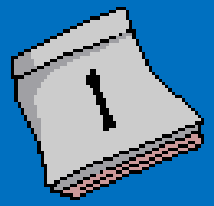
Java EE

Sonda AMMAR
Sonda.ammar@enetcom.usf.tn

Chapitre



Le JEE

Plan

- 1. Introduction
- 2. Qu'est ce que Java EE?
- 3. Architecture Java EE
- 4. Développement côté serveur
- 5. Echanges entre client et serveur
- 6. Le serveur d'applications
- 7. Le modèle MVC
- 8. Les Framework Java EE

Introduction

- Java Enterprise Edition (Java EE) est une norme proposée par Sun visant à définir un standard de développement d'applications d'entreprises multi-niveaux basées sur des composants,
- Aujourd'hui Java EE est devenue la plateforme de choix des grandes entreprises pour développer des applications web, et pour créer des applications distribuées,
- Java EE est aussi très utilisée dans la finance (banque).

Introduction

- Java EE est conçue comme un langage, une plateforme de développement d'application web.
- On peut comparer Java EE à PHP, Django ou a ASP.NET ➔ Java EE est un concurrent de ces langages.

Qu'est ce que Java EE?

- Java EE est basée sur le langage de développement **Java**.
- JEE est basé sur JSE (Java Standard Edition) qui contient les API de base de Java.
- Java permet de développer des applications.
- Ajouter un **ensemble de bibliothèques** qui ajoutent des fonctionnalités à Java.
- Java EE permet de développer des sites web **solides**, **robustes** et bien **structurés**

Java EE	=	Java	+	Ensemble de bibliothèques

Qu'est ce que Java EE?

- JEE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées.
- JEE est composée de deux parties essentielles :
  - Un ensemble de spécifications pour une infrastructure dans laquelle s'exécutent les composants écrits en Java : un tel environnement se nomme serveur d'applications.
  - Un ensemble d'API qui peut être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers

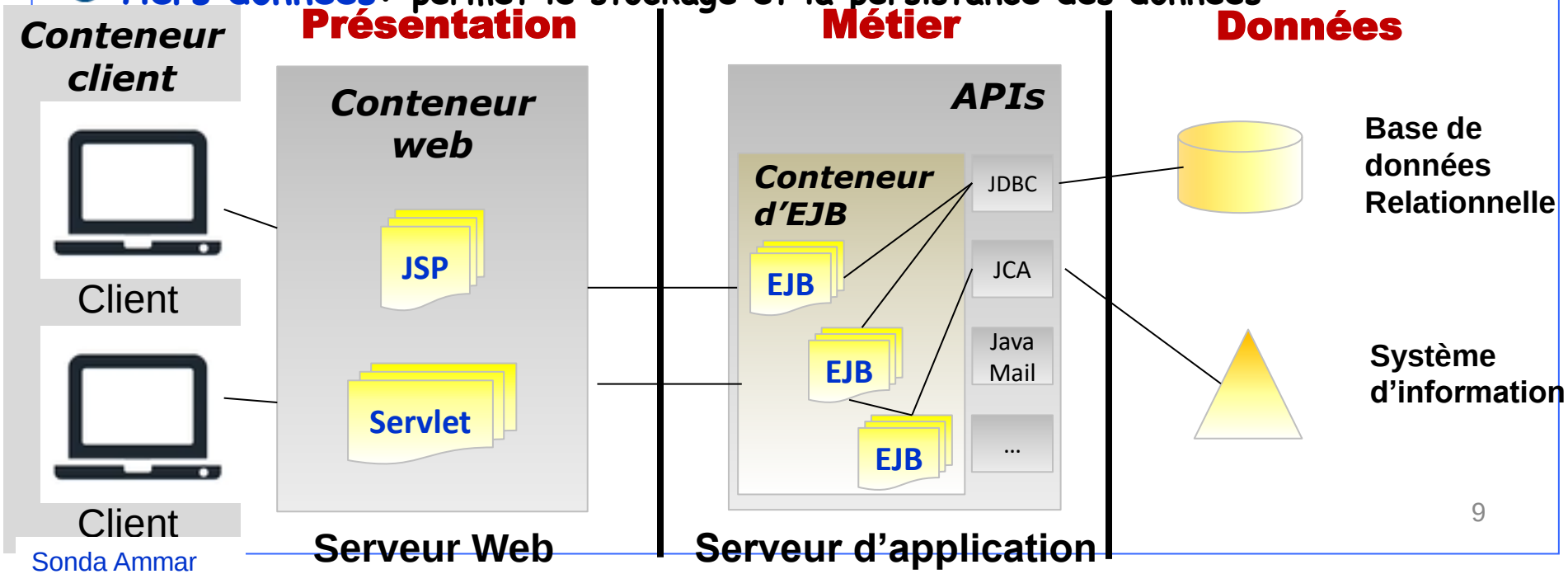
Qu'est ce que Java EE?

Les technologies essentielles

- **Servlets et JSP (JavaServer Pages) :** Pour le développement d'applications web dynamiques en Java.
- **EJB (Enterprise JavaBeans) :** Un composant serveur pour le développement côté serveur dans les applications d'entreprise.
- **JDBC (Java Database Connectivity) :** Une API Java pour interagir avec des bases de données relationnelles.
- **JMS (Java Message Service) :** Un service de messagerie pour permettre la communication entre des composants distribués.
- **JTA (Java Transaction API)** : Une API pour gérer les transactions dans les applications d'entreprise.
- **JPA (Java Persistence API) :** Pour la persistance des données dans les applications Java, facilitant l'interaction avec les bases de données.
- **JSF (JavaServer Faces) :** Un framework pour la création d'interfaces utilisateur basées sur des composants.

Architecture Java EE

- **JavaEE est une architecture 3-tiers :**
  - **Tiers présentation**: permet le dialogue avec l'utilisateur, affichage des données
  - **Tiers métier**: permet l'encapsulation des traitements (dans des EJB ou des JavaBeans), gestion du métier de l'application
  - **Tiers données**: permet le stockage et la persistance des données



Architecture Java EE

- **Pour exécuter des composants de natures différentes :**
  - JEE définit des conteneurs pour chaque composant.
  - JEE définit pour chaque composant des interfaces qui leur permettront de dialoguer avec autres composants lors de leur exécution.
- **Les conteneurs permettent aux applications d'accéder aux ressources et aux services en utilisant les API.**
- **Les appels aux composants se font par des clients en passant par les conteneurs. Les clients n'accèdent pas directement aux composants mais sollicitent le conteneur pour les utiliser.**

Architecture Java EE

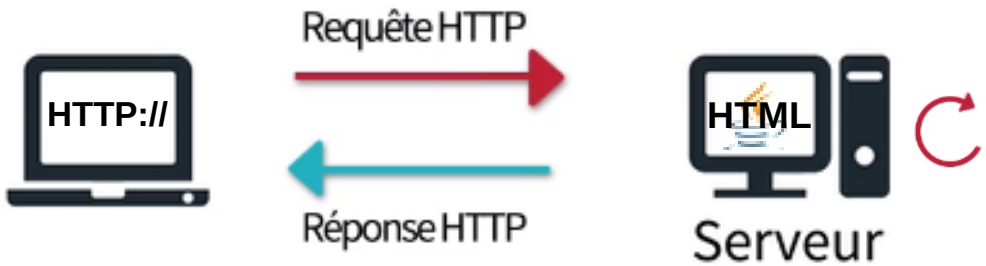
- **JavaEE permet une séparation claire entre :**
  - l'interface homme-machine
  - les traitements métiers
  - les données

Echanges entre client et serveur

- Que ce passe-t-il quand on va visiter un site web qui est codé en Java EE?
- Le visiteur va demander une page web au serveur.
- Le serveur possède le code Java (Java EE), il génère une page HTML qu'il retourne au visiteur

Echanges entre client et serveur

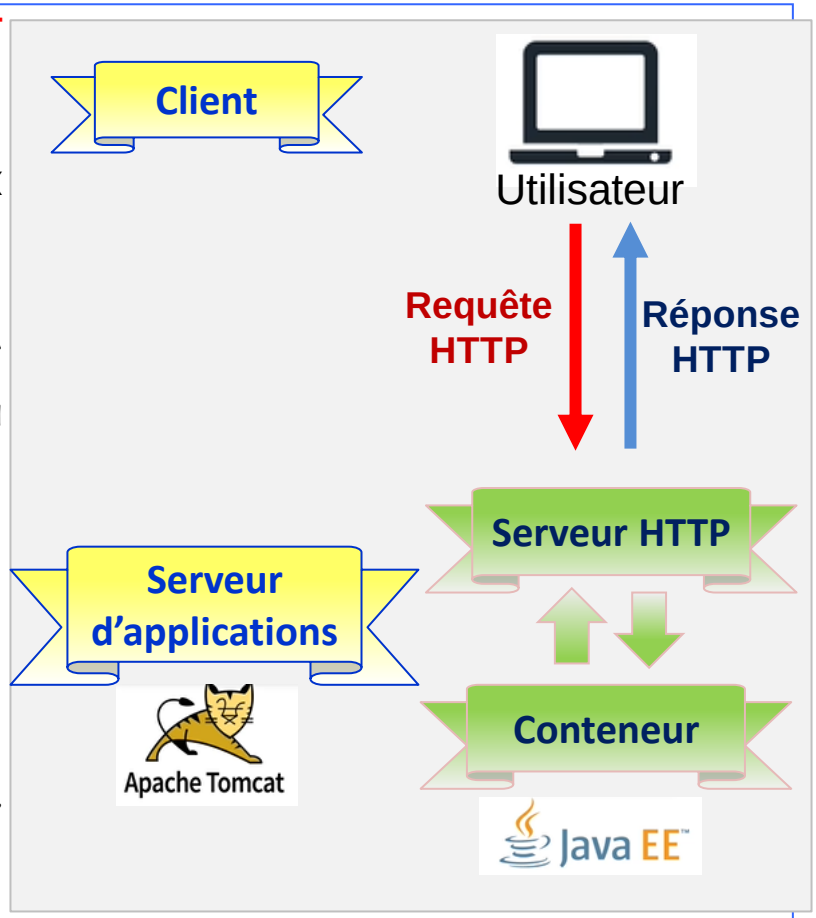
- **Le visiteur va demander une page web au serveur.**
  - Le visiteur saisie l'URL dans son navigateur
  - Le navigateur envoie une requête HTTP au serveur
- **Le serveur possède le code Java (Java EE), il génère une page HTML qu'il retourne au visiteur**
  - Le serveur exécute la requête pour générer une page HTML.
  - Le serveur retourne une réponse HTTP au visiteur



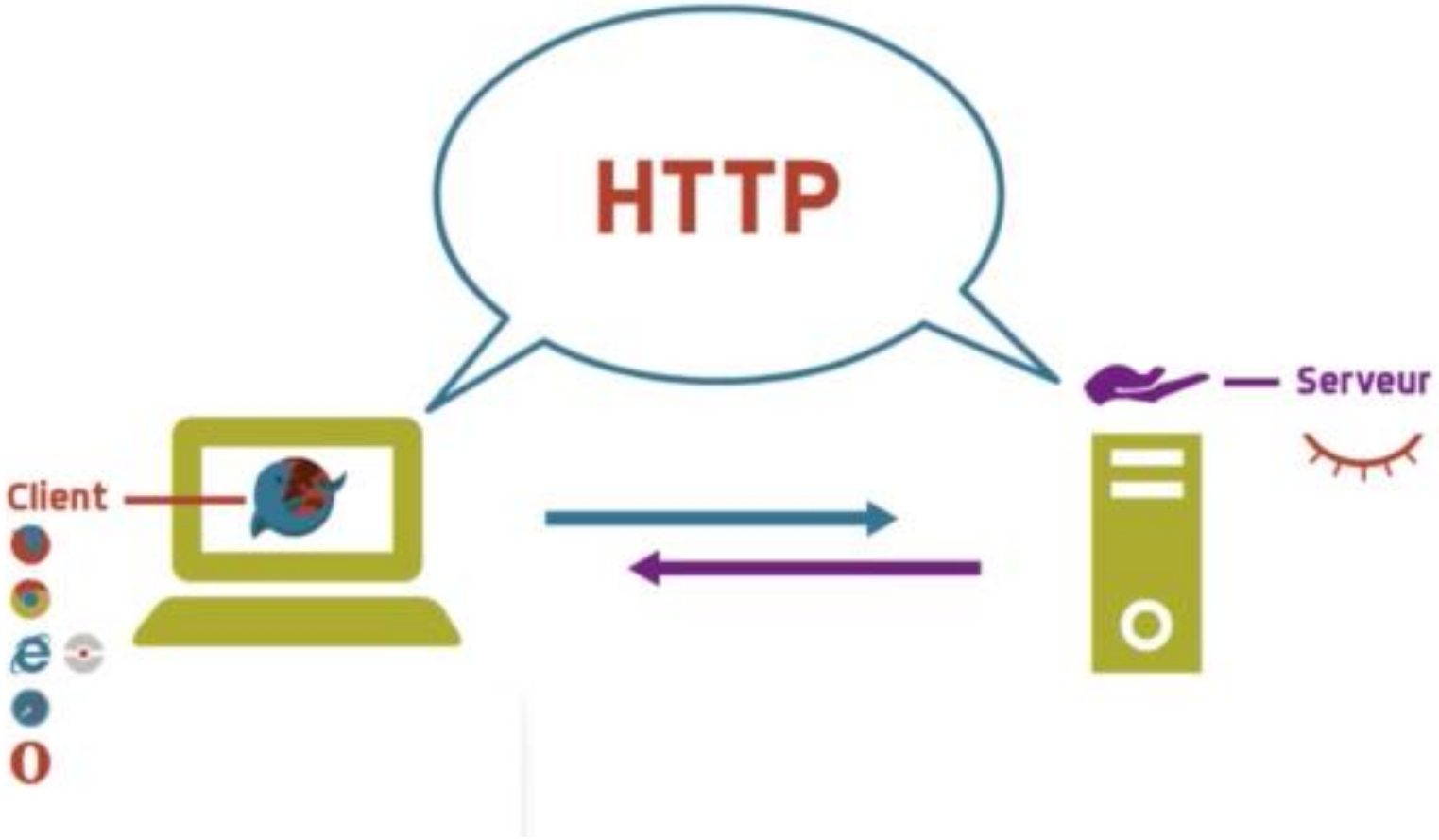
Echanges entre client et serveur

Le serveur d'applications

- Le visiteur envoi une requête HTTP au serveur d'applications.
- Le serveur d'applications est découpé en deux parties:
  - le serveur HTTP reçoit la requête HTTP qui sera lue, découpée, analysée et envoyée ensuite au conteneur
  - Le conteneur permet d'exécuter le code Java EE
- Le serveur d'applications (Apache Tomcat) renvoi la réponse HTTP au visiteur
  - Le serveur exécute la requête pour générer une page HTML.
  - Le serveur retourne une réponse HTTP au visiteur

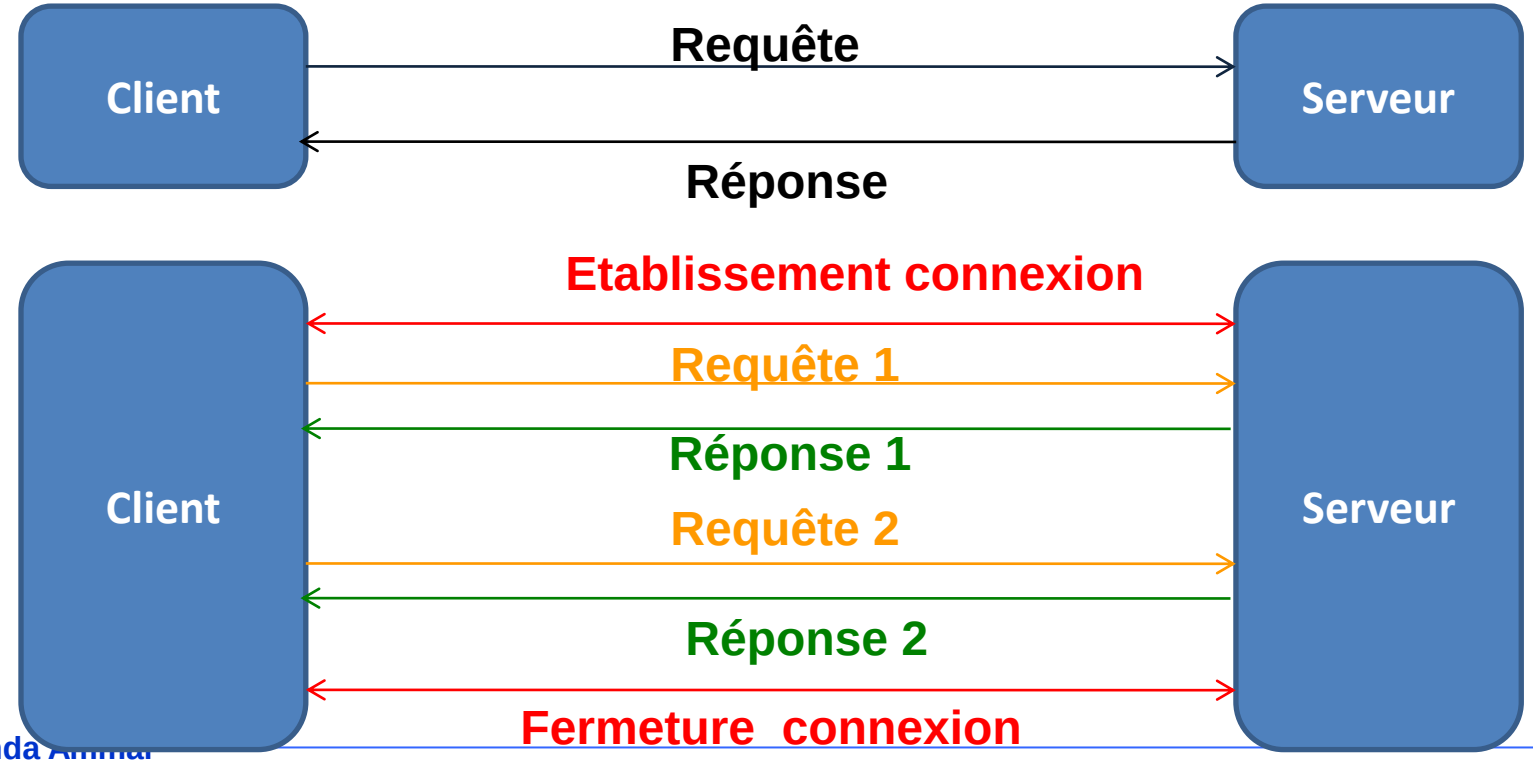


Echanges entre client et serveur

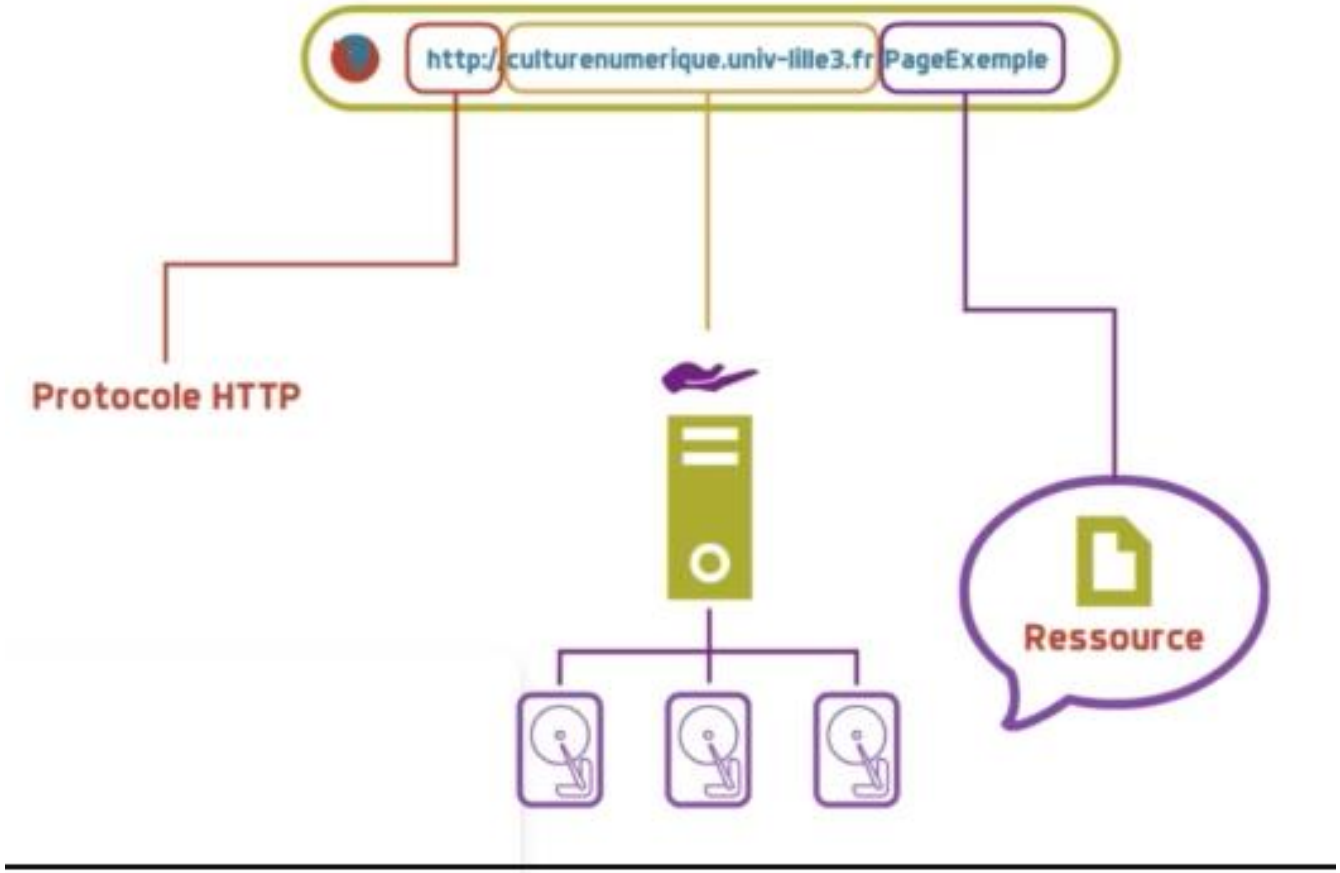


Le protocole HTTP

- Le protocole HTTP (Hyper Text Transfer Protocol) est indispensable dans la réalisation d'application Web.
- Le principe repose sur un couple requête/réponse



Le protocole HTTP



Les URI (Uniform Resource Identifier)

- **URI est une chaîne de caractères identifiant une ressource sur un réseau**
- **URI permet de :**
  - localiser une ressource (URL: Uniform Resource Locator)
    - **Exemple : http://www.google.com/**
  - Nommer une ressource (URN: Uniform Resource Name)
    - **Exemple : urn:isbn:0-395-36341-1**
  - Faire les deux en même temps (URL + URN)
- **Le terme URL est utilisé dans le développement Web pour identifier et localiser une ressource**
- **Structure d'une URL HTTP**

"http://"+hôte+[":"+port] ["/"+ chemin ressource+["? "+paramètres]]

Format d'une requête HTTP

<Méthode> <URI> HTTP / <Version>

[< Champ d'entête > : < Valeur>]

...

*** ligne vide ***

[Corps de la requête]

Méthodes HTTP

- **GET**
  - Demande pour **obtenir des informations**
  - Envoie des informations d'un formulaire
- **HEAD**
  - Demande pour obtenir **des informations concernant l'URI**
- **POST**
  - Envoi de données (contenu du formulaire vers le serveur, …). Ces données sont situées après l'entête et un saut de ligne.
- **PUT**
  - Mise à jour de la ressource à l'URI indiqué.
- **DELETE**
  - Suppression des données désignées par l'URI

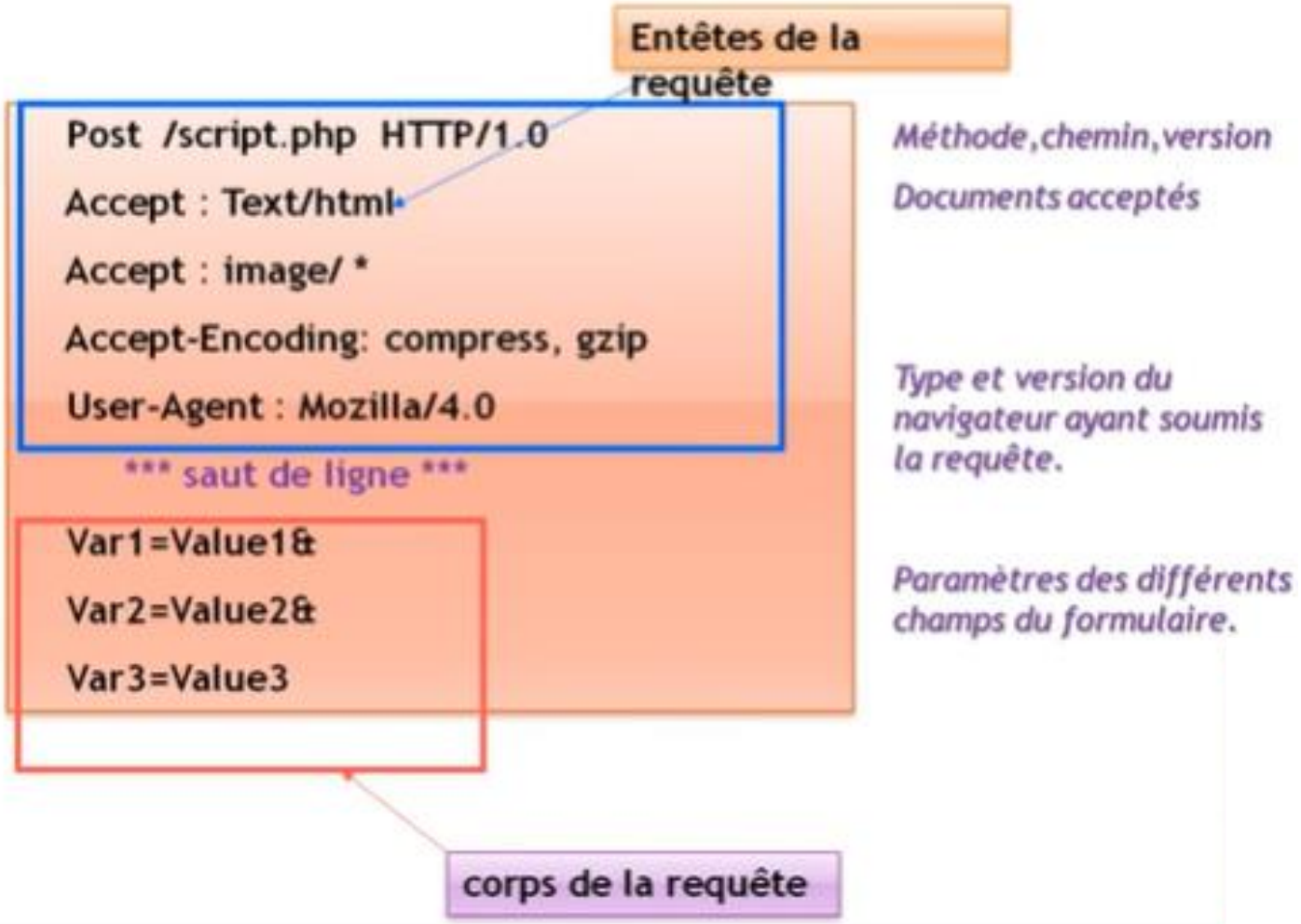
Requête HTTP

- Type de la requête : GET ou POST
- Host : nom du domaine du serveur auquel on envoie la requête, le port s'il est différent de 80
- Connection : comportement de la connexion
- Cache-control: Cette propriété indique au cache du serveur s'il doit recharger la page à chaque reprise ou suivant un délai précis
- Upgrade-Insecure-request: Indique au client que l'on préfèrera toujours une reconnection HTTPS si jamais celui-ci est en train de migrer entre HTTP et HTTPS
- User-agent: Il s'agit de la chaine de caractère permettant d'identifier formellement le navigateur qui a généré la requête; on y trouve sa version, son développeur, …
- Accept: Indique le type du contenu (de média) que le client est en mesure d'accepter.

Requête HTTP

- Referer : Définit l'URL de la page à partir de laquelle la ressource est demandée.
- Accept-encoding: Spécifie le type de compression du contenu que le client (le navigateur) peut traiter. Le contenu est en effet souvent compressé pour optimiser le temps de transfert !
- Cookie: On retrouve ici le cookie (une série de données) qui avait été précédemment enregistré sur le navigateur de l'utilisateur et qui est renvoyé au serveur lors de chaque requête.
- If-Modified-Since: Permet au serveur de renvoyer la réponse "304 Not modified" si jamais la page n'a pas été modifiée depuis la date indiquée

Exemple de requête HTTP



La réponse

Structure

HTTP/<Version> <Status> <Commentaire Status>

Content-Type: <Type MIME du contenu>

[< Champ d'entête >: <Valeur>]

...

*** Ligne vide ****

Document (Contenu html, xml, image, ...)

La réponse HTTP

Structure

- **L'entête de la réponse peut être découpé en 2 parties:**
  - Première ligne: statut de la réponse
    - version du protocole HTTP (HTTP/1.1)
    - Le code du statut (200)
    - L'interprétation textuelle du code de statut (ok)
  - Autres attributs de réponse : permettent au serveur de passer des informations complémentaires.
- **Le corps de la réponse: partie utilisée par le client pour l'affichage, il peut prendre plusieurs types.**

Exemple de réponse HTTP

Entêtes de la réponse

HTTP/1.0 200 OK — *Ligne de Status*

Date : Wed, 05Feb12 15:02:01 GMT

Server : Apache/1.3.24 — *Nom du Serveur*

Mime-Version 1.0

Last-Modified : Wed 02Oct11 24:05:01GMT — *Dernière modification*

Content-Type : Text/html — *Type de contenu*

Content-legnth : 4205 — *Sa taille*

*** saut de ligne ***

<HTML><HEAD>

....

</BODY></HTML>

Le fichier que le client va afficher

corps de la réponse

La réponse HTTP

Google

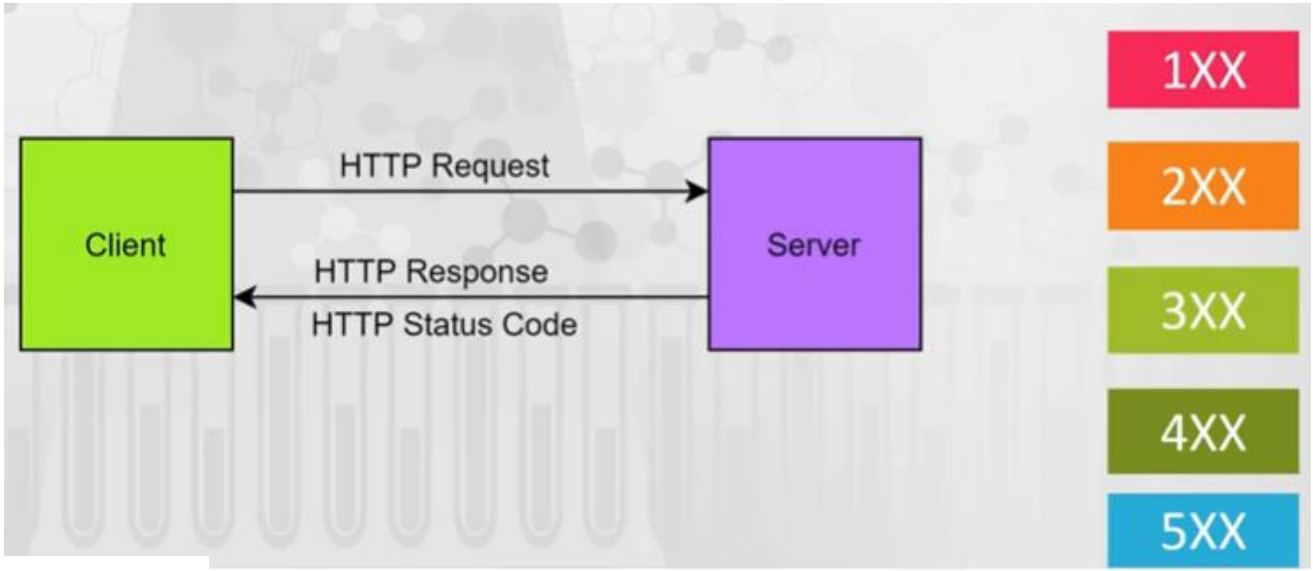
404. That's an error.

The requested URL /does_not_exist was not found on this server. That's all we know.

La réponse HTTP

Les codes du statut

- Chaque réponse est associée à un code de statut sur trois digits. Il permet d'indiquer au client l'état de la réponse et si une requête HTTP a été exécutée avec succès ou non . Le client peut ainsi réagir en conséquence.
- Ces codes sont classés en 5 catégories.



La réponse HTTP

Les codes du statut

- **Les 5 catégories:**
  - Les réponses informatives: codes d'informations indiquant que la requête initiée par le navigateur se poursuit
  - Les réponses de succès: codes de réussite renvoyés lorsque la demande du navigateur été reçue, comprise et traitée par le serveur
  - Les messages de redirection: codes de redirection renvoyés lorsqu'une nouvelle ressource a été substituée à la ressource demandée,
  - Les erreurs du client: codes d'erreur client indiquant qu'il y a eu un problème avec la demande.
  - Les erreurs du serveur: codes d'erreur du serveur indiquant que la demande a été acceptée, mais qu'une erreur sur le serveur a empêché l'exécution de la demande.

La réponse HTTP

Les codes du statut

Code	Libelle	Description
100	Continue	Continue
101	Switching Protocols	Changement de protocole
200	OK	OK
201	Created	Créé
202	Accepted	Acceptée
204	No Content	Req traitée avec succès mais aucune info en retour
206	Partial Content	Contenu partiel
300	Multiple Choices	La requête a plusieurs réponses possibles. L'utilisateur doit choisir l'une d'entre elles
301	Moved Permanently	La ressource a changé d’adresse de façon permanente
302	Found	La ressource a changé d’adresse temporairement
304	Not modified	La ressource n’a as changé. Le client peut utiliser la version en cache

La réponse HTTP

Les codes du statut

Code	Libelle	Description
400	Bad Request	Requête incorrecte
401	Unauthorized	Le client doit s'authentifier pour accéder à la ressource
403	Forbidden	Le client n'a pas le droit d'accéder à la ressource
404	Not Found	Ressource non trouvée
405	Method Not Allowed	Type de requête non autorisé
500	Internal Server Error	Le serveur a rencontré une erreur inattendue
501	Not Implemented	Le serveur ne sait pas traiter le type de requête pour la ressource visée
503	Service Unavailable	Le serveur n'est pas capable de répondre à cause d'une surcharge temporaire ou d'une maintenance

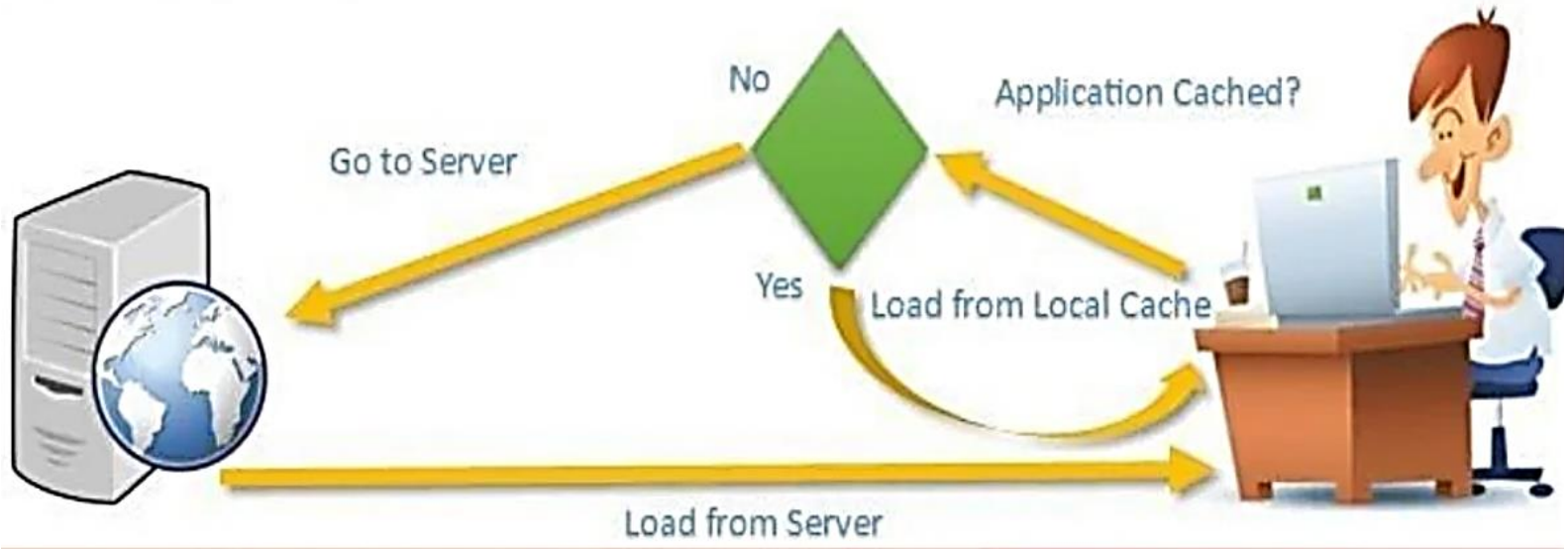
La réponse HTTP

Les attributs de réponse

- **Les attributs génériques : même attributs que celles de la requête**
- **Les attributs de réponse: entête d'une réponse. Ils ne sont pas tous obligatoires**.

Attribut	Description
Content-Encoding	Définit la méthode de compression du corps de la réponse
Content-Language	Définit la langue utilisée dans le corps de la réponse
Location	Définit l'URL de la ressource lors d'une redirection
Server	Définit les caractéristiques du serveur ayant généré la réponse
Set-Cookie	Permet au serveur d'envoyer des cookies au client.

C'est quoi un cache ?



Le cache

Constats

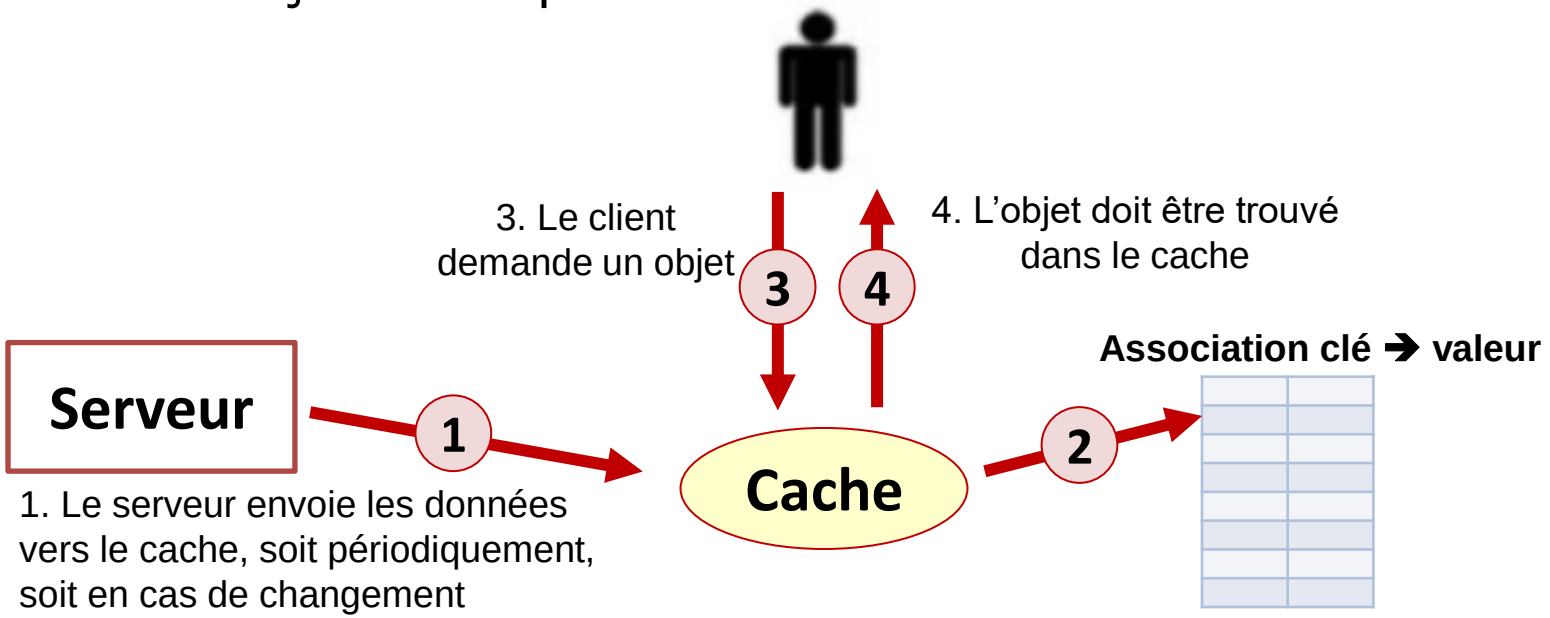
- Le point de vue classique suppose que la mémoire est rare et chère
- La mémoire ne coute pas cher, c'est la vision moderne
- Le cache est devenu un élément important dans les applications d'aujourd'hui
- L'accès aux données est simplifié par les caches
- Accéder plus rapidement à la copie que d'accéder plus lentement à la ressource
- On parle de durée de vie des objets
- Éviter la répétition des appels coûteux
- Une stratégie de cache doit être bien définie

Le cache

Cache passif (Push)

- La source (le serveur) envoie une copie des données vers le mécanisme du cache.
  - Le cache passif (mode push) implique un mécanisme d'abonnement à l'évènement du changement de la source auprès de cette dernière.

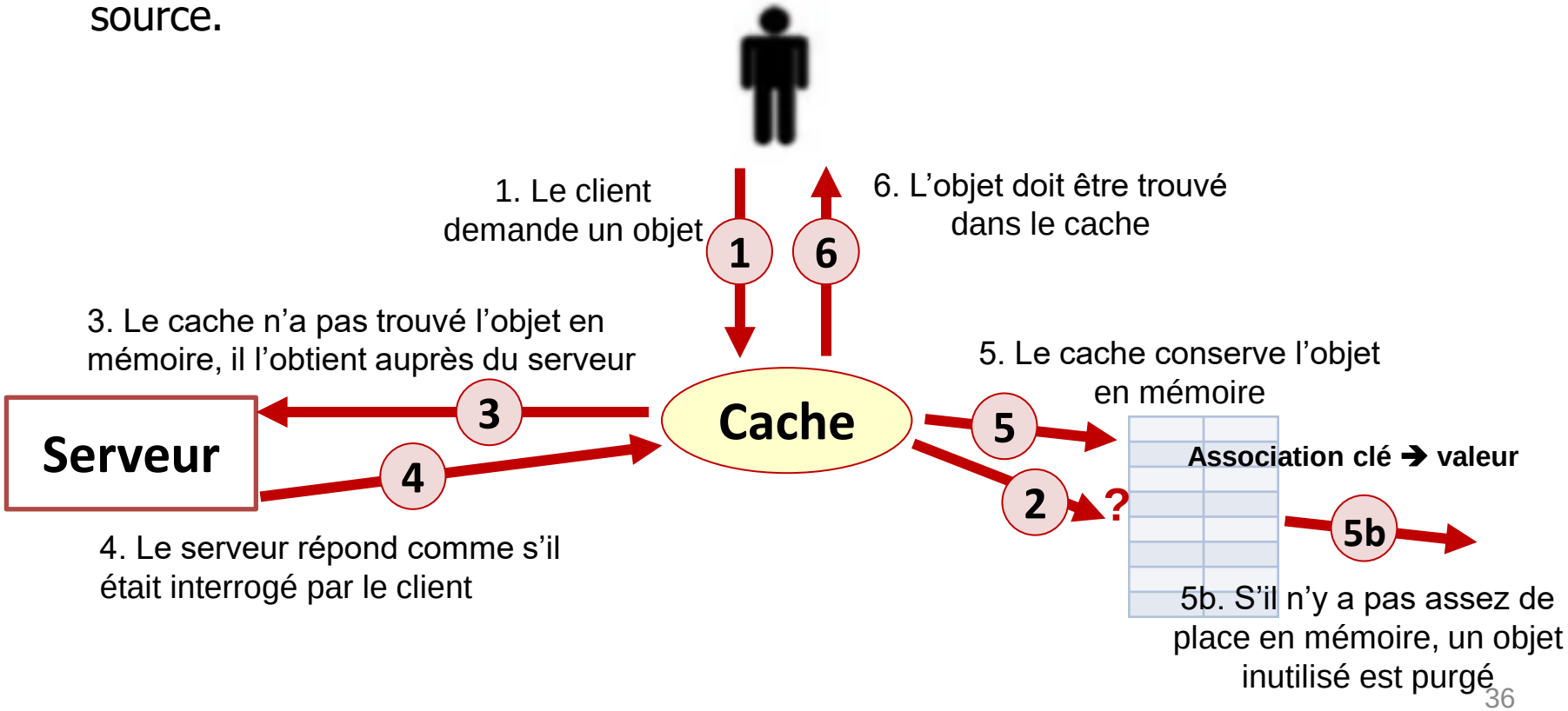
➔ Le cache reçoit une copie de la totalité de l'ensemble du contenu



Le cache

Cache actif (Pull)

- Le mécanisme du cache retire une copie des données de la source.
  - Le dispositif du cache cherche une copie des données originales auprès de la source.



Le cache

Cache en JAVA

Entrée

Clé

Id	Name
21	Ali
31	Rim
41	Mohamed
:	:
101	Rouaa

Valeur

- Un simple cache est représenté par un objet de type HashMap (collection de type Map) géré via un mécanisme de clé valeur pour accéder plus rapidement aux données.

La gestion du cache

- Elément essentiel dans le développement Web
- Améliorer les performances ou palier à des défaillances en limitant le transfert de données qui n'ont pas été modifiées.
- Le protocole HTTP met à disposition du client et du serveur un ensemble d'attributs pour dialoguer et pour savoir si une ressource en cache est toujours d'actualité ou obsolète.
- Exemple: Un client faisant une requête de type GET
- Si la requête correspond à la première demande alors la ressource est retournée avec en en-tête de réponse des attributs indiquant les possibilités de mise en cache.

La gestion du cache

Attributs	Description
Cache-Control	Plusieurs valeurs peuvent être attribuées: **private** : la réponse ne peut pas être mise en cache par un intermédiaire partagé par plusieurs clients **public :** la réponse peut être mise en cache par n'importe quel intermédiaire **max-age :** indiquer le temps en secondes durant lequel la ressource en cache est d'actualité
Pragma	Indiquer que la ressource ne doit pas être en cache
Etag	Associer un id à la ressource qui change à chaque fois où la ressource change.
Expires	Définir une date de fin de validité de la ressource mise en cache
Last-Modified	Définir la date de dernière modification de la ressource

La gestion du cache

- **Lors des requêtes successives vers la même ressource, le mécanisme de cache peut entrer en action**
  - S'il n'y avait pas de cache possible → la requête est envoyée comme si c'était le premier appel
  - Dans le cas où les attributs de réponse indiquaient une possible mise en cache :
    - Le client peut utiliser l'attribut Expires ou la directive max-age (Cache-Control) de la ressource mise en cache s'ils sont renseignés. Si la ressource est toujours d'actualité → aucune requête n'est effectuée.
    - Le client peut utiliser la valeur des attributs Etag et Last-Modified pour interroger le serveur pour voir s'il a une ressource plus récente.

Gestion de l'authentification

- Lorsque une ressource est protégée et nécessite une authentification auprès du serveur pour pouvoir y accéder, un couple d'attributs est utilisé.
- L'attribut WWW.Authenticate demandant au client de s'authentifier (Login/Pwd)
- Login:Pwd envoyés via Authorization au travers d'une nouvelle requête visant la même ressource.
- Si la requête doit passer par un proxy pour accéder à la ressource → le proxy demande au client de s'authentifier (selon sa configuration) → Proxy-Authenticate et Proxy-Authorization

Gestion de l'authentification

Java EE

Remarque

- Lorsque vous transmettez des données sensibles sur vos sites favoris et que ces derniers n'utilisent pas HTTPS (la version sécurisée du protocole HTTP) cela représente un réel danger, car toute requête peut être interceptée ! et dans une requête POST de transmission de formulaire vos données sont lisibles!

Les types de médias

- MIME (Multipurpose Internet Mail Extensions) : définir le type de contenu dans le corps de la requête et de la réponse.
- Attribut Content-Type
- Les navigateurs, dans leurs version actuelles, sont capables d'interpréter le contenu d'une réponse sans passer par Content-Type.
- Accept permet au client d'indiquer au serveur quel type de contenu il accepte pour la réponse.
- Un type de média est composé d'un type et un sous-type

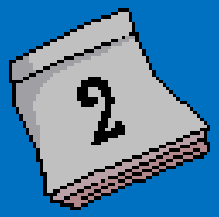
  {type} / {sous-type}
- 6 types : text, image, video, application, audio et multipart

Les types de médias

Types de médias	Description
text/plain	Contenu textuel brut
text/html	Contenu HTML
text/xml	Contenu XML
text/csv	Contenu CSV
text/css	Contenu CSS
image/jpeg	Image au format JPEG
image/png	Image au format PNG
image/svg+xml	Contenu SVG permettant de faire du dessin vectoriel
video/mpeg	Vidéo au format MPEG
video/mp4	Vidéo au format MP4

Types de médias	Description
application/javascript	Contenu JavaScript
application/octet-stream	Contenu binaire non typé
application/pdf	Contenu au format PDF
application/json	Contenu au format JSON (JavaScript Object Notation)
application/xml	Contenu au format XML

Chapitre



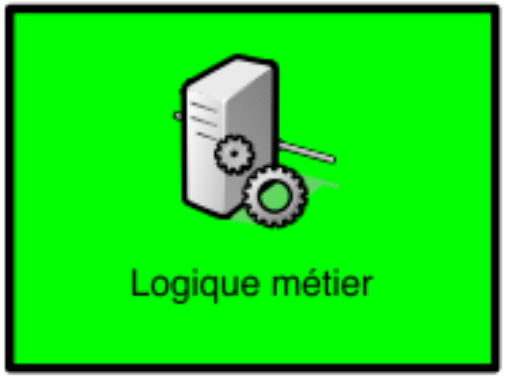
Le modèle MVC

Rappel

- Communication client-serveur ?
- Côté client : navigateur
- Côté serveur : serveur HTTP
- L'écoute des informations arrivées sur le port utilisé par le protocole HTTP, le port 80 et examiner chaque requête entrante.

Architecture JEE

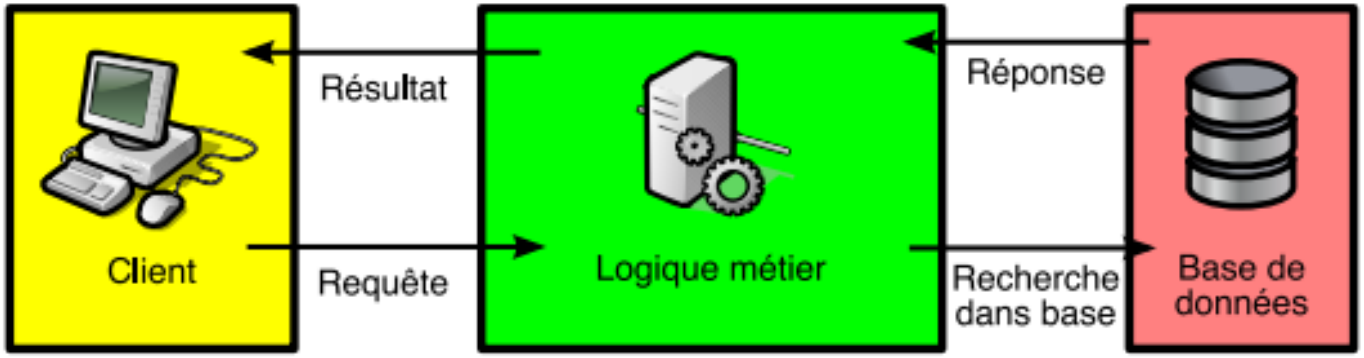
- **JavaEE est une architecture 3-tiers :**
  - **Tiers présentation**: permet le dialogue avec l'utilisateur, affichage des données
  - **Tiers métier**: permet l'encapsulation des traitements (dans des EJB ou des JavaBeans), gestion du métier de l'application
  - **Tiers donnée**: permet le stockage et la persistance des données



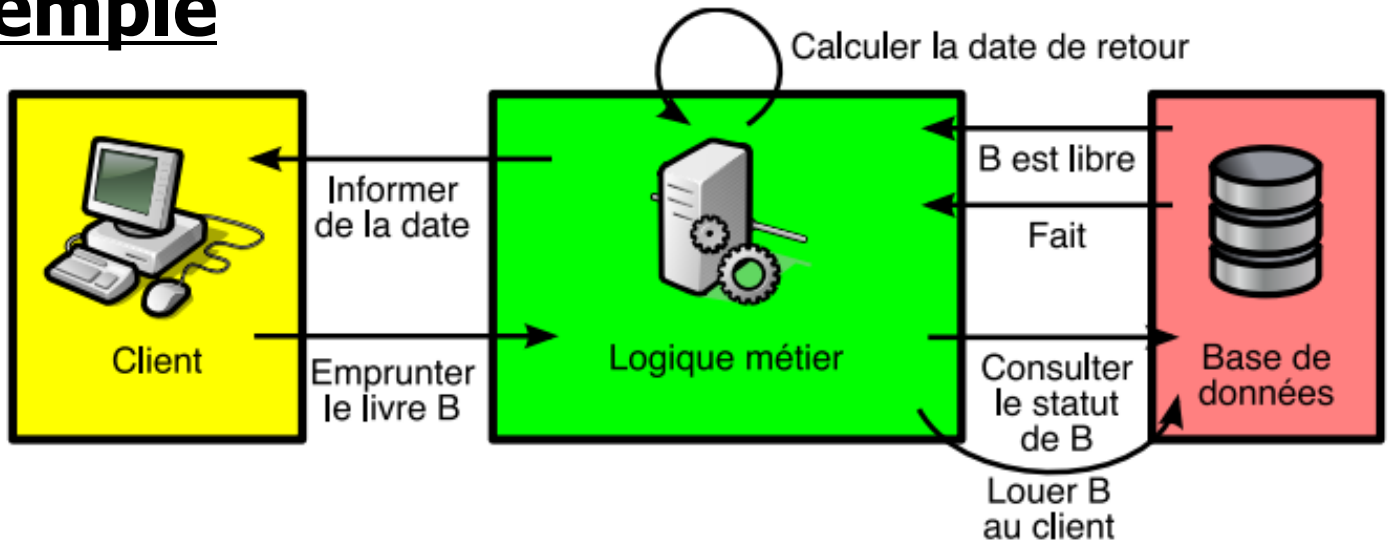
- **Le serveur d'applications fournit des services spécialisés et récurrents aux applications**

Architecture JEE

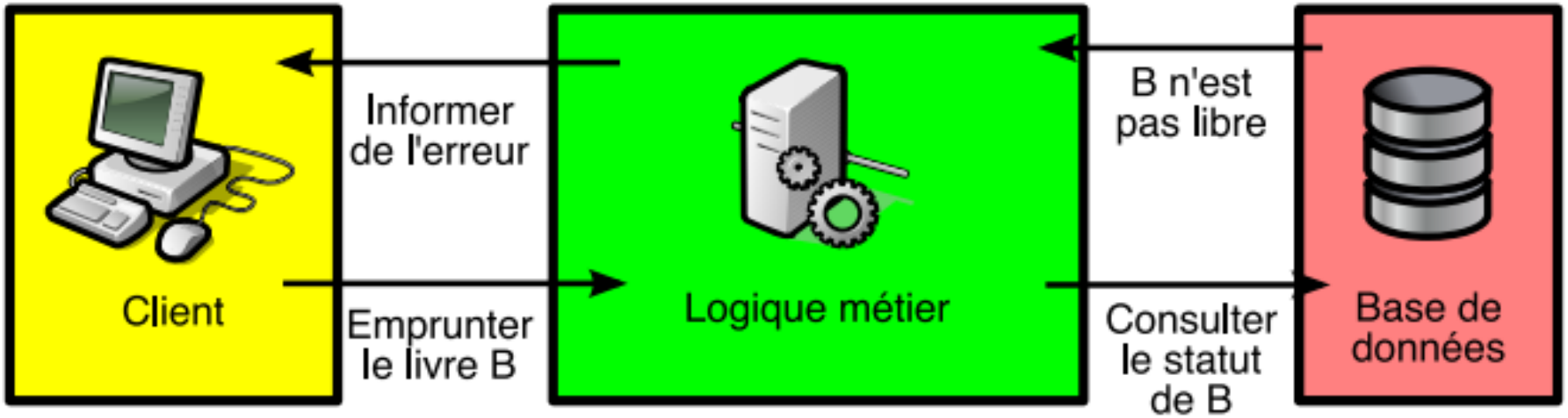
Processus



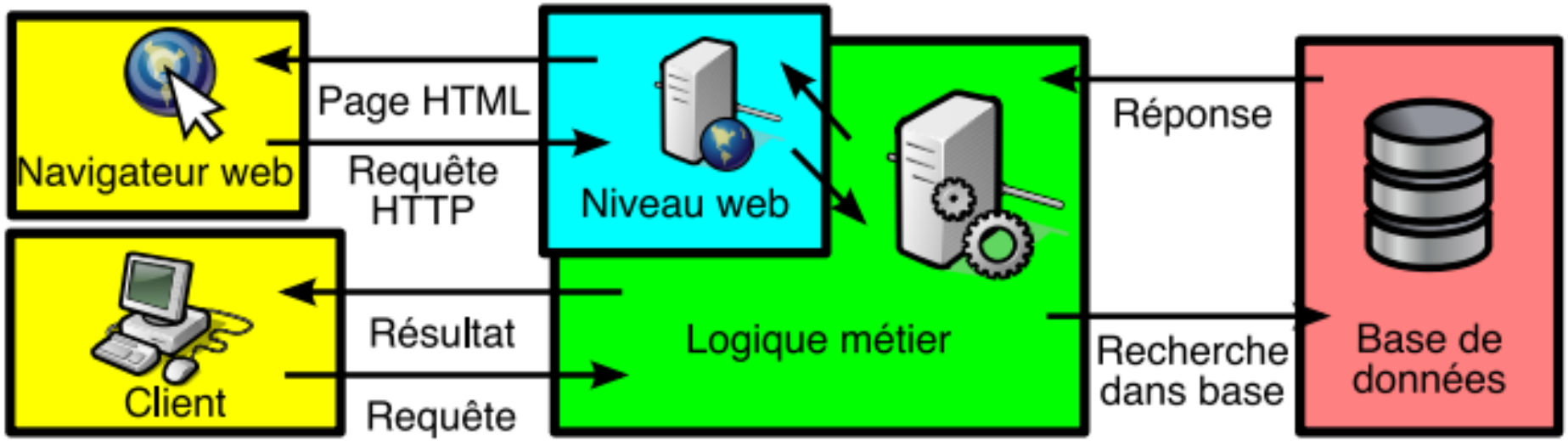
Exemple



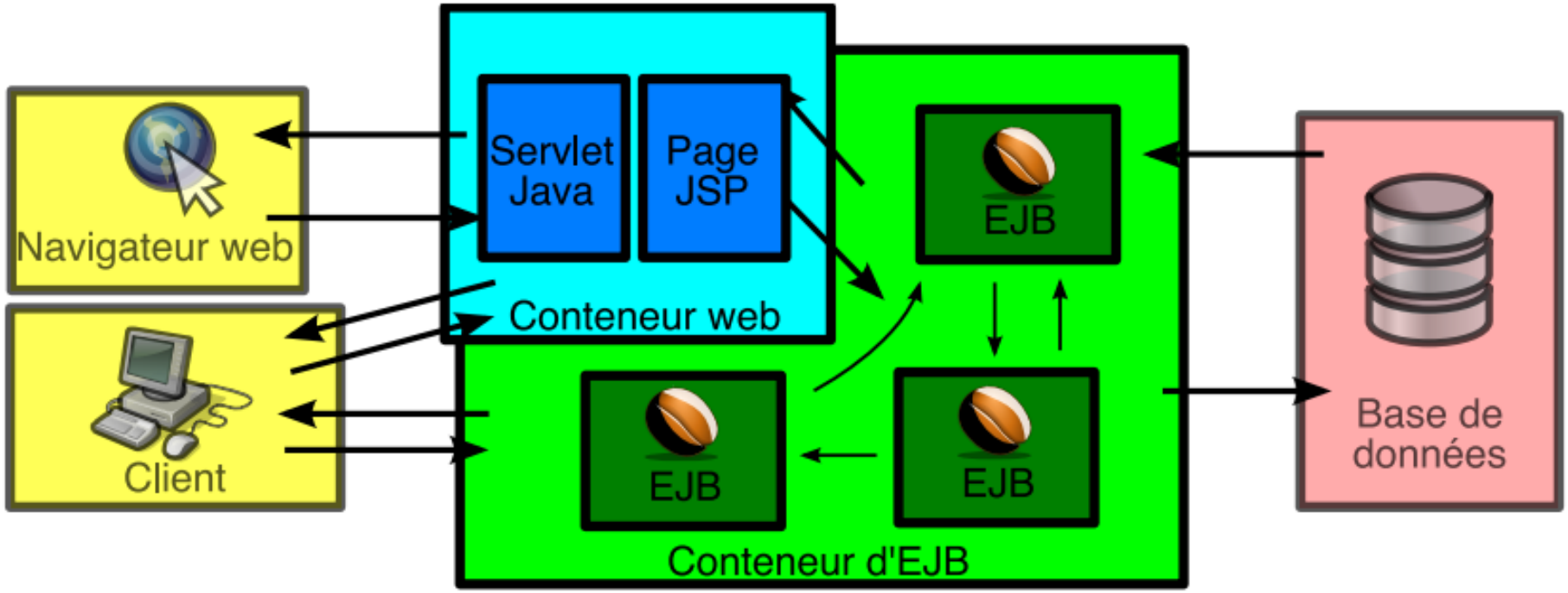
Exemple


Client
Informer de l'erreur
Emprunter le livre B
Logique métier
B n'est pas libre
Consulter le statut de B
Base de données

Niveau web


Navigateur web
Page HTML
Requête HTTP
Niveau web
Client
Résultat
Requête
Logique métier
Réponse
Recherche dans base
Base de données

Architecture JEE



Serveur d'applications

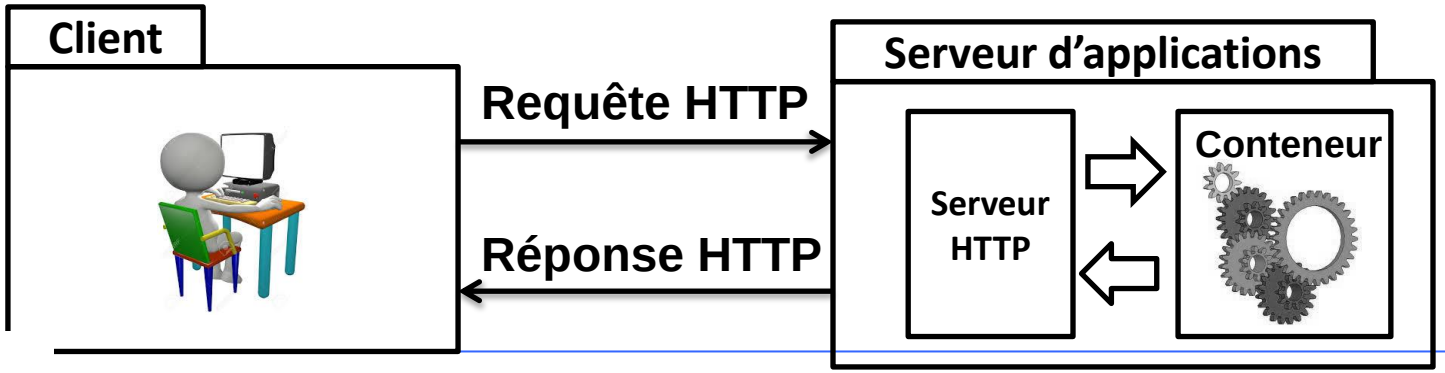
- Couche web
- Création de ressources web
  - à l'aide de programmes spécifiques (Servlets java)
  - à l'aide de langages étendant HTML pour ajouter des parties dynamiques calculées par le serveur, type JSP
- Les ressources sont accessibles pour le client à l'aide d'un simple navigateur → portabilité, simplicité

Serveur d'applications

- **Une fois une requête HTTP est lue et analysée, il faut traiter son contenu et renvoyer une réponse au client,**

➔ code qui indique le traitement à faire suite à l'arrivée d'une requête.

- Composant qui se charge d'exécuter le code + faire le travail d'un serveur HTTP → **Serveur d'applications**
- **Serveur d'applications = serveur HTTP + gestion d'objets de diverses natures → Conteneur**



Serveur d'applications

Modèle de conception pour JEE

- Travailler à plusieurs contributeurs sur un même projet/application (travail en équipe)
- Maintenir et corriger une application créée par d'autre programmeur/équipe
- Faire évoluer une application créée par d'autre programmeur/équipe

Modèle de conception pour JEE

- **Java EE n'impose aucun arrangement particulier de votre code.**
  - **Problème:** code mal organisé, difficulté de réutilisation (très difficile de trouver une fonction ou portion de code), difficulté de modification pour mise à jour,

Modèle de conception pour JEE

- **Java EE n'impose aucun arrangement particulier de votre code.**
  - **Problème:** code mal organisé, difficulté de réutilisation (très difficile de trouver une fonction ou portion de code), difficulté de modification pour mise à jour,
  - **Solution:** un **Design pattern MVC** (Modèle, Vue, Contrôleur)

    ➔ Adopter une architecture standard dans laquelle tout développeur peut se situer → Modèle MVC (Modèle-Vue-Contrôleur)

Le modèle MVC

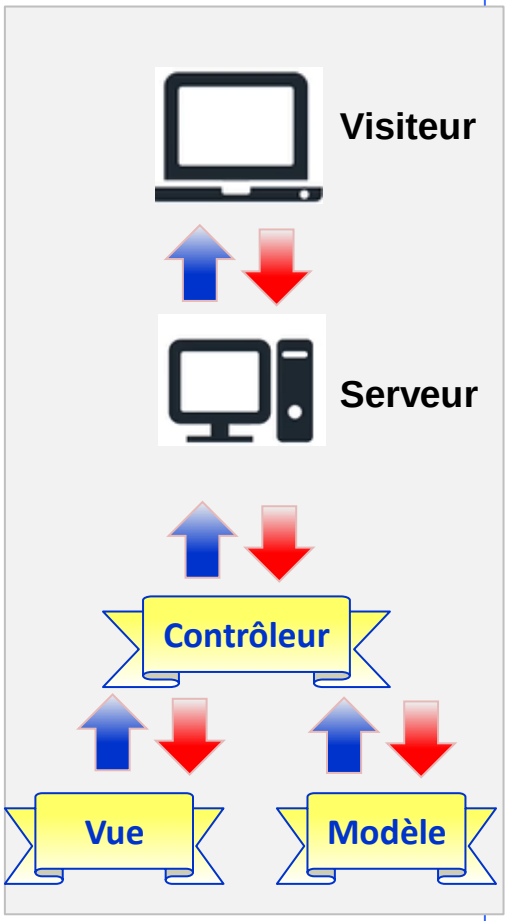
- **Il découpe l'application en couches distinctes et de ce fait impacte très fortement l'organisation du code.**
  - Le traitement, le stockage et la mise à jour des données de l'application doit être contenu dans la couche Modèle (M du MVC).
  - L'interaction avec l'utilisateur et la présentation des données (mise en forme, affichage) doit être contenu dans la couche Vue (V du MVC).
  - Le contrôle des actions de l'utilisateur et des données doit être contenu dans la couche Contrôle (C du MVC).

Le modèle MVC

Java EE

Quand on utilise MVC, on découpe le code en trois parties:

- Le visiteur qui envoi une requête HTTP, la transmet au serveur d'applications
- Le serveur d'applications transmet la requête au contrôleur
- Le contrôleur est une partie du code qui permet de faire le routage de l'information (décide qui va récupérer l'information et ensuite la traiter)
- Le contrôleur appelle le modèle qui contient les informations structurées de nos données et qui permet de faire des calculs.
- Le modèle envoi ces informations au contrôleur
- La vue génère ensuite une page web (une vue qui représente la page final envoyé au visiteur)
- La vue est alors renvoyée au contrôleur et par la suite au visiteur

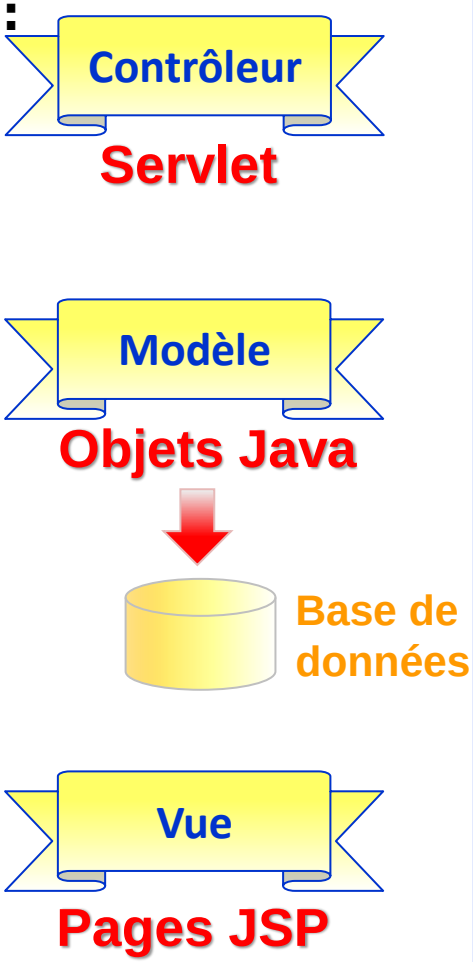


Le modèle MVC

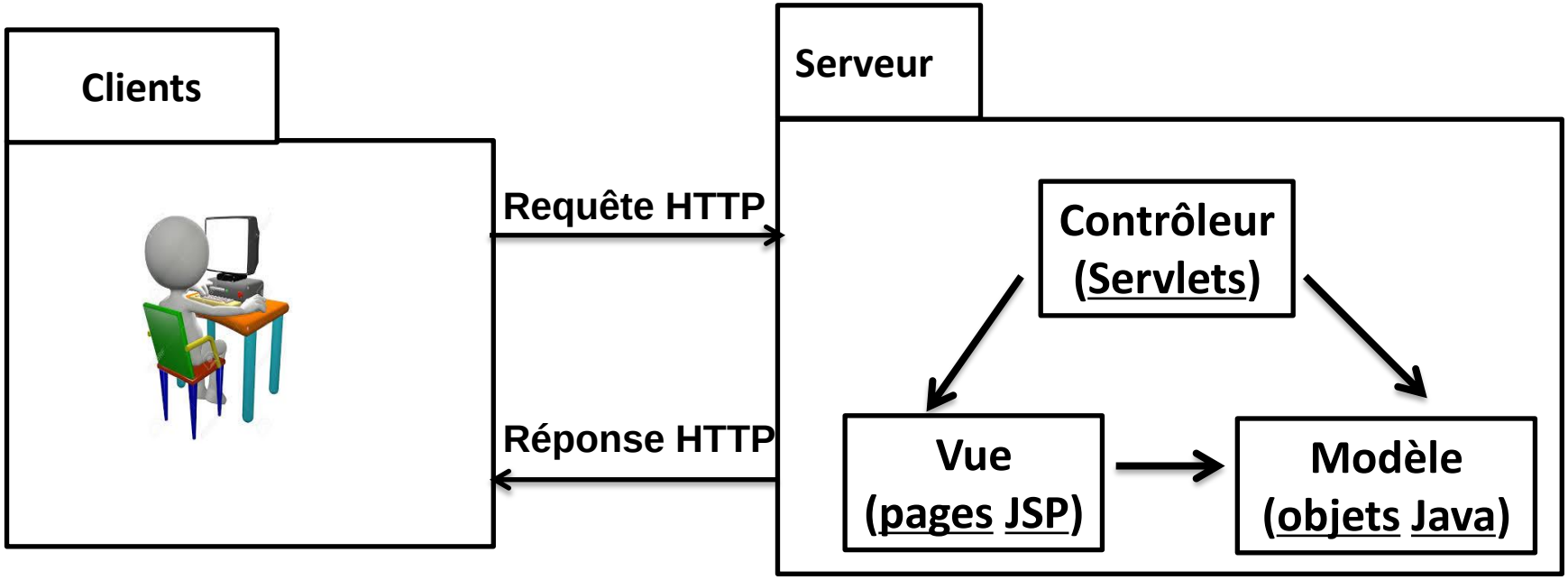
Noms des éléments

Java EE

- **Avec Java EE chacun de ces éléments possèdent un nom:**
  - Le contrôleur est appelé Servlet. La servlet est alors utilisée pour faire le routage des informations (à voir dans la prochaine partie).
  - Le modèle est en générale géré par des objets Java (des JavaBeans), Le modèle peut être amené à communiquer avec une base de données pour stocker les informations.
  - La vue est gérée par des pages JSP qui utilisent le code HTML et du code spécifique en Java. Cette vue est retournée par le contrôleur au visiteur.



Le modèle MVC avec JEE

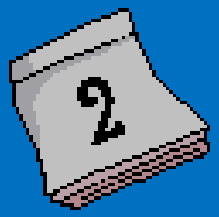


Les Framework Java EE

- **Il existe plusieurs Framework Java EE pour pouvoir développer avec MVC:**
  - Un Framework Java EE est un ensemble de bibliothèques structuré en MVC:
    - JSF
    - Struts
    - Spring
    - HIBERNATE
    - Spring Boot

Ensemble de bibliothèques structuré en MVC

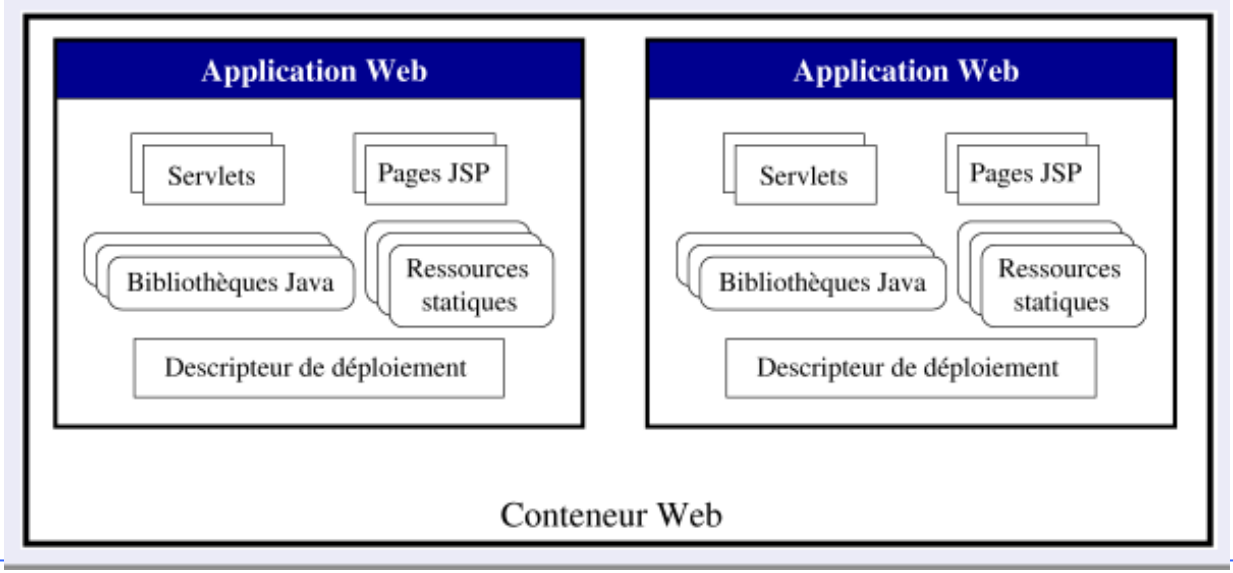
Chapitre



Niveau Web : Les Servlets

Application Web

- **Une application web est composée**
  - de Servlets/JSP
  - de bibliothèques de classes,
  - de ressources statiques (images, HTML, . . .),
  - un descripteur de déploiement (fichier XML)



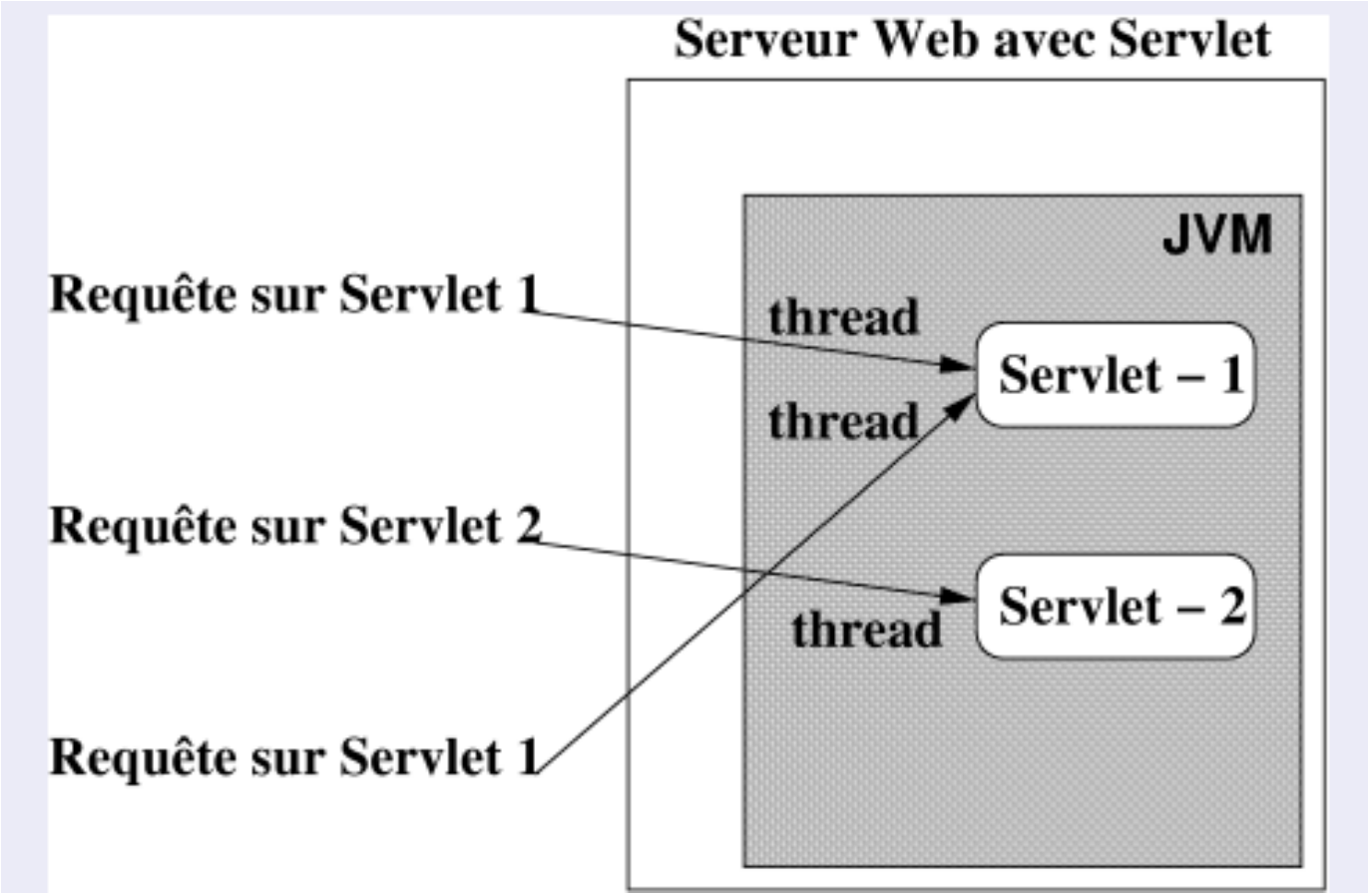
JEE et les applications Web

- **Composant Web : Servlets et JSP**
- **Application Web : ensemble de composants web, bibliothèques et ressources statiques dont l'organisation est décrite dans un descripteur de déploiement**
- **Conteneur Web :**
  - environnement d'exécution et de distribution des composants web et des ressources statiques
  - gère le cycle de vie des Servlets (instanciation, initialisation,…)

Que sont les Servlets ?

- Application java
- Exécutée côté serveur
- Traitement de requêtes HTTP et génération de réponses dynamiques
- Quelques avantages :
  - portabilité (java)
  - rapidité (la Servlet est chargée une seule fois)

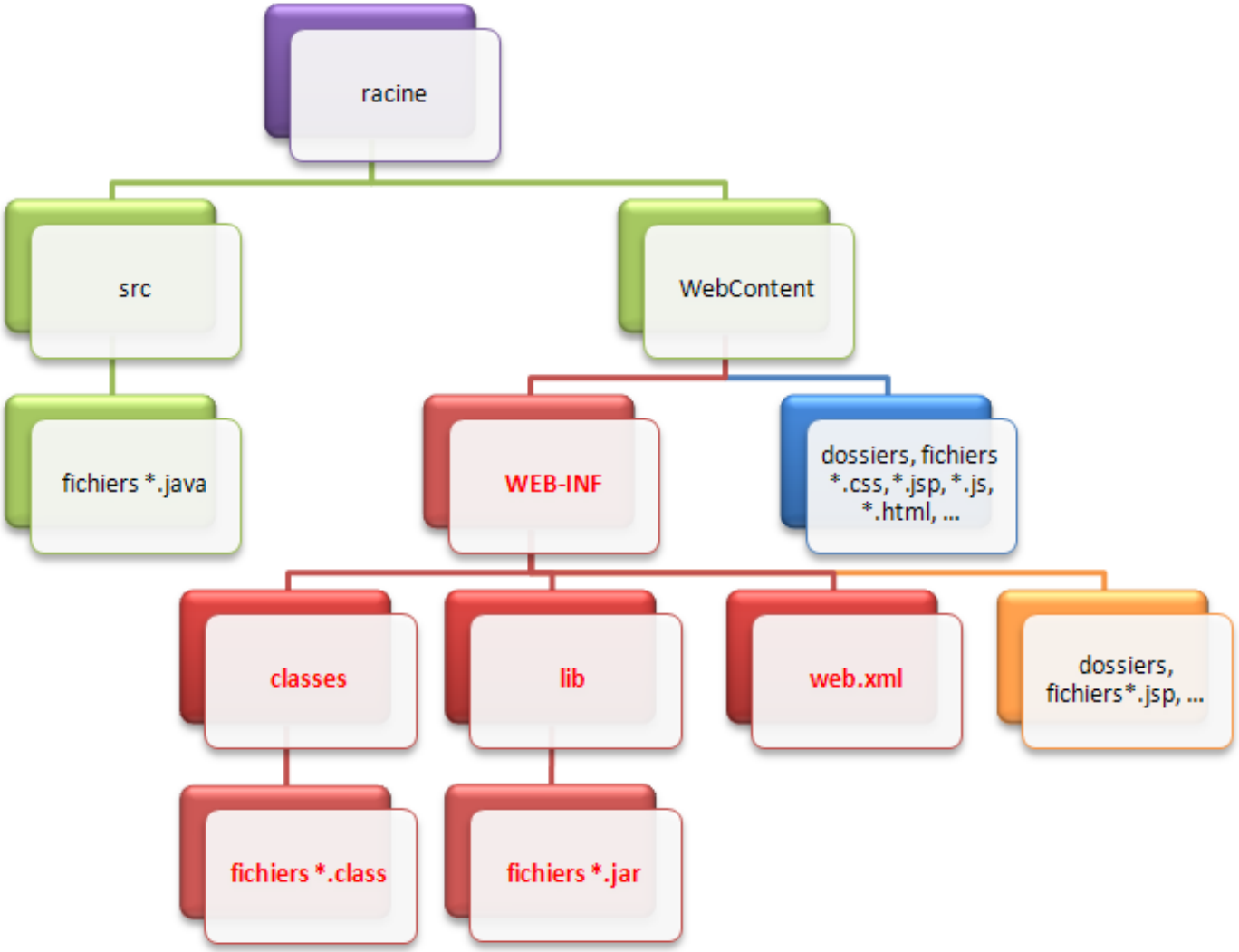
Fonctionnement des Servlets



Classe HttpServlet

- **Classe abstraite → passer par une servlet qui en hérite**
- **Propose les méthodes Java nécessaires au traitement des requêtes et réponses HTTP**
  - doGet () pour gérer la méthode GET
  - doPost () pour gérer la méthode POST
  - doHead () pour gérer la méthode HEAD

Structure des fichiers d'une application Web sous Eclipse



Servlet: fonctionnement

- Servlet existe --> comment l'associer à l'application?
- Associer la servlet à une URL → web.xml
- /WEB.INF
- Le contenu est impérativement placé entre <web-app> et </web-app>

Servlet: définition

```
<servlet>
<servlet-name>Test</servlet-name>
<servlet-class>com.sdzee.servlets.Test</servlet-class>
</servlet>
```

```
<servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>com.servlets.Test</servlet-class>
    <description>Ma première servlet de test.</description>
   <init-param>
        <param-name>auteur</param-name>
      <param-value>Folen</param-value>
   </init-param>
</servlet>
```

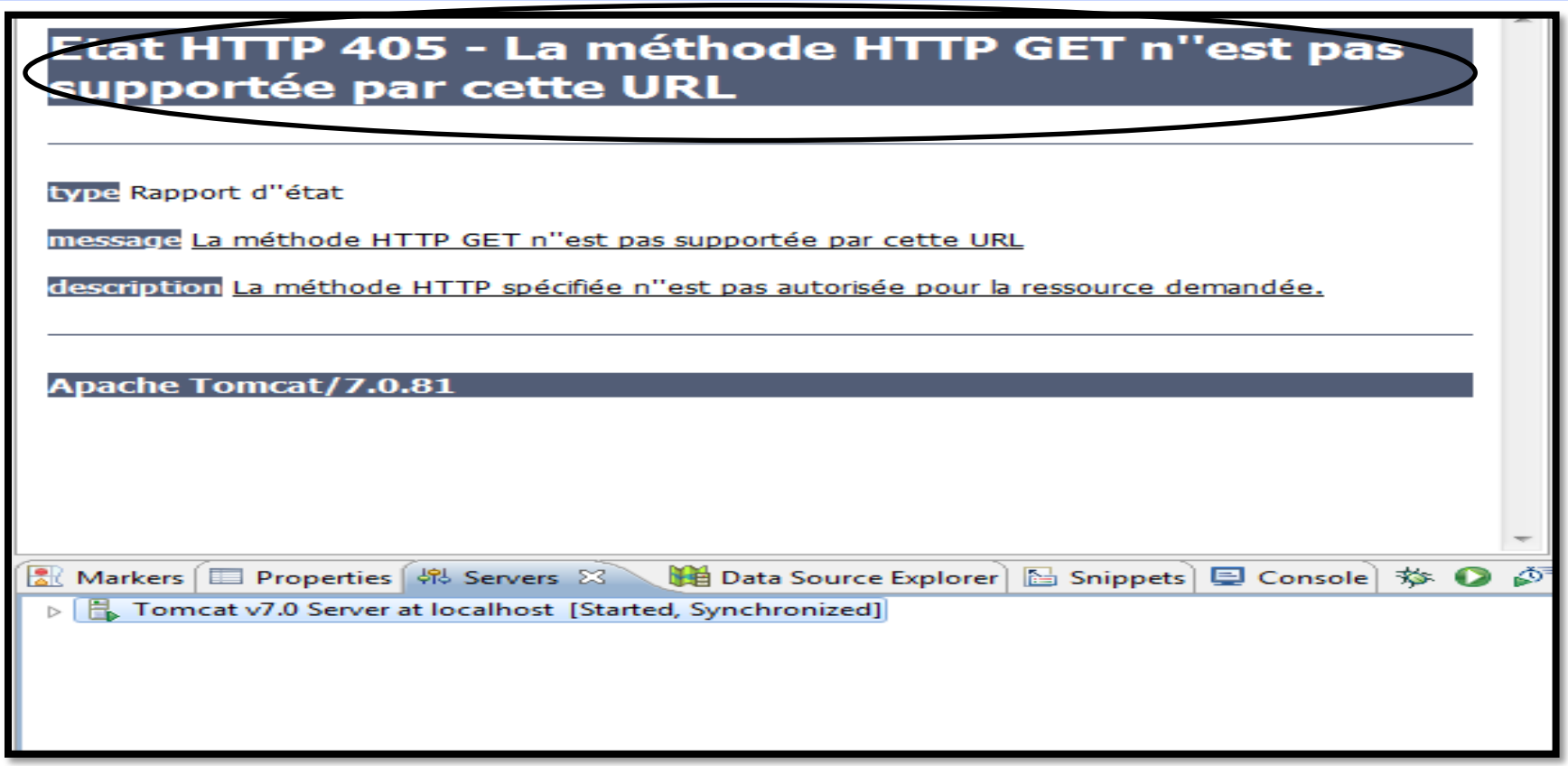
Mapping de la servlet

- **Correspondre la servlet créée à une URL pour qu'elle soit joignable par les clients!**

```
<servlet-mapping>
    <servlet-name>Test</servlet-name>
    <url-pattern>/toto</url-pattern>
</servlet-mapping>
```

- **<servlet-name> permet de préciser le nom de la servlet . Cette information doit correspondre avec le nom défini dans la précédente section.**
- **<url-pattern> permet de préciser la/les URL avec lesquelles la servlet sera accessible.**
- **Pourquoi un pattern et pas simplement une URL**

Servlet: Mise en service



La méthode doGet () de la classe mère HttpServlet !!

Servlet: Mise en service

Surcharger doGet () de la classe HttpServlet

```
package com.sdzee.servlets
import  java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlets ;
import javax.servlet.http.HttpServletRequest ;
import javax.servlet.http.HttpServletResponse;
public class Test1 extends HttpServlet {
        public void doGet (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    }
}
```

Servlet: Mise en service

Java EE

- **Résultat: 2 cas**
  - 1) affichage d'une page blanche (le cas de l'exemple) → requête terminée avec succès
  - 2) code / message d'erreur HTTP

Servlet: Résultat

Réponse

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Lenght:0
Date: Mon, 25 Sep 2017 06:37:33 GMT

Requête

GET /test/toto HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0
*Accept: text/html, application/xhtml+xml, application/xml; q=0.9, */*; q=0.8*
Accept-Language: fr, fr-fr; q=0.8, en-us;q=0.6,en;q=0.4,zh-cn;q=0.2
Accept-Encoding: gzip, deflate
Cache-Control: max-age=0

Remarque: **<load-on-startup> N </load-on-startup>**

<load-on-startup>

- La servlet est chargée lors de la première requête
- Utilisation de <load-on-startup> N </load-on-startup>
  - N >=0 : la servlet ayant la valeur la plus petite sera chargée avant la servlet ayant une valeur plus grande. En d'autres termes, le conteneur charge les servlets par valeur croissante.
    - La valeur 0 sera chargée avant les valeurs 1, 2, 3, etc.
  - N < 0 : la servlet sera chargée lors de l'envoi de la première requête.

Servlet: Mise en service

```
import  java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlets ;
import javax.servlet.http.HttpServletRequest ;
import javax.servlet.http.HttpServletResponse;
public class Test1 extends HttpServlet {
        public void doGet (HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    }
}
```

Servlet API

- **Les packages javax.servlet et javax.servlet.http représentent des interfaces et des classes pour l'API de servlet**
  - Le package javax.servlet contient de nombreuses interfaces et classes utilisées par le servlet ou le conteneur Web. Celles-ci ne sont spécifiques à aucun protocole.
  - Le package javax.servlet.http contient des interfaces et des classes responsables uniquement des requêtes http.

Servlet API

Java EE

Les interfaces du package javax.servlet :

- Servlet
- ServletRequest
- ServletResponse
- RequestDispatcher
- ServletConfig
- ServletContext
- SingleThreadModel
- Filter
- FilterConfig
- FilterChain
- ServletRequestListener
- ServletRequestAttributeListener
- ServletContextListener
- ServletContextAttributeListener.

Servlet API

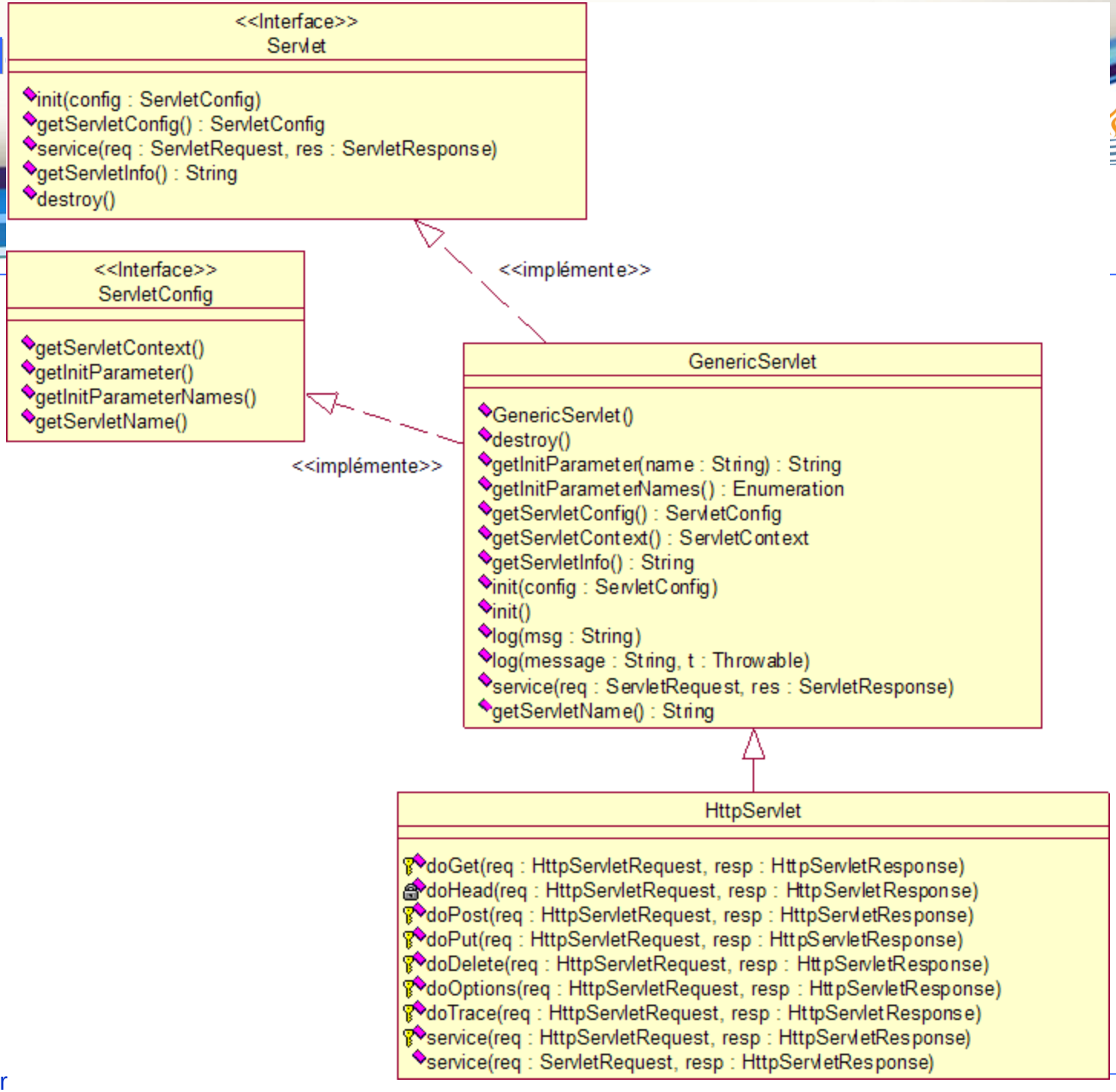
- Les interfaces du package javax.servlet.http :
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
  - HttpSessionListener
  - HttpSessionAttributeListener
  - HttpSessionBindingListener
  - HttpSessionActivationListener

Servlet API

Java EE

- Les classes du package javax.servlet.http :
  - HttpServlet
  - Cookie
  - HttpServletRequestWrapper
  - HttpServletResponseWrapper
  - HttpSessionEvent
  - HttpSessionBindingEvent

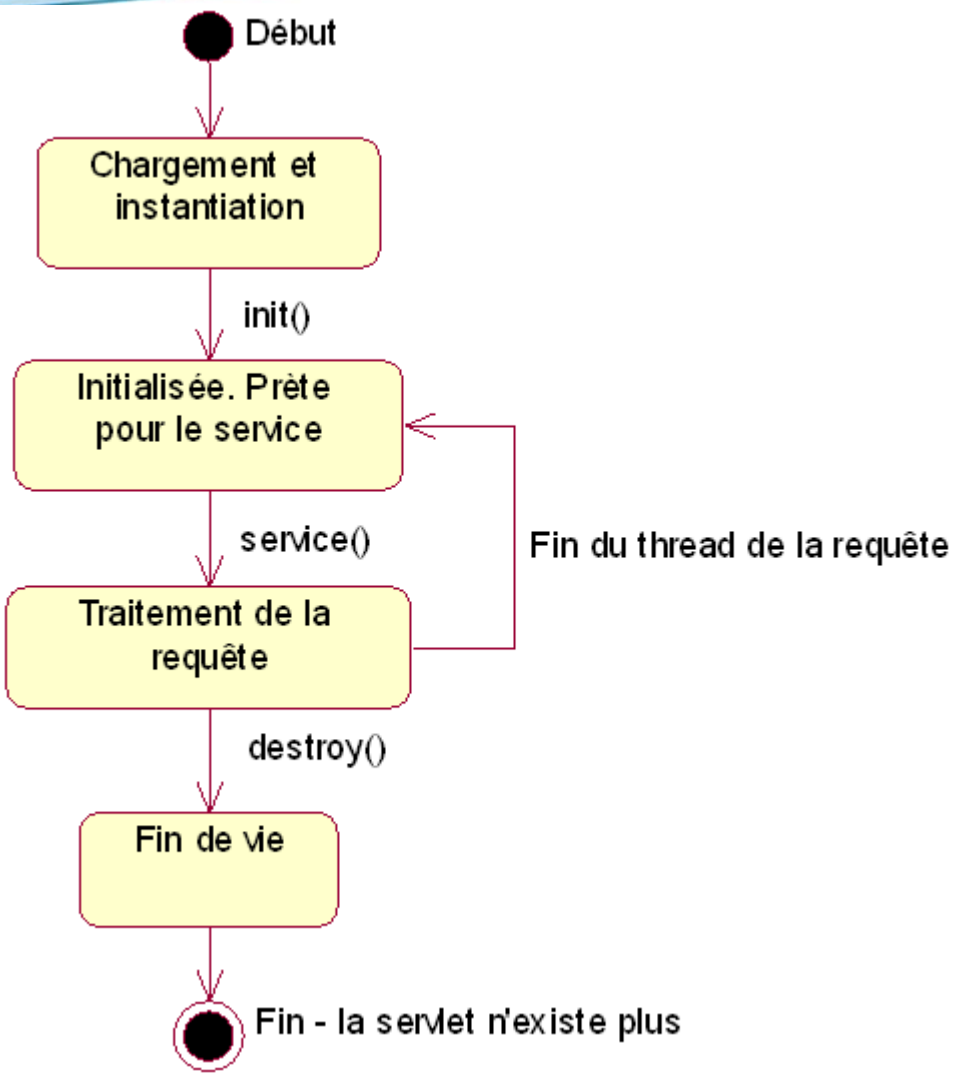
Servl


<<Interface>>
Servlet
init(config : ServletConfig)
getServletConfig() : ServletConfig
service(req : ServletRequest, res : ServletResponse)
getServletInfo() : String
destroy()
<<implémente>>
<<Interface>>
ServletConfig
getServletContext()
getInitParameter()
getInitParameterNames()
getServletName()
<<implémente>>
GenericServlet
GenericServlet()
destroy()
getInitParameter(name : String) : String
getInitParameterNames() : Enumeration
getServletConfig() : ServletConfig
getServletContext() : ServletContext
getServletInfo() : String
init(config : ServletConfig)
init()
log(msg : String)
log(message : String, t : Throwable)
service(req : ServletRequest, res : ServletResponse)
getServletName() : String
HttpServlet
doGet(req : HttpServletRequest, resp : HttpServletResponse)
doHead(req : HttpServletRequest, resp : HttpServletResponse)
doPost(req : HttpServletRequest, resp : HttpServletResponse)
doPut(req : HttpServletRequest, resp : HttpServletResponse)
doDelete(req : HttpServletRequest, resp : HttpServletResponse)
doOptions(req : HttpServletRequest, resp : HttpServletResponse)
doTrace(req : HttpServletRequest, resp : HttpServletResponse)
service(req : HttpServletRequest, resp : HttpServletResponse)
service(req : ServletRequest, resp : HttpServletResponse)

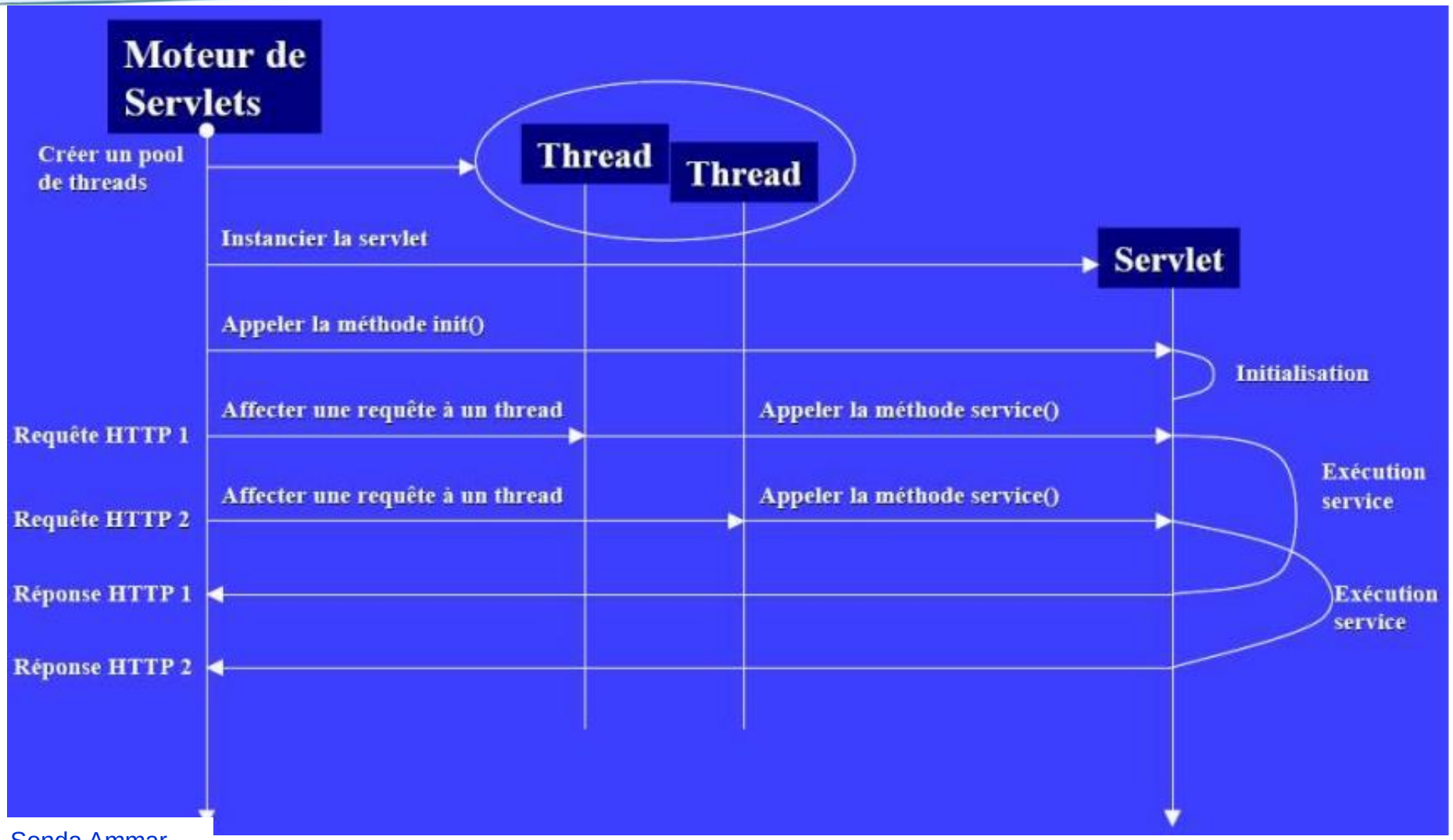
Servlet API

- **L'interface Servlet fournit un comportement commun à tous les servlets. Elle définit les méthodes que tous les servlets doivent implémenter.**
  - 3 méthodes de cycle de vie qui sont utilisées pour initialiser le servlet, traiter les requêtes et détruire le servlet:
    - public void init(ServletConfig config)
    - public void service(ServletRequest request,ServletResponse response)
    - public void destroy()
  - 2 méthodes de non-cycle de vie
    - public ServletConfig getServletConfig()
    - public String getServletInfo()

Cycle de vie de Servlet

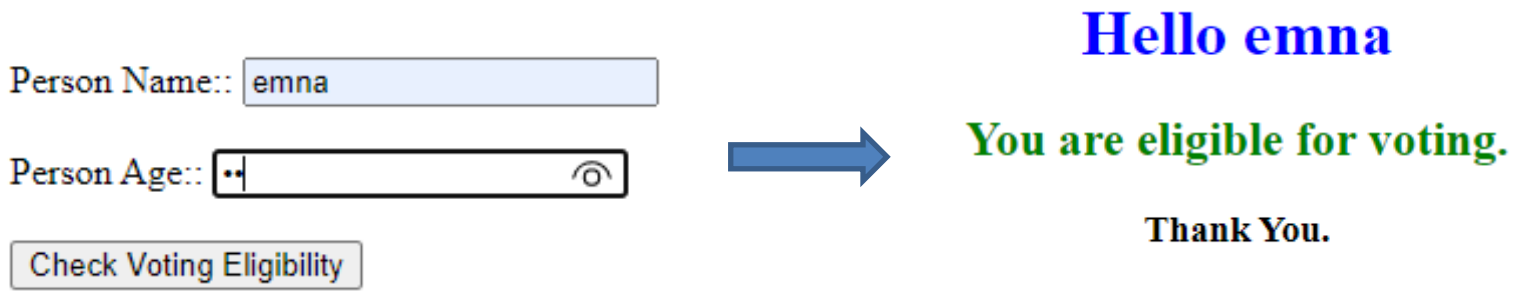


Cycle de vie de Servlet



Mise en œuvre d'une servlet

- Développer une application Web basée sur une Servlet pour vérifier si un utilisateur est éligible au vote ou non.
- Prenez un formulaire HTML et obtenez des commentaires de l'utilisateur.
- Sur la base de la vérification de l'âge saisie, il / elle est éligible pour voter ou non? Affichez le message approprié à l'utilisateur final. Supposons que 18 ans est l'âge minimum pour voter.



Mise en œuvre d'une servlet : le fichier appelant

web.xml | index.html | web.xml | Clock | 32

```
1
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="ISO-8859-1">
6  <title>test</title>
7  </head>
8  <body>
9  <div>
10     <form action="display" method="POST">
11      Person Name:: <input type="text" name="pname"><br><br>
12      Person Age:: <input type="password" name="page"><br><br>
13      <input type="submit" value="Check Voting Eligibility">
14      </form>
15 </div>
16 </body>
17 </html>
18
```

Mise en œuvre d'une servlet : Servlet

```
// declare variables
PrintWriter pw = null;
String name = null;
String tage = null;
int age = 0;

// set content type
res.setContentType("text/html");

// get PrintWriter object
pw = res.getWriter();

// get form data (from req parameter)
name = req.getParameter("pname");
tage = req.getParameter("page");

// convert String value to int
age = Integer.parseInt(tage);

// check age
pw.println("<h1 style='text-align: center; color:blue'>"
          + "Hello "+ name + "</h1>");
if(age < 18) {
   pw.println("<h2 style='text-align: center; color:red'>"
              +"You are not eligible for voting.</h2>"
              +"<h3 style='text-align: center; color:green'>"
              +"Please wait for more " + (18-age) + " years.<br>"
              +" Thank You.<h3>"
   );
} else  {
   pw.println("<h2 style='text-align: center; color:green'>"
              +"You are eligible for voting.</h2>"
              +"<h3 style='text-align: center'>"
              +"Thank You.<h3>"
```

Mise en œuvre d'une servlet : le fichier de déploiement

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 <display-name>WelcomeFile</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
   <servlet>
      <servlet-name>test</servlet-name>
      <servlet-class>chapitre1.MyServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>test</servlet-name>
      <url-pattern>/display</url-pattern>
   </servlet-mapping>

</web-app>
```

Vérification de formulaires

Données manquantes

- Champ manquant dans le formulaire
  - ✗ getParameter retourne null
- Champ renvoyé vide
  - ✗ getParameter retourne une chaine vide

```
String param=        request.getParameter("someName");
if((param == null) || (param.trim().equals(""))){
doSomethingForMissingValues(...);
} else{
doSomethingWithParameter(param);
}
```

Contexte d'application

- Application Web disposant de plusieurs servlets
- L'élément <init-param> défini dans le descripteur de déploiement, est associé à une servlet en particulier.
- Placer les mêmes paramètres d'initialisation <init-param> pour chacune de ces servlets ?

➔ Utilisation des paramètres d'initialisation de l'application Web, appelé paramètres de contexte <context-param>, qui sont accessibles pour tous les composants Web constituant l'application Web.

```
<context-param>
    <param-name>jdbc.Driver</param-name>
    <param-value>com.mysql.jdbc.Driver</param-value>
</context-param>
```

Contexte d'application

- Pour que chacune des servlets récupèrent les paramètres, il est nécessaire de solliciter l'objet correspondant au contexte de la servlet (contexte de l'application Web) à l'aide de la méthode getServletContext().
- Cet objet délivre les paramètres initiaux de l'application Web grâce à la méthode getInitParameter().

```
String param = getServletContext().getInitParameter('param_name');
```

Contexte d'application

- **getServletContext ()**: elle retourne un objet de type ServletContext.
- **getContextPath ()**: chaîne de caractères correspondant à l'URL
- **getAttribute (…)** et **setAttribute(…)** : lire et écrire un attribut au niveau du contexte d'application.
- **getNamedDispatcher(…)**: retourne un objet de type RequestDispatcher permettant d'accéder à une autre servlet ou une JSP

Les différents contextes

- **Lorsqu'une requête est prise en charge par une servlet, le traitement peut demander l'usage d'informations se situant dans différents contextes.**
  - 1er contexte: l'application en question
  - 2ème contexte: la session
  - 3ème contexte: la requête

Les différents contextes : Contexte de la session

- L'application Web peut maintenir l'information pour l'ensemble des servlets et des pages JSP au travers de plusieurs transactions au cours de la navigation de l'utilisateur, de sa première requête jusqu'à la fin de la conversation ➔ la persistance.
- La session est accessible au travers de la méthode getSession() ou getSession(boolean) de l'objet de type HttpServletRequest passé en paramètres des méthodes doXXX().
- public void invalidate() ; ➔ supprimer une session

Les différents contextes : Contexte de requête

- L'objet de type **HttpServletRequest** passé en paramètre des méthodes doXXX() permet d'accéder aux informations de la requête

Lecture des informations de l'URL

Méthode	Rôle
getScheme ()	Protocole utilisé (http/https)
getServerName ()	Nom de l'hôte de la machine (nom domaine, …)
getServerPort ()	Port utilisé par la req HTTP
getContextPath ()	Nom de l'application (nom du projet)
getServletPath ()	Chemin d'accès à la servlet
getQueryString ()	Chaîne de carac correspondant à tous les params passés à la req
getRemoteAddr ()	@ IP de l'émetteur de la req HTTP
getRemoteHost ()	Nom de l'hôte de l'émetteur de la req HTTP
getLocalAddr ()	@ IP de la carte réseau du serveur par laquelle la req HTTP a été reçue
getLocalName ()	Le nom d'hôte de la carte réseau du serveur par laquelle la req HTTP a été reçue
getLocalPort ()	Le port sur lequel a été reçue la req HTTP

Lecture de l'en-tête de la requête

Java EE

Méthode	Rôle
getcharacterEncoding ()	L'encodage utilisé pour le corps de la req ou null si la req ne définit pas d'encodage
getContentLength () et getContentLenghtLong ()	Taille en octet des informations situées dans le corps de la req ou -1 si la taille n'est pas connue
getContentType ()	Retourne le type de média ou null si le type n'est pas connu
getLocale ()	Retourne un objet de type java.util.Locale (Langues à utiliser au niv du navigateur)
getMethod ()	Méthode utilisée par la req (GET, POST, …)
getHeader (String name)	Retourne la valeur de l'entête passée en paramètre
getHeaders (String name)	Retourne la valeur de l'entête passée en paramètre sous la forme d'une collection (Accept-Language)
getIntHeader (String name)	Valeur de l'entête sous la forme d'un entier ou -1 si l'entête n'est pas présent
getDateHeader (String name)	Pour l lecture des paramètres comme If-Modified-Since
getHeaderNames (String name)	Collection de l'ensemble des en-têtes présents sur la req.

Lecture des paramètres

- **Elément primordial pour traiter une réponse**
- **Les informations saisies dans un formulaire sont envoyées sous forme de paramètres au niveau de la req.**
  - Req de type GET: URLValide?parametre1=val1&parametre2=val2…
  - Req de type POST: les paramètres sont structurés de la même manière mais dans le corps de la requête.
- **Dans un formulaire HTML**
  - le nom du paramètre → attribut name de la balise HTML
  - la valeur → valeur de saisie pour une zone de texte
    → value pour les balises de sélection

Lecture des paramètres dans la servlet

Java EE

Objet HttpRequestServlet

- getParameter (String name)
- getParameterValues (String name)
- getParameterNames ()
- getParameterMap () : objet de type Map <String, String[]>

Lecture des paramètres : Exemple

```
<!DOCTYPE html>
<html>
<head>
      …
</head>
<body>
     <form method= "get">
              <input type = "text" name = "inputText"/>
             <select name="selectSport" multiple="multiple">
                 <option value="1">Badminton</option>
                 <option value="2">Squash</option>
                 <option value="3">Tennis</option>
              </select> <br/>
             <input size="50" type="text "value="input sans attribut name. Pas de paramètre
     créé"/>  <br/>
             <input type="submit"  value="Valider"/>
      </form>
</body>
</html>
```

Lecture des paramètres : Exemple

Ajouter un nouveau terrain et les sports praticables sur celui-ci
Formulaire ***formulaireAjoutTerrain***

```
<!DOCTYPE html>
<html>
<head>
    …
</head>
<body>
        <form method= "post" action="/Projet_servlet_3_1/AjouterTerrain">
         Code:  <input type="text" name="codeTerrain"/>   <br/>
         Sports autorisés:
          <select name="sportsAutorise"multiple="multiple">
              <option value=1>Badminton</option>
              <option value=2>Squash</option>
              <option value=3>Tennis</option>
         </select>   <br/>
         <input type="submit"  value="Valider"/>
         </form>
</body>
```

Lecture des paramètres : Exemple

- Formulaire pointe vers l'URL Projet_servlet_3_1/AjouterTerrain
- URL Associée à la servlet AjouterTerrain

```
public class AjouterTerrain extends HttpServlet
        {
                protected void doPost (HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                String codeTerrain = request.getParameter ("codeTerrain");
                String [] sportsAutorises = request.getParameterValues
("sportsAutorises");
                System.out.println ("Code du terrain :" +codeTerrain);
                for (String codeSport:sportAutorises)
                    {
                        System.out.println("Code sport autorisé : "+
codeSport);
                    }
                }
```

L'en-tête de la réponse HttpServletResponse

Méthode	Rôle / Signification
setStatus (int sc)	Code de statut de la réponse
HttpServletResponse.SC_SWITCHING_PROTOCOLS	101: communication va changer de protocole→ communiquer avec WebSocket (protocole réseau + interface de programmtion)
HttpServletResponse.SC_OK	200
HttpServletResponse.SC_MOVED_PERMANTLY	301. Ressource demandée a changé d'URL
HttpServletResponse.SC_FORBIDDEN	403. Serveur a refusé de traiter la req
HttpServletResponse.SC_INTERNAL_SERVER_ERROR	500. Erreur de traitement côté serveur
setHeader (String name, String value)	Appliquer une valeur à un en-tête donné
setDateHeader (String name, long date)	L'écriture des en-têtes de type Date
setIntHeader (String name, int value)	L'écriture des en-têtes de type int

Corps de la réponse

Java EE

- **Contenu affiché par le client**
  - **Objet de type java.io.PrintWriter par l'appel de la méthode getWriter() (contenu textuel)**
  - **Javax.servlet.ServletOutputStream par l'appel de la méthode getOutputStream () (contenu binaire)**

L'exception **IllegalStateException** est levée si les deux méthodes sont appelées pour ***la même réponse***

Méthode(s)	Rôle
append(…) print(…) println (…) prinf(…) write (…)	Ajouter le contenu passé en paramètre
flush()	Vider le tampon
Close ()	Fermer le flux, pas d'ajout du contenu

Gestion de l'envoi de la réponse

Méthode	Rôle
setBufferSize (int size)	définir la taille souhaitée pour le tampon contenant le corps de la réponse
getBuffersize ()	Obtenir la taille actuelle du tampon
reset ()	Efface la réponse (entête et corps)
resetBuffer ()	Efface le contenu du corps
isCommited ()	Connaitre si la réponse a déjà commencé à être retournée au client
flushBuffer ()	Forcer l'envoi au client du contenu écrit au niveau de la réponse

Exemple de mise en œuvre : Ecriture de la réponse

```
//Les en-têtes
response.setStatus (HttpServletResponse.SC_ok).
response.setContentType("text/html");
response.setCharacterEncoding("UTF-8");
//Le corps
PrintWriter pw = response.getWriter ();
pw.println ("<!DOCTYPE html>");
pw.println ("<html>");
pw.println ("<head>");
pw.println ("<meta charset=\"UTF-8\">");
pw.println ("<title>Confirmation ajout d'un terrain</title>");
pw.println ("</head>");
pw.println ("<body>");
pw.println ("Confirmation ajout du terrain" + terrainAjoute.getCode());
pw.println ("</body>");
pw.flush ();
pw.close ();

//Réalisation du traitement métier  (dans les classes du projet)
Terrain terrainAjoute = new Terrain (codeTerrain, surface.BETON);
```

Remarques

- **Quatre étapes facilement identifiables**
  - 1) La lecture de la requête
  - 2) La réalisation du traitement métier (au niveau des classes java du projet)
  - 3) L'écriture des en-têtes de la réponse
  - 4) L'écriture du corps de la réponse → l'usage des JSP est plus adapté

Les annotations

- Package javax.servlet.annotation
- Une annotation est écrite directement dans le code, au sein d'une classe ou d'une méthode.
- L'information associée à un élément du code est accessible et visible directement ➔ n'est pas nécessaire de se référer à des données écrites dans des fichiers externes
- Syntaxe sans paramètres:
  - Constituée uniquement d'un mot-clé précédé du signe @: @Override
- Syntaxe avec paramètres
  - @WebServlet(name="TestServlet", urlPatterns = {"/test", "/ok"})

Les annotations: Exemple 1

```
<servlet>
    <servlet-name>Inscription</servlet-name>
    <servlet-class>com.sdzee.servlets.Inscription</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Inscription</servlet-name>
    <url-pattern>/inscription</url-pattern>
</servlet-mapping>
```

WEB-INF/web.xml

```
...

@WebServlet( name="Inscription", urlPatterns = "/inscription" )
public class Inscription extends HttpServlet {
    ...
```

Inscription.java

Les annotations: Exemple 2

```
<servlet>
    <servlet-name>Download</servlet-name>
    <servlet-class>com.sdzee.servlets.Download</servlet-class>
    <init-param>
        <param-name>chemin</param-name>
        <param-value>/fichiers/</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>Download</servlet-name>
    <url-pattern>/fichiers/*</url-pattern>
</servlet-mapping>
```

WEB-INF/web.xml

```
...

@WebServlet( urlPatterns = "/fichiers/*", initParams = @WebInitParam( name = "chemin", value = "/fichiers/" )
public class Download extends HttpServlet {
    ...
```

Inscription.java

Arborescence d'une archive WAR

Arborescence d'une archive WAR (Web ARchive)

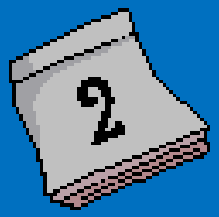
```
ApplicationWeb.war
  |_ fichiers.html
  |_ fichiers.jsp
  |_ repertoires\fichiers
  |_ src
  |    |_ sourcesServlets.java
  |    |_ sourcesbibliotheques.java
  |_ META-INF
  |    |_ MANIFEST.MF
  |_ WEB-INF
       |_ web.xml
       |_ classes
            |_ Servlets.class
            |_ lib
                 |_ bibliotheques.jar
```

- **un fichier ApplicationWeb.war (fichier archive web) ayant pour contenu applicationweb :**
  - cd ApplicationWeb ; jar cf applicationweb.war *

API: Déploiement

- Le répertoire ApplicationWeb est placé dans le répertoire d'applications du conteneur web
  - webapps pour Tomcat
  - JBOSS_DIR/server/JBOSS_MODE/deploy pour Jboss
- ou un fichier ApplicationWeb.war (fichier archive web) ayant pour contenu applicationweb :
  - cd ApplicationWeb ; jar cf applicationweb.war *

Chapitre



Systèmes distribués basés sur les Micro-services

Plan

- 1. Introduction
- 2. Les problématiques
- 3. L'architecture Microservices
- 4. Spring Boot Qu'est-ce que c'est ?

Introduction

- **Utilisation des applications gourmandes en ressources (streaming vidéo, banques, musique, e-commerce, etc.)**
  - Les entreprises sont confrontées à des challenges liés:
    - à la performance de ces services,
    - au coût de l'infrastructure technique et
    - au coût du développement et de la maintenance.

Les problématiques

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Problème n°1 :** comment faire en sorte que les applications proposées en ligne soient toujours disponibles et ne souffrent jamais de coupure ou de ralentissement, quelle que soit l'affluence des utilisateurs sur celles-ci ?
  - **Solution:** utiliser le Cloud. Celui-ci permet d'augmenter et de diminuer le nombre de ressources nécessaires au fonctionnement d'une application à la demande.

Les problématiques

Java EE

Les problématiques rencontrées par les entreprises

- Exigence: L'application doit être **scalable**: capable de s'étendre sur plus de ressources (serveurs, disques durs, BD), gardant une parfaite consistance dans ses données et une cohérence dans son comportement.
- Les applications conçues sur une architecture plus classique (3-tiers, par exemple) est très difficilement scalable.
- Il faut envisager, dès le départ, une autre approche dans la conception de l'application afin que celle-ci soit "Cloud-native" : conçue pour fonctionner et profiter pleinement des avantages qu'offre le cloud.

Les problématiques

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Problème n°2:** les entreprises se livrent à une "guerre de la mise à jour". comment faire en sorte que l'entreprise soit capable de faire évoluer son application de façon très fréquente et de répondre rapidement aux nouvelles fonctionnalités que propose la concurrence ?
  - Dans les applications traditionnelles, le processus de mise à jour est long et compliqué car il faut mettre à jour toute l'application.
  - Nécessité de re-tester et de déployer l'application complète. Cela est dû à la forte interdépendance au sein d'une application.

Les problématiques

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Solution:** Les entreprises doivent utiliser des architectures qui permettent de faire des mises à jour qui visent des composants ciblés, très rapidement, de façon fiable et sans se soucier des éventuelles conséquences sur le reste de l'application.

Les problématiques

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Problème n°3:** Les technologies utilisées pour développer ces applications évoluent très vite et les nouveautés offrent des avantages énormes. Comment les entreprises peuvent-elle s'adapter?
  - **Exemple:** entreprise qui permet de placer des transactions sur les marchés boursiers.
    - **de millions d'opérations par seconde. Il faut donc être capable de vendre et acheter très vite, à la microseconde près,**
    - **L'application de cette entreprise repose sur un algorithme puissant écrit en Java.**

Les problématiques

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Solution:** l'entreprise doit adopter une architecture qui lui permettra de passer d'une technologie à l'autre sur des portions individuelles de son application très facilement.

Les problématiques

Java EE™

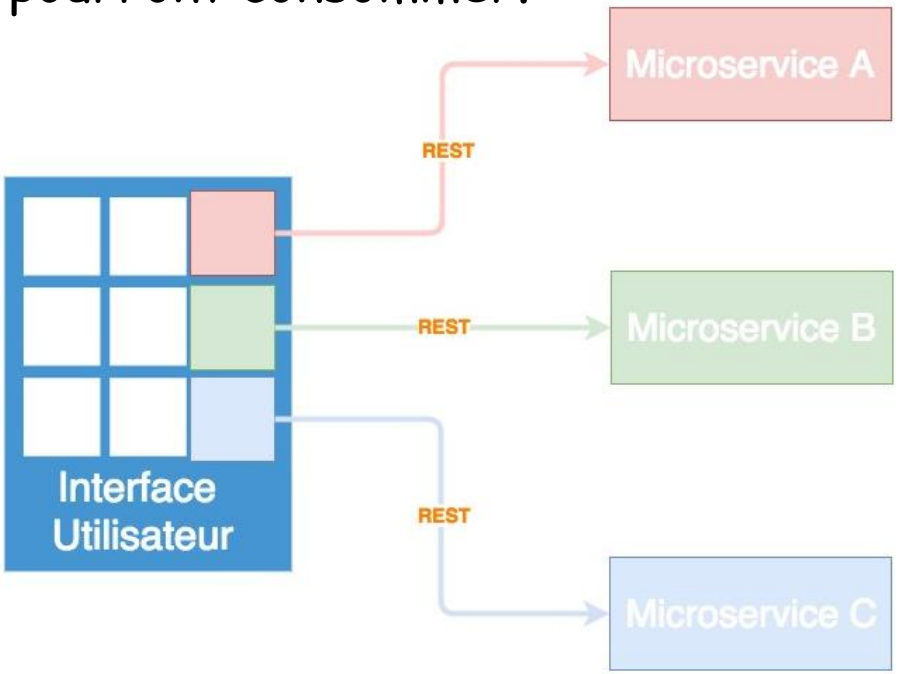
- **Les problématiques rencontrées par les entreprises**
  - **Problème n°4:** La sécurité, qui devient très compliquée à gérer dans les applications complexes, ainsi que les difficultés liées à la coordination entre développeurs. Comment les entreprises peuvent-elle s'en sortir?

Les problématiques

- **Les problématiques rencontrées par les entreprises**
  - **Solution:** les entreprises conçoivent leurs applications utilisant des architectures répondant à ces besoins et qui sont parfaitement adaptées au **cloud**.
  - L'**architecture Microservices** est apparue, apportant une réponse concrète à toutes ces préoccupations et la promesse d'obtenir, au bout, des applications dites "**Cloud-native**".

L'architecture Microservices

- **Quel est le principe de l'architecture Microservices ?**
  - Découper une application en petits services, appelés Microservices, parfaitement autonomes qui exposent une API REST que les autres Microservices pourront consommer.

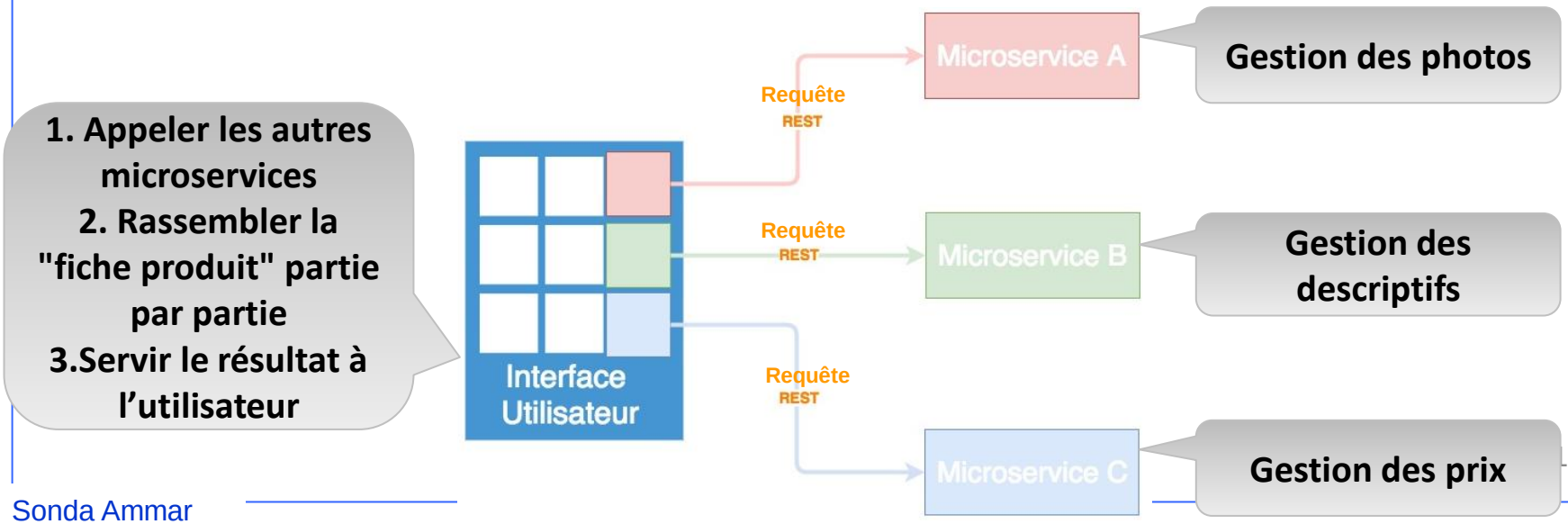


L'architecture Microservices

- **Exemple:** Application de vente de produits.
- Cette "fiche produit" est constituée d'une photo, d'un descriptif et d'un prix.
- Dans une architecture traditionnelle, vous avez une application au format WAR (**W**eb application **A**rchive) qui comporte tous les composants (JSP, servlets, classes Java, fichiers XML, et les pages web statiques (HTML, JavaScript…)). Lorsqu'un utilisateur demande une fiche de produit, l'application applique sa logique interne et va puiser dans une base de données, puis produit un HTML final.

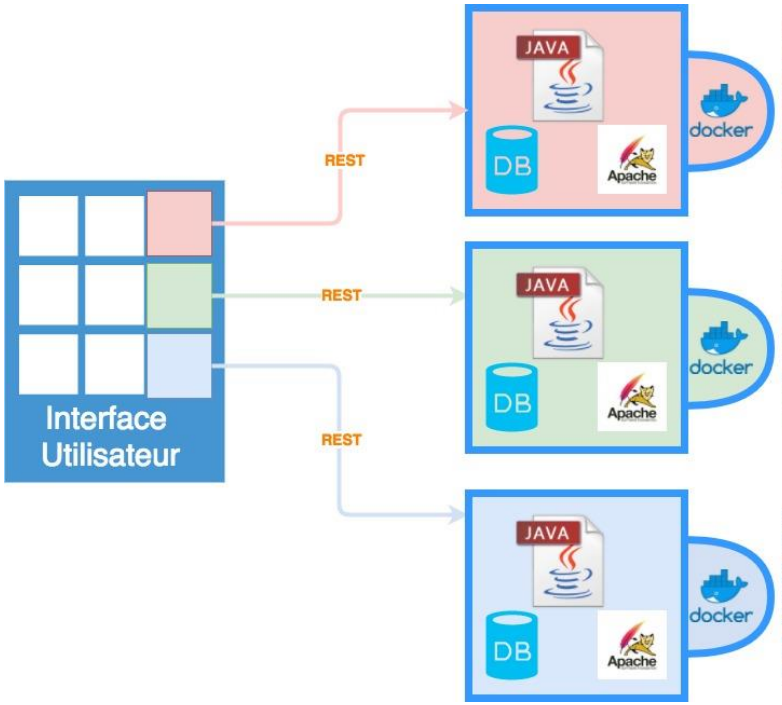
L'architecture Microservices

- **Exemple:** Application de vente de produits.
- Cette "fiche produit" est constituée d'une photo, d'un descriptif et d'un prix.
- Dans une architecture Microservice, L'interface utilisateur fait appel à un Microservice pour chaque composant à renseigner.

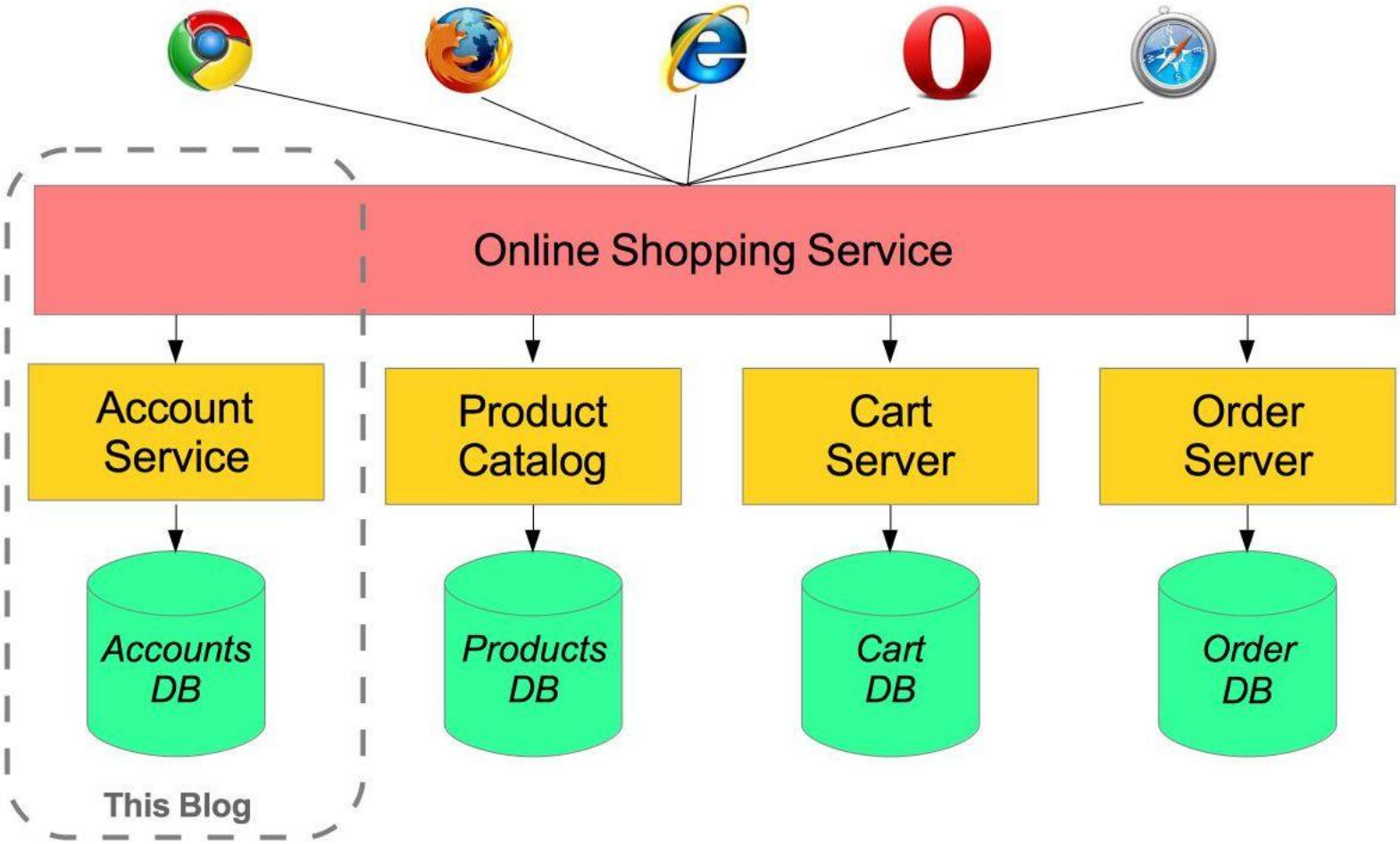


L'architecture Microservices

- **Exemple:** Application de vente de produits.
- Dans une architecture Microservice, Chaque Microservice est parfaitement autonome.
- Chaque Microservice comprte sa (ses) propre(s):
  - base de données,
  - serveur d'application (tomcat, jetty, ...),
  - Librairies, ...
- Ces Microservices sont chacun dans un container Docker, ils sont donc totalement indépendants



L'architecture Microservices



L'architecture Microservices

Java EE

- **Les solutions proposées aux problématiques rencontrées**
  - **Problème n°1 :** comment faire en sorte que les applications proposées en ligne soient toujours disponibles et ne souffrent jamais de coupure ou de ralentissement, quelle que soit l'affluence des utilisateurs sur celles-ci ?
  - **Solution:** Un Microservice peut être dupliquer sur plusieurs serveurs. Le Microservice qui s'occupe de l'interface utilisateur (UI) va appeler une des instances de ce service et recevoir la même réponse, quelle que soit l'instance choisie.

L'architecture Microservices

Java EE

- **Les solutions proposées aux problématiques rencontrées**
  - **Problème n°1 :**
  - **Solution:**
  - Microservices spécialisés dans le dispatching des requêtes vers les différentes instances.
  - Notre application est parfaitement élastique : vous pouvez lui envoyer 5 utilisateurs ou 5 millions, il suffit d'augmenter ou de diminuer le nombre d'instances en services.

L'architecture Microservices

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Problème n°2:** les entreprises se livrent à une "guerre de la mise à jour". comment faire en sorte que l'entreprise soit capable de faire évoluer son application de façon très fréquente et de répondre rapidement aux nouvelles fonctionnalités que propose la concurrence ?
  - **Solution:** il suffira de cibler directement le Microservice responsable de la fonctionnalité en question.

L'architecture Microservices

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Problème n°2:**
  - **Solution:**
  - Si on dispose d'une nouvelle technologie de pricing qui permet de proposer des prix compétitifs en fonction de la concurrence en temps réel, il suffit de modifier le Microservice C et de le redéployer
  - Le risque de bug ou d'indisponibilité de l'application devient très bas, car vous n'avez pas à builder votre application dans un grand WAR puis voir si le nouveau code soit parfaitement compatible et fonctionne correctement une fois déployé.

L'architecture Microservices

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Problème n°3:** Les technologies utilisées pour développer ces applications évoluent très vite et les nouveautés offrent des avantages énormes. Comment les entreprises peuvent-elle s'adapter?
  - **Solution:** Le cloisonnement complet des Microservices permet l'utilisation dans chacun une technologie différente.
  - Le Microservice A peut être en Java alors que le Microservice B est en C++ ou Node.js, l'essentiel étant qu'il expose une API REST.
  - Tirer profit des dernières avancées technologiques sans limite de langage, de framework ou d'environnement.

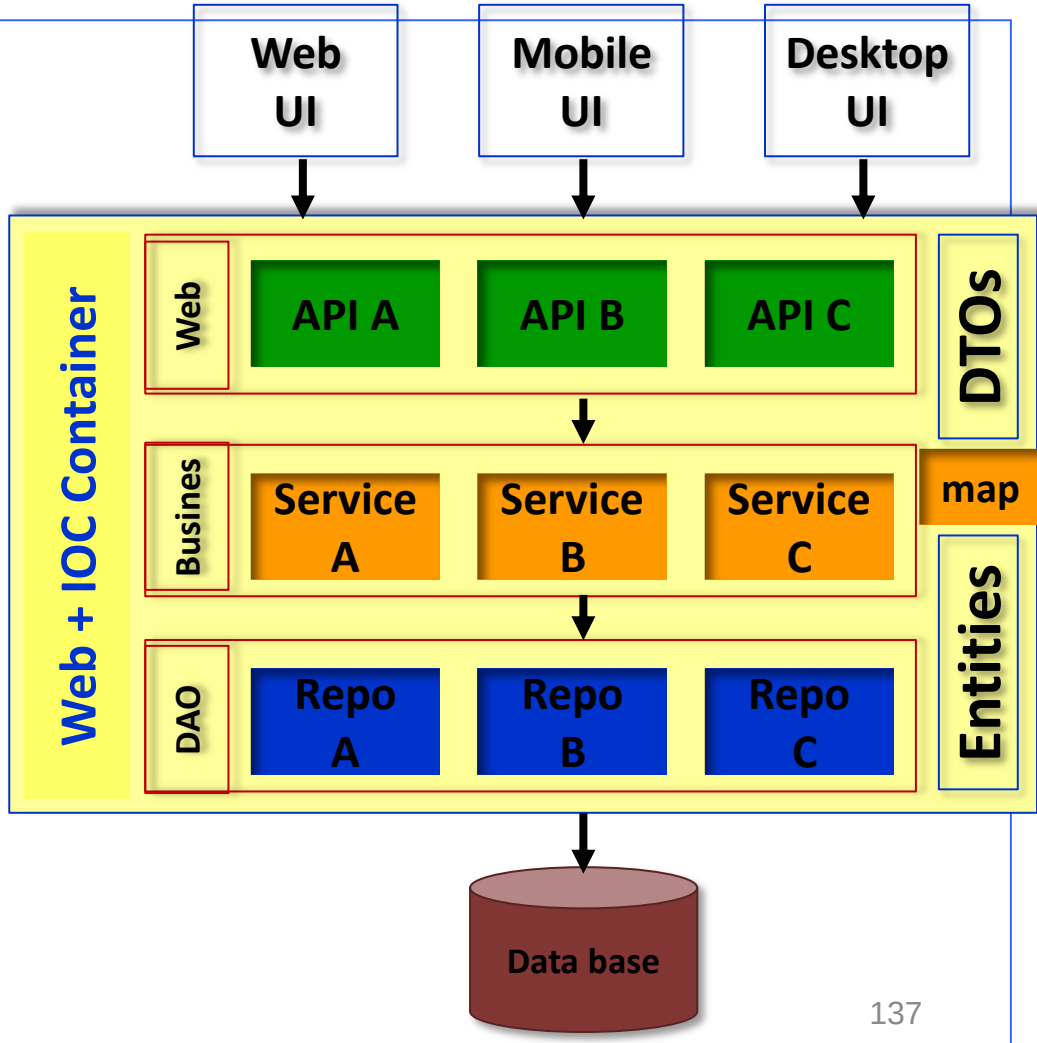
L'architecture Microservices

Java EE

- **Les problématiques rencontrées par les entreprises**
  - **Problème n°4:** La sécurité, qui devient très compliquée à gérer dans les applications complexes, ainsi que les difficultés liées à la coordination entre développeurs. Comment les entreprises peuvent-elle s'en sortir?
  - **Solution:** les équipes de développement deviennent plus autonomes, car ils n'ont plus besoin de consulter toutes les équipes pour concrétiser une idée et vérifier sa compatibilité technique avec ce que font les autres. Les équipes vont pouvoir travailler de façon autonome et aller au bout de leurs projets avec la technologie et les outils de leurs choix. On a donc **un gain de temps et d'innovation**.

L'architecture Monolithique

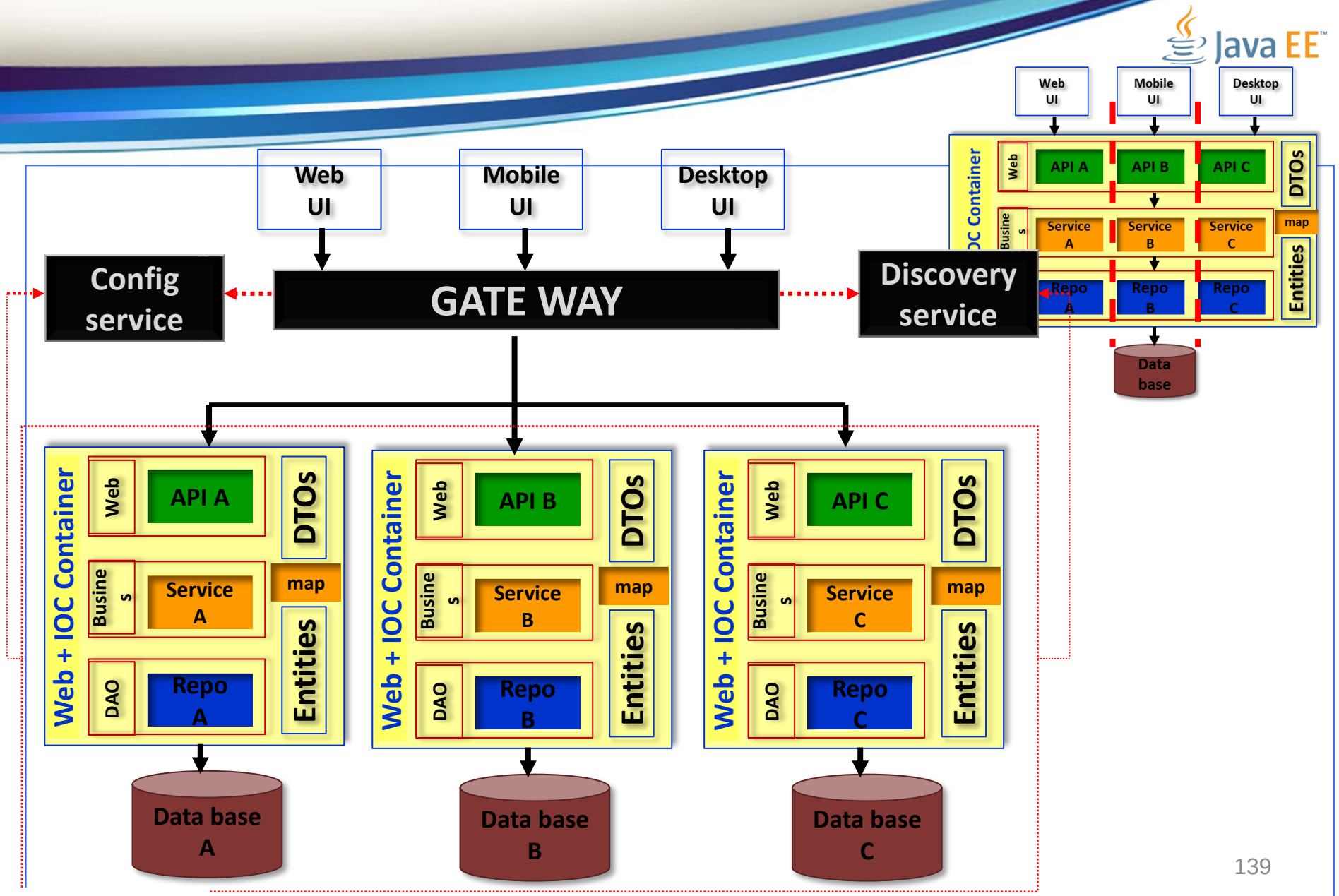
Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application



Contraintes principales des architectures Monolithique

- Elle centralise tous les besoins fonctionnels
- Elle est réalisée dans une seule technologie
- Chaque modification nécessite de:
  - Tester les régressions (influence d'une modification sur le reste de l'app)
  - Redéployer toute l'application
  - Difficile à faire au niveau fonctionnel
- Livraison en bloc (le client attend beaucoup de temps pour commencer à voir les premières versions)
- Mise en production prend beaucoup de temps
- Difficile à tester
- Performance (scalabilité)

L'architecture Micro-services



Les Micro services

- Les micro services sont une approche d'architecture et de développement d'une application composées de petits services.
- L'idée étant de découper un grand problème en petites unités implémentées sous forme de micro-services.
- Chaque service est responsable d'une fonctionnalité.
- Chaque micro-service est développé, testé et déployé séparément des autres.
- Chaque micro-service est développé en utilisant une technologie qui peut être différente des autres. (Java, C++, PHP, Python, …).
- Chaque service tourne dans un processus séparé.
- Utilisant des mécanisme de communication légers (REST).
- La seule relation entre les différents micro-services est l'échange de données effectué à travers les différentes APIs qu'ils exposent.
- Lorsqu'on les combinent, ces micro-services peuvent réaliser des opérations très complexes.

L’architecture Microservices

- **Le développement d’une application autour de l'architecture Microservices en Java exige l’utilisation d’un framework qui facilite le développement d‘une application totalement autonome.**

➔ Le framework Spring Boot.

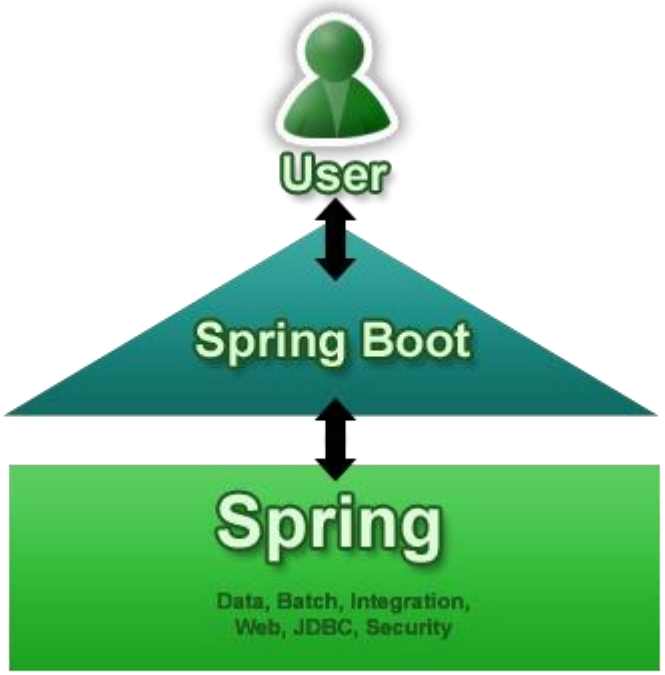
Spring Boot Qu'est-ce que c'est ?

- Une technologie de développement d'application WEB (framework) qui dispense d'écrire du code Java dans les interfaces

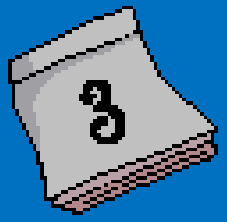
- **Spring Boot** est un framework qui facilite le développement d'applications fondées sur **Spring** en offrant des outils permettant d'obtenir une application packagée en jar, totalement autonome.

Spring Boot Qu'est-ce que c'est ?

- **Spring Boot peut s'expliquer simplement par l'illustration suivante:**



Chapitre



Spring Framework

Plan

- 1. Exigences d'un projet informatique
- 2. Constat
- 3. Architectures distribuées
- 4. Architectures J2EE
- 5. Spring Framework Architecture
- 6. Inversion de contrôle ou injection de dépendances

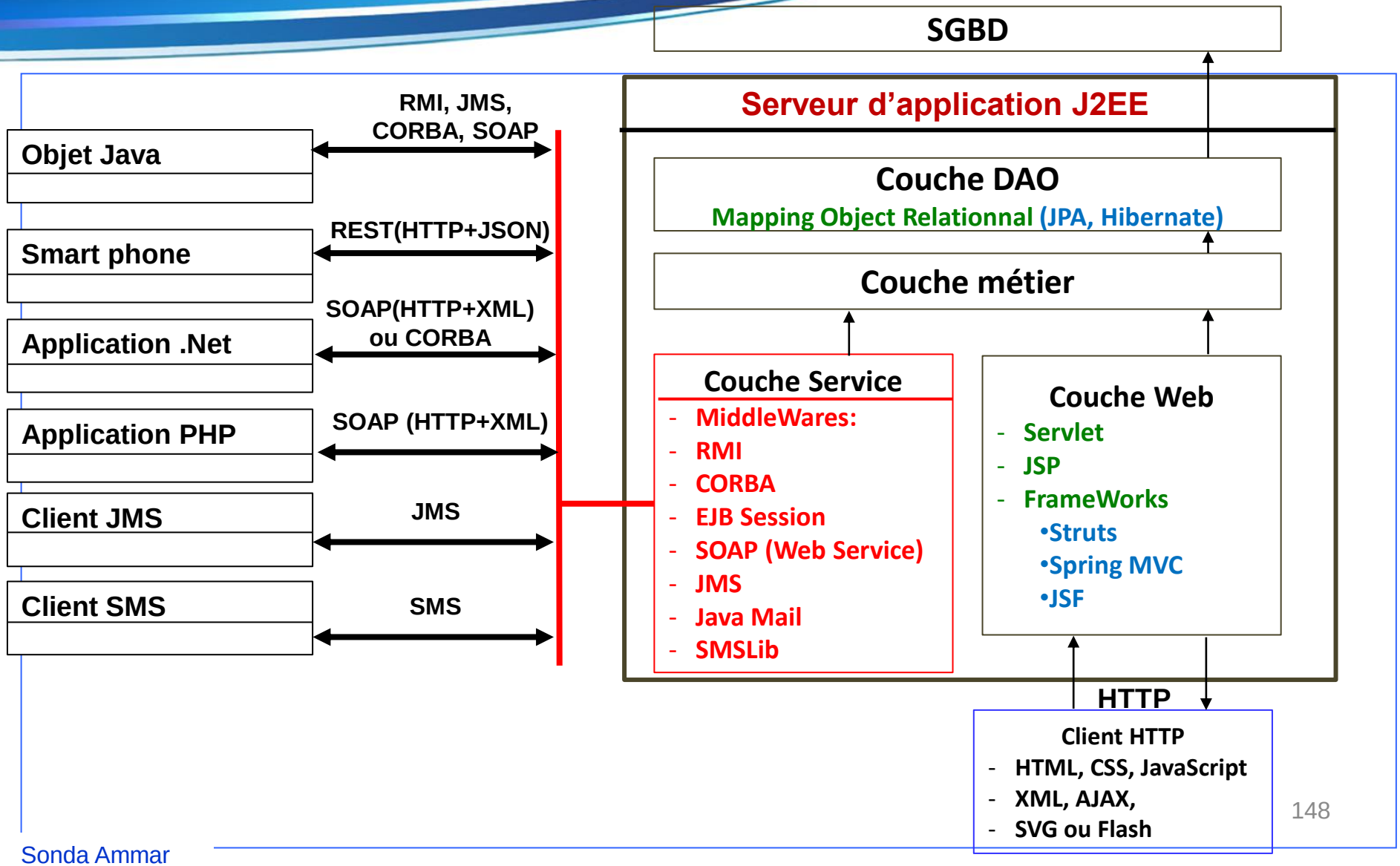
Exigences d'un projet informatique

- Exigences fonctionnelles:
  - Une application est crée pour répondre aux besoins fonctionnels des entreprises,
- Exigences Techniques:
  - Les performances:
    - Temps de réponse
    - Haute disponibilité et tolérance aux pannes,
    - Eviter le problème de montée en chargé.
  - La maintenance:
    - Une application doit évoluer dans le temps,
    - Doit être fermée à la modification et ouverte à l'extension
  - Sécurité
  - Portabilité
  - Distribution
  - Capacité de communiquer avec d'autres applications distantes.
  - Capacité de fournir le service à différents type de clients (Desktop, Mobile, http…)
  - …..

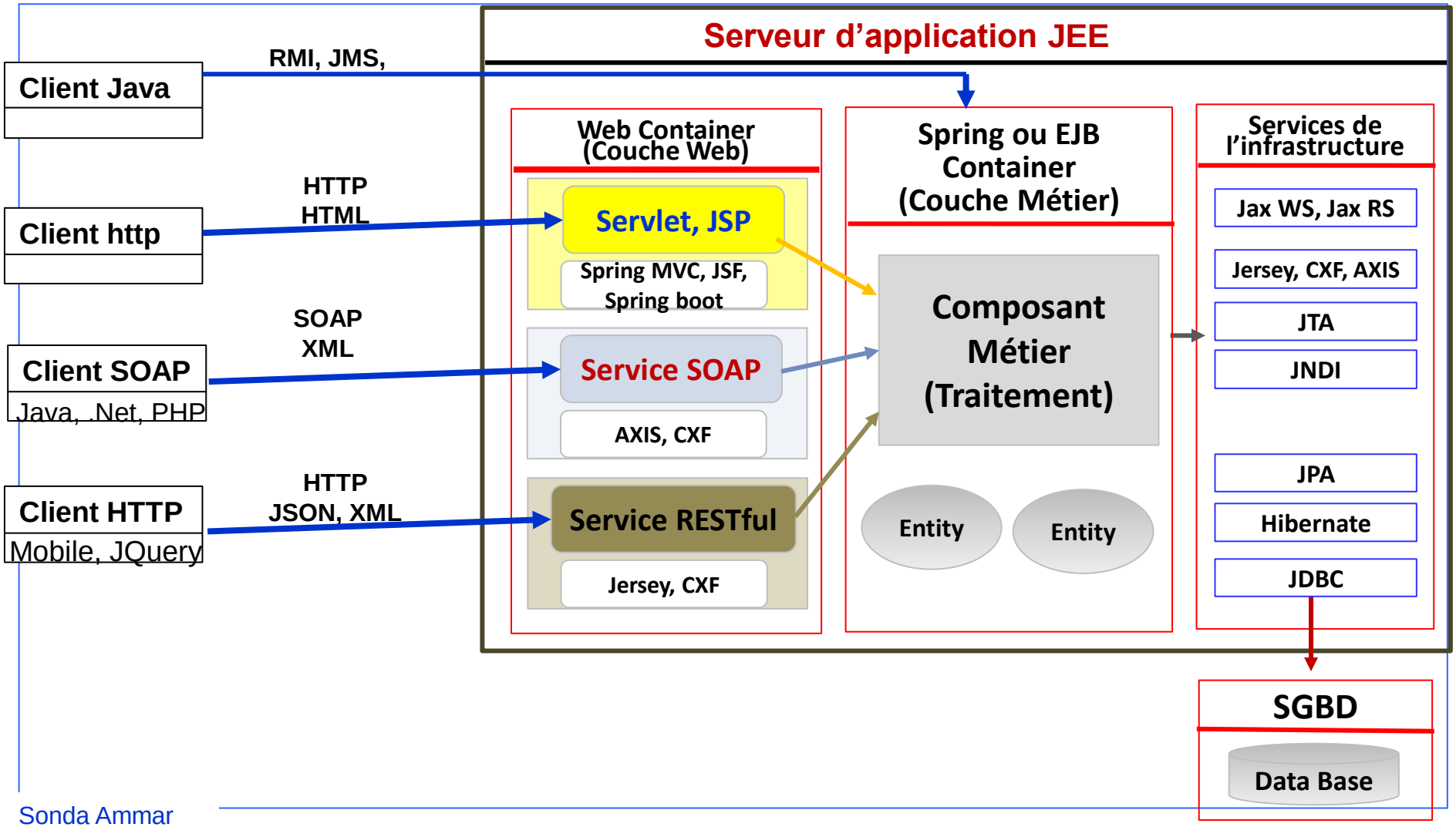
Constat

- Il est très difficile de développer un système logiciel qui respecte ces exigences sans utiliser l'expérience des autres :
  - Serveur d'application JEE:
    - JBOSS, Web Sphere,
    - GlassFish, Tomcat,
    - …
  - **Framework pour l'Inversion de contrôle:**
    - Spring (Conteneur léger)
    - EJB (Conteneur lourd)
  - Frameworks pour le bon fonctionnement de l'application:
    - Mapping objet relationnel (ORM) : JPA, Hibernate, Toplink, …
    - Applications Web : Struts, JSF, SpringMVC
    - …
  - Middlewares (problème de distribution) :
    - RMI, CORBA : Applications distribuées
    - JAXWS pour Web services SOAP
    - JAXRS pour les Web services RESTful
    - JMS : Communication asynchrone entre les application
    - …

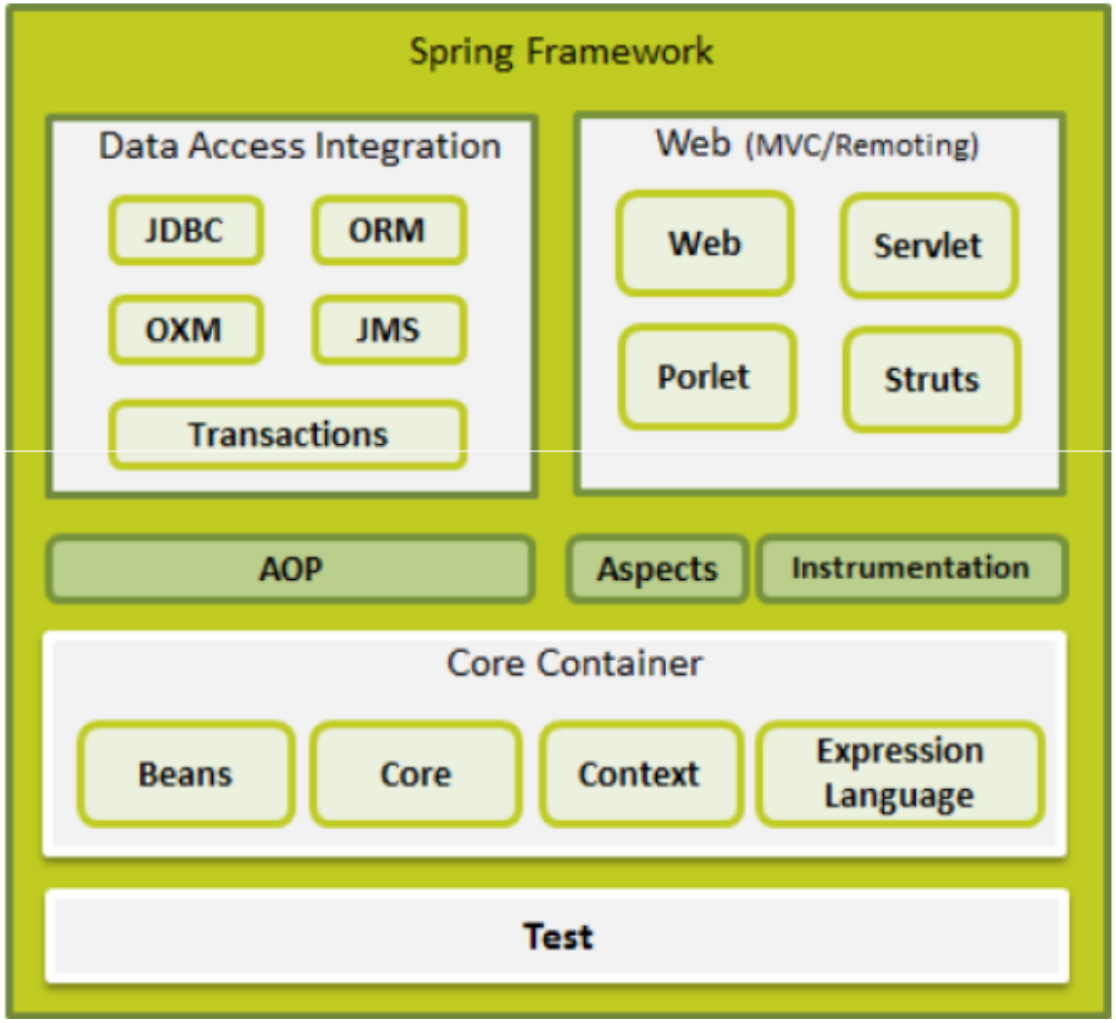
Architectures distribués



Architectures JEE



Spring Framework Architecture



Spring Framework Architecture

- Spring étant un framework pour le développement d'applications Java.
- Spring est modulaire, permettant de choisir les modules appropriés à votre application, sans être obligé d'utiliser le reste.
- Spring Framework fournit plus de 20 modules qui peuvent être utilisé dans les applications.

Spring Framework Architecture

- Spring est un framework open source JEE pour les applications n-tiers, dont il facilite le développement et les tests.
- Il est considéré comme un conteneur dit « léger », c'est-à-dire une infrastructure similaire à un serveur d'application JEE.
- Il prend donc en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets.
- Il permet de séparer le code métier du code technique
- Spring s'appuie principalement sur l'intégration de trois concepts clés :
  - l'inversion de contrôle ou injection de dépendance (IoC).
  - la programmation orientée aspect (AOP).
  - une couche d'abstraction.

- **Limites:** Prenons l'exemple d'une application web avec les dépendances : Hibernate et Spring MVC. Il faut créer les fichiers de configuration suivants : **appconfig-mvc.xml, web.xml, persitence.xml**.
- Il faut consulter la documentation ou inspirer d'un ancien projet. Ensuite écrire le code Java qui permet de lier ces fichiers XML à *ApplicationContext* de Spring.
- Ensuite, adaptez et testez, puis ré-adaptez et re-testez encore pendant un bon moment avant que tout fonctionne.
- Bien sûr, dès que vous faites un changement dans votre application, il ne faut pas oublier de mettre à jour tous les fichiers de configuration, puis débugger de nouveau. Cela devient très vite très fastidieux !

Inversion de contrôle

ou

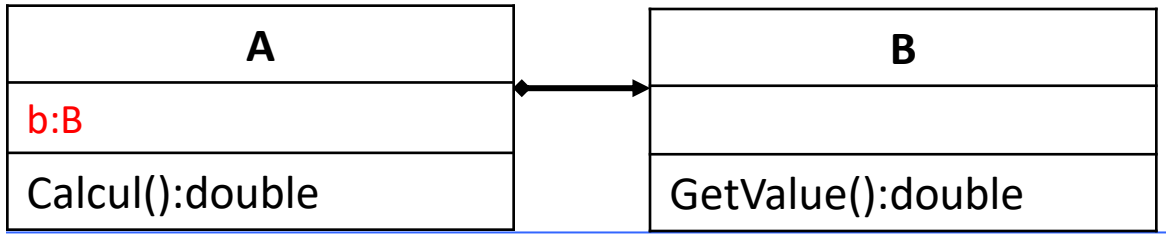
injection de dépendances

Rappels de quelques principes de conception

- Une application qui n'évolue pas meurt.
- Une application doit être **fermée à la modification et ouverte à l'extension**.
- Une application doit s'adapter aux changements
- Efforcez-vous à coupler faiblement vos classes.
- Programmer une interface et non une implémentation
- Etc..

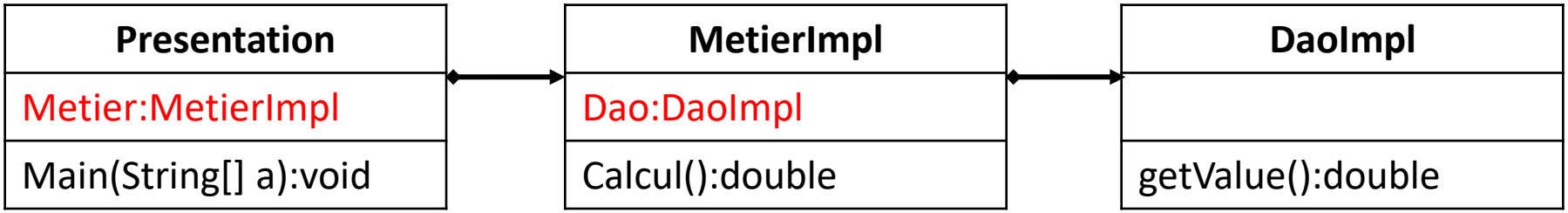
Couplage fort et Couplage faible

- **Quand une classe A est lié à une classe B, on dit que la classe A est fortement couplée à la classe B.**
- **La classe A ne peut fonctionner qu'en présence de la classe B.**
- **Si une nouvelle version de la classe B (soit B2), est crée, on est obligé de modifier dans la classe A.**
- **Modifier une classe implique:**
  - Il faut disposer du code source.
  - Il faut recompiler, déployer et distribuer la nouvelle application aux clients.
  - Ce qui engendre un cauchemar au niveau de la maintenance de l'application



Couplage fort et Couplage faible

Exemple




```
package metier;
import dao.DaoImpl;
public class MetierImpl {
private DaoImpl dao;
public MetierImpl() {
dao=new DaoImpl();
}
public double calcul(){
double nb=dao.getValue();
return 2*nb;
}
}
```

```
package dao;
public class DaoImpl {
public double getValue(){
return(5);}
}
```

```
package pres;
import metier.MetierImpl;
public class Presentation {
private static MetierImpl metier;
public static void main(String[]args) {
metier=new MetierImpl();
System.out.println(metier.calcul());
}
}
```

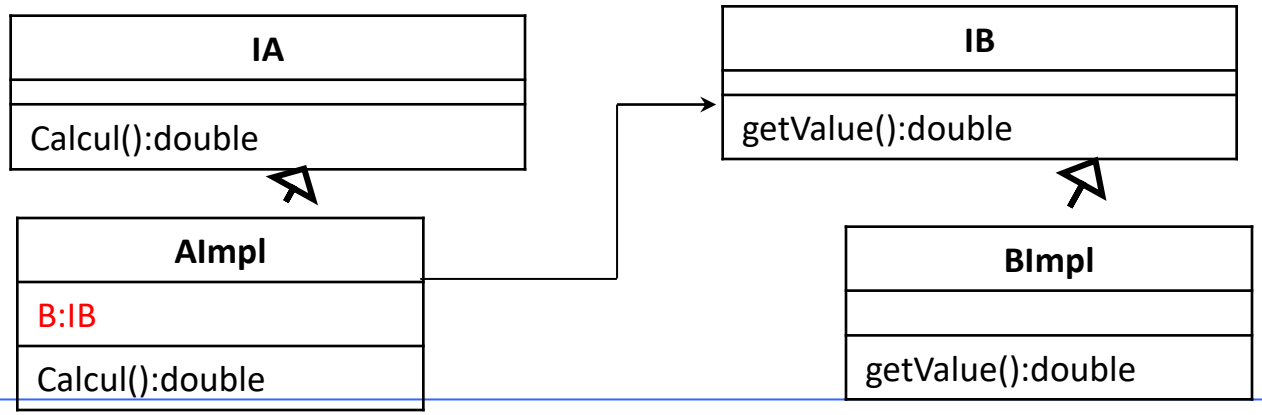
Couplage fort et Couplage faible

Problèmes du couplage fort

- Les classes MetierImpl et DaoImpl sont liées par un couplage fort. De même pour les classe Presentation et MetierImpl
- Ce couplage fort n'a pas empêché de résoudre le problème au niveau fonctionnel. Mais cette conception ne permet pas de créer une application fermée à la modification et ouverte à l'extension.
- La création d'une nouvelle version de la méthode getValue() de la classe DaoImpl, va nous obliger d'éditer le code source de l'application aussi bien au niveau de DaoImpl et aussi MetierImpl.
- De ce fait nous avons violé le principe « une application doit être fermée à la modification et ouverte à l'extension»
- Nous allons voir que nous pourrons faire mieux en utilisant le couplage faible.

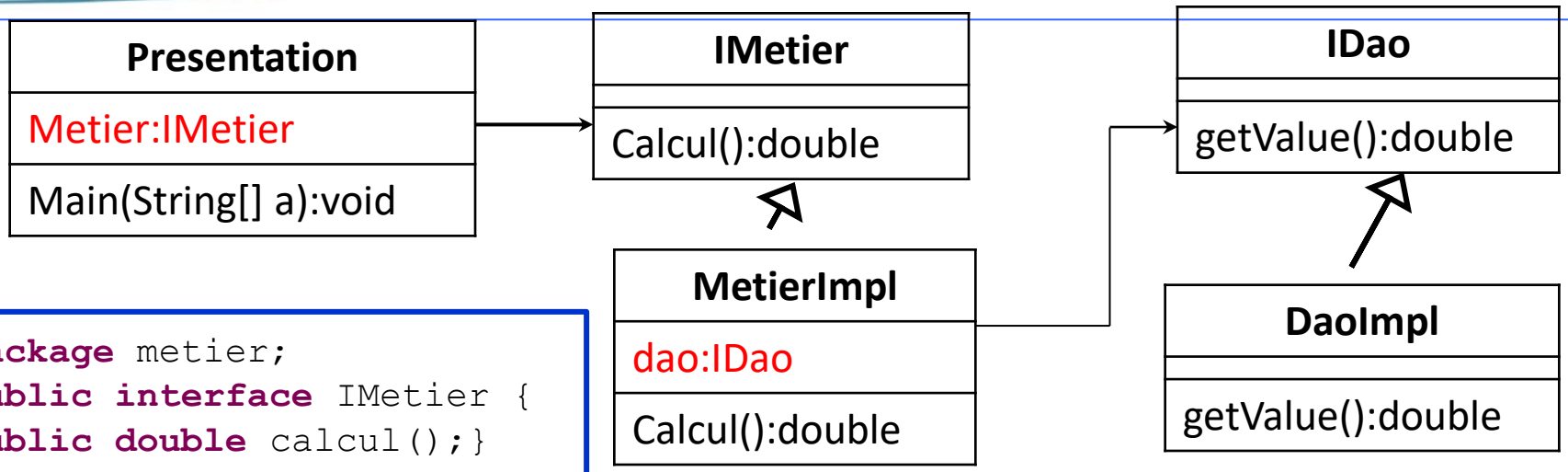
Couplage fort et Couplage faible

- Pour utiliser le couplage faible, nous devons utiliser les interfaces.
- Considérons une classe A qui implémente une interface IA, et une classe B qui implémente une interface IB.
- Si la classe A est liée à l'interface IB par une association, on dit que le classe A et la classe B sont liées par un couplage faible.
- Cela signifie que la classe B peut fonctionner avec n'importe quelle classe qui implémente l'interface IA.
- En effet la classe A ne connait que l'interface IB. De ce fait n'importe quelle classe implémentant cette interface peut être associée à la classe B, sans qu'il soit nécessaire de modifier quoi que se soit dans la classe A.
- Avec le couplage faible, nous pourrons créer des application fermée à la modification et ouvertes à l'extension.



Couplage fort et Couplage faible

Exemple




```
package metier;
public interface IMetier {
public double calcul();}
```

```
package metier;
import dao.IDao;
public class MetierImpl
implements IMetier {
private IDao dao=null;
public double setDao(Idao dao) {
This.dao=dao;}
public double calcul() {
double nb=dao.getValue();
return 2*nb; }
```

```
package dao;
public interface IDao {
public double getValue();
}
```

```
package dao;
public class DaoImpl implements IDao {
public double getValue() {
return 5;
}
}
```

Injection des dépendances

Package Explorer

- ClientWeb
- ClientWS
- ioc
- tpIO
  - src/main/java
    - dao
      - IDao.java
  - src/main/resources
  - src/test/java
  - src/test/resources
  - JRE System Library [J2SE-1.5]
  - src
  - target
  - pom.xml

1

*IDao.java

```
package dao;

public interface IDao {
  public double getValue();
}
```

Package Explorer

- ClientWeb
- ClientWS
- ioc
- tpIO
  - src/main/java
    - dao
      - DaoImpl.java
      - IDao.java
  - src/main/resources
  - src/test/java
  - src/test/resources
  - JRE System Library [J2SE-1.5]
  - src
  - target
  - pom.xml

2

IDao.java | *DaoImpl.java

```
package dao;

public class DaoImpl implements IDao {

    public double getValue() {
        // TODO Auto-generated method stub
        return 5;
    }

}
```

Injection des dépendances

Package Explorer

- ClientWeb
- ClientWS
- ioc
- tpIO
  - src/main/java
    - dao
      - DaoImpl.java
      - IDao.java
    - metier
      - IMetier.java
  - src/main/resources
  - src/test/java
  - src/test/resources
  - JRE System Library [J2SE-1.5]
  - src
  - target

3

IDao.java | DaoImpl.java | IMetier.java

```java
package metier;

public interface IMetier {
  public double calcul();
}
```

IDao.java | DaoImpl.java | IMetier.java | *MetierImpl.java | Presentation.java

```java
package metier;

import dao.IDao;

public class MetierImpl implements IMetier {
    private IDao dao=null;

    public void setDao(IDao dao) {
        this.dao = dao;
    }

    public double calcul() {
        double nb=dao.getValue();
        return nb*6;
    }

}
```

Injection des dépendances

4

Injection des dépendances

Java EE

- **Injection par <u>instanciation statique</u>:**

```
import metier.MetierImpl;
import dao.DaoImpl;
public class Presentation {
public static void main(String[] args) {
DaoImpl dao=new DaoImpl();
MetierImpl metier=new MetierImpl();
metier.setDao(dao);
System.out.println(metier.calcul());
}
}
```

Injection des dépendances

Injection par instanciation dynamique par réflexion:

- Fichier texte de configuration : config.txt

  ext.DaoImp

  metier.MetierImpl

```
dao.DaoImpl
metier.MetierImpl
```

```
import java.io.*;import java.lang.ref
import java.util.Scanner; import metier.IMetier;
import dao.IDao;
public class Presentation {
public static void main(String[] args) {
try {
     Scanner scanner=new Scanner(new File("config.txt"));
     String daoClassname=scanner.next();
     String metierClassName=scanner.next();
     Class cdao=Class.forName(daoClassname);
     IDao dao= (IDao) cdao.newInstance();
     Class cmetier=Class.forName(metierClassName);
     IMetier metier=(IMetier) cmetier.newInstance();
     Method meth=cmetier.getMethod("setDao",new
        Class[]{IDao.class});
     meth.invoke(metier, new Object[]{dao});
     System.out.println(metier.calcul());
} catch (Exception e) { e.printStackTrace(); }
}  }
```

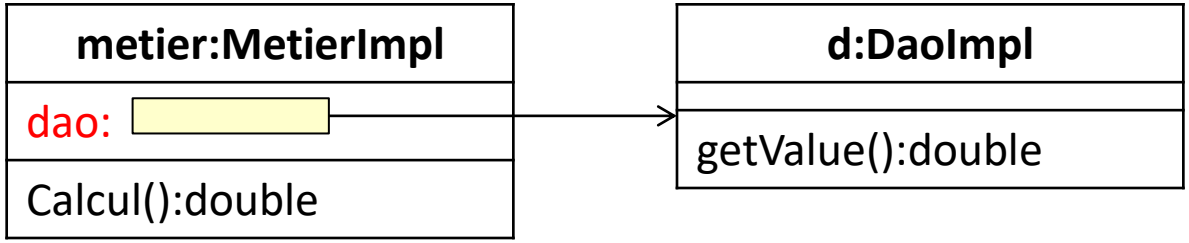
```
<terminated> Presentation (1) [Java Application] C
30.0
```

Injection des dépendances avec Spring

- L'injection des dépendance, ou l'inversion de contrôle est un concept qui intervient généralement au début de l'exécution de l'application.
- Spring IOC commence par lire un fichier XML qui déclare quelles sont les différentes classes à instancier et d'assurer les dépendances entre les différentes instances.
- Quand on a besoin d'intégrer une nouvelle implémentation à une application, il suffirait de la déclarer dans le fichier xml de beans spring.

Injection des dépendances dans une application java standard

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-
2.0.dtd" >
<beans>
<bean id="d" class="dao.DaoImpl"></bean>
<bean id="metier" class="metier.MetierImpl">
       <property name="dao" ref="d"></property>
</bean>
</beans>
```



Injection des dépendances dans une application java standard

```
package pres;
import metier.IMetier;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
public class PresentationSpring {
public static void main(String[] args) {
ClassPathXmlApplicationContext context=new
ClassPathXmlApplicationContext("config.xml");
IMetier metier=(IMetier) context.getBean("metier");
System.out.println(metier.calcul());
}
}
```

Exemple: sans utiliser d'inversion de contrôle

```
public void virement (int c1, int c2, double mt) {
  /*  Création d'une transaction  */
EntityTransaction transaction = entityManager.getTransaction();
  /*  Démarrer la transaction  */
transaction.begin();
Try {
```

Code Technique

```
      /*  Code métier  */
    Retirer(c1, mt);
    Verser(c2, mt);
```

Code Métier

```
  /*  Valider la transaction  */
transaction.commit();
} catch (Exception e) {
  /*  Annuler la transaction  en cas d'exception  */
transaction.rollback();
e.printStackTrace(); }
}
```

Code Technique

Exemple: utilisant l'inversion de contrôle

```
@Transactional
public void virement (int c1, int c2, double mt) {
    Retirer(c1, mt);
    Verser(c2, mt);
}
```

Code Métier

Avec l'annotation @Transactional, nous avons délégué la gestion des transactions au conteneur Spring IOC.

Maven dependencies

```
<dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.2.2.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>3.2.2.RELEASE</version>
        </dependency>
</dependencies>
```

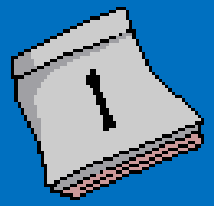
Injection des dépendances dans une application web

- **Dans une application web, SpringIOC est appelé au démarrage du serveur en déclarant le listener ContextLoaderListener dans le fichier web.xml:**

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-beans.xml</param-value>
</context-param>
<listener>
        <listener-class>
        org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
```

- **Dans cette déclaration, ContextLoaderListener est appelé par Tomcat au moment du démarrage de l'application. Ce listener cherchera le fichier de beans spring « spring-beans.xml » stocké dans le dossier WEB-INF ce qui permet de faire l'injection des dépendances entre MetierImpl et DaoImpl**

Chapitre



Spring Boot

Plan

- 1. Spring Boot Qu’est-ce que c’est ?
- 2. L'auto-configuration de Spring Boot
- 3. Les starters de Spring Boot
- 4. Créez un Microservice grâce à Spring Boot

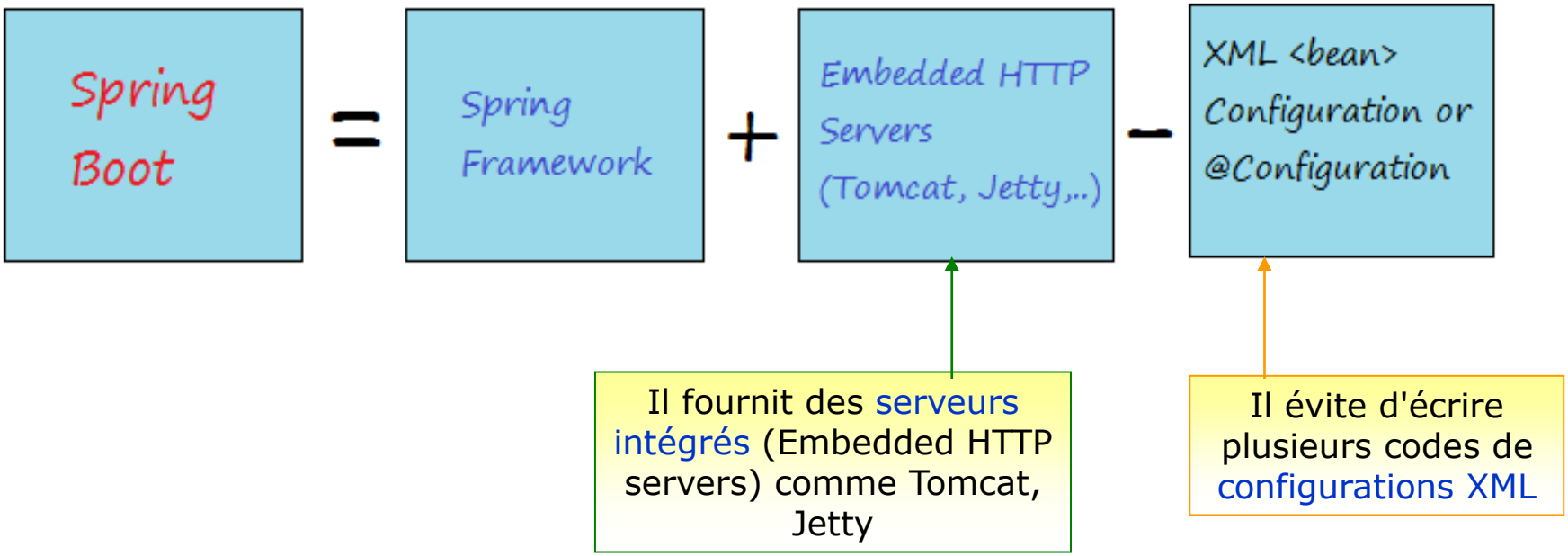
Spring Boot Qu'est-ce que c'est ?

- Une technologie de développement d'application WEB (framework) qui dispense d'écrire du code Java dans les interfaces

- **Spring Boot** est un framework qui facilite le développement d'applications fondées sur **Spring** en offrant des outils permettant d'obtenir une application packagée en jar, totalement autonome.

Spring Boot Qu'est-ce que c'est ?

- **Spring Boot peut s'expliquer simplement par l'illustration suivante:**



Comment Spring Boot permet-il de créer une application sans écrire une seule ligne de configuration, ou presque ?

Spring Boot Qu'est-ce que c'est ?

- Votre application est bardée de fichiers XML qui indiquent les configurations des servlets, des vues, des contenus statiques, etc. Ces fichiers de configuration deviennent un vrai challenge lorsque vous avez une application complexe.
- Spring est un formidable outil quand on essaye de développer une application web.
- Néanmoins, un de leur plus gros problèmes est la configuration

Comment Spring Boot permet-il de créer une application sans écrire une seule ligne de configuration, ou presque ?

Spring Boot Qu'est-ce que c'est ?

Java EE

Les avantages de Spring Boot sont:

- Il est facile de développer l'application basée sur Spring avec Java ou Groovy.
- Il diminue énormément du temps et augmente la productivité.
- Il évite d'écrire plusieurs de codes d'expressions standard (boilerplate Code), des Annotations et des configurations XML.
- Il est très facile d' intégrer des applications Spring Boot avec ses écosystème de Spring comme Spring JDBC, Spring ORM, Spring Data, Spring Security, etc.
- Il suis l'approche "Configuration par défaut" afin de diminuer le temps et l'effort de développement.

Spring Boot Qu'est-ce que c'est ?

Java EE

Les avantages de Spring Boot sont:

- Il fournit des serveurs intégrés (Embedded HTTP servers) comme Tomcat, Jetty .... afin de développer et de tester des applications web à la manière la plus facilement.
- Il fournit l'outil CLI (Command Line Interface) afin de développer et de tester des applications Spring Boot à partir de l'invite de commande (command prompt) facilement et rapidement.
- Il fournit beaucoup de plugins afin de développer et de tester des applications Spring Boot rapidement en utilisant les outils de Build comme Maven et Gradle
- Il fournit beaucoup de plugins afin de travailler avec des serveurs intégrés [illegible] base de données stockées dans la mémoire facilement.

Spring Boot Qu'est-ce que c'est ?

- **Spring Boot propose 2 fonctionnalités principales pour simplifier la configuration:**
  - L'auto-configuration,
  - Les starters

L'auto-configuration de Spring Boot

- **Cette fonctionnalité est la plus importante de Spring Boot.**
- **Elle permet de configurer automatiquement l'application à partir des jar trouvés dans le Classpath.**
  - Exemple: si des dépendances (Spring MVC, Hibernate, …) sont importés, Spring Boot consulte cette liste puis il produit la configuration nécessaire (appconfig-mvc.xml, web.xml, persitence.xml) pour que tout fonctionne correctement.
  - Voici la syntaxe de la configuration automatique avec Spring boot:

```
@EnableAutoConfiguration
```

Les starters de Spring Boot

- **Les starters permettent de créer un "squelette" prêt à l'emploi très rapidement. Les starters font gagner énormément de temps, notamment lorsqu'on commence le développement d'un Microservice.**
- **Les starters permettent d'importer un ensemble de dépendances selon la nature de l'application à développer afin de démarrer rapidement.**
- **Les starters permettent la gestion des versions.**
  - On n'a plus besoin de chercher quelles versions sont compatibles puis de les ajouter une à une dans le pom.xml,
  - Il suffit d'ajouter une simple dépendance au starter de votre choix. Cette dépendance va ajouter, à son tour, les éléments dont elle dépend, avec les versions.

Les starters de Spring Boot

- **Exemple: Pour créer un Microservice, on a besoin des dépendances suivantes:**
  - Spring ;
  - Spring MVC ;
  - Jackson (pour json) ;
  - Tomcat ;
  - ...
- **Avec Spring Boot, vous allez tout simplement avoir une seule dépendance dans votre pom.xml :**

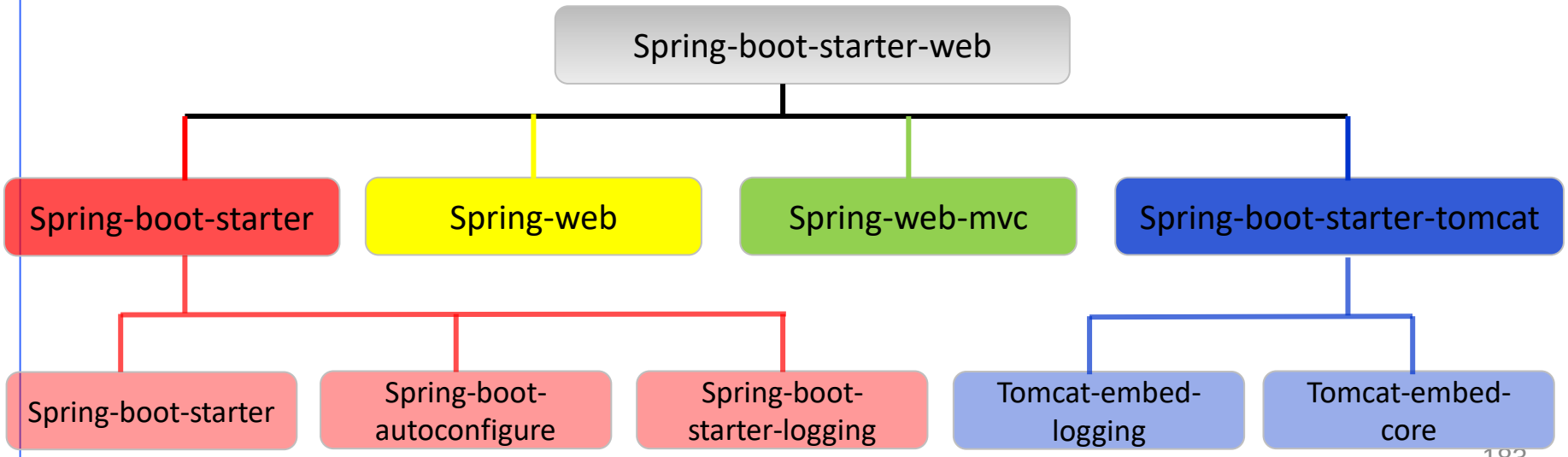
```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Tous les starters de Spring Boot sont au format : *spring-boot-starter-**nomdustarter***

Les starters de Spring Boot

- **Avec Spring Boot, vous allez tout simplement avoir une seule dépendance dans votre pom.xml :**

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```



Les starters de Spring Boot

- Ce starter va charger les dépendances présentes dans le pom suivant :

```
<?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/PO
M/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
   <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starters</artifactId>
    <version>1.5.9.RELEASE</version>
  </parent>
  <artifactId>spring-boot-starter-web</artifactId>
  <name>Spring Boot Web Starter</name>
```

```
<description>Starter for building web, including
RESTful, applications using Spring
     MVC. Uses Tomcat as the default embedded
container</description>
  <url>http://projects.spring.io/spring-boot/</url>
  <organization>
    <name>Pivotal Software, Inc.</name>
    <url>http://www.spring.io</url>
  </organization>
  <properties>
    <main.basedir>${basedir}/../..</main.basedir>
  </properties>
```

Les starters de Spring Boot

- Ce starter va charger les dépendances présentes dans le pom suivant :

```
<dependencies>
    <dependency>
<groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
<groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
tomcat</artifactId>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
    </dependency>
```

```
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
    </dependency>
  </dependencies>
</project>
```

Les starters de Spring Boot

Java EE

- **comment il devine les versions ?**

```
<?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/PO
M/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
   <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starters</artifactId>
    <version>1.5.9.RELEASE</version>
  </parent>
  <artifactId>spring-boot-starter-web</artifactId>
  <name>Spring Boot Web Starter</name>
```

Il permet de définir principalement :

- La version de Java à utiliser.
- Une liste complète des versions des dépendances prises en charge, ce qui permet d'ajouter des dépendances sans indiquer leur version comme dans le pom.xml du starter spring-boot-starter-web vu précédemment.

Les starters de Spring Boot

- **Spring-boot-starter-web n'est pas le seul starter disponible. Selon ce que vous comptez développer, vous trouverez pratiquement toujours un starter adéquat. Voici quelques exemples :**
  - spring-boot-starter-mail: pour les applications et services d'envoi de mails.
  - spring-boot-starter-thymeleaf: si vous souhaitez créer une application qui offre une interface utilisateur en utilisant le moteur de template thymeleaf.
  - spring-boot-starter-web-services: pour les services plus classiques utilisant SOAP (Simple Object Access Protocol).

Créez un Microservice grâce à Spring Boot

- **Spring initializr vous permet de composer votre application selon vos besoins.**

https://start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.1.0

Project Metadata

Artifact coordinates

Group

com.example

Artifact

demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Generate Project alt + ⏎

Don't know what to look for? Want more options? Switch to the full version.

Créez un Microservice grâce à Spring Boot

- **Tous les starters et ensembles de dépendances pour les cas d'usage les plus courants (starter web, bases de données, les moteurs de template, la sécurité, le messaging et autres outils très importants dans une architecture Microservices) seront affichés.**

Core

- DevTools — Spring Boot Development Tools
- Security — Secure your application via spring-security
- Lombok — Java annotation library which helps to reduce boilerplate code and code faster
- Configuration Processor — Generate metadata for your custom configuration keys
- Session — API and implementations for managing a user's session information
- Cache — Spring's Cache abstraction
- Validation — JSR-303 validation infrastructure (already included with web)
- Retry — Provide declarative retry support via spring-retry

Web

- Web — Full-stack web development with Tomcat and Spring MVC
- Reactive Web — Reactive web development with Netty and Spring WebFlux
- Rest Repositories — Exposing Spring Data repositories over REST via spring-data-rest-webmvc
- Rest Repositories HAL Browser — Browsing Spring Data REST repositories in your browser
- HATEOAS — HATEOAS-based RESTful services
- Web Services — Contract-first SOAP
- Jersey (JAX-RS) — RESTful Web Servic
- Websocket — Websocket develop

Template Engines

- Thymeleaf — Thymeleaf templating engine
- Freemarker — FreeMarker templating engine
- Mustache — Mustache templating engine
- Groovy Templates — Groovy templating engine

NoSQL

- Redis — Redis key-value data store, including spring-data-redis
- Reactive Redis — Redis key-value data store, including spring-data-redis
- MongoDB — MongoDB NoSQL Database, including spring-data-mongodb
- Reactive MongoDB — MongoDB NoSQL Database, including spring-data-mongodb and the reactive driver
- Embedded MongoDB — Embedded MongoDB for testing
- Elasticsearch — Elasticsearch search and analytics engine including spring-data-elasticsearch
- Solr — Apache Solr search platform, including spring-data-solr
- Cassandra — Cassandra NoSQL Database, including spring-data-cassandra
- Reactive Cassandra — Cassandra NoSQL Database, including spring-data-cassandra and the reactive driver
- Couchbase

Integration

- Spring Integration — Common spring-integration modules
- RabbitMQ — Advanced Message Queuing Protocol via spring-rabbit
- Kafka — Kafka messaging support using Spring Kafka
- Kafka Streams — Support for building stream processing applications with Apache Kafka Streams
- JMS (ActiveMQ) — Java Message Service API via Apache ActiveMQ
- JMS (Artemis) — Java Message Service API via Apache Artemis

Cloud Core

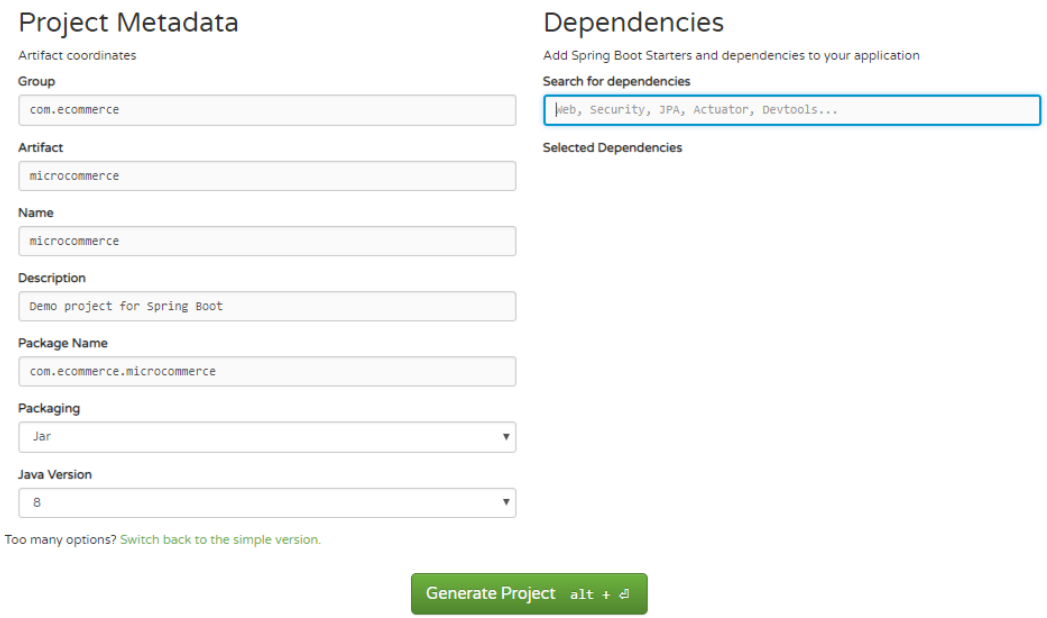
- Cloud Bootstrap — spring-cloud-context (e.g. Bootstrap context and @RefreshScope)
- Cloud Function — Functions as Spring Beans
- Cloud Security — Secure load balancing and routing with spring-cloud-security

SQL

- JPA — Java Persistence API including spring-data-jpa, spring-orm and Hibernate
- MySQL — MySQL JDBC driver
- H2 — H2 database (with embedded support)
- JDBC — JDBC databases
- MyBatis — Persistence support using MyBatis
- PostgreSQL — PostgreSQL JDBC driver
- SQL Server — Microsoft SQL Server JDBC driver
- HSQLDB — HSQLDB database (with embedded support)
- Apache Derby — Apache Derby database (with embedded support)

Créez un Microservice grâce à Spring Boot

- Développer un mini-système d'e-commerce fondé sur l'architecture Microservices avec Spring Boot.
- Un Microservice qui gère les produits à vendre (ajouter, supprimer, mettre à jour et afficher les produits).
- Création et importation à partir de Spring Initializr:
  - Les Metadata du projet:
    - Group : com.ecommerce
    - Artifact : microcommerce
    - Name : microcommerce
    - Packaging : jar
    - Java Version : 8



Créez un Microservice grâce à Spring Boot

Java EE

Les étapes à suivre:

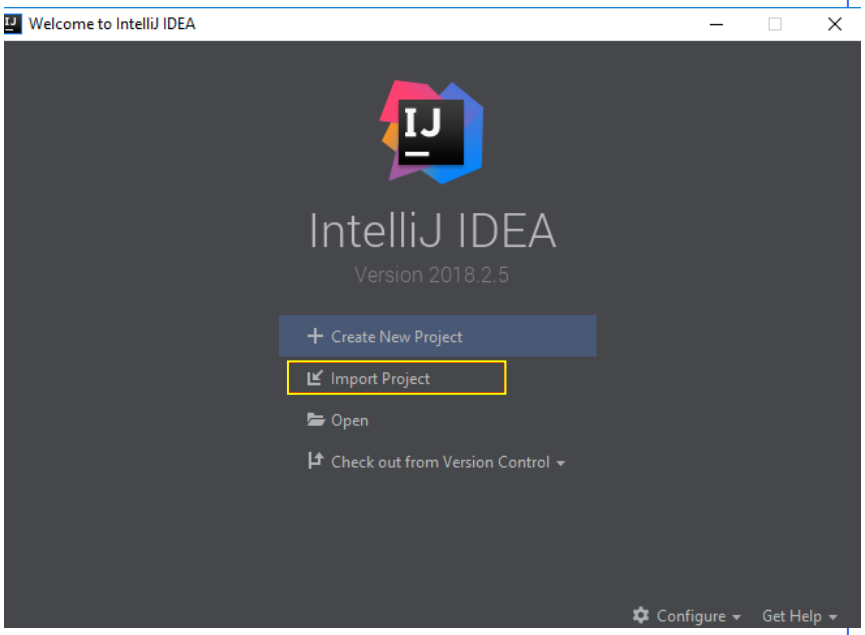
1. Sélectionnez le starter Web

2. Cliquez sur "Generate Project" et téléchargez l'application générée.
3. Dans ce cours, nous allons utiliser comme IDE : IntelliJ.
4. Procédez à l'extraction de l'application téléchargée "microcommerce.zip".

Java EE

Les étapes à suivre:

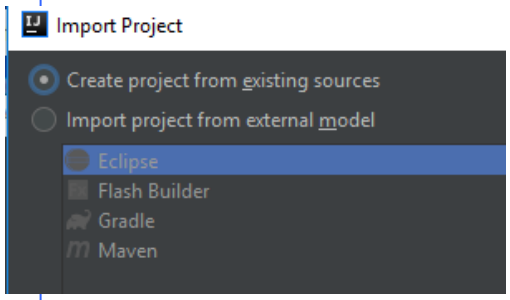
5. Une fois sur la page d'accueil d'IntelliJ, cliquez sur "Import Project":

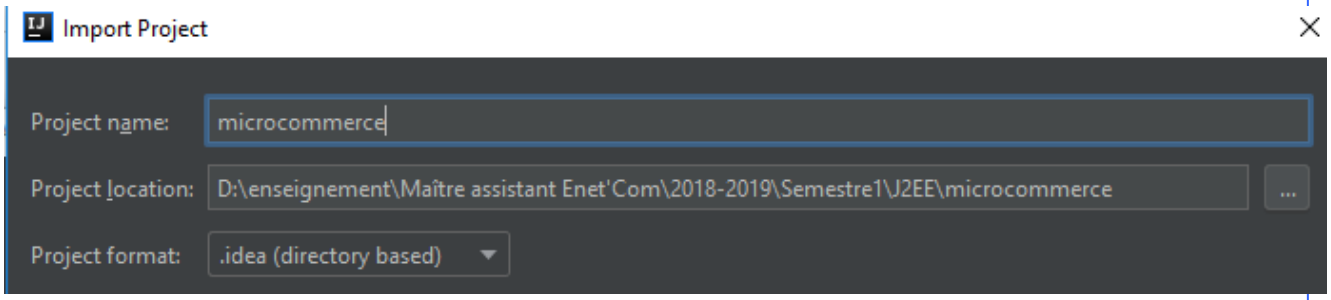


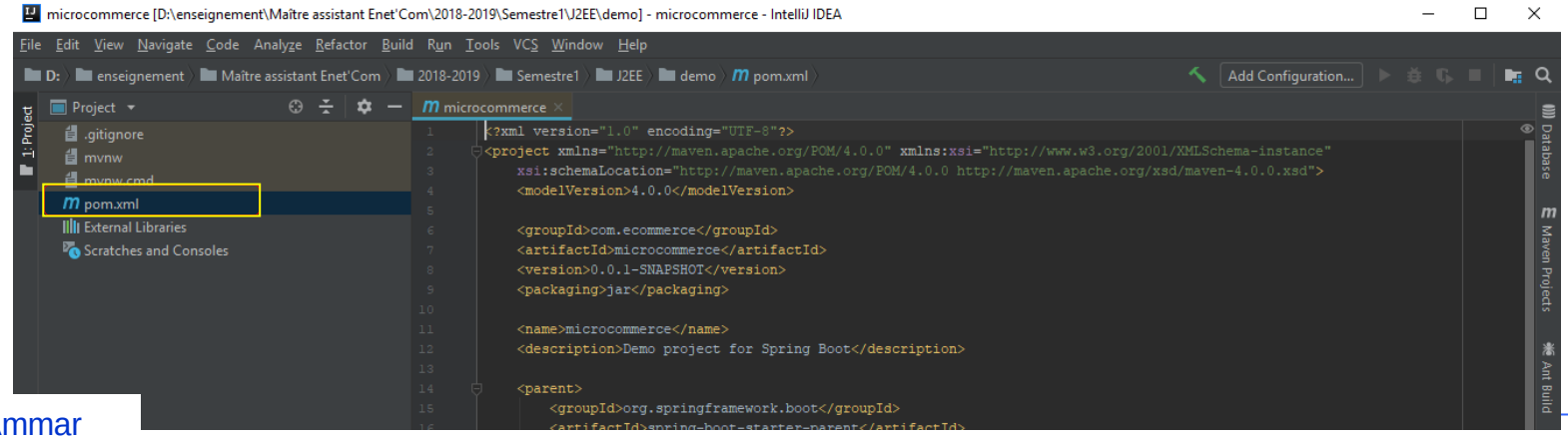
Créez un Microservice grâce à Spring Boot

Les étapes à suivre:

6. Sélectionnez le dossier de l'application microcommerce puis sélectionnez le pom.xml :





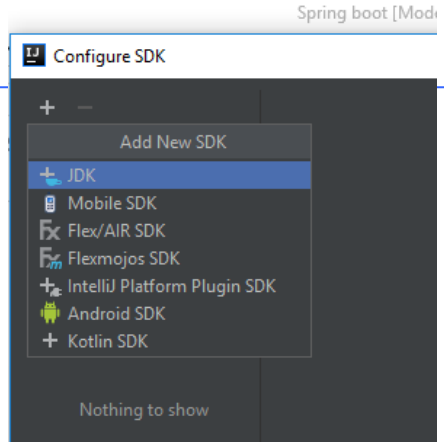


Créez un Microservice grâce à Spring Boot

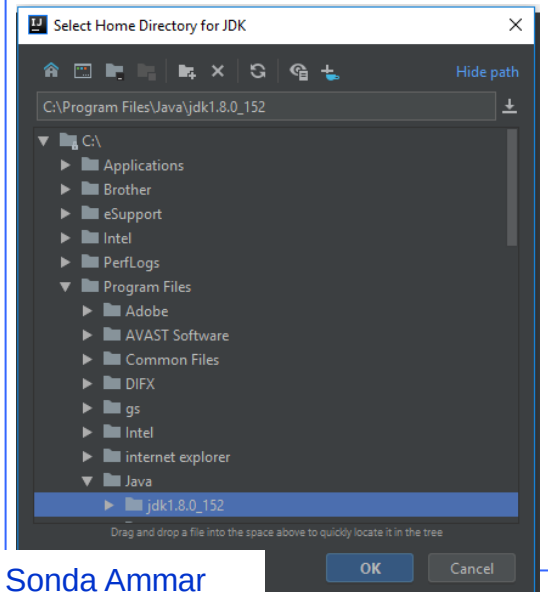
Les étapes à suivre:

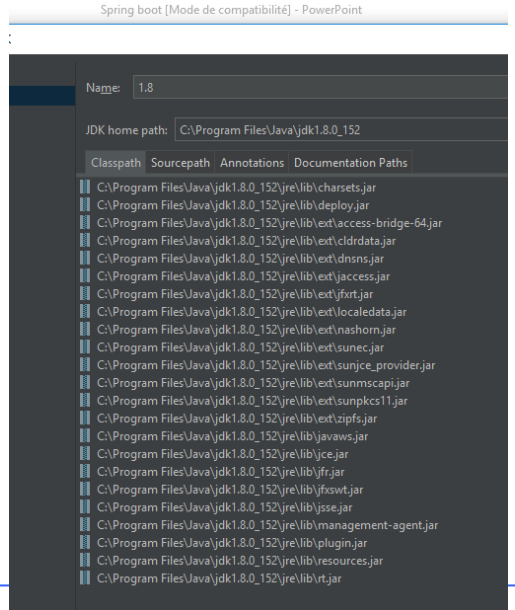
7. Ajouter SDK:

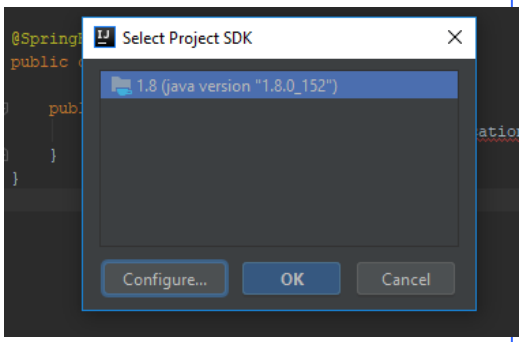
- Si vous venez d'installer IntelliJ, il vous demandera de choisir le JDK : cliquez sur "+" puis JDK :



- Sélectionnez le répertoire de votre JDK :







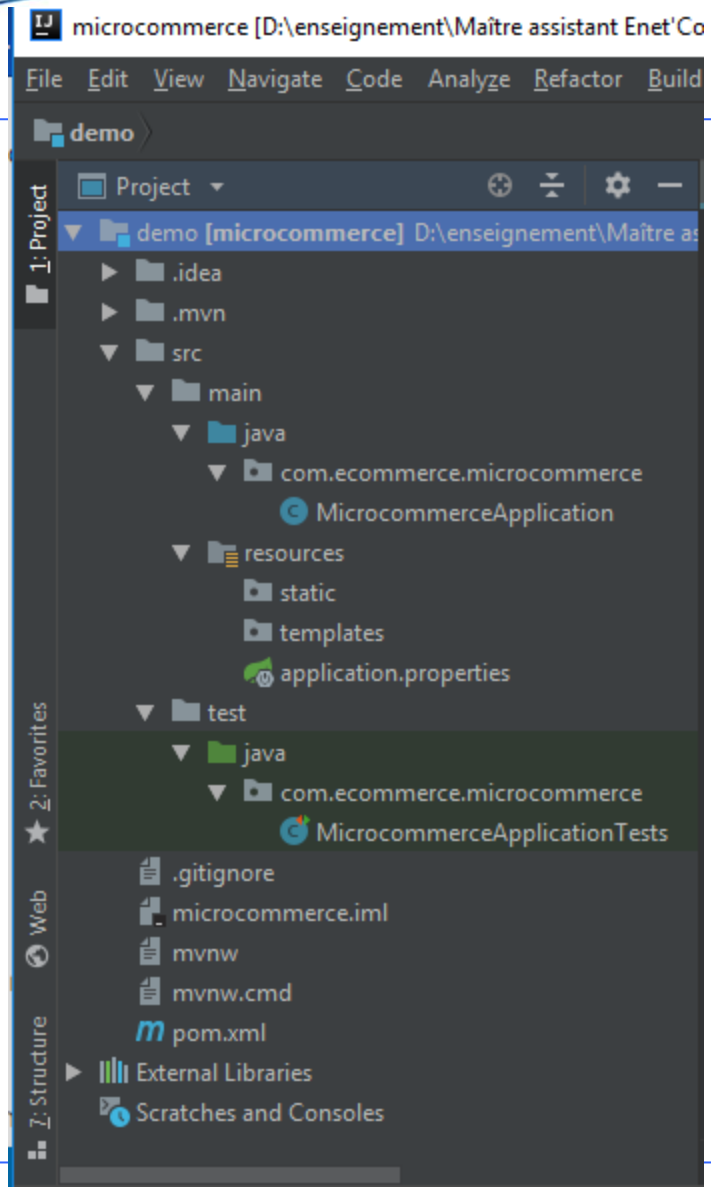
Créez un Microservice grâce à Spring Boot

Java EE

Les étapes à suivre:

8. Analyse du code obtenu

- Si tout s'est déroulé correctement, vous devriez avoir cette arborescence :



Créez un Microservice grâce à Spring Boot

Java EE

Les étapes à suivre:

9. Etudier en détail les différents éléments de cette arborescence

- Pom.xml

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

pom.xml hérite du parent spring-boot-starter-parent qui nous permet de ne plus nous soucier des versions des dépendances et de leur compatibilité.

Créez un Microservice grâce à Spring Boot

Java EE

Les étapes à suivre:

9. Etudier en détail les différents éléments de cette arborescence

- Pom.xml

```
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>
```

Nous avons sélectionné Java 8 dans l'interface de Spring Initializr, une balise XML *<java.version>* a été ajoutée dans la partie properties du fichier pom.xml.

Créez un Microservice grâce à Spring Boot

Java EE

Les étapes à suivre:

9. Etudier en détail les différents éléments de cette arborescence

- Pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Le pom.xml définit ensuite une dépendance vers le starter qui va ajouter à notre Microservice toutes les dépendances de base nécessaires pour démarrer rapidement.

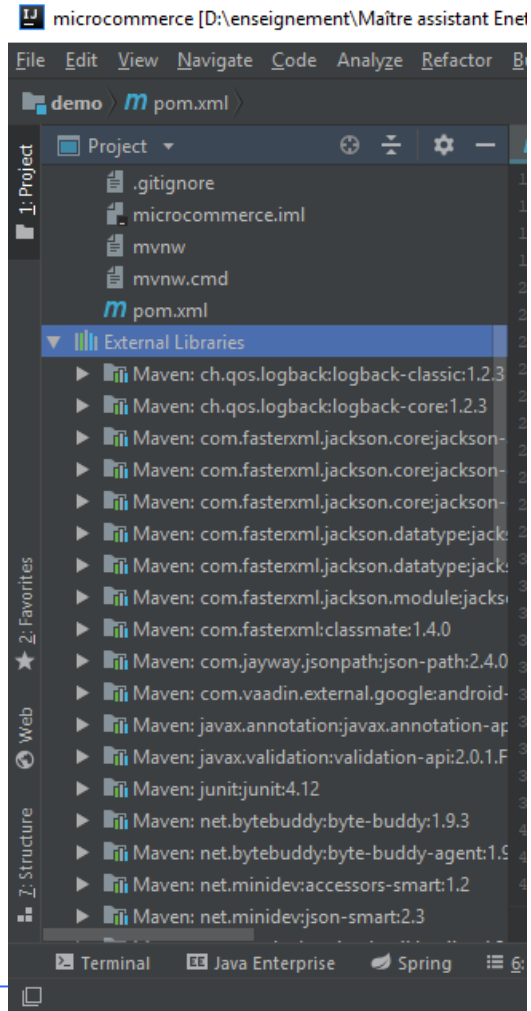
9. Etudier en détail les différents éléments de cette arborescence

External librairies

Pour voir la liste des dépendances importées, rendez-vous à gauche dans "External.

Vous avez dans cette liste principalement :

- Jackson : permet de parser JSON et faire le lien entre les classes Java et le contenu JSON.
- Tomcat : intégré, va nous permettre de lancer notre application en exécutant tout simplement le jar sans avoir à le déployer dans un serveur d'application.
- Hibernate : facilite la gestion des données.
- Logging grâce à logback et autres.



Créez un Microservice grâce à Spring Boot

Java EE

9. Etudier en détail les différents éléments de cette arborescence

- MicrocommerceApplication.java

Cette classe, générée automatiquement par Spring Boot, est le point de démarrage de l'application. Elle lance, entre autres, la classe SpringApplication, responsable du démarrage de l'application Spring. Cette classe va créer ApplicationContext dans lequel iront toutes les configurations générées automatiquement ou ajoutées par vos soins.

MicrocommerceApplication.java

```
1   package com.ecommerce.microcommerce;
2
3   import org.springframework.boot.SpringApplication;
4   import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6   @SpringBootApplication
7   public class MicrocommerceApplication {
8
9       public static void main(String[] args) {
10          SpringApplication.run(MicrocommerceApplication.class, args);
11      }
12  }
```

9. Etudier en détail les différents éléments de cette arborescence

- **MicrocommerceApplication.java**

L'annotation @SpringBootApplication, qui est une simple encapsulation de trois annotations:

```
MicrocommerceApplication.java
package com.ecommerce.microcommerce;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MicrocommerceApplication {

    public static void main(String[] args) {
        SpringApplication.run(MicrocommerceApplication.class, args);
    }
}
```

- **@Configuration** : donne à la classe actuelle la possibilité de définir des configurations qui iront remplacer les traditionnels fichiers XML. Ces configurations se font via des Beans.
- **@EnableAutoConfiguration** : l'annotation vue précédemment qui permet, au démarrage de Spring, de générer automatiquement les configurations nécessaires en fonction des dépendances situées dans notre classpath.
- **@ComponentScan** : Indique qu'il faut scanner les classes de ce package afin de trouver des Beans de configuration.

Java EE

9. Etudier en détail les différents éléments de cette arborescence

MicrocommerceApplication.java

L'annotation @SpringBootApplication, qui est une simple encapsulation de trois annotations: @Configuration, @EnableAutoConfiguration, @ComponentScan.

```java
MicrocommerceApplication.java
package com.ecommerce.microcommerce;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MicrocommerceApplication {

    public static void main(String[] args) {
        SpringApplication.run(MicrocommerceApplication.class, args);
    }
}
```

```java
...
...
@Configuration
@EnableAutoConfiguration
@ComponentScan

public class MicrocommerceApplication {

    public static void main(String[] args) {
        SpringApplication.run(MicrocommerceApplication.class, args);
    }
}
```

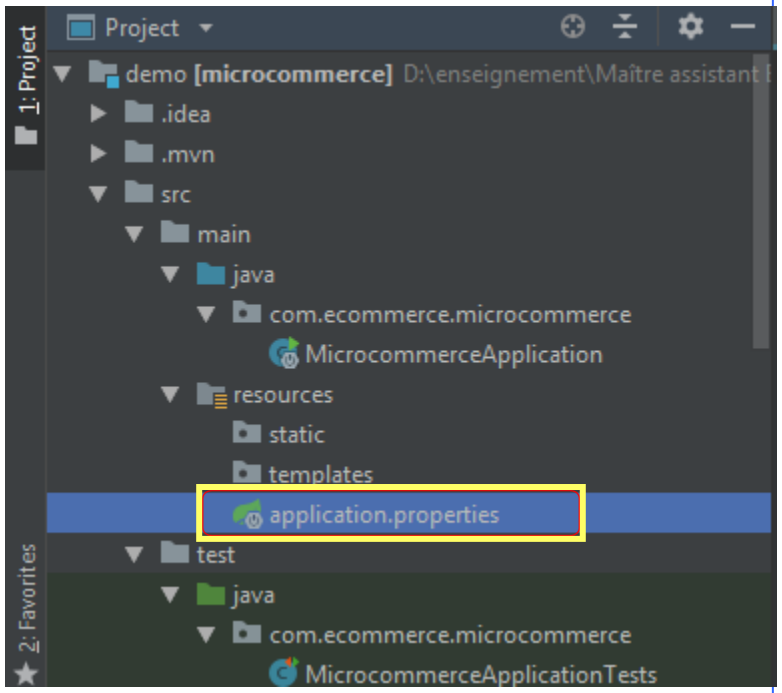
Java EE

9. Etudier en détail les différents éléments de cette arborescence

- application.properties

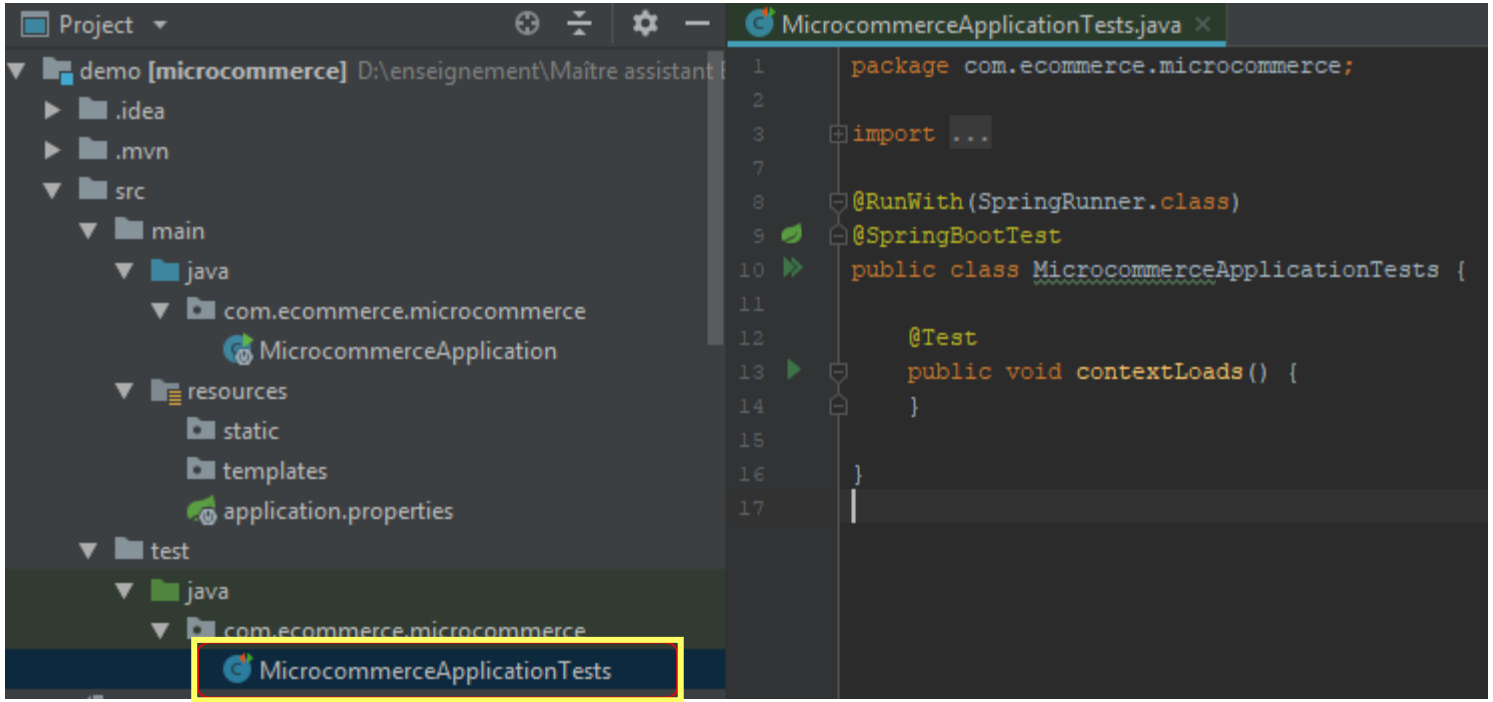
Ce fichier vous permet de modifier très simplement un nombre impressionnant de configurations liées à Spring Boot et ses dépendances.

Par exemple : changer le port d'écoute de Tomcat, l'emplacement des fichiers de log, les paramètres d'envoi d'emails, etc…



9. Etudier en détail les différents éléments de cette arborescence

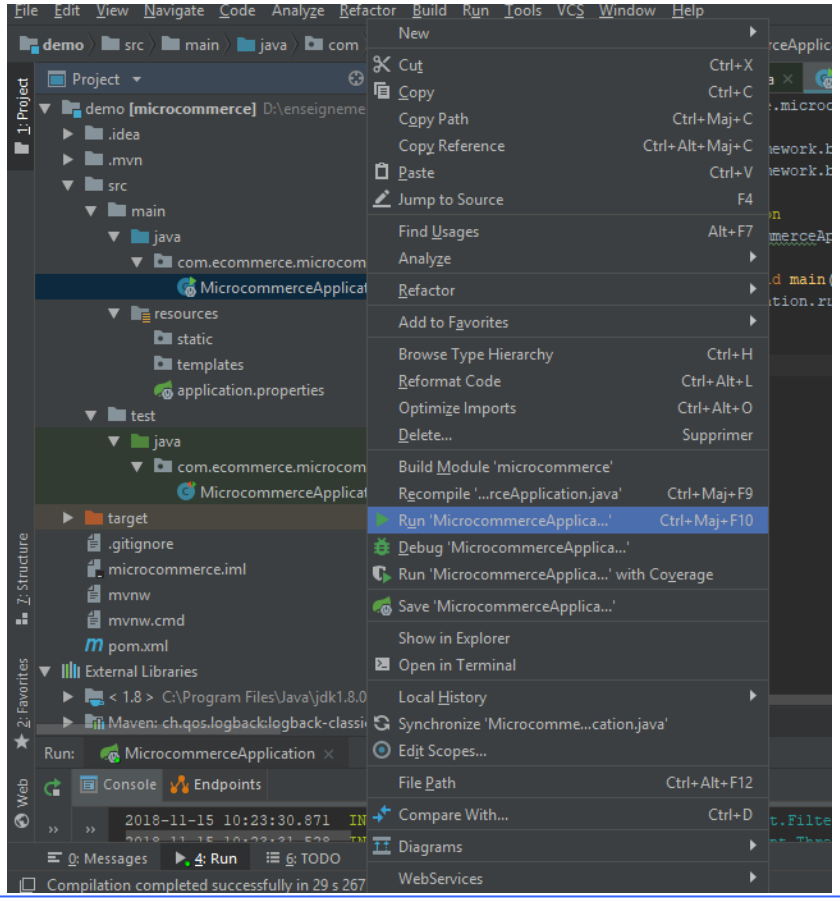
- MicrocommerceApplicationTests.java



Ce fichier vous permet d'écrire vos tests.

9. Etudier en détail les différents éléments de cette arborescence

- **Exécuter l'application**



9. Etudier en détail les différents éléments de cette arborescence

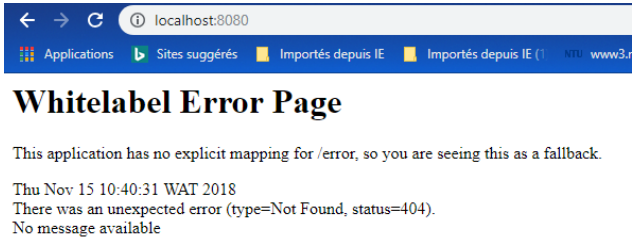
- **Le retour de l'exécution:**

```
Console   Endpoints

"C:\Program Files\Java\jdk1.8.0_152\bin\java.exe" ...

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.0.RELEASE)

2018-11-15 10:23:23.643  INFO 14468 --- [           main] c.e.m.MicrocommerceApplication           : Starting MicrocommerceApplication on DESKTOP-SIC37SA with PI
2018-11-15 10:23:23.671  INFO 14468 --- [           main] c.e.m.MicrocommerceApplication           : No active profile set, falling back to default profiles: def
2018-11-15 10:23:30.160  INFO 14468 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2018-11-15 10:23:30.229  INFO 14468 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-11-15 10:23:30.230  INFO 14468 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/9.0.12
2018-11-15 10:23:30.248  INFO 14468 --- [           main] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows opti
2018-11-15 10:23:30.735  INFO 14468 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-11-15 10:23:30.736  INFO 14468 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 6194
2018-11-15 10:23:30.861  INFO 14468 --- [           main] o.s.b.w.servlet.ServletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
2018-11-15 10:23:30.871  INFO 14468 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-11-15 10:23:30.871  INFO 14468 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-11-15 10:23:30.871  INFO 14468 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'formContentFilter' to: [/*]
2018-11-15 10:23:30.871  INFO 14468 --- [           main] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-11-15 10:23:31.528  INFO 14468 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2018-11-15 10:23:32.031  INFO 14468 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2018-11-15 10:23:32.040  INFO 14468 --- [           main] c.e.m.MicrocommerceApplication           : Started MicrocommerceApplication in 10.41 seconds (JVM runni
2018-11-15 10:40:31.560  INFO 14468 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2018-11-15 10:40:31.560  INFO 14468 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2018-11-15 10:40:31.567  INFO 14468 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 7 ms
```

Créez un Microservice grâce à Spring Boot

Java EE™

✘ **Utilisation du navigateur pour visualiser le résultat:** http://localhost:8080



??

On n'a encore pas fourni d'éléments à afficher !

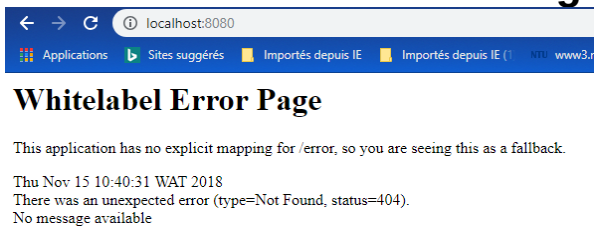
```
package com.ecommerce.microcommerce;
import
org.springframework.boot.SpringApplicatio
n;
import
org.springframework.boot.autoconfigure.Sp
ringBootApplication;
import
org.springframework.stereotype.Controller
;
import
org.springframework.web.bind.annotation.*
;
@Controller
@SpringBootApplication
```

```
public class
MicrocommerceApplication {
    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Hello World!";
    }
    public static void
main(String[] args) {

SpringApplication.run(Microcomme
rceApplication.class, args);
    }
}
```

Créez un Microservice grâce à Spring Boot

Java EE

✘ **Utilisation du navigateur pour visualiser le résultat:** http://localhost:8080

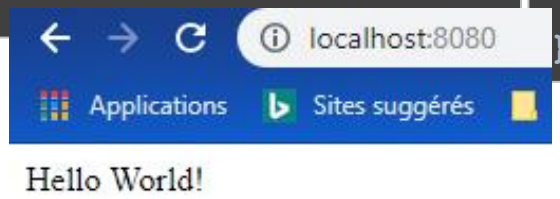


??

On n'a encore pas fourni d'éléments à afficher !

```
package com.ecommerce.microcommerce;
import
org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.Spri
ngBootApplication;
import
org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.*;
@Controller
@SpringBootApplication
```

```
public class
MicrocommerceApplication {
    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Hello World!";
    }
    public static void
main(String[] args) {

SpringApplication.run(Microcomme
rceApplication.class, args);
    }
}
```



Références

Livres

- **Java EE - Développez des applications web en Java (Nouvelle édition)**
- **Thierry Richard**

- **Conception d'applications en Java/JEE - 2e éd. - Principes, patterns et architectures**
- **Jacques Lonchamp**