Jakarta EE : Servlet

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

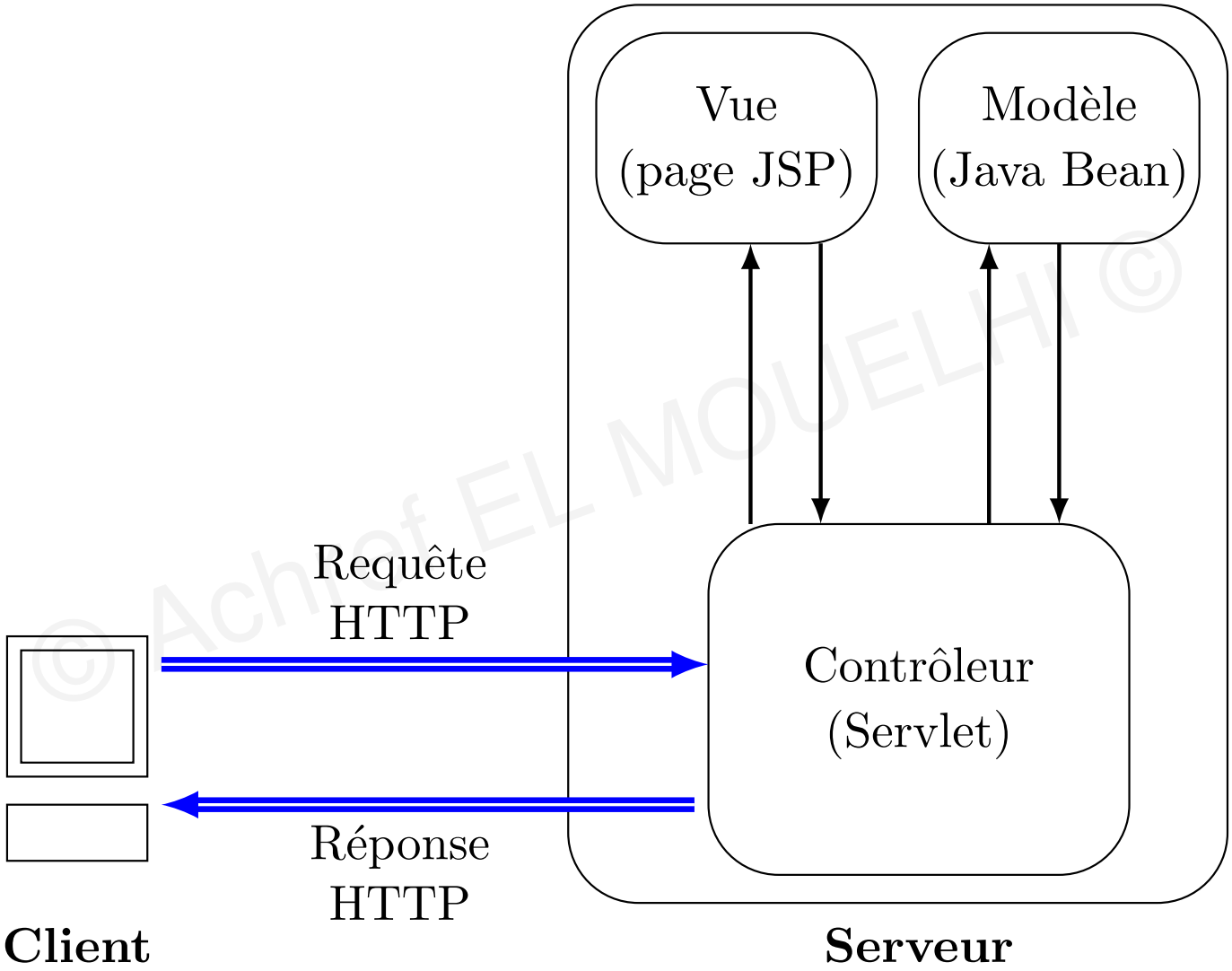
Plan

Jakarta EE

Servlet : le cœur d'une application **Jakarta EE**

- Classe **Java** héritant de la classe `HttpServlet` qui elle-même hérite de `GenericServlet`, qui implémente l'interface `Servlet`
- Recevant une requête **HTTP** (de type `GET`, `POST`...) et retournant une réponse **HTTP**
- Contrôleur du modèle MV**C** dans une application **Jakarta EE**

Jakarta EE



Jakarta EE

Architecture MVC (contexte)

- **Modèle : JavaBean / POJO / Services**
  - **JavaBean** : classe **Java** respectant une convention stricte ⇒ constructeur vide, attributs privés, getters/setters publics, sérialisable. Utilisé pour transporter des données entre couches (formulaire utilisateur).
  - **POJO** (**P**lain **O**ld **J**ava **O**bject) : classe **Java** ordinaire sans dépendance spécifique (ni interface ni annotation).
  - **Service** : composant métier (souvent une classe **Java**) qui contient la logique de traitement et interagit avec les **DAO** (**D**ata **A**ccess **O**bjects). Il ne contient pas de code de présentation.
- **Contrôleur : Servlet** ⇒ reçoit la requête **HTTP**, appelle le service, place les données dans la requête (via `setAttribute`) et redirige vers la vue.
- **Vue : JSP** ⇒ responsable uniquement de l'affichage (**HTML** + **EL** + **JSTL**). Éviter le code **Java** direct (scriptlets).

Jakarta EE

Servlet : classe Java héritant de `HttpServlet`

```
package org.eclipse.controller;

import javax.servlet.http.HttpServlet;

public class TestServlet extends HttpServlet {

}
```

Jakarta EE

Explication

- `HttpServlet` contient des méthodes abstraites, préfixées par `do()`, associées aux différentes méthodes (verbes) **HTTP**.
  - `doGet()` : s'exécute quand l'utilisateur demande une page (via la barre d'adresse, un lien hypertexte...)
  - `doPost()` : s'exécute quand l'utilisateur envoie des données via un formulaire par exemple
  - ...
- Chaque méthode prend en paramètre :
  - `HttpServletRequest` : contenant des informations sur la requête utilisateur
  - `HttpServletResponse` : permettant de personnaliser la réponse à retourner à l'utilisateur

Jakarta EE

Ajoutons les méthodes `doGet()` **et** `doPost()` **à** `TestServlet`

```
package org.eclipse.controller;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request, HttpServletResponse response) {
  }

  protected void doPost(HttpServletRequest request, HttpServletResponse response) {

  }
}
```

Jakarta EE

Mais quand cette Servlet sera exécutée ?

Quand l'utilisateur saisit une **URL** dans le navigateur, il envoie une requête **HTTP** à notre contrôleur (qui est en vrai une Servlet)

Jakarta EE

Mais quand cette Servlet sera exécutée ?

Quand l'utilisateur saisit une **URL** dans le navigateur, il envoie une requête **HTTP** à notre contrôleur (qui est en vrai une Servlet)

Et si on avait plusieurs Servlets, laquelle sera exécutée ?

- Chaque Servlet aura sa propre route (uniques)
- La Servlet ayant la route demandée sera exécutée

Jakarta EE

Comment associer une route à une Servlet ?

- soit avec l'annotation `@WebServlet`
- soit dans le fichier `web.xml`

Jakarta EE

Comment associer une route à une Servlet ?

- soit avec l'annotation `@WebServlet`
- soit dans le fichier `web.xml`

Commençons par créer une Servlet avec **Eclipse**

Jakarta EE

Pour créer une **Servlet** sous **Eclipse**

- Faire un clic droit sur `src` situé dans `Java Resources` de notre projet
- Aller dans `New` et choisir `Servlet`
- Remplir le champ `Java package:` par `org.eclipse.controller` (par example)
- Remplir le champ `Class name:` par un nom suffixé par le mot `Servlet` : TestServlet (par example)
- Cliquer sur `Next`

Jakarta EE

Routage par annotation (par défaut)

- On peut modifier ou supprimer l'`URL Mappings`. Remplaçons la chaîne existante (`/TestServlet`) par `/mapage`
- Cliquer sur `Next`
- Décocher la case `Constructors from superclass`
- Vérifier que les cases correspondantes aux deux méthodes `doGet()` et `doPost` sont cochées
- Valider en cliquant sur `Finish`

Jakarta EE

Le contenu généré par Eclipse

```
package org.eclipse.controller;

// les imports

@WebServlet("/mapage")
public class TestServlet extends HttpServlet {

  private static final long serialVersionUID = 1L;

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {
    response.getWriter().append("Served at: ").append(request.getContextPath());
  }

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Jakarta EE

Le fichier `web.xml`

- Placé dans le dossier `WEB-INF/` d'une application web.
- Sert à **décrire la configuration du déploiement** de l'application.
- Appelé **Deployment Descriptor** car il :
  - décrit comment l'application doit être déployée,
  - précise quelles servlets, filtres, écouteurs et paramètres utiliser,
  - contient des métadonnées interprétées par le conteneur (Tomcat, Jetty...).

Jakarta EE

Rôle historique du `Deployment Descriptor`

- Avant Servlet 3.0, la configuration se faisait exclusivement dans `web.xml`.
- Depuis **Servlet 3.0 (Java EE 6, 2010)**, on peut remplacer `web.xml` par des annotations.
- Le conteneur scanne automatiquement les classes pour détecter :
  - `@WebServlet` : pour déclarer une servlet
  - `@WebFilter` : pour un filtre
  - `@WebListener` : pour un écouteur

Jakarta EE

Si le fichier n'existe pas

- Faire un clic droit sur `WEB-INF` de `webapp` de notre projet
- Aller dans `New` et choisir `Other`
- Saisir `xml` dans la zone de recherche
- Choisir `XML File`
- Cliquer sur `Next` et choisir le nom `web.xml`

Jakarta EE

Contenu de web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="https://jakarta.ee/xml/ns/jakartaee"
   xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
   https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd" id="
   WebApp_ID" version="5.0">

  <display-name>cours-jee</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Jakarta EE

`<welcome-file-list>`

- Contient les différent formats fichiers qui peuvent être utilisés comme page d'accueil de l'application
- Ces fichiers seront directement dans `webapp`
- Pas besoin d'une Servlet pour les afficher
- Accessibles via la route `/` ou directement via leur nom

Jakarta EE

Dans `web.xml`**, on déclare la Servlet avant** `</web-app>`

```
...
<servlet>
  <servlet-name>TestServlet</servlet-name>
  <servlet-class>org.eclipse.controller.TestServlet</servlet-class>
</servlet>
```

Jakarta EE

Dans `web.xml`, on déclare la Servlet avant `</web-app>`

```
...
<servlet>
  <servlet-name>TestServlet</servlet-name>
  <servlet-class>org.eclipse.controller.TestServlet</servlet-class>
</servlet>
```

Explication

- `<servlet>` et `</servlet>` : déclaration de la Servlet
- `<servlet-name>` et `</servlet-name>` : permet d'attribuer un nom à la Servlet qu'on utilisera plus tard
- `<servlet-class>` et `</servlet-class>` : indique le chemin de la classe de la Servlet

Jakarta EE

Autres sous balises disponibles pour Servlet

- `<description>` et `</description>` : ajouter une description sur le fonctionnement de la Servlet (comme un commentaire)
- `<load-on-startup>` et `</load-on-startup>` : permet de forcer le chargement de la Servlet lors de démarrage
- ...

Jakarta EE

N'oublions pas, le rôle du `web.xml` :

- déclarer la Servlet (**c'est fait**)
- faire le mapping (assurer le routage si cela n'a pas été fait avec les annotations)

Jakarta EE

```
...
<servlet-mapping>
  <servlet-name>TestServlet</servlet-name>
  <url-pattern>/mapage</url-pattern>
</servlet-mapping>
</web-app>
```

Jakarta EE

```
...
<servlet-mapping>
  <servlet-name>TestServlet</servlet-name>
  <url-pattern>/mapage</url-pattern>
</servlet-mapping>
</web-app>
```

Explication

- `<servlet-mapping>` et `</servlet-mapping>` : pour faire le mapping Servlet/url
- `<servlet-name>` et `</servlet-name>` : permet d'indiquer le nom de la Servlet à appeler
- `<url-pattern>` et `</url-pattern>` : indique l'URL qui provoquera l'appel de la Servlet indiquée dans la la sous-balise précédente

Jakarta EE

Contenu de `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="
   https://jakarta.ee/xml/ns/jakartaee" xsi:schemaLocation="https://
   jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-
   app_5_0.xsd" id="WebApp_ID" version="5.0">

<servlet>
  <servlet-name>TestServlet</servlet-name>
  <servlet-class>org.eclipse.controller.TestServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>TestServlet</servlet-name>
  <url-pattern>/mapage</url-pattern>
</servlet-mapping>
</web-app>
```

Jakarta EE

Remarque

Dans la suite de ce cours, on n'utilisera que le routage par annotation.

Jakarta EE

Une seule étape à faire

- Cliquer sur `Run`
- Une page blanche affichée ayant comme
  - adresse : `http://localhost:8080/cours-jee/mapage`
  - contenu : `Served at: /cours-jee`

Jakarta EE

Si on teste une autre URL inexistante

- Écrire dans la zone d'adresse
  `http://localhost:8080/cours-jee/tapage`
- Une page HTTP 404 sera affichée

Jakarta EE

Comment afficher le `Hello World`

Il faut modifier la `Servlet` (l'objet `HttpServletResponse` qui est responsable de la réponse)

Jakarta EE

Nouveau contenu de la Servlet

```
public class TestServlet extends HttpServlet {

  private static final long serialVersionUID = 1L;

  protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

    response.getWriter().print("Hello World");
  }

  protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
    doGet(request, response);
  }
}
```

Jakarta EE

Pour exécuter une deuxième fois

- Cliquer sur `Run`
- Choisir `Continue without restarting` (pas besoin de redémarrer le serveur)

Jakarta EE

On peut indiquer l'encodage et le type du contenu de la réponse

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {

  // pour indiquer le type de réponse
  response.setContentType("text/html");

  // indiquer l'encodage UTF-8 pour éviter les problèmes avec les
     accents
  response.setCharacterEncoding("UTF-8");

  PrintWriter out = response.getWriter();
  out.println("Hello World");
}
```

L'objet `PrintWriter`

- s'obtient de l'objet `response`
- permet d'envoyer un (ou des) message(s) à l'utilisateur

Jakarta EE

Pour retourner une page HTML complète

```
protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException{
     response.setContentType("text/html");
     response.setCharacterEncoding("UTF-8");
     PrintWriter out = response.getWriter();
     out.println("<!DOCTYPE html>");
     out.println("<html>");
     out.println("<head>");
     out.println("<meta charset=\"utf-8\" >");
     out.println("<title>Projet JEE</title>");
     out.println("</head>");
     out.println("<body>");
     out.println("Hello World");
     out.println("</body>");
     out.println("</html>");
}
```

Jakarta EE

Constats

- Beaucoup de code dans la **Servlet** (trop long) pour un affichage. simple.
- Violation du modèle **MVC** : le contrôleur n'affiche pas de résultat.
- Selon le modèle **MVC**, le Contrôleur (**Servlet**) doit uniquement gérer la logique métier (traitement des données, sélection du modèle) et la Vue (**JSP**) doit gérer l'affichage.
- En insérant du code **HTML** dans la **Servlet**, ces rôles sont mélangés et le concept **Separation of Concerns** (Séparation des préoccupations) n'est pas respecté.

Jakarta EE

Constats

- Beaucoup de code dans la **Servlet** (trop long) pour un affichage. simple.
- Violation du modèle **MVC** : le contrôleur n'affiche pas de résultat.
- Selon le modèle **MVC**, le Contrôleur (**Servlet**) doit uniquement gérer la logique métier (traitement des données, sélection du modèle) et la Vue (**JSP**) doit gérer l'affichage.
- En insérant du code **HTML** dans la **Servlet**, ces rôles sont mélangés et le concept **Separation of Concerns** (Séparation des préoccupations) n'est pas respecté.

Solution

Utiliser des vues pour l'affichage (chapitre suivant).

Jakarta EE

A une Servlet peuvent être associées plusieurs routes

```
@WebServlet({"/route1", "/route2",... /routeN})
```

Remarque

Pour récupérer la route qui a permis d'exécuter la Servlet, on utilise l'objet de type `HttpServletRequest`.

Jakarta EE

`http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2`

Jakarta EE

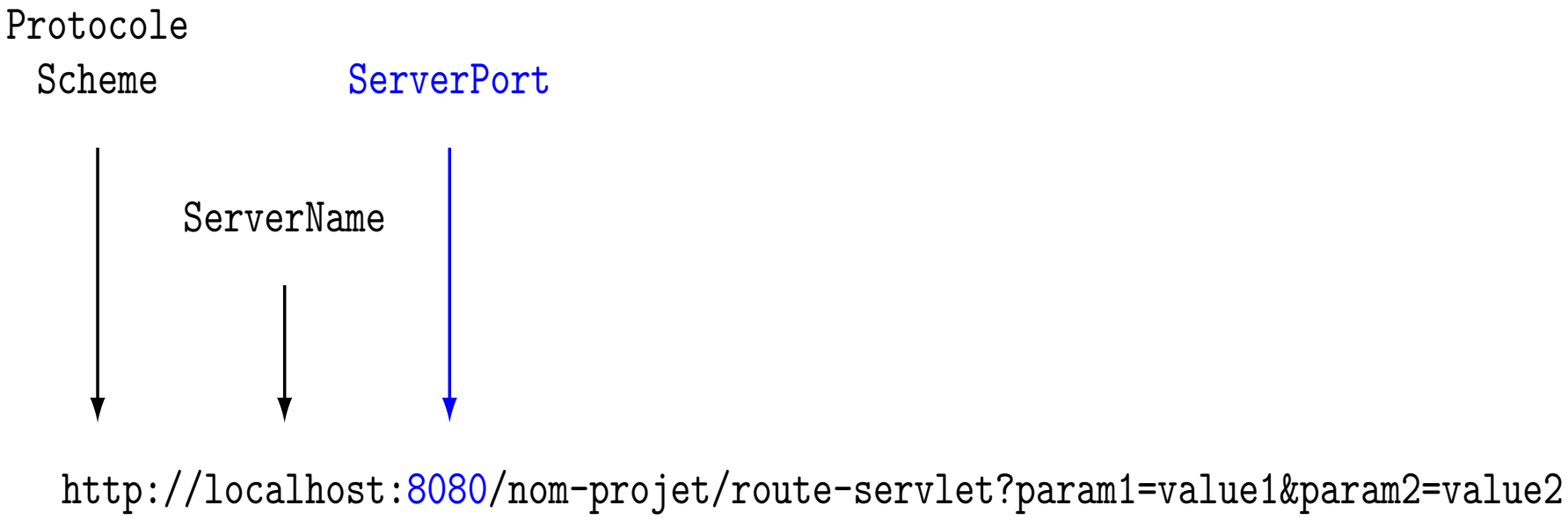
Protocole
Scheme

http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

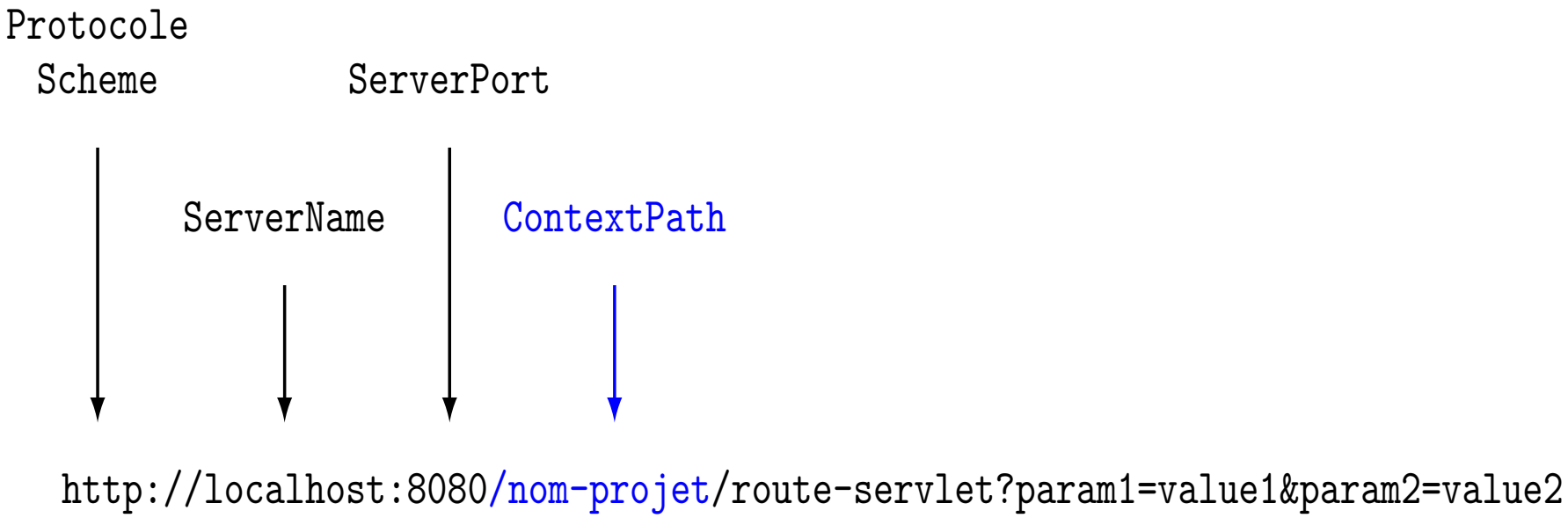
Jakarta EE


Protocole
Scheme
ServerName
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

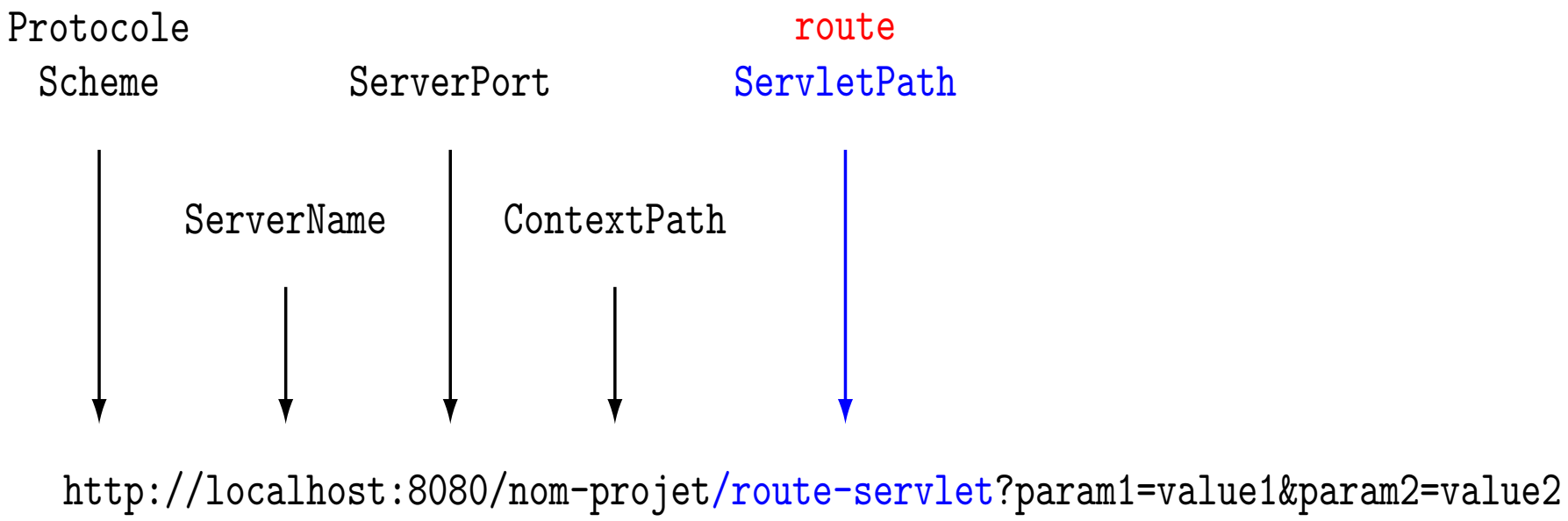
Jakarta EE


Protocole
Scheme
ServerPort
ServerName
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

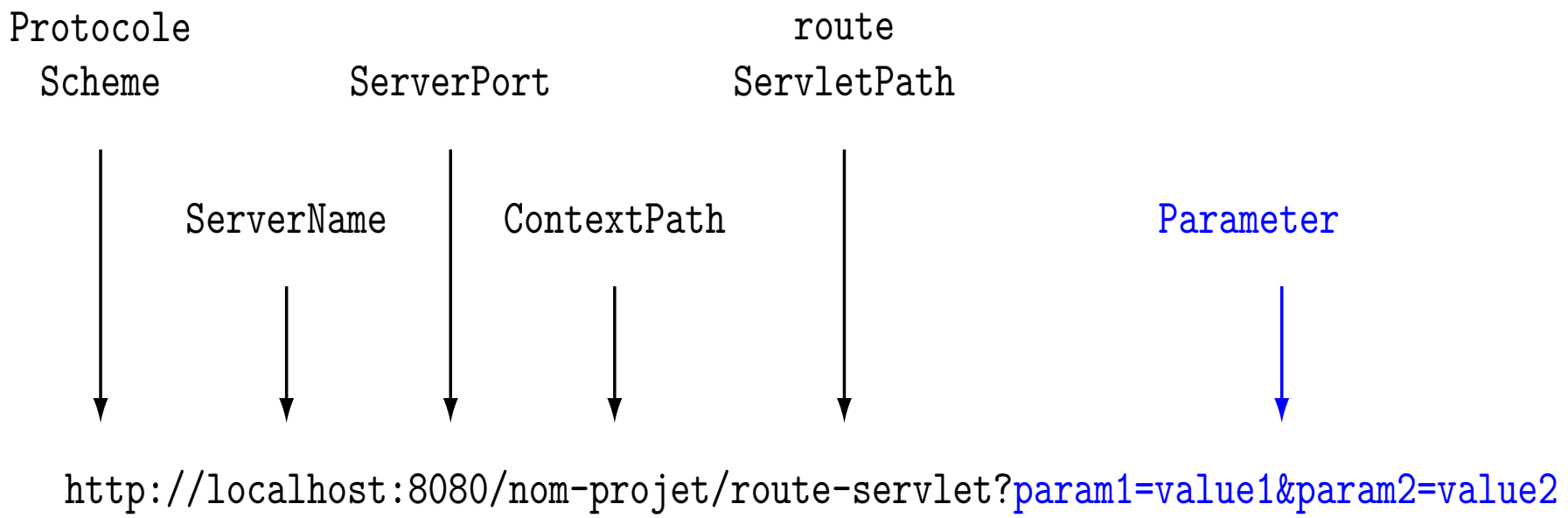
Jakarta EE


Protocole
Scheme
ServerPort
ServerName
ContextPath
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

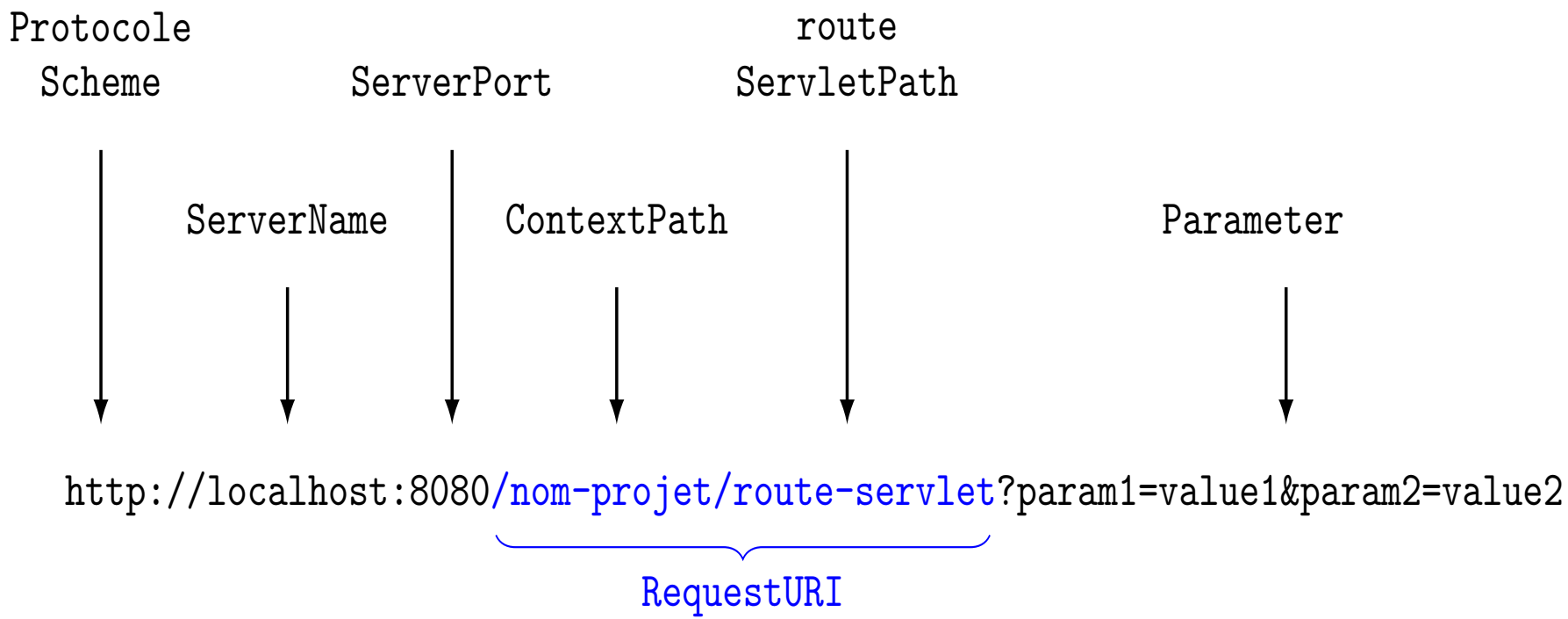
Jakarta EE


Protocole
Scheme
ServerPort
route
ServletPath
ServerName
ContextPath
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

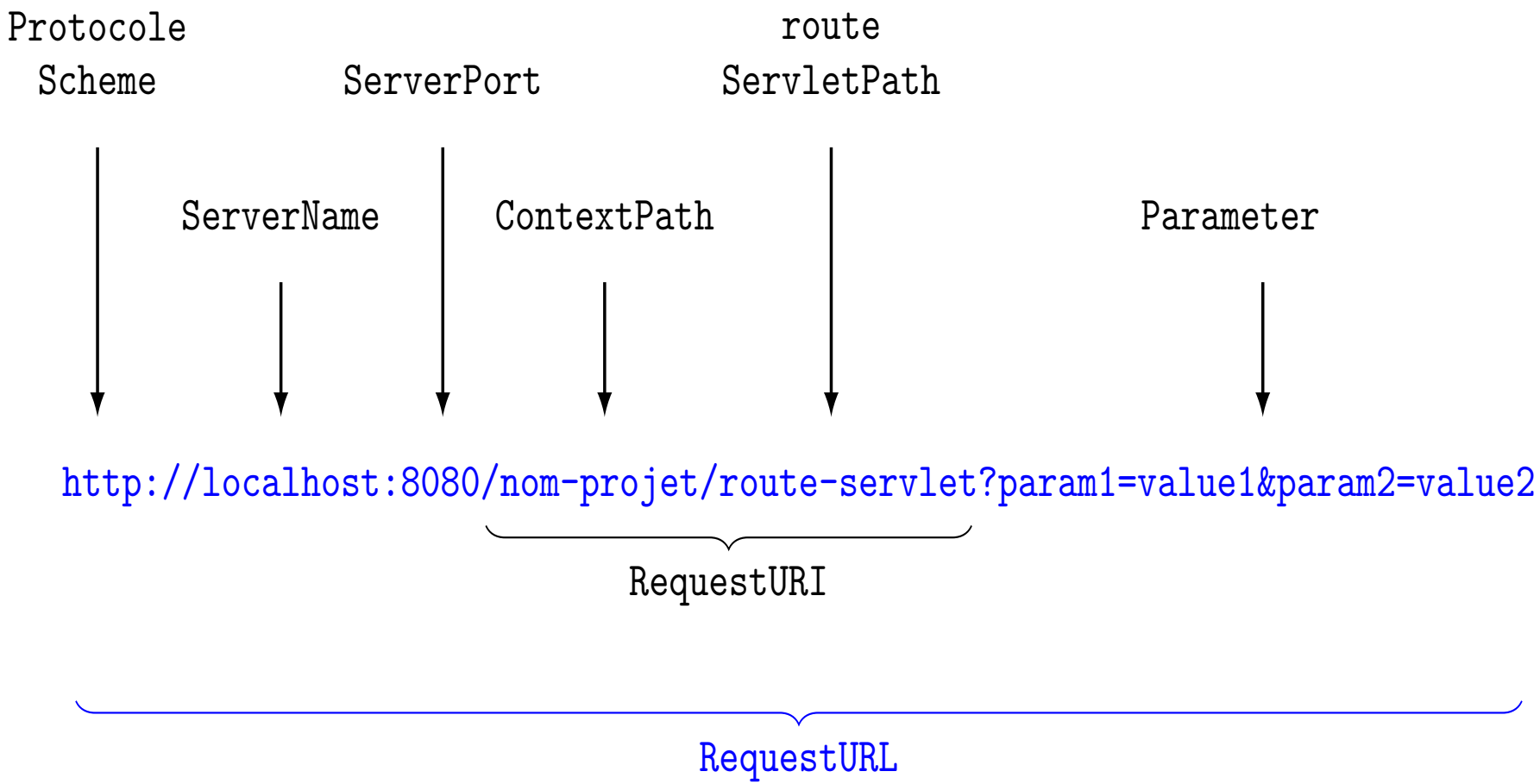
Jakarta EE


Protocole
Scheme
ServerPort
route
ServletPath
ServerName
ContextPath
Parameter
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

Jakarta EE


Protocole
Scheme
ServerPort
route
ServletPath
ServerName
ContextPath
Parameter
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2
RequestURI

Jakarta EE


Protocole
Scheme
ServerPort
route
ServletPath
ServerName
ContextPath
Parameter
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2
RequestURI
RequestURL

Jakarta EE

Comment récupérer toutes ces informations ?

Tout est défini dans l'objet de type `HttpServletRequest`.

Jakarta EE

Comment récupérer toutes ces informations ?

Tout est défini dans l'objet de type `HttpServletRequest`.

Exemples

Il suffit de préfixer le nom des propriétés précédentes par `get`

- `request.getContextPath()`
- `request.getServletPath()`
- `request.getServerPort()`
- ...

Jakarta EE

Exercice 1

- Créez une Servlet `CalculServlet` accessible via une des quatre routes suivantes
  - `/calcul/plus`
  - `/calcul/moins`
  - `/calcul/fois`
  - `/calcul/div`
- La Servlet doit afficher chaque fois la deuxième partie de la route demandé
  - `plus`
  - `moins`
  - `fois`
  - `div`

Jakarta EE

Récupérer les paramètres d'une requête

- Mais, une requête peut avoir de paramètres (par example `/mapage?nom=Wick&prenom=John`)
- Comment, dans ce cas, récupérer les paramètres ?

Jakarta EE

Récupérer les paramètres d'une requête

- Mais, une requête peut avoir de paramètres (par example `/mapage?nom=Wick&prenom=John`)
- Comment, dans ce cas, récupérer les paramètres ?

Solution

```
request.getParameter("nomParameter");
```

Jakarta EE

Exemple de récupération et d'affichage de paramètres de la requête

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {

    String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");
    PrintWriter out = response.getWriter();
    out.print("Hello " + nom + " " + prenom);
}
```

Jakarta EE

Exemple de récupération et d'affichage de paramètres de la requête

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {

    String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");
    PrintWriter out = response.getWriter();
    out.print("Hello " + nom + " " + prenom);
}
```

Remarques

- `request.getParameter(String)` retourne `null` si le paramètre n'est pas présent dans la requête.
- L'utilisation ultérieure de cette valeur `null` peut provoquer une `NullPointerException` (en appelant une méthode sur celle-ci par exemple)

Jakarta EE

Il est recommandé de vérifier si la valeur du paramètre est non nulle avant de l'utiliser

```
protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws ServletException,
   IOException {
     String nom = request.getParameter("nom");
     String prenom = request.getParameter("prenom");
     PrintWriter out = response.getWriter();
     if (nom != null && prenom != null) {

         out.print("Hello " + nom + " " + prenom);
     } else {
         out.print("Hello world");
     }
}
```

Récupérer les paramètres d'une requête

À ne pas confondre

- Les paramètres de requête : concept relatif aux requêtes **HTTP**
- Les attributs de requête : concept introduit dans **JEE** (à voir dans le prochain chapitre)

Jakarta EE

Exercice 2

Modifiez `CalculServlet` pour qu'elle

- accepte deux paramètres `a` et `b`
- Si la route demandée est
  - `/calcul/plus?a=2&b=5`, alors la Servlet affiche le résultat de l'addition.
  - `/calcul/moins?a=2&b=5`, alors la Servlet affiche le résultat de la soustraction.
  - `/calcul/fois?a=2&b=5`, alors la Servlet affiche le résultat de la multiplication.
  - `/calcul/div?a=2&b=5`, alors la Servlet affiche le résultat de la division.

Récupérer les paramètres d'une requête

Séquence de travail pour l'exercice

1. Utiliser `request.getServletPath()` pour déterminer quelle opération effectuer (`/calcul/plus`, `/calcul/moins`...).
2. Convertir les chaînes de caractères (`String`) des paramètres en nombres (`int` ou `double`) pour effectuer le calcul.
3. Effectuer le calcul nécessaires selon l'opérateur demandé.
4. Gérer les cas d'erreur, comme la division par zéro (`/div`) ou des paramètres non numériques.

Jakarta EE

Si le paramètre est présent plusieurs fois dans l'URL avec le même nom, alors on peut utiliser `getParameterValues()`

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    String[] noms = request.getParameterValues("nom");
    if (noms != null) {
        for (String nom : noms) {
            out.print("Hello " + nom);
        }
    }
}
```

Jakarta EE

Si le paramètre est présent plusieurs fois dans l'URL avec le même nom, alors on peut utiliser `getParameterValues()`

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    String[] noms = request.getParameterValues("nom");
    if (noms != null) {
        for (String nom : noms) {
            out.print("Hello " + nom);
        }
    }
}
```

Exemple d'**URL** pour tester

```
http://localhost:8080/test-jee/home?nom=wick&nom=dalton
```

Jakarta EE

Exercice 3

- Créez une Servlet `MoyenneServlet` accessible via la route `/moyenne`.
- En allant sur `localhost:8080/cours-jee/moyenne?note=12&note=15`, la moyenne des notes passées en paramètre sera affichée.
- Le nombre de notes est variable.

Jakarta EE

Pour récupérer tous les paramètres dans un `Map`, on peut utiliser `getParameterMap()`

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    Map<String, String[]> params = request.getParameterMap();
    for (var param : params.entrySet()) {
        out.print(param.getKey());
        for (var value : param.getValue()) {
            out.print(value);
        }
    }
}
```

Jakarta EE

Pour récupérer tous les paramètres dans un `Map`, on peut utiliser `getParameterMap()`

```
protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    Map<String, String[]> params = request.getParameterMap();
    for (var param : params.entrySet()) {
        out.print(param.getKey());
        for (var value : param.getValue()) {
            out.print(value);
        }
    }
}
```

Exemple d'**URL** pour tester

```
http://localhost:
8080/test-jee/home?nom=wick&nom=dalton&genre=homme&age=45
```

Jakarta EE

Rediriger vers une autre Servlet annotée par `@WebServlet("/MaServlet")`

```
response.sendRedirect("MaServlet");
```

Jakarta EE

Rediriger vers une autre Servlet annotée par `@WebServlet("/MaServlet")`

```
response.sendRedirect("MaServlet");
```

Ne pas mettre `"/"` avant `MaServlet`.

Jakarta EE

Rediriger vers une autre Servlet annotée par `@WebServlet("/MaServlet")`

```
response.sendRedirect("MaServlet");
```

Ne pas mettre `"/"` avant `MaServlet`.

On peut aussi reconstruire l'URL depuis le `contextPath`

```
response.sendRedirect(request.getContextPath() + "/MaServlet");
```

Jakarta EE

Remarques

- `sendRedirect` envoie une nouvelle requête client : changement d'**URL**.
- L'inclusion/renvoi, `req.getRequestDispatcher("/vue.jsp").forward(req, res)`, envoie une requête interne au serveur ⇒ l'**URL** ne change pas.
- L'inclusion est souvent utilisée pour passer le contrôle du Contrôleur à la Vue sans quitter le contexte de la requête initiale (à voir dans le chapitre suivant).