http://www.labo-sun.com
labo-sun@supinfo.com

LABORATOIRE SUPINFO DES TECHNOLOGIES Sun

Servlets et JavaServerPages

TRAVAUX PRATIQUES

Auteur : Cyril Joui & Hélène Semere
Version n° 2.0 – 3 novembre 2004
Nombre de pages : 24

Paris Academy of Computer Science SUPINFO Paris Ecole Supérieure d'Informatique

Ecole Supérieure d'Informatique de Paris
23. rue Château Landon 75010 – PARIS
www.supinfo.com

Table des matières

1. Servlets

1.1. Première Servlet, afficher « Hello World »

1.1.1. Objectif

Le but de cet exercice est de vous initier à la conception de Servlets. Vous allez construire la Servlet la plus simple possible, le classique « Hello World ».

1.1.2. Sujet de l'exercice

Tout d'abord, il vous faudra configurer votre moteur de servlet pour pouvoir lancer la première Servlet.
Puis, vous allez créer et configurer un projet dans lequel vous insèrerez toutes vos Servlets.
Vous devrez, ensuite, implémenter l'interface **javax.servlet.Servlet** et par conséquent, les différentes méthodes requises pour que votre Servlet puisse fonctionner. Toutes ces méthodes vous ont été présentées dans le cours.
Vous vérifierez, enfin, que la Servlet s'affiche bien lorsqu'on l'ouvre dans un navigateur Web.

1.1.3. Mode opératoire

1.1.3.1. Création d'un projet permettant de réaliser des Servlets

Dans Eclipse, vous allez créer un nouveau projet Java :
Entrez dans le menu *File/New/Project…* comme sur la Figure 1 :

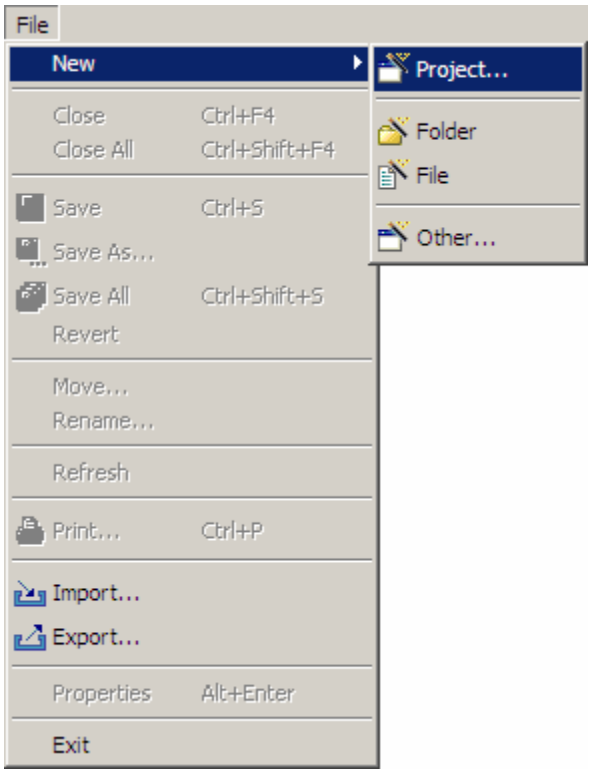


Figure 1 - Nouveau projet Java

Sélectionnez *Dynamic Web Project* dans la catégorie *Web* et cliquez sur *Next* :

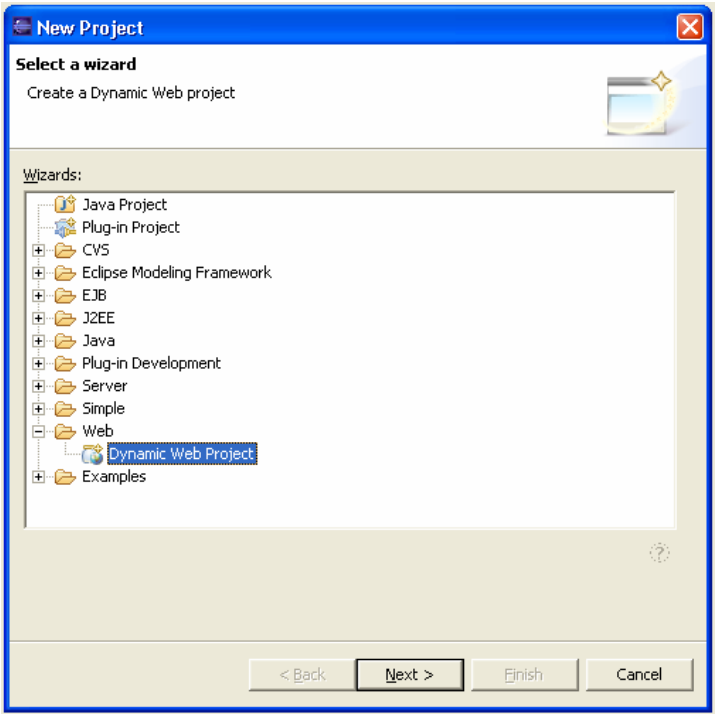


Figure 2 - Nouveau projet Java (bis)

Vous choisirez ensuite le nom du projet (*« TDServle1-HelloWorld »*) et l'emplacement que vous voulez. Décocher la case « Add module to an EAR project » (Voir Figure 3) :

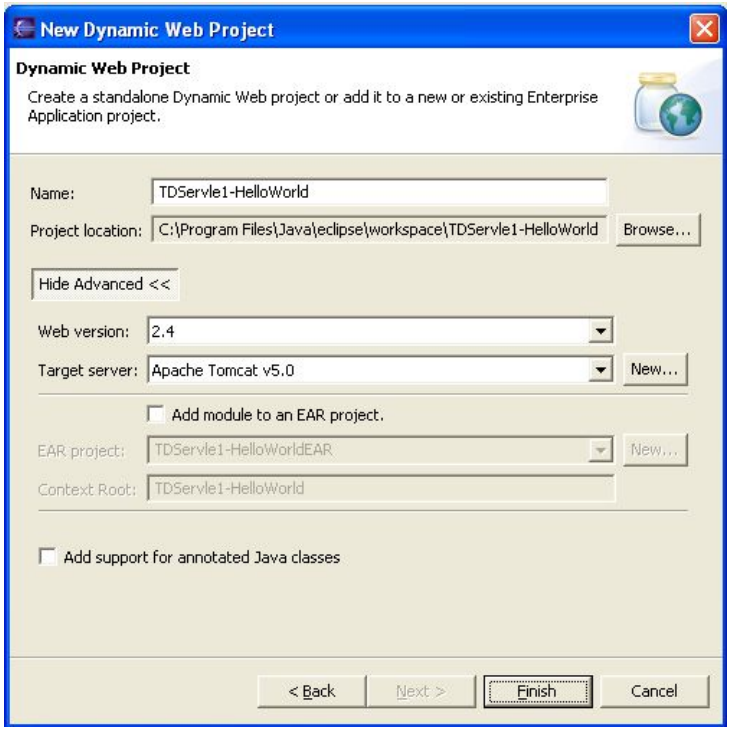
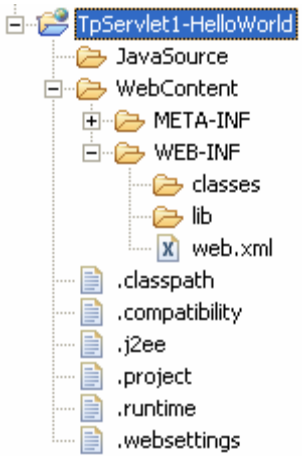


Figure 3 - Nom et chemin du projet

Une fois que vous avez cliqué sur « Finish » un projet est créé avec un ensemble de dossiers et fichiers par défaut :



Description des dossiers :

- **JavaSource** : représente le package par défaut pour vos servlets. Il n'est pas recommandé de créer vos servlets directement à la racine mais de les organiser en package.
- **WebContent/WEB-INF** : dossier de configuration de votre application web.
- **WebContent/WEB-INF/web.xml** : fichier de configuration de votre application web (nous allons le voir en détail par la suite)
- **WebContent/WEB-INF/classes** : dossier qui contient l'ensemble des classes compilées.
- **WebContent/WEB-INF/lib** : dossier qui contient l'ensemble des librairies externes pour votre projet (il est ajouté automatiquement dans votre classpath).

Les différents fichiers (commençant par un '.') sont des fichiers de configuration.
Toutes les bibliothèques tel que « servlet.jar » sont automatiquement liées à votre projet (grâce au plugin), il n'est donc pas nécessaire des les ajouter.

1.1.3.2. Conception de la Servlet qui dit « Coucou »

Nous allons, pour travailler, ouvrir une perspective *Java*. Pour cela, allez dans le menu et choisissez *Window/Open Perspective/Java* :

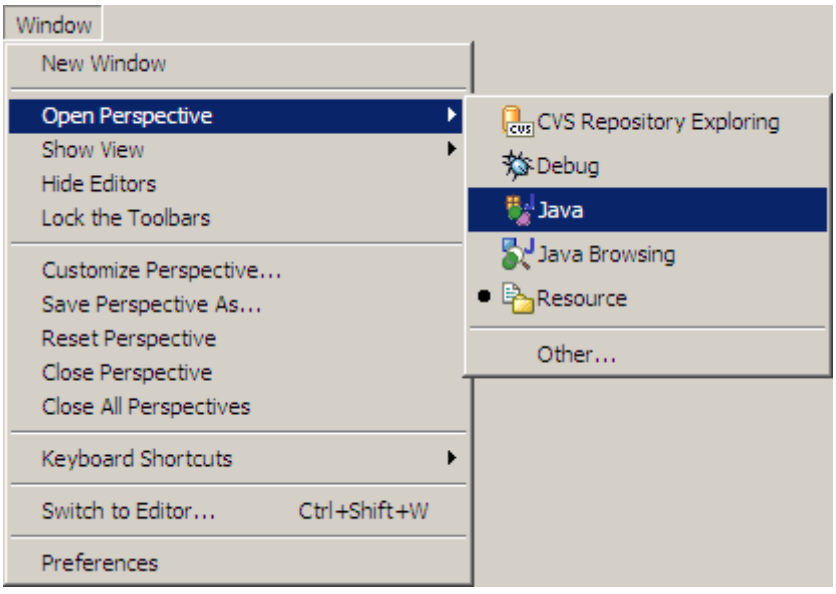


Figure 4 - Ouverture d'une perspective Java

Nous allons maintenant pouvoir nous concentrer sur l'implémentation de la première Servlet. Cliquez avec le bouton droit de la souris sur le projet créé et sélectionnez *New/Class… :*

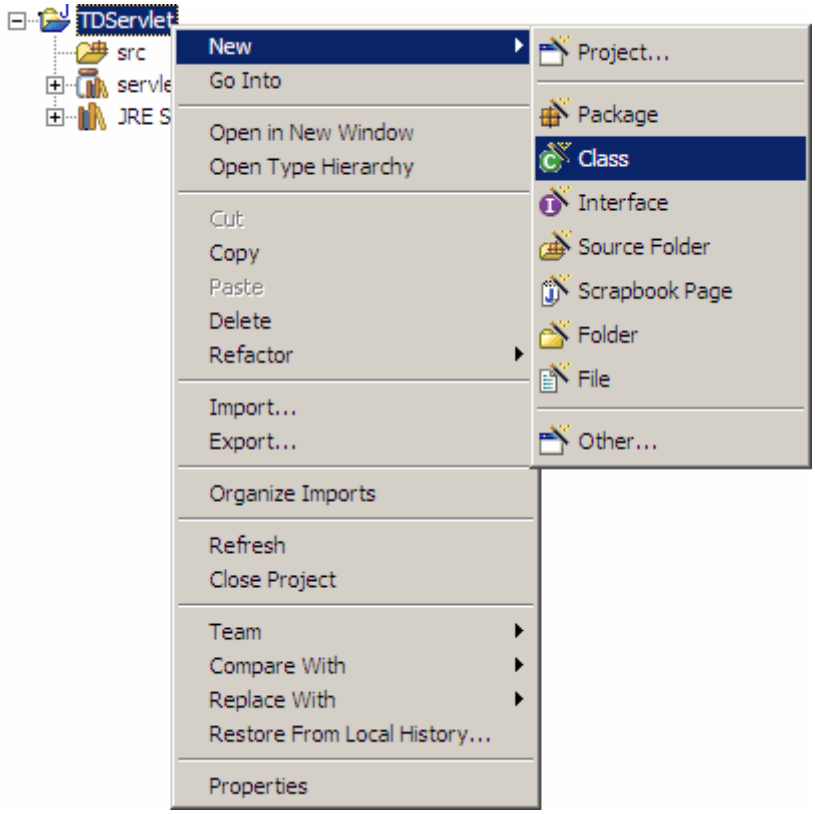


Figure 5 - Lancement de l'assistant de conception d'une nouvelle classe

Vous allez voir apparaître l'assistant de conception d'une classe. Vous pouvez commencer par écrire le nom de la Servlet à créer dans le champ *Name* (ici *« HelloWorld »*). Voyez sur la Figure 12 la partie concernée de l'assistant :

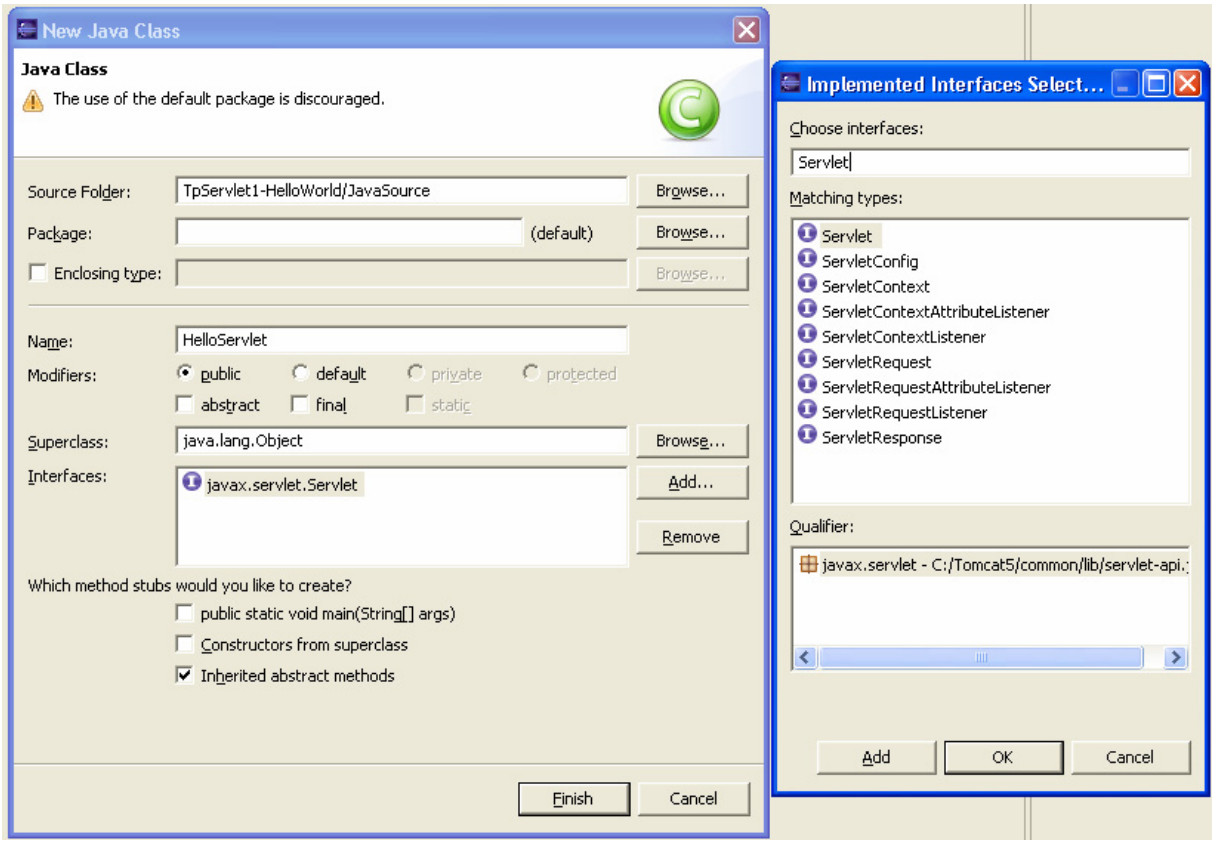


Figure 6 - Assistant de conception de classe et d'interface

Ensuite, il faut préciser que l'on souhaite implémenter l'interface *javax.servlet.Servlet* Cliquez, pour cela, sur le bouton *Add…* en face de *Interfaces* (Voir Figure 12). Sur la fenêtre qui apparaît (Voir Figure 12), entrez le nom de l'interface : *« Servlet »*. Sélectionnez, dans la liste, en dessous, *Servlet*, si ce n'est pas déjà fait. Cliquez, maintenant, sur le bouton *OK* pour valider votre choix.

Pour terminer la conception de la classe *HelloWorld*, il ne vous reste plus qu'à vérifier que le choix des méthodes à générer automatiquement soit correct (Voir Figure 13). La seule case à cocher est la dernière, soit celle qui précise que l'on veut créer les méthodes à implémenter de l'interface choisie (*init*, *destroy*, *service*, *getServletConfig* et *getServletInfo*).

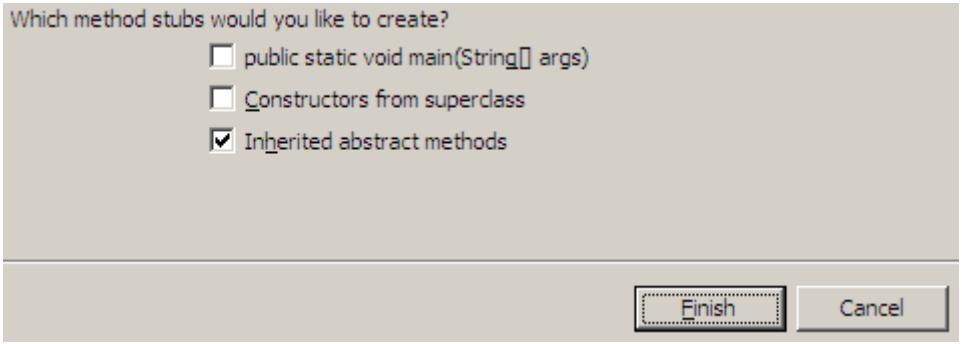


Figure 7 - Génération automatique de méthodes

Enfin, cliquez sur le bouton *Finish* et votre classe *HelloWorld* va être générée avec les méthodes à implémenter.

Maintenant, en vous appuyant sur ce qui a été vu dans le cours, vous pouvez ajouter le contenu des méthodes suivantes :

- init()
- destroy()
- service()
- getServletConfig()
- getServletInfo()

Voici le code que vous pouvez introduire dans la méthode *service* :

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html><body>");
out.println("Hello World");
out.println("</body></html>");
out.close();
```

Si ce code vous parait compliqué, ce n'est pas important car vous aurez les explications dans les chapitres suivants du cours.

1.1.3.3. Configuration du mapping de la servlet

Nous allons maintenant configurer le contexte avec laquelle vous pourrez accéder à votre servlet. Sur certains serveur vous pouvez accéder directement à votre servlet en ajoutant à votre contexte principale : **/servlet/package/NomClasseServlet**.
Nous allons personnaliser le « mapping » afin d'unifier notre développement. Nous allons mapper notre servlet au context : **/Hello**.

Pour cela éditez le fichier **web.xml** et ajoutez les lignes suivantes :

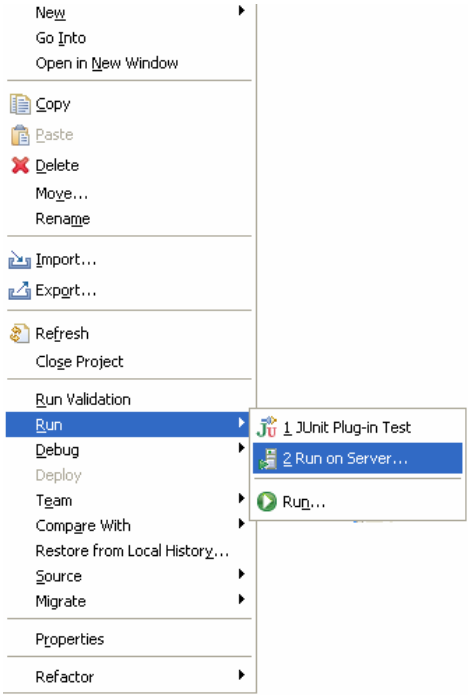
```
<!-- Définition des servlets présentes dans l'application -->
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>HelloServlet</servlet-class>
</servlet>

<!-- Mappage de noms pour les servlets -->
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/Hello</url-pattern>
</servlet-mapping>
```

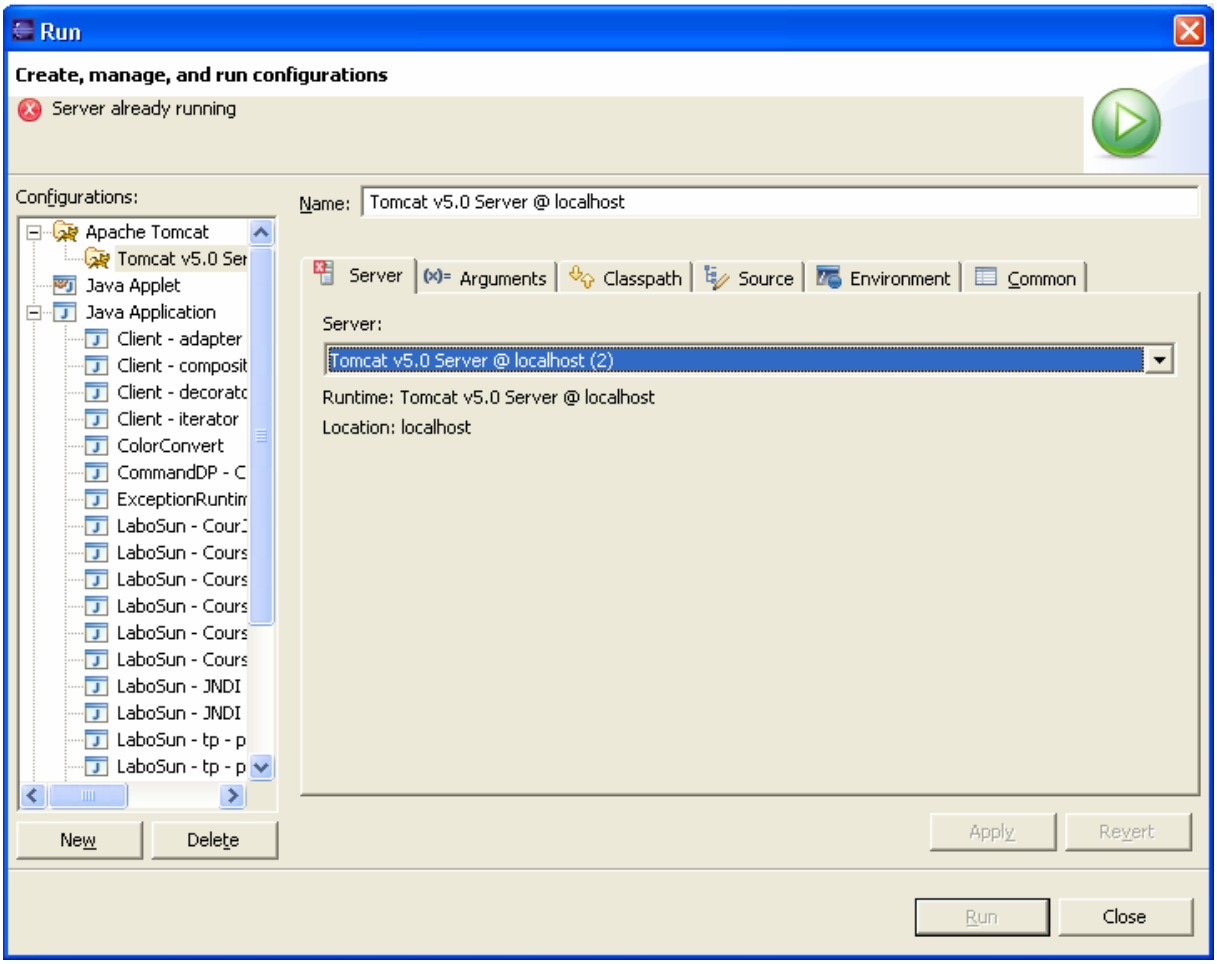
Pour plus d'explications concernant ces lignes veuillez vous reporter au cours.

1.1.3.4. Test de la Servlet

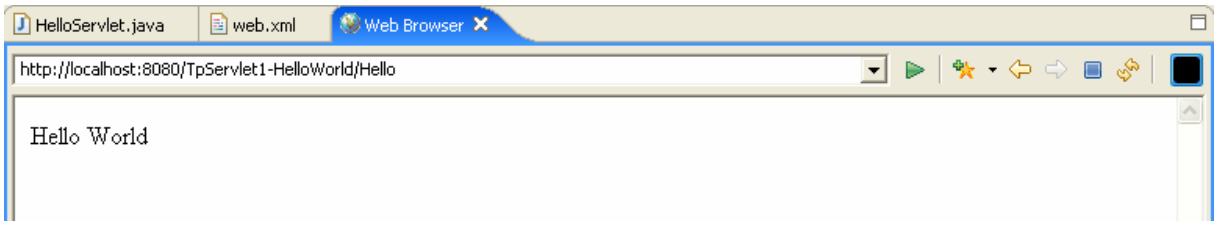
Maintenant que vous venez de créer et configurer votre premier servlet, il faut pouvoir y accéder. Pour cela, faites un clic droit sur votre projet **> Run > Run on server …**



La fenêtre suivante de choix du serveur apparaît :



Il vous suffit de sélectionner le serveur sur lequel vous voulez lancer votre application web et cliquer sur Run. Cela va ouvrir un Browser avec une adresse du type : http://localhost:8080/TpServlet1-HelloWorld/. Pour accéder à votre servlet il vous suffit d'aller à : (en se basant sur notre adresse) http://localhost:8080/TpServlet1-HelloWorld/Hello. En effet, nous avons mappé notre servlet sur le contexte /Hello.



1.2. Récupérer les valeurs d'un formulaire

1.2.1. Objectif

L'objectif de cet exercice est de vous apprendre à utiliser les formulaires avec les servlets. Les formulaires sont des éléments graphiques HTML, afin de bien séparer le code et le design. Le mieux est donc de faire le traitement de ces formulaires dans une servlet. C'est ce que nous allons voir ici.

1.2.2. Sujet de l'exercice

Créer un formulaire dans une page html et une servlet RecupFormServlet qui permettra de récupérer les éléments du formulaire et de les afficher.

1.2.3. Mode opératoire

Créer tout d'abord une page index.html dans laquelle vous ajouter un formulaire HTML contenant un champ *nom, prenom* et une liste de valeur à choix multiple : *multiValue*.
Créer votre servlet de récupération et mapper là sur le contexte : **/RecupFormServlet** (bien entendu faite pointer votre formulaire HTML vers ce contexte).
Votre servlet doit écrire les valeurs du formulaire via l'objet PrintWriter (vu dans l'exercice précédent). Pour récupérer les valeurs du formulaire dans votre servlet utiliser la méthode : ***getParameter(String name)*** et ***getParameterValues(String name)*** pour les éléments multivalués.

1.3. Gestion des cookies

1.3.1. Objectif

Le but de cet exercice est d'apprendre à se servir des cookies dans une application web. Vous devrez être capable à la fin de cet exercice de gérer les cookies (ajout / modif / suppression).

1.3.2. Sujet de l'exercice

Créer une page et une servlet qui vous permettrons d'ajouter / modifier / supprimer des cookies. Votre page devra contenir un formulaire dans lequel on pourra rentrer le nom et la valeur de la propriété à ajouter au cookie.

1.3.3. Mode opératoire

Pour ajouter un cookie, utilisez la méthode : ***addCookie()*** de l'objet **HttpServletResponse.**
Pour récupérer les objets Cookie déjà existant, utilisez ***getCookies()*** de l'objet **HttpServletRequest**.

1.4. Gestion des sessions

1.4.1. Objectif

Le but de cet exercice est d'apprendre à utiliser les sessions. Vous devrez être capable de gérer une session utilisateur, y ajouter des données, les modifier et les supprimer.

1.4.2. Sujet de l'exercice

De même qu'au TP précédent, créer une page et une servlet qui vous permettra de gérer (ajouter/modifier/supprimer) les valeurs de la session.

1.4.3. Mode opératoire

La gestion des sessions est assez similaire à la gestion des cookies. Cependant vous travailler sur un objet **HttpSession** que vous obtenez via la méthode : ***getSession()*** de l'objet **HttpRequest**.
Ensuite vous pouvez ajouter / modifier un attribut de la session via ***setAttribute()*** soit supprimer un attribut avec ***removeAttribute().***

1.5. Partage de données entre servlets

1.5.1. Objectif

L'objectif de cet exercice est d'utiliser le ***ServletContext*** afin de savoir partager des informations entre les servlets.

1.5.2. Sujet de l'exercice

Nous souhaitons partager des informations entre servlets. Il vous faut donc créer un ensemble qui permette d'ajouter / modifier / supprimer des valeurs au contexte mais également d'afficher les valeurs du contexte dans deux servlets différentes (afin de vérifier que les données sont partagées entre servlets).

1.5.3. Mode opératoire

Créer une servlet : **Servlet1** et une autre **Servlet2** qui auront pour but d'afficher les valeurs du contexte. Une autre servlet **ContextGereServlet** permettra d'ajouter / modifier / supprimer des valeurs au contexte.

L'objet **ServletContext** peut être récupéré via la méthode : ***getServletContext()*** de votre servlet.
Ensuite utilisez ***setAttribute*** et ***removeAttribute*** pour ajouter / modifier et supprimer les valeurs dans votre contexte.

La récupération des valeurs du contexte se fait depuis la méthode ***getAttributeNames()*** qui retourne un objet **Enumeration** qui contient l'ensemble des noms des attributs du contexte. Pour récupérer la valeur d'un attribut particulier, utilisez : ***getAttribute(String name)*** où name est le nom de l'attribut.

1.6. Afficher le nombre de sollicitations d'une Servlet

1.6.1. Objectif

Le but de cet exercice est de vous apprendre à utiliser les classes *GenericServlet* et *HttpServlet*. Dans les prochains TD, vous utiliserez *HttpServlet* comme base pour créer chaque Servlet.

1.6.2. Sujet de l'exercice

Vous allez réaliser une Servlet qui affiche le nombre de requêtes effectuées sur celle-ci et une deuxième séparant les requêtes effectuées par la méthode GET des requêtes effectuées par la méthode POST.

1.6.3. Mode opératoire

1.6.3.1. Calcul du nombre de requêtes

Vous devez, tout simplement, implémenter une Servlet *CounterServlet* qui dérive de la classe *GenericServlet*. Il vous suffit pour cela, de définir la méthode ***service()*** de l'interface *javax.servlet.Servlet*, comme il a été présenté dans le cours.
Vous devez ensuite ajouter un système permettant d'afficher à chaque requête le nombre total de requêtes effectuées depuis la dernière instanciation de la Servlet. (Ex. : au 4^{e} appel de la Servlet en question, l'affichage pourrait être « la Servlet a été appelée 4 fois »).

1.6.3.2. Synchronisation

Vous devez maintenant ajouter une notion de synchronisation. Pour cela il faut se rappeler que la même Servlet peut être appelée par deux threads ou plus au même instant. Il faut éviter qu'il y ait des conflits entre les valeurs stockées au niveau de la classe et leur affichage.
Il faut donc s'assurer que, lorsqu'on modifie une variable de la classe, celle-ci n'est utilisée nulle part ailleurs. On peut empêcher qu'une méthode soit exécutée par plusieurs threads à la fois, grâce au mot-clé ***synchronised***, utilisé comme dans cet exemple :

```
public synchronized void think()
{
        ...
}
```

Trouver une autre manière de gérer cette synchronisation des appels.

1.6.3.3. Séparation des méthodes GET et POST

Tout en conservant cette notion de synchronisation, vous devez séparer les requêtes en GET des requêtes en POST et ce en remplaçant la classe mère *GenericServlet* par la classe *HttpServlet*.
Concrètement, vous devrez définir les méthodes ***doGet()*** et ***doPost()*** de la classe *HttpServlet*. Ces deux méthodes doivent se comporter comme la méthode ***service()***. C'est-à-dire qu'elles doivent afficher le nombre de requêtes faites sur la Servlet depuis la dernière instanciation.
Il y aura, par contre, un petit changement à effectuer. La méthode ***doGet()*** devra incrémenter le nombre de requêtes faite avec la méthode GET et la méthode ***doPost()*** devra incrémenter le nombre de requêtes faite avec la méthode POST.

Dans les deux cas, il faudra afficher sur la page résultante, le nombre de requêtes en GET et le nombre de requêtes en POST.

1.7. Différents types de contenus

1.7.1. Objectif

Le but de cet exercice est de vous apprendre à générer des contenus de types différents en sortie d'une Servlet. Vous pourrez alors afficher une page HTML, une page XML, une image, …

1.7.2. Sujet de l'exercice

Vous devrez réaliser une Servlet à partir de *HttpServlet,* qui répond à une requête par une image. Vous appellerez ensuite cette Servlet à partir d'une balise image <IMG>.

1.7.3. Mode opératoire

1.7.3.1. Génération de l'image par une Servlet

Dans la fonction ***doGet()*** d'une Servlet *ImageServlet,* vous devez, tout d'abord ouvrir le fichier image grâce à un objet de type ***FileInputStream*** comme suit :

```
FileInputStream fis = new FileInputStream("java.gif");
```

La différence avec cet exemple est que vous aurez besoin de connaître le chemin relatif de l'image par rapport à la Servlet en cours. Pour cela vous avez à disposition la fonction ***getServletContext()*** qui retourne un objet de type *ServletContext* qui lui contient une méthode ***getRealPath()*** qui retourne le chemin local d'une ressource à partir d'une adresse relative. Notre exemple devient alors :

```
String adresseAbsolueImage = getServletContext().getRealPath("java.gif");
FileInputStream fis = new FileInputStream(adresseAbsolueImage);
```

Dans cet exemple, l'image "java.gif" se trouve à la racine du serveur Web.

Il faut, maintenant, définir le type de contenu grâce à la méthode ***setContentType()*** de la réponse. Voici le type de contenu d'une image Gif :

```
response.setContentType("image/gif");
```

Ensuite, vous devrez lire les données de l'image pour les envoyer sur la réponse. La méthode ***getOutputStream()*** nous fourni un flot de sortie, de type *ServletOutputStream*, dans lequel on peut écrire ce qu'on lit de l'image au fur et à mesure.

```
ServletOutputStream os = res.getOutputStream();
```

Ecrivez le code qui copie les données du flot d'entrée vers le flot de sortie.

1.7.3.2. Récupération de l'image

Pour récupérer le résultat de votre servlet il vous suffit de créer un fichier html (index.html par exemple) et d'insérer le code suivant (suivant comment vous accéder à votre servlet) :

```
<img src="/ContextPrincipale/servlet/ImageServlet">
```

2. Java Server Page

2.1.TD1 : Première page JSP

2.1.1. Objectif

Le but de cet exercice est de vous initier à la conception d'une page Web. Vous allez construire la page la plus simple possible, le classique « Hello World ».

2.1.2. Sujet de l'exercice

Vous allez créer un projet Web dynamique dans lequel vous allez créer une page JSP.
Vous afficherez ensuite le message Hello World sur cette page en utilisant des scriplets(<%%>).

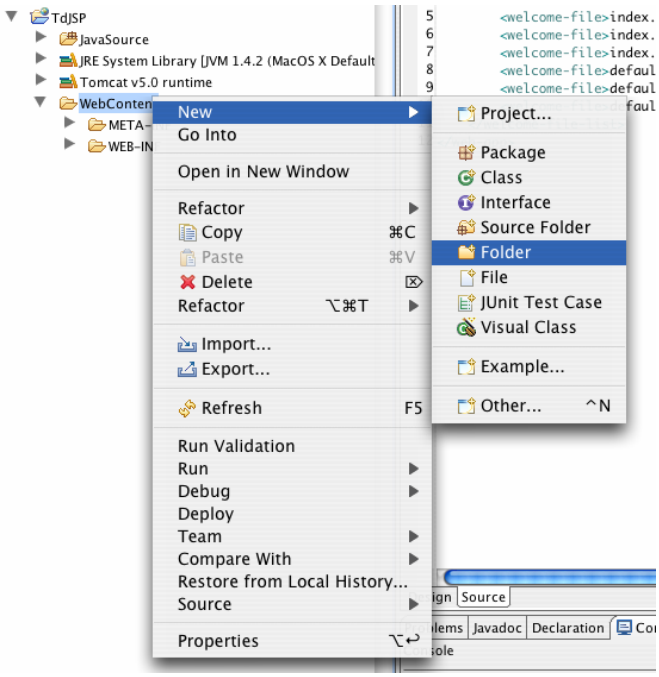
2.1.3. Mode opératoire

2.1.3.1. Création du projet Web dynamique

(Voir le chapitre 1.1.3.1)

2.1.3.2. Création de la page JSP

Vous devez créer votre page dans le dossier WebContent. Pour une meilleure organisation, vous allez donc créer un sous-dossier TD1 dans ce dossier :
Pour cela, cliquez droit sur le dossier WebContent, puis New et Folder.



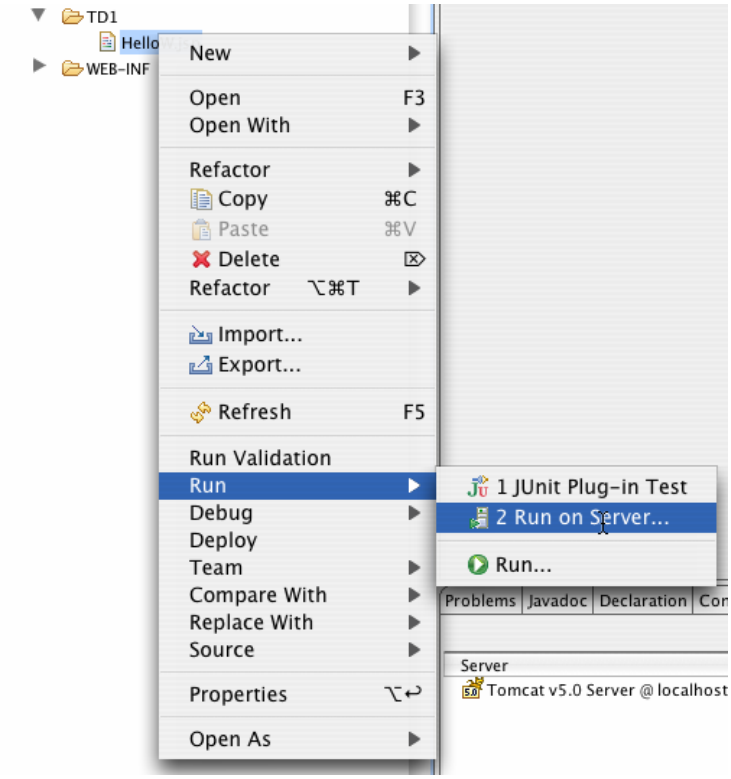
Vous spécifiez ensuite le nom du dossier, TD1.

Pour créer la page JSP, vous allez cliquer droit sur votre dossier TD1 et créer un nouveau fichier HelloW.jsp, comme suit et donner le nom de votre fichier, sans oublier de préciser l'extension jsp :

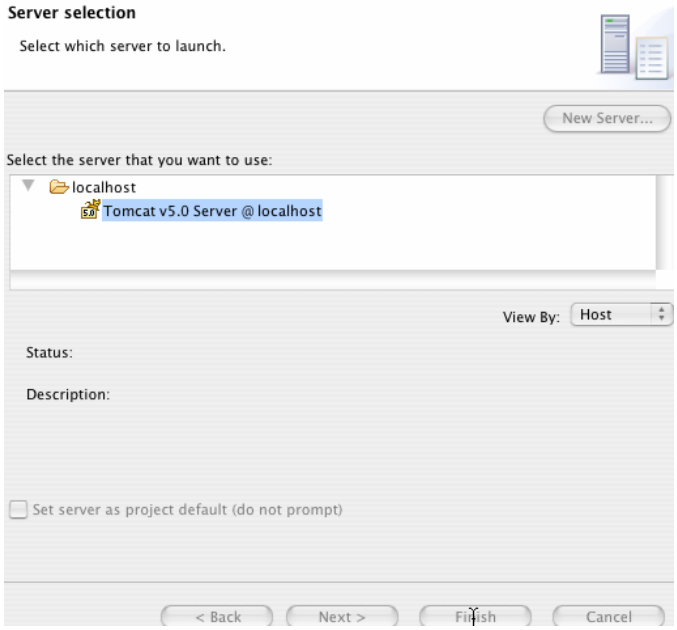
Vous avez donc créé votre première page JSP, vous pouvez maintenant écrire votre code afin d'afficher le message « Hello World ! ».

2.1.3.3. Lancer la première page JSP

Une fois votre code écrit dans la page JSP, vous allez cliquer droit sur votre page HelloW.jsp et sélectionner Run ->Run on Server…



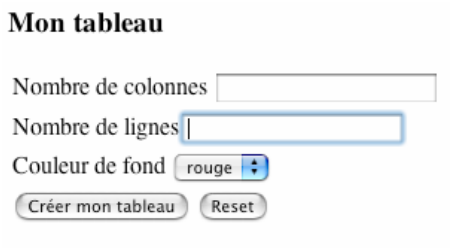
Puis, vous allez sélectionner le serveur sur le lequel vous souhaitez lancer la page :



Vous verrez alors un browser s'ouvrir avec le résultat de votre page JSP :

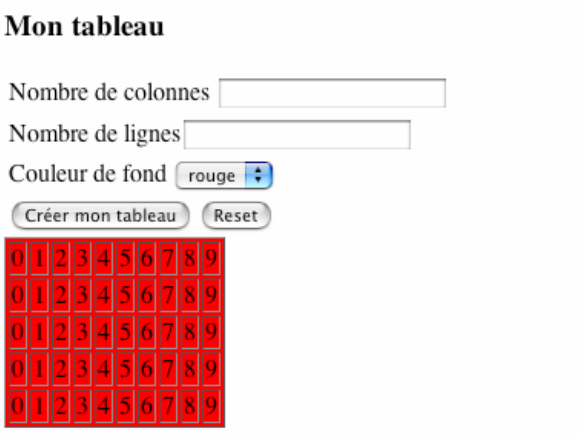
2.2.TD2 : Tableau personnalisé

Créer un formulaire permettant d'entrer un nombre de lignes et un nombre de colonnes et de sélectionner une couleur (voir le modèle qui suit) :



Le bouton 'Créer mon tableau' permet de créer un tableau avec les caractéristiques spécifiées et le bouton 'Reset' permet de mettre tous les champs à vide.
(On spécifiera des valeurs par défaut grâce aux balises de déclaration)

Résultat :

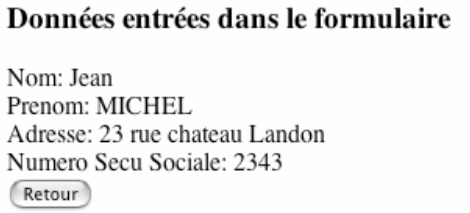


2.3.TD3 : Création d'un formulaire

Créer un formulaire simple permettant d'entrer ses nom(String), prénom(String), adresse(String) et numéro de sécurité sociale(int).

Créer un composant JavaBean, ClientBean qui aura pour paramètres ceux cités précédemment et qui prendra pour valeurs, celles entrées dans le formulaire.

Afficher ensuite ces données sur votre page en les récupérant de votre instance de JavaBean.



Votre classe ClientBean devra permettre d'initialiser l'objet avec des valeurs par défaut pour chacun des champs.
Le bouton 'Retour' permettra, lui, de rediriger la page vers l'enregistrement.

Ce Td devra être réalisé sur une même et seule page.

2.4.TD4 : Ma première balise personnalisée

2.4.1. Simple balise

2.4.1.1. Objectif

Le but de ce td est de créer sa propre balise personnalisée.

2.4.1.2. Sujet de l'exercice

Elle ne fera pas de traitement de corps ; c'est-à-dire qu'elle ne prendra pas de contenu entre ses balises et donc n'aura aucune modification à faire sur aucun contenu.
Cependant elle devra prendre des attributs :

- un message (obligatoire)
- une couleur (facultatif)

2.4.1.3. Mode opératoire

2.4.1.3.1. Création du tag handler

Comme il l'a été expliqué dans le cours, un tag handler est une classe JAVA devant implémenter directement ou indirectement l'interface **Tag**.
Vous allez donc dans un premier temps créer un package mesTags dans lequel vous créerez une classe ColorTag. Ce package sera créé dans le dossier JavaSource qui va contenir toutes vos classes JAVA.
Ne faisant pas de traitement de corps, votre balise étendra la classe TagSupport et redéfinira la méthode ***doEndTag()***.

2.4.1.3.2. Création du fichier tld

Une fois votre tag handler créé, il faut le définir dans un fichier XML que l'on nommera MonTld.tld. Ce fichier doit être placé dans le répertoire WebContent/WEB-INF/, et pour une meilleure organisation nous le placerons dans un dossier tlds qui regroupera tous les fichiers .tld de notre projet.
Quand on parle de définir un tag, il s'agit en fait de définir le nom de la balise qui sera reconnu par la page JSP, les attributs que cette balise devra ou pourra prendre en paramètres et les caractéristiques de ces attributs.
C'est donc dans cette page .tld que vous allez définir que votre balise

- doit prendre obligatoirement un message et optionnellement une couleur.
- ne doit prendre aucun contenu

2.4.1.3.3. Configuration du descripteur de déploiement

Pour configurer le descripteur de déploiement, vous allez ouvrir le fichier web.xml qui se trouve à la racine du répertoire WEB-INF et ajouter une balise taglib où vous préciserez la localisation de votre fichier .tld (entre des balises <taglib-location>/WEB-INF/tlds/MonTld.tld</taglib-location>) et le chemin que l'on spécifiera dans la page JSP (entre des balises <taglib-uri>/SupinfoTags</taglib-uri>).

Cette opération n'est pas obligatoire, en effet, on pourrait directement spécifier le chemin /WEB-INF/tlds/MonTld.tld dans la page JSP. L'intérêt est que dans le cas ou on utiliserait cette même librairie pour d'autres pages JSP, si on souhaitait modifier le nom de notre librairie de balise ou bien son emplacement, il suffirait de modifier le descripteur de déploiement et on éviterait d'avoir à retoucher toutes les pages JSP concernées.

2.4.1.3.4. Création de la page JSP

Pour la création de votre page JSP, vous allez procéder comme dans le TD1.
Vous vous appuierez ensuite sur le cours pour utiliser la balise que vous aurez créé.

2.4.2. Mon TagTableau

2.4.2.1. Objectif

Le but de ce TD est de continuer à se familiariser avec la création de customTag.

2.4.2.2. Sujet de l'exercice

Dans ce td, il s'agit de créer une balise avec traitement de corps.
Cette balise devra en effet permettre de créer un tableau avec un nombre de colonnes, de lignes et un fond de couleurs spécifiés en paramètre.
Par ailleurs, elle prendra entre ses balises un contenu qu'elle affichera dans chaque case du tableau.

2.4.2.3. Mode opératoire

Pour ce td, le tag handler devra étendre la classe BodyTagSupport afin de pouvoir traiter le corps du tag. Vous devrez alors redéfinir la méthode ***doAfterBody()*** puisque c'est dans cette méthode que vous allez pouvoir récupérer et modifier un contenu (à l'aide de la méthode ***getBodyContent()***).
Cette méthode ***doAfterBody()*** devra retourner EVAL_BUFFER_BUFFERED si le corps doit être réévalué(le container rappelle la méthode ***doAfterBody()***, SKIP_BODY dans le cas contraire. Dans ce dernier cas, le container de jsp appellera automatiquement la méthode ***doEndTag()***.

2.5.TD5 : Les librairies JSTL

2.5.1. Première utilisation d'une librairie JSTL

2.5.1.1. Objectif

Le but de ce td est de vous initier à l'importation et l'utilisation de librairies personnalisées.

2.5.1.2. Sujet de l'exercice

Pour ce td, vous allez reprendre l'énoncé du Td3 sur la création de formulaire en utilisant cette fois-ci les balises de la librairie Core qui fait partie des JSTL.

2.5.1.3. Mode opératoire

Vous reprendrez les explications du cours pour récupérer les fichiers nécessaires à l'utilisation des balises de la librairie Core.

Il faudra en effet, cette fois-ci afficher les données entrées dans le formulaire à l'aide des balises **out**.

3. Interaction Servlets ⇔ JSPs

3.1. Interaction entre JSPs

3.1.1. Objectif

Cet exercice à pour objectif de vous présenter comment utiliser la redirection entre 2 jsps.

3.1.2. Sujet de l'exercice

Créer deux pages jsps dont une appelera l'autre via une redirection si le paramètre forward vaut 1.

3.1.3. Mode opératoire

Créer une page index.jsp et une page forward.jsp. Dans la page index.jsp vérifiez si la valeur du paramètre est égal à 1 et faite un redirection.
Pour cela vous utiliserez **<jsp:forward page="/pageForward.jsp" />**.

3.2. Interaction entre Servlet et JSP

3.2.1. Objectif

Cet exercice à pour objectif de vous présenter comment utiliser la redirection entre une servlet et une jsp.

3.2.2. Sujet de l'exercice

Vous devez utilisez une servlet qui permettra de rediriger le visiteur sur les pages demandées via un paramètre. Pour l'exercice nous testerons cela avec 3 pages (accueil, page1, page2) et une servlet.

3.2.3. Mode opératoire

Créer tout d'abord la servlet qui permettra de rediriger la demande. Celle-ci devra recevoir un paramètre (dans l'URL) qui se nommera Action (format entier).
Vous utilisez l'objet **RequestDispatcher** que vous pouvez récupérer via la méthode :
getServletContext().getRequestDispatcher(String url) pour effectuer votre redirection.
Rediriger vers la page demandé via : ***forward()*** puis via ***include().*** Quelle est la différence ? (si vous ne voyez pas tenter de rajouter cette ligne à la fin de votre méthode ***doGet*** de votre servlet :

```
res.getWriter().print("<br>Fin servlet<br>");
// res étant l'objet HttpResponse de votre servlet
```

3.3. Interaction entre Servlets et transmission de données

3.3.1. Objectif

Dans cet exercice vous allez tester non seulement la redirection entre servlets mais également le passage de paramètre pendant cette redirection.

3.3.2. Sujet de l'exercice

Vous allez créer deux servlets : une principale et une autre secondaire. La principale appellera la secondaire si elle a reçu le paramètre secondaire (en GET). Celle si transmettra alors une chaîne de caractère à la deuxième qui l'affichera et rendra la main à la principale.

3.3.3. Mode opératoire

Créer une servlet **MainRedirectServlet** et une **SecondRedirectServlet**. Dans la première faite une redirection vers la seconde si le paramètre « secondaire » a la valeur 1 (en GET).
Passer une chaîne de caractère à la seconde servlet (il suffit de d'utiliser la syntaxe : SecondServlet ?param1=valeur1&param2=valeur2).

3.4. Gestion de commandes de jeux, noté

3.4.1. Objectif

Le but de cet exercice est de vous apprendre à appliquer les différentes connaissances concernant les servlet et JSP.

3.4.2. Sujet de l'exercice

Vous devrez réaliser une application Web mêlant servlet et JSP gérant les commandes des utilisateurs sur un site de vente en ligne de jeux vidéo. Ce type de système est souvent appelé *gestion de caddie* ou *panier* (d'autre termes anglais : *shopping cart*, *basket*). Vous allez, pour réaliser ce système, utiliser l'outil de session disponible pour y stocker le minimum de données possible et simuler des commandes.

Voici les fonctions attendues :

- Afficher les produits présents dans la boutique (nom produit,prix,description…)
- Voir description détaillée d'un produit
- Ajouter des produits dans un caddie
- Afficher le caddie (modifier,supprimer des produits…)

Vous stockerez la liste des produits de la boutique et du caddie dans une collection de votre choix.

Vous devrez nous fournir un WAR de cette application.