TD 6 : JDBC

Objectifs :

Utilisation, mise en œuvre de **JDBC** pour l'accès à une base Oracle.
Programmation via les méthodes publiques des **interfaces de java.sql**.
Utilisation d'une bibliothèque **driver JDBC** (classes implémentant les interfaces de java.sql)

Enoncé :

L'objectif de ce td est la connexion à une base de données (interrogation, mise-à-jour) depuis un programme Java. Les avions, pilotes, vols, passager, départs et réservations correspondent à une base de données pour laquelle un script Oracle *Avion* est disponible pour créer cette base. L'exécution du script et le contrôle des résultats que vous obtiendrez dans les exercices qui suivent se font à l'aide de la connexion Oracle sqlplus

Tout ce qui sera fait dans ce TD peut être repris tel quel pour une BD de même schéma relationnel Mysql ou autre à condition de disposer du driver adéquat (en général facile à télécharger sur le web).

Les essais que vous réaliserez dans ce td doivent permettre *in fine* de visualiser tous les vols au départ d'une ville donnée à une date donnée (avec les places disponibles).

L'ensemble constitue une application client/serveur 2-tier:
1. sur le serveur (de données): il y a la BD Oracle ;
2. sur le client : votre application Java s'exécutant sous Eclipse et utilisant JDBC.

1er exercice : afficher tous les pilotes (matricule, nom, age) de la base Oracle à l'aide du driver JDBC Oracle ojdbc6.jar

Le fichier ojdbc6.jar correspond au driver JDBC-Oracle compatible avec la version Oracle de l'IUT. Vous ajouterez cette bibliothèque à votre projet Eclipse et invoquerez la classe du driver *oracle.jdbc.OracleDriver* dans Class.forName. Cette bibliothèque se trouve sur le serveur commun étudiant. L'url de connexion pour ce driver est "jdbc:oracle:thin:@localhost:1521:XE".

2ème exercice : requête avec paramètre pour la sélection de tuples (WHERE)

On veut afficher tous les vols (*numvol*) partant d'une ville donnée, la ville étant saisie au clavier.
Vous effectuerez cette exercice avec une Statement en concaténant la ville saisie à la chaine paramètre de la méthode executeQuery (sans oublier d'ajouter les quotes encadrant un littéral VARCHAR en langage SQL).

3ème exercice : requête pré-compilée avec arguments (PreparedStatement)
Il est clair que la manœuvre de concaténation deviendra vite fastidieuse et non portable pour rechercher des dates. Les PreparedStatement permettent de répondre à ces problématiques.
Vous allez donc maintenant, en gardant l'objectif d'affichage pour une ville donnée, passer par une PreparedStatement avec un argument (la ville) qui sera affectée à l'aide d'un setString précédant l'executeQuery.

Et ensuite :
Ceci fait, il ne reste plus qu'à voir dans la javadoc la classe java.sql.Date et son fonctionnement pour atteindre l'objectif cité au début de cet énoncé (en révisant un peu java.util.GregorianCalendar). Notez qu'un encadrement de date sera nécessaire pour extraire les vols à une date donnée (si on veut tous les vols au 14/07/2014, il faudra créer 2 objets java.sql.Date, l'un au 14/07/2014, l'autre au 15/07/2014 et exécuter une requête cherchant les vols entre ces 2 dates. En effet, le champ heuredepart SQL d'un vol contient la date et l'heure et une requète SQL testant l'égalité ne pourrait sélectionner que les vols partant à 00:00:00h de la java.sql.Date créée à partir des données saisies.