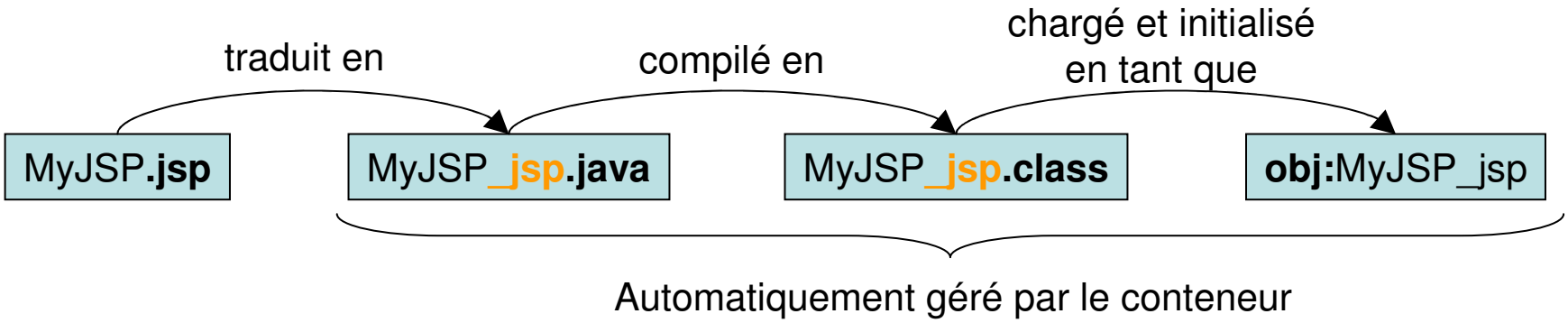
Java EE

Cours de 2e année ingénieur
Spécialisation « Génie Informatique »

fabien.romeo@fromeo.fr
http://www.fromeo.fr

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>
```

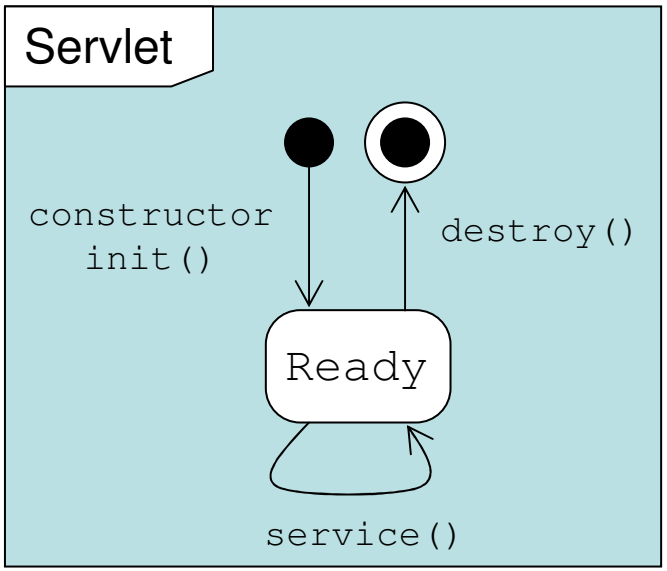
- Servlet généré par Tomcat

```
public final class Clock_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");
      // ...
}
```

Cycle de vie d'un servlet

1. Chargement de la classe
2. Instanciation du servlet
   - constructeur par défaut
3. Appel de init()
4. Appel(s) de service()
   - 1 thread par requête
5. Appel de destroy()



Types des éléments de scripts

- 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

Correspondance JSP/Servlet

- JSP d'origine

```
<h2>foo</h2>
<%= bar() %>
<% baz(); %>
<%! private int accessCount = 0; %>
```

- Code du servlet résultant de la traduction

```
public final class XXX_jsp ... {
 public void _jspService(HttpServletRequest request,
                         HttpServletResponse response)
     throws ServletException, IOException {
   response.setContentType("text/html");
   HttpSession session = request.getSession();
   JspWriter out = response.getWriter();
   out.println("<h2>foo</h2>");
   out.println(bar());
   baz();
   ...
 }
 private int accessCount = 0;
}
```

Types des éléments de scripts JSP : 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

Les attributs « contentType » et « pageEncoding »

- Format
  - <%@ page contentType="MIME-Type" %>
  - <%@ page contentType="MIME-Type; charset=Character-Set" %>
  - <%@ page pageEncoding="Character-Set" %>

- But
  - Spécifier le type MIME de la page générée par le servlet résultant de la page JSP

Intégration
des servlets et des JSP :

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

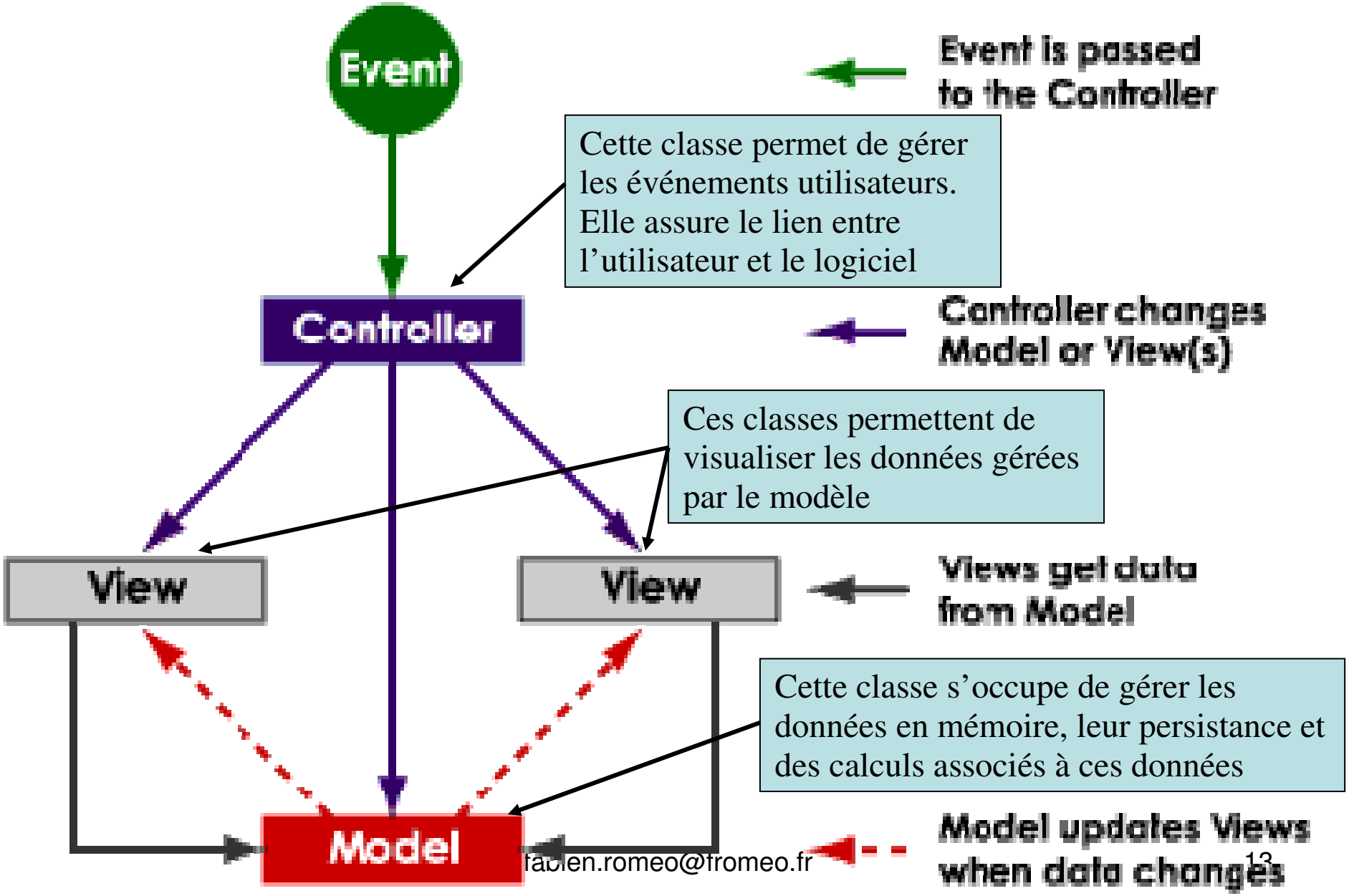
Pourquoi combiner Servlets & JSP?

- Classiquement : utilisation des JSP pour faciliter le développement et la maintenance du contenu HTML
  - Pour du code dynamique simple, appel du code d'un servlet à partir de scripts JSP
  - Pour des applications un peu plus complexes, utilisation de classes appelées à partir de scripts JSP
- Mais ça n'est pas suffisant
  - Pour des traitements complexes, démarrer avec des JSP n'est pas pratique
  - Mais surtout, l'idée derrière les JSP est qu'une seule page possède une forme, une présentation de base stable

Possibilités pour traiter une seule requête

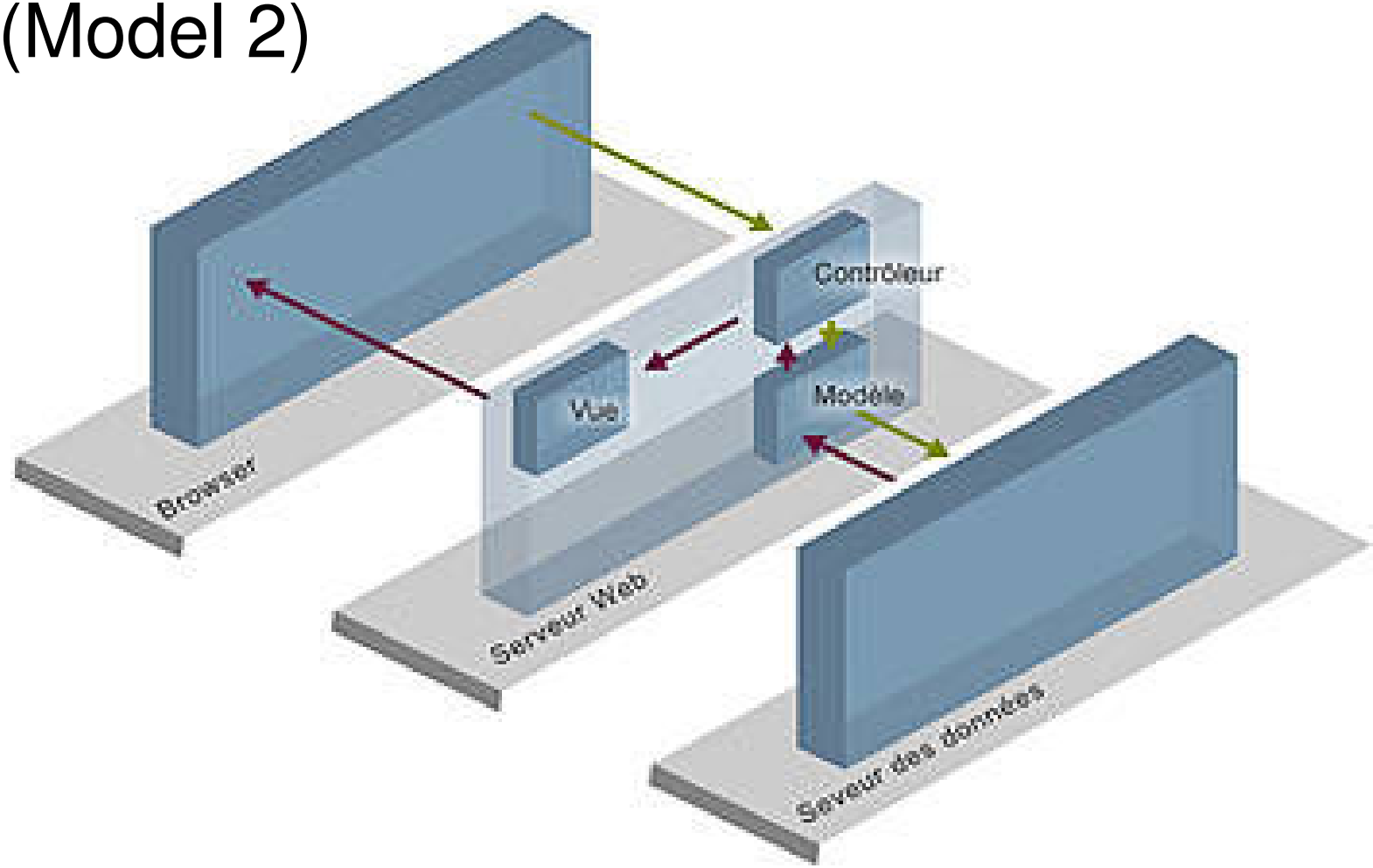
- Servlet seuls. Marche bien quand :
  - L'*output* est de type binaire. Ex : une image
  - Il n'y a pas d'*output*. Ex : redirections
  - La forme/présentation est variable. Ex : portail
- JSP seules. Marche bien quand :
  - L'*output* est de type caractère. Ex : HTML
  - La forme/présentation est stable.
- Architecture MVC. Nécessaire qd:
  - 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
  - On a un traitement complexe des données mais une présentation relativement fixe

Rappel : MVC

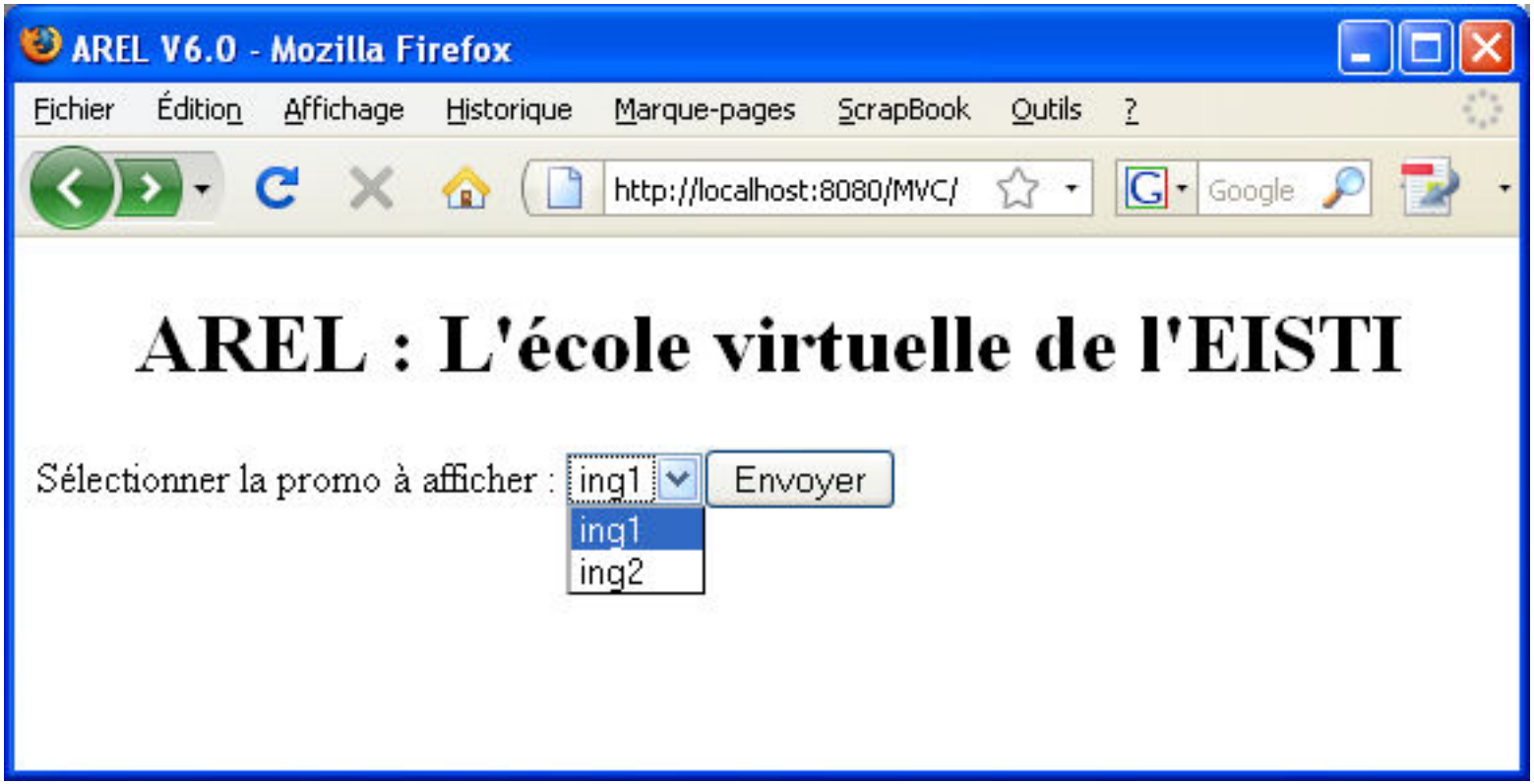

Event
Event is passed to the Controller
Cette classe permet de gérer les événements utilisateurs. Elle assure le lien entre l'utilisateur et le logiciel
Controller
Controller changes Model or View(s)
Ces classes permettent de visualiser les données gérées par le modèle
View
View
Views get data from Model
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 (Model 2)

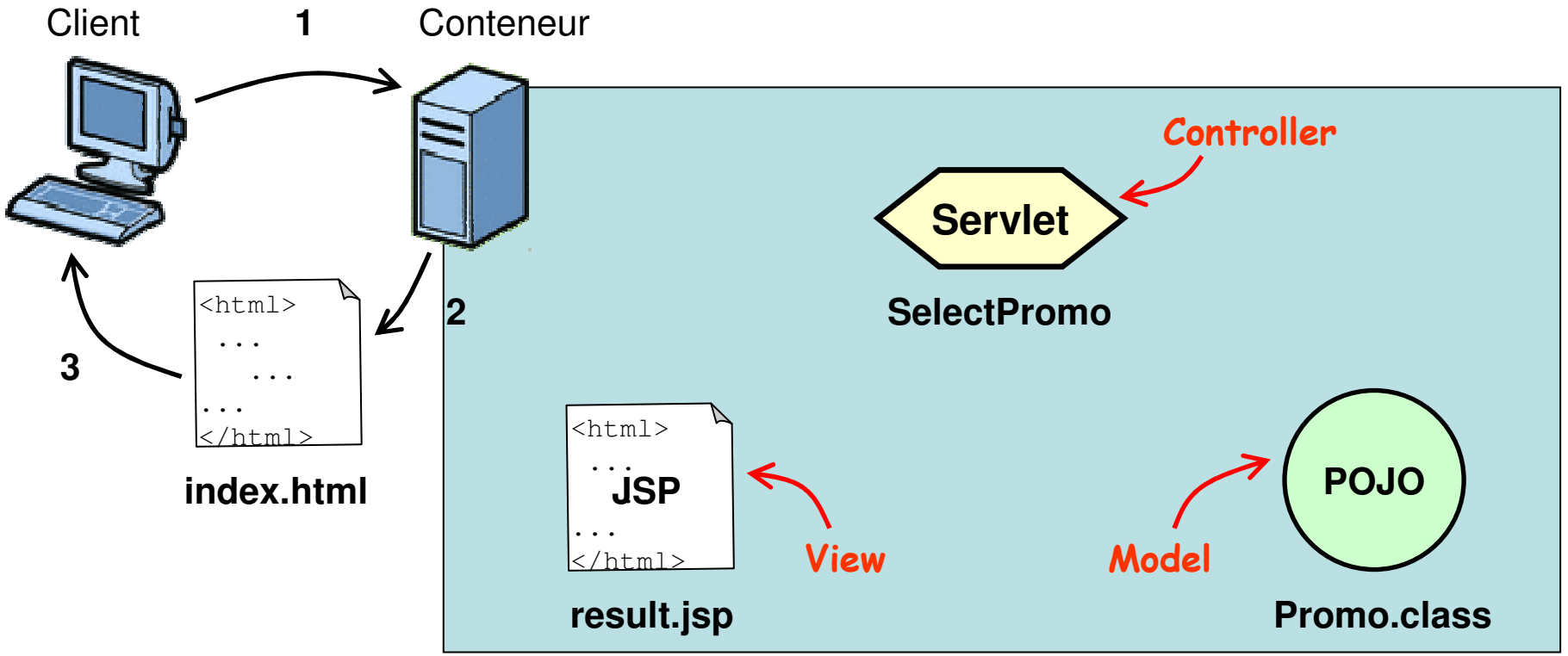


Ex : AREL V6 - liste des promos



MVC : étape 1

Le client récupère un formulaire (form.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 :

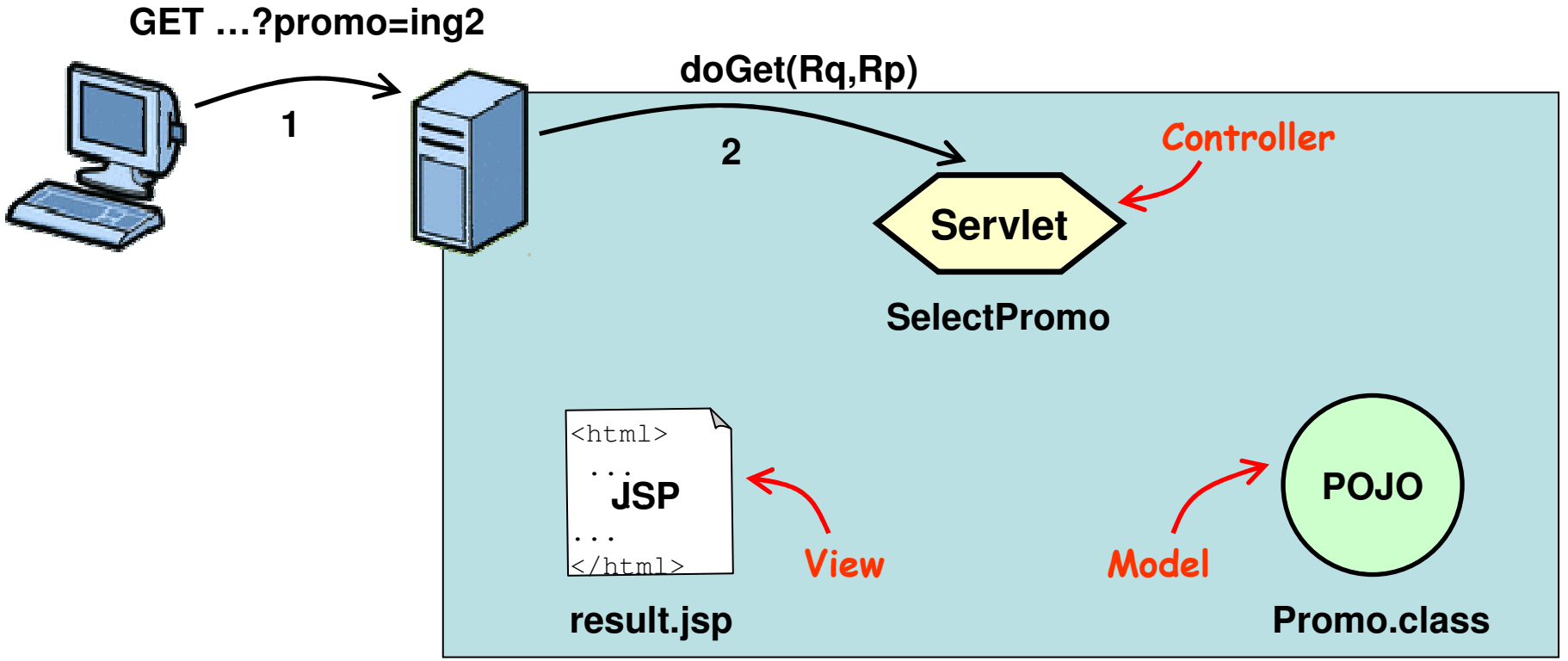
 <select name="promo" size="1">
  <option>ing1</option>
  <option>ing2</option>
 </select><input type="SUBMIT" />

</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");

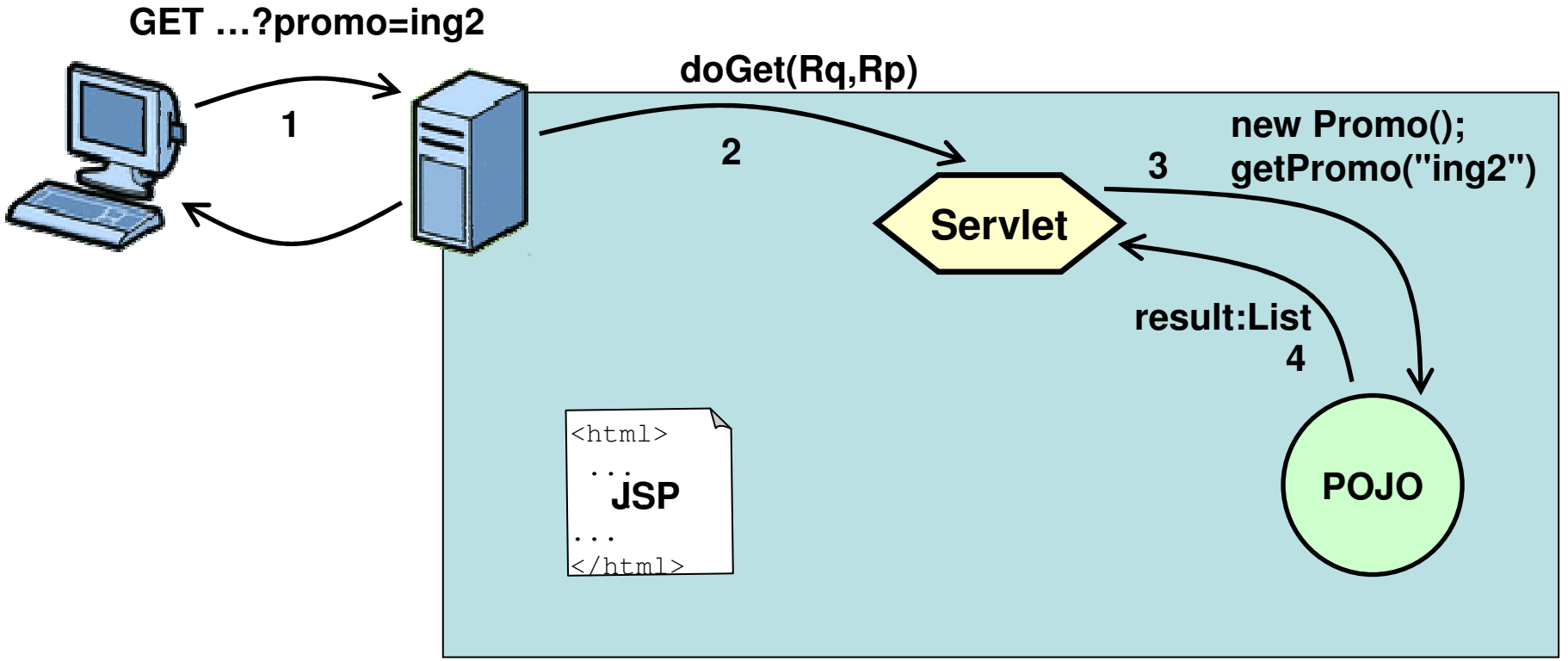
    //...
  }
}
```

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 ...;

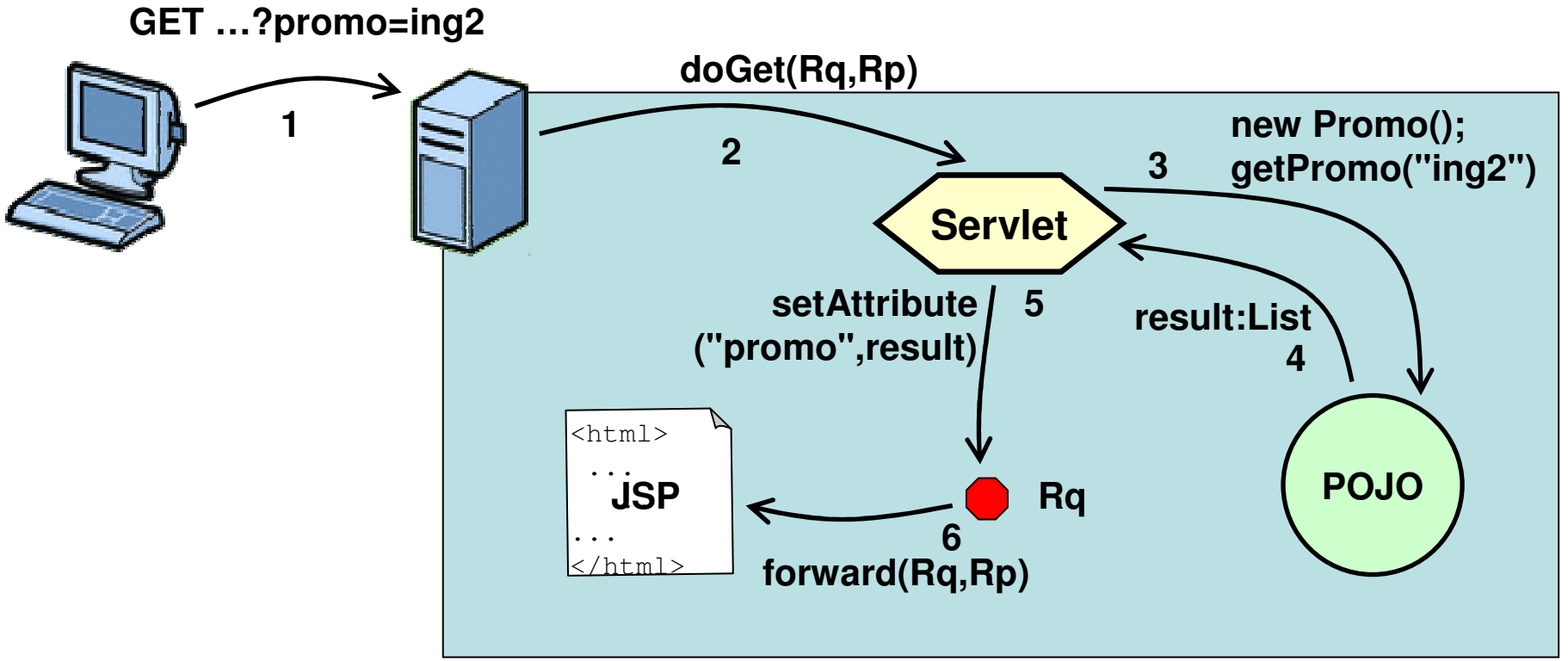
public class Promo {

  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

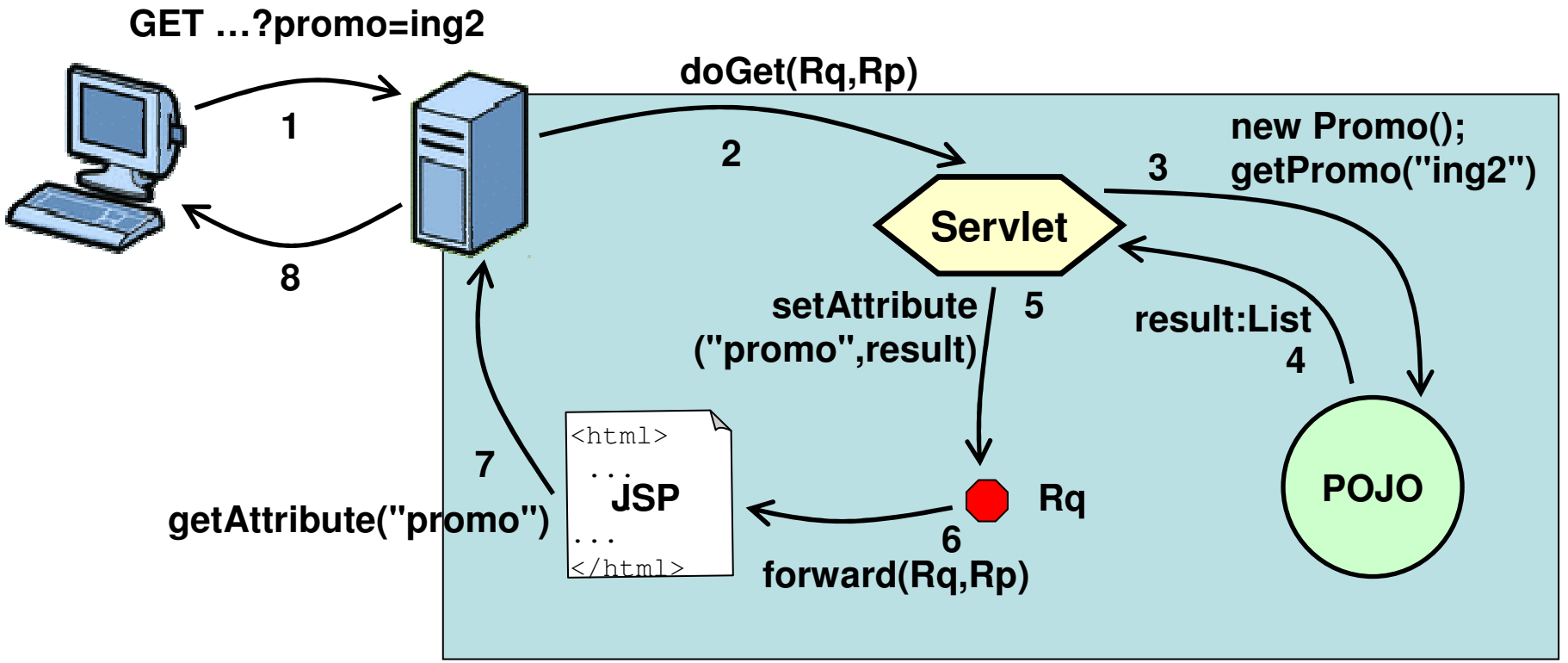
```
package arel;

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);
    RequestDispatcher view =
      request.getRequestDispatcher("result.jsp");
    view.forward(request, response);
  }
}
```

MVC : étape 5

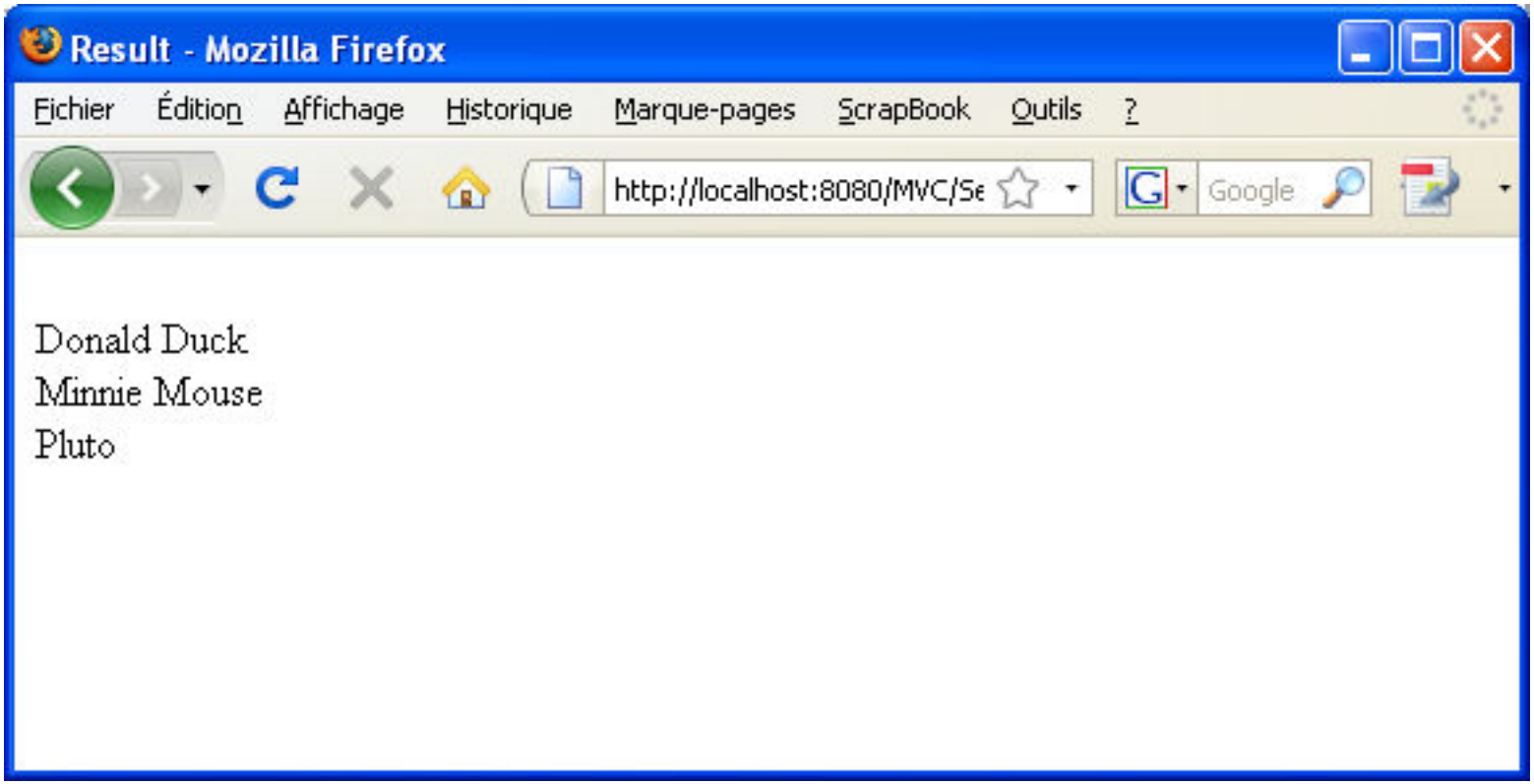
7. La JSP (view) traite la réponse transmise par le *controller*
8. La page HTML résultante est reçue par le client



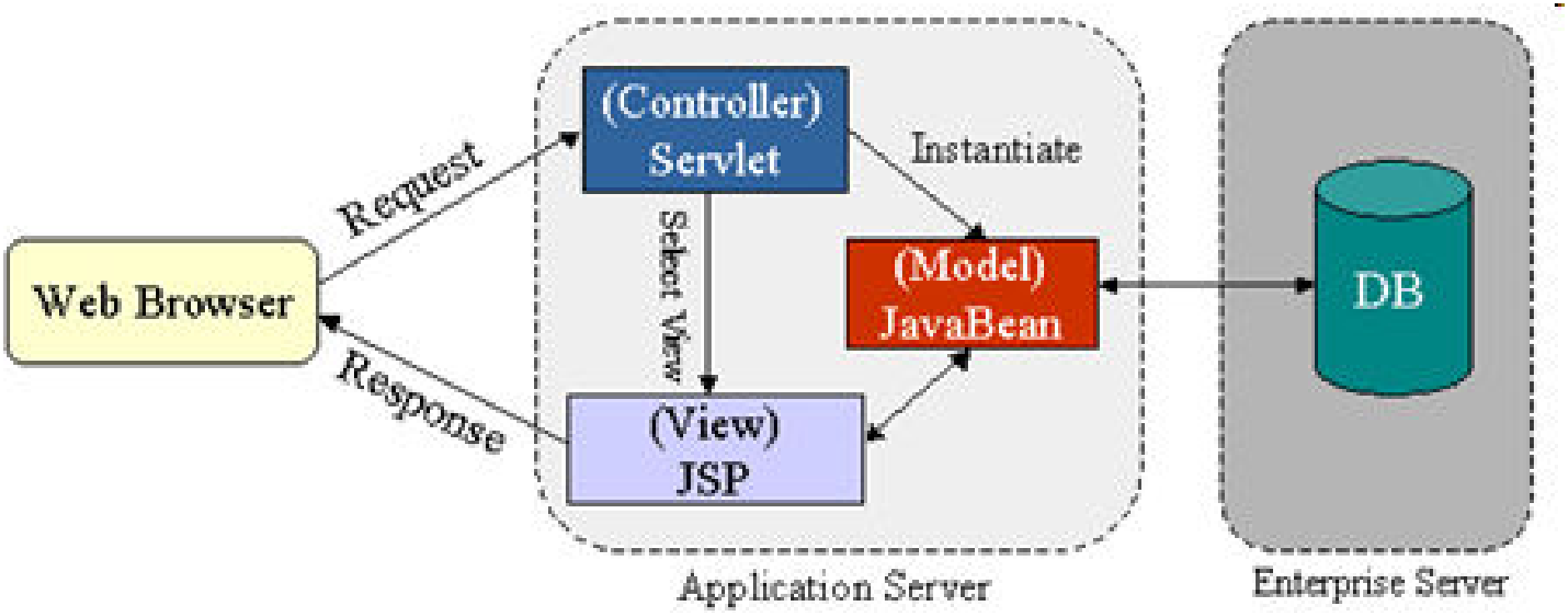
View : result.jsp

```
<%@ 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>
```

Ex : AREL V6 - liste des promos



Architecture Web JEE : Model 2 (MVC)



Architecture Web JEE : Model 1 (pas MVC)