Java et les bases de données

L3Pro SCT – Bases de données et programmation

Mathieu Sassolas

IUT de Sénart Fontainebleau
Département Informatique

Année 2015-2016
Cours 5

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
IUT SENART FONTAINEBLEAU

Plan de la séance

1. Java, la suite
   - Importer du code existant
   - Les exceptions
2. SQL et Java
3. TD/TP

Plan de la séance

1. Java, la suite
   - Importer du code existant
   - Les exceptions
2. SQL et Java
3. TD/TP

La structure des packages

- Pour réutiliser du code déjà codé par d'autres.
- Structure en paquetage (*package*) : un dossier du système de fichier est un packetage.
- Le compilateur (`javac`) et la machine virtuelle (`java`) vont toujours chercher des classes « standard » dans un dossier défini lors de l'installation, ainsi que le dossier courant.
- On peut si besoin lui demander d'aller en chercher ailleurs en spécifiant le `classpath` :

```
javac -classpath .:../MesAutres/Packages
                                    ClasseLocale.java
```

(On peut modifier la variable d'environnement si on se sert souvent de ces chemins.)

Importer des packages
Pourquoi ?

- Pour utiliser une méthode d'un autre paquetage, il faut donner tout le chemin depuis le classpath.
- Exemple court : `Journee.bissextile()` pour utiliser la fonction `bissextile` de la classe `Journee`.
- Exemple long : on a mis notre projet de calendrier dans un dossier `MonCalendrier`, on souhaite depuis l'extérieur appeler la fonction `bissextile` : `MonCalendrier.Journee.bissextile()`
- Si on l'utilise souvent : on veut lui dire une seule fois où chercher.

Attention !

Cela cache le fait que certaines fonctions sont définies hors du fichier courant et peut compliquer un débugage ultérieur !

Importer des packages
Comment ?

Où ? Au début du fichier `.java` (avant le `public class ...`!) :

Import de classe

```
import MonCalendrier.Journee;
...
    Journee.bissextile(annee);
```

Import de fonctions

```
import MonCalendrier.Calendrier.jolieDate;
import MonCalendrier.Journee.*;
...
    bissextile(annee);
...
    jolieDate(s);
```


Qu'est-ce qu'une exception ?

- Une exception est une erreur qui interrompt le fonctionnement de la fonction
- Elles sont très utilisées lors de l'interaction d'un programme avec le reste du système (lecture de fichier, connexion à une base de donnée)
- Elles sont souvent accompagnées d'un message d'erreur.
- On peut les attraper pour que l'erreur ne se propage pas dans le reste du code.
- On doit déclarer dans la signature de la fonction toute exception lancée ou qui n'aurait pas été attrapée.

Lancer une exception


```
public static String toutesLettres(int i)
                                throws Exception {
    if (i<0) {
      throw (new Exception("<0"));
    }
    else {...}
}
```

Attraper une exception

```
try {
  // Du code qui lance peut
  // être une Exception:
  toutesLettres(i-42)
}
catch (Exception err) {
  System.out.prinln("Il y a eu un problème: "
                                +e.getMessage());
  // Que fait on en cas d'erreur?
}
```

Java refuse de compiler si on appelle une fonction qui lance une exception et qu'on ne l'attrape pas ni ne la déclare.

Exemple : lire des entrées

Le `main` de GeometryForNoobs

```
public static void main (String[] args) {
  if (args.length != 4) {usage();}
  else {
    try {
      float x1 =  Float.parseFloat(args[0]);
      float x2 =  Float.parseFloat(args[1]);
      float y1 =  Float.parseFloat(args[2]);
      float y2 =  Float.parseFloat(args[3]);
      System.out.println("==============...=");
      System.out.println("       Bienvenue...");
      calcul(x1,y1,x2,y2);
      System.out.println("        Merci...");
      System.out.println("==============...=");
    } catch (NumberFormatException e) {usage();}
  }
}
```


Plan de la séance

Globalement

- Java fournit des classes permettant modéliser un dialogue avec une base de données.
- Tout est dans le package `java.sql` (à importer, donc !).
- Il faut un pilote (en l'occurence JDBC) pour que Java puisse véritablement se connecter à une base de donnée (ici Postgres).
- La connexion est un peu technique donc elle sera un peu cachée dans une classe `ConnexionMerit` fournie.

Quelle est l'interface ?

- Après connexion, on peut créer des `Statements`, qui sont à voir comme un invité de commande une fois que l'on a lancé `psql merit`.
- On ne peut lui passer que du SQL (pas `\i` ou `\dt`...).
- On ne met pas le ; en fin de SQL.
- On peut avoir plusieurs `Statement` si on veut accéder à une base de diverses manières ; pour ce qu'on fera, deux suffisent
- Les opération principales d'un `Statement` : `executeQuery` et `executeUpdate`.
- Les deux lancent (potentiellement) des `SQLException` qu'il faut attraper (ou déclarer).

Obtenir un `Statement`

Attention !

Ceci est la manière qui utilise la classe `ConnexionMerit` qui cache des détails techniques.

```
ConnexionMerit db =
        ConnexionMerit.obtain("merit","etudiant");
Statement st = db.getStatement();
Statement st2 = db.getStatementBis();
```


`executeQuery`

Utile pour les `SELECT` car retourne un ensemble de résultats sous la forme d'un objet `ResultSet`.

`ResultSet`

- Une sorte de tableau avec autant de colonnes que demandées (attention donc au `SELECT *`) et une ligne par tuple retourné.
- Lisible une seule fois (« itérateur ») :
  - on démarre avec un curseur au dessus de la première ligne ;
  - `next()` avance à la ligne suivante (retourne `false` si il n'y en a pas) ;
  - On peut aller chercher le contenu de la i-ème colonne de la ligne courante avec get<Type>(i) : `getString`, `getInt`, `getDate`...
  - Cf. documentation de `ResultSet`.

`executeQuery`

Utile pour les `SELECT` car retourne un ensemble de résultats sous la forme d'un objet `ResultSet`.

- On ne connaît pas à l'avance le nombre de lignes de ce tableau.
- On ne peut pas utiliser le `Statement` qui l'a produit tant que l'on a pas fini de le lire (dualement: réutiliser le Statement « efface » le `ResultSet`).

en a pas) ;
  - On peut aller chercher le contenu de la i-ème colonne de la ligne courante avec get<Type>(i) : `getString`, `getInt`, `getDate`...
  - Cf. documentation de `ResultSet`.

Exemple : affichage de la liste des employés

```
String req =
    "SELECT firstname,lastname FROM employees";
ResultSet rs = st.executeQuery(req);
while (rs.next()) {
  System.out.println(rs.getString(1)+" "
                     +rs.getString(2));
}
rs.close(); // C'est plus propre.
```


executeUpdate

But : modification sans retour de la part de Postgres

- Exécute la requête mais ne renvoie qu'un entier : le nombre de mises à jour faites.
- Peut également être utilisée pour un `INSERT INTO`.
- Peut aussi servir pour des `ALTER TABLE` ou `CREATE TABLE`, mais le code Java c'est que rarement l'endroit pour le faire !

Mise à jour de la table `employees`

```
String maj = "UPDATE employees SET
           lastname = 'Sassolas'
     WHERE lastname = 'Patterson'"
int nbUp = st.executeUpdate(maj);
System.out.println(nbUp+" màj effectuées");
```


Plan de la séance

Fin du cours

↳ C'est l'heure du TP ↵