MVC

INTÉGRATION SERVLETS & JSP : MVC

POURQUOI COMBINER SERVLETS & JSP?

- L' utilisation des JSP pour faciliter le développement et la maintenance du contenu HTML :

1. Pour du code dynamique simple, appel du code d' un servlet à partir de scripts JSP
2. Pour des applications un peu plus complexes, utilisation de classes appelées à partir de scripts JSP

- Mais ce 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

DES POSSIBILITÉS :

- Servlet seul, marche bien quand :

1. L'*output est de type binaire. Ex : une image*
2. Il n'y a pas d'*output. Ex : redirections*
3. La forme/présentation est variable.

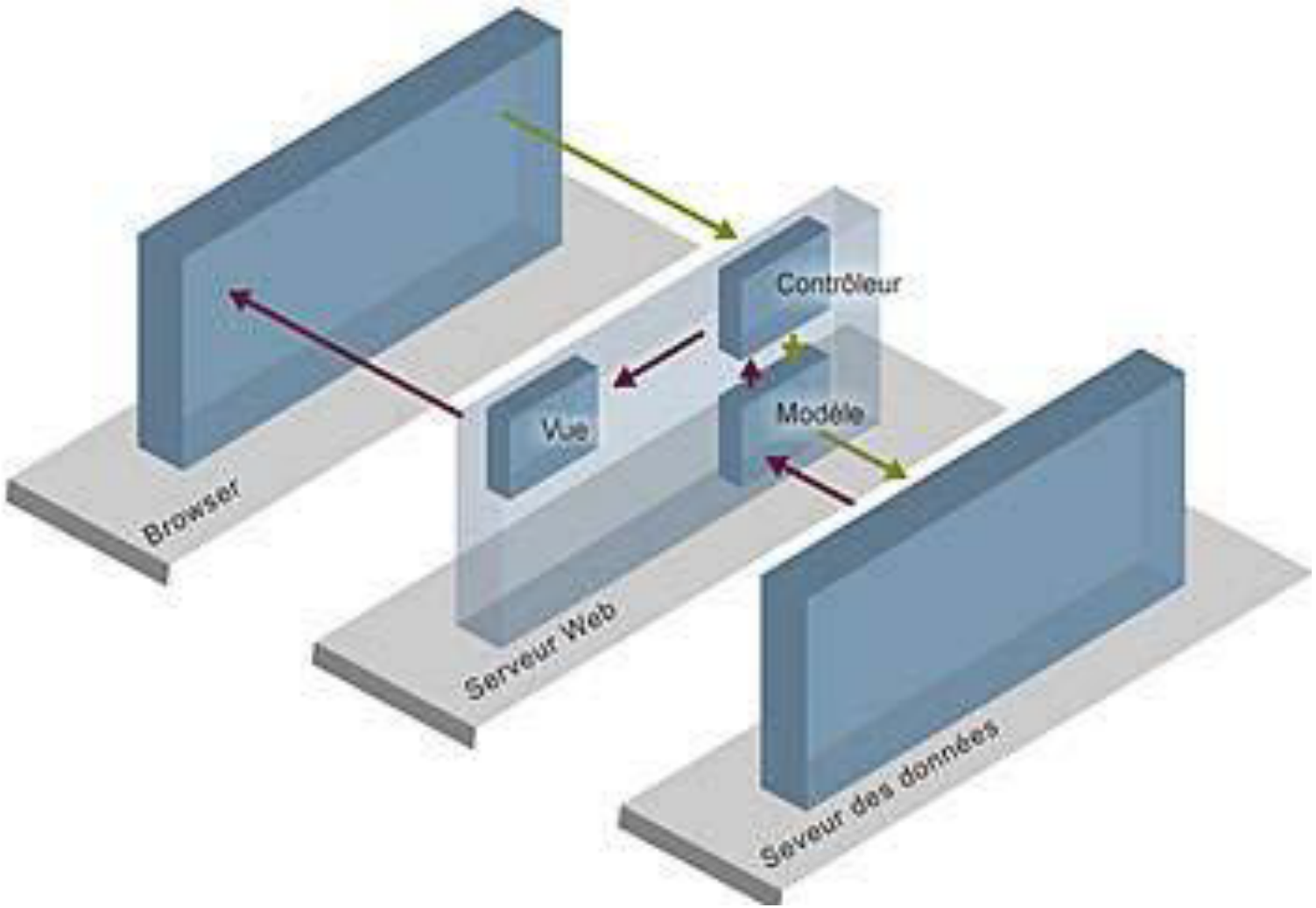
- JSP seules, marche bien quand :

- L'*output est de type caractère. Ex : HTML*
- La forme/présentation est stable.

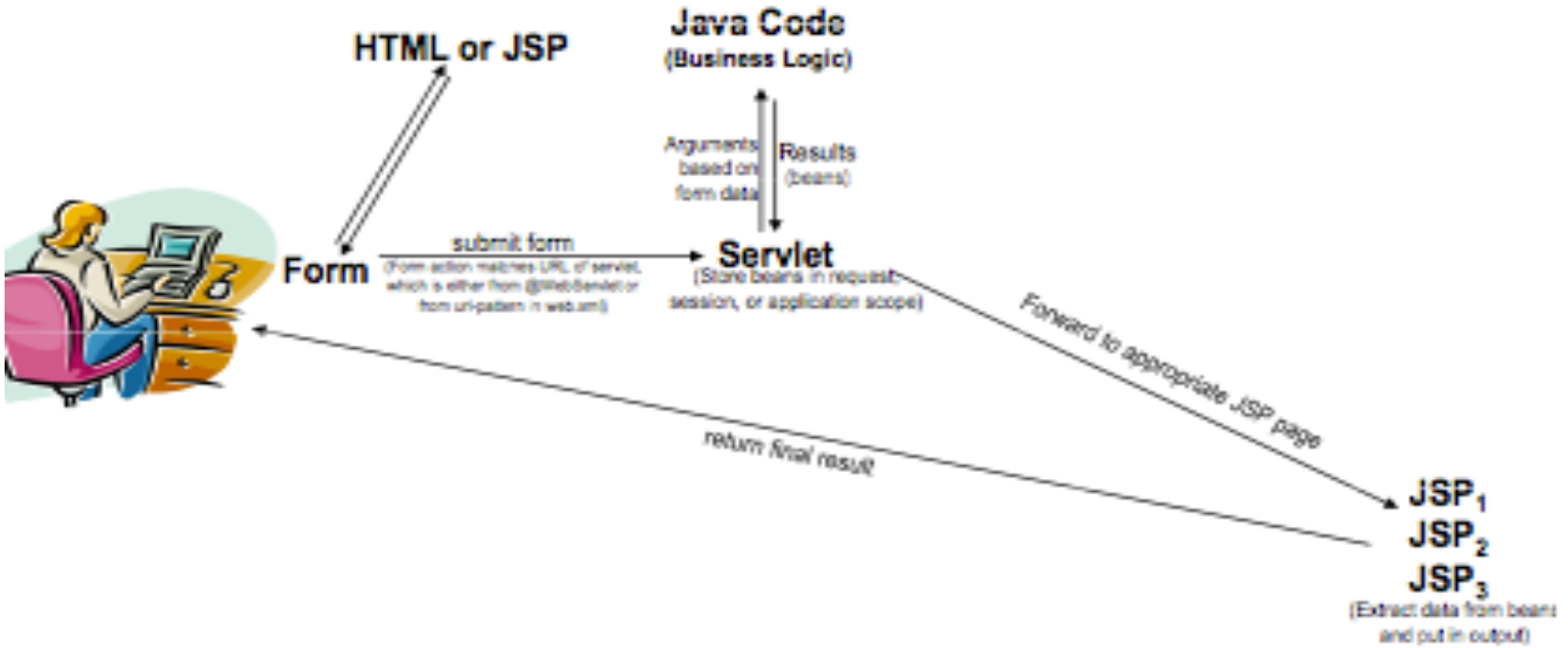
- Architecture MVC, Nécessaire quand :

- Même requête peut donner des résultats différents.
- Des équipes de Dev, Ex : Web designer et logique métier.
- Traitements complexes des données mais une présentation simple.

ARCHITECTURE :



MVC EN 5 ÉTAPES :

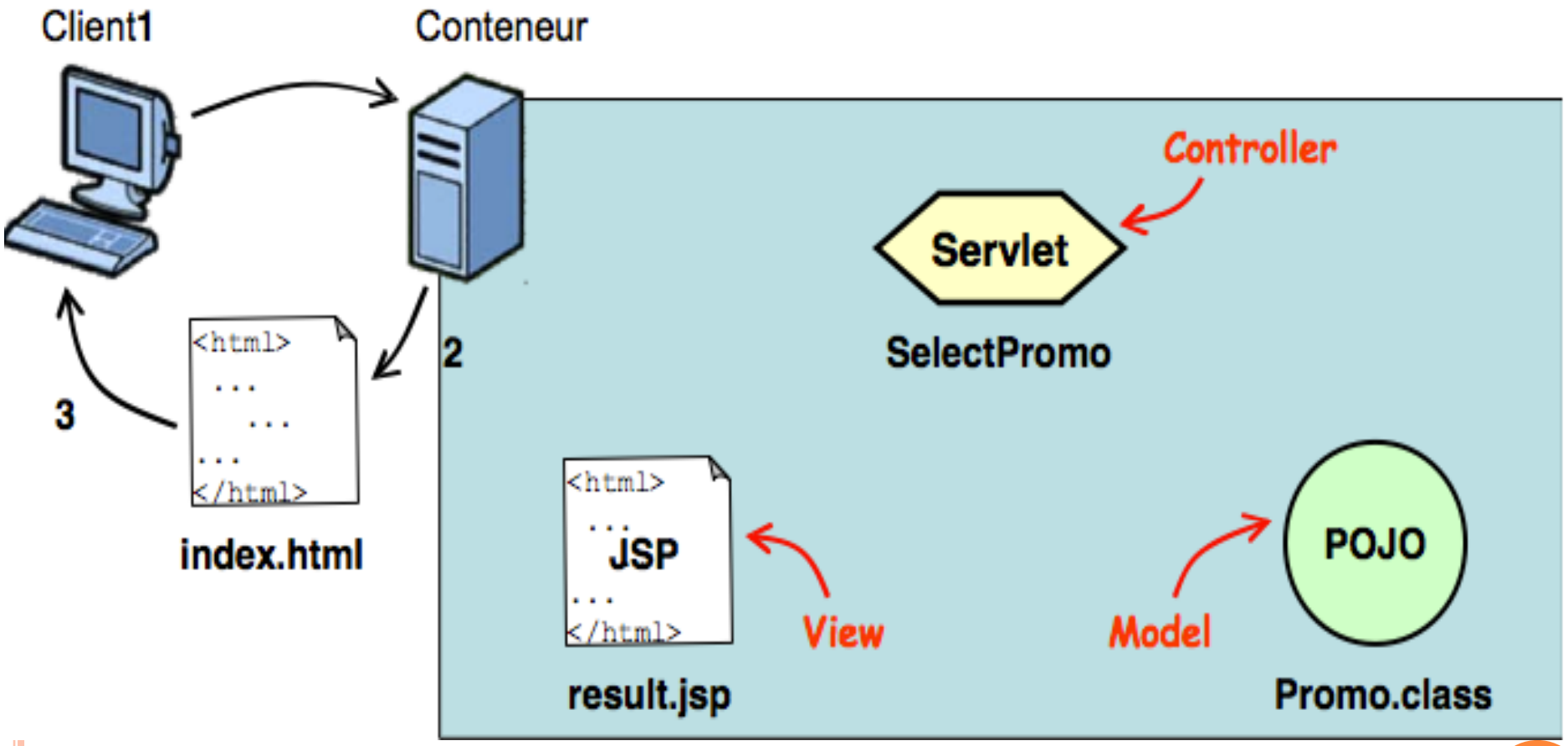


IMPLEMENTING MVC AVEC REQUESTDISPATCHER

1. **Define beans to represent result data**
2. **Use a servlet to handle requests**
3. **Obtain bean instances**
   The servlet invokes business logic (application-specific code) or data-access code to obtain the results.
4. **Store the bean in the request, session, or servlet context**
   The servlet calls setAttribute on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request
5. **Forward the request to a JSP page.**
   The servlet determines which JSP page is appropriate to the situation and uses the forward method of RequestDispatcher to transfer control to that page.
6. **Extract the data from the beans.**

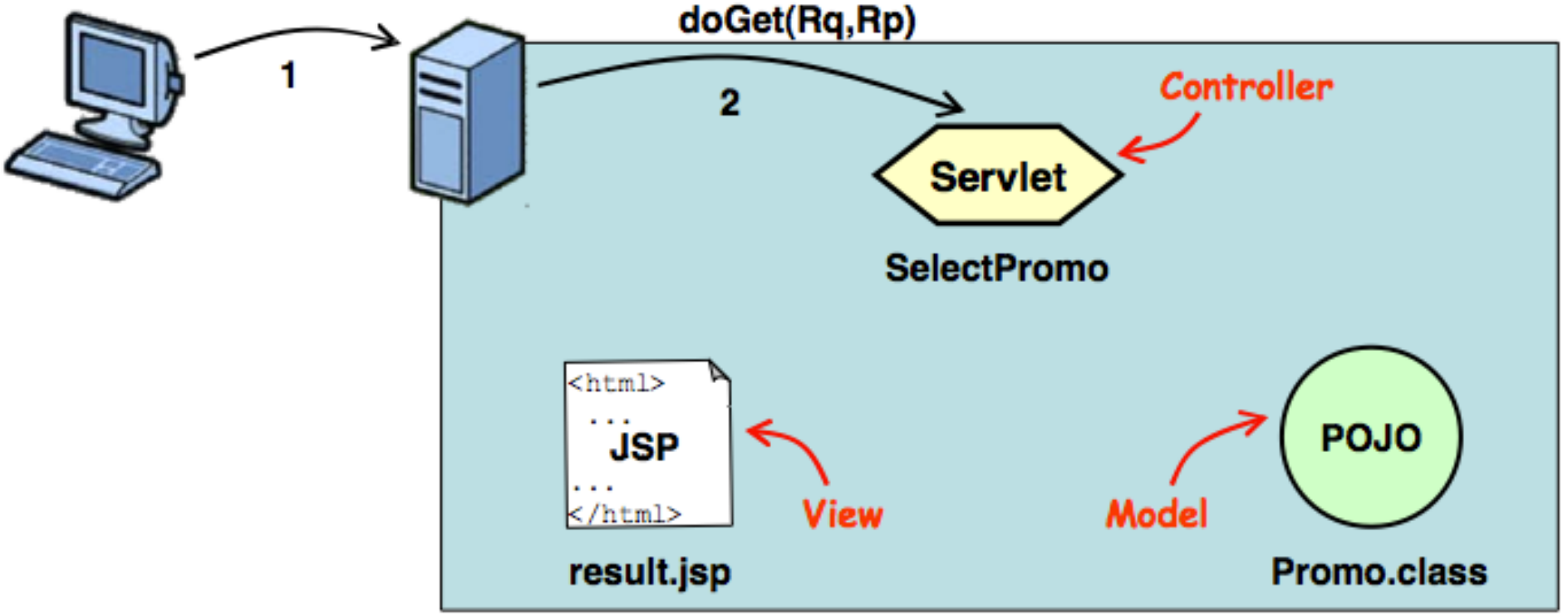
ETAPE 1 :

- le client récupère le formulaire (URL)



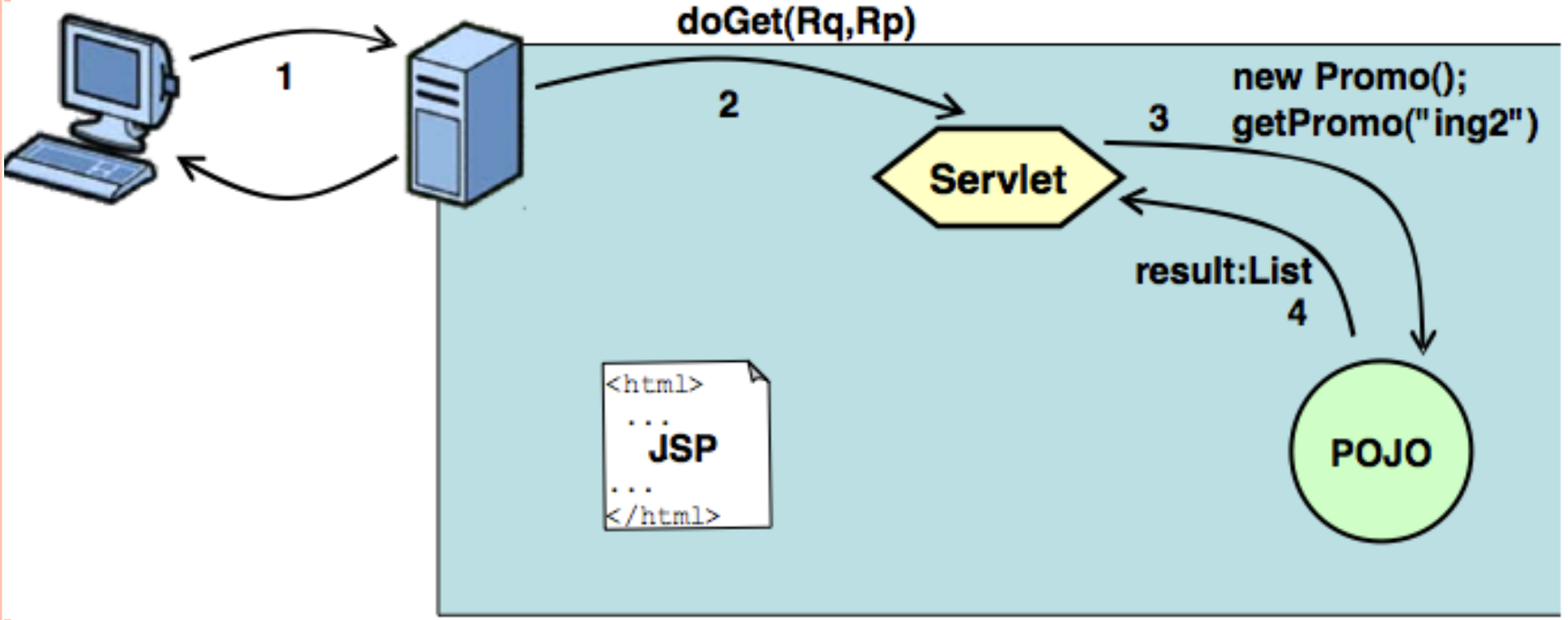
ETAPE 2 :

- Le client envoie son formulaire (GET/POST)
- Le conteneur transmet au servlet correspondant (*controller*) *et récupère les paramètres.*



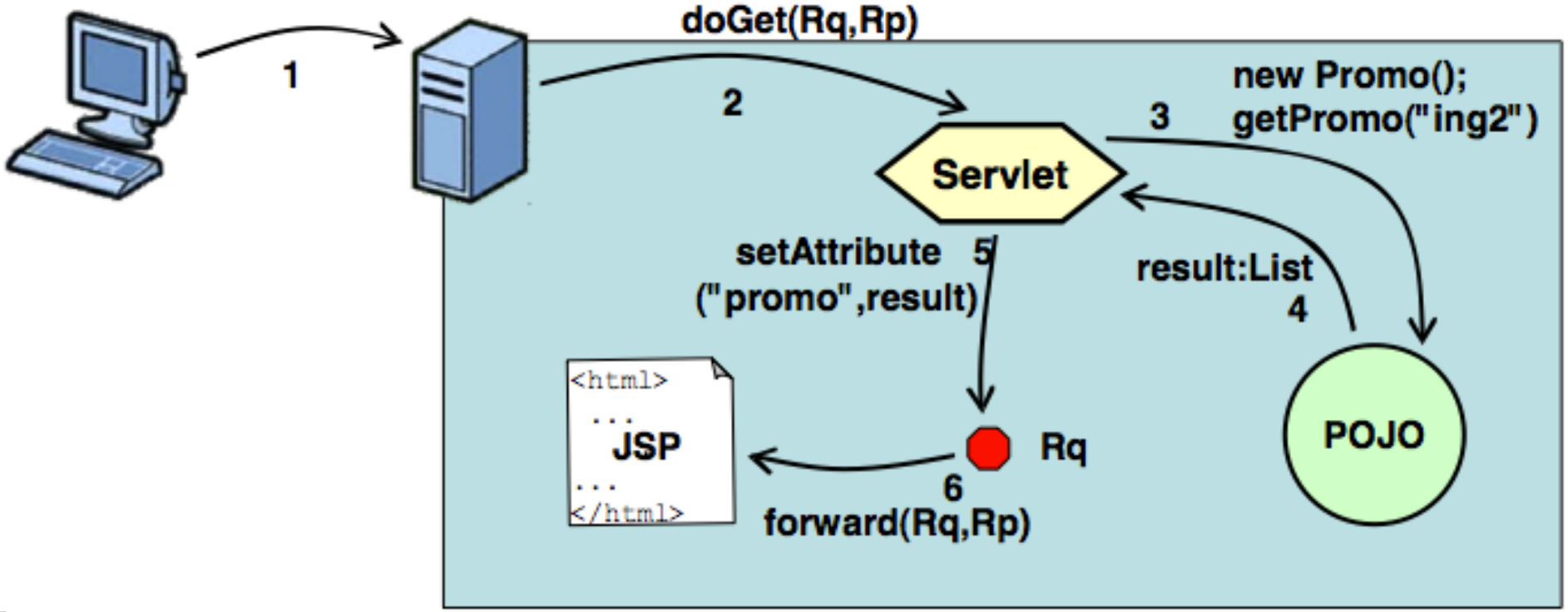
ETAPE 3 :

- Le servlet *controller interroge le model.*
- Le *model retourne au controller le résultat correspondant*



ETAPE 4 :

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

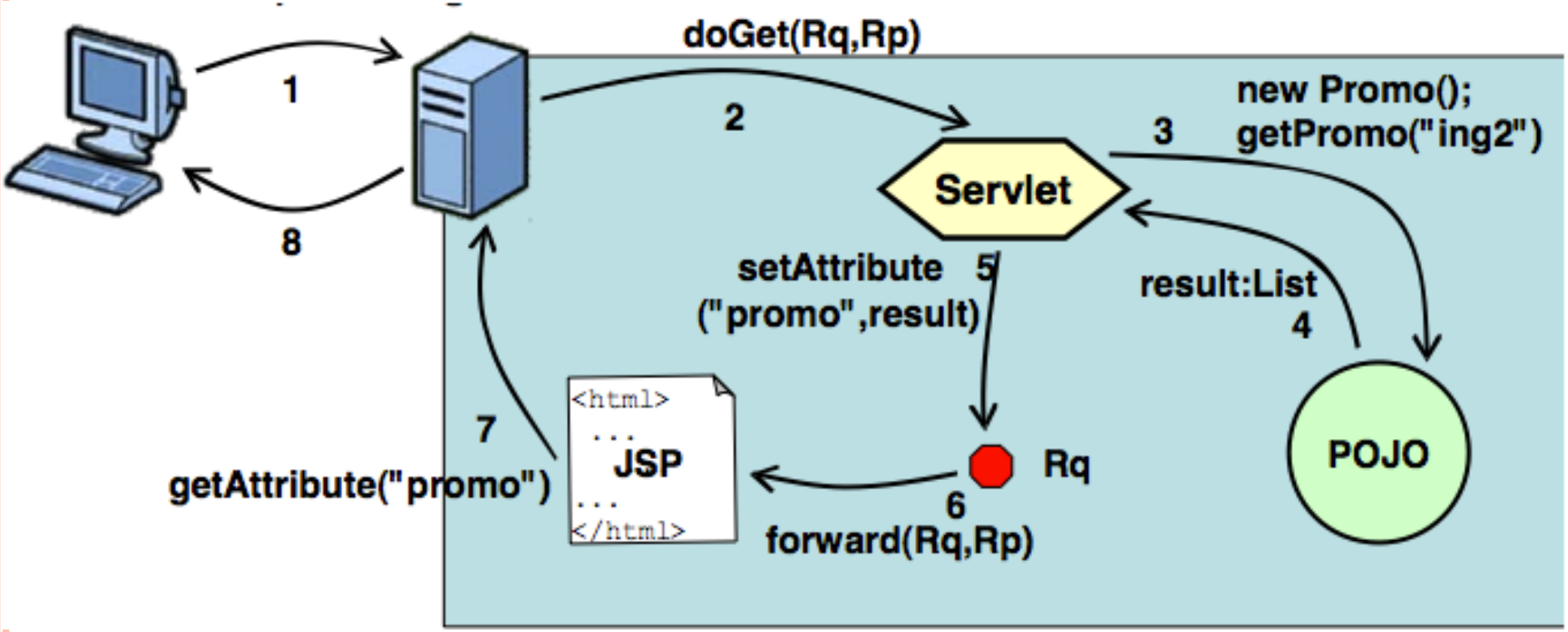


ETAPE 4 :

```
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
```

ETAPE 5 :

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



ETAPE 5 :

- **Servlet**

```
request.setAttribute("key", value);
RequestDispatcher dispatcher =
  request.getRequestDispatcher
                              ("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

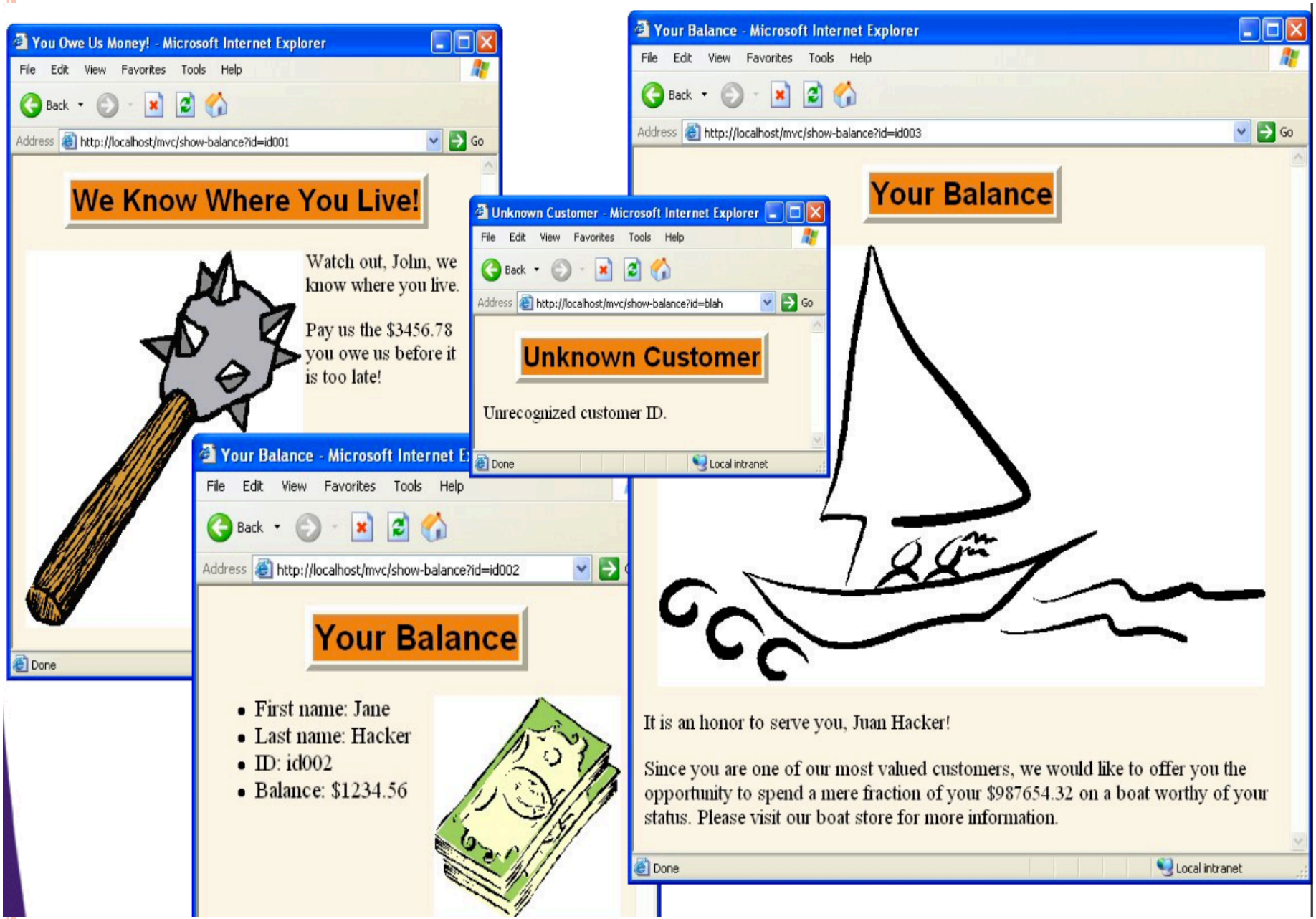
- **JSP 1.2**

```
<jsp:useBean id="key" type="somePackage.ValueObject"
             scope="request" />
<jsp:getProperty name="key" property="someProperty" />
```

- **JSP 2.0**

```
${key.someProperty}
```

Exercice : une petite Application de Banque



Bean Client

```
public class Customer {
  private final String id, firstName, lastName;
  private final double balance;

  public Customer(String id,
                  String firstName,
                  String lastName,
                  double balance) {
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.balance = balance;
  }

  // getId, getFirstName, getLastName, getBalance. No setters.

  public double getBalanceNoSign() {
    return(Math.abs(balance));
  }
}
```

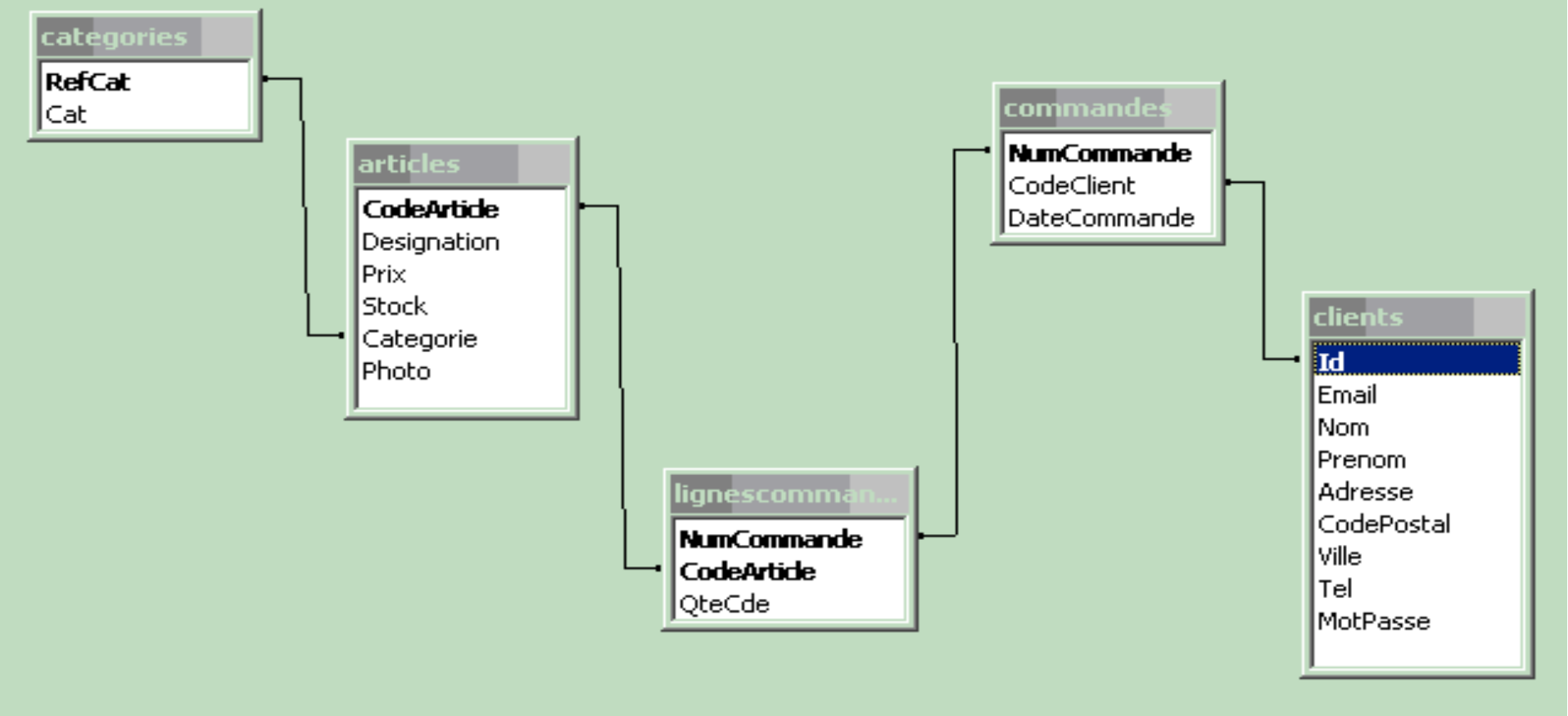
Since the constructor is called from Java only (never from JSP), the requirement for a zero-arg constructor is eliminated. Also, since bean state is set only with constructor, rather than with jsp:setProperty, we can eliminate setter methods and make the class immutable.

Servlet Code

```
public class ShowBalance extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String customerId = request.getParameter("customerId");
    CustomerLookupService service = new CustomerSimpleMap();
    Customer customer = service.findCustomer(customerId);
    request.setAttribute("customer", customer);
    String address;
    if (customer == null) {
      request.setAttribute("badId", customerId);
      address = "/WEB-INF/results/unknown-customer.jsp";
    } else if (customer.getBalance() < 0) {
      address = "/WEB-INF/results/negative-balance.jsp";
    } … /* normal-balance and high-balance cases*/ …}
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
```

Micro-Projet

Notre base de données



- L'identifiant client est un entier auto incrémenté.
- Un internaute s'identifie par le couple (mail,password).
- Mail est déclaré « unique ».

- **Navigation souhaitée**.
- Nous réaliserons progressivement ce mini site.
- Stocker l'identifiant client en session.