Systèmes d'information répartis
TD2 initiation aux servlets
avec Tomcat et Eclipse

1. Création d'une servlet connectée à une BD

1.1. Installer WAMPSERVER

Pour simplifier votre travail de gestion de la base de données, nous n'allons pas installer MySQL de manière isolée mais dans une distribution avec le serveur web Apache et l'application Web très pratique phpMyAdmin.

- Avant tout, wamp a besoin d'une dll pour fonctionner. Installez vcredist_x64.exe que vous trouverez dans l'archive
- Récupérez Wamp Server dans l'archive
- Installez Wamp Server dans c:\serveur

Remarque : Si vous avez EasyPhp installé sur votre portable, n'installez rien et utilisez EasyPhp

1.2. Installer le jdbc

Le Java Data Base Connectivity (JDBC) pour MySQL s'appelle Connector/J.

- Installez mysql-connector-java-gpl-5.1.31.msi
- Copier C:\Program Files (x86)\MySQL\MySQL Connector J\mysql-connector-java-5.1.31-bin.jar dans le répertoire C:\Serveur\apache-tomcat-8.0.9\lib. Cela permettra à vos servlets d'utiliser les classes du package pour se connecter à MySQL

1.3. Création d'une table 'personne' et ajout d'enregistrements

- Démarrez WampServer
- Démarrez PhpMyAdmin
- Allez dans la base **test** (créez-la si nécessaire)
- Créez une table **personne** (cle, nom, prenom)
    - cle clé primaire de type int
    - nom varchar
    - prenom varchar
- Ajoutez quelques enregistrements

1.4. Création du pool de connexion

Il y a deux manières d'accéder à une base de données. Soit chaque application J2EE crée sa connexion soit il existe un gestionnaire de connexions qui crée et gère les connexions et les files d'attente pour l'ensemble des applications. Il est évident que seule cette deuxième solution permet d'optimiser la charge sur le SGBD. Cette solution consiste à mettre en œuvre un pool de connexion.
Pour que cela fonctionne, chaque application qui a besoin d'une connexion doit pouvoir la demander au pool. A cet effet, J2EE permet de définir une ressource de manière globale au serveur par un nom qui sera ensuite réutilisable par n'importe quelle application J2EE. C'est le JNDI (Java Naming and Directory Interface).
Pour déclarer une ressource de niveau global au serveur, **il suffit d'éditer avec Notepad++ et de rajouter une balise <Resource />dans le fichier c:\serveur\Tomcat5_5\conf\serveur.xml à l'intérieur de la balise existante GlobalNamingResources** de la façon suivante :

```
<GlobalNamingResources>
…
      <!-- JNDI de la base test -->
      <Resource name="base_test"
            type="javax.sql.DataSource"
            password=""
            driverClassName="com.mysql.jdbc.Driver"
            maxIdle="2"
            maxWait="500"
            username="root"
            url="jdbc:mysql://localhost/test?autoReconnect=true"
            maxActive="500"
```

```
      />
…
</GlobalNamingResources>
```

Le nom visible de la base de données test est maintenant base_test.

Remarque : N'éliminez pas la datasource qui est déjà présente !!

1.5. Lien entre la ressource JNDI et l'application

La deuxième étape consiste à faire la mappage entre notre application web et la ressource JNDI. Pour cela, nous allons enrichir le contexte de notre application. Modifiez le fichier conf\catalina\localhost\ProjetServlet.xml de la façon suivante :

```
<Context path="/ProjetServlet"
      reloadable="true"
      docBase="c:\eclipseworkspace\ProjetServlet"
      workDir="c:\eclipseworkspace\ProjetServlet\work" >
      <ResourceLink
            name="base_test"
            global="base_test"
            type="javax.sql.DataSource"/>
</Context>
```

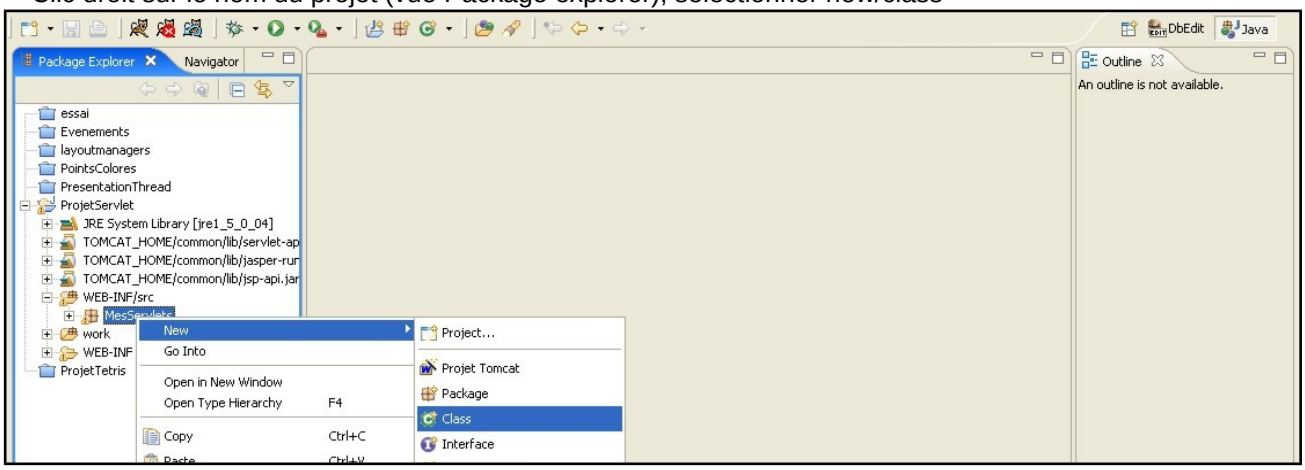
Ce fichier me permet de faire le lien entre la ressource utilisée dans mon fichier et le nom global. Il est alors par exemple, extrêmement simple de modifier une application web si la base de données change de nom…

1.6. Création de la servlet

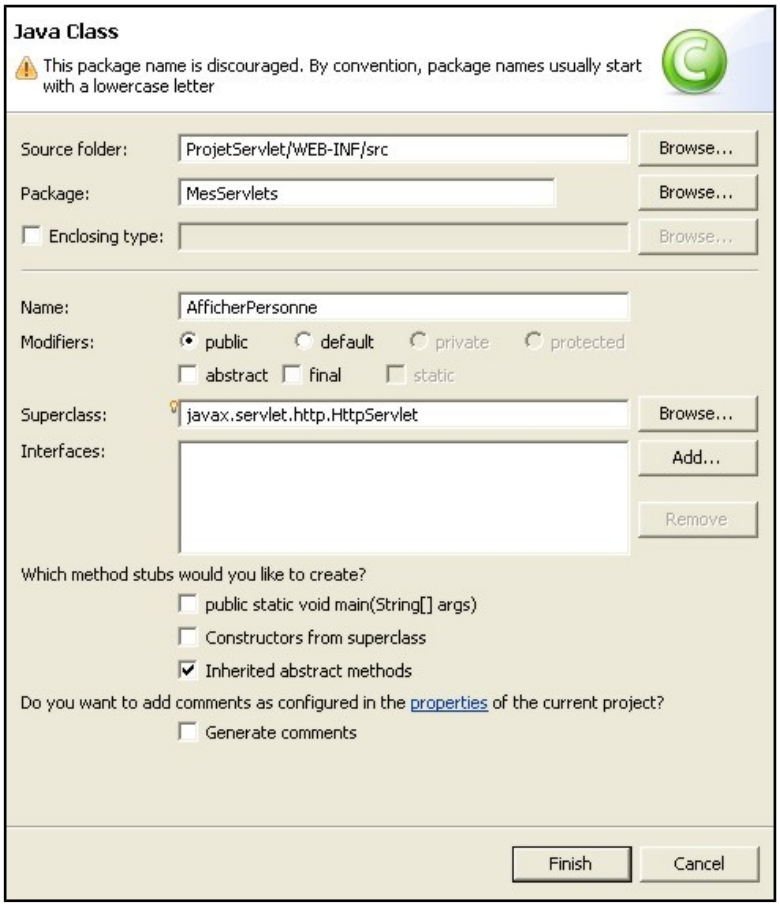
Nous allons créer une servlet toute simple qui récupère les enregistrements dans la table personne et les renvoie sous forme de texte. Nous pourrons ainsi visualiser le résultat dans un navigateur web.

Nous restons toujours dans le même projet. Repasser dans la perspective java.

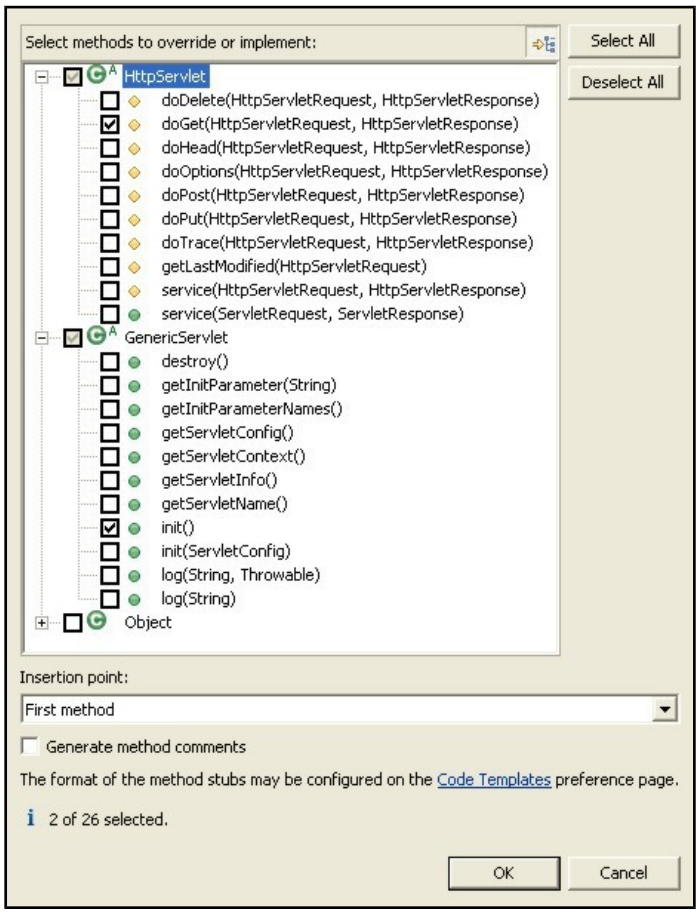
- Clic droit sur le nom du projet (vue Package explorer), sélectionner new/class



- Saisir les infos correspondant à votre nouvelle classe. Vous allez créer un package dans lequel vous allez stocker toutes vos servlets. Ici, MesServlets. Attention de bien définir la superclasse.



- Nous allons maintenant ajouter automatiquement les en-têtes
  - Clic droit sur la classe (cadre de gauche), sélectionner Source/override implement methods. Sélectionner doGet et init



Avant de pouvoir connecter la servlet à la base de données, nous devons tout d'abord créer le pool de connexion. Ce pool nous permettra notamment de maîtriser le nombre de connexions simultanées autorisées.

1.7. Connexion à la base

Une servlet est chargée soit au lancement du serveur Tomcat soit lors de sa première utilisation et reste chargée en mémoire. Dans ces deux cas, il suffit de se récupérer une connexion à la base dans le pool une fois lors de ce chargement. Pour cela, nous allons utiliser la méthode init. Une fois la connexion obtenue, tous les appels à la servlet utiliseront cette connexion.
La méthode init() est lancée une fois lors de l'activation de la servlet.
Dans la méthode init, nous allons récupérer un pointeur sur la ressource JNDI (la base test) puis nous connecter à cette ressource. Pour cela, nous allons charger le contexte puis récupérer un pointeur sur la ressource base_test.

- Ajouter deux attributs dans la classe pour le stockage de la connexion :

```
Connection BD;
DataSource ds;
```

- Ecrire le corps suivant pour la méthode init() :

```
public void init() throws ServletException     {
   try {
   System.out.println("Récupération du contexte");
   Context initCtx = new InitialContext();
   System.out.println("lookup de env");
   Context            envCtx            =            (Context)
initCtx.lookup("java:comp/env");
   System.out.println("lookup de base_test");
   ds=(DataSource) envCtx.lookup("base_test");
   //System.out.println("Datasource chargée");
   }
   catch(Exception er) {
      System.out.println("Erreur  de  chargement  du  contexte  "
+ er);
   }  }
```

- **Ajouter les imports nécessaires avec l'option organize imports. Pour info, la classe Connection fait partie de java.sql.**
- Vous pouvez remarquer que l'on ne se connecte pas à la base comme vous avez pu le faire en php mais que l'on récupère une connexion existante. La connexion est gérée par le pool au niveau du serveur qui attribue aux demandeurs les connexions disponibles.

1.8. Affichage du contenu d'une table

Ecrivons maintenant le contenu de la méthode doGet()

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
  throws ServletException, IOException {
    try {
    BD=ds.getConnection();
    Statement s = BD.createStatement();
    ResultSet r = s.executeQuery("select * from personne");
    PrintWriter out=null;
    resp.setContentType("text/html");
    out = resp.getWriter();
    out.println("<html>");
    out.println("<head><title> Test servlet </title></head>");
    out.println("<body>");
    out.println("Contenu de la table personne <BR>");
    out.println("<table>");
    out.println("<TR>");
    out.println("<TD>Nom</TD>");
    out.println("<TD>Prénom</TD>");
    out.println("</TR>");
    while (r.next()) {
      out.println("<TR>");
      out.println("<TD>");
      out.println(r.getString("nom"));
      out.println("</TD>");
      out.println("<TD>");
      out.println(r.getString("prenom"));
      out.println("</TD>");
      out.println("</TR>");
      }
    out.println("</table>");
    out.println("</body>");
    out.println("</html>");
```

```
      r.close();
      s.close();
      BD.close();
      s = null;
      r = null;
   } catch (java.sql.SQLException ex) {
      System.out.println("Erreur d'exécution de la requête SQL \n"+ex);
   }
}
```

Vous pouvez voir ici deux classes très importantes pour la consultation de bases de données : Statement et RecordSet. **Statement** vous permet de manipuler des tables et d'exécuter vos requêtes. **RecordSet** contient le jeu d'enregistrement résultat de la requête.

1.9. Mappage de la servlet

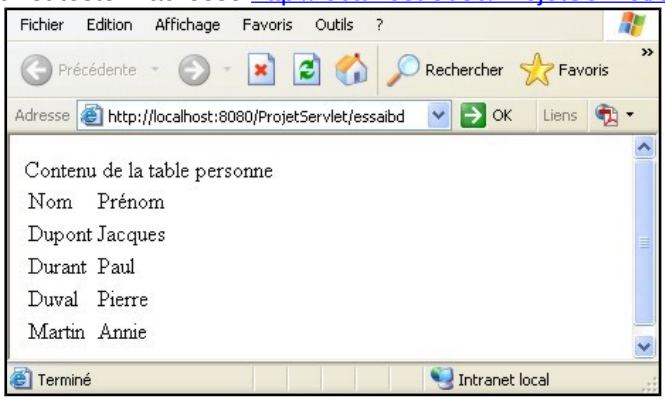
Tout comme pour notre première servlet, nous devons mapper notre servlet pour qu'elle soit reconnue par le serveur Tomcat. Vous devez rajouter les lignes suivantes dans le fichier web.xml dans le répertoire WEB-INF :

```
<servlet>
  <servlet-name>ServletBD</servlet-name>
  <servlet-class>MesServlets.AfficherPersonne</servlet-class>
  <description>Servlet d'essai de la connection BD</description>
</servlet>

<servlet-mapping>
  <servlet-name>ServletBD</servlet-name>
  <url-pattern>/essaibd</url-pattern>
</servlet-mapping>
```

1.10. Exécution de la servlet

- Lancer Tomcat (il faut le démarrer ou le redémarrer pour la prise en compte du pool de connexion).
- Démarrer votre navigateur et testez l'adresse http://localhost:8080/ProjetServlet/essaibd



2. Sérialisation de requêtes

Nous allons étudier ici les possibilités offertes par les servlets en tant que services offerts à un programme java.
Nous allons écrire une application qui permet la recherche de personnes dans la table personne à partir de leur nom.
Nous sommes ici en présence d'une application 3 tiers :

1. Le client est une **applet** qui se chargera de la partie interface graphique avec l'utilisateur (saisie du nom à chercher et affichage des résultats)
2. Côté serveur une **servlet** se chargera via le JDBC de l'exécution de la requête
3. Toujours côté serveur, le dernier tiers est le SGBD mysql

1.11. La sérialisation

La sérialisation des résultats d'une requête

Pour renvoyer le résultat de la requête dans un objet, cet objet doit être sérialisable. On dit qu'un objet java est sérialisable quand il peut être **écrit dans un format flux d'octets** depuis une machine virtuelle Java et **reconstruit sans perte** sur une autre machine virtuelle Java. Or les ResultSet ne sont pas sérialisables. En effet, un ResultSet contient notamment une référence à la connexion à la base de données, ce pointeur, si il est transmis au client, n'aura plus aucun sens…
Il nous faut donc trouver une solution pour transférer les données et uniquement les données du ResultSet vers votre applet. Il 'suffit' de créer un objet semblable au ResultSet mais contenant uniquement les données sérialisables que vous voulez transmettre à l'applet. Ce travail a déjà été réalisé par des programmeurs. Pour cela, vous allez utiliser le package **SResultSet** (Serialised ResultSet).

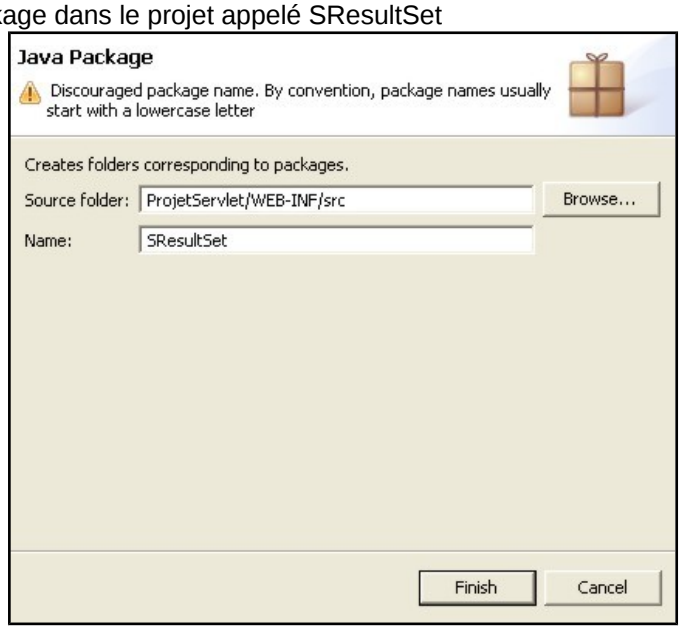
Utilisation du package SResultSet

Vous pouvez récupérer le package SResultSet dans l'archive.
Le package SResultSet contient deux classes :

- XList qui est une hash table qui va contenir les résultats de la requête
- SResultSet qui est la sœur de ResultSet mais contenant uniquement les données sérialisables..

Ce package est utilisé par la servlet le SResultSet pour formater le ResultSet et l'envoyer à l'applet. L'applet va utiliser le SResultSet pour formater l'envoi de la servlet et récupérer les données transférées. Pour ajouter ces classes au projet :

- Créez un nouveau package dans le projet appelé SResultSet



- Importez les classes téléchargées dans le package

Voyons maintenant comment fonctionne cette sérialisation.

1.12. Du côté du client : l'applet

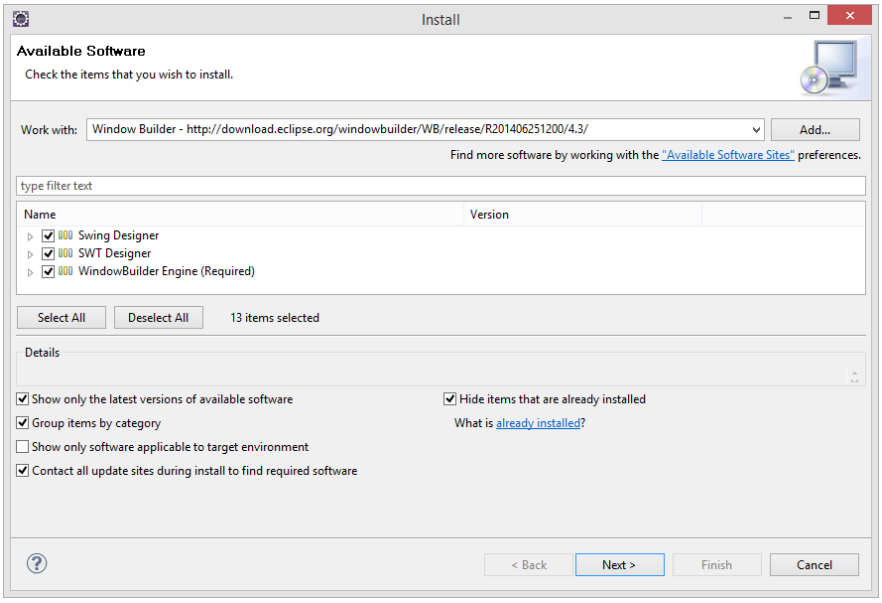
Rappel :

Une applet java est une classe (ou un ensemble de classes) Java qui est compilée côté serveur puis téléchargée sur le poste client pour être exécutée dans le navigateur Web du client. Ce type de code java téléchargé chez le client permet d'alléger le travail du serveur et de proposer à l'utilisateur l'interface riche d'un client lourd. Cette solution a l'inconvénient de consommer du temps de téléchargement si l'applet est volumineuse et de créer des risques pour la sécurité du poste client en cas d'applet malveillante.
Pour fonctionner, il est nécessaire que le navigateur web du client dispose d'un plugin java. L'applet sera référencée comme toute ressource dans le code HTML d'une page web. HTML dispose à cet effet d'une balise <APPLET></APPLET>. Enfin, côté serveur web, l'applet compilée doit être téléchargeable.

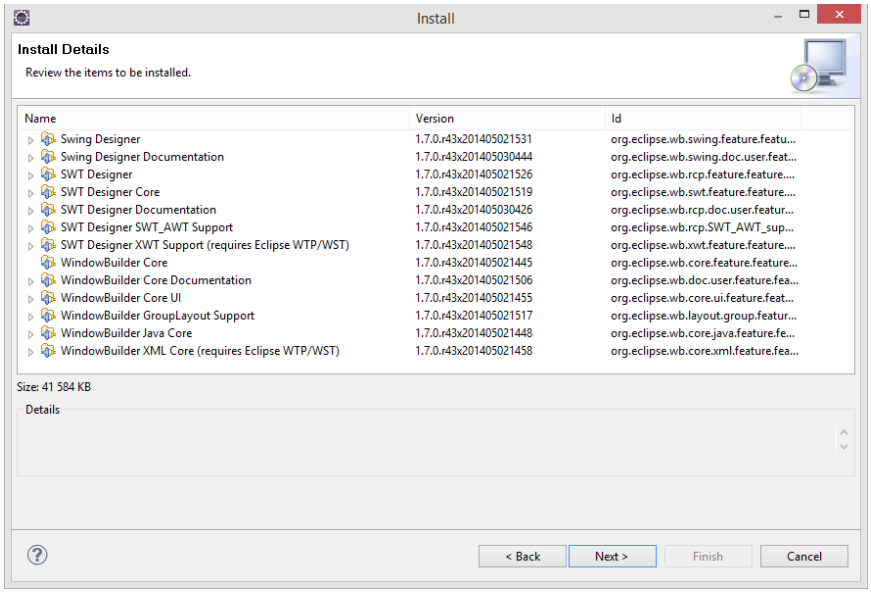
Installation de WindowBuilder

Pour créer des interfaces, vous pouvez travailler graphiquement en utilisant le plug-in WindowBuilder (http://www.eclipse.org/windowbuilder/).

- Dans Eclipse, Menu Help>Install New Software
- Bouton Add et ajouter le site d'upload suivant : http://download.eclipse.org/windowbuilder/WB/release/R201406251200/4.3/
- Faitez attention d'installler le plug-in pour la bonne version d'Eclipse !
- Sélectionnez tout puis Next



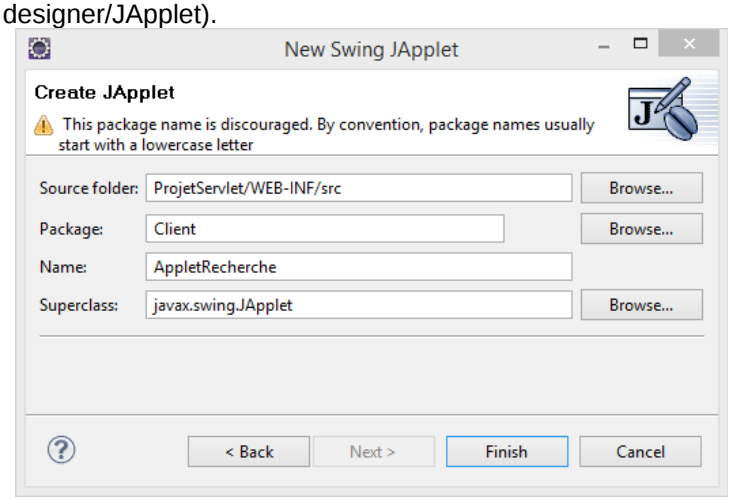
- Next



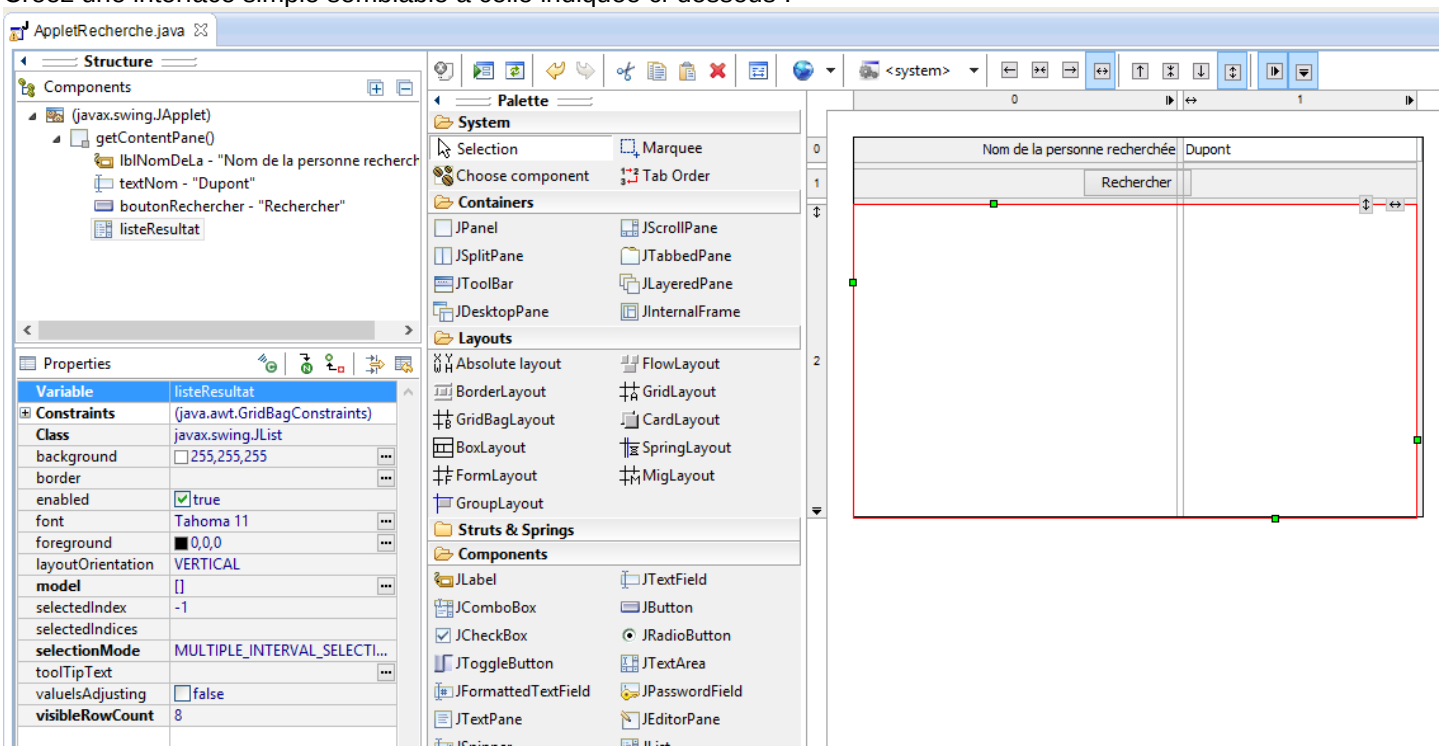
- Accepter les termes de la licence
- WindowBuilder s'installe.
- Redémarrer Eclipse

Création d'une applet

Toujours dans le projet que nous avons créé au début, créez un package pour le côté client et ajoutez une applet (dans la catégorie New->Other/Swing designer/JApplet).



Cliquez sur l'onglet design en bas de la fenêtre de code qui vient de s'ouvrir.
Créez une interface simple semblable à celle indiquée ci-dessous :



Commencez par changer le layout de l'applet et à passer en GridBagLayout (un élément d'interface par case comme un tableau HTML) en exécutant un clic droit dans le fond de l'applet.
Ensuite ajoutez les éléments d'interface suivants :

- Un JLabel pour l'intitulé de la zone de texte
- Une zone de saisie **textNom** (JTextField) permettra à l'utilisateur de saisir le nom de la personne à chercher.
- Le bouton Rechercher **boutonRechercher** (JButton) crée le lien avec la servlet et récupère le résultat de la requête
- Une zone de liste **listeResultat** (JList) permet d'afficher le résultat de la requête

Pensez à ajouter l'événement qui vous permet de gérer le clic sur la souris. Pour cela, faire un clic droit sur le boutonRechercher -> Add event handler ->action ->action performed

Exécution de la servlet et récupération du résultat

Côté client, nous allons :

- nous connecter au serveur,
- envoyer à la servlet le nom de la personne à rechercher,
- récupérer le résultat de la requête dans un sResultSet
- afficher le résultat de la requête

Ajoutez dans l'applet une méthode qui permet d'appeler la servlet.

```
private void Rechercher() {
  String res;
```

```
   String nom=textNom.getText();
   try
   {
    // Connexion à la servlet
    URL url=new URL("http://localhost:8080/ProjetServlet/requete");
    URLConnection connexion=url.openConnection();
    connexion.setDoOutput(true);
    // Récupération du flux de sortie
    ObjectOutputStream fluxsortie = new
ObjectOutputStream(connexion.getOutputStream());
    // Envoi du nom à rechercher
    fluxsortie.writeObject(nom);
    // Récupération du flux d'entrée
    ObjectInputStream fluxentree = new
ObjectInputStream(connexion.getInputStream());
    // Récupération du résultat de la requête
    SerializedResultSet donnees=(SerializedResultSet) fluxentree.readObject();
    // affichage du résultat
    donnees.first();
    Vector contenu=new Vector();
    contenu.clear();
    listeResultat.setListData(contenu);
    for (int i=0; i<donnees.recordCount();i++)
    {
     res=donnees.getString("nom")+" "+donnees.getString("prenom");
     contenu.addElement(res);
     donnees.next();
    }
    if (donnees.recordCount()==0)
     {
     res="Pas de personne correspondante";
     contenu.addElement(res);
     }
    listeResultat.setListData(contenu);

   }
   catch (Exception sql)
   {
    System.out.println("erreur "+sql);
   }

 }
```

Appelez cette méthode depuis l'actionPerformed du boutonRechercher.

1.13. Du côté du serveur : la servlet

Créez et mappez comme nous l'avons vu précédemment un nouvelle servlet que nous allons appeler ServletRequete. Cette servlet est chargée :

- De se connecter à la base de données (via le jndi)
- D'exécuter la requête
- De transformer le ResultSet en sResultSet
- De renvoyer le résultat au client (ici l'applet)

Nous allons ici utiliser le mode de transfert POST pour récupérer le nom de la personne recherchée envoyé par le client. Le code que vous devez obtenir doit être proche du suivant :

```
public class ServletRequete extends HttpServlet {
 private DataSource ds;
 Connection BD;
 String nomPersonne;
 SerializedResultSet sresultat;
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException {
  try {
   // Récupération du flux d'entrée envoyé par l'applet
   ObjectInputStream entree=new ObjectInputStream(request.getInputStream());
   nomPersonne=(String)entree.readObject();
   // Préparation du flux de sortie
   ObjectOutputStream sortie=new ObjectOutputStream(response.getOutputStream());
   // Execution de la requête
```

```
    sresultat=ExecuterRequete();
    sresultat.first();
    // Envoi du résultat au client
    sortie.writeObject(sresultat);
   } catch (Exception ex) {
   System.out.println("Erreur d'exécution de la requête SQL : "+ex);
   }
  }

  public SerializedResultSet ExecuterRequete()
  {
   try
   {
    // Exécution de la requête
    BD=ds.getConnection();
    Statement s = BD.createStatement();
    ResultSet r = s.executeQuery("select * from personne where nom= '"+nomPersonne+"'");
    // Transformation du ResultSet en sResultSet
    java.sql.ResultSetMetaData columnNames = r.getMetaData();
    SResultSet.SerializedResultSet sResultSet = new SResultSet.SerializedResultSet();
   for (int i = 1; i <= columnNames.getColumnCount(); i++) {
    sResultSet.addColumn(columnNames.getColumnName(i), i);
    }
    while (r.next()) {
     for (int column = 1; column <= columnNames.getColumnCount(); column++) {
     sResultSet.addColumnData(column, r.getObject(column));
     }
    }

    r.close();
    s.close();
    BD.close();
    s = null;
    r = null;
    return sResultSet;
   }
   catch (java.sql.SQLException ex) {
      System.out.println("Erreur d'exécution de la requête SQL \n"+ex);
      return null;
   }
  }

  public void init() throws ServletException {
   try {
    Context initCtx = new InitialContext();
    System.out.println("lookup de env");
    Context envCtx = (Context) initCtx.lookup("java:comp/env");
    System.out.println("lookup de base_test");
    ds=(DataSource) envCtx.lookup("base_test");
   }
   catch(Exception er) {
    System.out.println("Erreur de chargement du contexte " + er);
   }
  }
}
```

1.14. Exécution finale

Pour tester votre code il faut :

- Que Tomcat soit démarré, que votre contexte soit chargé et qu'il n'y ait pas d'erreur dans la console
- Que vous démarriez l'applet : Clic droit sur AppletRecherche>Run As> Java Applet

L'applet viewer démarre et simule le chargement de l'applet dans un navigateur Web.