Service Web, J2EE

5ème Année Ingénierie des Systèmes d'Informations

Architectures JEE

JSP et Servlets

Architecture des services web

Services web JEE

C’est quoi JAVA?

- « Java » est un langage de programmation, mais également une plate-forme.
- Le nom complet est « Java SE » pour Java Standard Edition(ancienne appellation « J2SE »)
- Contient de nombreuses bibliothèques, ou API:
  - java.lang,
  - java.io,
  - java.math,
  - java.util, etc.
- Toutes ces bibliothèques contiennent un nombre conséquent de classes et de méthodes prêtes à l'emploi pour effectuer toutes sortes de tâches.

Les plate-formes Java

- Les plate-formes Java se composent de trois éditions, destinées à des usages différents :
  - **JME** : *Java Micro Edition* est prévue pour le développement d'applications embarquées, notamment sur des PDA (*Personal Digital Assistant*) et terminaux mobiles (smartphone, …) ;
  - **JSE** : *Java Standard Edition* est destinée au développement d'applications pour des ordinateurs personnels;
  - **JEE** : *Java Enterprise Edition*, destinée à un usage professionnel avec la mise en œuvre de serveurs.
- Chaque édition propose un environnement complet pour le développement et l'exécution d'applications basées sur Java et comprend notamment une machine virtuelle de Java (Java Virtual Machine) ainsi qu'une bibliothèque de classes.

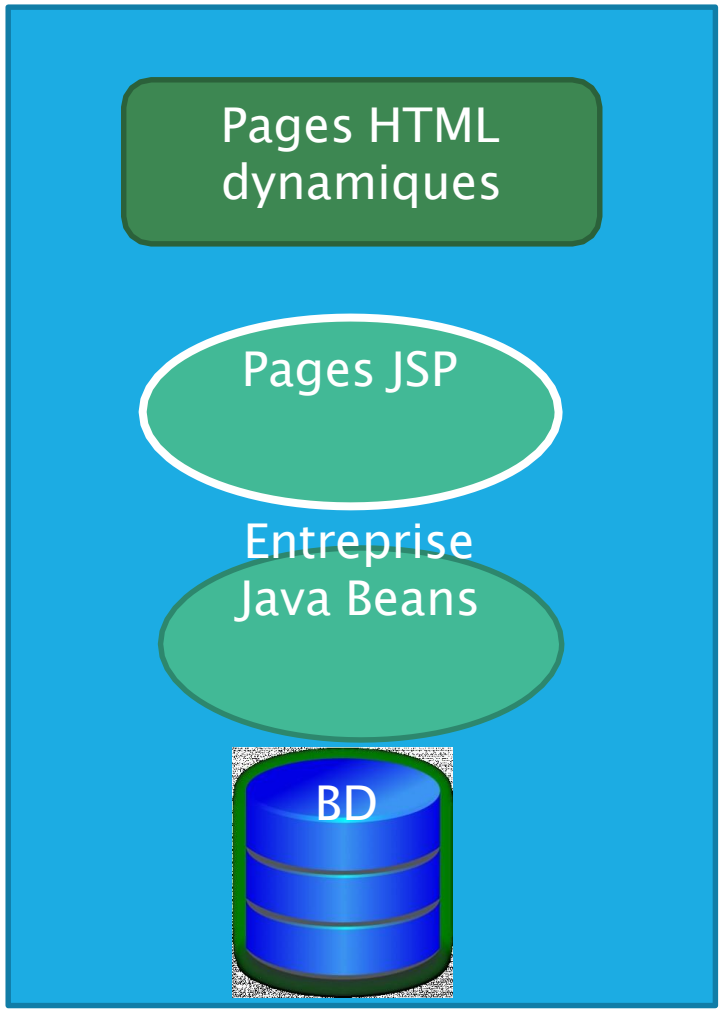
C’est quoi JAVA EE?

- « Java EE » acronyme de **Java Enterprise Edition (ancienne appellation** « J2EE »)
- C’est une extension de la plate-forme Standard Edition.
- la plate-forme Java EE est construite sur le langage Java et la plate-forme Java SE.
- Java EE permet de faciliter le développement d'applications complexes, robustes et distribuées, déployées et exécutées sur un serveur d'applications.

Pourquoi choisir JAVA EE?

- Java EE a été créé pour le développement d'applications d'entreprises.
- Java EE est conçu pour faciliter le travail en équipe sur un même projet
  - l'application est découpée en couches, et le serveur sur lequel tourne l'application est lui-même découpé en plusieurs niveaux.
- Java EE fournit un ensemble d'extensions au Java standard afin de faciliter la création d'applications centralisées.

Application multi-tiers



Tiers client

Tiers Web

Tiers métier

Tiers ressources (EIS)

Machine client

Machine serveur Java EE

Machine serveur de bd

Les API de Java EE

Les API (*Application Programming Interface*) de Java EE peuvent se répartir en deux grandes catégories :

1. **Les composants**
   - Les composants web
   - Les composants métier
2. **Les services**
   - Les services d'infrastructures
   - Les services de communication

Les API de Java EE

Les composants :

- **Les composants Web** : Servlets et JSP (Java Server Pages). Il s'agit de la partie chargée de l'interface avec l'utilisateur (on parle de *logique de présentation*).
- **Les composants métier** : EJB (Enterprise Java Beans). Il s'agit de composants spécifiques chargés des traitements des données propres à un secteur d'activité (on parle de *logique métier* ou de *logique applicative*) et de l'interfaçage avec les bases de données.

Les API de Java EE

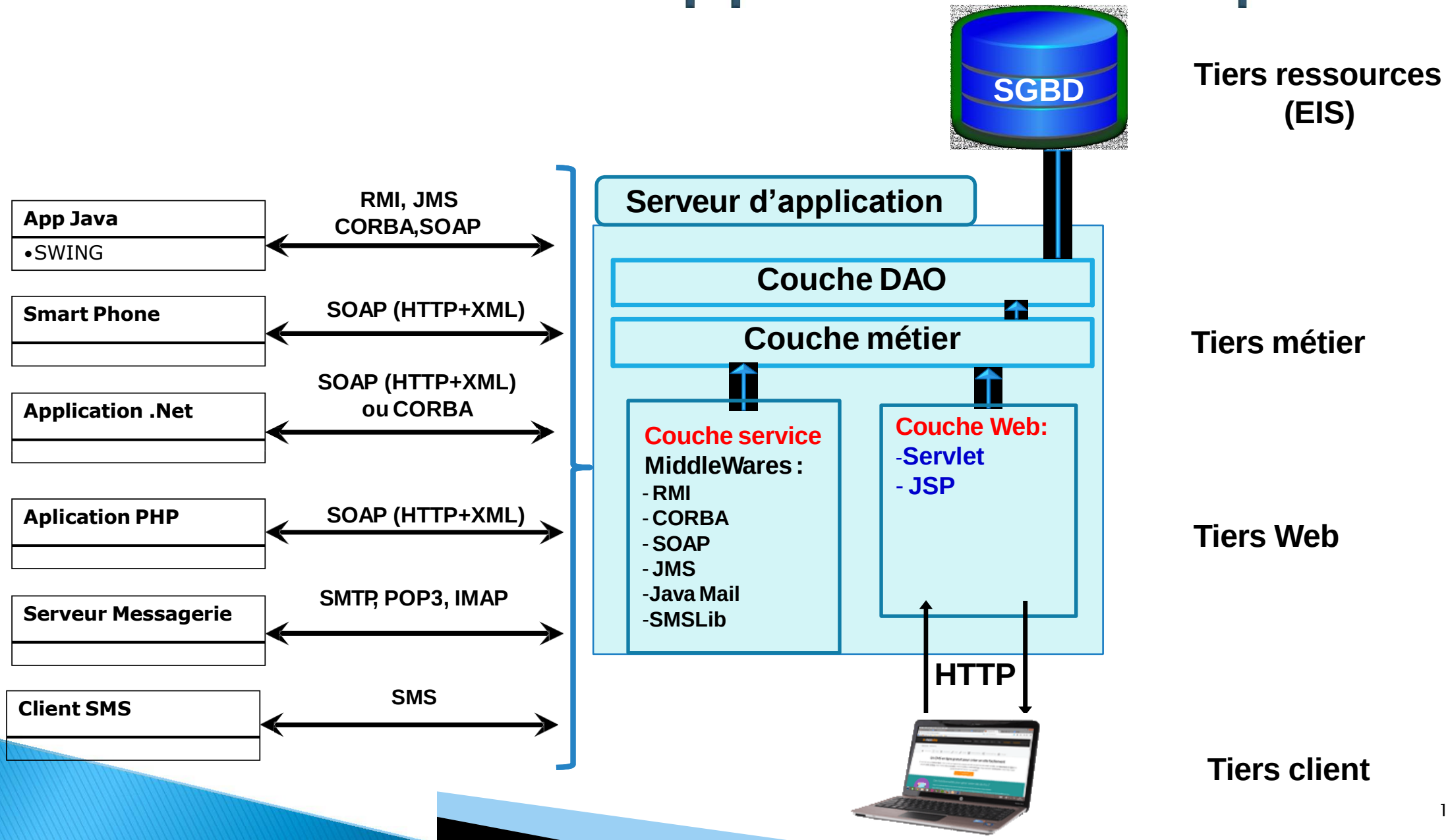
Les **services :**

- Les **services d'infrastructures** :
  - **JDBC** (*Java DataBase Connectivity*) est une API d'accès aux bases de données relationnelles.
  - **JNDI** (*Java Naming and Directory Interface*) est une API d'accès aux services de nommage et aux annuaires tels que DNS, NIS, LDAP, COS Naming pour les objets Corba, RMI etc.
  - **JTA/JTS** (*Java Transaction API/Java Transaction Services*) est une API définissant des interfaces standard avec un gestionnaire de transactions.
  - **JCA** (*JEE Connector Architecture*) est une API de connexion au système d'information de l'entreprise EIS (ERP, SGBD, applications anciennes écrites en C, C++, COBOL…).
  - **JMX** (*Java Management Extension*) permet de gérer le fonctionnement d'une application Java en cours d'exécution.

Les API de Java EE

- **Les services de communication :**
  - **JAAS** (*Java Authentication and Authorization Service*) est une API de gestion de l'authentification et des droits d'accès.
  - **JavaMail** est une API permettant l'envoi de courrier électronique.
  - **JMS** (*Java Message Service*) fournit des fonctionnalités de communication asynchrone (appelées *MOM* pour *Middleware Object Message*) entre applications.
  - **RMI-IIOP** (*Remote Method Invocation Over Internet Inter-ORB Protocol*) est une API permettant la communication synchrone entre objets.

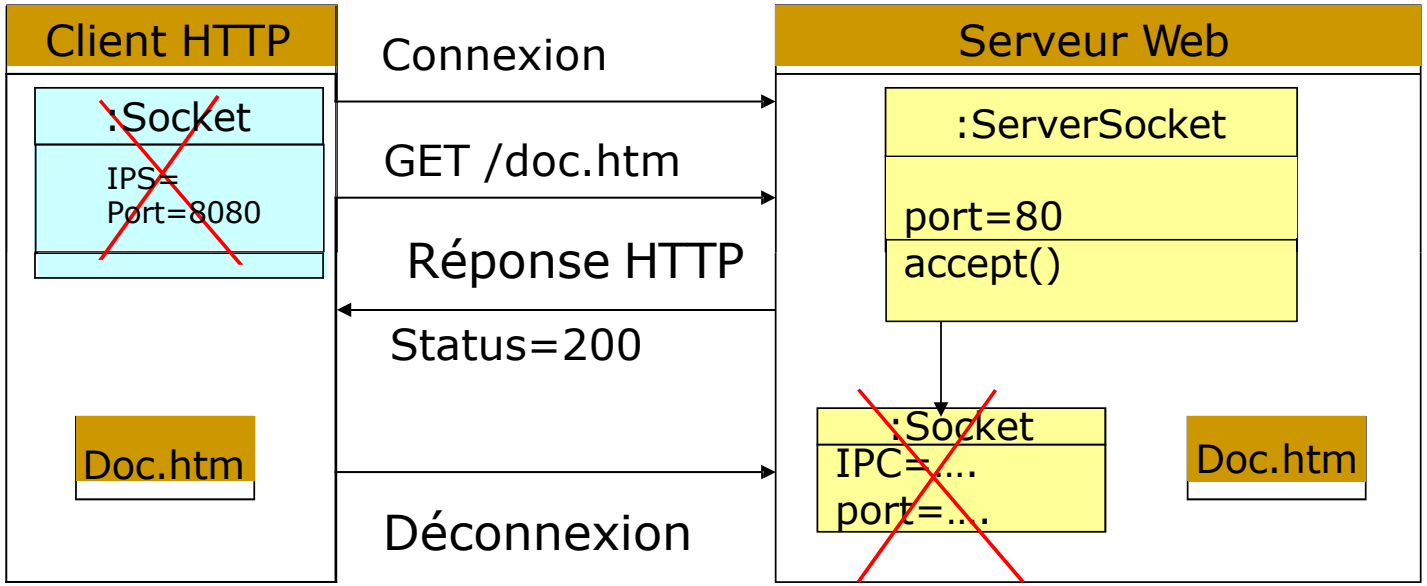
Architectures d'une Application Entreprise



Le protocole HTTP

- **HTTP :HyperText Transfert Protocol**
  - Protocole qui permet au client de récupérer des documents du serveur
  - Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP…)
  - Ce protocole permet également de soumissionner les formulaires
- **Fonctionnement (très simple en HTTP/1.1)**
  - Le client se connecte au serveur (crée une socket)
  - Le client envoie une demande au serveur Requête HTTP
  - Le serveur renvoi au client une réponse(status=200) ou d'une erreur (status=404 quand la ressource demandée n'existe pas)
  - Déconnexion

Le protocole HTTP: Connexion

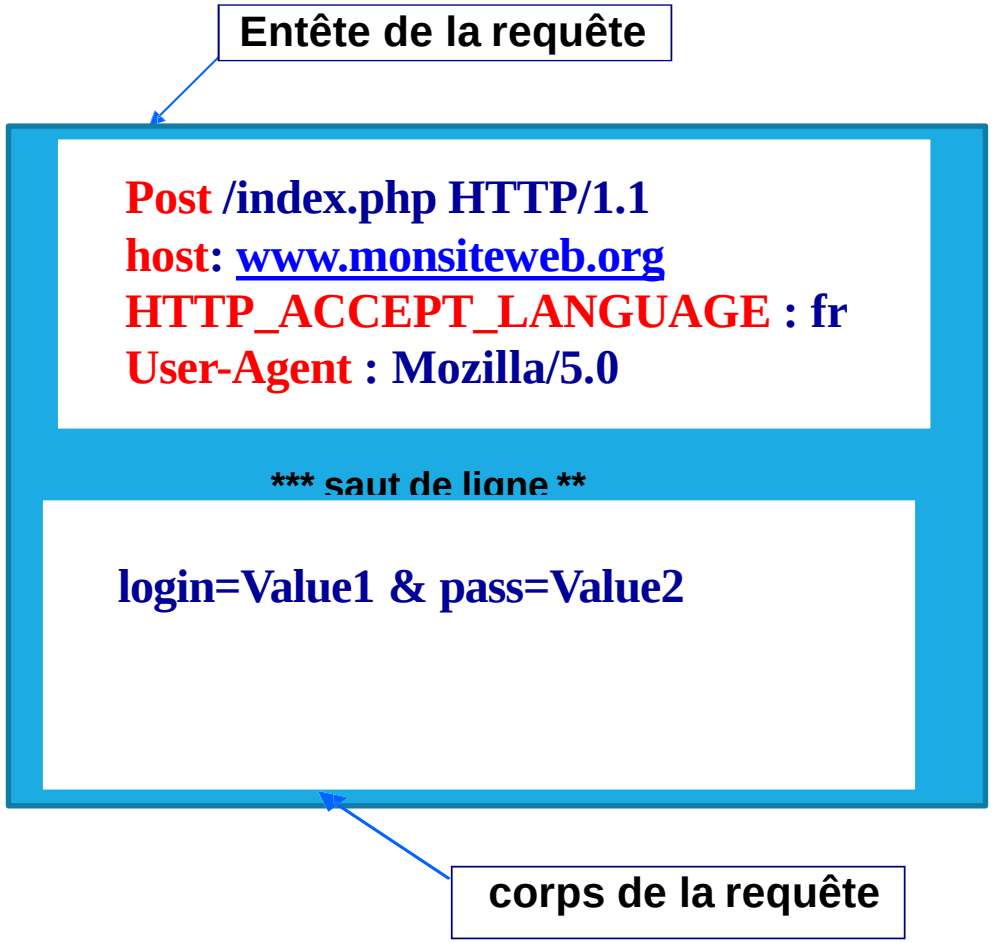


Méthodes du protocole HTTP

Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:

- GET : Pour récupérer le contenu d'un document
- POST : Pour soumissionner des formulaires (Envoyer, dans la requête, des données saisies par l'utilisateur)
- PUT pour envoyer un fichier du client vers le serveur
- DELETE permet de demander au serveur de supprimer un document.
- HEAD permet de récupérer les informations sur un document (Type, Capacité, Date de dernière modification etc…)

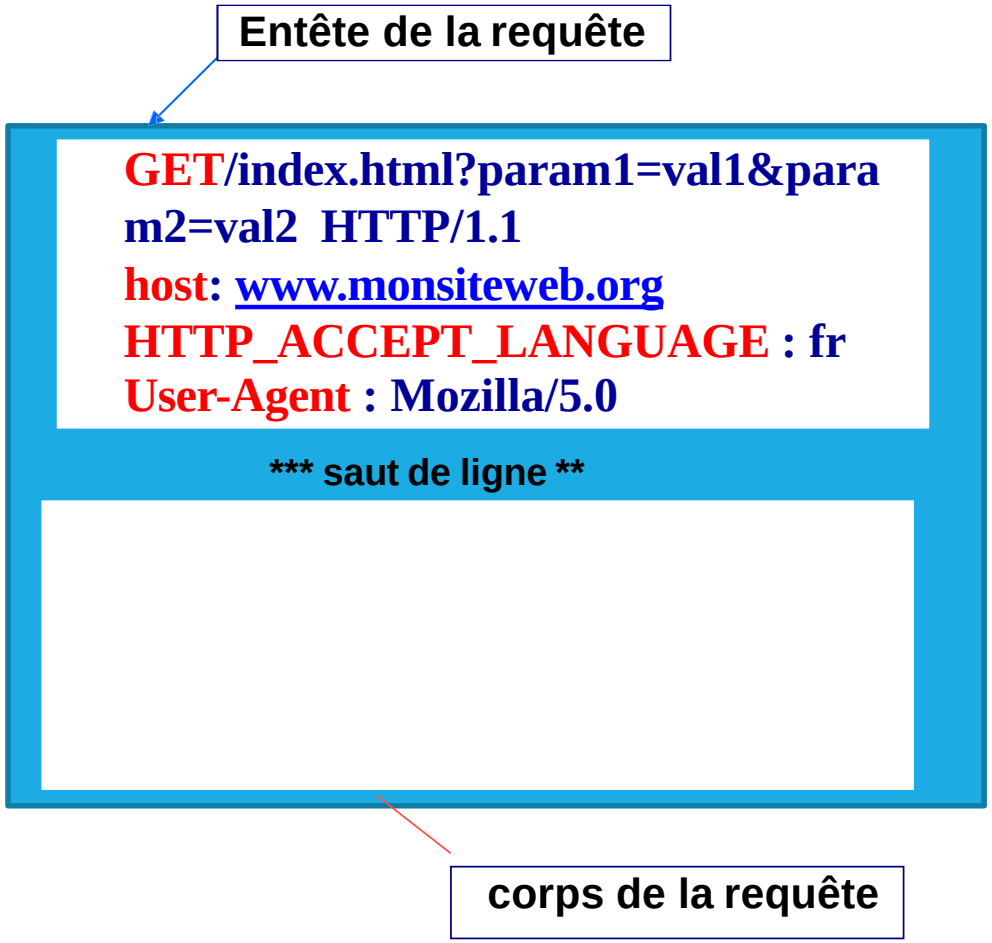
La requête POST (envoyé par le client) :



Méthode,chemin,version
Nom de domaine
Code de la langue
Type et version du navigateur

Paramètres des différents champs du formulaire.

La requête GET(envoyé par le client) :



La requête GET(retournée par le serveur)

Ligne de Status

Date du serveur

Nom du Serveur

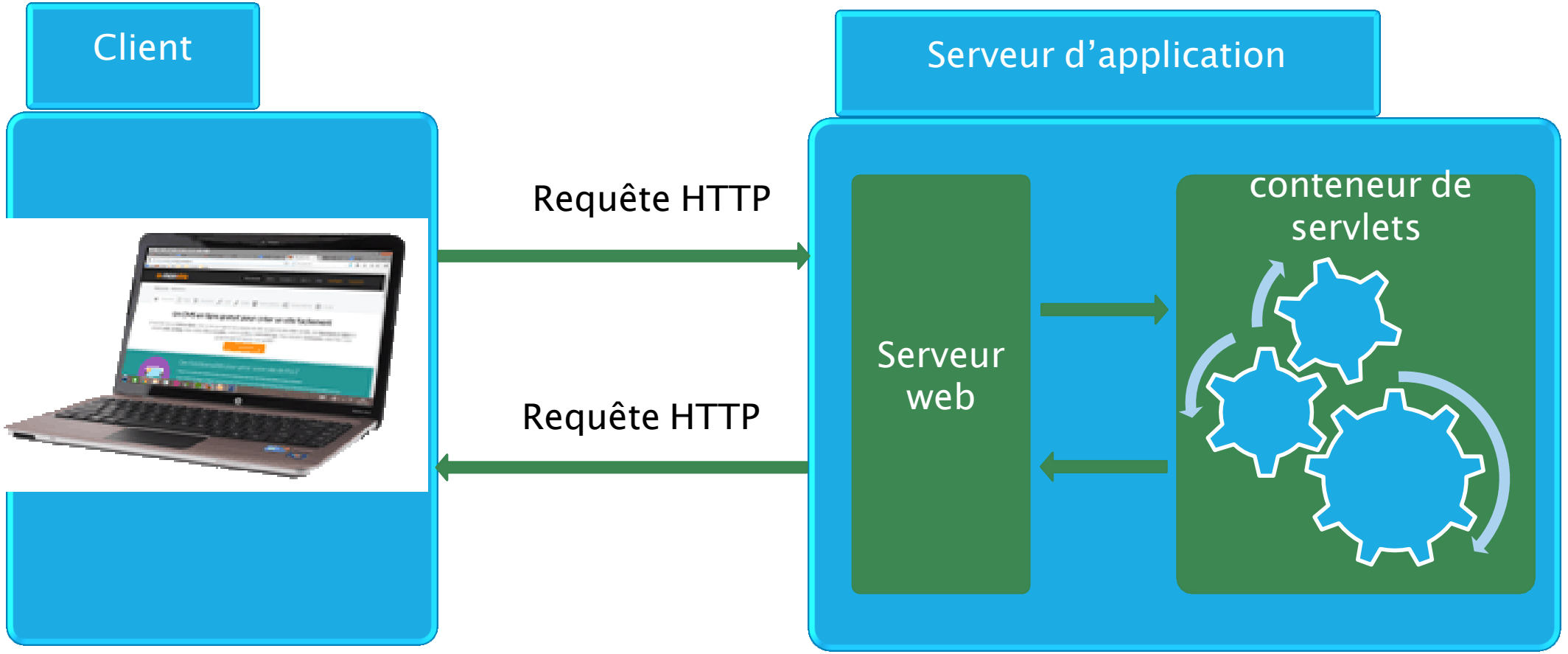
Dernière modification

Type de contenu

Sa taille

Le fichier que le client va afficher

Serveur d'application



Serveur d'application

les solutions propriétaires et payantes

- WebLogic(Oracle)
- WebSphere(IBM)

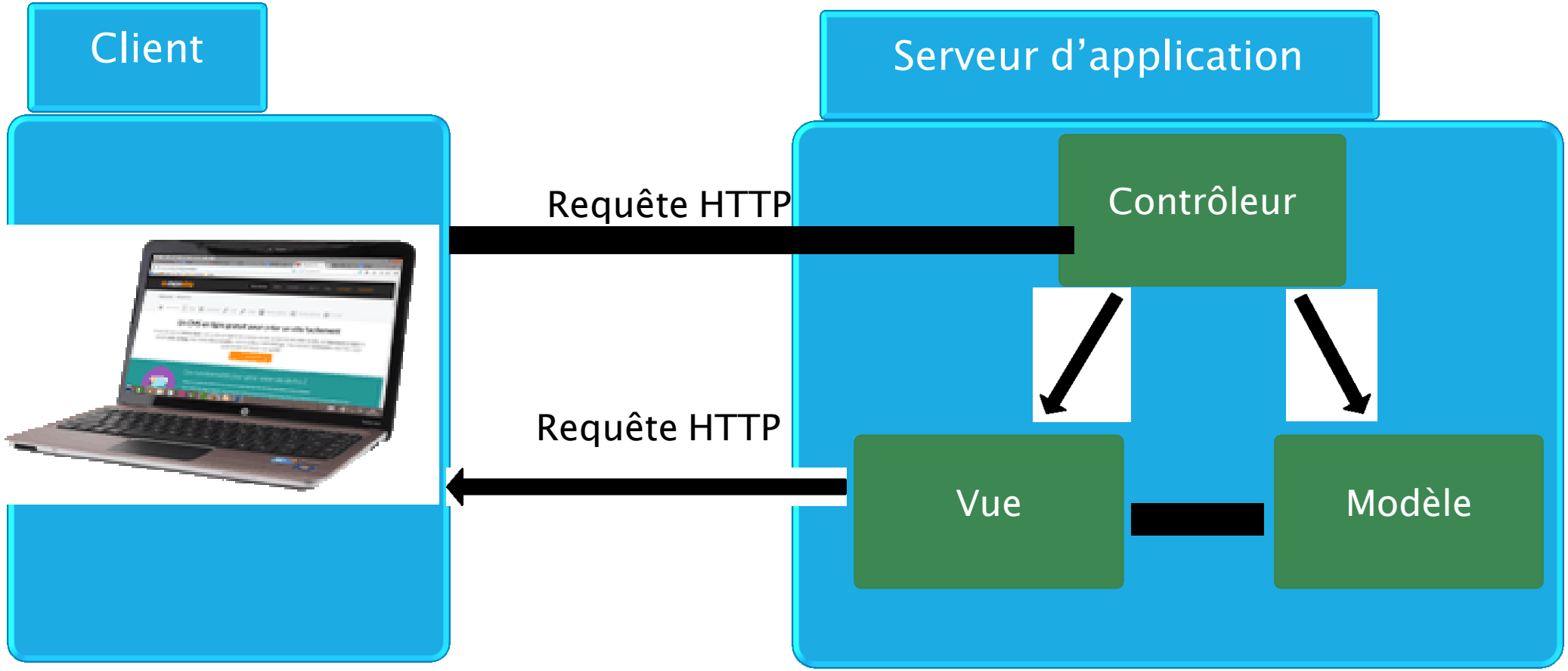
les solutions libres et gratuites

- Apache Tomcat
- Jboss
- GlassFish
- JOnAS

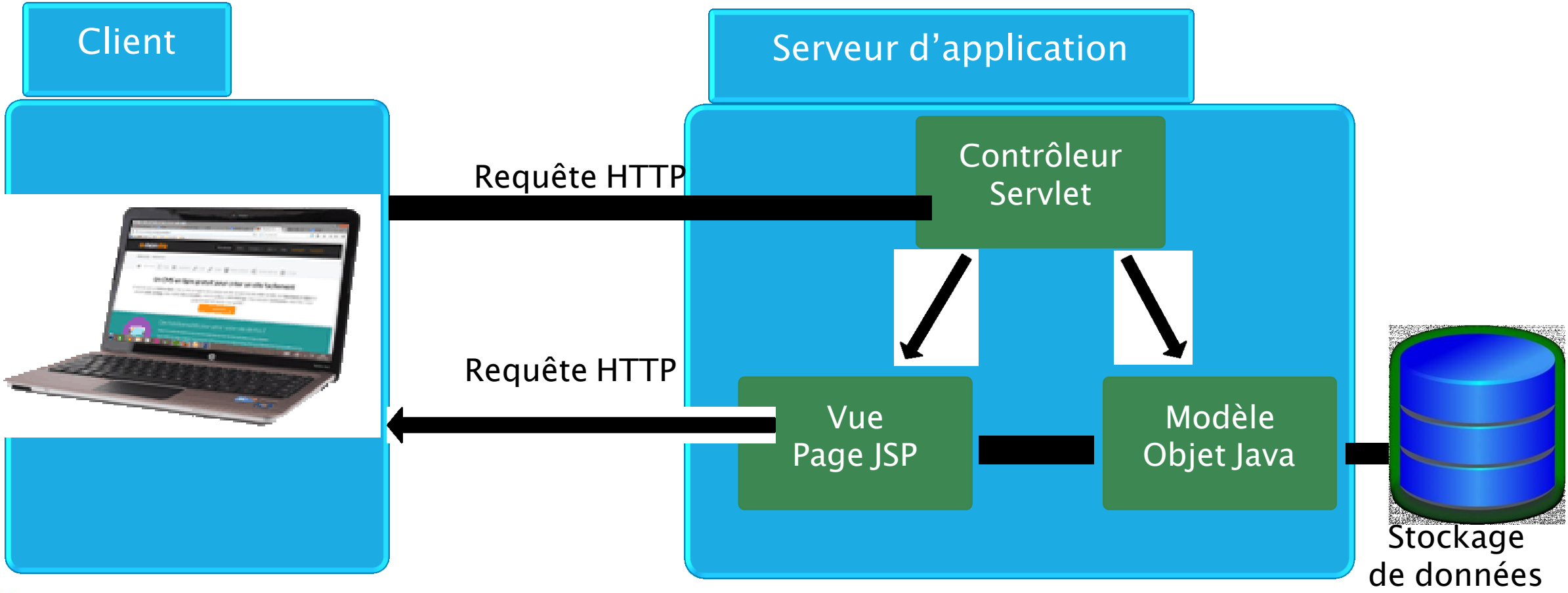
le modèle MVC (Modèle-Vue-Contrôleur)

- Design pattern, modèle de conception (ou encore patron de conception),
- Il découpe littéralement l'application en couches distinctes,
  - tout ce qui concerne le traitement, le stockage et la mise à jour des données de l'application doit être contenu dans la couche nommée "Modèle" (le M de MVC) ;
  - tout ce qui concerne l'interaction avec l'utilisateur et la présentation des données (mise en forme, affichage) doit être contenu dans la couche nommée "Vue" (le V de MVC) ;
  - tout ce qui concerne le contrôle des actions de l'utilisateur et des données doit être contenu dans la couche nommée "Contrôle" (le C de MVC).

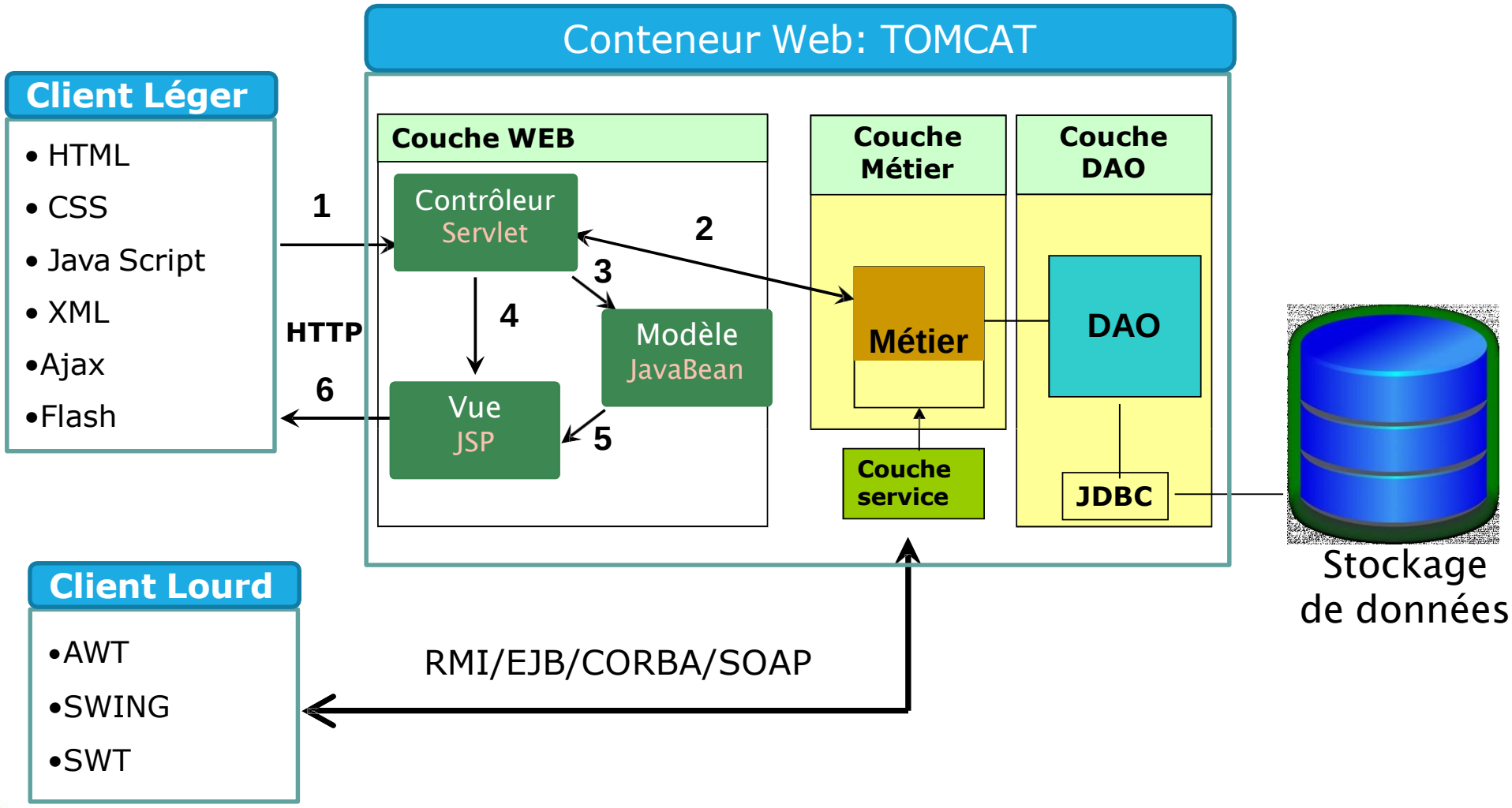
Le modèle MVC



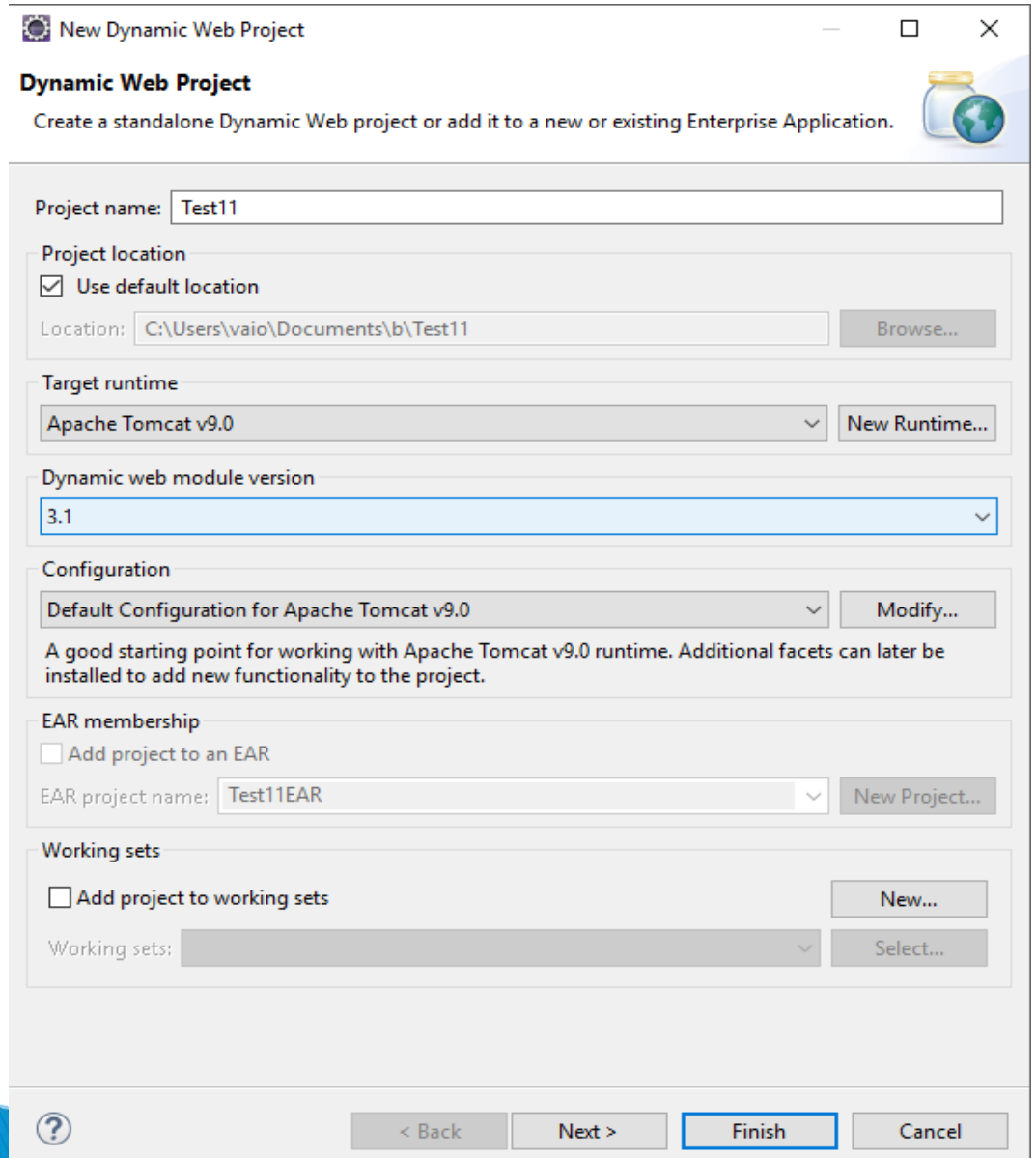
Le modèle MVC appliqué au Java EE



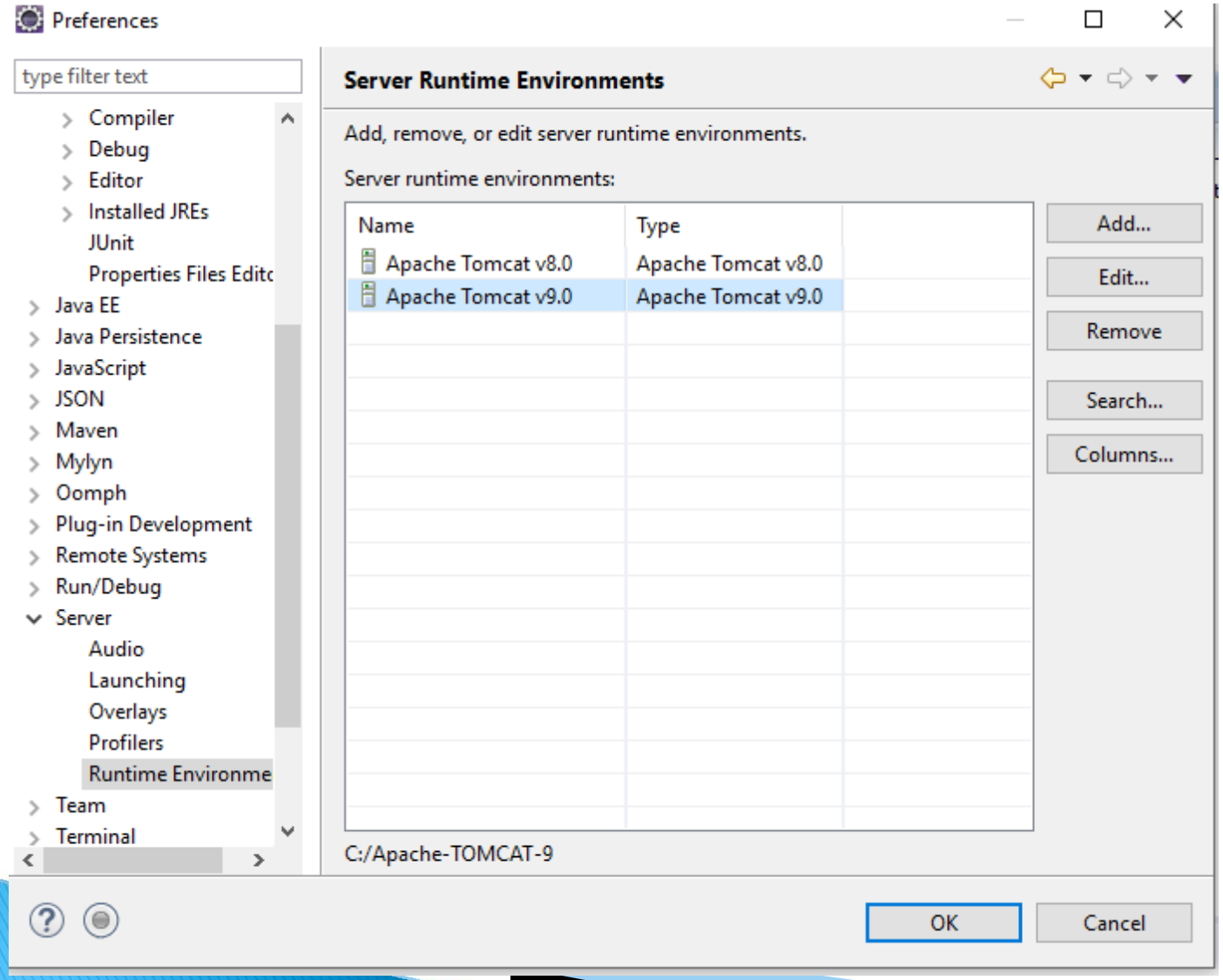
Architecture Java EE



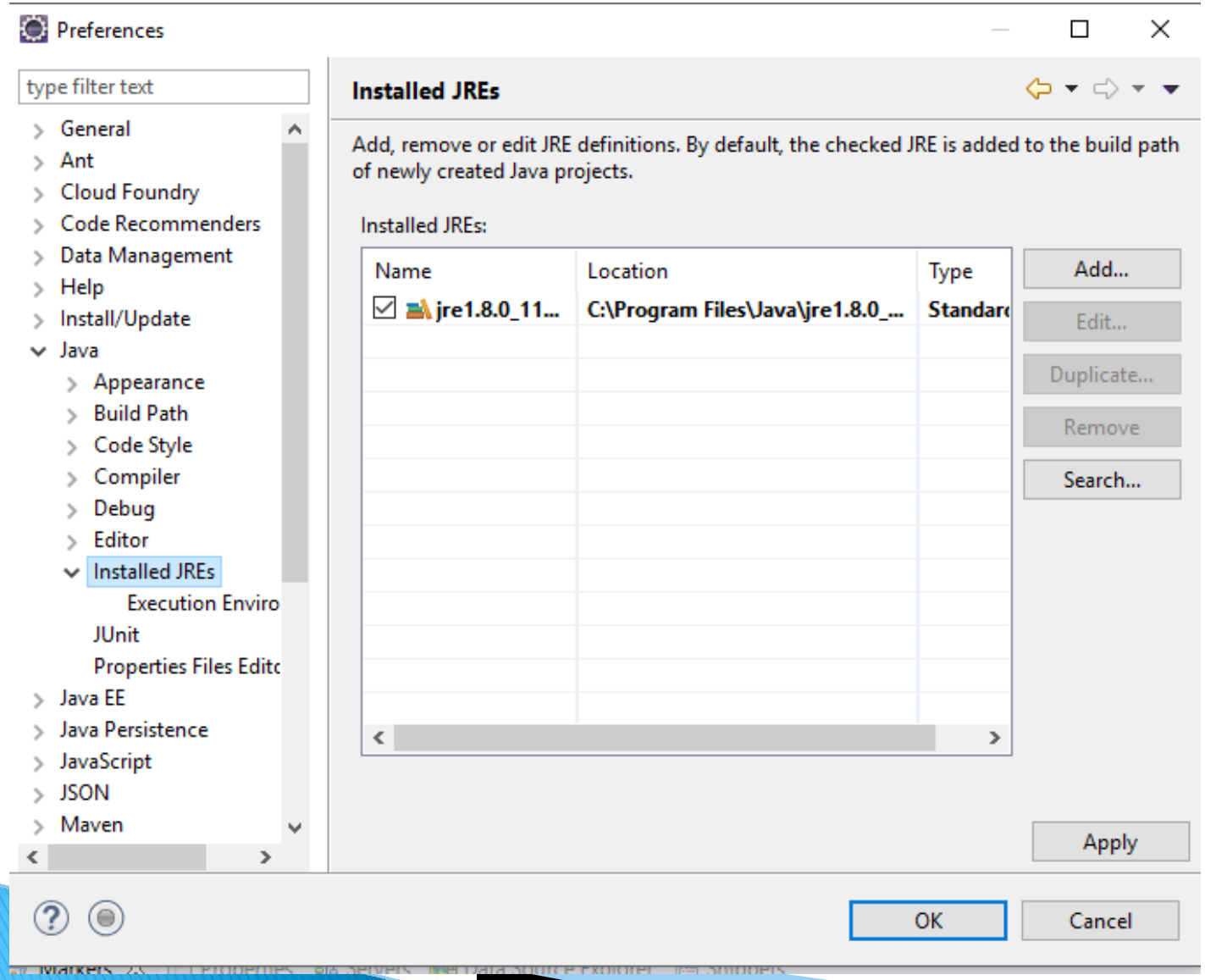
Environnement de travail sous Eclipse



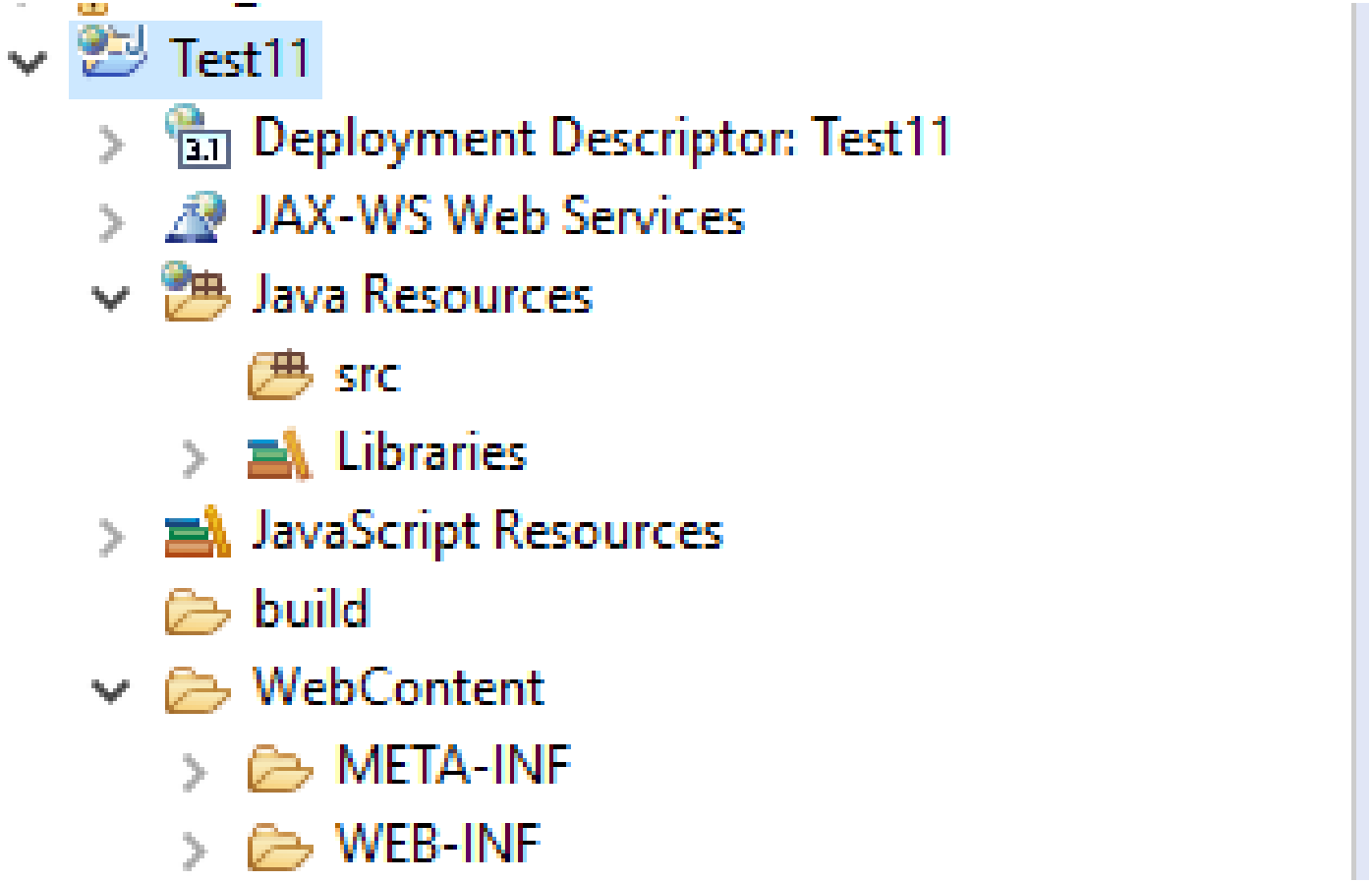
Environnement de travail sous Eclipse



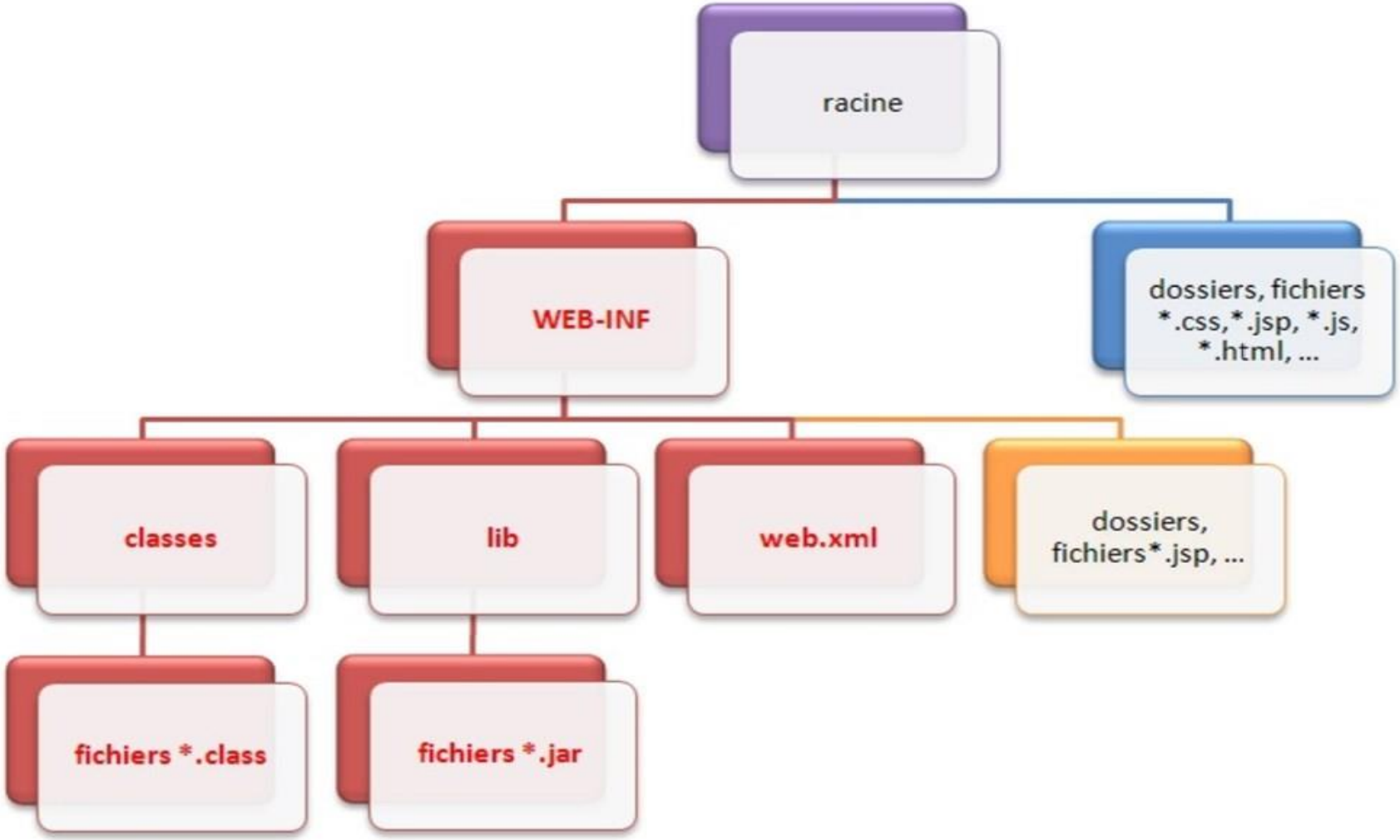
Environnement de travail sous Eclipse



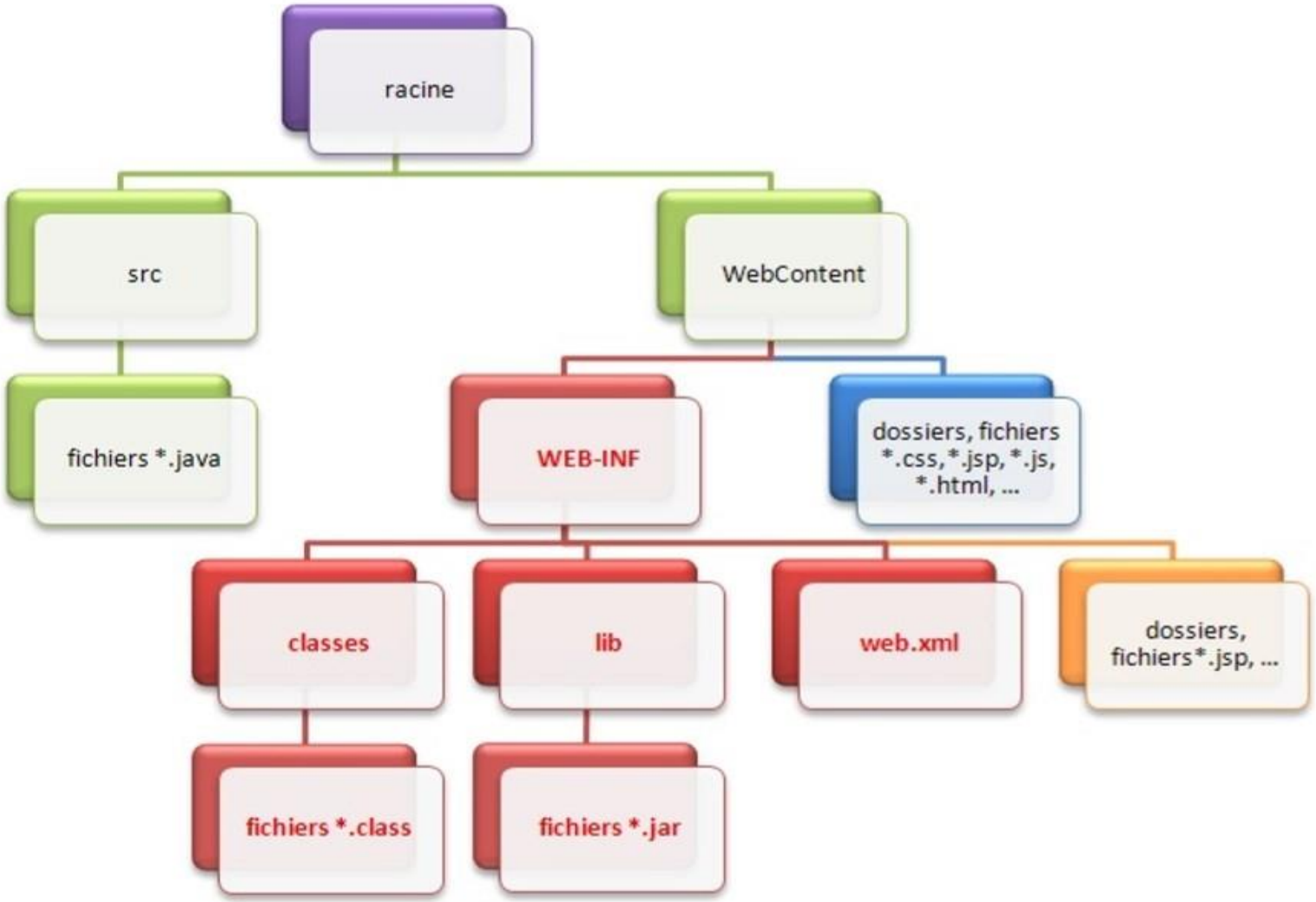
Environnement de travail sous Eclipse



Structure standard d'une application Java EE



Application Java EE sous Eclipse



Partie 2

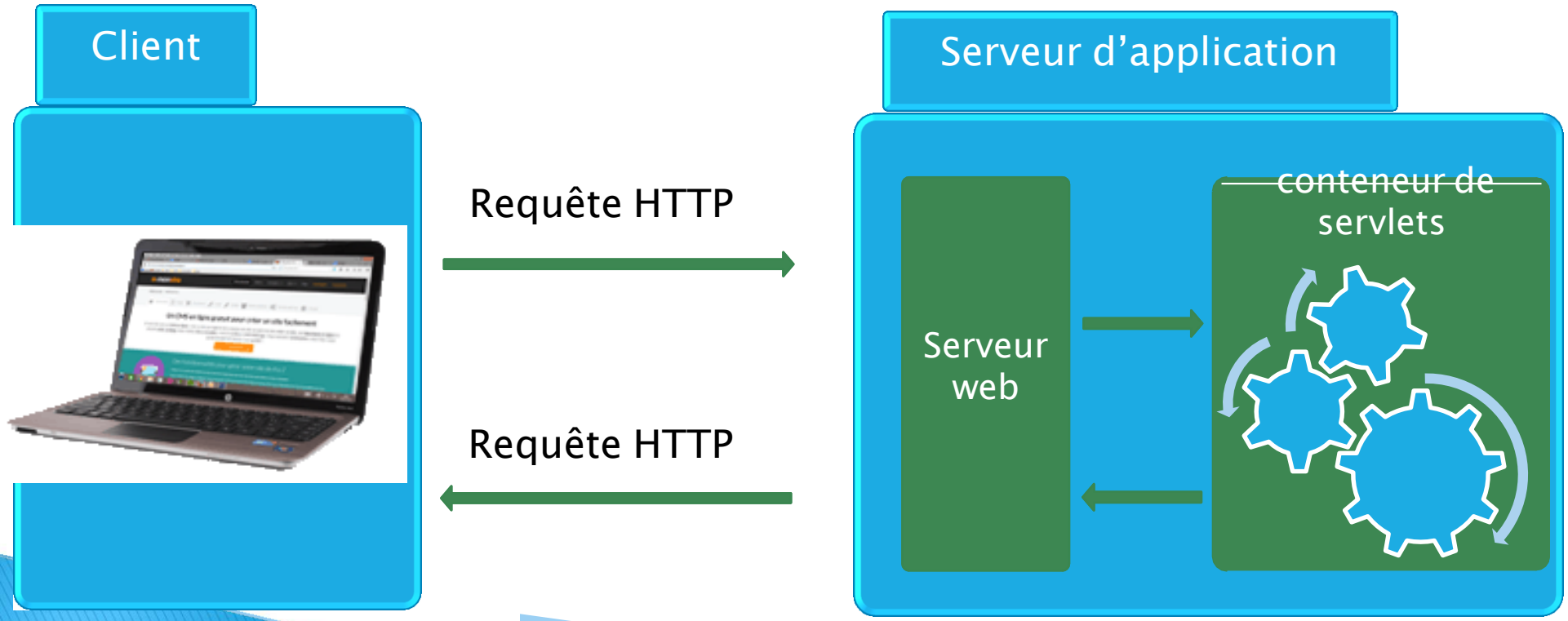
- Les Servlets

Introduction aux servlets

- Une servlet est une simple classe Java, qui a la particularité de permettre le traitement de requêtes et la personnalisation de réponses.
- Une Servlet s'exécute dans un **moteur de Servlet** ou **conteneur de Servlet** permettant d'établir le lien entre la Servlet et le serveur Web

Introduction aux servlets

- les servlets sont indépendantes du serveur web
  - Elles s'exécutent dans un **Conteneur de servlets** utilisé pour établir le lien entre la servlet et le serveur Web



Pourquoi utiliser des servlets?

- Les servlets ont de nombreux avantages par rapport aux autres technologies côté serveur:
  - Peut utiliser toutes les API Java afin de communiquer avec des applications extérieures, se connecter à des bases de données, accéder aux entrée-sorties…
  - Sont indépendantes du serveur Web.
  - Se chargent automatiquement lors du démarrage du serveur ou bien lors de la connexion du premier client.
  - La résidence en mémoire leur permettent :
    - De traiter les demandes des clients grâce à des threads.
    - D'occuper moins de mémoire et de charge du processeur.

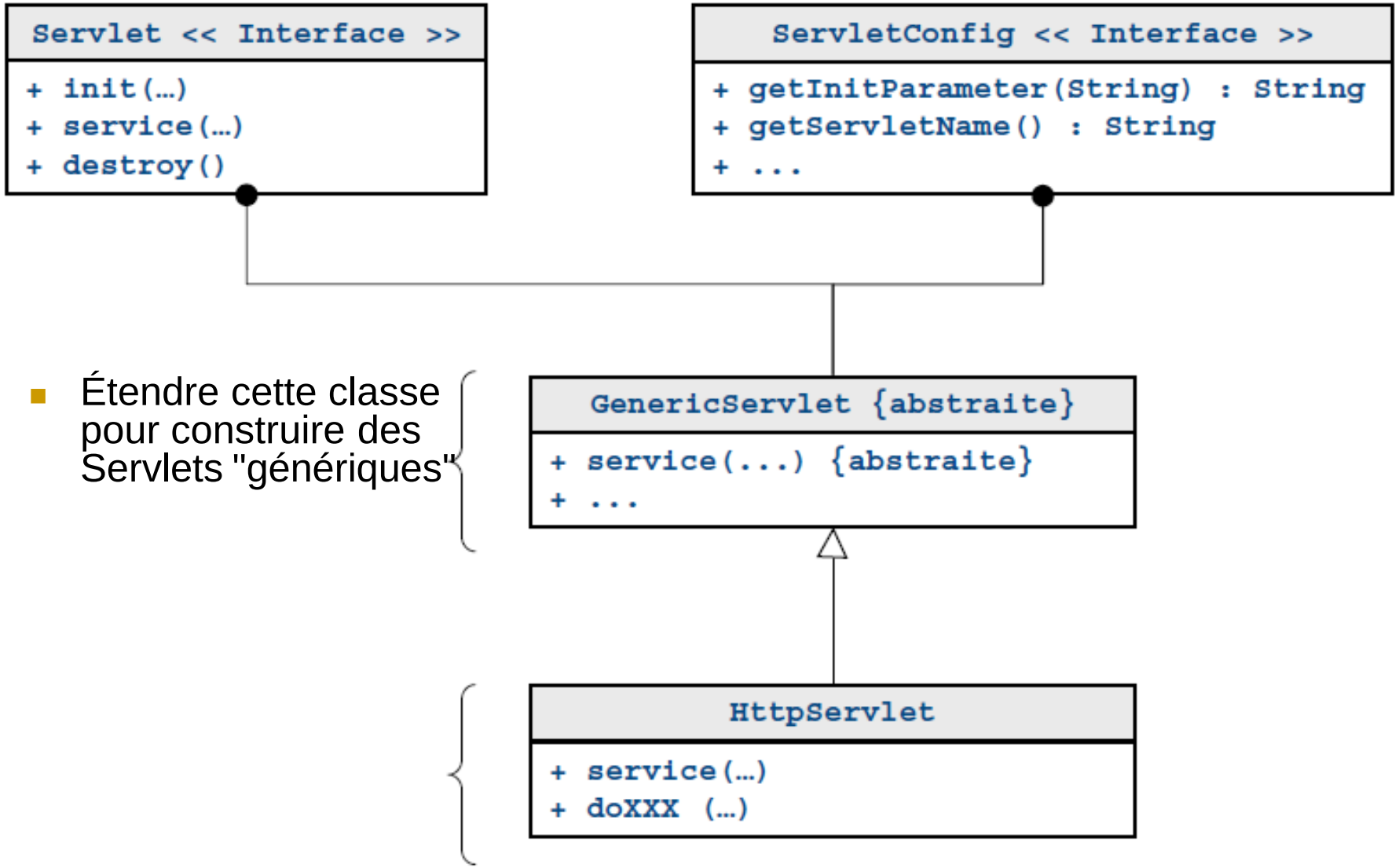
Conteneur de servlets

- Un conteneur de servlets permet d'établir le lien entre la Servlet et le serveur Web
- Il prend en charge et gère les servlets:
  - chargement de la servlet
  - gestion de son cycle de vie
  - passage des requêtes et des réponses
- Deux types de conteneurs
  - Conteneurs de Servlets autonomes : c'est un serveur Web qui intègre le support des Servlets
  - Conteneurs de Servlets additionnels : fonctionnent comme un plug-in à un serveur Web existant

Conteneur de servlets

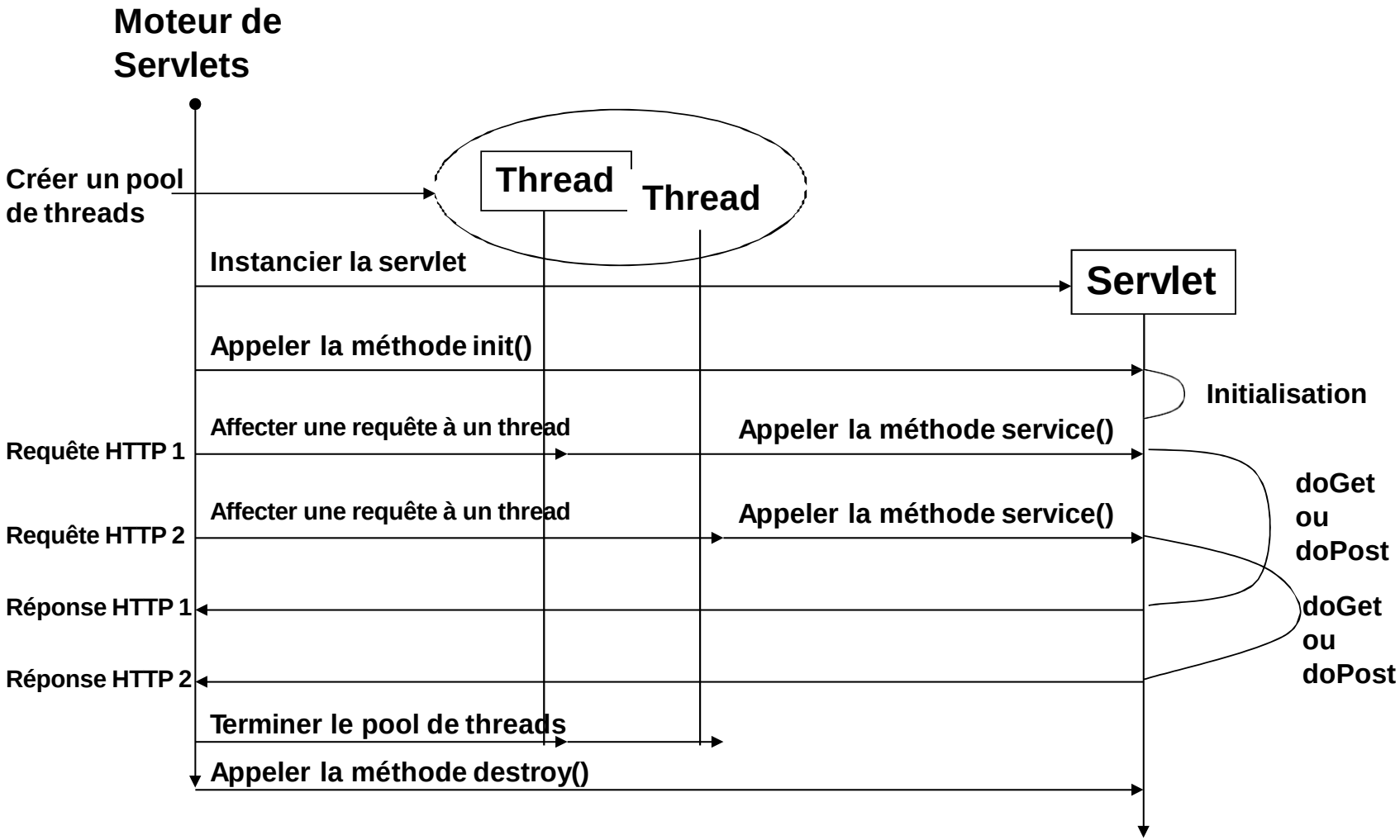
- Apache Tomcat est un conteneur web libre de servlets et JSP Java EE. Issu du projet Jakarta, c'est un des nombreux projets de l'Apache Software Foundation. Il implémente les spécifications des servlets et des JSP du Java Community Process3, est paramétrable par des fichiers XML et des propriétés, et inclut des outils pour la configuration et la gestion. Il comporte également un serveur web.
- Dans le reste du cours et des TP, nous utiliserons le conteneur web Tomcat pour déployer nos servlets.

L'API Servlet



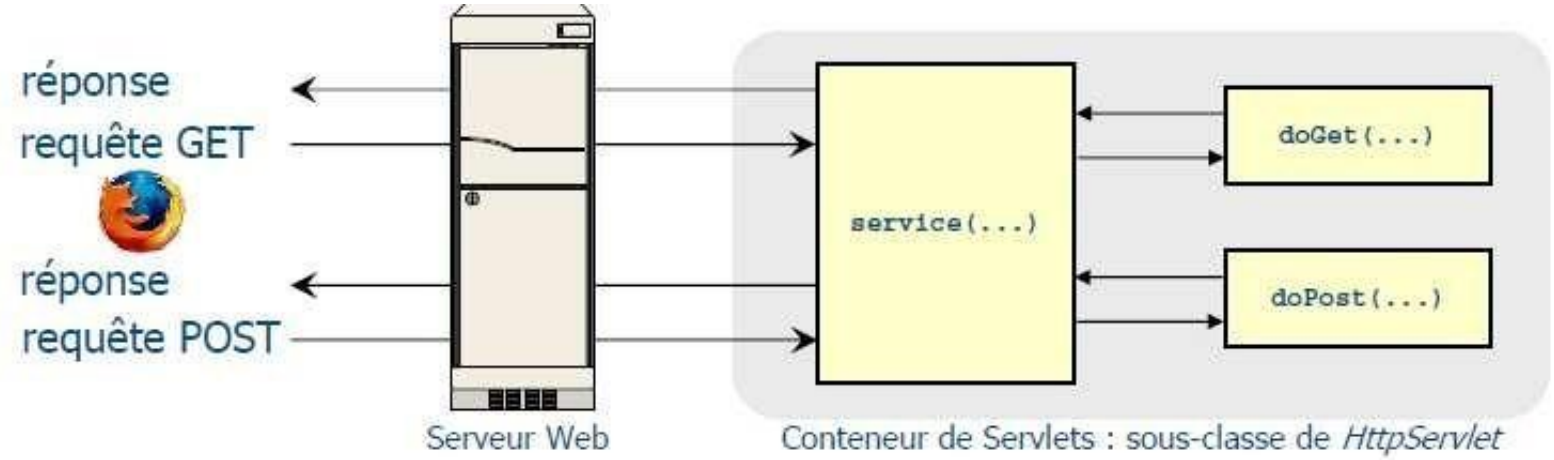
- Étendre cette classe pour construire des Servlets "génériques"

Gestion des servlets



HttpServlet

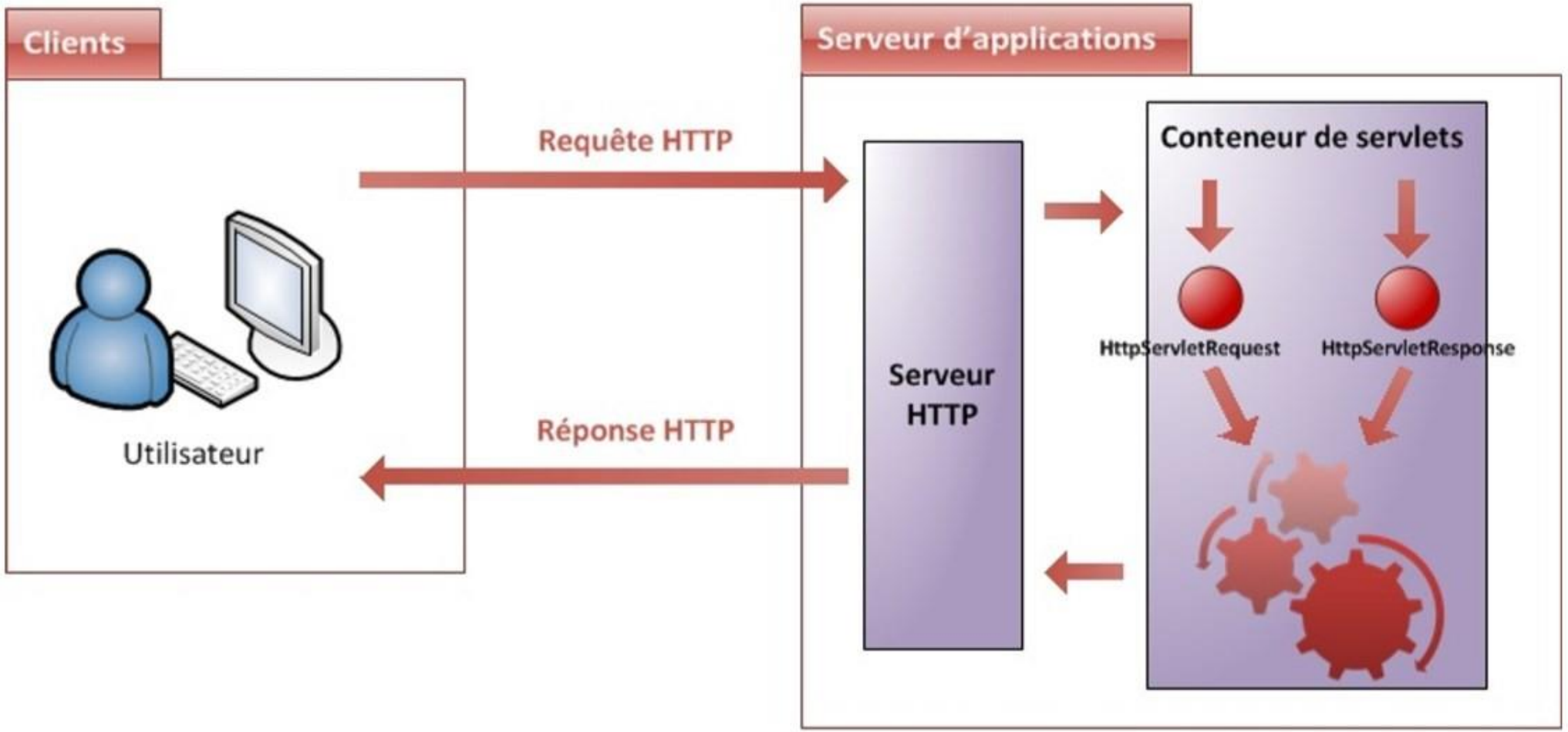
- Dans la suite du cours nous allons utiliser uniquement des Servlets qui réagissent au protocole HTTP d'où l'utilisation de la classe HttpServlet
- HttpServlet redéfinit la méthode service(…)
- service(…) lit la méthode (GET, POST, …) à partir de la requête
- Elle transmet la requête à une méthode appropriée de HttpServlet destinée à traiter le type de requête (GET, POST, …)



Fonctionnement d'une servlet

- Lorsqu'une servlet est appelée par un client, la méthode *service()* est exécutée. Celle-ci est le principal point d'entrée de toute servlet et accepte deux objets en paramètres:
  - L'objet Http*ServletRequest* encapsulant la requête du client, c'est-à-dire qu'il contient l'ensemble des paramètres passés à la servlet (informations sur l'environnement du client, cookies du client, URL demandée, ...)
  - L'objet Http*ServletResponse* permettant de renvoyer une réponse au client (envoyer des informations au navigateur).

Fonctionnement d'une servlet



Conteneur et paire d'objets requête/réponse

Développement d'une servlet

- Une servlet est une classe qui hérite de la classe HttpServlet et qui redéfinit les méthodes du protocole HTTP.
- Si la méthode utilisée est GET, il suffit de redéfinir la méthode :

```
public void doGet(
    HttpServletRequest request, HttpServletResponse
    response)
```

- Si la méthode utilisée est POST, il suffit de redéfinir la méthode :

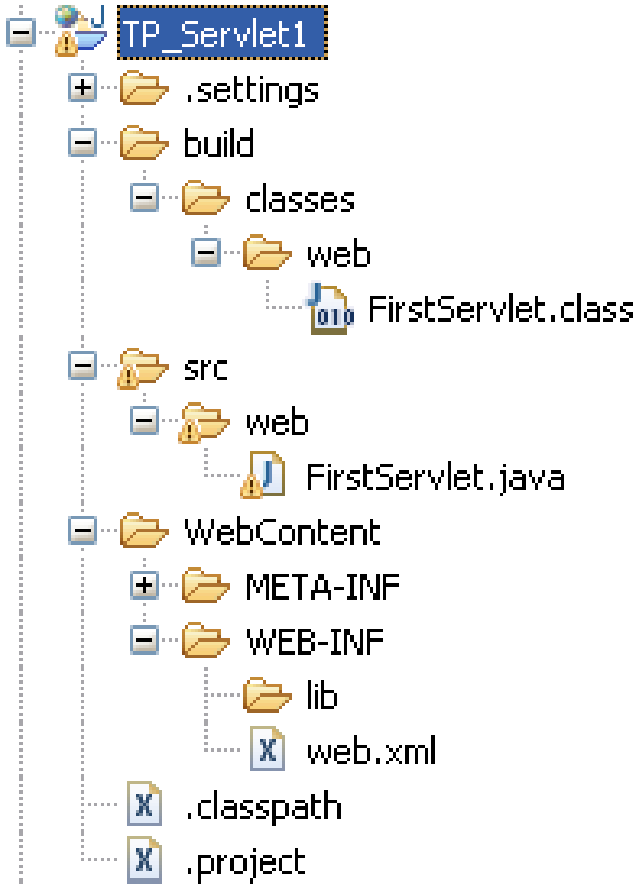
```
public void doPost(
    HttpServletRequest request, HttpServletResponse
    response)
```

Première Servlet

```
 package web;
Import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;
Public class FirstServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,  HttpServletResponse    response)
throws  ServletException, IOException {
                        response.setContentType("text/html");
                        PrintWriter out =response.getWriter();
                        out.println("<HTML>");
                        out.println("<HEAD><TITLE> Titre </TITLE></HEAD>");
                        out.println("<BODY>");
                        out.println("Ma première servlet");
                        out.println("</BODY>");
                        out.println("</HTML>");  out.close();
        }
}
```

Structure d'un projet Web Java EE (sous Eclipse)

- Le dossier src contient les classes java
- Le byte code est placé dans le dossier build/classes
- Les dossier WebContent contient les documents Web comme les pages HTML, JSP, Images, Java Script, CSS …
- Le dossier WEB-INF contient les descripteurs de déploiement comme web.xml
- Le dossier lib permet de stocker les bibliothèques de classes java (Fichiers.jar)



Déploiement d'une servlet

- Pour que le serveur Tomcat reconnaisse une servlet, celle-ci doit être déclarée dans le fichier web.xml qui se trouve dans le dossier WEB-INF.
- Le fichier web.xml s'appelle le descripteur de déploiement de Servlet.
- Ce descripteur doit déclarer principalement les éléments suivant :
  - Le nom attribué à cette servlet
  - La classe de la servlet
  - Le nom URL à utiliser pour faire appel à cette servlet via le protocole HTTP.

web.xml

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

<display-name>TP_Servlet1</display-name>

<servlet>
  <servlet-name>FirstServlet</servlet-name>

  <servlet-class>web.FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/fs</url-pattern>
</servlet-mapping>

</web-app>
```

Balise de description de l'application WEB

Nom de la Servlet "Identification"

Classe de la Servlet

Définition d'un chemin virtuel

Nom de la Servlet considér "Identification"

URL associée à la servlet

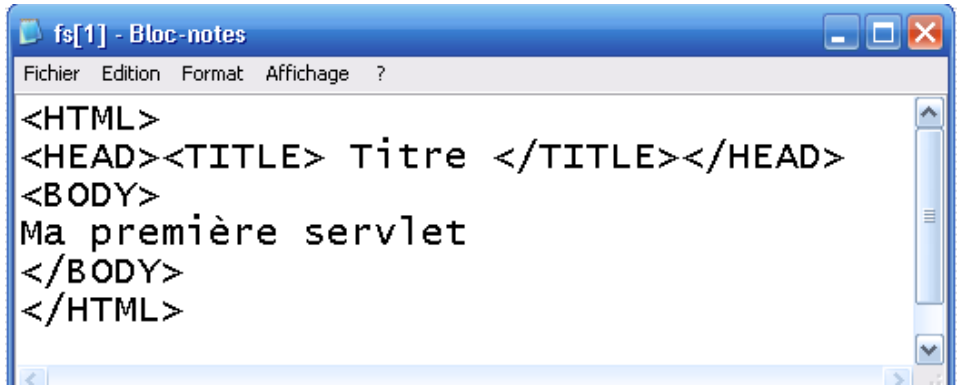
Servlet 3.1

- Pour un projet web Java EE, utilisant un module web, à partir de la version 3.0, le fichier web.xml n'est pas obligatoire pour configurer une Servlet.
- Dans ce cas, le déploiement d'une servlet peut se faire en utilisant des annotations:

```
package web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
@WebServlet(name="cs",urlPatterns={"/fs","*.do"})
public class FirstServlet extends HttpServlet {

}
```

Tester la servlet



Code source coté client

Partie 3

- Les page JSP

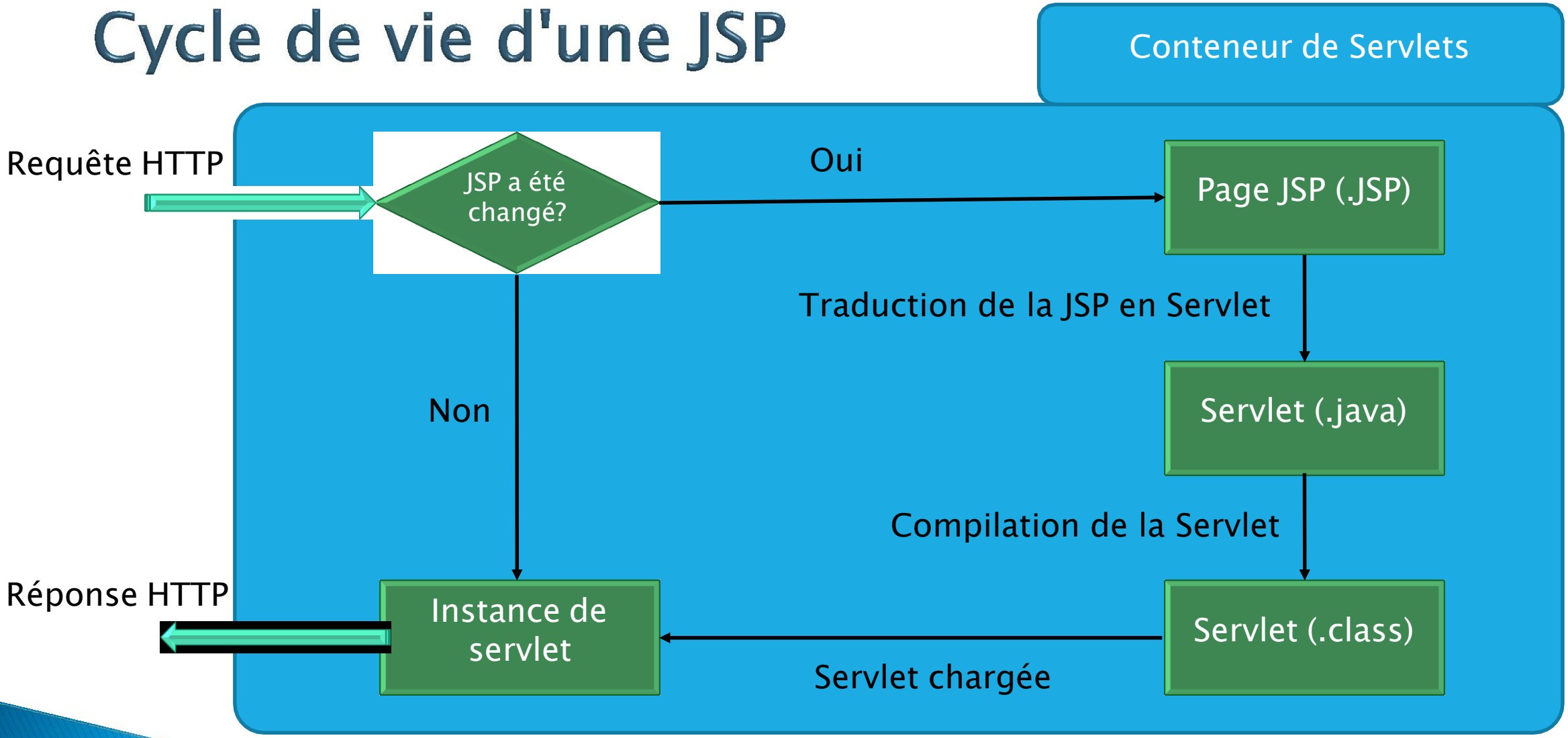
Introduction au page JSP

- Les pages JSP sont une des technologies de la plate-forme Java EE les plus puissantes, simples à utiliser et à mettre en place.
- Le langage JSP combine à la fois les technologies HTML, XML, servlet et JavaBeans (un objet Java) en une seule solution permettant aux développeurs de créer des vues dynamiques.
- Le modèle MVC recommande une séparation nette entre le code de contrôle et la présentation.
- Le modèle MVC recommande une séparation nette entre le code métier et la présentation.

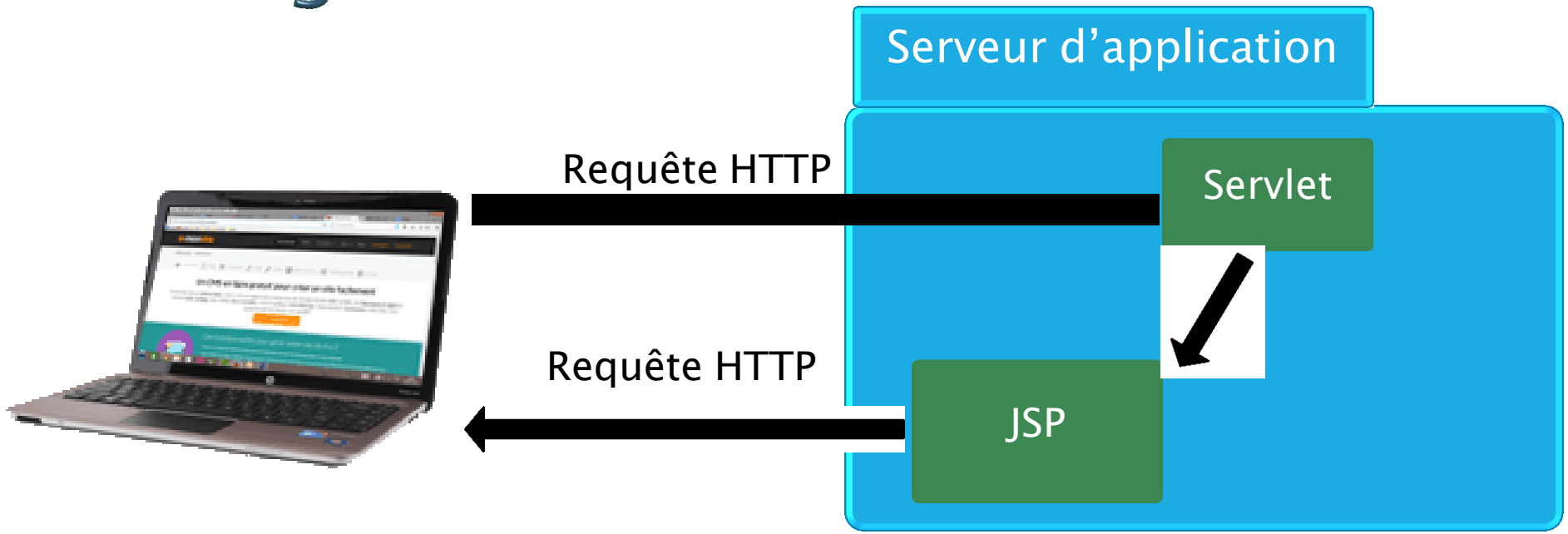
Introduction au page JSP

- les pages JSP sont des documents au format texte, à l'opposé des classes Java que sont les servlets. Elles contiennent des balises qui combinent à la fois simplicité et puissance, via une syntaxe simple, semblable au HTML et donc aisément compréhensible par un humain ;
- la simplicité d'accès aux objets Java;
- des mécanismes permettant l'extension du langage utilisé au sein des pages JSP.

Cycle de vie d'une JSP



Relation d'une page JSP avec la servlet: Forwarding




```
public void doGet( HttpServletRequest request, HttpServletResponse response )
throws ServletException, IOException {
   request.getRequestDispatcher( "/WEB-INF/test.jsp" ).forward( request,
   response );
}
```

Transmission de données

- L’objet de requête (HttpServletRequest):
  - setAtribute();
  - getAtribute();

```
public void doGet( HttpServletRequest request, HttpServletResponse response
) throws ServletException, IOException{
        String message = "Transmission de variables";
        request.setAttribute( "message", message );
        request.getRequestDispatcher( "/WEB-INF/test.jsp" ).forward(request,
        response );
}
```

Transmission de données

```
<%@ page pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8" />
		<title>Test</title>
	</head>
	<body>
		<p>Ceci est une page générée depui .</p>
		<p>
		<%
			String attribut = (String) request.getAttribute("message");
			out.println( attribut );
		%>
		</p>
	</body>
</html>
```

la méthode getAttribute() renvoie un objet global de type Object=>Transtypage(casting)

Données issues du client : les paramètres

- Ils sont envoyés par le client au serveur directement au sein de l'URL
- Il est uniquement possible d'y accéder en lecture
- L'objet de requête (HttpServletRequest):
  - getParameter(); // retourne un String

http://localhost:8080/test/index.do?nom=mohamed

```
public void doGet( HttpServletRequest request, HttpServletResponse response ) throws ServletException, IOException{
        String nom= request.getParameter("nom");
        String login="Bienvenue " + nom
        request.setAttribute( "login", login);
        request.getRequestDispatcher( "/WEB-INF/test.jsp" ).forward(request, response );
}
```

Différence entre les paramètres et les attributs

- Un attribut de requête est en réalité un objet stocké dans l'objet HttpServletRequest, et peut contenir n'importe quel type de données.
- Les attributs de requête sont utilisés pour permettre à une servlet de transmettre des données à d'autres servlets ou à des pages JSP.
- Un paramètre de requête est une chaîne de caractères placée par le client à la fin de l'URL de la requête HTTP.
- Les paramètres de requête sont utilisés pour permettre à un client de transmettre des données au serveur.

Partie 4

- JavaBean

C’est quoi un Bean?

- Le JavaBean, Souvent raccourci en "bean", désigne un composant réutilisable.
- Un bean est un simple objet Java qui suit certaines contraintes, et représente généralement des données du monde réel.
- Un bean peut par exemple être transmis d'une servlet vers une page JSP (ou une autre servlet) en tant qu'attribut de requête.

C’est quoi un Bean?

- Un bean :
  - doit être une classe publique ;
  - doit avoir au moins un constructeur par défaut, public et sans paramètres. Java l'ajoutera de lui-même si aucun constructeur n'est explicité ;
  - peut implémenter l'interface Serializable, il devient ainsi persistant et son état peut être sauvegardé ;
  - ne doit pas avoir de champs publics ;
  - peut définir des propriétés (des champs non publics), qui doivent être accessibles via des méthodes publiques getter et setter, suivant des règles de nommage.

Structure d'une Bean

```
public class Client {
    private String nom;
    private String prenom;
    private String email;
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }}
```

Eléments syntaxiques d'une JSP

Une page JSP peut être formée par les éléments suivants :

- Les expressions
- Les déclarations
- Les directives
- Les scriptlets
- Les actions
- Les JSTL

Expressions

- Les expressions JSP sont, des expressions Java qui vont être évaluées à l'intérieur d'un appel de méthode *print*.
- Une expression commence par les caractères <%= et se termine par les caractères %>.
- Comme l'expression est placée dans un appel de méthode, il est interdit de terminer l'expression via un point-virgule.

Syntaxe: <%=expression%>
Equivalent à: out.println(expression) ;
Exemple : <%=new Date()%>
Equivalent à: out.println(new Date()) ;

Déclarations

- Dans certains cas, un peu complexe, il est nécessaire d'ajouter des méthodes et des attributs à la servlet qui va être générée (en dehors de la méthode de service).
- Une construction JSP particulière permet de répondre à ces besoins. Elle commence par les caractères *<%!* et se termine, par les caractères *%>*. Voici un petit exemple d'utilisation.
- Exemple:

```
<%@ page language="java" %>
<HTML>
<%! private int userCounter = 0; %>

<BODY>
Vous êtes le <%= ++userCounter %><SUP>ième</SUP> client du site</P>
</BODY><HTML>
```

Directives

- Une directive permet de spécifier des informations qui vont servir à configurer et à influer sur le code de la servlet générée.
- Ce type de construction se repère facilement étant donné qu'une directive commence par les trois caractères <%@.
- Notons principalement deux directives :
  - *<%@ page ... %>*
  - *<%@ include ... %>*
  - *<%@ taglib ... %>*
- Voyons de plus près quelques unes des possibilités qui vous sont offertes.

Directive <%@ page .. %>

- La directive page définit des informations relatives à la page JSP. Voici par exemple comment importer des classes Java;
- En fait, cette directive accepte de nombreux paramètres dont les principaux sont:
  - <%@ page language="java" %>
  - <%@ page import="package|classe %>
  - <%@ page session="true|false" %>
  - <%@ page extends="classe" %>
  - <%@ page errorPage="url" %>
  - <%@ page isErrorPage="true|false"%>

Directive <%@ include .. %>

- La directive *<%@ include ... %>* est très utile si plusieurs pages se doivent de partager une même ensemble d'information.
- C'est souvent le cas avec les entêtes et les pieds de pages. Dans ce cas, codez ces parties dans des fichiers séparés et injectez les, via cette directive, dans tous les autre fichiers qui en ont besoin.
  - Voici un petit exemple d'utilisation de cette directive:

```
<%@ page language="java" %>
<HTML>
<BODY>
<%@ include file="header.jsp" %>
<!-- Contenu de la page à générer -->
<%@ include file="footer.jsp" %>
</BODY><HTML>
```

Scriptlets

- Les scriptlets correspondent aux blocs de code introduit par le caractères <% et se terminant par %>.
- Ils servent à ajouter du code dans la méthode de service.
- Le code Java du scriptlet est inséré tel quel dans la servlet générée: la vérification, par le compilateur, du code aura lieu au moment de la compilation totale de la servlet équivalent.
- L'exemple complet de JSP présenté précédemment, comportait quelques scriptlets :

```
<%
for(int i=1; i<=6; i++) {
 out.println("<H" + i + " align=\"center\">Heading " +i+  "</H" + i + ">");
}
%>
```

Les actions

- Les actions constituent une autre façon de générer du code Java à partir d'une page JSP.
- Les actions se reconnaissent facilement, syntaxiquement parlant : il s'agit de tag XML ressemblant à <jsp:tagName ... />.
- Enfin, le tag peut, bien entendu comporter plusieurs attributs.

Action jsp:forward

- Cette action vous permet de rediriger une requête vers une autre page au niveau du serveur.
- Syntaxe:

```
<jsp:forward page="relativeURL"/>
ou
<jsp:forward page="relativeURL" >
        <jsp:param name="parameterName" value="parameterValue"/>
</jsp:forward>
```

```
<% request.getRequestDispatcher(
"/page.jsp" ).forward( request, response ); %>
```

On récupère les paramètres par la méthode getParameter() de l'objet request

Action jsp:include

- Comme vu précédemment, l'action include vous permet d'insérer un fichier de manière "dynamique".

  <jsp:include page="relative URL" />

  <% request.getRequestDispatcher("page.jsp").include(request, response); %>

- Contrairement à la directive include qui insère les fichiers au moment où le fichier JSP est compilé sur le serveur et transformé en servlet, l'action include insère les fichiers seulement lorsqu'une requête est placée sur la page demandée. Ceci en coûte une légère perte en performance mais gagne en flexibilité.

Action jsp:useBean

- Le tag <jsp:useBean> permet de localiser une instance ou d'instancier un bean pour l'utiliser dans la JSP.

```
<%
com.beans.Etudiant etudiant=null;
etudiant= (com.beans.Etudiant) request.getAttribute( " etudiant " );
if (etudiant == null ){
etudiant = new com.beans.Etudiant();
request.setAttribute( " etudiant ", etudiant );
} %>
```

L'attribut id permet de donner un nom à la variable qui va contenir la référence sur le bean.

L'attribut class permet d'indiquer la classe du bean.

L'attribut scope permet de définir la portée durant laquelle le bean est défini et utilisable

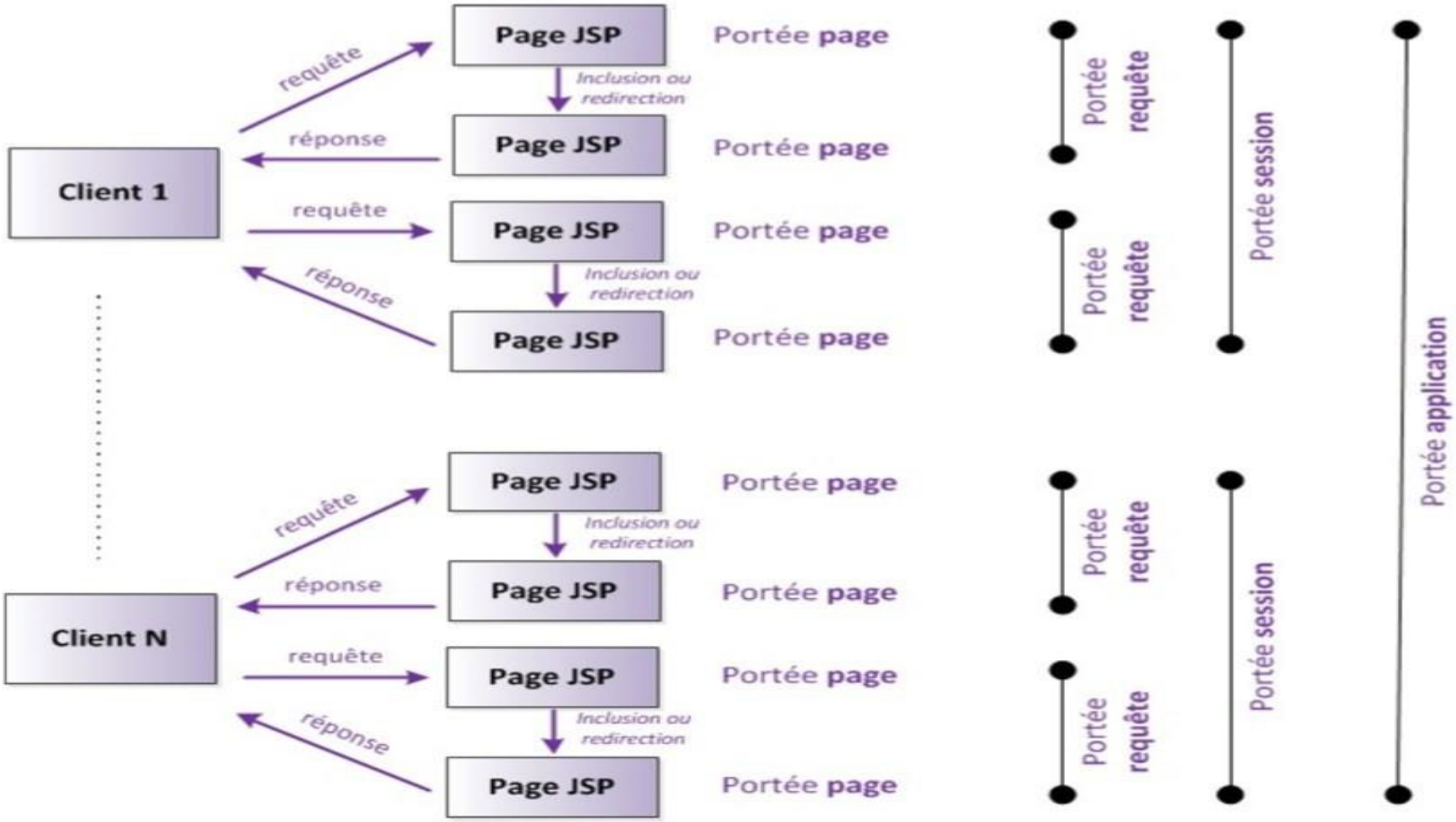
```
<jsp:useBean id="etudiant" class="com.midvi.web.beans.Etudiant" scope="request" />
```

La portée des objets

- L'attribut scope permet de définir la portée durant laquelle le bean est défini et utilisable. La valeur de cette attribut détermine la manière dont le tag localise ou instancie le bean. Les valeurs possibles sont :

Valeur	Rôle
page	Le bean est utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus. C'est la valeur par défaut.
request	le bean est accessible durant la durée de vie de la requête. La méthode getAttribute() de l'objet request permet d'obtenir une référence sur le bean.
session	le bean est utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instanciée le bean. Le bean est utilisable tout au long de la session par toutes les pages qui y participent.
application	le bean est utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le bean.

La portée des objets



Exemple

TestBean.jsp

```
<html>
<HEAD>
<TITLE>Essai d'instanciation d'un bean dans une
JSP</TITLE>
</HEAD>
<body>
<p>Test d'utilisation d'un Bean dans une JSP </p>
<jsp:useBean id="personne" scope="request"
class="test.Personne" />
<p>nom initial = <%=personne.getNom() %></p>
<%
personne.setNom("mon nom");
%>
<p>nom mise à jour = <%= personne.getNom()
%></p>
</body>
</html>
```

Personne.java

```
package test;
public class Personne {
private String nom;
private String prenom;
public Personne() {
this.nom = "nom par defaut";
this.prenom = "prenom par defaut";
}
public void setNom (String nom) {
this.nom = nom;
}
public String getNom() {
return (this.nom);
}
public void setPrenom (String prenom) {
this.prenom = prenom;
}
public String getPrenom () {
return (this.prenom);
}
}
```

Action jsp:getProperty

- Le tag <jsp:getProperty> permet d'obtenir la valeur d'une des propriétés d'un bean.

```
<jsp:useBean id="etudiant" class="com.midvi.web.beans.Etudiant" scope="request" />
```

```
<jsp:getProperty name="etudiant" property="nom"/>
```

Même effet que
<%= etudiant.getPrenom() %>

Action jsp:setProperty

- Le tag <jsp:setProperty> permet de modifier une propriété du bean utilisé.

```
<jsp:useBean id="etudiant" class="com.midvi.web.beans.Etudiant" scope="request" />
```

```
<jsp:setProperty name=“etudiant" property="prenom" value=“mohamed" />
```

Même effet que
<%= etudiant.setPrenom(« mohamed");%>

Expression Langage (EL)

- La technologie EL est fondée sur les JavaBeans et sur les collections Java, et existe depuis la version 2.4 de l'API Servlet.
- Les expressions EL remplacent les actions standard de manipulation des objets.
- Une expression EL permet d'effectuer des tests, interprétés à la volée lors de l'exécution de la page.
- la syntaxe d'une expression EL:

```
${ expression }
```

Réalisation de tests

- opérateurs arithmétiques, applicables à des nombres : +, −, *, /, % ;

```
${ 10 / 4 }          ${ 10 mod 4 }
```

- opérateurs logiques, applicables à des booléens : &&, ||, ! ;

```
${ !true || false }
```

- opérateurs relationnels, basés sur l'utilisation des méthodes equals() et compareTo() des objets comparés : == ou eq, != ou ne, < ou lt, > ou gt, <= ou le, >= ou ge.

```
<p>12 est inférieur à 8 : ${ 12 lt 8 }.</p>
```

Réalisation de tests

- Deux autres types de test sont fréquemment utilisés au sein des expressions EL :
- les conditions ternaires, de la forme : test ? si oui : sinon ;

```
${ 'a' > 'b' ? 'oui'  : 'non' }
```

- les vérifications si vide ou null, grâce à l'opérateur empty.

```
${ empty 'test' ? 'vide' : 'non  vide'  }
```

Manipulation d'objets:beans

- pour accéder à une propriété d'un bean d'une manière simple et efficace:

${ bean.propriete }

```
<jsp:useBean id="etudiant"
class="com.beans.Etudiant" />
<jsp:getProperty name="
etudiant" property="nom" />
```

${etudiant.nom}

où

${etudiant.getNom()}

Manipulation d'objets: collections

```
<%
   java.util.List<String> modules= new
   java.util.ArrayList<String>();
   modules.add( "Java EE" );
   modules.add( "programmation mobile" );
   modules.add( "web sémantique");
   modules.add( "Web dynamique" );
   request.setAttribute( "modules" , modules);
%>
```

```
${ modules.get(1) }
${modules[1] }
${modules['1'] }
${modules["1"] }
```

Manipulation d'objets: tableaux

```
<%
   String[] modules= {"Java EE" ,
   "programmation mobile" , "web
   sémantique", "Web dynamique" };
   request.setAttribute( "modules" , modules);
%>
```

```
${modules[1] }
${modules['1'] }
${modules["1"] }
```

Manipulation d'objets: objets implicites

- Il existe deux types :
  - ceux qui sont mis à disposition via la technologie JSP ;
    - pageContext, application, session, request, response, exception, out, config, page.
  - ceux qui sont mis à disposition via la technologie EL:
    - il s'agit d'objets gérés automatiquement par le conteneur lors de l'évaluation des expressions EL.

Manipulation d'objets: objets implicites

Catégorie	Identifiant	Description
JSP	**pageContext**	Objet contenant des informations sur l'environnement du serveur.
Portées	**pageScope**	Une Map qui associe les noms et valeurs des attributs ayant pour portée la page.
	requestScope	Une Map qui associe les noms et valeurs des attributs ayant pour portée la requête.
	sessionScope	Une Map qui associe les noms et valeurs des attributs ayant pour portée la session.
	applicationScope	Une Map qui associe les noms et valeurs des attributs ayant pour portée l'application.
Paramètres de requête	**param**	Une Map qui associe les noms et valeurs des paramètres de la requête.
	paramValues	Une Map qui associe les noms et multiples valeurs ** des paramètres de la requête sous forme de tableaux de String.
En-têtes de requête	**header**	Une Map qui associe les noms et valeurs des paramètres des en-têtes HTTP.
	headerValues	Une Map qui associe les noms et multiples valeurs ** des paramètres des en-têtes HTTP sous forme de tableaux de String.
Cookies	**cookie**	Une Map qui associe les noms et instances des cookies.
Paramètres d'initialisation	**initParam**	Une Map qui associe les données contenues dans les champs `<param-name>` et `<param-value>` de la section `<init-param>` du fichier web.xml.

Manipulation d'objets: objets implicites

- pour param et header, une seule valeur est associée à chaque nom de paramètre, via une Map<String,String> ;

${param.langue}

Langue est un paramètre passé dans la requête

- pour paramValues et headerValues par contre, ce sont plusieurs valeurs qui vont être associées à un même nom de paramètre, via une Map<String,String[]>.

${paramValues.langue[1]}

http://localhost:8080/test/test.jsp?langue=fr&langue=ar