Projet N°2
MVC : Le Contrôleur avec les SERVLETS

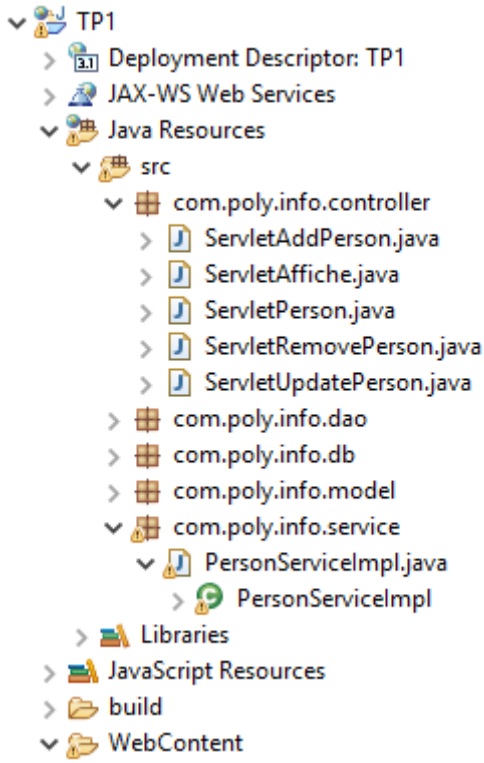
AUDITOIRE : 3 EME GI & IG

1) Création du contrôleur Servlet :

Les Servlets sont l'un des composants JEE les plus utilisés. Elles permettent de gérer des requêtes HTTP et de fournir au client une réponse HTTP et forment ainsi la base de la programmation Web JEE.

Créez un package nommé « **com.poly.nfo.controller** » qui va contenir les **servlets** de notre projet, puis faites un clic droit sur le répertoire `src`, puis choisissez `New > Servlet`.

Voici l'arborescence de votre projet :



La Javadoc nous donne des informations utiles concernant le fonctionnement de cette classe : pour commencer c'est une classe abstraite, ce qui signifie qu'on ne pourra pas l'utiliser telle quelle et qu'il sera nécessaire de passer par une servlet qui en hérite. On apprend ensuite que la classe propose **les méthodes Java nécessaires au traitement des requêtes et réponses http**. Ainsi, on y trouve les méthodes :

- `doGet()` pour gérer la méthode GET ;
- `doPost()` pour gérer la méthode POST ;
- `doHead()` pour gérer la méthode HEAD.

Ce qu'il faut retenir :

- une servlet HTTP **doit hériter** de la classe abstraite **HttpServlet** ;
- une servlet **doit implémenter** au moins une des méthodes doXXX(), afin d'être capable de traiter une requête entrante.

Commençant par la 1ére servlet qui permet d'afficher toutes les personnes.

- **ServletAffiche**

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    // TODO Auto-generated method stub

    PersonServiceImpl P = new PersonServiceImpl();


    request.setAttribute("list", P.getAllPersons());

    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);


}
```

- **ServletAddPerson**

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

    PersonServiceImpl P = new PersonServiceImpl();

    Person person = new Person();

    person.setAge(Integer.parseInt(request.getParameter("age")));
    person.setName(request.getParameter("name"));
    P.addPerson(person);
    request.setAttribute("list", P.getAllPersons());

    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);

}
```

- **ServletRemovePerson**

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

    PersonServiceImpl P = new PersonServiceImpl();
    try {
        P.deletePerson(Integer.parseInt(request.getParameter("id")));
        request.setAttribute("state", true);

    } catch (Exception e) {
        // TODO: handle exception

        request.setAttribute("state", false);

    }

    request.setAttribute("list", P.getAllPersons());
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);
}
```

- **ServletUpdatePerson**

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletE
    PersonServiceImpl P = new PersonServiceImpl();
    Person person = new Person();
    person.setId(Integer.parseInt(request.getParameter("id")));

    person.setAge(Integer.parseInt(request.getParameter("age")));
    person.setName(request.getParameter("name"));
    P.updatePerson(person);
    request.setAttribute("list", P.getAllPersons());

    RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
    rd.forward(request, response);

}
```

2) Mise en place

Concrètement, il va falloir configurer quelque part le fait que notre servlet va être associée à une URL. Ainsi lorsque le client la saisira, la requête HTTP sera automatiquement aiguillée par notre conteneur de servlet vers la bonne servlet, celle qui est en charge de répondre à cette requête. Ce "quelque part" se présente sous la forme d'un simple fichier texte : le fichier **web.xml**.

C'est le cœur de votre application : ici vont se trouver tous les paramètres qui contrôlent son cycle de vie. Nous n'allons pas apprendre d'une traite toutes les options intéressantes, mais y aller par étapes. Commençons donc par apprendre à lier notre servlet à une URL.

Ce fichier de configuration doit impérativement se nommer **web.xml** et se situer juste sous le répertoire **/WEB-INF** de votre application. Si vous avez suivi à la lettre la procédure de création de notre projet web, alors ce fichier est déjà présent. Éditez-le, et supprimez le contenu généré par défaut. Si jamais le fichier est absent de votre arborescence, créez simplement un nouveau fichier XML en veillant bien à le placer sous le répertoire **/WEB-INF** et à le nommer **web.xml**. Voici la structure à vide du fichier :

La mise en place d'une servlet se déroule en deux étapes : nous devons d'abord déclarer la servlet, puis lui faire correspondre une URL.

Définition de la servlet

La première chose à faire est de déclarer notre servlet : en quelque sorte il s'agit de lui donner une carte d'identité, un moyen pour le serveur de la reconnaître. Pour ce faire, il faut ajouter une section au fichier qui se présente ainsi sous sa forme minimale:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
  <display-name>TP1</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>

  <servlet>
    <servlet-name>ServletAddPerson</servlet-name>
    <servlet-class>com.poly.info.controller.ServletAddPerson</servlet-class>
  </servlet>
```

La balise responsable de la définition d'une servlet se nomme logiquement `<servlet>`, et les deux balises obligatoires de cette section sont très explicites :

- `<servlet-name>` permet de donner un nom à une servlet. C'est ensuite via ce nom qu'on fera référence à la servlet en question. Ici, on nomme notre servlet **ServletAddPerson**.
- `<servlet-class>` sert à préciser le chemin de la classe de la servlet dans votre application. Ici, notre classe a bien pour nom **ServletAddPerson** et se situe bien dans le package **com.poly.info.controller**.

Mapping de la servlet

Il faut ensuite faire correspondre notre servlet fraîchement déclarée à une URL, afin qu'elle soit joignable par les clients :

```
<servlet-mapping>
  <servlet-name>ServletAddPerson</servlet-name>
  <url-pattern>/ServletAddPerson</url-pattern>
</servlet-mapping>
```

La balise responsable de la définition du mapping se nomme logiquement `<servlet-mapping>`, et les deux balises obligatoires de cette section sont, là encore, très explicites.

- `<servlet-name>` permet de préciser le nom de la servlet à laquelle faire référence. Cette information doit correspondre avec le nom défini dans la précédente déclaration de la servlet.
- `<url-pattern>` permet de préciser l'URL relative à travers laquelle la servlet sera accessible. Ici, ça sera **/ServletAddPerson.**

Mise en marche

Nous venons de créer un fichier de configuration pour notre application, nous devons donc redémarrer notre serveur pour que ces modifications soient prises en compte. Il suffit pour cela de cliquer sur le bouton "start" de l'onglet **Servers**.

Faisons le test, et observons ce que nous affiche notre navigateur lorsque nous tentons d'accéder à l'URL http://localhost:8080/TP1/ServletAddPerson que nous venons de mapper sur notre servlet.