# *Mise en oeuvre des Servlets et des JSP*

## 1 Le conteneur WEB Tomcat d'Apache (20m)

### 1.1 Charger, installer et tester

- Commencez par récupérer le conteneur WEB Tomcat [1] (ou ici [2]) (version **8.0.x** non RPM, package **core**) et installez-le (décompressez l'archive).
- Rendez exécutables les scripts se trouvant dans le répertoire `bin` avec

```
chmod u+x apache-tomcat-*/bin/*.sh
```

- Lancez le serveur avec le script `startup.sh` du répertoire `bin`.
- Testez ensuite les exemples présents en vous connectant à l'adresse `http://localhost:8080`. **Attention** : commencez par les exemples de servlets puis les exemples en JSP 1.2 et laissez les exemples JSP 2.0 pour plus tard. **N'oubliez pas**, à la fin du TP, de stopper le moteur de servlets (commande `shutdown.sh`).

### 1.2 Préparer le manager

**Dans un premier temps, vous pouvez sauter cette section**.

Le *manager* [3] est une application WEB permettant de contrôler les applications WEB déployées sur le moteur de servlets. Pour l'utiliser, vous devez créer un utilisateur (dans le moteur de Servlets) qui joue le rôle de *manager*.

Pour ce faire, modifiez la définition des utilisateurs (fichier `conf/tomcat-users.xml`) en ajoutant un rôle `manager` et faites en sorte que l'utilisateur `tomcat` ait ce rôle. Relancez le serveur Jakarta Tomcat (`shutdown.sh` puis `startup.sh`). Essayez de gérer les applications WEB avec l'application *manager* [4].

Faites de même pour le rôle `admin` et testez l'application *d'administration* [5] du serveur tomcat.

## 2 Eclipse JEE et les applications WEB (20m)

Nous allons utiliser le plugin `WTP` [6] qui est intégré par défaut dans Eclipse pour JEE. Pour ce faire, suivez ces étapes **les unes après les autres** :

1. Lancer la version JEE de Eclipse [7] (commande `eclipse-ee-neon`).
2. Vérifiez que le JRE par défaut est bien en version **1.8** ou supérieure.
3. Dans le menu *Windows/Preferences* choissisez l'onglet « *Server / Runtime Environments* » et ajoutez un nouveau **serveur local de type Apache Tomcat 8.0**.
4. Créez un nouveau projet de type « *Web / Dynamic Web project* » avec le nom `monapp`. Lors de la création prenez soin de lui associer le **Target Runtime** correspondant au serveur Apache Tomcat. A ce stade, votre projet doit ressembler à ceci :

1. http://tomcat.apache.org/
2. ref:ress-jsp
3. http://localhost:8080/manager/html/
4. http://localhost:8080/manager/html/
5. http://localhost:8080/admin/
6. http://www.eclipse.org/webtools/
7. http://www.eclipse.org/

```
monapp
  | Deployment Descriptor      version agréable de web.xml
  + Java Ressources: src       les sources de votre projets
  |   | Apache Tomcat...       les librairies de Tomcat
  |   | EAR Libraries          (non utilisé)
  |   | JRE System Library     les libraires de WEB-INF/lib/
  | JavaScript Support
  | build/                     zone de travail)
  + WebContent                 votre application WEB)
  |   | META-INF               (le manifest)
  |   + WEB-INF                configuration de votre app.
  |   |   | lib/               les librairies de votre app
  |   |   | web.xml            configuration de votre app
```

5. Créez dans le répertoire `WebContent` une page JSP et nommez-la `index.jsp`. Elle doit ressembler à ceci

```
<%@ page language="java"
    contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Title</title>
</head>
<body>
  <p>Hello.</p>
</body>
</html>
```

6. Sélectionnez votre projet et exécutez-le sur le serveur Tomcat (menu « *Run as... / Run on server* »). Faites en sorte d'associer définitivement votre projet et le serveur Tomcat (une petite case à cocher avant le lancement).

7. A ce stade, vous devez pouvoir accéder à votre application via l'URL ( `http://localhost:8080/monapp/` ). Vous pouvez maintenant ajouter des pages HTML, JSP et/ou des sources Java dans votre projet.

## 3 Mon premier bean (20m)

- Modifiez la page précédente pour qu'elle affiche la date du jour et l'heure à chaque exécution (créez pour cela une instance de la classe `java.util.Date` et affichez la).
- Faites tourner les exemples présentés en cours[8] d'affichage et de manipulation d'un produit.
- Faites varier la portée (le `scope`) du `<jsp:useBean>` et observez les différences dans les trois cas[9] ( `request` , `session` , `application` ).
  - ▷ **Conseil 1** : Modifiez le constructeur vide du bean produit et placez-y un affichage afin de suivre les créations de nouvelles instances. Vous pouvez également placer du code JSP à l'intérieur de l'action `<jsp:useBean>` . Ce code est exécuté lorsque le bean est créé.
  - ▷ **Conseil 2** : Pour bien manipuler les *beans* de portée `session` , utilisez plusieurs navigateurs (firefox ou chrome).

8. jsp.html#javabean
9. servlet.html#scope

▷ **Conseil 3** : Faites en sorte que votre bean implante l'interface `HttpSessionBindingListener` [10] et mettez en place des traces pour suivre la vie de ce bean. Dans un deuxième temps, diminuez la durée de vie des sessions (ajoutez le code ci-dessous dans le fichier `web.xml`) afin d'observer la suppression automatique des beans.

```
<session-config>
   <session-timeout>1</session-timeout><!-- une minute -->
</session-config>
```

# 4 Une petite application (2h00)

## 4.1 Étape 1 : présenter

- Créez un « bean » `Person` représentant une personne (numéro (identifiant), nom, prénom, date de naissance, adresse e-mail)
- Créez une page JSP `person.jsp` qui présente (dans un tableau HTML) le contenu d'une instance de `Person` accessible en session.
- Créez une servlet `edition` que ne réalise aucune action (pour l'instant).
- Créez une page JSP `edition.jsp` afin de produire un formulaire HTML d'édition des caractéristiques d'une personne placée en session. La soumission de ce formulaire va appeler la servlet `edition`.

## 4.2 Étape 2 : traiter

Modifiez la servlet `edition` afin qu'elle réalise les actions suivantes

- Créer une instance de `Person` ou la récupérer à partir de la session si elle existe déjà.
- Affecter cette instance avec les paramètres de la requête HTTP (nom, prénom, date de naissance, etc.).
- Appeler la page JSP `person.jsp` en utilisant le code ci-dessous :

```
// Appeler une page JSP depuis une servlet
request.getRequestDispatcher(pageJsp).forward(request, response);
```

## 4.3 Étape 3 : ajouter une couche métier fictive

Modifiez votre projet afin d'ajouter une classe métier orientée vers le traitement des personnes :

10. servlet.html#HttpSessionBindingListener

```
package fr.myapp.bus;

import java.util.Collection;
import java.util.Map;

public class PersonManager {

    final private Map<Integer, Person> persons;

    public PersonManager() {
        throw new IllegalStateException("Not yet implemented");
    }

    public Collection<Person> findAll() {
        throw new IllegalStateException("Not yet implemented");
    }

    public void save(Person p) throws Exception {
        throw new IllegalStateException("Not yet implemented");
    }

    public void check(Person p) throws Exception {
        throw new IllegalStateException("Not yet implemented");
    }

}
```

### 4.4 Étape 4 : valider

- Dans la Manager : Terminez la méthode `check` (le nom est obligatoire et l'email doit être valide).
- Dans la Servlet : Ajoutez une phase de validation des données. Faites en sorte, si les données ne sont pas valides, de revenir au formulaire en proposant les anciennes valeurs.
- Modifiez la page `edition.jsp` et la servlet de manière à faire apparaitre des messages d'erreur (en rouge) à coté des champs fautifs. Où pouvez-vous stocker ces messages ?

### 4.5 Étape 5 : enregistrer et lister

- Dans la Manager : Terminez la méthode `save` (enregistrez la personne dans la `Map`). Prévoyez d'initialiser cette `Map` avec deux ou trois personnes.
- Dans la Servlet : Si les données sont valides, enregistrez l'instance indexée par le numéro de la personne.
- Créez la page JSP `lister.jsp` qui va lister les personnes stockées et prévoir, pour chacune, un lien vers la servlet de la forme

```
<a href="edition?numero=12345">Nom d'une personne</a>
```

  La servlet va utiliser la méthode HTTP pour savoir si elle doit lancer l'édition d'une personne identifiée par un numéro (méthode `GET`) ou valider et enregistrer une personne (méthode `POST`).

### 4.6 Étape 6 : ajouter

Ajoutez à votre page `lister.jsp` le lien ci-dessous afin de pouvoir ajouter une nouvelle personne :

```
<a href="edition">Ajouter une personne</a>
```

**Important** : prévoyer de placer en session une information sur le type d'édition en cours : création **ou** modification. En cas de modification, vous pourriez stocker l'identifiant de l'objet en cours d'édition.

### 4.7 Étape 7 : supprimer

Modifiez votre page `lister.jsp` et votre servlet pour ajouter et traiter le cas de la suppression :

```
<a href="supprimer?numero=123456">Supprimer cette personne</a>
```

La servlet utilisera la méthode `request.getServletPath()` pour savoir sous quel nom elle a été appelée.