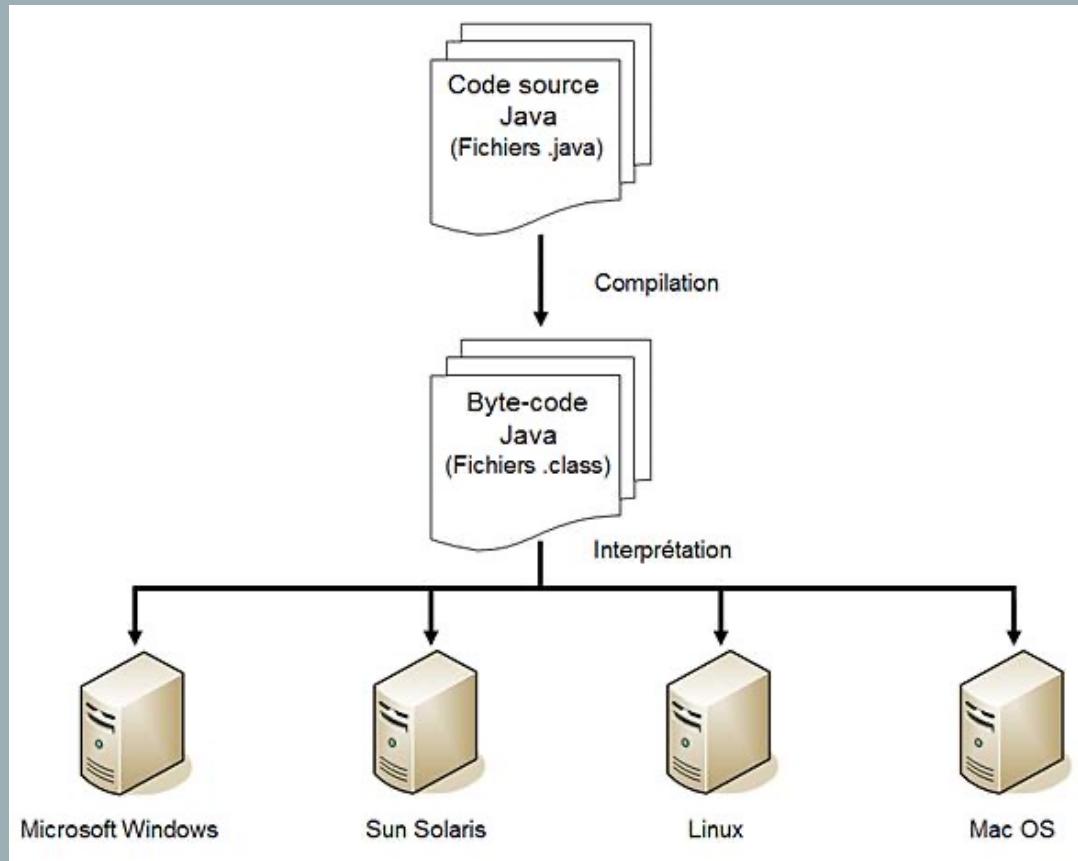
SOMMAIRE

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



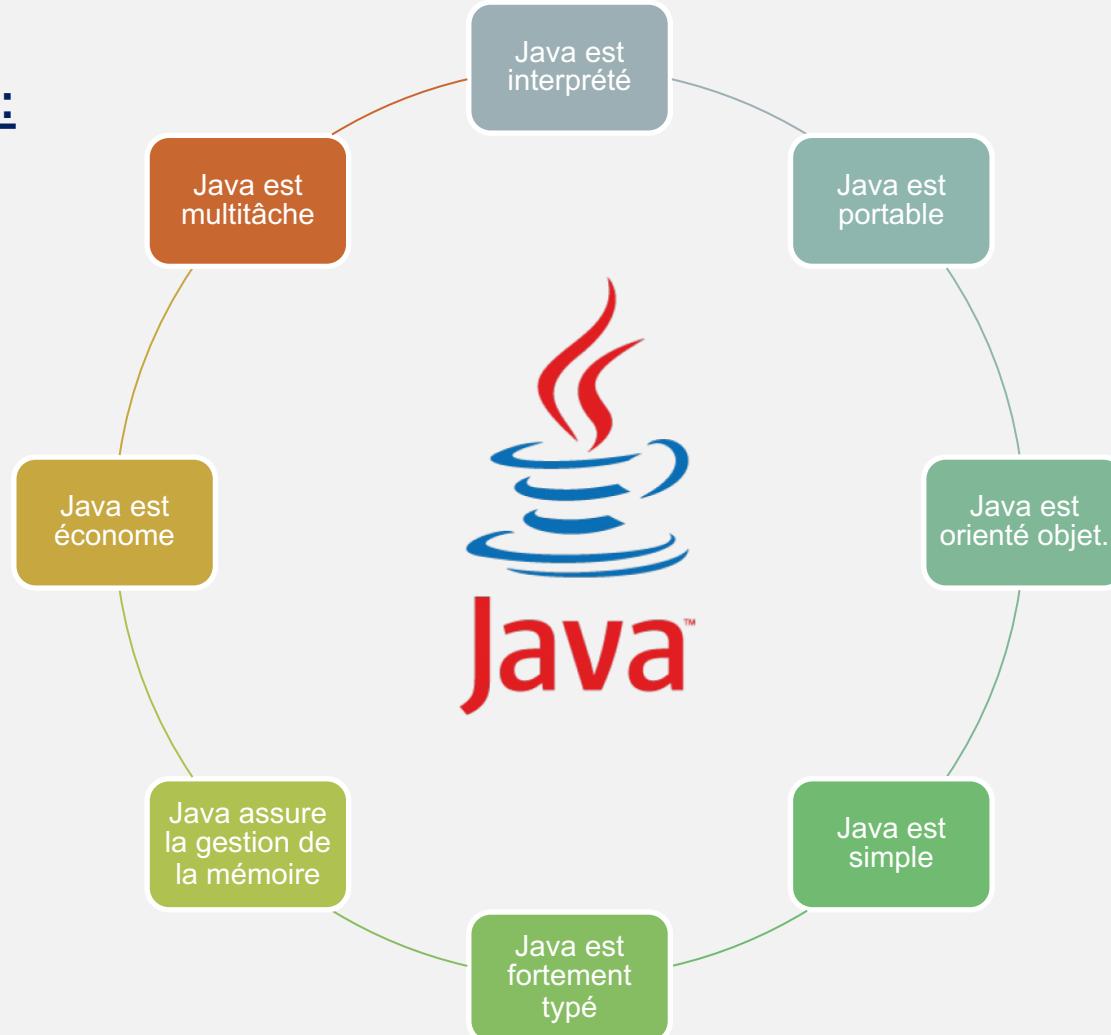
WORA (*Write Once, Run Anywhere : écrire une fois, exécuter partout*).

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

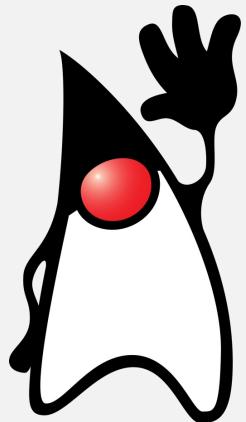
Les caractéristiques :



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

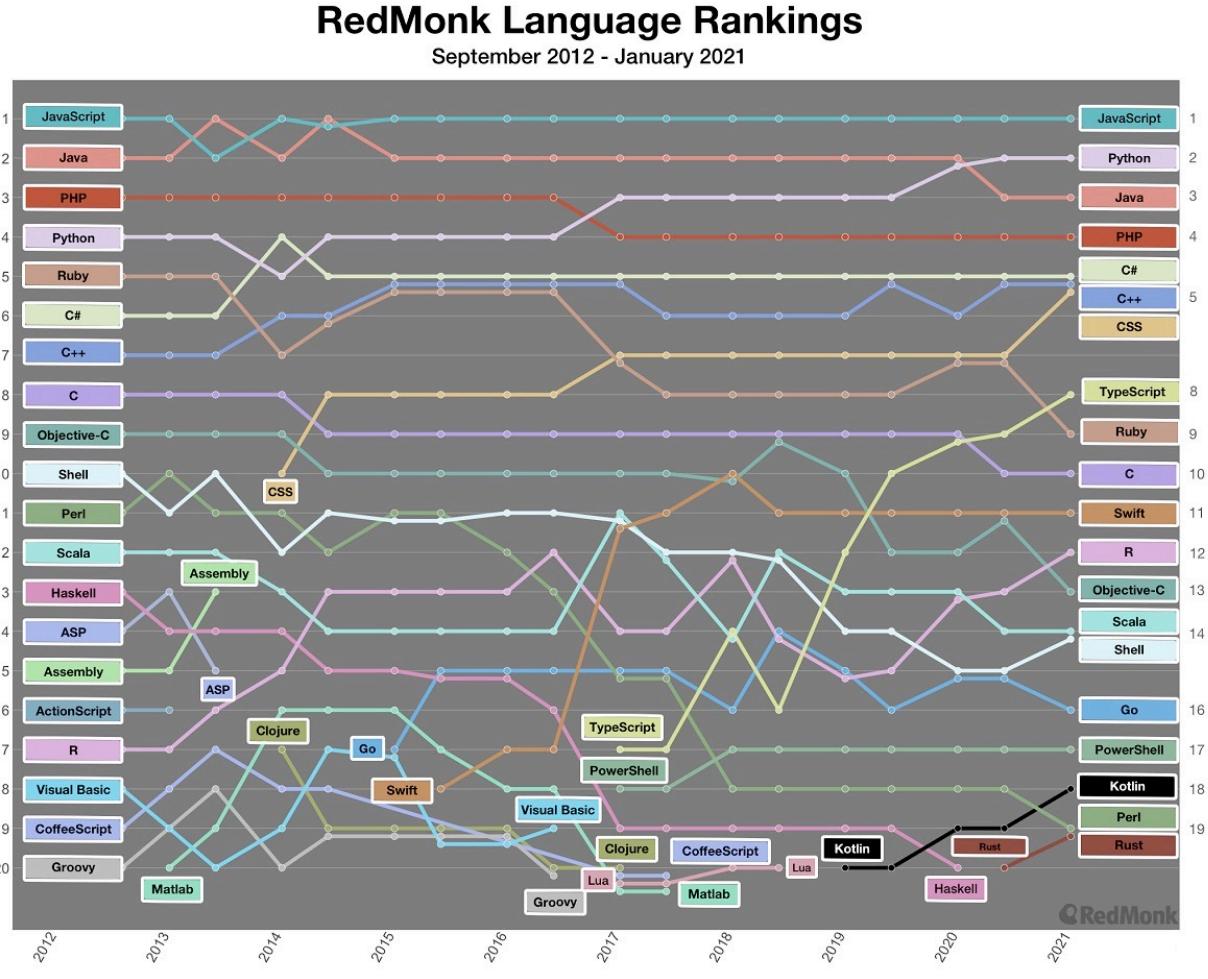
Java est compilé	Le source est compilé en pseudo code ou bytecode puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout). En effet, le bytecode , s'il ne contient 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](#)

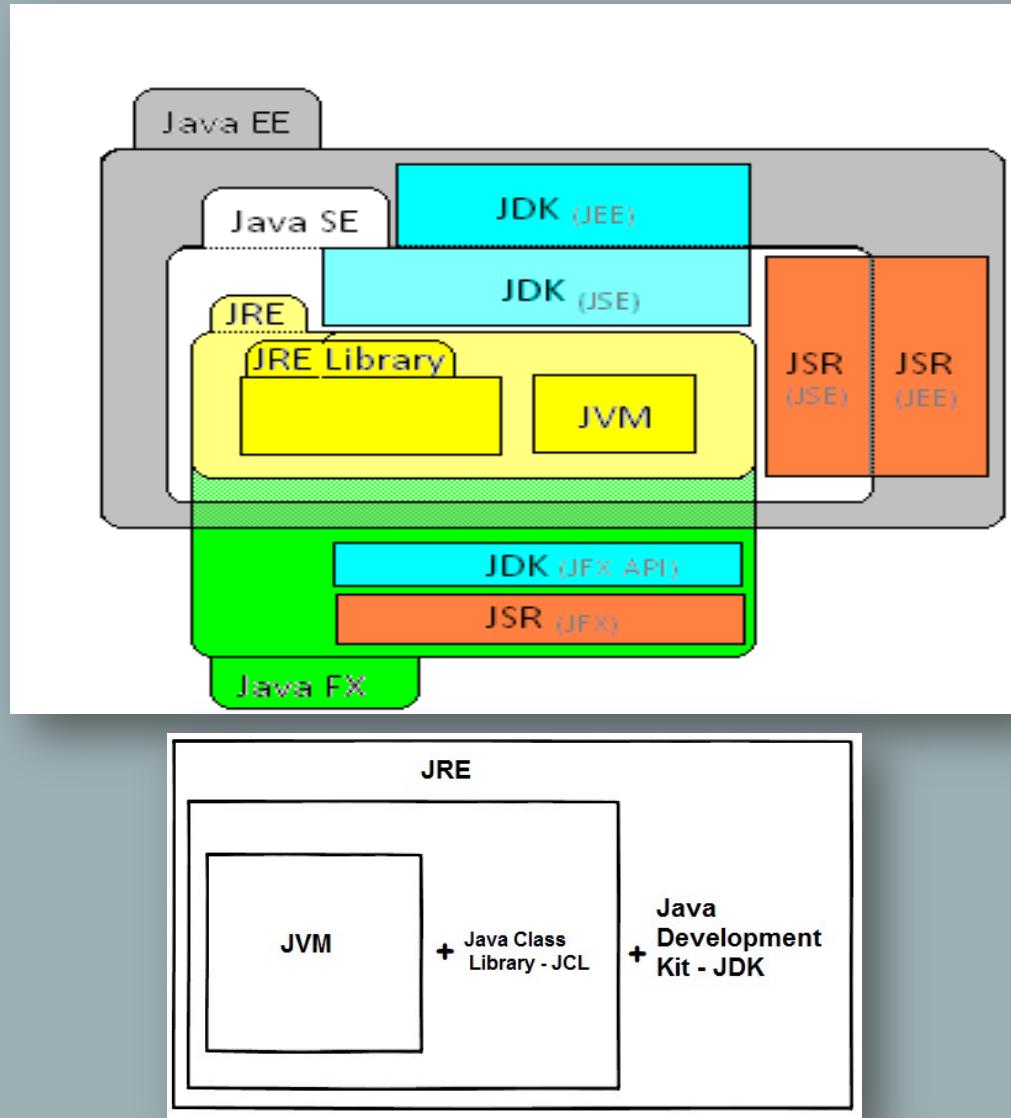
I – 2 - La plateforme java

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.

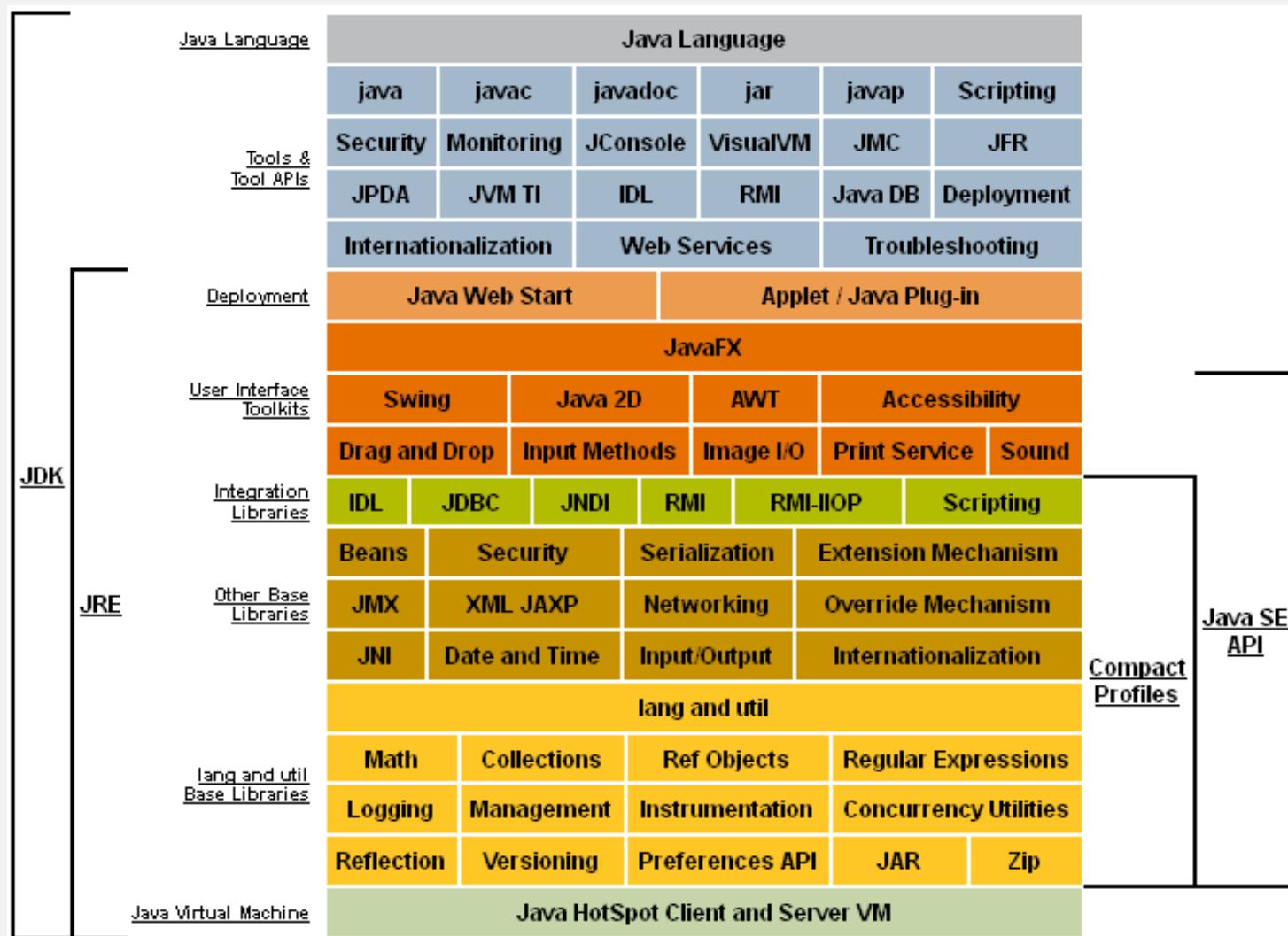
I – 2 - La plateforme java



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.

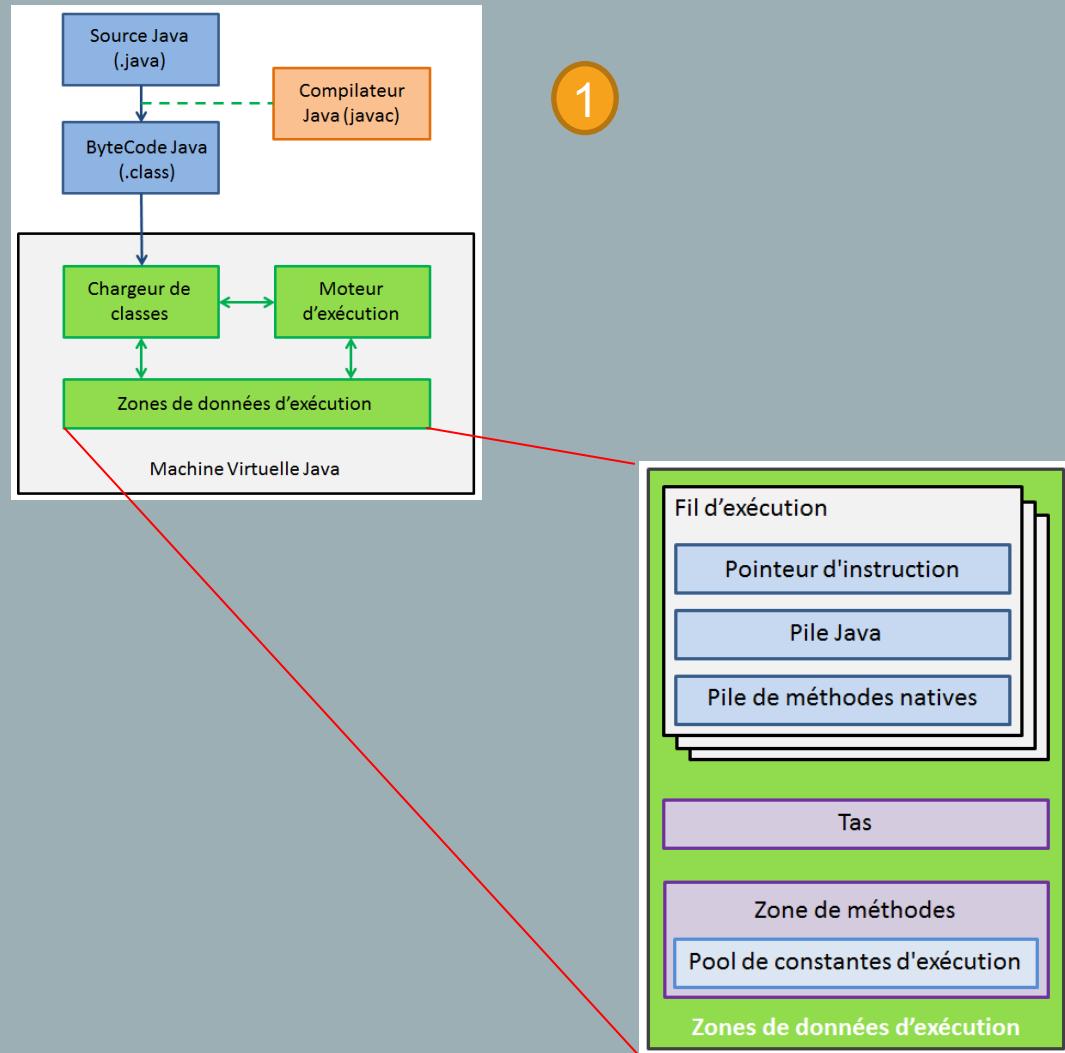
I - 2 La plateforme java



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

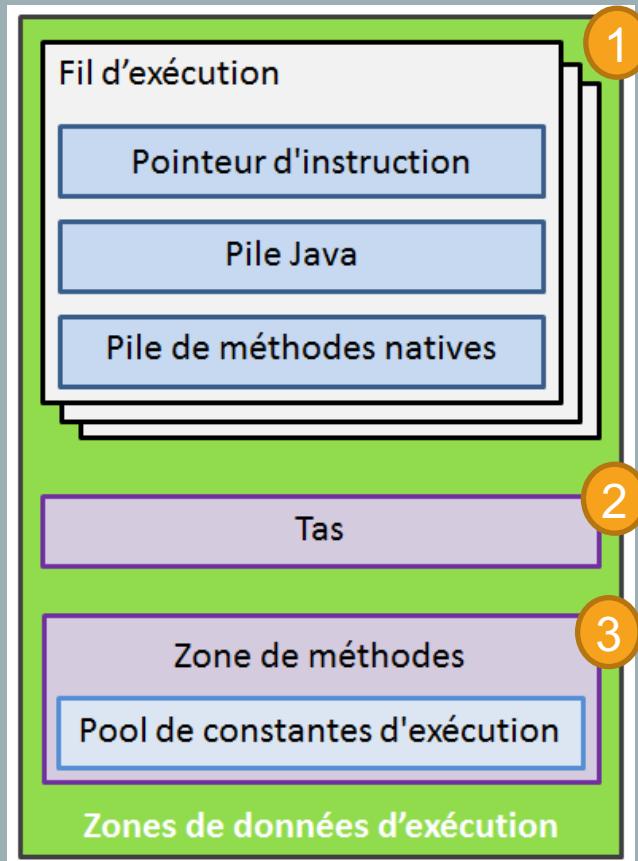


Lors de l'exécution de notre code Java :

- 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

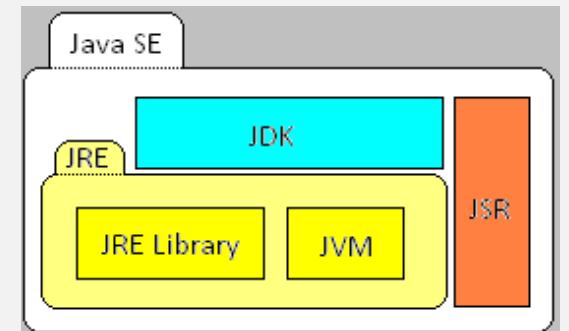


Lors de l'exécution de notre code Java :

- 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)

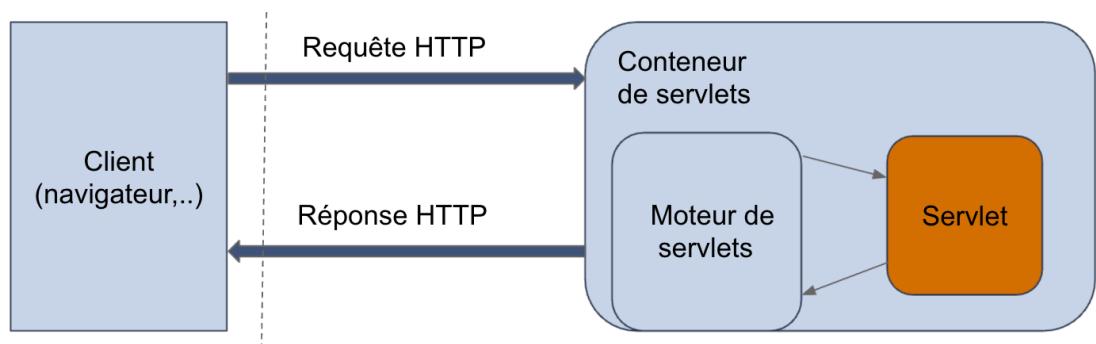
- JEE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées.
- Elle est composée de 3 parties essentielles :
 - Des modèles de composants pour développer les applications, livrés sous forme de bibliothèques de programmation.
 - Une plateforme de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

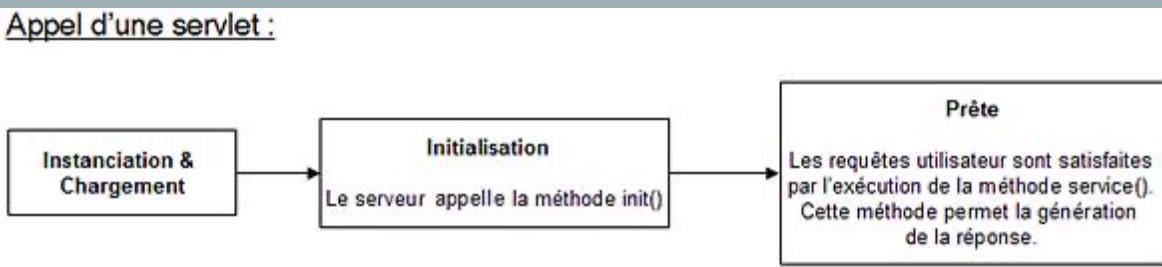


- Une servlet est une classe de l'API JEE/Servlet. Elle est conçue pour être déployée et exécutée dans un conteneur de servlet (type **Apache Tomcat**).
- Les servlets sont des composants logiciels entièrement écrits en Java.
- Son rôle est d'écouter les requêtes HTTP entrantes et d'y répondre. Elle 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.

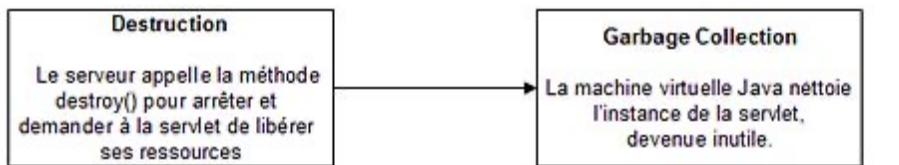
I-5 Les servlets

Cycle de vie d'une servlet

Appel d'une servlet :



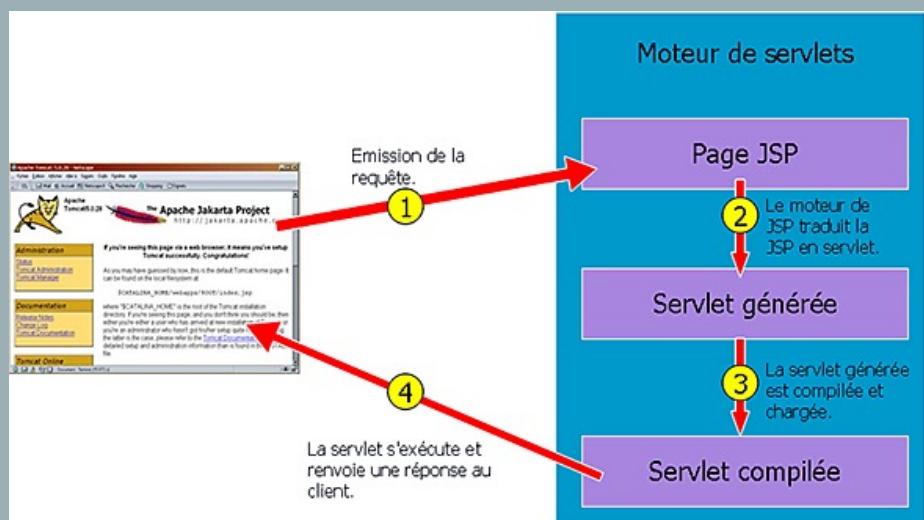
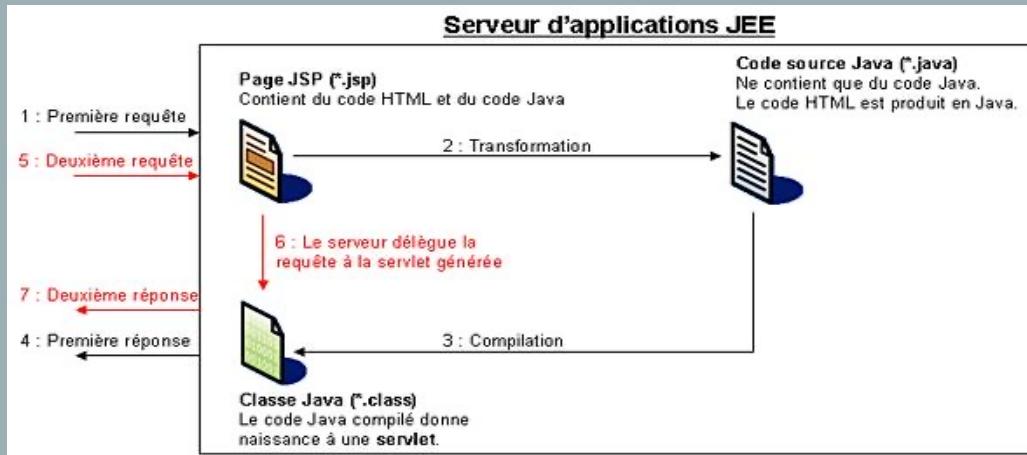
Arrêt du serveur ou de l'application :



- 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

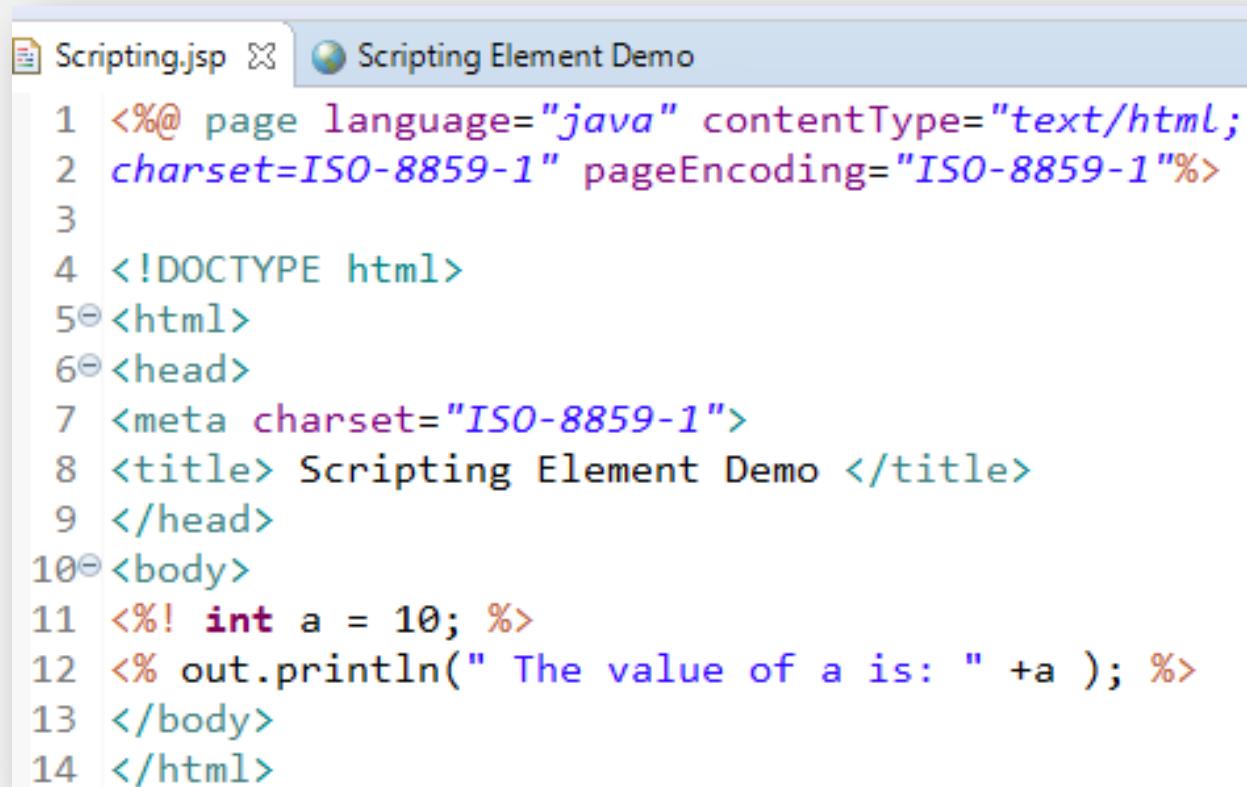
I-5 Java Server Pages : JSP



- 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.

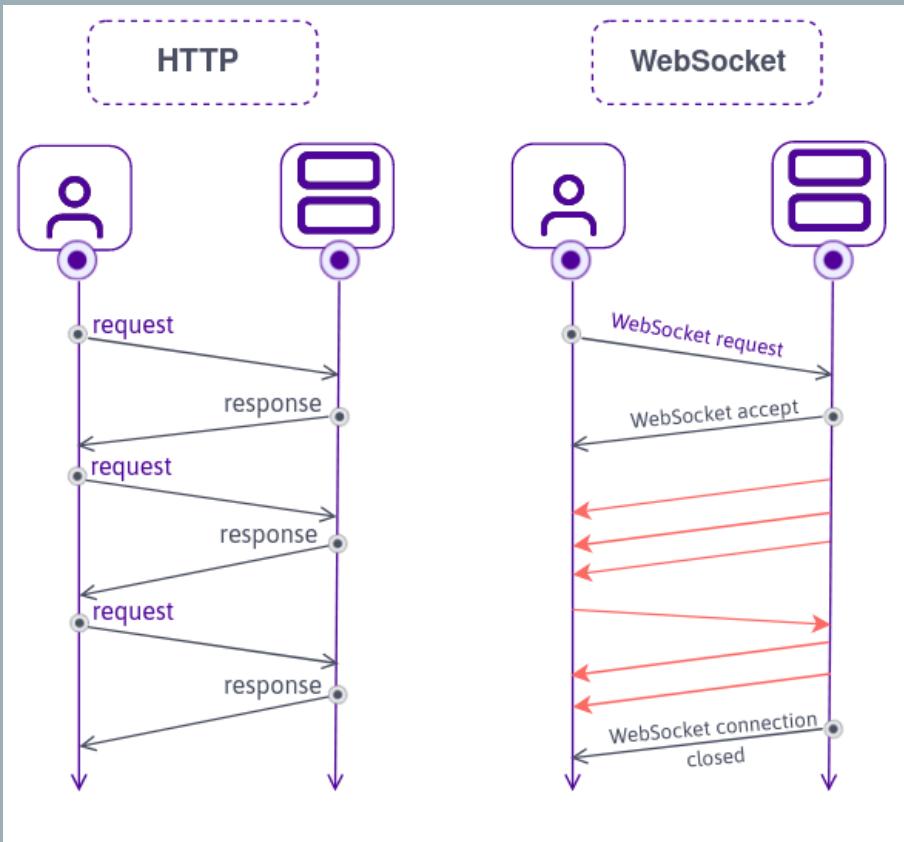
I-5 Entreprise javabeans: ejb

- 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).

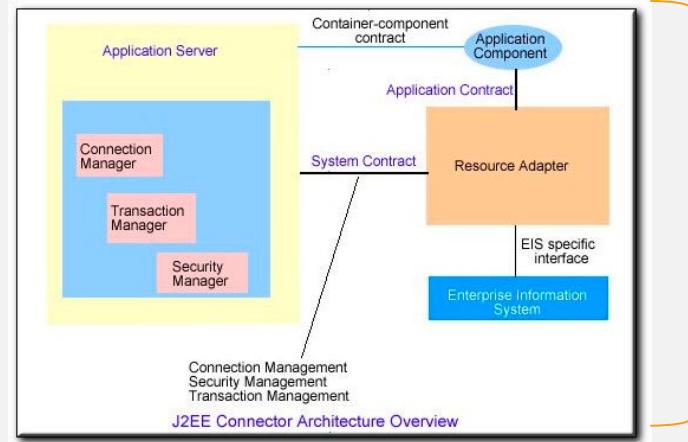
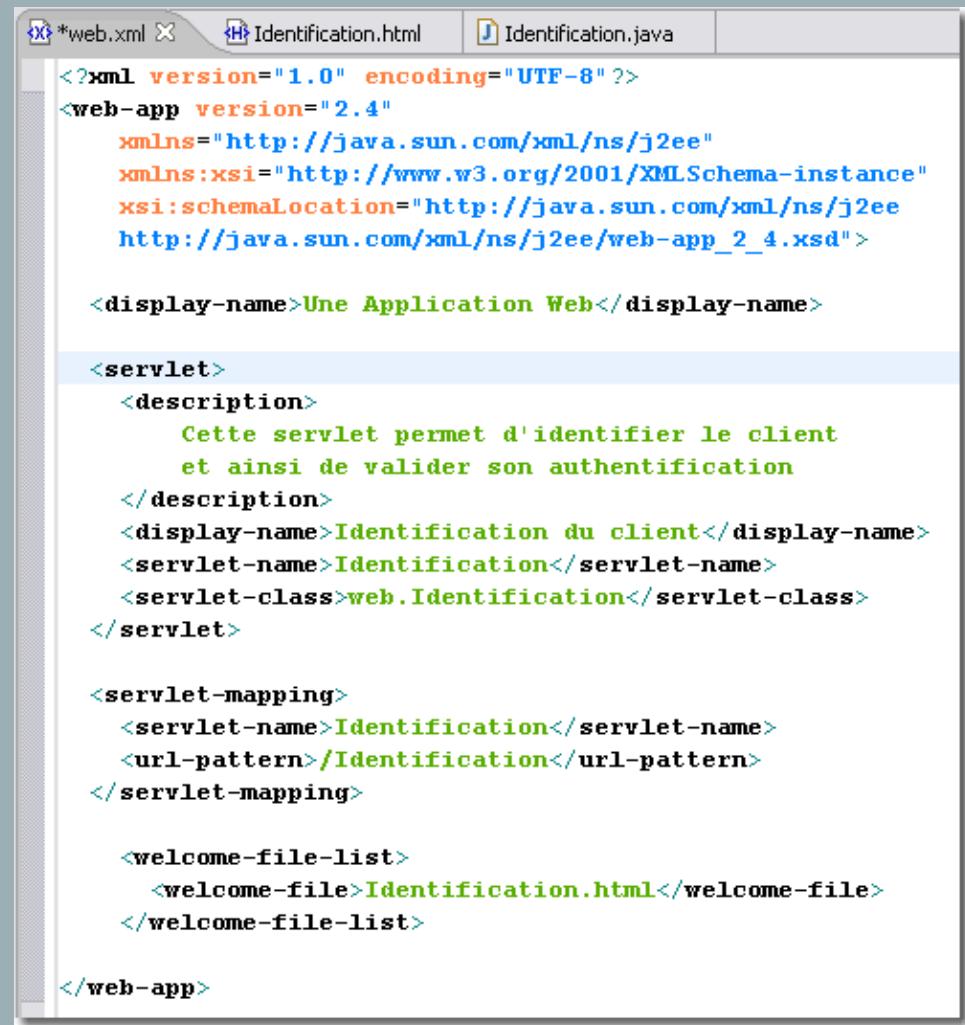


Schéma
JCA

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>
```

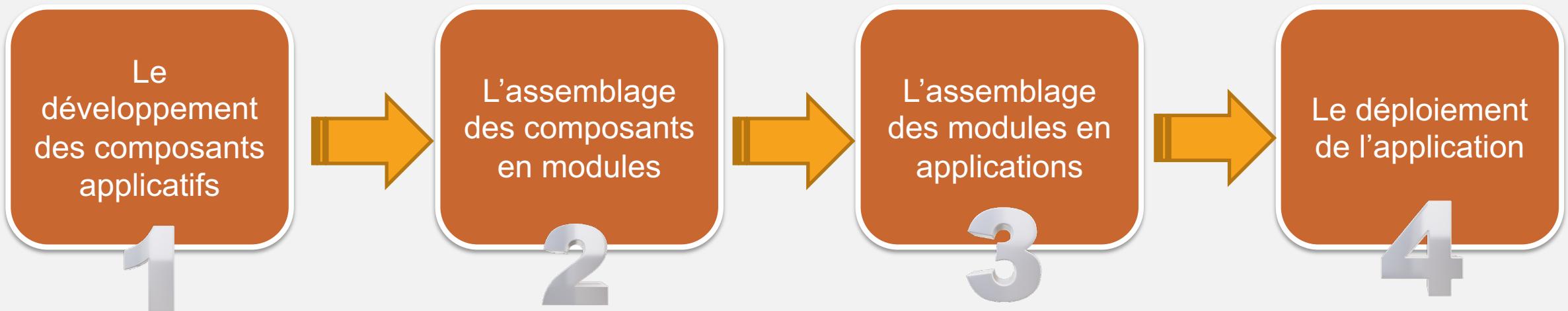
XML :

- **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

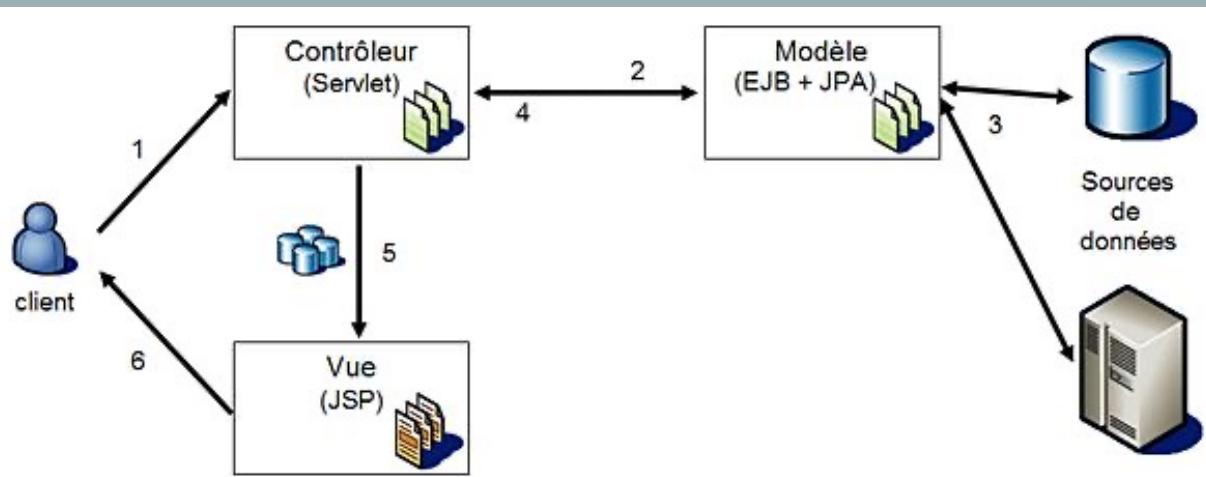
En plus de fournir un modèle de composants, **JEE** standardise également la manière dont ces composants doivent être assemblés avant de pouvoir être installés dans un **serveur JEE**.

Cycle de conception application avec
JEE :

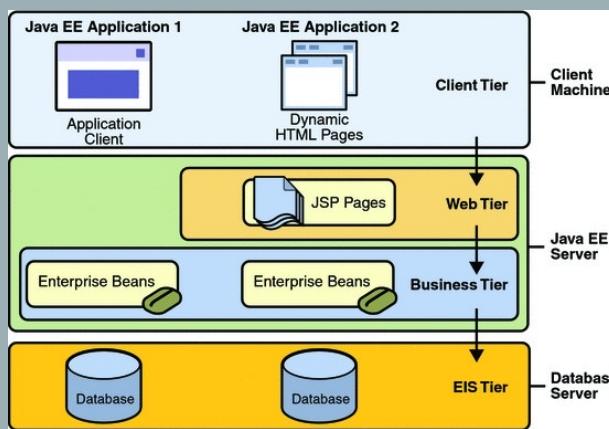
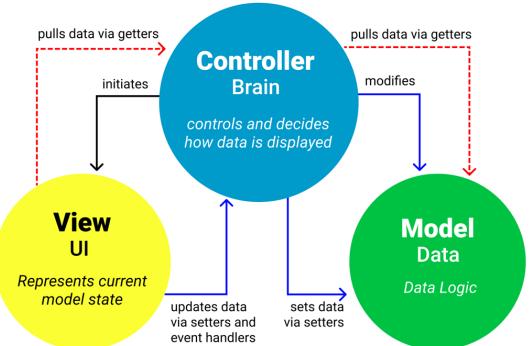


Modèle Vue Contrôleur

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 nécessaire 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).

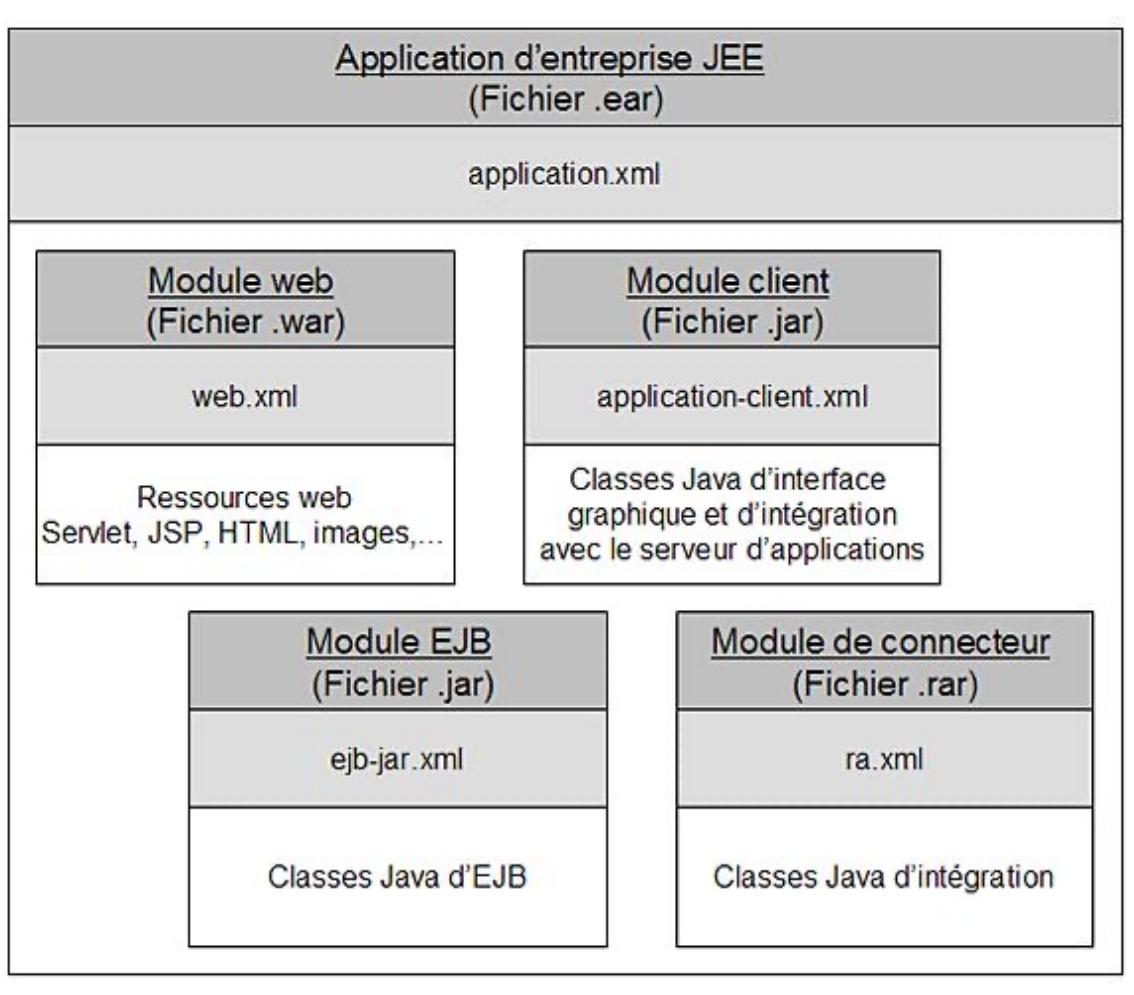
Modules Client :

- 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:



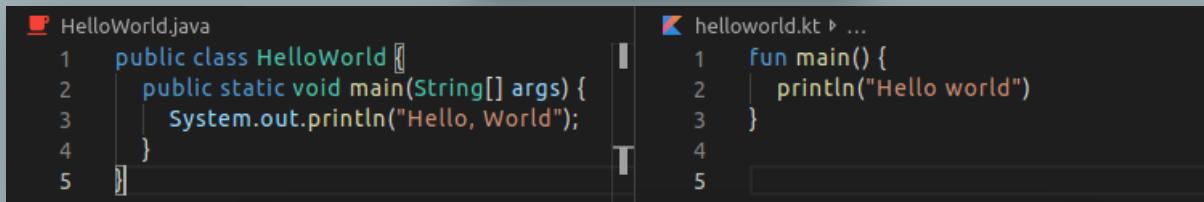
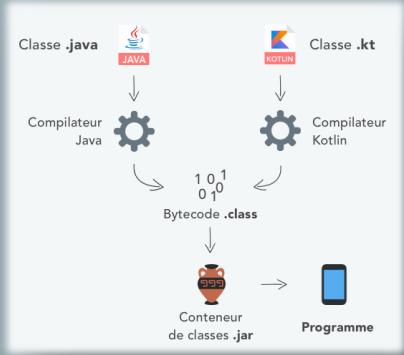
- Lancé en 2003 par James Strachan et *Bob McWhirter*, le projet Open Source Groovy est un langage de script dont les caractéristiques principales sont :
 - Un langage orienté objet pour la **JVM Java** qui s'inspire entre autres de **Python**, **Java**, **Ruby** et **Smalltalk**.
 - Un langage dynamique 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: 

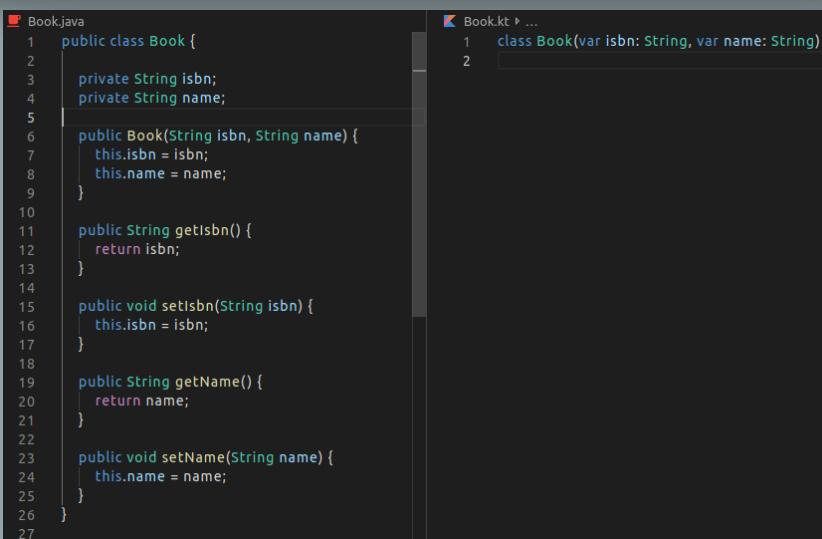
- Kotlin, développé et mis au point pendant plus de 5 ans par l'entreprise JetBrains (l'éditeur de l'IDE IntelliJ) :

- **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.



```
Java code (HelloWorld.java):  
1 public class HelloWorld {  
2     public static void main(String[] args) {  
3         System.out.println("Hello, World");  
4     }  
5 }  
  
Kotlin code (helloworld.kt):  
1 fun main() {  
2     println("Hello world")  
3 }  
4  
5
```

This screenshot shows a comparison between Java and Kotlin code within an IDE. On the left, a Java file named 'HelloWorld.java' contains a simple 'Hello, World' application. On the right, a Kotlin file named 'helloworld.kt' also contains the same application logic. The code is presented in a clean, modern syntax that is characteristic of the Kotlin language.



```
Java code (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 }  
  
Kotlin code (Book.kt):  
1 class Book(var isbn: String, var name: String)
```

This screenshot highlights a specific class, 'Book', shown in both Java and Kotlin. The Java version on the left defines a class with private fields for ISBN and name, and provides methods to get and set these values. The corresponding Kotlin code on the right uses properties (var) to achieve the same functionality in a more concise and readable way. The Kotlin code also includes a constructor that initializes both fields.

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	✗	✓	✓	✓