

Atelier de programmation Java EE

Niveau : 2IM

Instructions de départ:

- La partie 2 et la partie 3 sont au choix (Choisissez de faire la partie 2 **ou** la partie 3)
 - o La partie 2 contient JDBC/Servlets/JSP.
 - o La partie 3 contient Sessions/Servlets/JSP.
- La partie 1 contient seulement Servlet/JDBC.

Instructions pour la sauvegarde:

Avant l'envoi des copies sur le serveur:

- Sauvegarder tous les fichiers sur Eclipse.
- Copier votre projet directement à partir d'Eclipse.

Éléments fournis:

Fichier *Bibliothèque.sql* : Ce fichier contient le script SQL permettant de créer et de peupler la base de données.

Projet Eclipse, contenant :

- Classe `com.isamm.Connecteur` : Cette classe contient les attributs et les méthodes nécessaires pour établir une connexion avec une base de données. La classe `Connecteur` propose la méthode statique `getConnection()` qui effectue les tâches suivantes:
 - Chargement du pilote en mémoire,
 - Etablissement d'une connexion avec la base de données,
 - Renvoi de l'objet de type `java.sql.Connection`.

L'utilisation de la classe « `Connecteur` » pour la connexion à la base de données se fait de la manière suivante: `Connection c = Connecteur.getConnection();`

- Classe « `com.isamm.Etudiant` » représentant la table « `etudiant` »
- Classe « `com.isamm.Livre` » représentant la table « `Livre` ».
- Classe « `com.isamm.Emprunt` » (appartenant qu'à la partie 3, **à ne pas utiliser** dans la partie 1, ni dans la partie 2).
- Dossier `/lib` contenant le Pilote JDBC (driver) permettant d'interfacer un programme JAVA avec MySQL. Le pilote a été rajouté dans le `buildPath` et dans le `deployment assembly` (Aucune autre action n'est nécessaire).

Le schéma de la base de données Bibliothèque est le suivant:

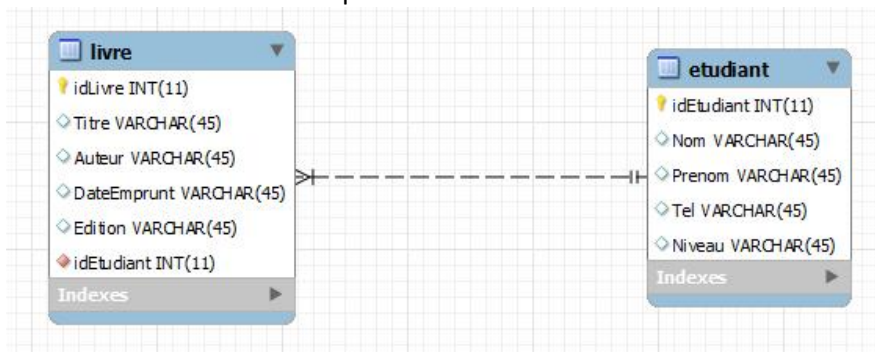


Figure 1 : Schéma de la base de données

Un étudiant peut emprunter plusieurs livres.

Un livre peut être emprunté par un seul étudiant.

« idEtudiant » est une clé étrangère vers l'id de la table « etudiant ».

Travail préliminaire :

- Copier le projet du serveur vers le lecteur disque « D:\ »
- Importer le projet dans votre Eclipse en faisant un clic droit sur la vue « Project Explorer » / « Import » et choisissez : « Existing Projects into Workspace »

Travail demandé: On désire développer une application Web pour la gestion d'une bibliothèque.

Partie 1 : Recherche Livre

1. Créer la page "*RechercheEtudiant.html*" permettant à l'utilisateur de chercher un étudiant dans la base de données. (voir Figure 2).

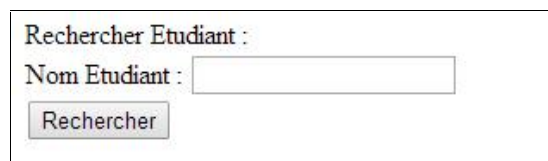


Figure 2 : Page de recherche (*identification.html*)

2. Définir la page "*RechercheEtudiant.html*" comme page d'accueil dans le descripteur de déploiement.
3. Créer une servlet nommée « *recherche* » qui vérifie l'existence de l'étudiant dans la base de données.
4. Si l'étudiant n'existe pas, l'utilisateur est redirigé vers la page "*erreur.html*" que vous créerez aussi dans le projet.

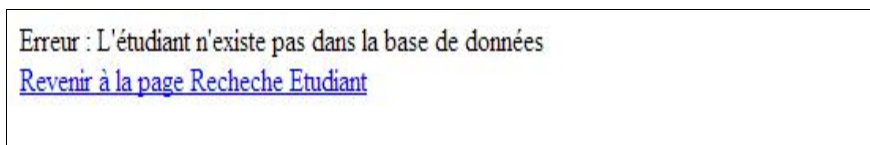


Figure 3 : Vue sur la page *erreur.html*

Si l'étudiant existe :

Créer un objet du type *com.isamm.Etudiant* dans lequel les attributs sont remplis de l'enregistrement de la base de données « etudiant » cherché.

Créer un attribut dans le request dont la valeur est l'étudiant créé.

Rediriger vers la Servlet « *emprunt* » dont les détails sont dans la partie 2 de l'exercice.

Partie 2 [Cette partie contient JDBC/Servlet/JSP] : Consultation des Livres empruntés par l'étudiant

1. Créer une servlet nommée « *ListeLivreEmpruntés* » qui :
 - a. Récupère l'objet Etudiant passé en request dans la servlet « *recherche* ».
 - b. Interroge la base de données pour récupérer la liste des livres empruntés par l'étudiant passé en request :

- Pour chaque Livre reçu, lui créer un objet Livre, remplir ses attributs et le rajouter dans une liste (*java.util.List*) de Livre.
 - c. Crée un attribut dans le request nommé « *Livresempruntés* » dont la valeur est la liste des livres récupérée précédemment.
 - d. Crée un attribut dans le request dont la valeur est l'objet Etudiant créé dans a.
 - e. Redirige vers la page « *ListeEmprunts.jsp* » dont les détails sont dans la section suivante.
2. Créer la page « *ListeEmprunts.jsp* » qui (voir figure 4):
- a. affiche la liste des livres empruntés déjà envoyés par la servlet « *emprunt*».
 - b. affiche le prénom et le nom de l'utilisateur connecté à partir du request.


Bonjour Mohamed Ben Ahmed		Prénom et Nom de l'étudiant, récupérés à partir de request
Liste des livres empruntés :		
Titre	Auteur	Date Emprunt
UML 2 par la pratique	Pascal Roques	01/03/2014
Algorithmes et structures de données génériques	Michel Divay	15/03/2014

Figure 4 : Page *ListeEmprunts.jsp*

Partie 3[Cette partie contient Servlet/JSP/Sessions] : Consultation des Livres empruntés par l'étudiant

1. Créer une servlet nommée « *ListeLivreEmpruntésSession* » qui :
- a. Récupère l'objet Etudiant passé en request dans la servlet « *recherche* ».
 - b. Crée une session et enregistre l'objet utilisateur récupéré précédemment dans cette session.
 - c. Interroge la base de données pour récupérer la liste des livres empruntés par l'étudiant. Récupère la liste des livres reçus à partir de la classe « *Emprunt* »
 - c. Crée un attribut dans le request nommé « *ListeEmprunts* » dont la valeur est la liste des livres récupérés.
 - d. Rediriger vers la page « *LivresEmpruntésSession.jsp* » dont les détails sont dans la section suivante.
2. Créer la page « *LivresEmpruntésSession.jsp* » qui (voir figure 4):
- a. affiche la liste des mails déjà envoyés par la servlet « *ListeMessagesRecusSession* ».
 - b. affiche le prénom et le nom de l'utilisateur connecté à partir de la session.


Bonjour Mohamed Ben Ahmed		Prénom et Nom de l'étudiant connecté, récupérés à partir de la session
Liste des livres empruntés :		
Titre	Auteur	Date Emprunt
UML 2 par la pratique	Pascal Roques	01/03/2014
Algorithmes et structures de données génériques	Michel Divay	15/03/2014

Figure 5 : Page *LivresEmpruntésSession.jsp*

3. Créer la Servlet « *Deconnection* » qui :
- a. Tue la session ouverte.

b. Redirige vers la page « *identification.html* »

Annexe :

- La redirection d'une servlet à une autre page ou servlet se fait par le biais du code suivant :
`getServletContext().getRequestDispatcher("/page.jsp").forward(request,response);`
- L'importation d'une classe dans une page jsp se fait par le biais de la directive page suivante:
`<%@ page import="nomPackage.Classe" %>`

- Définition d'une page d'accueil (page1.html) dans le descripteur de déploiement web.xml :
`<welcome-file-list> <welcome-file>page1.html</welcome-file></welcome-file-list>`

- liste non ordonné en HTML : `item1deuxième item`

- Tableau en HTML `<table border="1">`
`<tr> <td>Row 1, Column 1</td> <td>Row 1, Column 2</td> </tr>`
`<tr> <td>Row 2, Column 1</td> <td>Row 2, Column 2</td> </tr>`
`</table>`

- Utilisation des images : ``
- Manipulation des couleurs du texte:``
- Couleur de fond d'une ligne d'un tableau : `<tr bgcolor="#0099CC"> </tr>`
- Déclaration d'un formulaire `<form method="get" action="ServletTraitement">....</form>`
 - Champs de saisie de texte : `<input type="text" name="prenom">`
 - Zone de texte libre : `<TEXTAREA rows="3" name="commentaire">votre commentaire</TEXTAREA>`
 - Liste :
`<SELECT name="niveau">`
`<OPTION VALUE="2IM" selected="selected">deuxième année IM</OPTION>`
`<OPTION VALUE="3IM">troisième année IM</OPTION>`
`</SELECT>`
 - Boutons radios :
Homme : `<INPUT type="radio" name="sexe" value="M" checked>`
Femme : `<INPUT type="radio" name="sexe" value="F">`
 - Cases à cocher : `<INPUT type="checkbox" name="loisirs" value="lecture">`
`Lecture`
`<INPUT type="checkbox" name="loisirs" value="voyage">Voyage`
 - Bouton d'envoi : `<INPUT type="submit" value="Envoyer">`
- Codage des données par la méthode *get* : la méthode *get* envoie le formulaire dans l'URL comme suit :

`http://urlDeLApplication:port/action?name1=value1&name2=value2&.....&namen=valuen`

- Exemple de génération d'un contenu HTML par une servlet dans la méthode *doGet* ou *doPost*

```
response.setContentType("text/html");
PrintWriter p = response.getWriter();
p.println("<html><head><title>Exemple de
titre</title></head><body><h1>contenu</h1>");
....
```

- Traitement des formulaires : Les paramètres du formulaire sont envoyés dans un objet de type **HttpServletRequest**. Les méthodes de lecture des paramètres:
 - `public String getParameter(String name)` : permet de retourner la valeur d'un champ dont on a passé le nom en argument.
 - `public String[] getParameterValues(String name)` : utilisée pour récupérer la ou les valeurs retournées par un champs à retour multiple (checkbox).

Remarques : Si le champ dont le nom est fourni aux méthodes précédentes n'existe pas, la valeur **null** est retournée.

- La redirection des requêtes : définie à l'aide de l'objet **RequestDispatcher** :

```
RequestDispatcher rd = getServletContext().getRequestDispatcher("/UrlDestination");  
rd.forward(request, response);
```

Il est possible de compacter l'écriture précédente sous la forme :

```
getServletContext().getRequestDispatcher("/UrlDestination").forward(request,  
response);
```

- Utilisation des ArrayList
 - Déclaration et instanciation : `ArrayList<NomDeLaClasse> nomDeLaListe=new ArrayList<NomDeLaClasse>();`
 - Ajout d'un élément dans la liste : `nomDeLaListe.add(objetDeNomDeLaClasse);`
 - Tester si la liste est vide : `nomDeLaListe.isEmpty()`
 - Récupérer l'objet d'indice i de la liste : `nomDeLaListe.get(i)`
 - Supprimer l'objet d'indice i de la liste : `nomDeLaListe.remove(i)`
 - Retourner le nombre d'objets dans la liste : `nomDeLaListe.size()`
- Conversion d'une chaîne en un double : `Double.parseDouble(uneChaineDUnDouble)`
- Conversion d'une chaîne en un int : `Integer.parseInt(uneChaineDUnInt)`

Les pages JSP :

- Déclaration d'une variable globale : `<%! private String message = "Bonjour" ;%>`
- Affichage d'une variable moyennant une expression : `<%= message %>`
- Inclusion d'une page : `<%@ include file="unAutreFichier" %>`
- Déclaration du type du contenu de la page : `<%@page contentType="text/html" %>`
- Importation d'une classe java : `<%@page import="java.util.*" %>`
- Objets prédéfinis dans une page jsp : **request**, **response**, **session** et **out**

```
<h1>Affichage avec des expressions:</h1><br>  
Protocol : <%= request.getProtocol() %><br>  
Scheme : <%= request.getScheme() %><br>  
ServerName:<%= request.getServerName() %><br>  
ServerPort:<%= request.getServerPort() %><br>  
RemoteAddr <%= request.getRemoteAddr() %><br>  
RemoteHost<%= request.getRemoteHost() %><br>  
Method : <%= request.getMethod() %><br>
```

Les sessions :

- **Création de session** : `HttpSession getSession(true)` de la classe `HttpServletRequest`, renvoie la session courante ou une nouvelle session si aucune session n'a été créée lors de la navigation de l'utilisateur sur l'application. Si l'argument « false » a été passée à la méthode `getSession(boolean b)` ou si aucun argument ne lui a été passé, la méthode renvoie null si aucune session n'a été créée.
- **Sauvegarde de paramètres** dans la Session :
 - Envoi de paramètre de type String : `session.setAttribute("nom", "ali");`
 - Envoi de paramètre de type int : `session.setAttribute("age", 18);`
 - Envoi de paramètre de type Personne : `session.setAttribute("personne", new Personne());`
- **Récupération de paramètres** d'un objet de type Session
Exemple de récupération de paramètres dans une page JSP:

```
<% String monNom = (String)session.getAttribute("nom");%>  
<% int monAge = (int) session.getAttribute("age");%>  
<% Personne maPersonne = (Personne)session.getAttribute("personne");%>
```

Les bases de données – JDBC

- choix du driver :

```
Class.forName("com.mysql.jdbc.Driver");
```

- établissement de connexion

```
conn=DriverManager.getConnection("jdbc:mysql://localhost/base_de_produits","iduser","password");
```

- interrogation de la base

```
String requete = "select * from base_de_produits.produit";  
ResultSet res ;  
Statement statement = conn.createStatement() ;  
res = statement.executeQuery(requete);
```

- Exploitation du résultat

```
while (resultSet.next()) {  
    String nom = resultSet.getString("nom");  
    String category = resultSet.getString("category");  
    int quantite = resultSet.getInt("quantite");  
    double prixunitaire = resultSet.getDouble("prixunitaire");  
    System.out.println("nom="+nom+", categorie="+category+", quantite="+quantite+", prix unitaire="+prixunitaire);  
}
```

- Détails sur l'exécution des requêtes

Pour exécuter une requête SQL, l'interface **Statement** dispose de trois méthodes (elles reçoivent toutes les trois la requête en argument) :

- **executeQuery**, s'applique à une requête de sélection et fournit en résultat un objet de type *ResultSet* ;
- **executeUpdate** s'applique à une requête de mise à jour d'une table ou de gestion de la base ; elle fournit en résultat (de type *int*), le nombre d'enregistrements modifiés dans le premier cas ou la valeur -1 dans le second (gestion) ;
- **execute** s'applique à n'importe quelle requête. Elle fournit en résultat un booléen valant *true* si la requête SQL fournit des résultats (sous forme d'un objet de type *ResultSet*) et *false* sinon. Il faut alors, suivant le cas, utiliser l'une des deux méthodes *getResultSet* pour obtenir l'objet résultat ou *getUpdateCount* pour obtenir le nombre d'enregistrements modifiés. Cette dernière méthode s'avère surtout utile lorsque l'on doit exécuter une requête de nature inconnue ; c'est ce qui peut se produire avec un programme qui exécute des requêtes SQL fournies en données, par exemple dans un fichier texte.

Bon travail,