Java EE

-

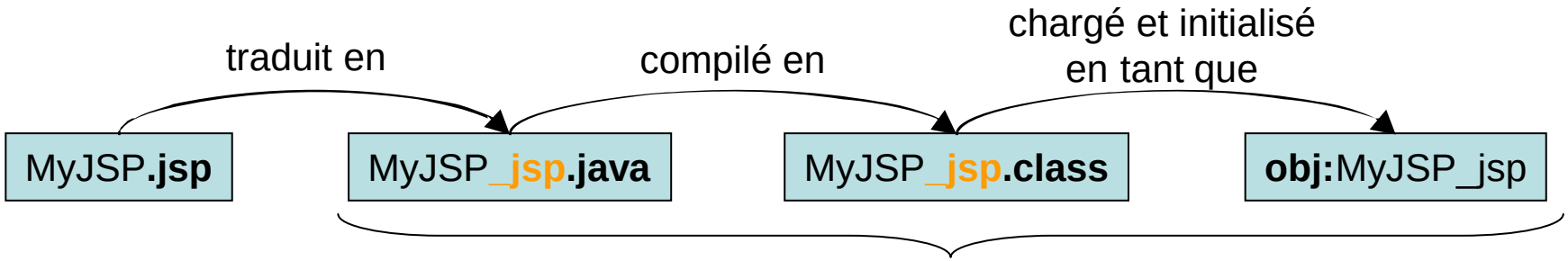
Cours 3

Cours de 2e année ingénieur
Spécialisation « Génie Informatique »

Rappels

JSP

- Les servlets facilitent le traitement avec java des requêtes et réponses HTTP, **mais** ils ne sont pas appropriés à l'écriture de code HTML
  - **out.println("<html><head><title>"+title+"</title>…");**
- Les JSP permettent d'intégrer du code java dans une page HTML
  - **<h1>Time on server</h1>**
    ** <%= new java.util.Date() %> **
- Mais au final une JSP n'est qu'un servlet!



Correspondance JSP/Servlet

- JSP d'origine

```
<h1>Time on server</h1>
<p><%= new  java.util.Date() %></p>
<% baz(); %>
<%! private int accessCount = 0; %>
```

- Servlet généré par Tomcat

```
public final class XXX_jsp
        extends org.apache.jasper.runtime.HttpJspBase
        implements org.apache.jasper.runtime.JspSourceDependent{
  public void _jspService(HttpServletRequest request,
                                        HttpServletResponse response)
              throws java.io.IOException, ServletException{
            response.setContentType("text/html");
            JspWriter out = response.getWriter();
            out.write("<h1>Time on server</h1>\r\n");
            out.write("<p>");
            out.print( new java.util.Date() );
            out.write("</p>\r\n");
            baz();
            }
     private int accessCount = 0;
}
```

Types des éléments de scripts JSP

- Expressions
  - Format : <%= expression %>
    Format XML:<jsp:expression> expression </jsp:expression>
  - Evaluée et insérée dans la sortie du servlet
    Se traduit par out.print(expression)
- Scriptlets
  - Format : <% code %>
    Format XML:**<jsp:scriptlet> code </jsp:scriptlet>**
  - Inséré tel quel dans la méthode _jspService du servlet (appelée par service)
- Déclarations
  - Format : <%! code %>
    Format XML: <jsp:declaration> code </jsp:declaration>
  - Insérée telle quelle dans le corps de la classe servlet, en dehors de toute méthode existante

Notions supplémentaires sur les JSP :

Les directives de page
<%@ directive …

Les directives de page

- Donnent des informations sur le servlet qui sera généré pour la page JSP
- Principalement utilisées pour:
  - L'importation de classes et paquetages
  - Le type MIME généré par la JSP

L’attribut «import»

- Format
  - <%@ page import="paquetage.classe" %>
  - <%@ page import="paquetage.classe1,...,paquetage.classeN" %>
- But
  - Générer les instructions d’importation
- Remarque
  - Bien que les pages JSP peuvent être n’importe où sur le serveur, les classes utilisées par les pages JSP doivent être dans le répertoire des classes de l’application Web (c’est-à-dire: *…/WEB-INF/classes*)

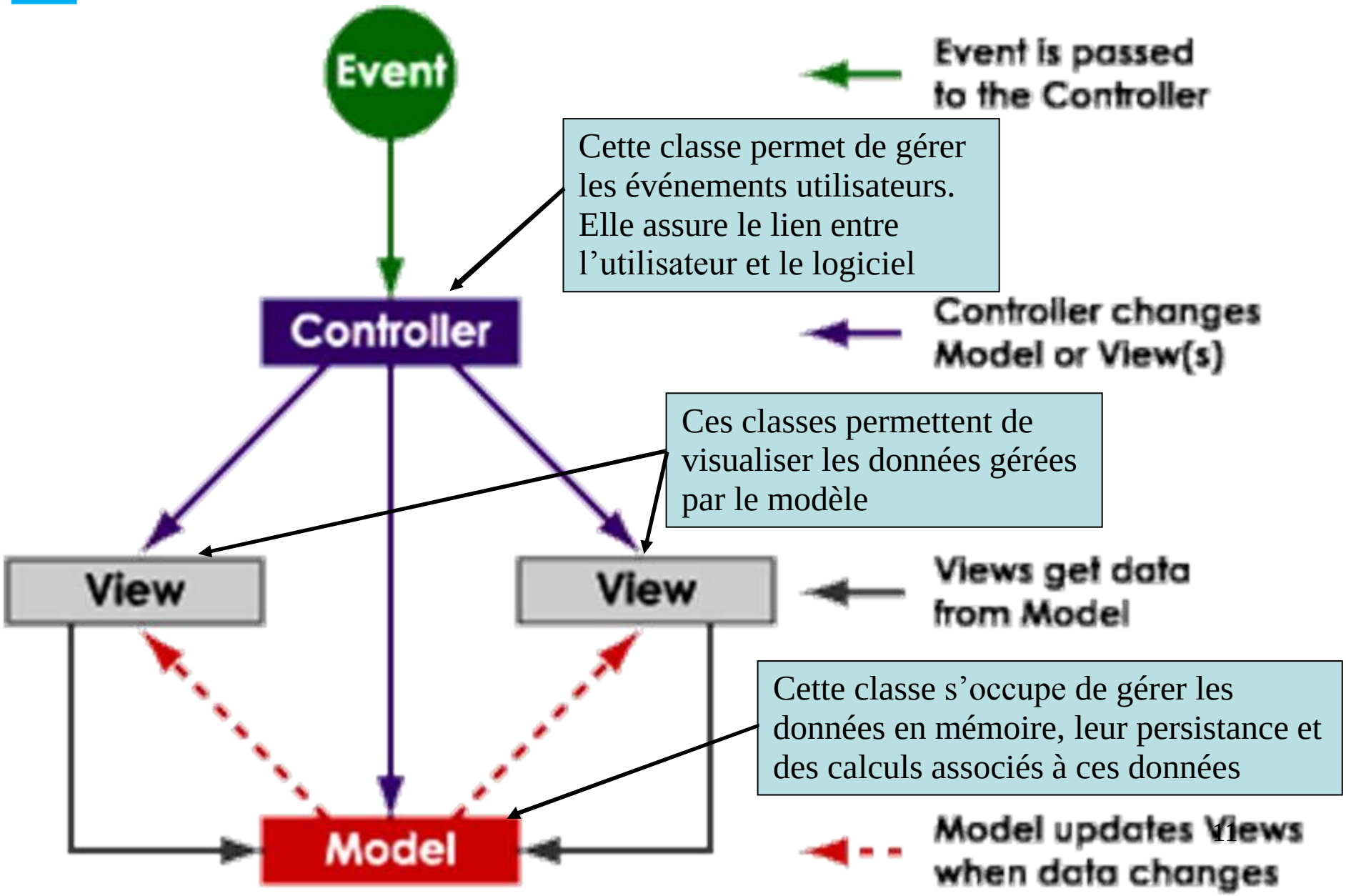
Intégration
des servlets et des JSP:

Application du design pattern Model-View-Controller (MVC)

Possibilités pour traiter une seule requête

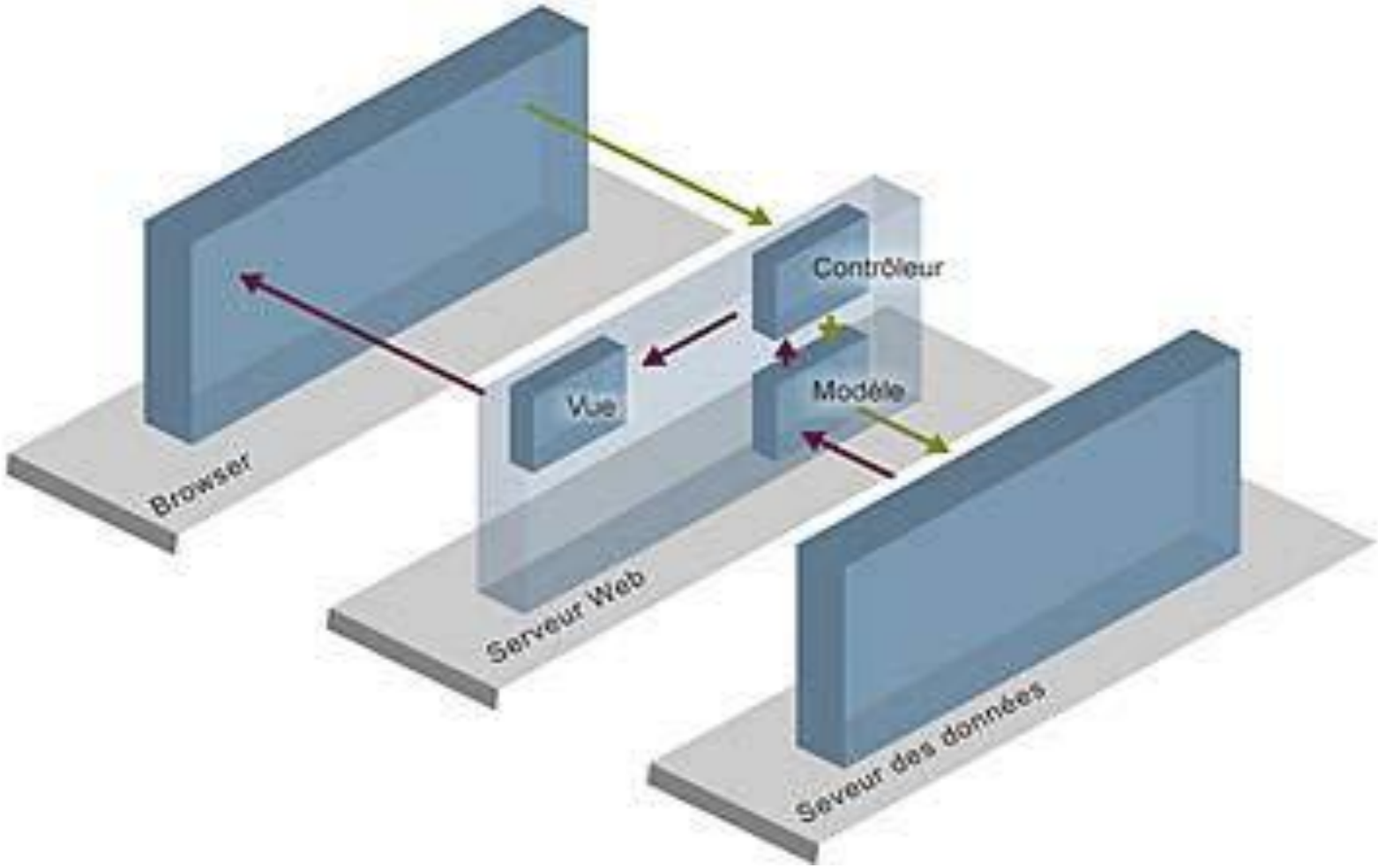
- Servlet seuls. Marche bien quand:
  - L'*output* est de type binaire. Ex : une image
  - Il n'y a pas d' *output*. Ex : redirections
  - La forme/présentation est variable. Ex : portail
- JSP seules. Marche bien quand:
  - L'*output* est de type caractère. Ex : HTML
  - La forme/présentation est stable.
- Architecture MVC. Nécessaire quand :
  - Une même requête peut donner des résultats visuels réellement différents
  - On dispose d'une équipe de développement conséquente avec une partie pour le dev. Web et une autre pour la logique métier
  - Ona un traitement complexe des données mais une présentation relativement fixe

Rappel : MVC


Event
Event is passed to the Controller
Cette classe permet de gérer les événements utilisateurs. Elle assure le lien entre l'utilisateur et le logiciel
Controller
Controller changes Model or View(s)
Ces classes permettent de visualiser les données gérées par le modèle
View
View
Views get data from Model
Cette classe s'occupe de gérer les données en mémoire, leur persistance et des calculs associés à ces données
Model
Model updates Views when data changes

Rappel : MVC

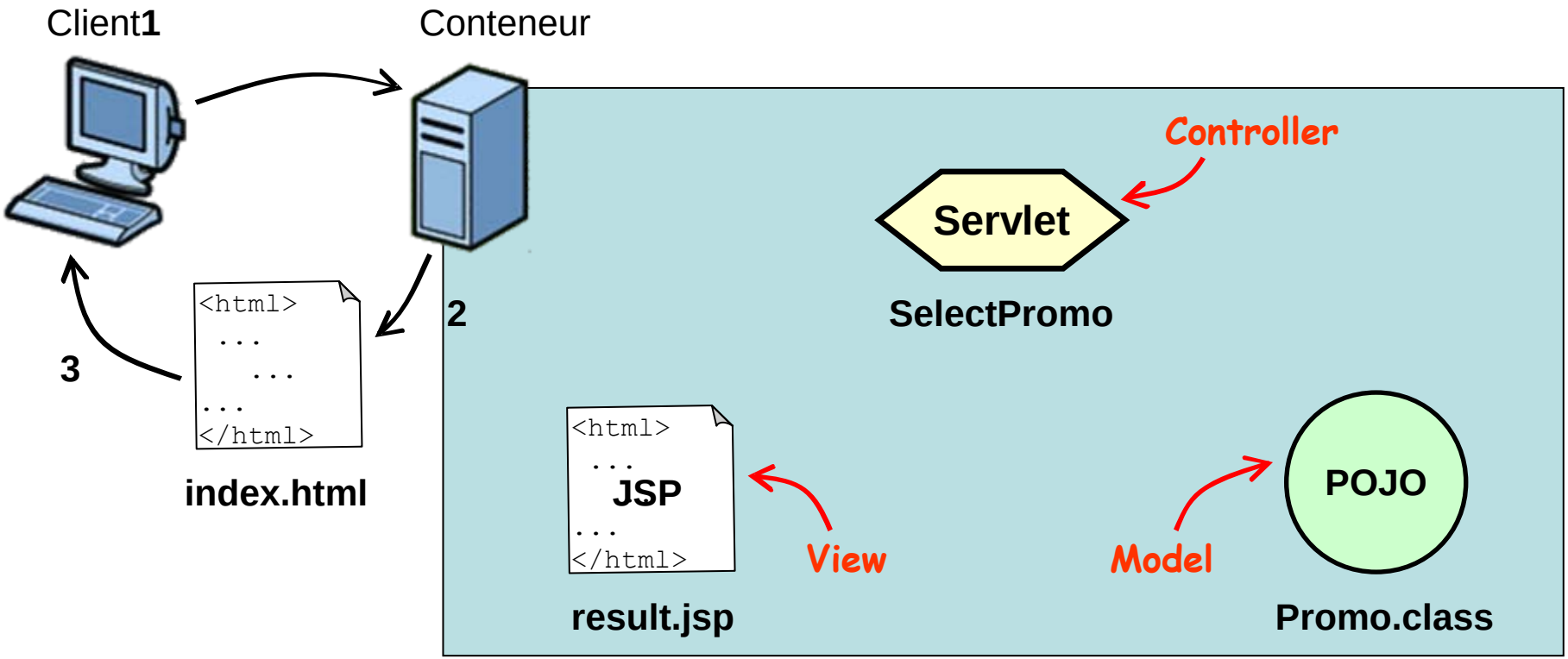
- Application dans le cadre d'Internet



Ex : AREL V6 - liste des promos

MVC : étape 1

Le client récupère un formulaire (index.html)
pour passer une requête avec paramètres (1, 2, puis 3)

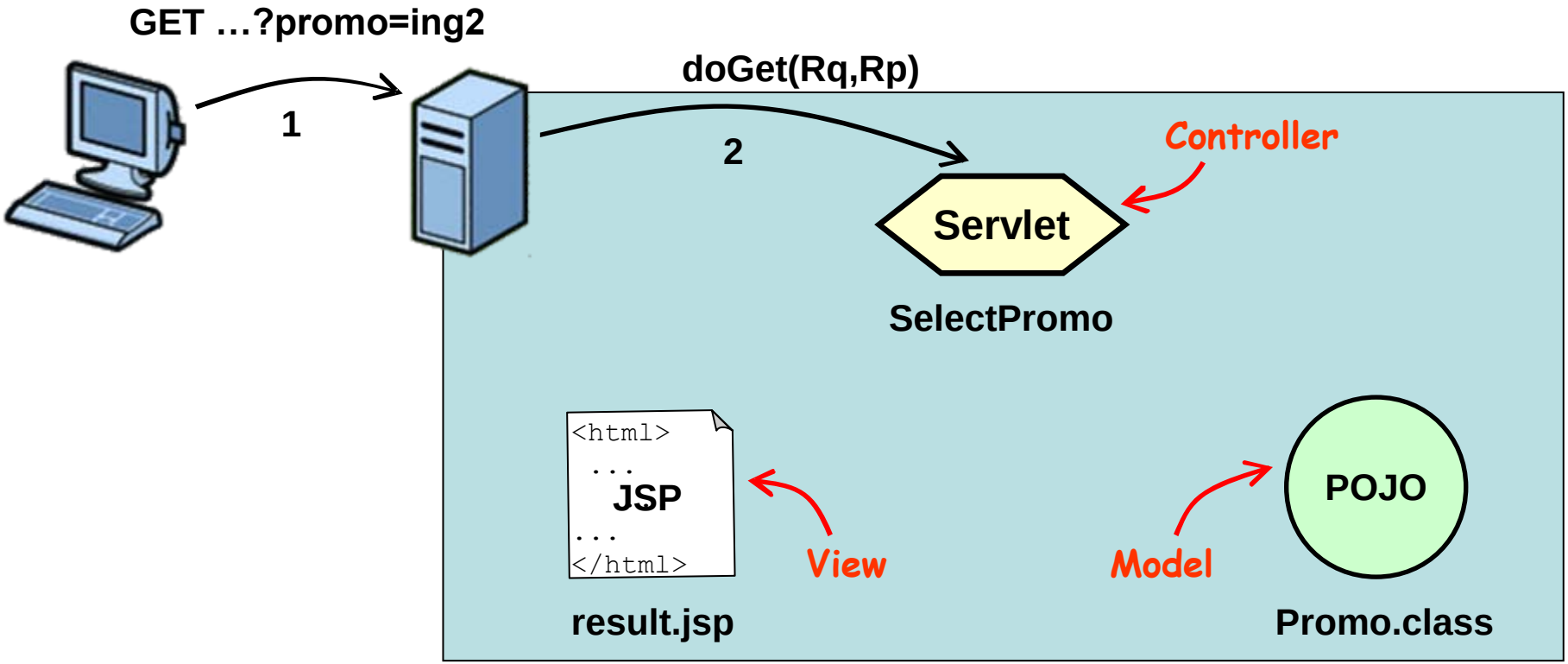


Formulaire : index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=ISO-8859-1">
<title> AREL V6.0 </title>
</head>
<body>
  <h1 align="center"> AREL: L'école virtuelle de l'EISTI </h1>

<form method="GET" action="http://localhost:8080/MVC/SelectPromo">
Sélectionner la promo à afficher:

 <select name="promo" size="1">
  <option>ing1</option>
  <option>ing2</option>
 </select><input type="SUBMIT" />

</form>
</body>
</html>
```

MVC : étape 2

1. Le client envoie son formulaire (GET/POST avec paramètres)
2. Le conteneur transmet au servlet correspondant (le *controller*)



Controller : SelectPromo.java

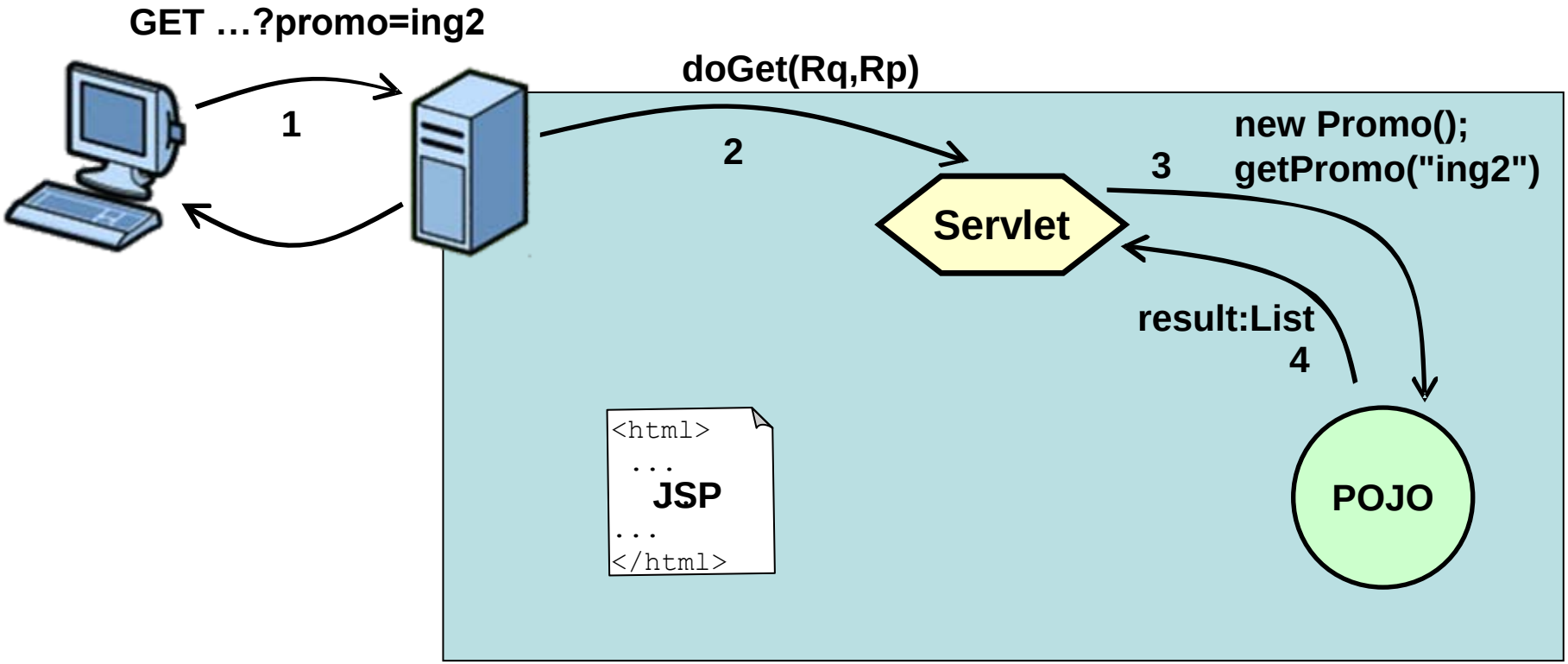
```
package arel;

import ...;

public class SelectPromo extends javax.servlet.http.HttpServlet
                         implements javax.servlet.Servlet{
   //...
   protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                 throws ServletException, IOException{

      String promoName = request.getParameter("promo");

      //...
   }
}
```

Configuration : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         id="WebApp_ID" version="2.5">
  <display-name>MVC</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <description></description>
    <display-name>SelectPromo</display-name>
    <servlet-name>SelectPromo</servlet-name>
    <servlet-class>arel.SelectPromo</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SelectPromo</servlet-name>
    <url-pattern>/SelectPromo</url-pattern>
  </servlet-mapping>
</web-app>
```

MVC : étape 3

3. Le servlet *controller* interroge le *model* sur «ing2»
4. Le *model* retourne au *controller* le résultat correspondant

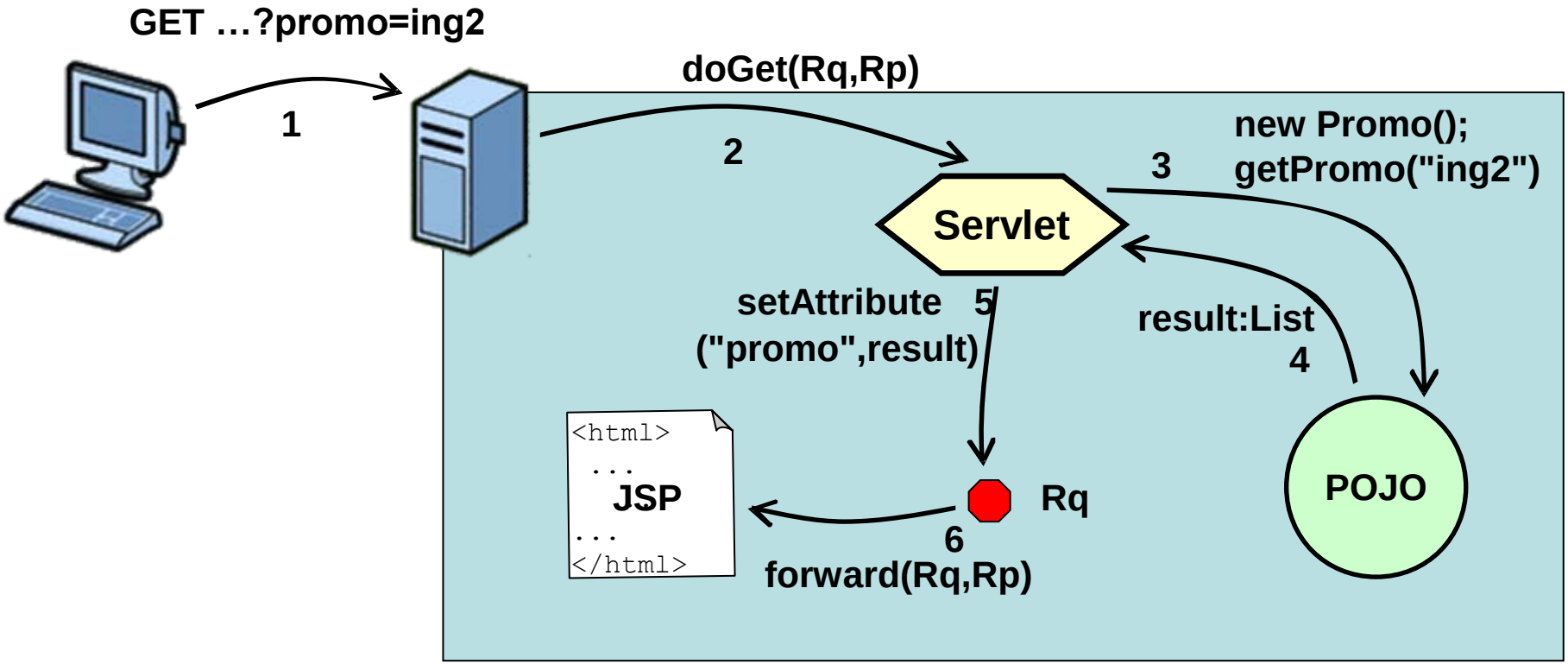


Model : Promo.java

```
package arel;
import ...;

public class Promo{

  public List<String> getPromo(String promo){
    List<String> promoList= new ArrayList<String>();
    if(promo.equals("ing1")){
      promoList.add("Donald Duck");
      promoList.add("Minnie Mouse");
      promoList.add("Pluto");  //...
    } else if (promo.equals("ing2")){
      promoList.add("Mickey Mouse");
      promoList.add("Daisy Duck");
      promoList.add("Goofy");//...
    } else{ return null;}

    return promoList;
  }
}
```

MVC : étape 4

5. Le *controller* utilise les données du *model* pour sa réponse
6. Le *controller* transmet sa réponse à la *view* (JSP)

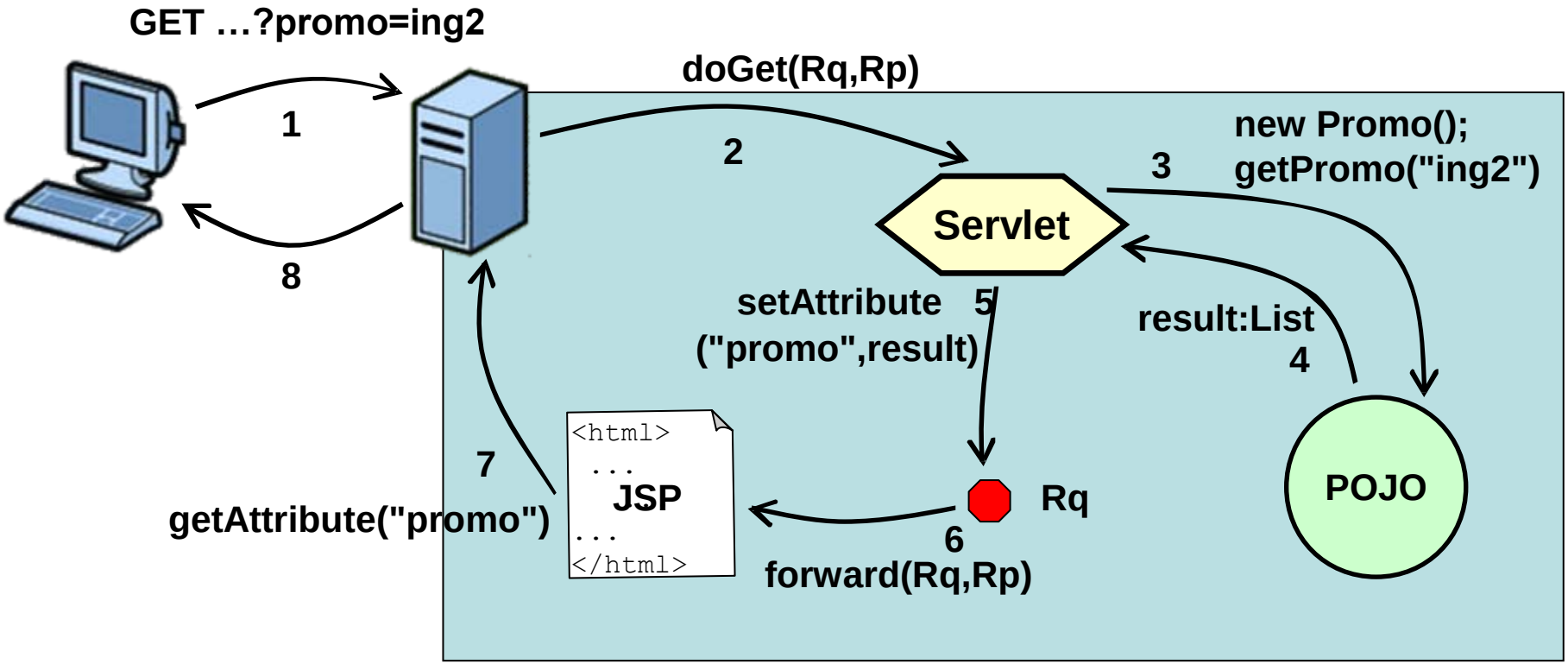


Controller : SelectPromo.java

```
package arel;

import ...;

public class SelectPromo extends javax.servlet.http.HttpServlet
                         implements javax.servlet.Servlet{
  //...
  protected void doGet(HttpServletRequest request,
                       HttpServletResponse response)
                throws ServletException, IOException{
    String promoName = request.getParameter("promo");
    Promo promo = new Promo();
    List<String> result = promo.getPromo(promoName);
    request.setAttribute("promo", result); // On ajoute l'attribut
    RequestDispatcher view =                // promo à la requête
      request.getRequestDispatcher("result.jsp");
    view.forward(request, response); // On forward la requête
  }                                  // à la JSP
}
```

MVC : étape 5

7. La JSP (view) traite la réponse transmise par le *controller*
8. La page HTML résultante est reçue par le client

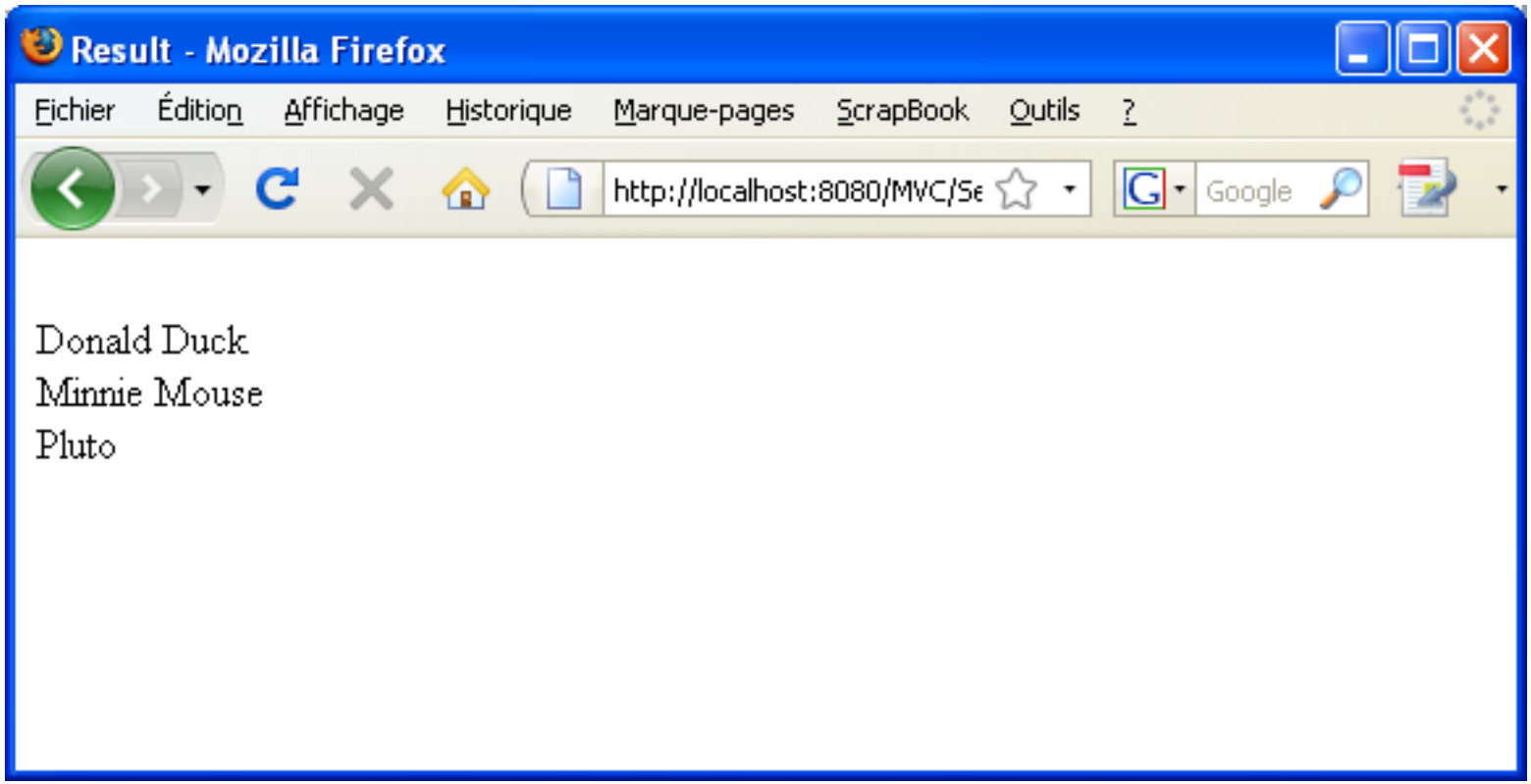


View : result.jsp

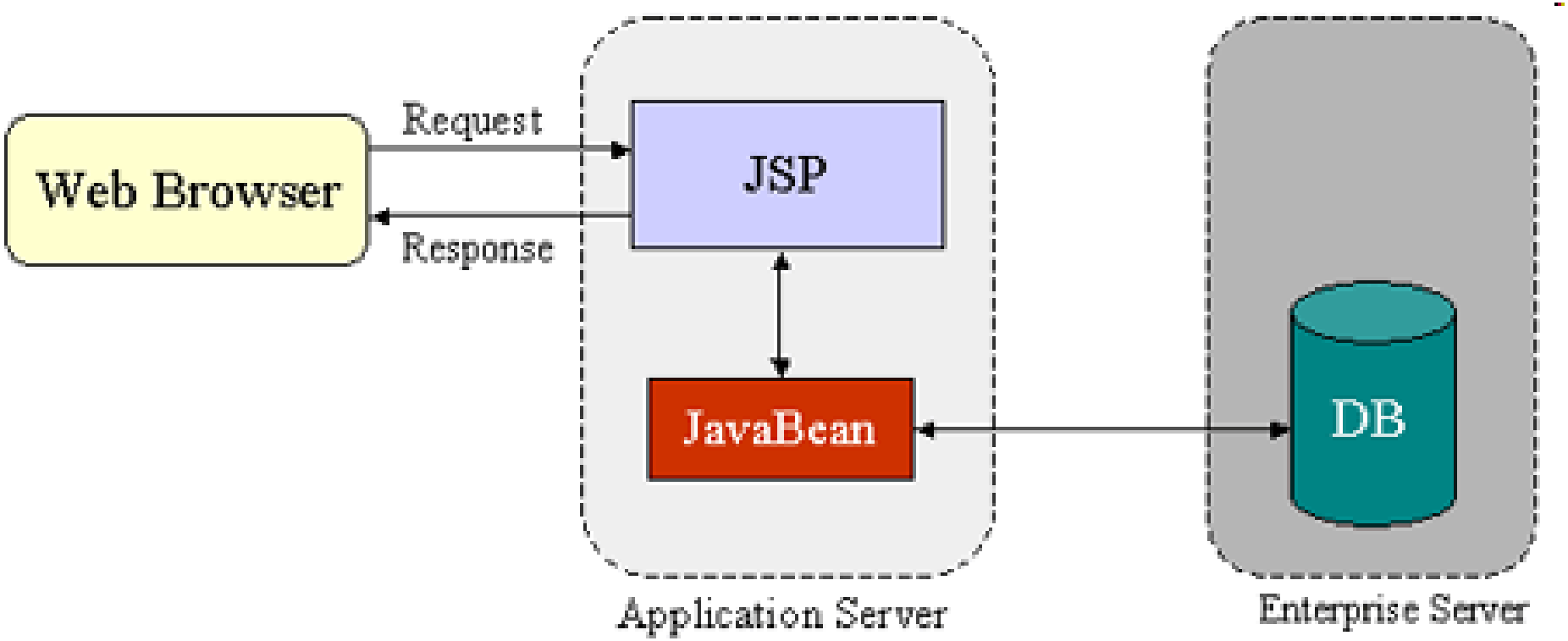
```
<%@ page import="java.util.*" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=ISO-8859-1">
<title>Result</title>
</head>
<body>
<%
  List<String> promoList=(List<String>)request.getAttribute("promo");
  Iterator it=promoList.iterator();
  while(it.hasNext()){
    out.print("<br />"+it.next());
  }
%>
</body>
</html>
```

On récupère l'attribut *promo* ajouté à la requête lors de l'étape 4

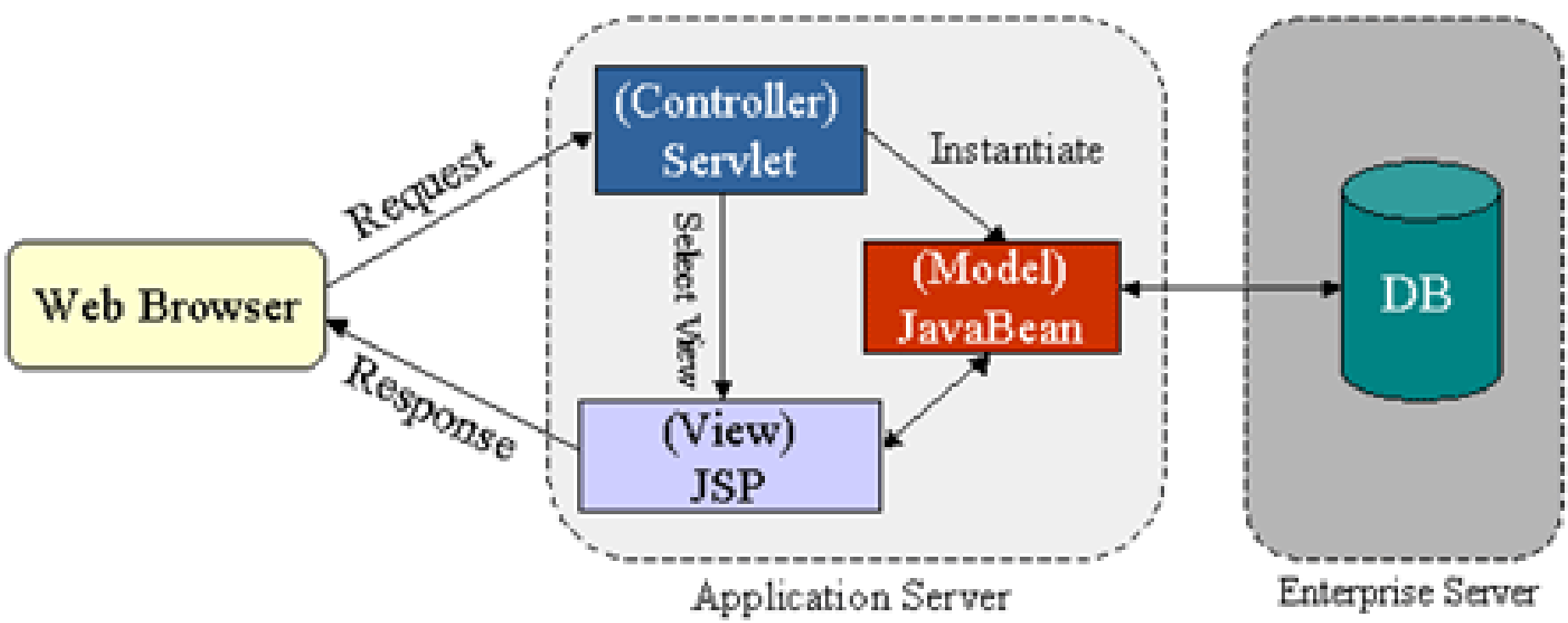
Ex : AREL V6 - liste des promos



Architecture Web JEE : Model 1 (pas MVC)



Architecture Web JEE : Model 2 (MVC)



JavaBeans

JavaBeans?

Les **JavaBeans** sont des classes Java(POJO) qui suivent certaines conventions:

– Doivent avoir un **constructeur vide** (zéro paramètre)

=> On peut satisfaire cette contrainte soit en définissant explicitement un tel constructeur, soit en ne spécifiant aucun constructeur

– Ne doivent **pas avoir d'attributs publics**

=> Une bonne pratique réutilisable par ailleurs…

– La valeur des attributs doit être manipulée à travers **des accesseurs** (getters et setters)

=> Si une classe possède une méthode getTitle qui retourne un String, on dit que **le bean possède une propriété** String nommée title

=> Les propriétés Boolean utilisent isXXX à la place de getXXX

Réf. sur les beans : `http://docs.oracle.com/javase/tutorial/javabeans/`

Pourquoi faut-il utiliser des accesseurs

- Dans un bean, on ne peut pas avoir de champs publics
- Donc, il faut remplacer

```
public double speed;
```

Par

```
private double speed;

public double getSpeed() {
      return(speed);
}

public void setSpeed(double newSpeed) {
      speed=newSpeed;
}
```

Pourquoi faut-il faire cela pour **tout** code Java?

Pourquoi faut-il utiliser des accesseurs?

1) On peut imposer des contraintes sur les données

```
public void setSpeed(double newSpeed){
      if(newSpeed < 0){
            SendErrorMessage(...);
            newSpeed = Math.abs(newSpeed);
      }
      speed = newSpeed;
}
```

Pourquoi faut-il utiliser des accesseurs

2) On peut changer de représentation interne sans changer d'interface

```
// newSpeed and internal representation are in km units
public void setSpeed(double newSpeed){
      speed = newSpeed;
}

// Now using miles units for internal representation (but
// keeping km units for newSpeed)
public void setSpeed(double newSpeed){
      speed = convertKMToMiles(newSpeed);
}
```

Pourquoi faut-il utiliser des accesseurs?

3) On peut rajouter du code annexe

```
public double setSpeed(double newSpeed){
      speed = newSpeed;
      updateSpeedometerDisplay();
}
```

Comment utiliser des beans ?

Une fois la classe créée, les beans sont des objets Java sur lesquels on peut faire ces actions :

- instanciation d'un nouveau bean
- récupération de la valeur d'une propriété du bean
- affectation/modification de la valeur d'une propriété du bean

Pour faire cela dans une Servlet, ça ne pose pas de problème particulier, le bean est traité comme un objet Java standard.

Cependant, dans une JSP, l'utilisation des balise de scriptlets n'est pas très « propre », on utilisera plutôt l'une des manière suivantes :

- **balises JSP (JSP 1.2)** : <jsp:useBean>, <jsp:getProperty>, <jsp:setProperty>
- **utilisation des EL (JSP 2.0)** (si le serveur le supporte) : ${...}

⇒Dans ce cours nous verrons la deuxième approche, la première n'étant aujourd'hui utilisée que pour des raisons de compatibilités avec la version du serveur installé

Exemple MVC : Modèle

```
package model;


public class PersonneBean {
        private String nom;
        private String prenom;

        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;
        }
}
```

Les Beans doivent être déployés dans le même répertoire que les autres classes Java

```
WEB-INF/classes/nomPackage
```

Les Beans doivent **toujours** être dans des packages

Exemple MVC : Contrôleur

-En MVC, **les beans sont crées/modifiés par le contrôleur**, en fonction de la requête du client
=> surtout pas par la vue !

```
import model.PersonneBean;


@WebServlet("/MyServlet")
public class MyServlet extends HttpServlet {
            protected void doGet(...)
            {
                        String urlVue = "vue.jsp";                          // URL de la vue à appeler
                        String nom = request.getParameter("...");           // Récupération du champs nom
                                                                            //(par exemple via un formulaire)
                        String prenom = request.getParameter("...");        // Récupération du champs prenom
                        if(nom == null || nom.trim().equals("")) {
                                    // Erreur : nom non saisi
                        }
                        else if(prenom == null || prenom.trim().equals("")) {
                                    // Erreur : prenom non saisi
                        }
                        else // Cas sans erreur : on traite la requête, et crée les beans nécessaires
                        {
                                    PersonneBean bean = new PersonneBean(); // Instanciation d'un bean
                                                                            // de type PersonneBean
                                    bean.setNom(nom);                       // Affectation de la propriété nom
                                    bean.setPrenom(prenom);                 // Affectation de la propriété prenom
                                    request.setAttribute("myBean", bean);   // Attacher ce bean au
                                                                            // scope de requête
                        }
                        // Forward à la vue:
                        request.getRequestDispatcher(urlVue).forward(request, response);
            }
}
```

Exemple MVC : Vue

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%><html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Ma Vue</title>
</head>
<body>
        Salut ${myBean.prenom} ${myBean.nom} !
</body>
</html>
```

- En MVC, les beans sont uniquement consultés par la vue, pour faire son affichage.
-L'instruction ${...} est une EL (Expression Language),
nous reviendrons dessus au prochain TP
=> L'EL permet de récupérer l'information sur un bean (s'il a été mis préalablement dans un scope par une servlet avec un *setAttribute*):
`${nomBean.property}`
- *nomBean* est celui défini lors du setAttribute fait par la servlet
- *property* est le nom de la propriété que l'on veut accéder (attention aux normes d'écriture Java => respectez la casse)