Rappel JDBC

- Accès Java à une base de données
- Etapes :

1. **Connexion à la BD**
2. **Exécution d'une requête**
3. **Parcours du résultat**
4. **Clôture de la connexion**

Où et quand créer la connexion ?

Création de la connexion JDBC

- Première solution :

  À chaque requête nécessaire

```
protected bool isLoginValid(String login, String pwd) {
   try (Connection conn = DriverManager.getConnection(…) {
      …
      conn.close();
   } catch (SQLException e) {…}
}
```

- Problème :

  `getConnection()` : opération coûteuse

Création de la connexion JDBC (suite)

- Autres solutions naïves :
  - Conserver la connexion dans un attribut du servlet
  - Utilisation de méthodes statiques

```
public class DatabaseConnection {

    private static Connection conn = null;

    public static Connection getConnection() {
        if (conn == null) conn = DriverManager.getConnection(…);
        return conn;
    }
}
```

- Appli Web = multi-utilisateurs
  - JDBC déconseille le partage de connexions entre threads
  - Si chaque thread attend que l'autre ait fini pour utiliser la connexion (avec `synchronized p.ex.`), perte de réactivité

Création de la connexion JDBC (suite)

- Solution : pool de connexions
  - Le serveur ouvre plusieurs connexions simultanément pour les threads s'exécutant en parallèle
  - Mais une fois qu'un thread a fini, sa connexion peut être réutilisée par d'autres

Le serveur tomcat intègre un tel pool

- Ce pool est représenté par un objet de classe `javax.sql.DataSource`
- Il fournit une connexion active sur demande
- La connexion doit être rendue en fin de traitement
- Il gère la liste de connexions actives :
  - En crée de nouvelles si nécessaire
  - Ferme celles inactives depuis trop longtemps
- Configurable : nombre max. de connexions, timeout, etc.

Configuration d'un pool (Datasource)

- On indique au serveur les informations de connexion dans un fichier de configuration (context.xml)
  - Avantage : si ces informations changent, code source inchangé, un seul fichier à modifier
- Le pool est alors mis en place lors du déploiement
- Pour obtenir la référence du pool dans le code, on utilise un annuaire JNDI (Java Naming and Directory Interface)
  - On peut utiliser des appels de méthodes de l'API JNDI
  - Ou on peut utiliser des annotations java dans le code :

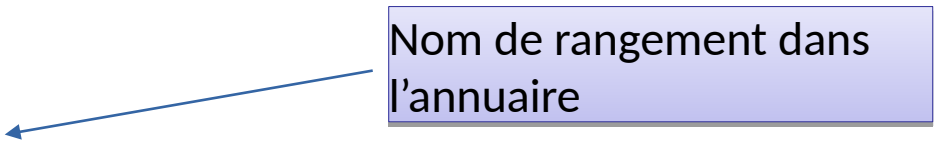
```
@Resource(name = "ref. JNDI du pool")
private DataSource ds;
```

L'attribut ds sera alors initialisé automatiquement par le serveur avec la référence du pool.

Configuration d'un pool (Datasource)

context.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/bibliotheque">
  <Resource
  auth="Container"
  type="javax.sql.DataSource"
  driverClassName="oracle.jdbc.OracleDriver"
  url="jdbc:oracle:thin:@oracle1.ensimag.fr:1521:oracle1"
  maxTotal="2"
  maxIdle="1"
  maxWaitMillis="20000"
  name="jdbc/bibliotheque"
  password="XXXX"
  username="XXXX"
  />
</Context>
```



Configuration d'un pool (Datasource)

CheckUser.java :

```
@WebServlet(…)
public class CheckUser extends HttpServlet {

  @Resource(name="jdbc/bibliotheque")
  private DataSource ds;
  …
  boolean isLoginValid (String login, String password) {
      try (Connection conn = ds.getConnection()) {

        Statement st = conn.createStatement();
        requeteSQL = "…";
        …
      } catch (SQLException e) {
        …
      }
    …
  }
  …
}
```

Obtention d'une nouvelle connexion du pool

Restitution automatique de la connexion au pool à la sortie du bloc try (qu'il y ait eu une exception ou non)

Restitution explicite : conn.close()