

Architecture des applications Web sous JEE JSP, Servlet et EJB

Conception et mise en œuvre

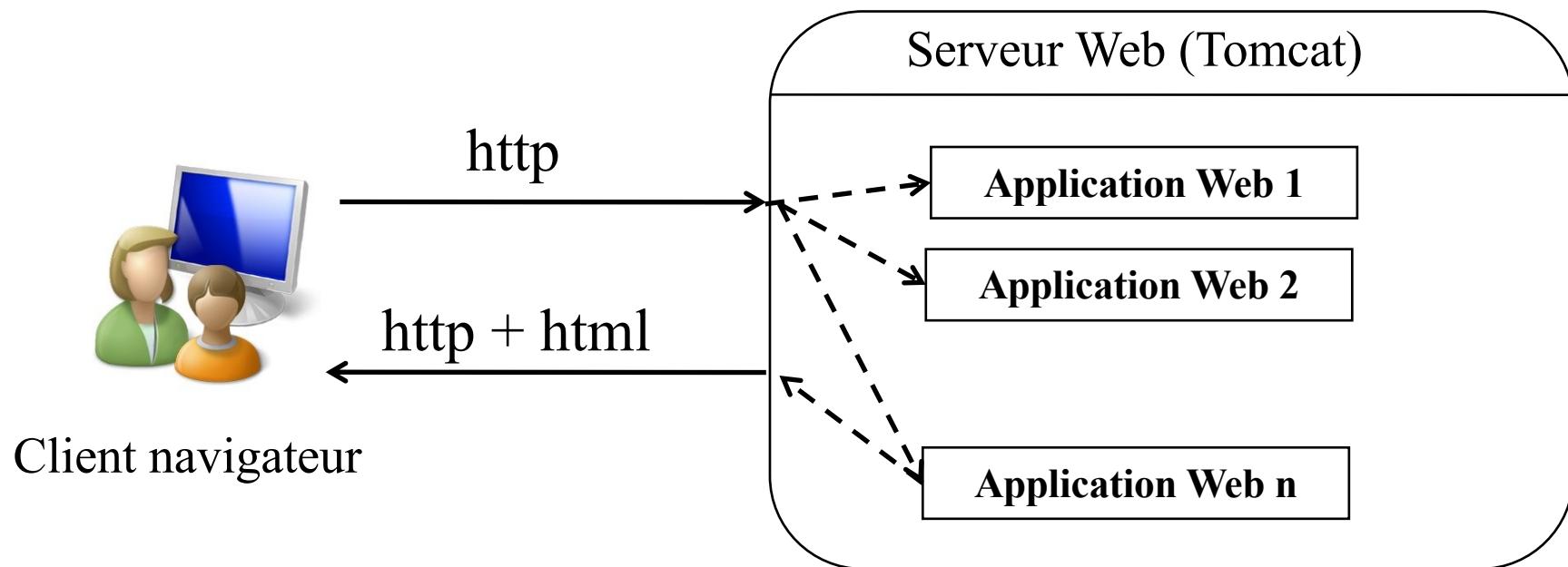
Licence professionnelle SIL

Mourad Ouziri

mourad.ouziri@parisdescartes.fr

Applications web

- ☞ Application accessible sur le Web, donc en HTTP
- ☞ Retourne des résultats visualisables par un navigateur Internet, donc en HTML/CSS/JS



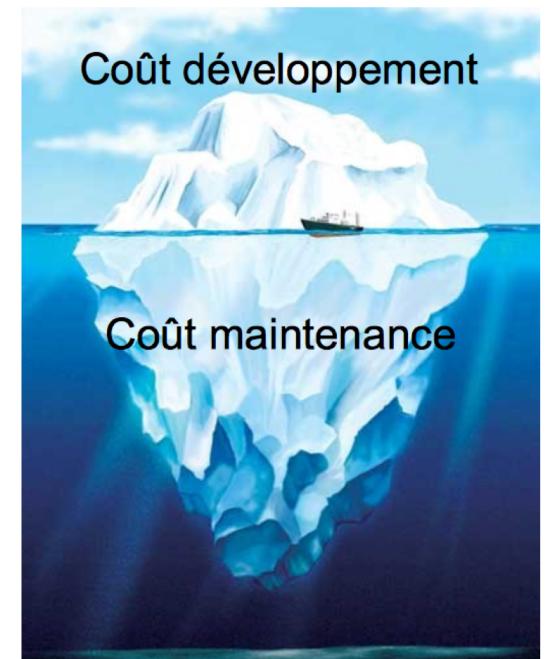
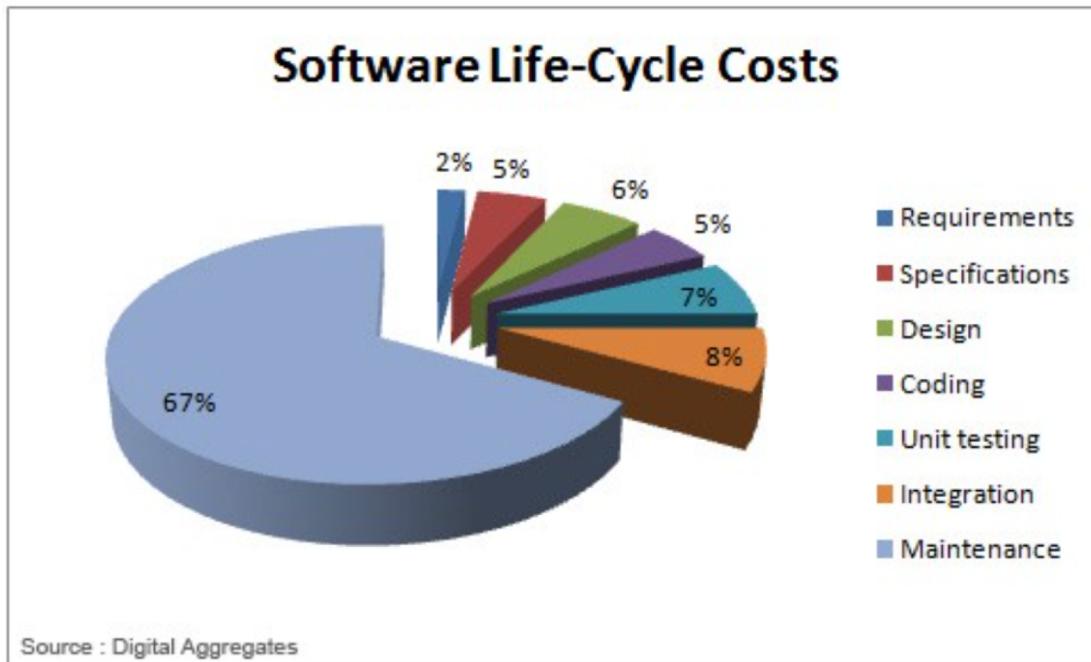
Qualité des applications

- ☞ Objectif : développer des applications de qualité
 - Plusieurs critères de qualité (ISO 9126, FURPS, etc.)
- ☞ Maintenabilité, un critère de qualité !
- ☞ Maintenance logicielle
 - Les changements qui doivent être apportés à un logiciel après sa livraison à l'utilisateur (*Martin et McClure 1983*)
 - La totalité des activités qui sont requises afin de procurer un support, au meilleur coût possible, d'un logiciel. Certaines activités débutent avant la livraison du logiciel, donc pendant sa conception initiale (*SWEBO 2005*)

Qualité des applications

☞ Maintenabilité, un facteur de qualité important !

- Evolution constante des attentes des utilisateurs et des technologies
- Coût moyen des applications très élevé
- Amortissement des coûts de développement



☞ Comment optimiser la maintenabilité ? **Architecture logicielle** !⁴

Architecture logicielle

☞ Définition

- Architecture d'une application = Structure de l'application
- Schéma des différents composants du logiciel informatique, leurs rôles et leurs interactions

☞ Eléments d'architecture

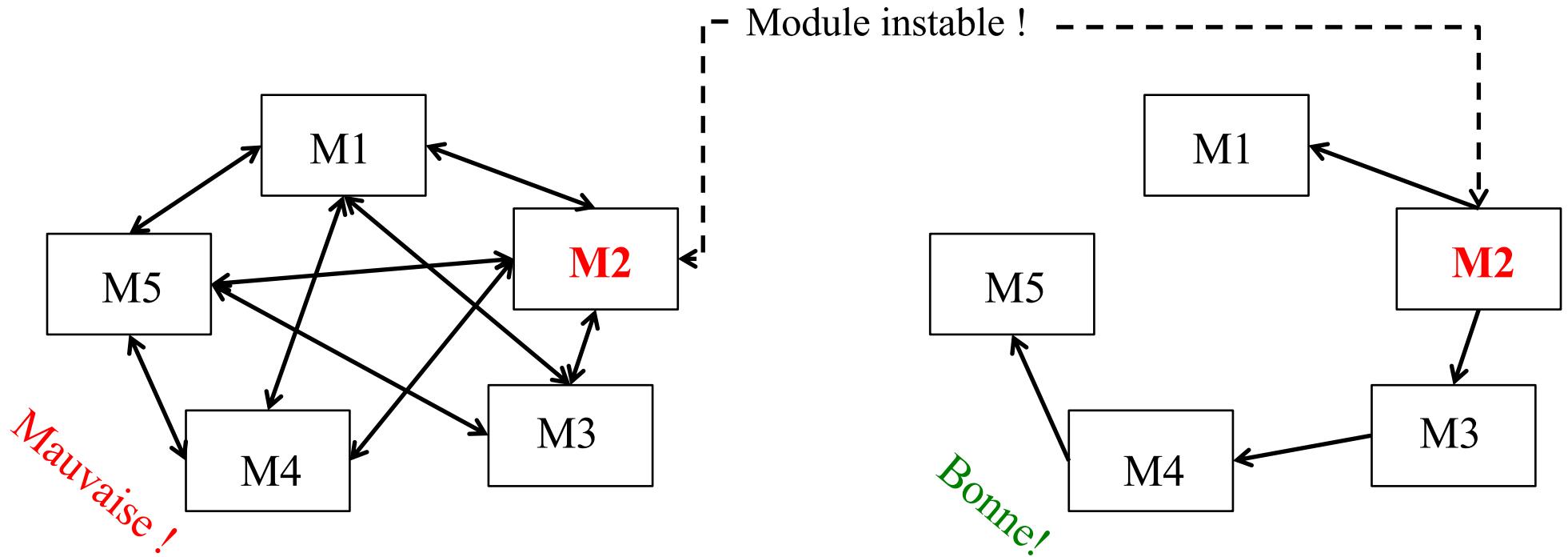
- Composants, rôles, dépendances/communication inter-composants

☞ Objectif

- Concevoir des applications robustes vis-à-vis des évolutions futures
- Concevoir des applications maintenables à moindre coût

Architecture logicielle

☞ Schéma d'architecture

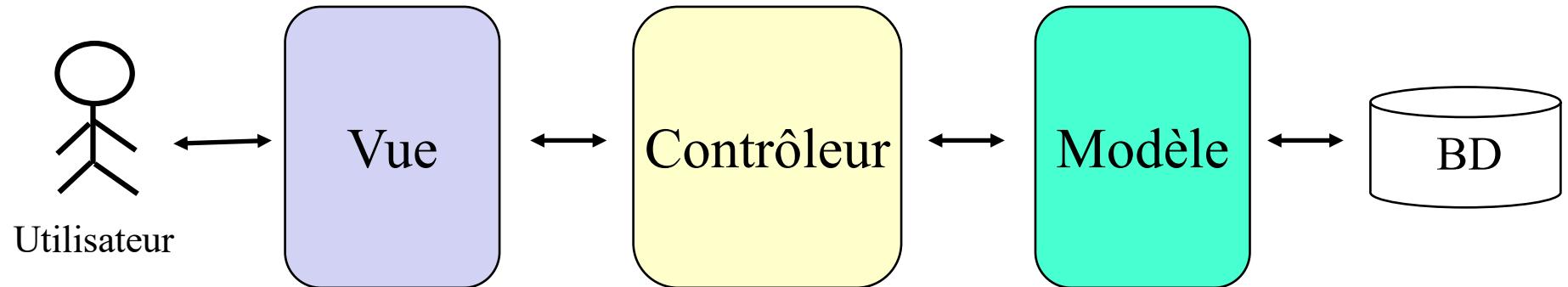


☞ Facteur de maintenabilité : amplification du changement par propagation !

– Coût = Coûts liés à M2 + Coûts des modules impactés par propagation (M1, M3, M4, M5)

Architecture MVC

Modèle-Vue-Contrôleur



☞ Rôles :

- Vue : interaction avec les utilisateurs (environnement extérieur de l'appli)
- Modèle : logique métier (ou le fonctionnel) de l'application
- Contrôleur : aiguillage des requêtes vers le Modèle et choix des Vue responsables de l'affichage des résultats.

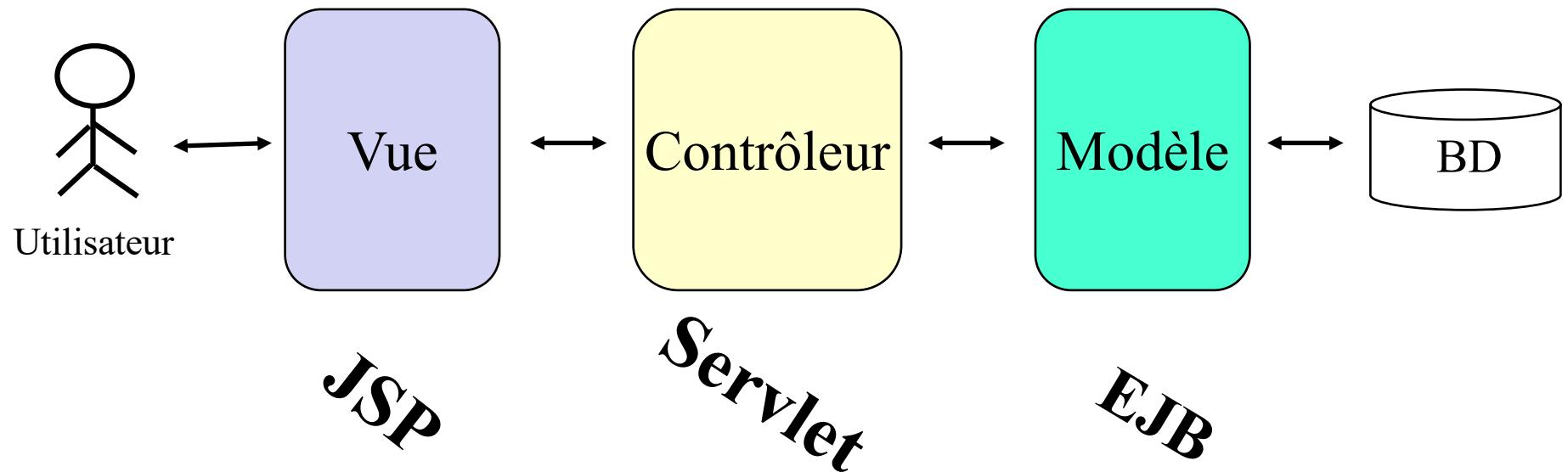
Protège le Modèle des dépendances (notamment de la Vue) !!!

☞ Maintenabilité

- Localisation des responsabilités (bogues et évolutions)
- Hypothèse de travail : stabilité du Modèle !

Architecture MVC

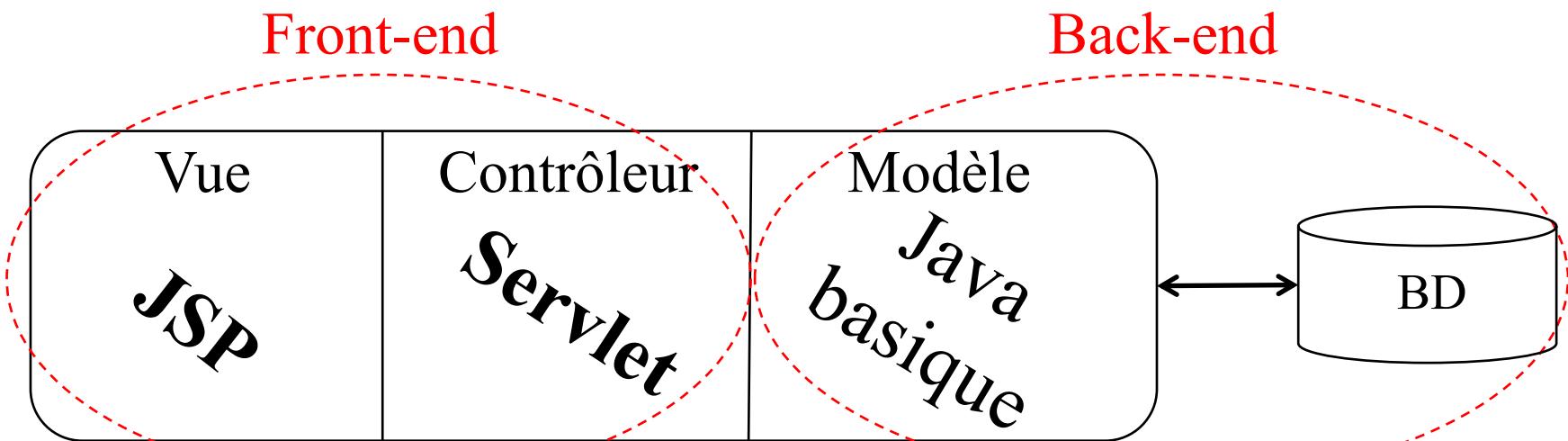
Modèle-Vue-Contrôleur



- ☞ JSP : langage de programmation de pages HTML dynamiques
- ☞ Servlet : traitements liés aux protocoles (notamment http avec HttpServlet)
- ☞ POJO (Plain Old Java Object) : programmation Java de base (J2SE)

Architecture MVC sous J2EE

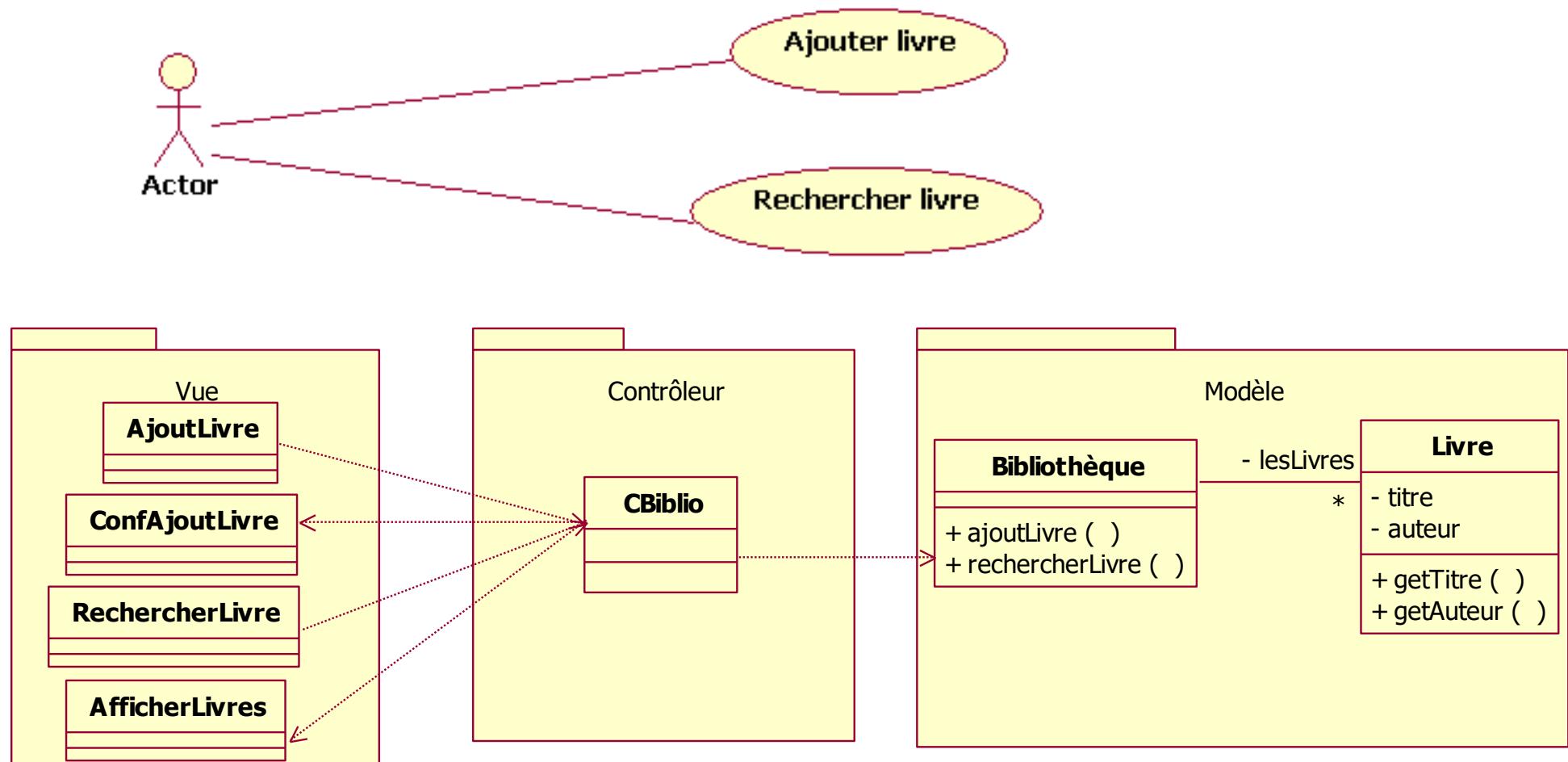
Partie 1 : Servlet et JSP



- ☞ Vue : JSP
- ☞ Contrôleur : Servlet
- ☞ Modèle : Java classique
 - Programmation Java basique – POJO (Plain Old Java Object)
 - Persistance, transactionnel, sécurité, concurrence !

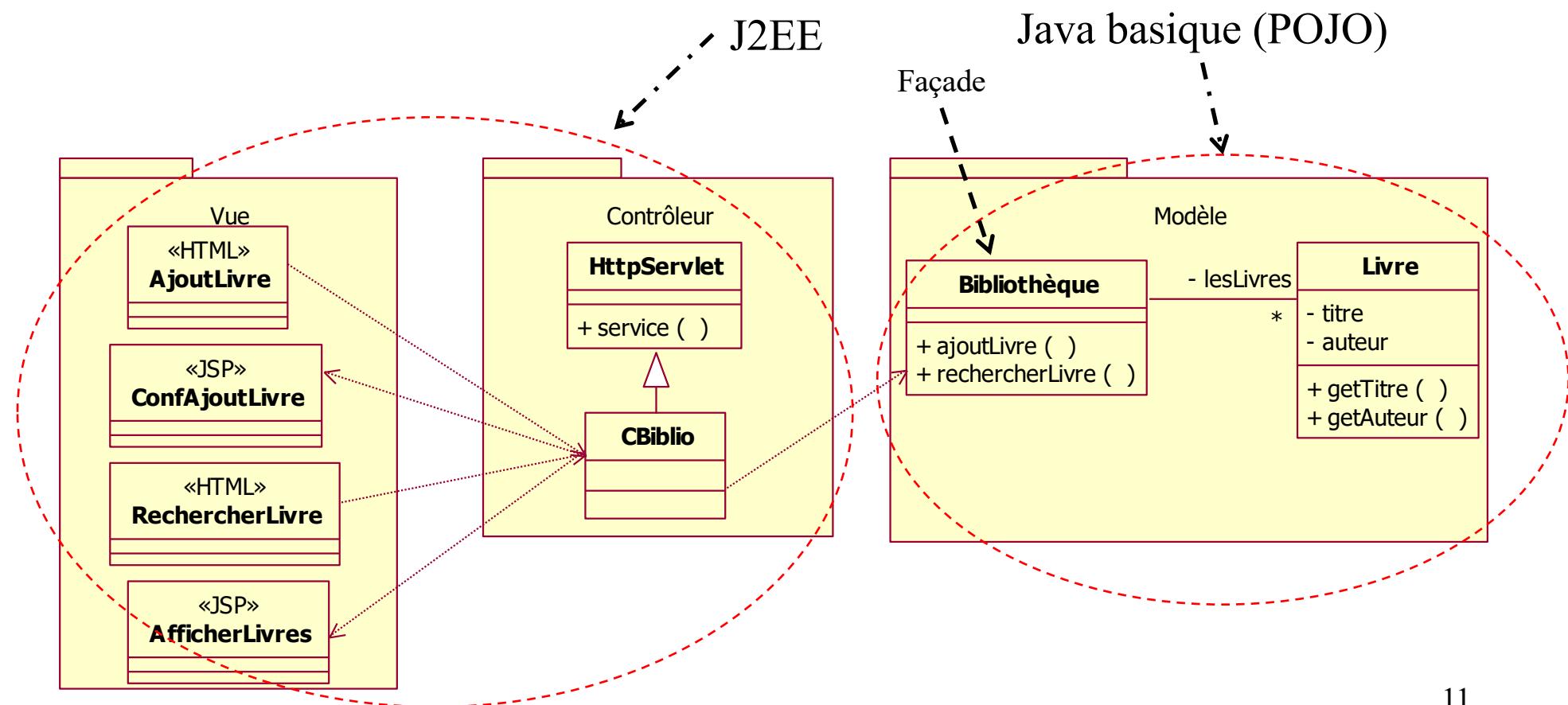
Analyse de l'architecture MVC avec UML

☞ Exemple : gestion simplifiée d'une bibliothèque



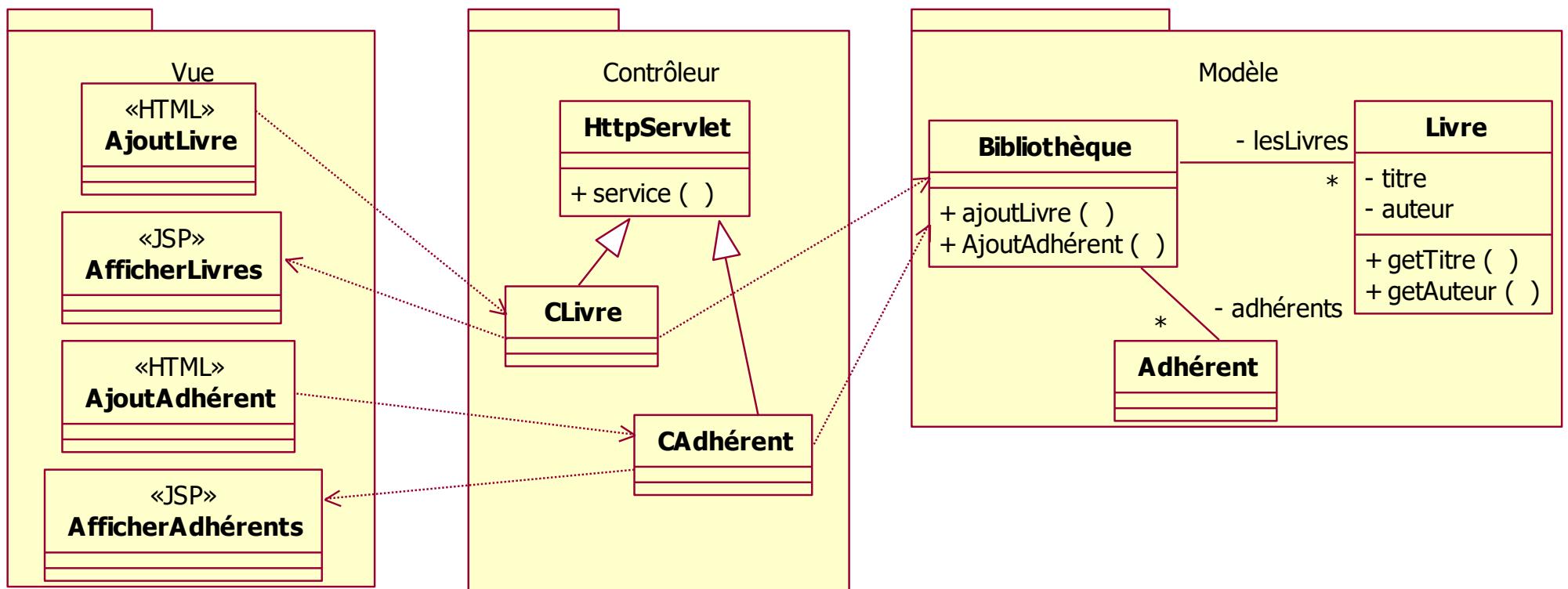
MVC avec Java, JSP et Servlet

☞ Architecture mono-contrôleur (MVC 2)



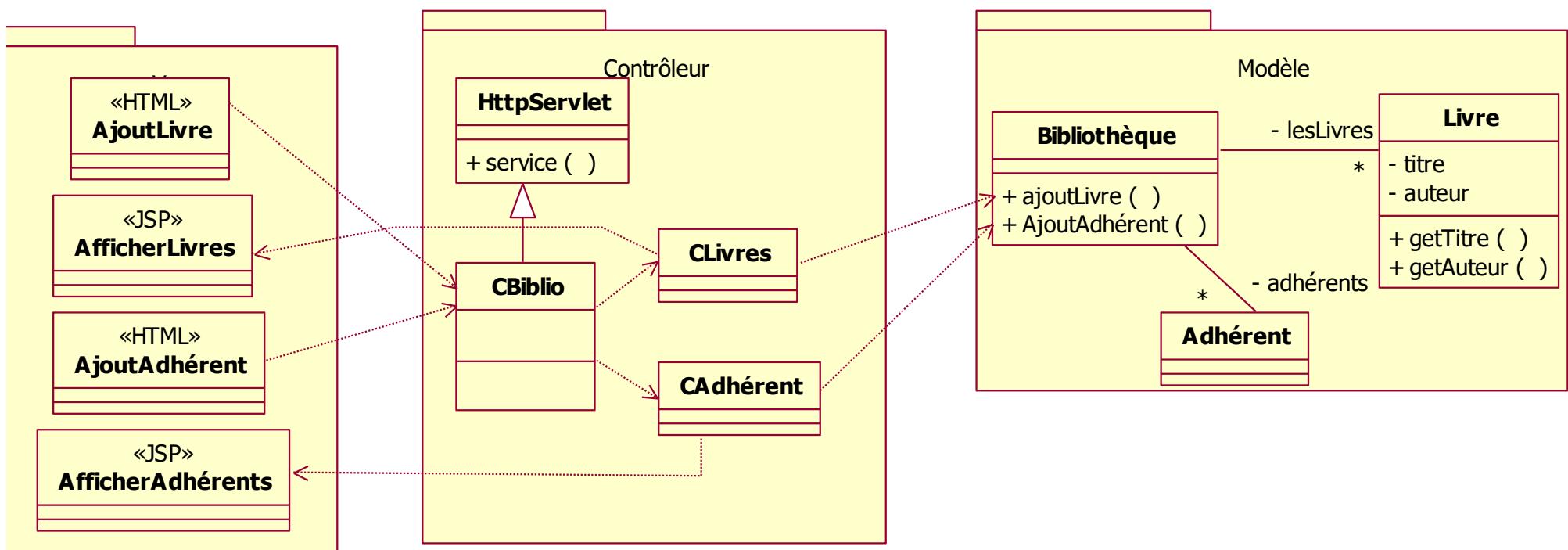
MVC avec Java, JSP et Servlet

☞ Architecture multi-contrôleurs (MVC 1)



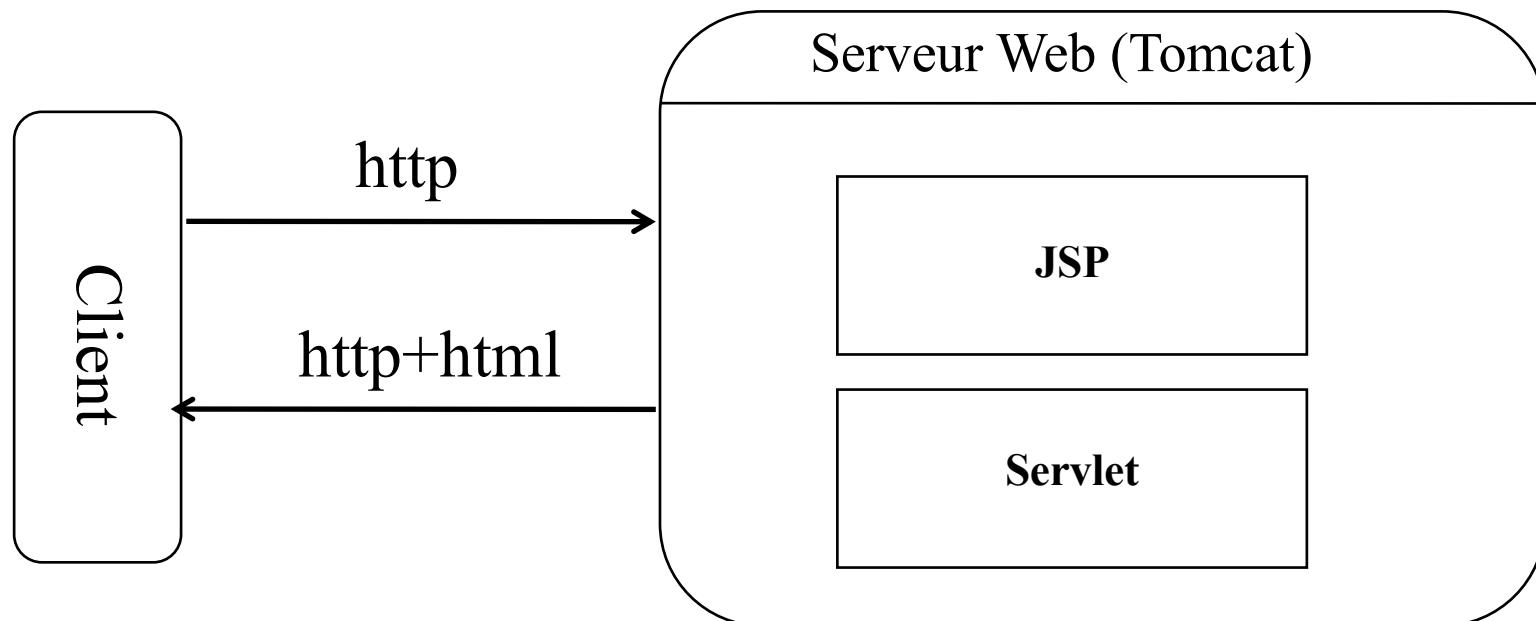
MVC avec Java, JSP et Servlet

☞ Architecture à contrôleurs mixtes



JSP et Servlet : rappels

- ☞ API (Servlet) et langage (JSP) de programmation côté **serveur**
- ☞ Pour le développement d'applications Web
- ☞ Accessibles via HTTP
- ☞ Résultats au format HTML



Rappel JSP

- ☞ JSP = Page HTML dynamique (incluant des variables !)
- ☞ Page JSP = Page HTML incluant des scripts java
- ☞ Le langage JSP est un langage de scripts
- ☞ Les scripts sont des instructions Java placée entre les balises <% %>
- ☞ Chaque script est interprété par un serveur JSP qui différencie le code JSP du code HTML grâce aux balises <% %>

Rappel JSP

☞ Les cinq principales balises du langage

1. Les balises de scriptlets permettant d'inclure du code java

`<% ... %>`

2. Les balises permettant d'évaluer une expression et afficher sa valeur dans la page `<%= ... %>`

3. Les balises de directives (classes importées, prise en compte de session, etc.) `<%@ ... %>`

4. Les balises permettant de définir des variables globales (partagées par toutes les requêtes http) à la page `<%! ... %>`

5. Les balises de commentaires `<%-- ... --%>`

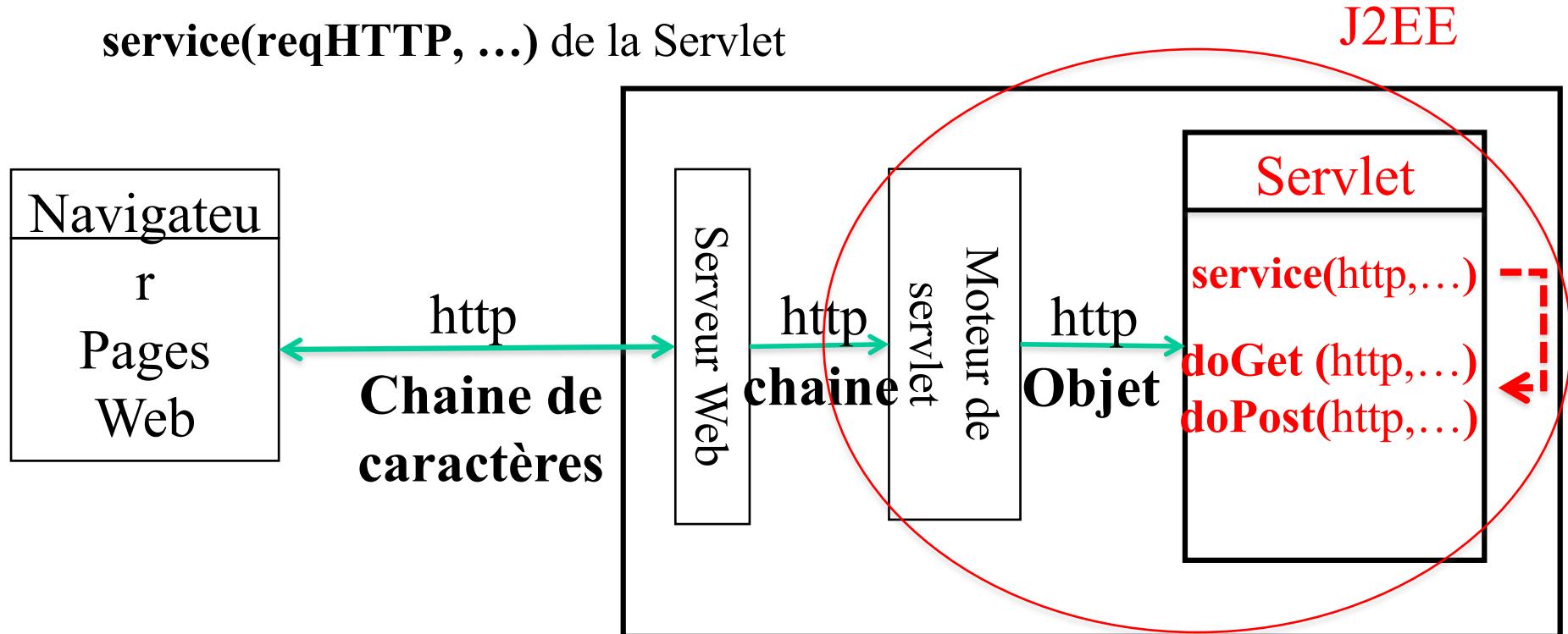
Rappel JSP

- ☞ Les balises scriptlets et d'affichage d'expressions

```
<%@ page import="java.util.Date" %>  
<% Date today = new java.util.Date();%>  
<html>  
  <body>  
    Bonjour, on est le <%=today.toString()%> !  
  </body>  
</html>
```

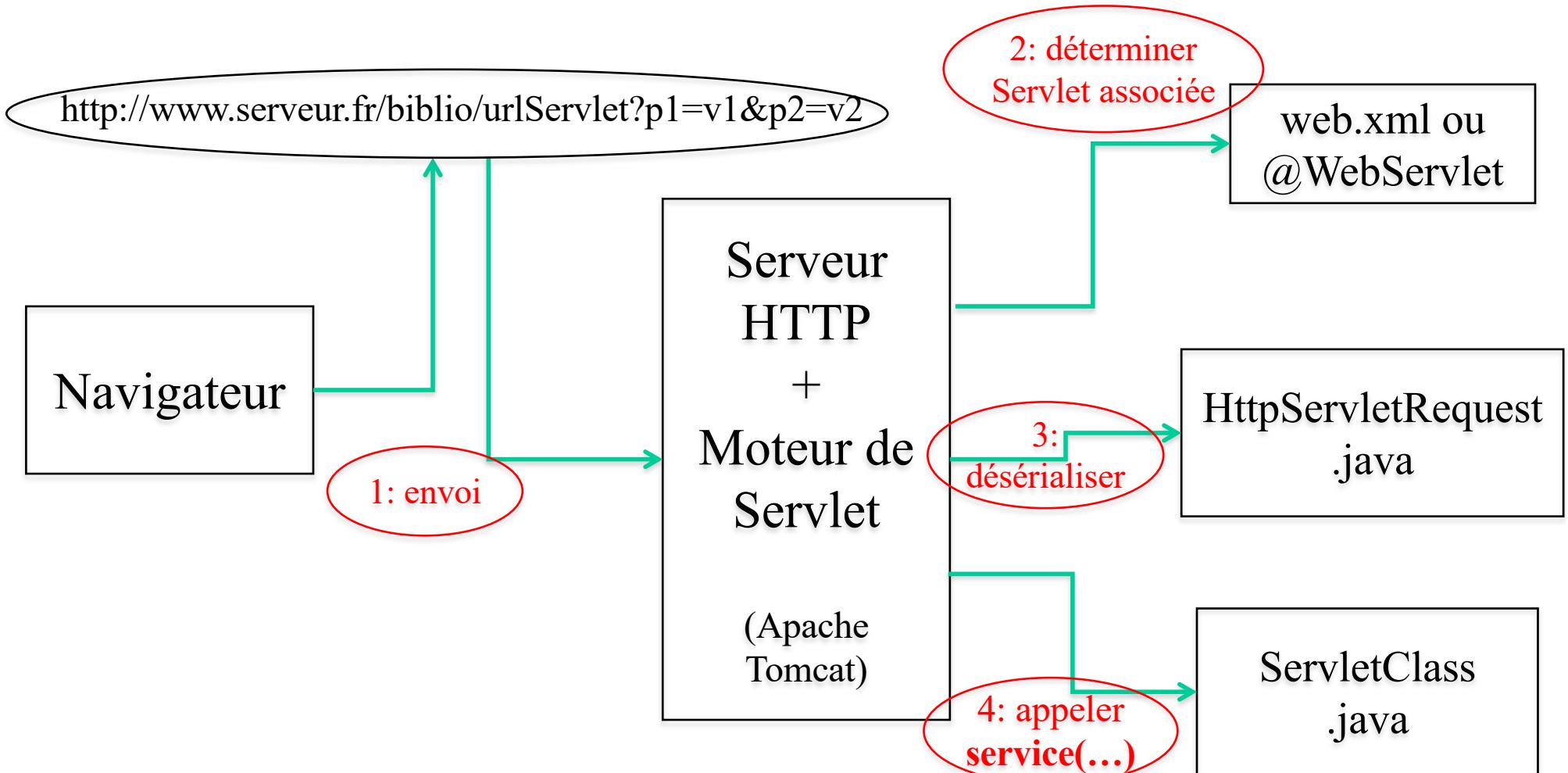
Rappel *Servlet*

- Moteur de Servlet : environnement d'exécution de Servlet
 - Offre les services génériques de traitements de requêtes HTTP : réception de requêtes, redirection, sérialisation/désérialisation
 - Transmet les requêtes HTTP à la Servlet : appelle la méthode **service(reqHTTP, ...)** de la Servlet



Rappel *Servlet*

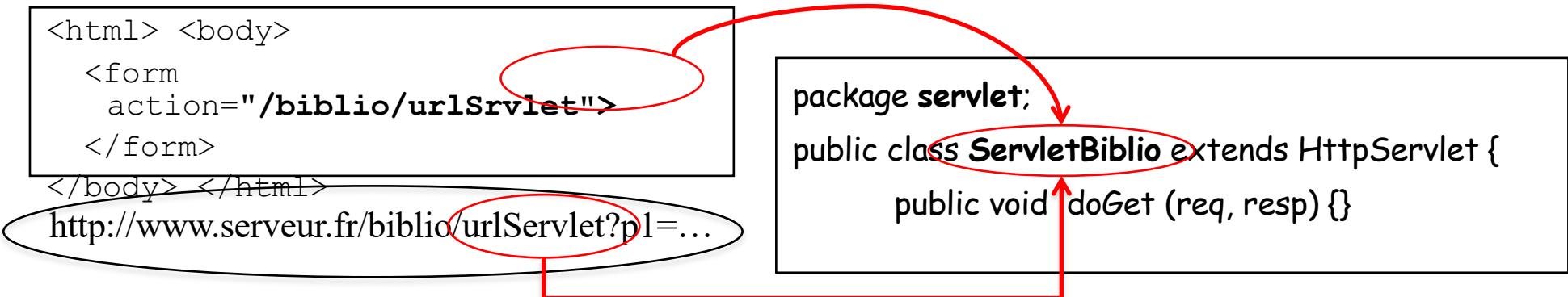
- Transmission des requêtes http à la *Servlet*



Rappel *Servlet*

Descripteur de déploiement

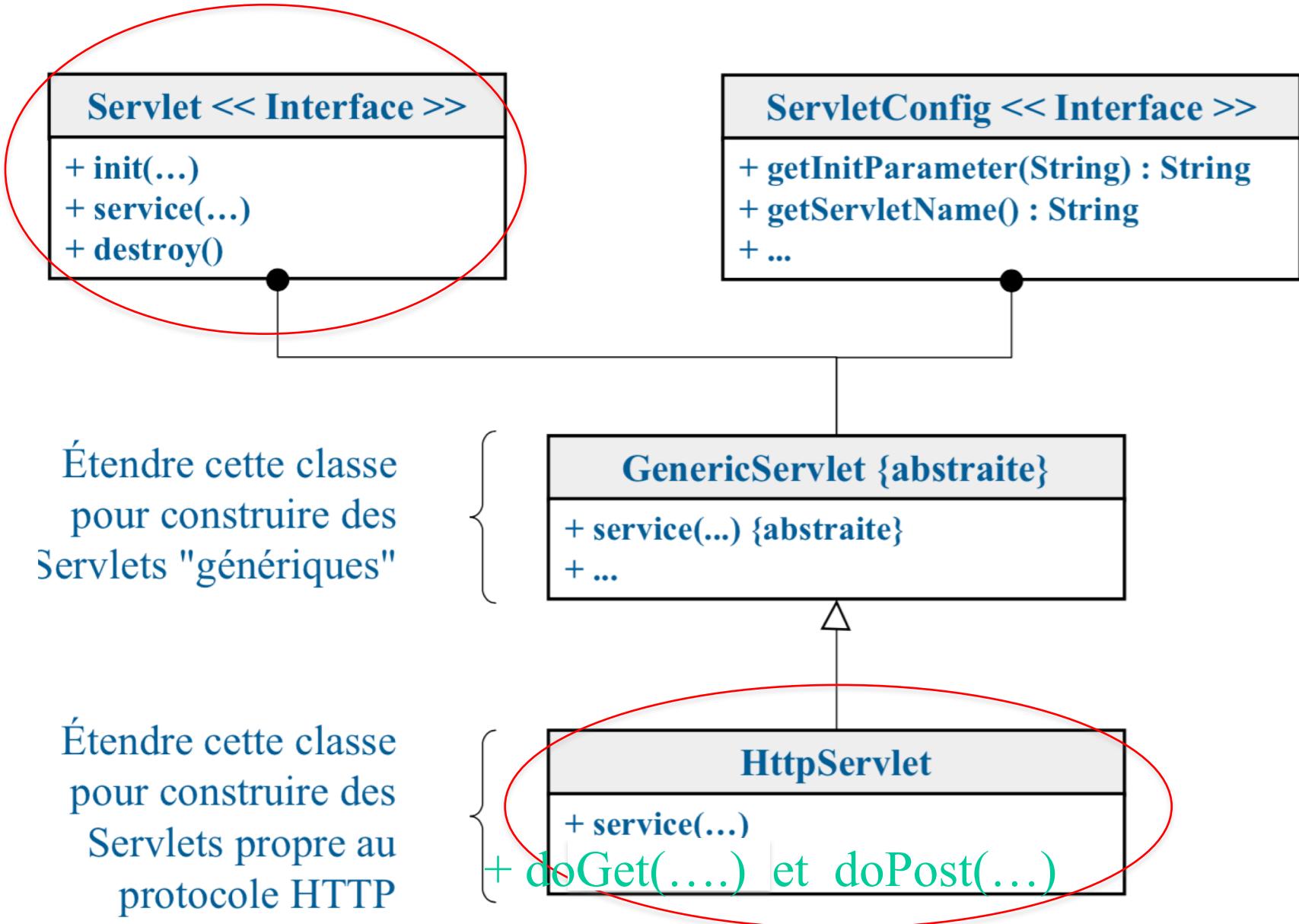
- Associer une URL à la Servlet associée : **web.xml**



☞ Fichier de déploiement de l' application Web : *web.xml*

```
<web-app>
  <servlet>
    <servlet-name> myServlet </servlet-name>
    <servlet-class> servlet.ServletBiblio </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name> myServlet </servlet-name>
    <url-pattern>/urlServlet </url-pattern>
  </servlet-mapping>
</web-app>
```

Rappel *Servlet* API



Rappel *Servlet* implémentation

http://www.serveur.fr/biblio/hello?nom=ouziri

```
@WebServlet ("hello")
public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        String nom = request.getParameter("nom");
        response.setContentType("text/html"); // format du contenu de la
réponse
        PrintWriter out = response.getWriter();
        out.println("<html>") // générer le résultat dynamique
            out.println("<body>");
            out.println( "Bonjour " + nom);
            out.println("</body>");
        out.println("</html>");
    }
}
```

Vue en JSP

☞ Nous distinguons deux types de vues

1. Vues requêtes :

Permettent l' envoi des requêtes (demandes) des utilisateurs

Transmettent la demande au contrôleur (Servlet)

Souvent implémentées en HTML pure

2. Vues réponses :

Permettent l' affichage des résultats pour l' utilisateur

Reçoivent du contrôleur (Servlet) le résultat à afficher

Souvent implémentées en JSP

Vue en JSP/HTML

```
<html>
  <body>
    <h1>Ajout d'un livre </h2>
    <form action="/bibliotheque/controleur" method="POST">
      Titre du livre : <input type="text" name="titreLivre" /> <br>
      Son auteur : <input type="text" name="auteurLivre" /> <br>

      <input type="hidden" name="operation" value="ajoutLivre" />
      <p> <input type="submit" value="Valider">
    </form>
  </body>
</html>
```

ajoutLivre.html

```
<html>
  <body>
    <h1>Rechercher un livre </h2>
    <form action="/bibliotheque/controleur" method="POST">
      Titre du livre : <input type="text" name="titreLivreRecherche" />
      <br>
      <input type="hidden" name="operation" value="rechercherLivre" />
      <input type="submit" value="Valider">
    </form>
  </body>
</html>
```

rechercherLivre.html

- ☞ Toutes les vues-requêtes utilisent le même paramètre (caché) pour s'identifier auprès du contrôleur

Vue en JSP

Communication JSP (Vue) → Servlet (Contrôleur)

☞ JSP (Vue) : Envoi d' une requête à la Servlet (Contrôleur)

1. Une requête HTTP est envoyée lorsqu' on clique sur **Submit**
2. Les données du formulaires sont transmises dans la requête HTTP sous forme de paramètres
3. Exemples : requêtes de type GET :
 - a. La vue **ajoutLivre.html**

```
http://url_du_serveur/bibliothèque/contrôleur?titreLivre=titre_sai  
si&auteurLivre=nom_saisi&opération=ajoutLivre
```

b. La vue **rechercherLivre.html**

```
http://url_du_serveur/bibliothèque/contrôleur?titreLivreRecherche=  
titre_saisi&opération=rechercherLivre
```

Contrôleur : HttpServlet

☞ Contrôleur : classe **HttpServlet**

```
public abstract class javax.servlet.http.HttpServlet {  
    public void service (HttpServletRequest req, HttpServletResponse resp) throws ... ;  
    protected void doPost (HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, java.io.IOException ;  
    protected void doGet (HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, java.io.IOException ;  
}
```

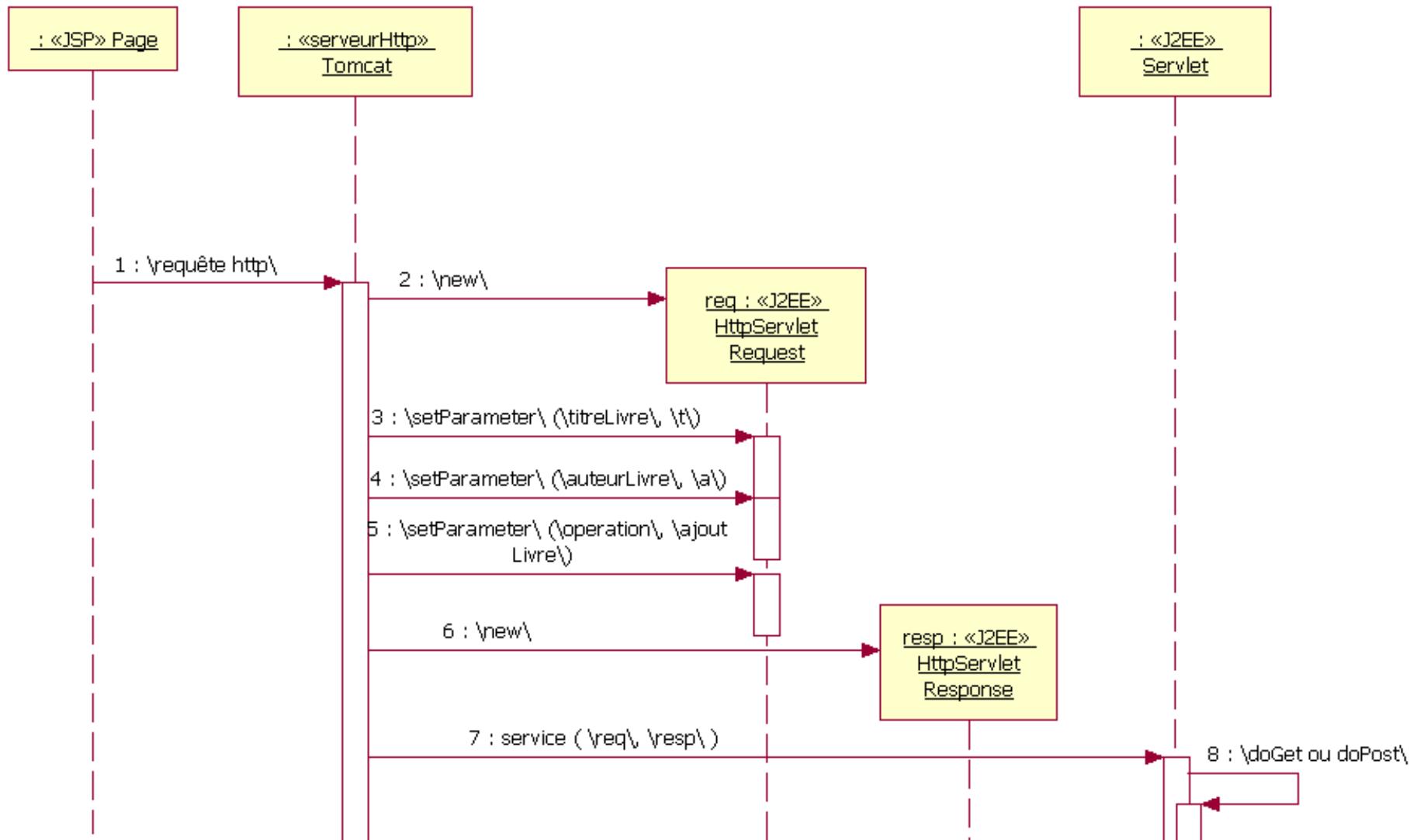
☞ A chaque réception d' une requête :

1. Appel de la méthode service (req, resp) en lui transmettant les paramètres de la requête http dans le paramètre *HttpServletRequest*
2. La méthode service transfert la requête (*HttpServletRequest*) à doPost ou à goGet.

J2EE avec UML

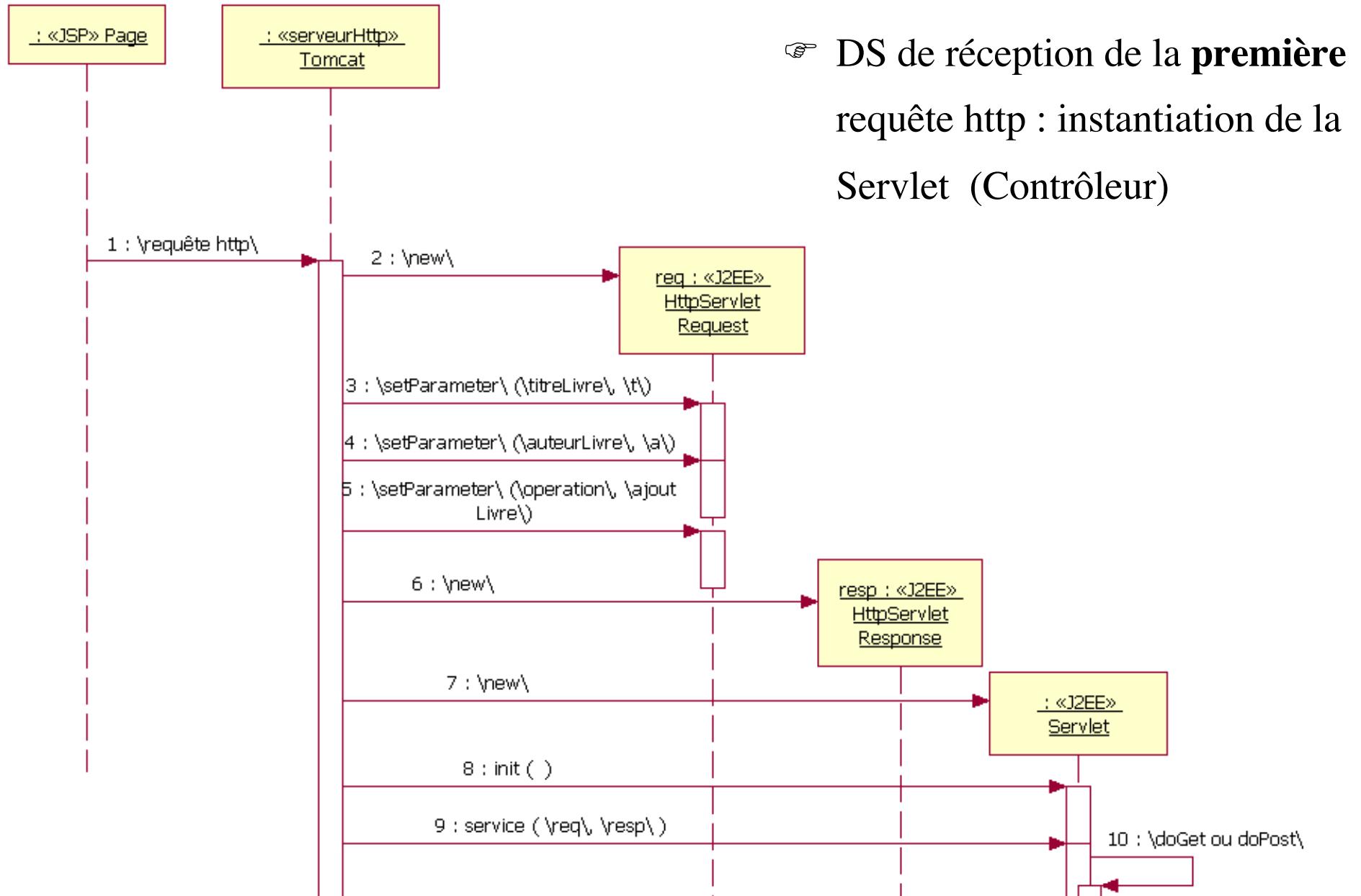
Réception des requêtes par la Servlet (Contrôleur)

- DS de réception d' une requête http par la Servlet (contrôleur)



J2EE avec UML

Réception des requêtes par la Servlet (Contrôleur)



Contrôleur : HttpServlet

Traitement des requêtes

👉 Contrôleur : classe HttpServlet

```
package controleur;  
import modele.*;  
public class ControleurBiblio extends HttpServlet {  
    private Bibliotheque modeleBiblio;  
    public void init (ServletConfig cf) throws ServletException {  
        super.init (cf);  
        modeleBiblio = new Bibliotheque();  
    }  
    public void service (HttpServletRequest req, HttpServletResponse rep) throws ... {  
        1. Identifier le service demandé et récupérer ses paramètres  
        2. Invoquer le service demandé du modèle  
        3. Transmettre les résultats à la vue  
    }  
    public void destroy () {modeleBiblio = null; }  
}
```

Contrôleur : HttpServlet

Traitement des requêtes

- ☞ Contrôleur (**HttpServlet**) : identifier la demande

```
public class ControleurBiblio extends HttpServlet {  
    public void service (HttpServletRequest req, HttpServletResponse resp){  
        // Identifier la demande  
        String demande = req.getParameter("operation");  
        if (demande.equals("ajoutLivre")) {  
            1. récupérer les paramètres http de la demande d'ajout d'un livre  
            2. Transmettre la demande au le modèle (Java pure)  
            3. Transmettre le résultat à la page JSP se chargeant de son affichage  
        }  
        if (demande.equals( "rechercherLivre")) {  
            1. récupérer les paramètres http de la demande de recherche d'un livre  
            2. Faire exécuter la demande par le modèle (Java pure)  
            3. Transmettre le résultat à la page JSP se chargeant de son affichage  
        }  
    }  
}
```

Contrôleur : HttpServlet

Traitement des requêtes

- ☞ Contrôleur : traitement de la demande d' ajout d' un livre

```
public class ControleurBiblio extends HttpServlet {  
    public void service (HttpServletRequest req, HttpServletResponse response) {  
        // identifier la demande  
        String demande = req.getParameter("operation");  
        if (demande.equals( "ajoutLivre")) {  
            // récupérer les paramètres http de la demande  
            String titre = req.getParameter ("titreLivre");  
            String auteur = req.getParameter ("auteurLivre");  
            // Transmettre la demande au modèle  
            boolean res = modeleBiblio.ajouterLivre(titre, auteur);  
            // Transmettre le résultat à la page JSP confirmationAjout  
            req.setAttribute ("confAjout", res);  
            RequestDispatcher d = req.getRequestDispatcher("/vue/confirmationAjout.jsp")  
            d.forward(req, response);  
            if (demande("rechercherLivre")) { // traitement de recherche }  
        }  
    }  
}
```

Contrôleur : HttpServlet

Traitement des requêtes

☞ Servlet (Controluer) : Transmettre une donnée à la vue (Page JSP)

1. Type simple (int, String, etc) : transmise sous forme de paramètre et récupérable avec la méthode : *setParameter (nomduparamètre, donnée)*
2. Types complexe (objets) : Transmis sous forme d' attribut et récupérable avec la méthode : *setAttribute (nomdelattribut, donnée)*

☞ Où ? Dans un des trois objets

- Dans la requête http : *HttpServletRequest*
- Dans la session de l' utilisateur
- Dans un objet commun à tous les utilisateurs

Vue JSP

Vues réponse de l' application

- ☞ Vue réponse : présentation des données (affichage des résultats)
- ☞ Où récupérer les données ? Dans un des trois objets JSP implicites
 - Dans la requête http : objet *request*
 - Dans la session de l' utilisateur : objet *session*
 - Dans un objet commun à tous les utilisateurs : objet *application*
- ☞ Deux types de données :
 1. **Paramètres** http: récupérable avec *getParameter (nomduparamètre)*
 2. Données de l'application : déposés dans la requête http sous forme **d'attribut** et récupérable avec la méthode : *getAttribute (nomdelattribut)*

Vue JSP

Vues réponse de l' application

☞ Exemple : la vue (Page JSP) de confirmation d' ajout d' un livre

- Récupérer le résultat de l' ajout et le titre du livre (types simples)
- Afficher le message

```
<%                                         confirmationAjout.jsp
    Boolean confirmation = (boolean) request.getAttribute("confAjout");
    String titre = request.getParameter("titreLivre");
%>
<html>  <body>
    <h1> Confirmation d'ajout du livre </h1>
    Le livre <%=titre%>
    <% if (confirmation) { %>
        a bien été ajouté !
    <% } else { %>
        n'a pu être ajouté !
    <% } %>
</body> </html>
```

Contrôleur Servlet

Envoi des résultats aux Vues réponse

- ☞ Contrôleur : traitement de la demande de recherche d' un livre

```
public class ControleurBiblio extends HttpServlet {  
    public void service (HttpServletRequest req, HttpServletResponse resp) {  
        // identifier la demande  
        String op = req.getParameter("operation");  
        if (op.equals( "rechercherLivre")) {  
            // récupérer les paramètres de la demande  
            String titre = req.getParameter ("titreLivreRecherche");  
            // Transmettre la demande au modèle  
            Livre liv = modeleBiblio.rechercherLivre(titre);  
            // Transmettre le résultat à la page JSP confirmationAjout  
            req.setAttribute ("livreTrouve", liv);  
            RequestDispatcher d = req.getRequestDispatcher("/vue/AfficherLivre.jsp");  
            d.forward(req, res);  
    }  
}
```

Vue JSP

Vues réponse de l' application

☞ La vue AfficherLivre.JSP

- Récupérer du livre trouvé (type complexe)
- Afficher le détail du livre

```
<%@ page import="modele.Livre" %>                               AfficherLivre.jsp
<%
    Livre unLivre = (Livre) request.getAttribute("livreTrouve");
    String titreRecherche = request.getParameter("titreLivreRecherche");
%>
<html>  <body>
    <h1> Résultat de la recherche </h1>  <br>
    <% if (unLivre == null) { %>
        Aucun livre ne correspond au titre <%= titreRecherche %> !
    <% } else { %>
        Titre du livre : <%=unLivre.getTitre()%>  <br>
        Auteur du livre : <%=unLivre.getAuteur()%>
    <% }%>
</body> </html>
```

Vue JSP

Vues réponse de l' application

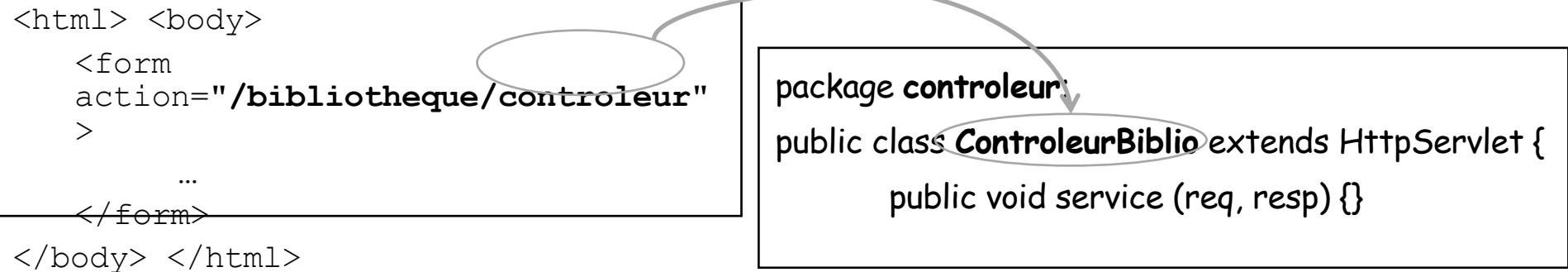
☞ Vue – Structure d' une page JSP

```
<%@ page import="monPackage.monBean" ... %>          // importations de classes  
<%  
    // récupération des données envoyées par le contrôleur.  
    // Utiliser les variables implicites : request, session et application  
%>  
<html>  
    ...  
    // on cherchera ici à minimiser le code java  
</html>
```

Contrôleur Servlet

Configuration

☞ Association URL / Servlet (Contrôleur)

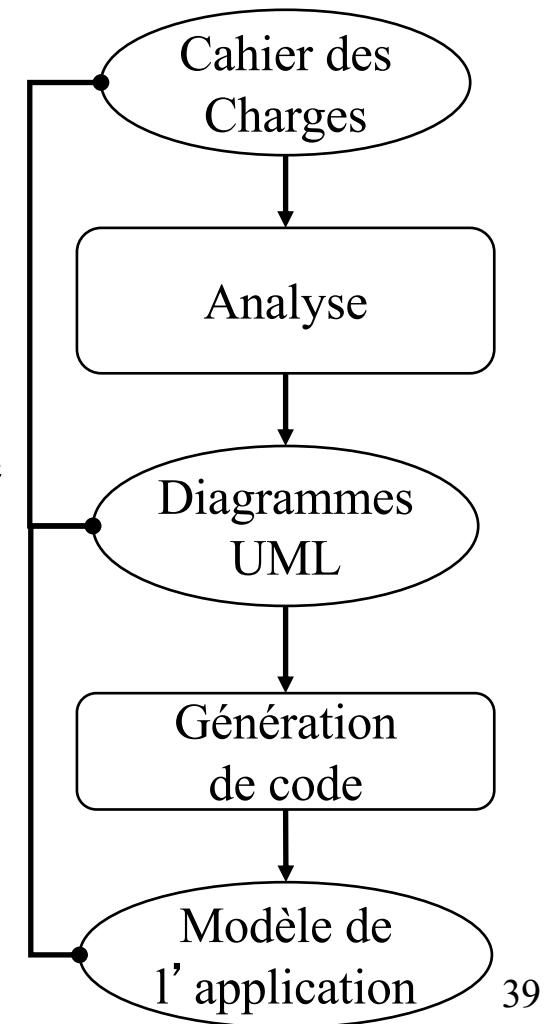


☞ Fichier de déploiement de l'application Web : *web.xml*

```
<?xml version="1.0" ?>
<web-app>
  <servlet>
    <servlet-name> ControleurServlet </servlet-name>
    <servlet-class> contrôleur.ControleurBiblio </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name> ControleurServlet </servlet-name>
    <url-pattern> /controleur </url-pattern>
  </servlet-mapping>
</web-app>
```

Modèle : Java (POJO)

- ☞ Modèle de l' application
 - ⇒ Cœur de l' application
 - ⇒ Obtenu en suivant une démarche d' analyse
 - objet
 - ⇒ Codage des diagrammes d' analyse



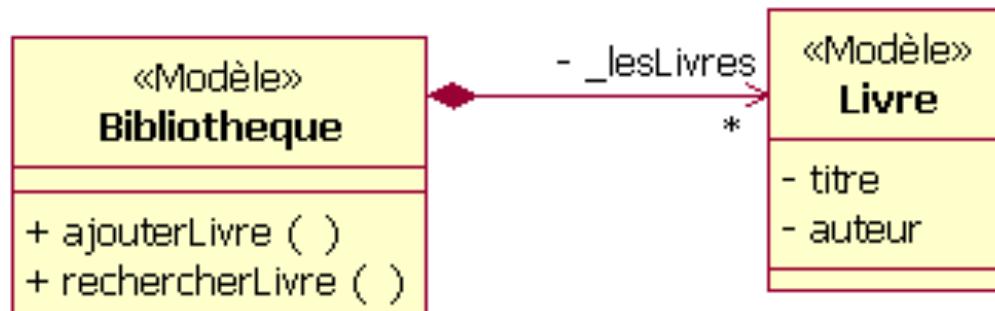
Modèle : Java (JaavaBean)

☞ Modèle

- Cœur de l' application
- Partie réutilisable, donc indépendante de son environnement (POJO)

☞ Conception du Modèle : en deux phases

- Sans persistance dans une base de données (POJO)
- Avec persistance dans une base de données via JDBC
- Avec persistance dans une base de données (EJB, plus tard!)



Modèle sans persistance : Java (POJO)

☞ Logique métier – Traduction des diagrammes UML (1)

```
package modele;  
public class Bibliotheque {  
    private ArrayList<Livre> lesLivres;  
    public Bibliotheque () {  
        lesLivres = new ArrayList<livre>();  
    }  
    public boolean ajouterLivre (String titre, String auteur) {  
        l = new Livre ();      l.setTitre (titre);    l.setAuteur (auteur)  
        ok = lesLivres.add (l);  
        return ok;  
    }  
    public Livre rechercherLivre (String titre) {  
        for (Livre l:lesLivres) {  
            if (l.getTitre == titre) return l;  
        }  
        return null;  
    }
```

Modèle sans persistance : Java (POJO)

☞ Logique métier – Traduction des diagrammes UML (2)

```
package modele;

public class Livre {
    private String titre;
    private String auteur;
    public Livre () {}
    public void setTitre (String t) {this.titre = t;}
    public void setAuteur (String a) {this.auteur = a;}
    public String getTitre (String t) {return this.titre;}
    public String getAuteur (String a) {return this.auteur;}
}
```

Application MVC

Structure sur le disque

- 👉 Organisation des fichiers sur disque

