TP nº 7

Consignes les exercices ou questions marqués d'un ⋆ devront être d'abord rédigés sur papier (afin de se préparer aux épreuves écrites du partiel et de l'examen). En particulier, il est recommandé d'être dans les même conditions qu'en examen : pas de document autres que le cours ni de calculatrice. Tous les TPs se font sous Linux.

1 Préambule

Le but TP est le suivant :
- configurer Eclipse pour un intégrer un serveur Tomcat
- écrire une application Web simple, pour manipuler les concepts Web de bases en Java/JSP : sessions, formulaires, cookies, ...
- **il est interdit de renommer les classes et packages Java**
- **il ne faut pas importer le projet Eclipse du TP avant d'avoir correctement configuré Eclipse !**
- **il est important de bien configurer Eclipse pour le deuxième TP noté**

2 Configuration d'Eclipse

Effectuer dans l'ordre les opérations suivantes. Si une opération provoque une erreur, ne pas continuer mais essayer de la résoudre (en appelant éventuellement le chargé de TP).

1. Démarrer Eclipse
2. Menu `Help` → `Install New Software...`
3. **Décocher** la case `Contact all update sites during install to find required software`
4. Dans le menu `Work with:` choisir le site de mise à jour :

   `Neon - http://download.eclipse.org/releases/neon`

   Attendre que la liste des paquets apparaisse
5. Dans la liste des paquets, déplier la catégorie :

   `Web, XML, Java EE and OSGi Enterprise Development`

   puis cocher ensuite les paquets :

   ```
   Eclipse Java EE Developer Tools
   Eclipse Java Web Developer Tools
   Eclipse Web Developer Tools
   Eclipse XSL Developer Tools
   JSF Tools 3.9.0.v201605262035
   JSF Tools - Tag Library Metadata (Apache Trinidad)
   JST Server Adapters Extensions
   JST Server Adapters
   JST Server UI
   ```

   Si l'un de ces paquets n'apparaît pas, c'est qu'il est déjà installé sous Eclipse.
6. Cliquer sur `Next` puis encore sur `Next`
7. Cocher «`I accept the terms of the license agreement`»

8. Cliquer sur `Finish`
9. Les plugins Eclipse sélectionné s'installent. Toujours répondre « `Yes` » aux questiond d'Eclipse (faire confiance aux certificats, redémarrer Eclipse). Eclipse est relancé après installation.
10. Une fois Eclipse redémarré, aller dans le menu `File` → `New` → `Other`
11. Choisir `Server` → `Server`
12. Dans la liste, choisir `Apache/Tomcat v9.0 Server`. Attention, il se peut que la liste soit petite (bug dans Eclipse). Vous pouvez agrandir un peu la fenêtre pour faire apparaître tous les choix. Laisser les valeurs par défaut dans les champs.
13. Cliquer sur le bouton `Next >`
14. Dans le champ `Tomcat installation directory`, saisir `/public/kn/apache-tomcat-9.0.12`
15. Cliquer sur le bouton `Finish`

3 Prise en main

Télécharger sur le site du cours le projet `Eclipse` et **l'importer dans `Eclipse`**. L'architecture du projet est la suivante :

— Le répertoire `src` contient les sources Java du projet
— Le répertoire `WebContent` contient les ressources Web : fichiers HTML et JSP, classes du projet une fois compilées et fichiers `jar` externes que l'on souhaite rajouter.

Questions

1. Dans l'explorateur de projet, cliquer sur le répertoire `WebContent`, faire `New` et choisir `JSP File`. Appeler la page `index.jsp`.
2. Dans la section `<body>. . .</body>`, rajouter du texte fixe.
3. ⋆ Exécuter le fichier `index.jsp` (bouton `Run`). Choisir le serveur `Tomcat v9.0` et cocher `Always use this server when running this project` puis `Finish`. Que se passe-t-il ?

> **Réponse:** Eclipse démarre une instance du serveur Tomcat et affiche dans un onglet un navigateur Web pointant vers la page `index.jsp`

4. ⋆ Copier l'URL de la page et la copier dans le navigateur du bureau (`Google Chrome` ou `Firefox`). La page est-elle accessible ?

> **Réponse:** Oui. La page est accessible tant que le serveur `Eclipse` et le projet sont démarrés.

5. Modifier le corps de la page pour afficher l'heure à laquelle elle a été générée. On pourra utiliser la classe java `Date`. Attention, les suggestions de code d'Eclipse ne fonctionnent pas dans les pages JSP, il convient donc d'ajouter manuellement les imports de packages, en utilisant la syntaxe JSP vue en cours.
6. ⋆ On souhaîte rajouter à cette page un compteur de visites (i.e. afficher dans la page le nombre de fois où celle-ci a été visitée). De quels moyens dispose-t-on pour stocker de l'information persistante (hors fichier et bases de données). Quel est le moyen le plus approprié pour un compteur de visite ?

> **Réponse:** On dispose de trois moyen pour stocker de l'information. En premier lieu, les cookies. Ces derniers sont stockés coté client et donc inadaptés à un compteur de visite. En second lieu, les variables de sessions. Celles-ci sont stockées côté serveur et accessibles depuis l'objet implicite `session`. Cependant, il y a un objet session par client HTTP, donc ce n'est pas non plus un moyen approprié. Enfin, l'objet implicite `application` permet de stocker des objets pendant toute la durée de vie de l'application, *i.e.* jusqu'au redémarrage du serveur Web. Les objets sont stockés par application et donc visible depuis toutes les sessions. C'est un moyen approprié pour stocker notre compteur.

7. Implémenter le compteur de visites et le tester.

4 Base de données de Films, interface Web

Le projet `Eclipse` du TP contient les mêmes classes qu'au TP 5 : `Movie`, `IMovieDB` et `MovieDB` (les versions non `java.rmi`). Le but est d'écrire de petites pages Web permettant d'accéder aux films (requêtes et ajout).

1. Créer une page `all_movies.jsp` (sous le répertoire `WebContent`) permettant créer une instance de `MovieDB` et d'importer le fichier `movie.txt` se trouvant dans le répertoire `resources`. Une fois la base remplie, faites en sorte que la page affiche tous les films sous forme d'une liste non-énumérée (éléments `ul` et `li`).
2. ⋆ Que se passe-t'il à chaque fois qu'un utilisateur recharger la page ? Comment peut-on stocker l'objet `MovieDB` de manière à ce qu'il ne soit pas re-chargé à chaque visite d'un même client.

> **Réponse:** La base est rechargée à chaque visite de la page. On peut stocker l'objet `MovieDB` dans une *variable de session*.

3. Créer deux pages, `query_movie.jsp` et `display_movie.jsp`. La page `query_movie.jsp` contient un formulaire permettant de saisir une chaîne de caractères et un bouton de soumission. Le formulaire passe les paramètres en `POST` et a pour cible `display_movie.jsp`. Cette page affiche tous les films contenant la chaîne de caractères reçue dans le titre (en utilisant `.queryTitle()` sur un objet `IMovieDB`).
4. ⋆ On souhaite maintenant que le champ de texte du formulaire soit pré-rempli avec la dernière chaîne de caractères saisie, même si la session HTTP a expiré. Proposer un moyen d'accomplir cela.

> **Réponse:** Il suffit de metter dans la page `display_movie.jsp` un cookie contenant la valeur de la chaîne de caractères passée. Dans la page `query_movie.jsp`, on utilise la valeur de ce cookie, s'il est présent pour pré-remplir le champ de saisie (en mettant la valeur du cookie dans l'attribut `value` de l'élément `input`).

5. Rajouter autant de formulaire que de crièteres de recherche dans `MovieDB`
6. Rajouter un formulaire de recherche de film par ID
7. Rajouter un formulaire de suppression de film par ID
8. Rajouter un formulaire d'ajout de film