Débuter pas à pas avec Wicket et Eclipse JEE 1.4

par Philippe Sénéchal (Acceuil)

Date de publication : 24 août 2008

Dernière mise à jour :

Wicket est un framework Java utilisé pour développer des applications web. Ce tutoriel s'adresse à tous ceux qui souhaitent découvrir le développement d'applications web à l'aide de ce framework sans connaissance préalable de JEE.

I - Préambule

La complexité et la multiplicité des modèles de programmation que l'on peut utiliser dans JEE créent une étape difficile à franchir quand on veut aborder ce domaine après un apprentissage du langage java. Wicket représente à mon avis une possibilité de découvrir les notions de base de la programmation d'application web en se servant quasi exclusivement de ses connaissance en java et en Html. Ce tutoriel s'applique à suivre cette démarche avec une approche pas à pas de la mise en oeuvre de wicket et de ses mécanismes de base.

II - Installation d'un projet Wicket

Une application web Java est en général contenue dans un fichier war (Web Application aRchive) et présente la structure suivante :

répertoire racine
  +META-INF
  +WEB-INF
    +classes : les servlet (classes traitant les requêtes HTTP) et autres classes
    +lib : les librairies utilisées par l'application (fichiers .jar)
  web.xml : le descripteur de déploiement de l'application
  ressources (pages html, images, fichiers css...)

Pour créer un projet d'application web utilisant wicket, il va donc falloir créer cette structure, et référencer les librairies nécessitées par notre application. On peut le faire simplement à partir d'Eclipse selon la méthode décrite ici : **création d'une application wicket avec eclipse**

Wicket a toutefois prévu une méthode de construction de projet appelée QuickStart que l'on peut trouver sur son site à l'adresse suivante : **http://wicket.apache.org/quickstart.html**.
Cette méthode utilise un outil de **build** appelée Maven2, pour construire ce projet. Il nous faudra donc tout d'abord télécharger et installer Maven 2.
Il va falloir ensuite utiliser la **ligne de commande** avec l'ordre généré sur le site par le quickstart. On peut le faire directement à partir d'eclipse si on a installé le plugin Wicked Shell qui permet d'utiliser la ligne de commande de Windows avec Eclipse.

Attention à ne pas confondre Wicked et Wicket !

II-A - Installation de Maven 2 :

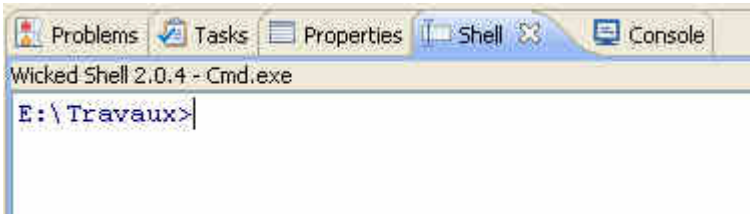
On peut télécharger Maven 2 sur cette page : **http://maven.apache.org/download.html#Installation**.
Il faut ensuite décompresser l'archive dans le répertoire de votre choix: ici ce sera : E:\Programmation. Maven se trouve alors dans le répertoire: E:\Programmation\apache-maven-2.0.9.
Il reste a positionner les variables d'environnement M2_HOME et M2. Pour accéder aux variables d'environnement, sous Windows, il faut à partir du menu *démarrer*, cliquer sur *panneau de configuration*, puis sur *Système*, sur l'*onglet avancé*, puis enfin sur le bouton *Variables d'environnement*.
Dans les variables utilisateurs, il faut cliquer sur *nouveau*, et remplir les cases comme suit :

- Nom de la variable : M2_HOME Valeur de la variable : E:\Programmation\apache-maven-2.0.9
- Nom de la variable : M2 Valeur de la variable : %M2_HOME%\bin
- Nom de la variable : JAVA_HOME Valeur de la variable : C:\Program Files\Java\jdk1.6.0_01 (si elle n'existe pas déjà)
- Mettre à jour la variable : PATH en rajoutant la valeur : %M2%;%Path%

Il existe un plugin maven pour eclipse (m2eclipse) que l'on peut installer à partir de l'update d'éclipse à cette adresse : **http://m2eclipse.sonatype.org/update/**.

II-B - Installation de Wicked Shell

On peut installer Wicked Shell via l'update d'eclipse en suivant les instructions de cette page : **http://www.wickedshell.net/download.html**
Dans le menu *preference* d'Eclipse, il suffit de cocher la case *Use workspace as startup directory* pour que la ligne de commande pointe sur le répertoire de travail d'Eclipse. Ici, c'est le dossier E:\Travaux.
On affiche ensuite la fenêtre de Wicked Shell à l'aide du menu *Show view*:



On peut alors tester l'installation de Maven avec la commande : mvn --version , qui si tout est bien installé devrait afficher le numéro de version de Maven.

II-C - Utilisation du QuickStart

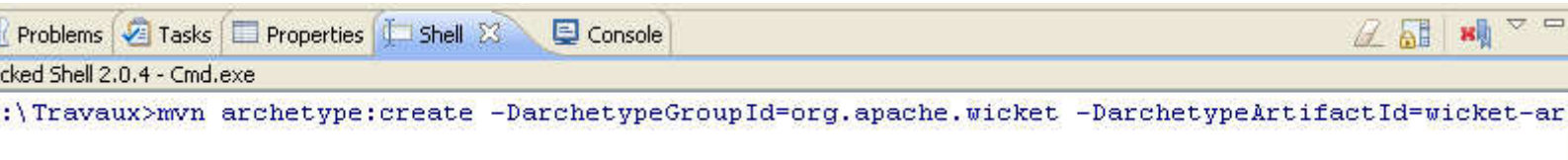
On va remplir les 3 lignes suivantes de la page *Quickstart* du site Wicket:

- *GroupId* qui correspondra au package java (ici on l'appellera : com.tuto)
- *ArtifactId* qui correspondra au dossier contenant le projet (ici : project)
- *Version* qui correspond à la version de Wicket que l'on veut utiliser. (ici : 1.4-SNAPSHOT)

On obtient alors la ligne de commande suivante que l'on va copier:

```
mvn archetype:create -DarchetypeGroupId=org.apache.wicket -DarchetypeArtifactId=wicket-archetype-
quickstart
-DarchetypeVersion=1.4-SNAPSHOT -DgroupId=tuto -DartifactId=project -DremoteRepositories=http://
wicketstuff.org/maven/repository/
```

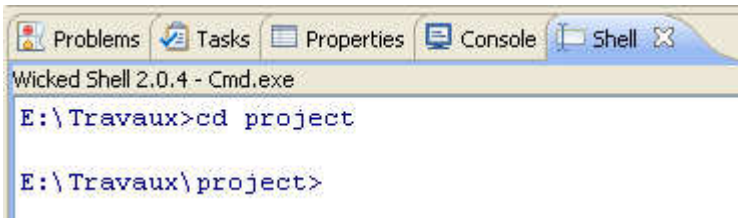
On va ensuite coller cette commande sur la fenêtre de wicked Shell :



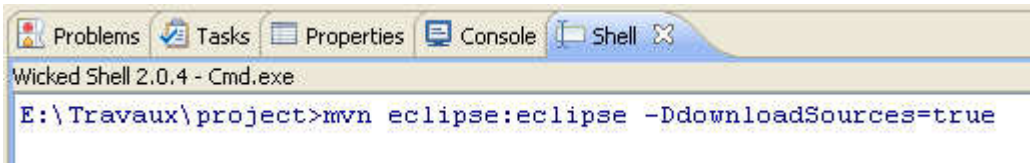
En tapant sur *Entrée*, le projet se construit alors dans le dossier : E:\Travaux\project.

II-D - Ouverture du projet avec Eclipse

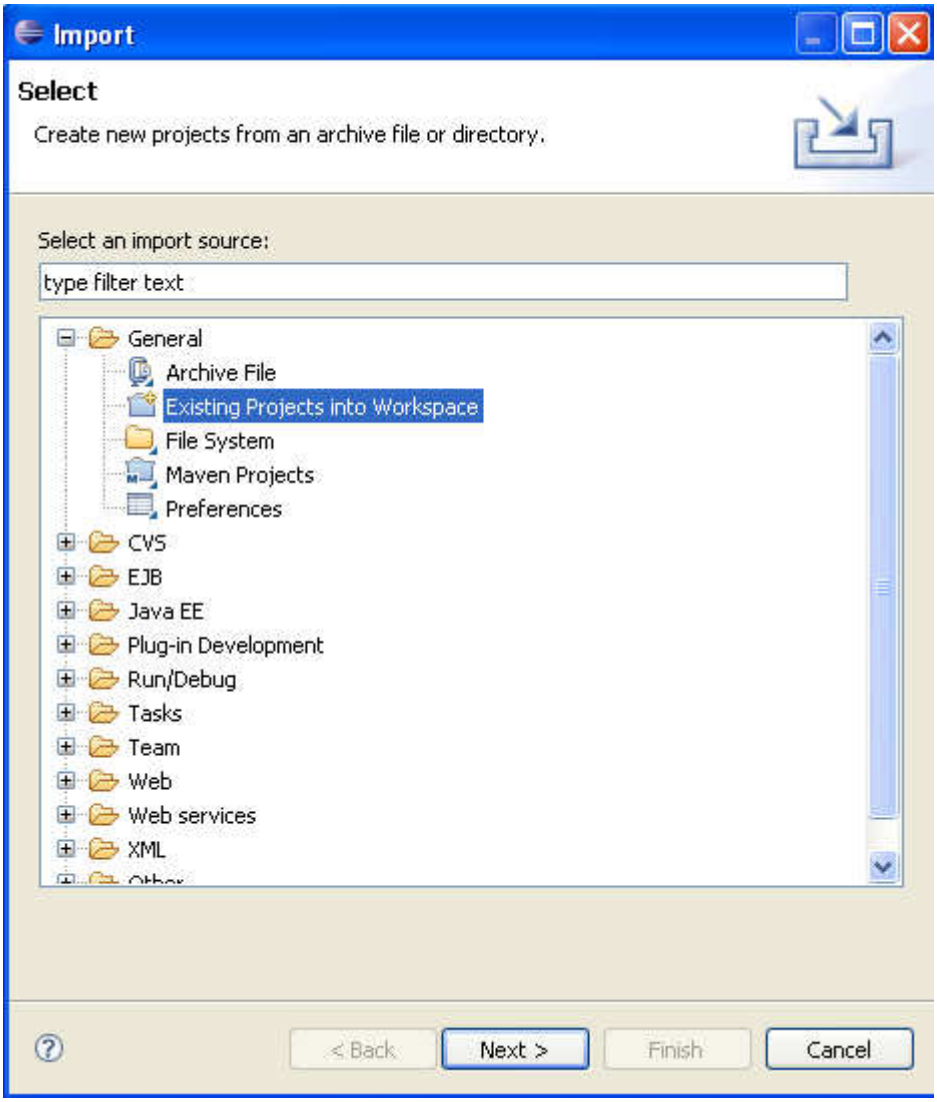
Il va maintenant falloir travailler avec ce projet dans Eclipse J2EE. Pour se placer dans le répertoire de notre projet, on tape la commande suivante : cd project dans la fenêtre Wicked Shell.



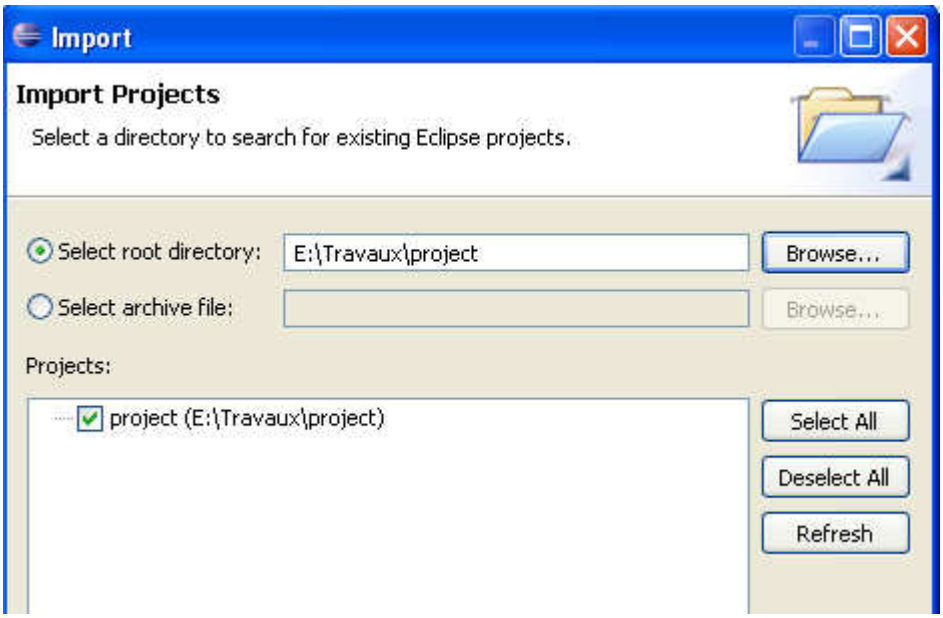
Puis on colle la commande suivante qui permettra l'utilisation du projet par Eclipse : mvn eclipse:eclipse -DdownloadSources=true



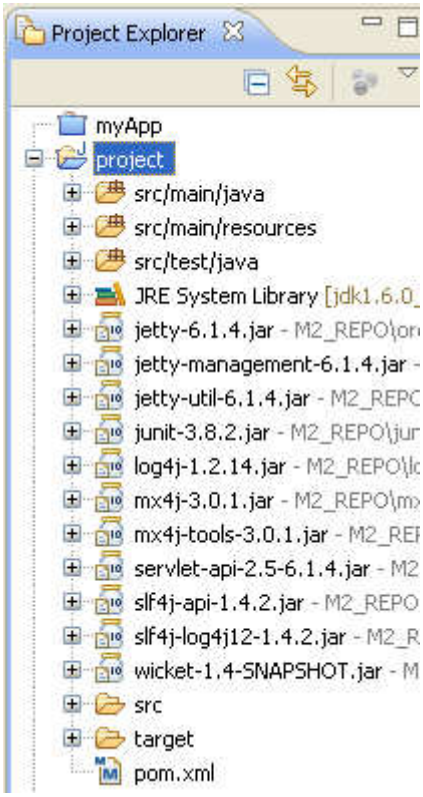
Il faut ensuite ouvrir le menu *File* d'Eclipse puis cliquer sur *Importer* . Dans la fenêtre *Import*, cliquer sur le noeud *General*, puis sur *Existing Project in Workspace* :



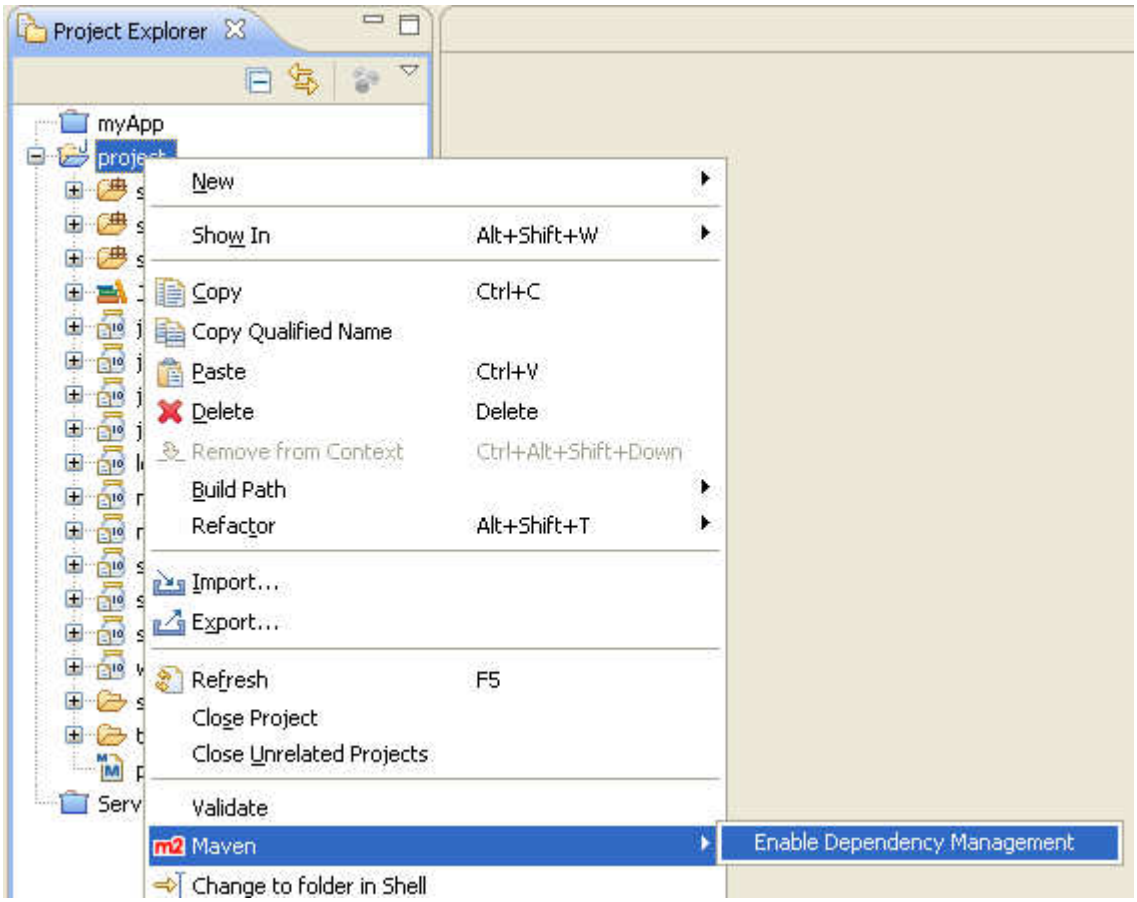
Puis sélectionner le répertoire du projet avant de fermer la fenêtre :



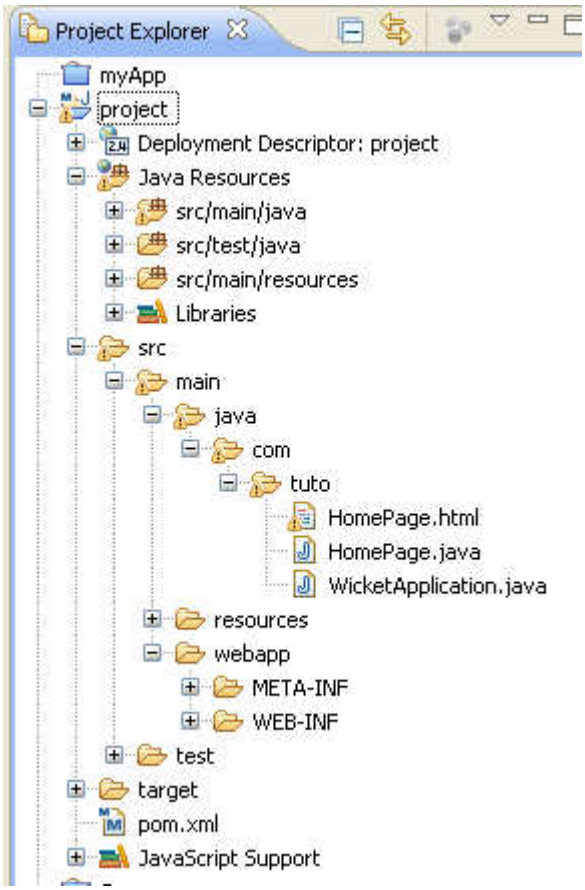
Notre projet est maintenant intégré à Eclipse, et en cliquant sur son nom dans l'onglet *Project Explorer*, on peut visualiser les différents répertoires et fichiers qui le compose.



Il reste ensuite à intégrer les dépendances en cliquant sur *Enable dependency management* comme suit :



Notre projet a alors cette structure :



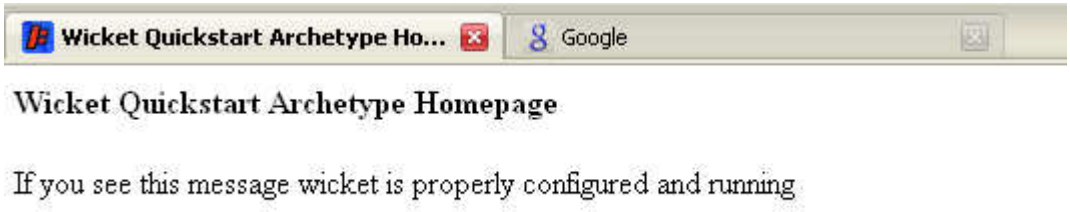
Le projet est maintenant prêt à l'emploi sous Eclipse, mais pour déployer le projet Wicket sur internet, il nous faut un conteneur de **servlet**, c'est à dire une application permettant d'exécuter des servlets sur un serveur web.

III - Installation d'un conteneur de servlet

Je vais ici détailler l'installation de deux conteneurs de servlet : Jetty et Tomcat.

III-A - Installation de Jetty

Jetty est un serveur open source 100% Java. Il est déjà présent dans l'installation que l'on vient de faire avec le QuickStart, puisque l'on trouve ses librairies dans les dépendances maven.
On peut l'utiliser sans autres préparations en tapant dans la fenêtre de Wicked Shell : mvn jetty:run. Si tout s'est bien passé, on peut alors visualiser dans notre navigateur la page suivante de notre application Wicket à cette adresse : http://localhost:8080/project/



Jetty a un autre avantage : le hot deployment; c'est à dire que le moteur de servlet va vérifier (selon un intervalle de temps que l'on va définir) si les sources ont été modifiées, et redéployer l'application aussitôt, sans que l'on ait a l'arrêter et à le redémarrer. Pour cela, il faudra rajouter deux lignes dans le fichier de configuration pom.xml du projet.

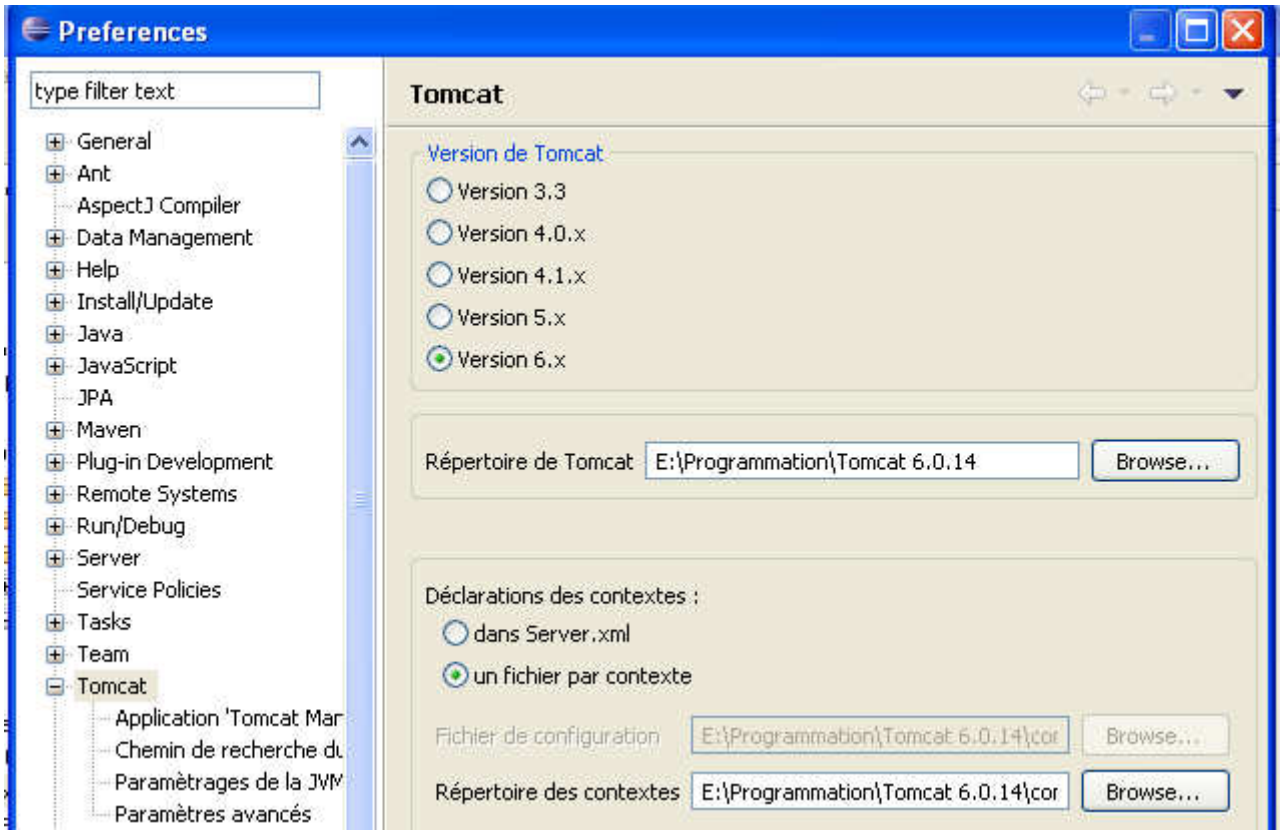
Pour pouvoir l'arrêter à partir de la ligne de commande en tapant mvn jetty:stop, il va aussi nous falloir rajouter 2 lignes dans ce même fichier.
Ces modifications sont surlignées ici en bleu :

Mais à côté de ces avantages, l'utilisation de Jetty dans la configuration Maven a 2 inconvénients à mon avis : la nécessité d'utiliser la ligne de commande, et l'absence de prévisualisation du résultat dans Eclipse. Pour pallier à cela, on peut installer Jetty selon la méthode utilisée ci-dessous pour l'installation de Tomcat.

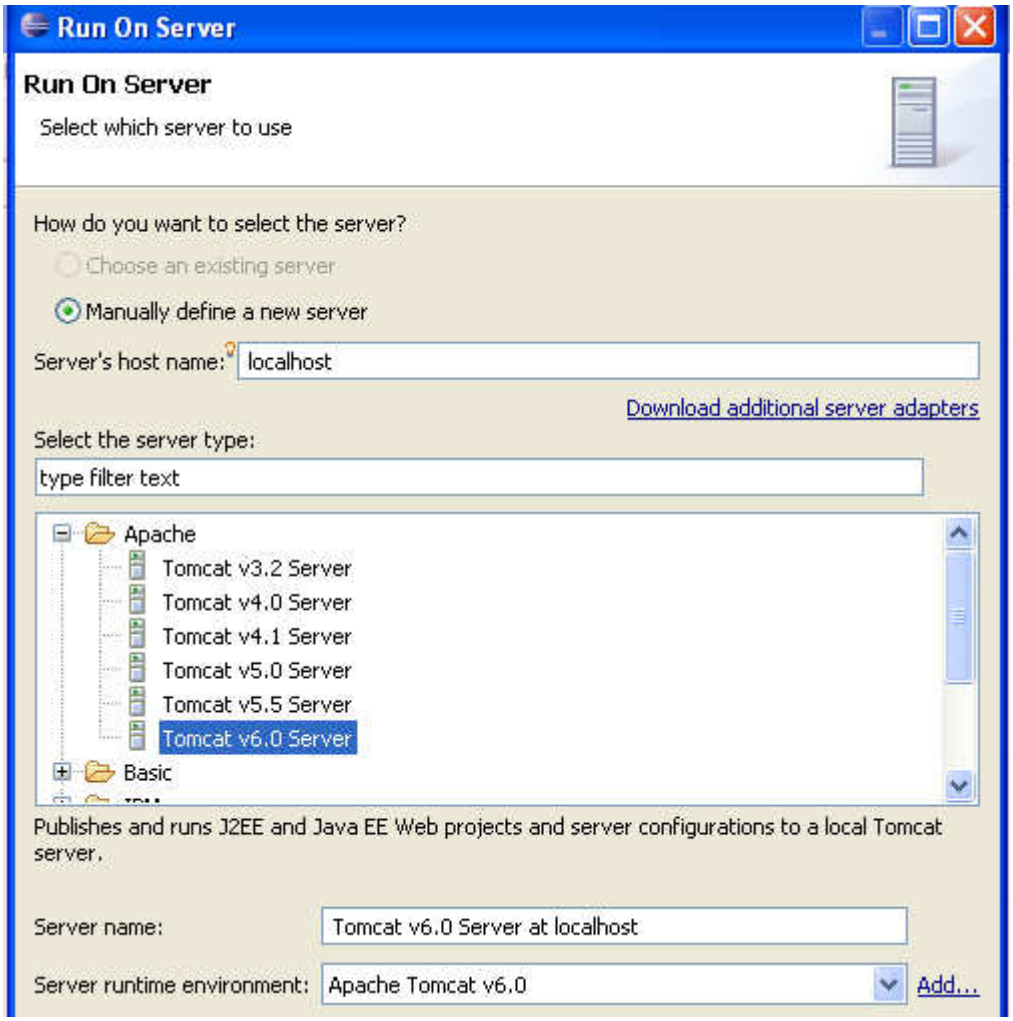
Quand jetty est démarré avec la commande mvn jetty:run, l'invite de commande qui pourrait permettre de stopper le serveur n'est plus active. Pour pallier à celà, il peut être utile de lancer la commande start dans la fenêtre Wicked shell, avant de démarrer le serveur, ce qui permet d'ouvrir une autre fenêtre de commande à partir de laquelle on pourra arrêter le serveur.

III-B - Installation de Tomcat

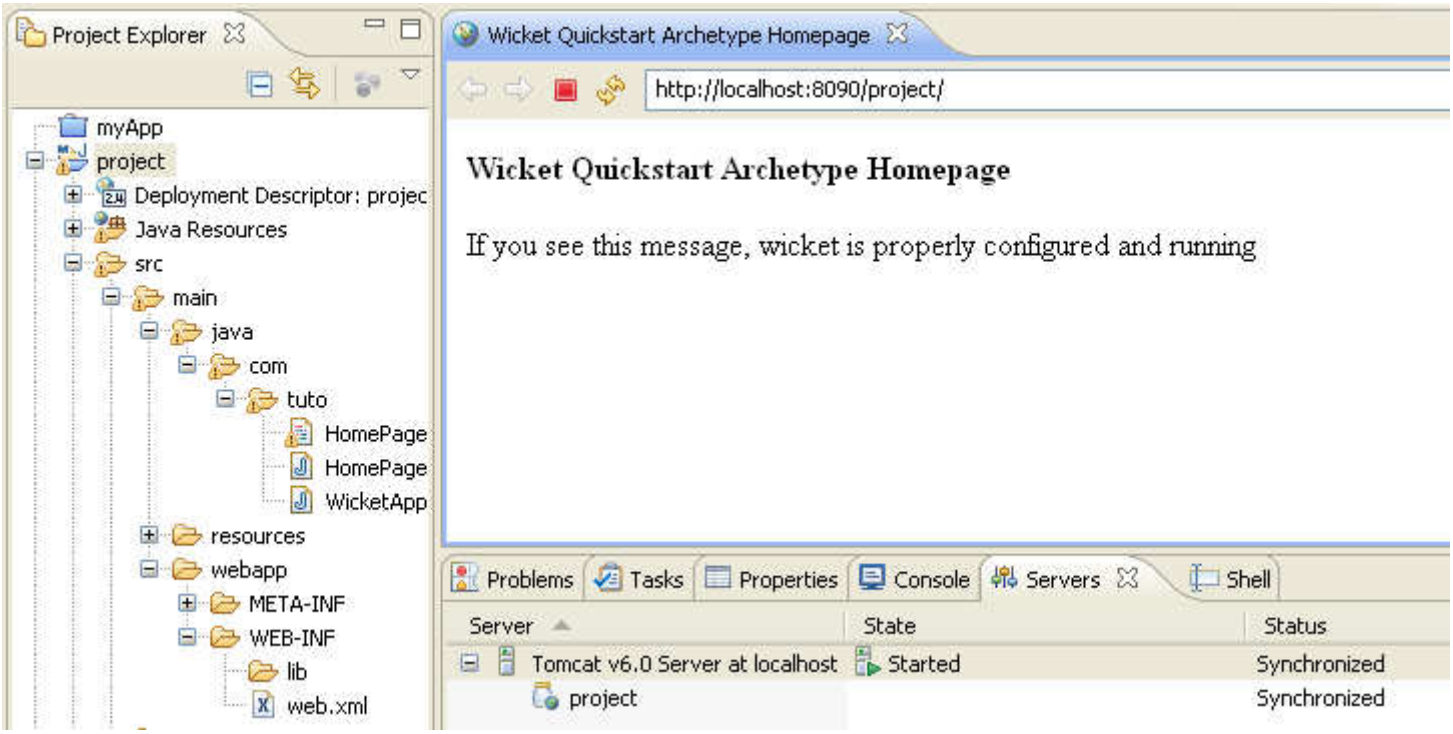
Tomcat est le conteneur de servlet open source java probablement le plus utilisé. Il peut être téléchargé dans sa version 6 à cette adresse : **http://tomcat.apache.org/download-60.cgi**
Il s'agit d'un fichier compressé que l'on décompressera dans le dossier de notre choix : ici ce sera : E:/Programmation.
Il existe un plugin très utile pour Tomcat, que l'on peut télécharger à cette adresse : **http://www.eclipsetotale.com/tomcatPlugin.html** . Il s'agit d'un fichier compressé que l'on décompressera dans le dossier *Dropins* d'Eclipse.
Il faut ensuite renseigner le chemin de Tomcat dans la fenêtre *préférence* d'Eclipse comme suit :



Tomcat est prêt à fonctionner. Nous allons lancer notre application avec Tomcat en faisant un clic droit sur notre projet, puis en cliquant sur *Run As* puis sur *Run on Server* : On se retrouve alors sur cette fenêtre, ou l'on choisira *Tomcat v6.0 Server*:



Tomcat va alors charger notre projet et l'afficher dans le navigateur interne d'Eclipse sur le port 8090 de localhost. La fenêtre *Servers* d'Eclipse permet de démarrer et d'arrêter Tomcat à volonté.



Après ce préambule qui nous a permis d'installer et de faire fonctionner le projet Wicket du Quickstart, nous allons maintenant nous intéresser à son contenu et aux principes de Wicket.

IV - Principes de Wicket

IV-A - Le fichier de configuration :

Le fichier web.xml du dossier *webbapp* est le descripteur de déploiement de l'application, et contient les informations de configuration de celle-ci. Sans entrer dans les détails, ce fichier informe le conteneur de servlet qu'il faut utiliser un filtre wicket, et définit la classe d'initialisation de l'application : ici elle se nomme WicketApplication.

IV-B - La classe d'initialisation :

Il s'agit de la classe WicketApplication comme nous l'a apprit le fichier web.xml. Cette classe étend la classe WebApplication, et sa méthode getHomePage() retourne la classe servant à afficher la page d'acceuil de notre application nommée ici : HomePage. Les nombreuses méthodes de cette classe permettent aussi de configurer l'application.

IV-C - La page d'accueil de l'application :

Toute page wicket est constituée de deux éléments :

- une classe qui doit hériter de la classe WebPage du paquet wicket (ou d' une classe dérivée elle même de WebPage),
- un fichier html portant le même nom que la classe. Ce fichier pourra utiliser des balises et attributs wicket servant à identifier les composants java(**http://cwiki.apache.org/WICKET/wickets-xhtml-tags.html**).

Ces deux éléments doivent être situés dans le même dossier. (Ce dernier point peut être modifié de différentes façons comme on peut le voir ici : **http://cwiki.apache.org/WICKET/control-where-html-files-are-loaded-from.html**).
On retrouve donc dans nos sources les deux fichiers : HomePage.java et HomePage.html.

La page *html* est classique, hormis la ligne suivante :

```
<span wicket:id="message">message will be here</span>
```

On constate deux choses :

- dans la balise *span*, il y a un attribut wicket:id intitulé message
- et à l'affichage, ce n'est pas la phrase message will be here qui apparaît, mais If you see this message, wicket is properly configured and running. Si l'on regarde maintenant notre fichier HomePage.java on retrouve ces deux éléments comme paramètres de la classe Label. Dans les imports on constate que cette classe provient du paquet wicket.

extrait de HomePage.java

```
add(new Label("message", "If you see this message, wicket is properly configured and running"));
```

On en déduit donc qu'à la clé message, est associée la valeur If you see this message, wicket is properly configured and running, et que c'est cette valeur qui sera utilisée à la place de celle contenue dans la balise *span*. De cette manière, on peut utiliser un attribut *wicket:id* pour chaque composant que l'on veut rajouter sur nos pages.
On a là les principes essentiels sur lequel le framework est construit : l'utilisation du namespace wicket dans les balises du code html, avec dans la classe java correspondant au fichier html du même nom, les composants java permettant l'affichage et la manipulation des données de l'application.

V - Personnalisation de notre application

On a vu comment installer et déployer une application Wicket, et comment s'articulent les fichiers composants notre application. On va maintenant personnaliser celle-ci, pour pénétrer un peu plus dans le mécanisme de fonctionnement de Wicket. Pour ce faire, on va rajouter comme dans toute application web classique, un peu de *css* pour habiller l'application, et créer une application à deux pages, ce qui va nous permettre d'aborder l'utilisation des liens et de la mise en page avec Wicket.

V-A - Création d'un gabarit :

On va donner à nos pages une structure classique en ajoutant dans le body de la page *html* :

- une en-tête (id="en tete") comprenant un bandeau avec un logo (id="bandeau") et un menu horizontal (id="menu"),
- un corps (id="corps")
- et un pied de page avec un autre logo (id="pied").
- Pour coller le pied de page en bas de la page, on va rajouter un id="wrapper" autour de l'en-tête et du corps.

On rajoute un lien vers le fichier *css* dans la balise *Head*, et on supprime le titre.
Pour utiliser les fichiers *css* et les images, il faut les télécharger à l'aide des liens ci-dessous, et les copier dans le dossier webapp

- **base.css**
- **logo.png**
- **logoFoot.gif**

On place notre *label* dans le *div* id="corps" et on modifie le message associé dans le code de HomePage.java. (En mettant par exemple : Bienvenue sur notre page d'accueil).

On obtient alors ce code que l'on va utiliser pour le fichier HomePage.html

HomePage.html

```
<html xmlns:wicket="http://wicket.apache.org/">
    <head>
        <title>Accueil</title>
        <link rel="stylesheet" type="text/css" href="base.css" />
    </head>
    <body>
        <div id="wrapper">
            <div id="en tete">
                <div id="bandeau">
                    <img src="logo.png">
                </div>
                <div id="menu">
                </div>
            </div>
            <div id="corps">
                <span wicket:id="message"></span>
            </div>
        </div>
        <div id="pied">
            <img src="logoFoot.gif">
        </div>
    </body>
</html>
```

V-B - Mise en page :

Maintenant que notre application a une identité visuelle, il serait intéressant que toutes nos pages l'utilisent. Pour ce faire, on va utiliser le gabarit de page que nous venons de créer dans le chapitre précédent. IL suffit de retirer la ligne contenant le message d'accueil qui caractérise notre HomePage, pour avoir une page vierge ayant cette identité visuelle.
Nous allons donc après avoir effacé la ligne span wicket:id="message" /span , renommer dans nos sources le fichier HomePage.html en BasePage.html, et y associer le fichier java suivant :

BasePage.java

```
package com.tuto;

import org.apache.wicket.PageParameters;
import org.apache.wicket.markup.html.WebPage;

public class BasePage extends WebPage{

    public BasePage(final PageParameters parameters) {
    }
}
```

Il nous faut maintenant recréer un fichier HomePage.html, et modifier le fichier HomePage.java pour qu'il utilise notre page de base. Nous allons utiliser le principe de *Markup Inheritance* (mettant en jeu des mécanismes d'héritage):
Cette méthode consiste à employer une balise wicket:child/ dans le corps de notre BasePage.html, qui sera remplacée lors de l'appel de la page par le contenu html que l'on placera entre les balises wicket:extend et /wicket:extend de notre HomePage.
De la même manière, on placera des balises wicket:head et /wicket:head dans les 2 fichiers *html*, et on rajoutera un titre au fichier HomePage.html. Le code de notre HomePage.html sera alors :

HomePage.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  <wicket:head>
    <title>Accueil</title>
  </wicket:head>
```

HomePage.html
```
  <body>
    <wicket:extend>
      <span wicket:id="message"></span>
    </wicket:extend>
  </body>
</html>
```

Et celui de notre BasePage.html contiendra :

BasePage.html
```
<html xmlns:wicket="http://wicket.apache.org/">
  <head>
    <wicket:head>
      <link rel="stylesheet" type="text/css" href="base.css" />
    </wicket:head>
  </head>
  <body>
    <div id="wrapper">
        <div id="en tete">
             <div id="bandeau">
                 <img src="logo.png">
             </div>
             <div id="menu">
             </div>
         </div>
            <div id="corps">
                <wicket:child/>
            </div>
     </div>
     <div id="pied">
         <img src="logoFoot.gif">
     </div>
  </body>
</html>
```

Quand à notre fichier HomePage.java, il suffit de le dériver de BasePage.java, et d'ajouter notre label au niveau de son constructeur :

HomePage.java
```
package com.tuto;

import org.apache.wicket.PageParameters;
import org.apache.wicket.markup.html.basic.Label;

public class HomePage extends BasePage {

    public HomePage(PageParameters parameters) {
        super(parameters);
        add(new Label("message", "Bienvenue sur notre page d'accueil"));
    }
}
```

Si on visualise notre page web, on peut s'apercevoir qu'elle n'est pas modifiée, mais nous savons maintenant que nous pouvons utiliser ce principe pour créer d'autres pages avec la même identité visuelle.

V-C - Navigation :

Commençons par créer une nouvelle page html que l'on va nommer PageOne sur le modèle de notre HomePage :

PageOne.html
```
<html xmlns:wicket="http://wicket.apache.org/">
```

PageOne.html
```
   <wicket:head>
     <title>Accueil</title>
   </wicket:head>
   <body>
     <wicket:extend>
       <span wicket:id="premierePage"></span>
     </wicket:extend>
   </body>
</html>
}
```

Puis le fichier java correspondant sur le modèle de HomePage.java :

PageOne.java
```
package com.tuto;

import org.apache.wicket.PageParameters;
import org.apache.wicket.markup.html.basic.Label;

public class PageOne extends BasePage {

    public PageOne(PageParameters parameters) {
        super(parameters);
        add(new Label("premierePage", "Ceci est une nouvelle page de notre application"));
    }
}
```

Pour pouvoir utiliser cette nouvelle page, il va falloir utiliser un mécanisme de navigation que l'on va baser sur des liens . On va placer ces liens dans un menu dont on a déjà déterminé la place lors de la création du gabarit de la page vierge : c'est à dire dans le div id="menu" de notre BasePage.html, puisque les liens devront pouvoir être utilisé à partir de chaque page.
On va placer ces liens dans une liste, que l'on affichera selon une disposition horizontale grâce au fichier css. Ces liens seront affichés dans une balise html classique a href, avec à nouveau un attribut wicket:id qui permet de référencer l'adresse dans le fichier java correspondant.

BasePage.html
```
<html xmlns:wicket="http://wicket.apache.org/">
  <head>
    <link rel="stylesheet" type="text/css" href="base.css" />
  </head>
  <body>
    <div id="wrapper">
      <div id="en tete">
        <div id="bandeau">
          <img src="logo.png">
        </div>
        <div id="menu">
          <ul>
             <li><a href="#" wicket:id="homePage">Acceuil</a></li>
             <li><a href="#" wicket:id="pageOne">Page 1</a></li>
          </ul>
        </div>
      </div>
      <div id="corps">
      </div>
    </div>
    <div id="pied">
      <img src="logoFoot.gif">
    </div>
  </body>
</html>
}
```

Et dans le fichier BasePage.java on utilise la classe BookmarkablePageLink du paquet *wicket*, qui associe à la clé référencée dans le wicket:id du lien, la classe de la page en question.

BasePage.java

```
package com.tuto;

import org.apache.wicket.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.link.BookmarkablePageLink;

public class BasePage extends WebPage{

    public BasePage(final PageParameters parameters) {
        add(new BookmarkablePageLink("homePage", HomePage.class));
        add(new BookmarkablePageLink("pageOne", PageOne.class));
    }

}
```

Avec wicket, les liens dérivent de la classe Link, qui se sert de la méthode onClick() dans laquelle on peut fournir un traitement et utiliser des méthodes setResponsePage(...) pour indiquer la page à afficher. Les deux classes suivantes dérivent de la classe Link:

- la classe BookmarkablePageLink. Cette classe crée une page sans état (*stateless* page); les pages pourront être marqués comme favori. L'adresse de la page utilise les éléments : wicket:bookmarkablePage=: suivi du chemin de la classe dans le paquet. Ici on aura donc :
  http://localhost:8080/project/?wicket:bookmarkablePage=:com.tuto.PageOne
  Les méthodes setResponsePage(Class c) et setResponsePage(Class c, PageParameters pp) créent des pages de ce type.
- avec la classe PageLink la page crée ne pourra pas être utilisée dans les favoris: elle crée une *statefull* page (page maintenue en mémoire côté serveur); les liens seront maintenus le temps de la session, et auront la forme :
  http://localhost:8080/project/?wicket:interface=:X::::, X étant incrémenté à chaque changement de page.
  La méthode setResponsePage(Page p) crée une page de ce type.

Le chapître suivant va nous permettre de mieux appréhender la signification de ces termes *stateless* et *statefull, et* le fonctionnement de ces liens.

VI - Session

Le protocole HTTP est un protocole dit sans état, c'est à dire qu'il traite chaque requête indépendemment les unes des autres. Or, s'adressant potentiellement à plusieurs utilisateurs; une web application va nécessiter un contexte propre à chacun englobant la sucession des pages renvoyées à ces utilisateurs, et l'interactivité générée par ces pages. On appelle ce contexte Session. C'est au conteneur de servlet que revient la gestion de ces Sessions.
Nous allons aborder l'utilisation de cette notion dans wicket en demandant à notre application de souhaiter la bienvenue à chaque utilisateur par son prénom, que l'on retrouvera ensuite sur chaque page.
Il va donc falloir créer un formulaire demandant à l'utilisateur son prénom, et une classe User pour pouvoir utiliser cette donnée.

VI-A - L'utilisateur

Notre classe User va tout simplement comporter un champ String avec ses accesseurs, et un constructeur surchargé.

User.java

```
package com.tuto;

public class User{
```

User.java

```
 private String name;

 public User(){
 }

 public User(String s){
  setName(s);
 }

 public String getName() {
  return name;
 }

 public void setName(final String name) {
  this.name = name;
 }

}
```

VI-B - La Session

Pour pouvoir accéder à la session à partir du code, il va falloir redéfinir dans notre classe WicketApplication, la méthode newSession(Request, Response) de la classe Application (la classe parente de WebApplication). Cette méthode retourne un Objet de la classe abstraite Session, parente de WebSession. On va donc créer une classe que l'on va nommer UserSession, qui va dériver de WebSession, dans laquelle on va utiliser un champ User avec ses accesseurs. Notre utilisateur appartiendra alors à la session et non à une WebPage.

WicketApplication.java

```
package com.tuto;

import org.apache.wicket.Request;
import org.apache.wicket.Response;
import org.apache.wicket.Session;
import org.apache.wicket.protocol.http.WebApplication;
import org.apache.wicket.protocol.http.WebSession;

public class WicketApplication extends WebApplication{

 public WicketApplication(){
 }

 @Override
 public Session newSession(Request request, Response response){
          return new UserSession(request);
     }

 @Override
 public Class getHomePage(){
  return Login.class;
 }

}
```

UserSession.java

```
package com.tuto;

import org.apache.wicket.Request;
import org.apache.wicket.protocol.http.WebSession;

public final class UserSession extends WebSession{

    private User user;
```

UserSession.java

```
    protected UserSession(Request request){
        super(request);
    }

    public User getUser(){
        return user;
    }

    public void setUser(final User user){
        this.user = user;
    }
}
```

VI-C - Le Formulaire

On va créer une nouvelle page pour notre formulaire à l'aide des deux fichiers sources requis que l'on appellera Login.java et Login.html.
Pour Login.html, on va utiliser une classique balise Form, avec une question, une zone de texte et un bouton, auxquels on va rajouter un attribut wicket:id.

Login.html

```
<html xmlns:wicket="http://wicket.apache.org/">
        <wicket:extend>
            <form wicket:id="accueilForm">
             <h3>Quel est votre prénom ? </h3>
             <input type="text" wicket:id="monNom">
             <input type="submit" value="OK">
            </form>
            <br/>
        </wicket:extend>
</html>
```

Le fichier Login.java sera plus conséquent, car il va contenir le traitement de l'information.
Logiquement avec wicket, à la balise Form correspond une classe Form. On va utiliser une classe interne que l'on va nommer LoginForm dérivée de Form, qui va contenir un champ String pour récupérer le prénom, avec ses accesseurs, un TextField et une méthode onSubmit() pour répondre à l'appui sur le bouton.

LoginForm.java

```
package com.tuto;

import org.apache.wicket.PageParameters;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.model.PropertyModel;

public class Login extends BasePage{

 public Login(PageParameters parameters) {
  super(parameters);
  add(new LoginForm("loginForm"));
     }

     private class LoginForm extends Form {

      private String leNom;

      public LoginForm(String id) {
           super(id);
           TextField fieldname = new TextField("monNom",new PropertyModel(this,"monNom"));
              fieldname.setRequired(true);
              add(fieldname);
      }
```

LoginForm.java

```
        public void onSubmit() {
         UserSession session = (UserSession)getSession();
         session.setUser(new User(getMonNom()));
                setResponsePage(new HomePage(null));
            }
           public String getMonNom() {
    return leNom;
   }
   public void setMonNom(String leNom) {
    this.leNom = leNom;
   }
      }

}
```

Ce code introduit deux nouveautés dans notre tutoriel :

- La notion de modèle : dans l'instanciation du TextField, on trouve comme deuxième paramètre une classe PropertyModel, qui introduit ici la notion de modèle. Cette notion justifie à elle seule un tutoriel , ce qui a été fait il y a peu de temps par **Jawher Moussa** : **Exploration des modèles de Wicket**. Je me contenterai donc ici de vous orienter vers son article pour comprendre et approfondir cette notion qui est fondamentale dans l'utilisation de wicket.
- L'utilisation de l'accès à la session dans une WebPage : on crée une intance de notre UserSession à l'aide d'un cast; puis on crée un User dont on a récupéré le prénom entré dans le TextField et stocké par le PropertyModel dans le champ leNom; puis on transmet notre instance de User à notre UserSession.
  De la même façon, on va accéder à la session pour utiliser le prénom de l'utilisateur dans les messages de nos pages.

extrait de HomePage.java

```
...
public HomePage(PageParameters parameters) {
 super(parameters);
 User nom = ((UserSession)getSession()).getUser();
 add(new Label("message", "Bienvenue sur notre page d'accueil "+nom.getName()));
 ....
}
...
```

extrait de PageOne.java

```
...
public PageOne(PageParameters parameters) {
    super(parameters);
    UserSession session = (UserSession)getSession();
    User nom = session.getUser();
    add(new Label("premierePage", "Bienvenue sur la page une de notre application "+nom.getName()));
}
...
```

Par ailleurs, on remarque l'utilisation de la méthode setRequired(true) avec le TextField, qui empêche la validation d'une entrée nulle au niveau du formulaire. Pour améliorer cette fonctionnalité, on peut ajouter à notre page un composant FeedbackPanel, qui comme son nom le laisse deviner va nous permettre d'afficher divers retours provenants des composants de la page. Plusieurs méthodes de la classe Component peuvent être utilisées à cet effet en fonction du type de message qui doit être retourné : info(..) debug(..) error(..). Sa mise en oeuvre est toute simple: il suffit d'ajouter dans les deux fichiers sources les lignes suivantes :

extrait de Login.html

```
<html xmlns:wicket="http://wicket.apache.org/">
        <wicket:extend>
         <span wicket:id="feedback"></span>
            <!--form ...>
```

extrait de Login.html

```
            < code du Form>
            </form-->
            <br/>
        </wicket:extend>
</html>
```

extrait de Login.java

```
package com.tuto;

// imports ...

public class Login extends BasePage{

 public Login(PageParameters parameters) {
  super(parameters);
  add(new LoginForm("loginForm"));
  add(new FeedbackPanel("feedback"));
    }

    private class LoginForm extends Form {
     // cf code plus haut
    }
}
```

Il suffit de valider le formulaire avec une zone de texte vide pour tester l'usage du FeedbackPanel. De la même façon, à titre d'exemple, on peut ajouter un feedbackpanel à notre page d'accueil, avec un contenu d'information.

extrait de HomePage.html

```
...
        <wicket:extend>
            <h2 wicket:id="message"></h2><br/>
            <div id="feedbackPanel">
             <span wicket:id="feedback"></span>
            </div>
        </wicket:extend>
...
```

extrait de HomePage.java

```
...
public class HomePage extends BasePage {

 public HomePage(PageParameters parameters) {
  super(parameters);
  User nom = ((UserSession)getSession()).getUser();
  add(new Label("message", "Bienvenue sur notre page d'accueil "+nom.getName()));
  add(new FeedbackPanel("feedback"));
  info("Vodre id de session = "+((UserSession)getSession()).getId());
  String date = DateFormat.getDateInstance().format(new Date());
  String time = DateFormat.getTimeInstance(DateFormat.SHORT).format(new Date());
  info("La page a été ouverte le " + date + " à " + time);
 }
}
```

VI-D - Les liens

Il nous reste à modifier la navigation au niveau de l'application, de telle façon que toute nouvelle session démarre par notre page d'identification. Pour cela, il suffira de modifier la classe retournée par la méthode getHomePage() de WicketApplication.java, en remplaçant HomePage.class par Login.class

Il persiste toutefois un problème au niveau de notre page d'identification, car en utilisant le gabarit de notre application, on va pouvoir utiliser les liens du menu pour aller directement à la page d'accueil et à la page Une sans s'être identifié. Cette possibilité aboutit à une page d'erreur, car ces pages utilisent un argument null. Il va donc falloir modifier les deux liens du menu au niveau du fichier BasePage.java.
Pour cela, on va utiliser la méthode onClick() dans laquelle on va tester la saisie du formulaire, puis diriger vers la page adéquate en utilisant la méthode *statefull* setResponsePage(Page).

BasePage.java

```
package com.tuto;

import org.apache.wicket.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.link.Link;

public class BasePage extends WebPage{

    public BasePage(final PageParameters pageParameter) {
     add(new Link("homePage"){
            public void onClick() {
             UserSession session = (UserSession)getSession();
             if(session.getUser()!=null){
              setResponsePage(new HomePage(null));
             }
            }
     });
     //add(new PageLink("homePage",HomePage.class));
        add(new Link("pageOne"){
         public void onClick() {
             UserSession session = (UserSession)getSession();
             if(session.getUser()!=null){
              setResponsePage(new PageOne(null));
             }
            }
        });
    }

}
```

VI - Ressources

- Ressources concernant les applications web java :
  Comprendre l'environnement web
  Le Comportement des Navigateurs Internet
  Le Protocole HTTP
  La technologie des servlets
  Applications web : sécuriser la session utilisateur
  Tomcat et applications web
  Configuration de Tomcat
  Les bases du développement web MVC en Java
- Ressources concernant wicket :
  Le site de wicket
  Hands on wicket : une très bonne vue d'ensemble de wicket
  Redécouvrez le web avec Wicket
  Mise en oeuvre d'Apache Wicket
  Exploration des modèles de Wicket
  Introduction to the Wicket Web Framework : utiliser wicket avec netbeans
  Wicket : the first steps