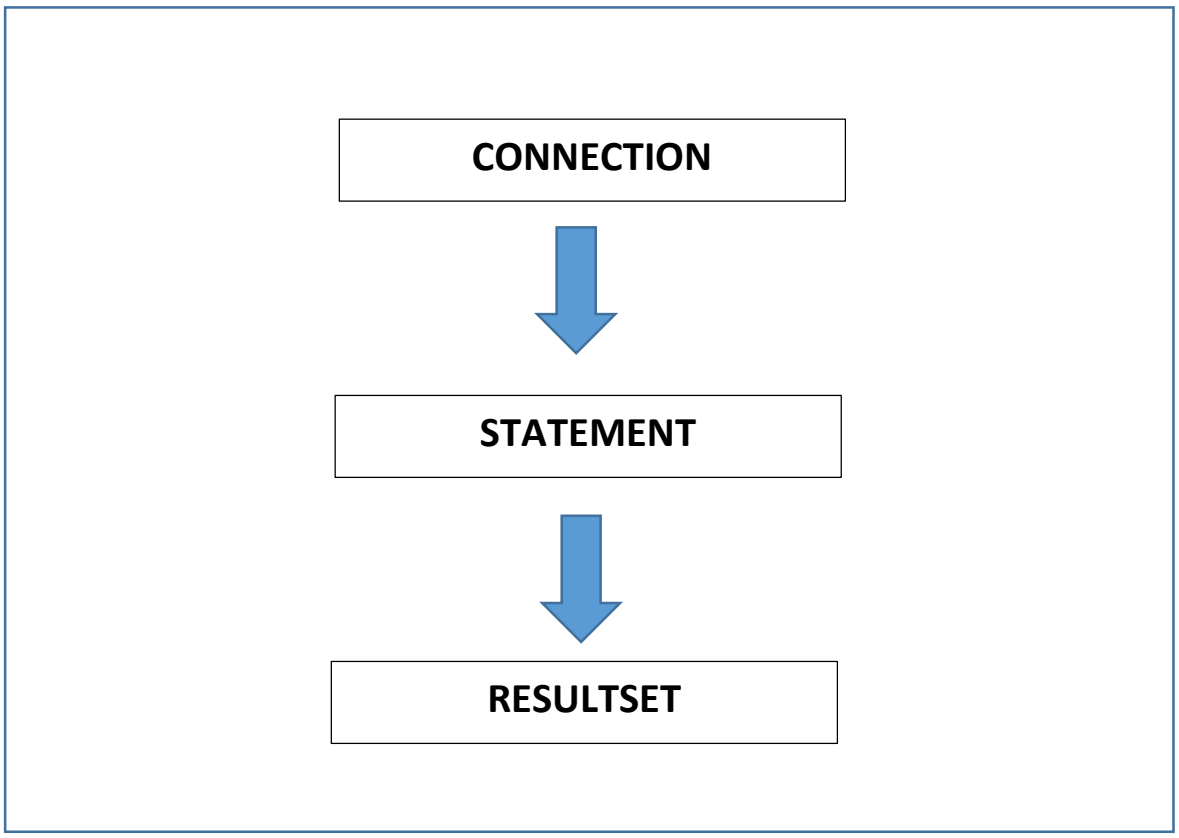
Éléments JDBC

Base de la base

- Le point d'entrée pour utiliser JDBC est la **Connection** (qui est une interface). La Connection est ce que vous récupérez une fois connecté à la BDD.
- A partir de la **Connection**, vous récupérez un **Statement** qui vous permet d'effectuer des requêtes SQL.
- Une fois la requête effectuée via le **Statement** vous récupérez un **ResultSet**, qui contient le résultat de la requête.



Initialisation

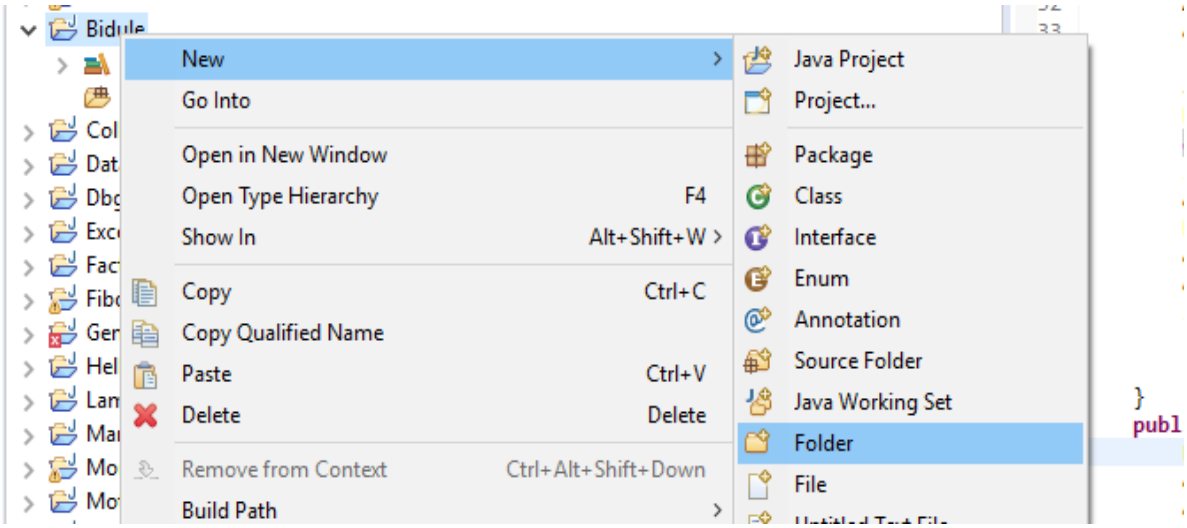
Pour utiliser JDBC, il faut un driver.

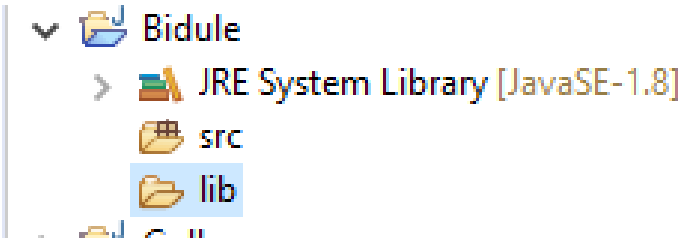
- C'est quoi un driver ?
  - C'est quelque chose qui permet d'accéder à une BDD donnée (Mysql, PostGre , etc). En gros, cela permet de transformer votre programme Java en client SQL.
- Sous quelle forme ça se présente ?
  - Sous forme d'un jar ou d'un zip.

- C’est quoi un jar ?
    - Une sorte de zip.
- Ca contient quoi un jar ?
    - Des classes java (compilées, donc) qui permettent de gérer le JDBC pour la BDD considérée.
- Comment ça s’utilise ?
    - On y vient.

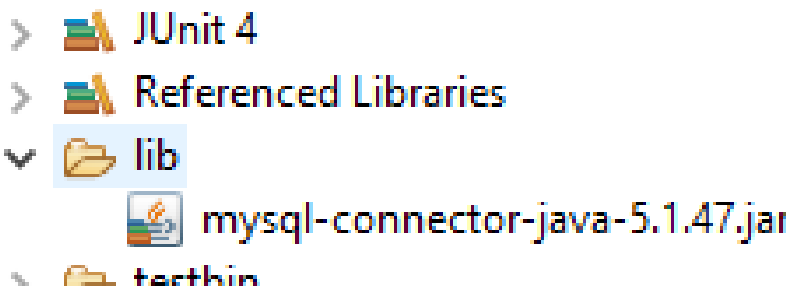
On va prendre l’exemple d’Eclipse.

Il faut d’abord créer un répertoire **lib** (par ex.) dans votre projet.



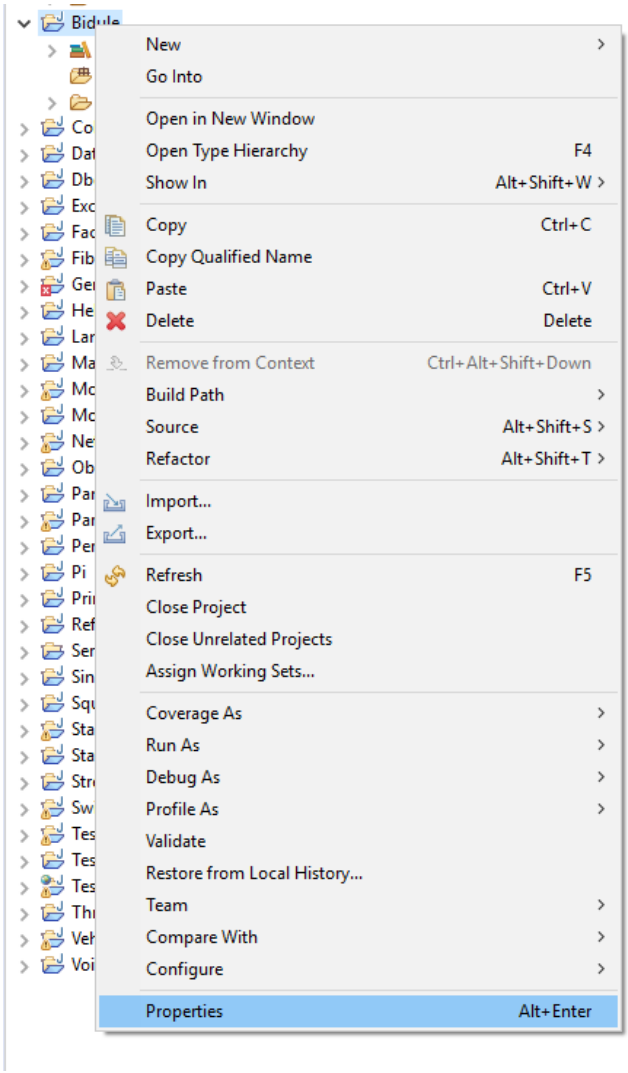


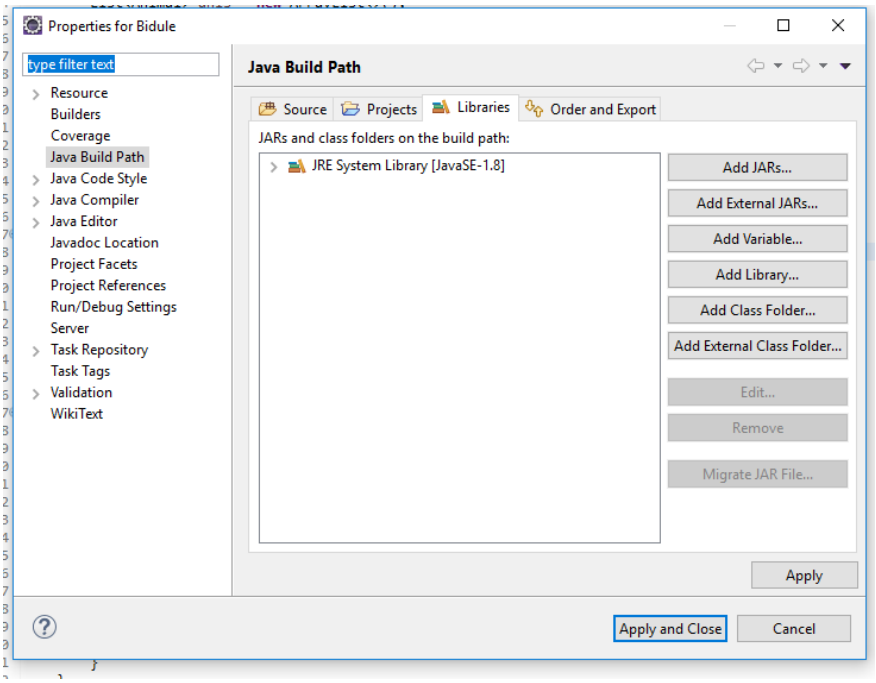
Ensuite, on recopie le jar ou le zip dans le répertoire **lib**.



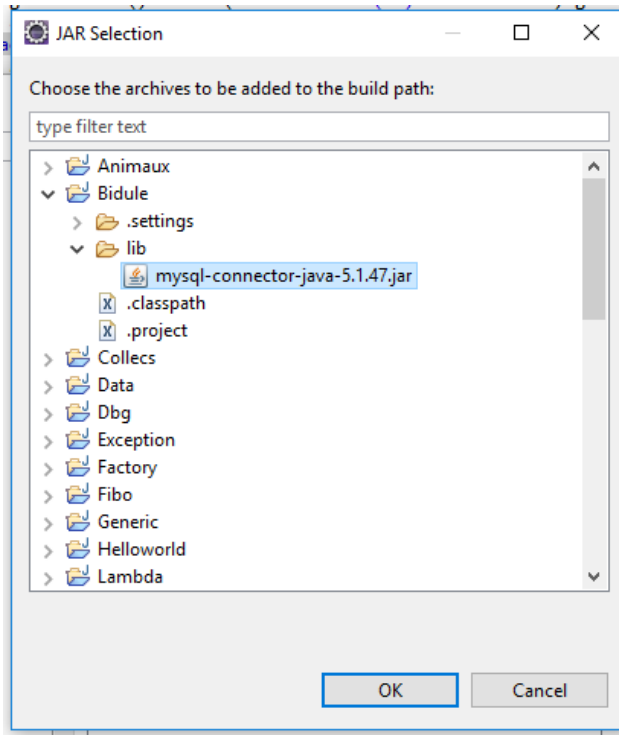
Après il faut dire que le projet utilise ce jar.

Pour ça, il faut faire click droit sur le projet, et choisir **Properties/Java build path/Libraries**





On click sur **Add jar …** et on choisit le jar ou zip qui est dans lib.



On fait **OK** puis **Apply and Close** et c'est fini.

Initialisation logicielle

Dans le code Java, il y a une petite opération à faire avant d'utiliser le JDBC. Cette fois, le programme dit qu'il va utiliser le driver (le jar donc).

Il faut faire :

```
Class.forName(driverName);
```

Il y a en fait plusieurs façons de faire, mais celle-ci convient tout à fait.

Pour une base MySql, le nom du driver est typiquement : `com.mysql.jdbc.Driver`

Il faut donc taper :

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

Création de la Connection

Pour créer une Connection, on fait :

```
Connection cnx = DriverManager.getConnection(url, user, password);
```

L'url est l'adresse du serveur de BDD. Elle est de la forme :

jdbc:type_sql://nom_domaine_ou_adresse_IP_serveur_BDD:port_serveur/nom_base

Par exemple, pour une BDD MySql, ce sera quelque chose comme ça :

jdbc:mysql://localhost:3306/Bestiole

user est le nom du user associé à la base et **password** son mot de passe.

Au final, on a quelque chose comme ça :

```
Connection cnx =
DriverManager.getConnection("jdbc:mysql://localhost:3306/Bestiole",
"root","root");
```

ATTENTION : pour que ce soit clair, on a utilisé **root** dans l'exemple, mais en développement, on ne doit pas utiliser **root**.

ATTENTION (BIS) : **TOUTES** les objets JDBC et leurs méthodes génèrent des exception. N'oubliez pas d'utiliser des try/catch/finally

Une Connection **DOIT** toujours être fermée après utilisation via la fonction **close()**. On a donc quelque chose du genre :

```
Connection cnx = null ;
try {
   cnx = DriverManager.getConnection("jdbc:mysql://localhost:3306/Bestiole",
         "root","root");
   …
   …
   …
} catch (Exception ex) {
….
….
} finally {
    cnx.close() ;
}
```

Récupération du Statement

Grace à la **Connection**, on peut créer un **Statement** via la fonction **createStatement()** de **Connection**.

Comme ceci :

```
Statement st = cnx.createStatement();
```

Une fois que vous avez un Statement, vous pouvez exécuter une requête SQL sur le serveur de la BDD.

Il y a deux fonctions pour exécuter une requête SQL :

- Pour un **SELECT**, on emploie **executeQuery(String sql)**
- Sinon, on emploie **execute(String SQL)**

Exemples :

```
Statement st = cnx.createStatement();
               st.execute("INSERT INTO SPECIE (COMMON_NAME,LATIN_NAME) values
                         ('LOUP','LUPUS')");


Statement st = cnx.createStatement();
               st.executeQuery("SELECT * FROM SPECIE") ;
```

Mais ….
Tout n'est pas si simple : dans le cas d'un **INSERT**, il peut être intéressant de récupérer l'id de l'enregistrement nouvellement créé. Dans ce cas, il faut préciser que le Statement renvoie cet id.
Ca donne :

```
Statement st = cnx.createStatement();
               st.execute("INSERT INTO SPECIE (COMMON_NAME,LATIN_NAME) values
                         ('LOUP','LUPUS')", Statement.RETURN_GENERATED_KEYS);
```

Le ResultSet

Dans le cas d'un SELECT ou dans le cas d'un INSERT (si on veut récupérer le nouvel id), il faut pouvoir récupérer le résultat de la requête.

Les données sont donc récupérées dans un **ResultSet**.

- Cas du **SELECT**. C'est tout simplement :

```
Statement st = cnx.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM SPECIE");
```

- Cas du **INSERT**. Il faut faire :

```
Statement st = cnx.createStatement();
st.execute("INSERT INTO SPECIE (COMMON_NAME,LATIN_NAME) values
                        ('LOUP','LUPUS')", Statement.RETURN_GENERATED_KEYS);
ResultSet rs = st.getGeneratedKeys();
```

Le **ResultSet** est une sorte de liste qu'il faut parcourir dans une boucle. Par exemple :

```
ResultSet rs = st.executeQuery("SELECT * FROM SPECIE");
while(rs.next()) {
…
}
```

La fonction **next()** renvoie **true** tant qu'il y a quelque chose dans le ResultSet. On boucle donc tant qu'il y a quelque chose dans le ResultSet.

A chaque itération de la boucle, le ResultSet contient un enregistrement. Il faut donc pouvoir récupérer les différents champs de cet enregistrement.

On peut récupérer ces champs via les fonctions **getString(), getInt() , getLong(), getDate()**,etc …

La syntaxe générale est la suivante : **getXXX(nom_du_champ_dans_la_table)**

Exemple :

```
Statement st = cnx.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM SPECIE");
while(rs.next()) {
	//on sait que l'ID est un long
	long id = rs.getLong("ID");
	// on sait que c'est un VARCHAR donc String
	String common = rs.getString("COMMON_NAME");
	// on sait que c'est un VARCHAR donc String
	String latin = rs.getString("LATIN_NAME");

	System.out.println("id = "+id+" common = "+common+" latin = "+latin);
}
```

Sortie :

```
id = 1 common = Chat latin = Felis silvestris catus
id = 2 common = Chien latin = Canis lupus familiaris
id = 3 common = Lapin latin = Oryctolagus cuniculus
```

- Cas du **INSERT**

```
st.execute("INSERT INTO SPECIE (COMMON_NAME,LATIN_NAME) values ('LOUP','LUPUS')",
Statement.RETURN_GENERATED_KEYS);
ResultSet rs = st.getGeneratedKeys();
while(rs.next()) {
	long id = rs.getLong("GENERATED_KEY");
	System.out.println("new key for LOUP is "+id);
}
```

Sortie :

```
new key for LOUP is 5
```

ATTENTION : si on a une requête du genre **`SELECT s.common_name FROM SPECIE as s`**, il faudra faire quelque chose du genre **`st.getString("s.common_name")`** pour récupérer le champ.

PreparedStatement

Pour des raisons de sécurité et de règles d'écriture, il faut en fait employer les **PreparedStatement.**

Les **PreparedStatement** sont identiques aux **Statement**, à ceci près que les requêtes SQL sont directement associées au **PreparedStatement**.
On utilise la fonction **prepareStatement()** de Connection pour obtenir un PreparedStatement.

Exemple :

```
PreparedStatement st = cnx.prepareStatement("SELECT * FROM SPECIE");
```

Dans ce cas, évidemment les execute() et executeQuery() n'ont pas de paramètres.

```
PreparedStatement st = cnx.prepareStatement("SELECT * FROM SPECIE");
ResultSet rs = st.executeQuery();
while(rs.next()) {
	System.out.println("common = "+rs.getString("COMMON_NAME"));
}
```

Ces requêtes SQL peuvent accepter des paramètres sur le modèle :

```
INSERT INTO SPECIE (COMMON_NAME,LATIN_NAME) values (?,?)
```

On voit ici qu'il y a 2 paramètres identifiés par des **?**

Pour mettre à jour les valeurs des paramètres on utilise les fonctions **setString(), SetLong(), SetInt(), SetDate(),** etc de **PreparedStatement** .
Les fonctions **setXXX()** sont de la forme : **setXXX(index_du_paramètre,valeur_du_paramètre)**

ATTENTION : les paramètres commencent à 1 et pas à zéro.

Exemple :

```
PreparedStatement st = cnx.prepareStatement("INSERT INTO SPECIE
(COMMON_NAME,LATIN_NAME) values (?,?)",PreparedStatement.RETURN_GENERATED_KEYS);
st.setString(1, "Baleine");
st.setString(2, "Balenus balenus");
st.execute();
ResultSet rs = st.getGeneratedKeys();
while(rs.next()) {
	long id = rs.getLong("GENERATED_KEY");
	System.out.println("new key is "+id);
}
```

ATTENTION : le ***RETURN_GENERATED_KEYS*** est au niveau du **preparedStatement()** et pas de **execute()**