Stockage de données en Java

Fichiers en Java

INTRODUCTION AUX INPUT/OUTPUT

- Le package **java.io** contient les classes de base pour gérer les flux de données, de type caractère ou binaire, et les fichiers.
- Il a été étendu en Java 4 par le package **java.nio**, aussi appelé API NIO, et en Java 7 par l'API NIO2.
- L'un des points délicats est **la portabilité:** une application Java doit pouvoir tourner sur n'importe quelle machine, sans recompilation. Il suffit de constater simplement les différences entre l'écriture des chemins et noms de fichiers entre les systèmes Unix et Windows pour entrevoir l'étendue du problème.

Notion de fichier

- Un fichier en Java est un objet instance de la classe **File** (ou une classe implémentant **Path** à partir de Java 7)
- Une instance de File ou d'une classe implémentant Path ne permet pas d'écrire ou de lire le contenu d'un fichier réel, mais bien de manipuler ce fichier en tant que tel.
- La classe File et l'interface Path définissent la notion **de chemin** dans un système de fichiers. Ce chemin se compose de:

✓ Un préfixe qui dépend du système d'exploitation: slash sous Unix (/), une lettre de lecteur sous Windows (D:) ou un double anti-slash pour les éléments de réseau Windows (\\, qui s'écrit en fait \\\\ du fait qu'il faut échapper l'anti-slash dans le code Java).

- Une séquence de chaînes de caractères (peut être vide ou un seul élément ou plusieurs). Ces éléments constituent les noms associés à ce fichier. Cette séquence représente le chemin vers ce fichier.
- un caractère de séparation qui dépend du système d'exploitation. Pour cela, il est défini dans un champ public statique de la classe File (File.separator). La classe implémentant Path le détecte automatiquement (plus portative)
- Pour accéder au répertoire courant dans lequel s'exécute l'application, on utilise une propriété système de la machine Java : user.dir

System.getProperty("user.dir")

- **voir deux premières instructions dans « fichier1 » sous NetBeans IDE**
- **Voir constructeurs de File et Path dans « fichier1 » sous NetBeans IDE**

Méthodes de File et Path

- Surchage de equals(Object) et toString()
- File et Path implémentent Comparable: l'ordre choisi est simplement l'ordre lexicographique du nom du fichier ou du répertoire représenté. On prendra garde que sous Unix les différences entre majuscules et minuscules sont supportées, ce qui n'est pas le cas sous Windows:

Pour File: int compareTo(File)

Pour Path: int compareTo(Path)

	Méthodes de File	
boolean	isDirectory() isFile()	Tests whether the file denoted by this abstract pathname is a directory (normal file)
boolean	exists() canRead() canWrite() canExecute() isHidden()	Differents tests
void	setReadable(boolean) setWritable(boolean) setExecutable(boolean)	Differents sets
long	length()	Returns the length of the file denoted by this abstract pathname.

	Méthodes de File	
String	getName()	Returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
String	getPath()	Converts this abstract pathname into a pathname string.
String	getAbsolutePath()	Returns the absolute pathname string of this abstract pathname.

	Méthodes de Path	
Path	getName(int)	Returns a name element of this path as a Path object.
int	getNameCount()	Returns the number of name elements in the path
Path	getParent()	Returns the *parent path*, or null if this path does not have a parent
Path	getRoot()	Returns the root component of this path as a Path object, or null if this path does not have a root component.
boolean	isAbsolute()	Tells whether or not this path is absolute.

Comment réaliser les autres méthodes de File avec Path ?

Classe Files

- Paths est une classe finale qui ne dispose d'une méthode get qui permet de créer une classe implémentant Path
- Files aussi une classe finale qui manipule des classes implémentant Path:

	Méthodes statiques de Files	
boolean	isDirectory(Path)	Tests whether a file is a directory
boolean	exists(Path) isReadable(Path) isWritable(Path) isExecutable(Path) isHidden(Path)	Differents tests
long	size()	Returns the size of a file (in bytes).

Il existe une méthode qu'on ne va pas la détailler pour donner les permissions (faute de temps)

setPosixFilePermissions(**Path**, **Set**<**PosixFilePermission**>)

Création/ suppression de fichier (spécifique classe File)

- createNewFile() : demande la création de ce fichier au système de fichier. Cette création ne peut se faire que si le fichier à créer n'existe pas déjà. Si la création n'a pu avoir lieu alors cette méthode retourne false. Si une erreur a été rencontrée, alors la méthode jette une **IOException**.
- delete() : demande l'effacement de ce fichier ou répertoire. Retourne false si cet effacement n'a pu avoir lieu.
- mkdirs(): crée le répertoire représenté par cette instance de File.
- deleteOnExit() : demande à la machine Java d'effacer automatiquement ce fichier ou ce répertoire quand l'application se termine.

Création/ suppression de fichier (méthodes statiques de Files)

	Méthodes	
Path	createFile(Path)	Creates a new and empty file, failing if the file already exists
Path	createDirectory(Path)	Creates a new directory.
Path	createLink(Path,Path)	Creates a new link (directory entry: first attribute) for an existing file *(optional operation)*
void	delete(Path)	Deletes a file
void	deleteIfExists(Path)	Deletes a file if it exists.

CONTENU D'UN RÉPERTOIRE

- La classe File expose quelques méthodes utiles pour lire le contenu d'un répertoire et non pas un fichier.
- ✓ list() : retournent tous les fichiers et répertoires de ce répertoire (sous forme de String)
- ✓ listeFiles() : retournent tous les fichiers et les sous-répertoires directs de ce répertoire.
- ✓ listFiles(FileFilter) retournent to retournent tous les fichiers et les sous-répertoires directs de ce répertoire, qui satisfont le filtre passé en paramètre. FileFilter est une interface fonctionnelle dont la méthode est:

void accept(File)

- ❖ L'interface Path dispose de :

Iterator <Path> iterator()

❖ **voir projet « fichier2 » sous NetBeans IDE**

Il existe des méthodes qu'on ne va pas la détailler pour parcrourir les répertoires avec Files et utilisant Path (faute de temps):

✓ public static Path walkFileTree(Path start, FileVisitor<? super Path> visitor) throws IOException

✓public static Path walkFileTree(Path start, Set<FileVisitOption> options, int maxDepth, FileVisitor<? super Path> visitor) throws IOException

Flux d’entrée – sortie

Notion de Flux

- Un flux de sortie est un endroit sur lequel on peut écrire des données.
- Les sorties les plus courantes sont :
  - ✓ un fichier, qu'il soit à accès séquentiel ou aléatoire
  - ✓ la console système ;
  - ✓ un tableau de caractères, ou d'octets
  - ✓ une URL (ou URI), dans le cas des flux HTTP
  - ✓ une socket, pour la communication ;
  - ✓ un tuyau de communication entre threads.
- Pour chacune de ces sorties, il existe une classe Java qui permet d'écrire dessus.
- À chaque type de flux de sortie est associé un flux d'entrée, qui permet de lire les données qui ont été écrites, dans les mêmes conditions

SÉRIALISATION D'OBJETS

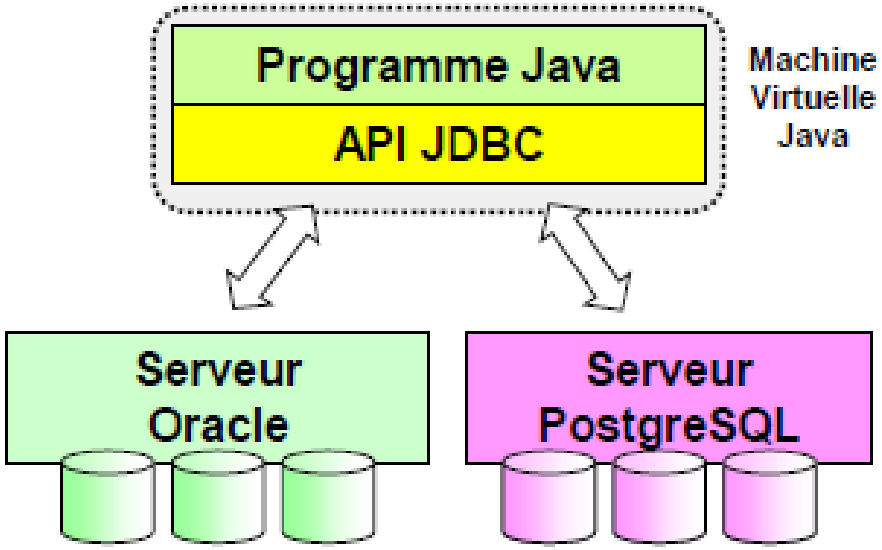
- Pour des développeurs Java, *sérialiser* un objet consiste à le convertir en un tableau d'octets, que l'on peut ensuite écrire dans un fichier, envoyer sur un réseau au travers d'une socket etc...
- Il suffit de passer tout objet qui implémente l'interface **Serializable** à une instance de **ObjectOutputStream** pour sérialiser un objet.
- L'interface Serializable n'expose aucune méthode; implémenter cette interface consiste donc juste à déclarer cette implémentation
- Si cet objet ne comporte pas de champ spécifiques comme des connexions à des bases de données, des fichiers ou des threads, cette sérialisation se déroulera sans problème

❖ **voir projets « fichierserialise », « SerializerPersonne » et «SerialisationPersonne » sous NetBeans IDE**

Connexions aux DB

JDBC

- **Java DataBase Connectivity (JDBC)** = API java standard (classes Java) qui permet un accès et interaction homogène à des bases de données depuis un programme Java au travers du langage SQL.

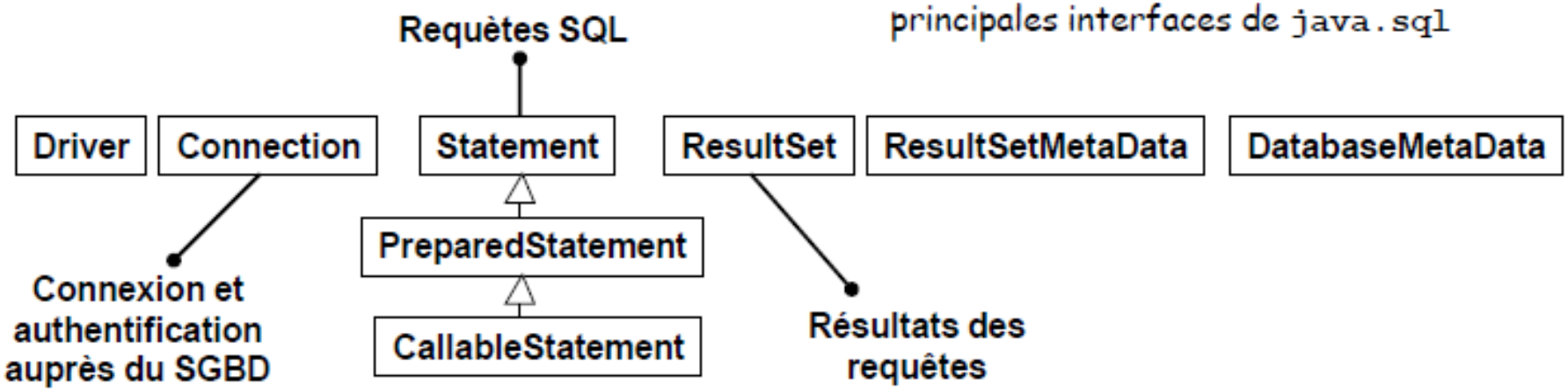


L'API est indépendante des SGBD. Un changement du SGBD ne doit pas impacter le code applicatif

Rappels Bases de données

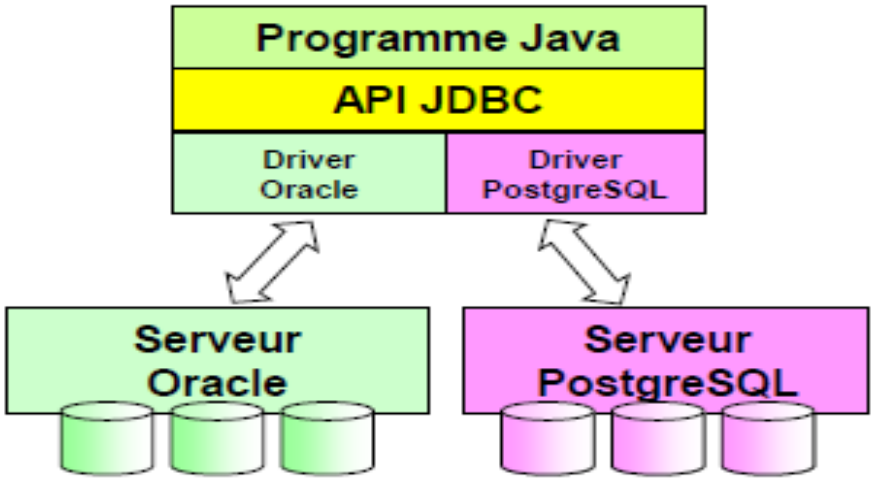
- Les bases de données (BDD) s'agit d'un système de fichiers transparents pour l'utilisateur permettant de stocker des données organisées.
- Plusieurs utilisateurs peuvent accéder simultanément aux données dont ils ont besoin
- Les données sont ordonnées par « tables », c'est-à-dire par regroupements de plusieurs valeurs.
- Le langage permettant d'interroger des bases de données est le langage SQL (Structured Query Language)
- Deux éléments nécessaires: BD et SGBD .
- Exemples: PostgreSQL, MySQL , SQL Server, Oracle, Access.

Interfaces de l'API JDBC



PILOTES JDBC

- Le code applicatif est basé sur les interfaces du JDBC . Pour accéder à un SGBD il est nécessaire de disposer de classes **implémentant ces interfaces.**
- ✓ Elles **dépendent du SGBD adressé.**
- ✓ L'ensemble de ces classes pour un SGBD donné est appelé **pilote (driver) JDBC**

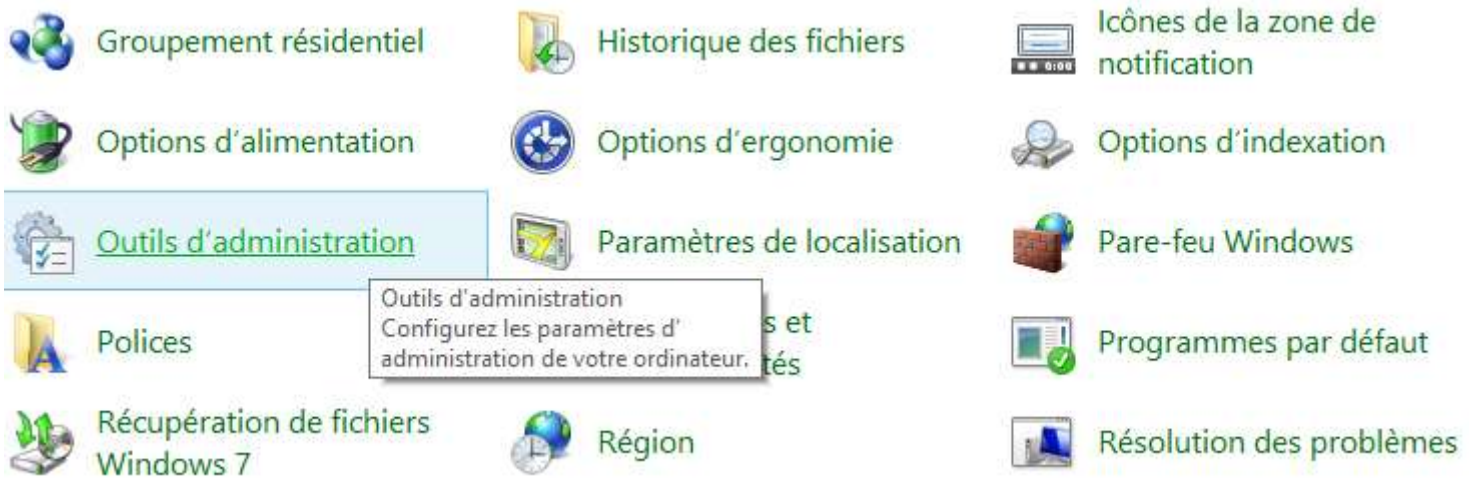


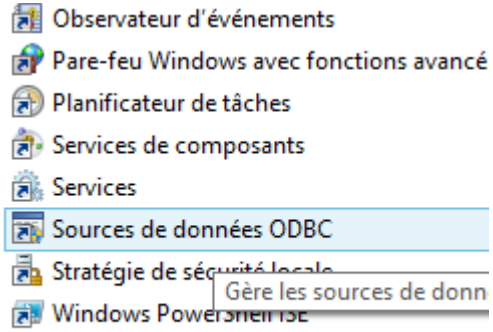
- Les classes d'implémentation du driver jdbc sont dans une archive (fichier jar ou zip) qu'il faut intégrer (sans la décompresser) au niveau de l'application au moment de l'exécution
- Exemple: pilote ODBC pour Access, et le pilote mysql pour Mysql
- Quelques pilotes viennent avec le java (d'origine, comme **odbc:sun.jdbc.odbc.JdbcOdbcDriver**), et d'autre doivent être ajoutés à l'environnement (le cas de **mysql: com.mysql.jdbc.Driver**)

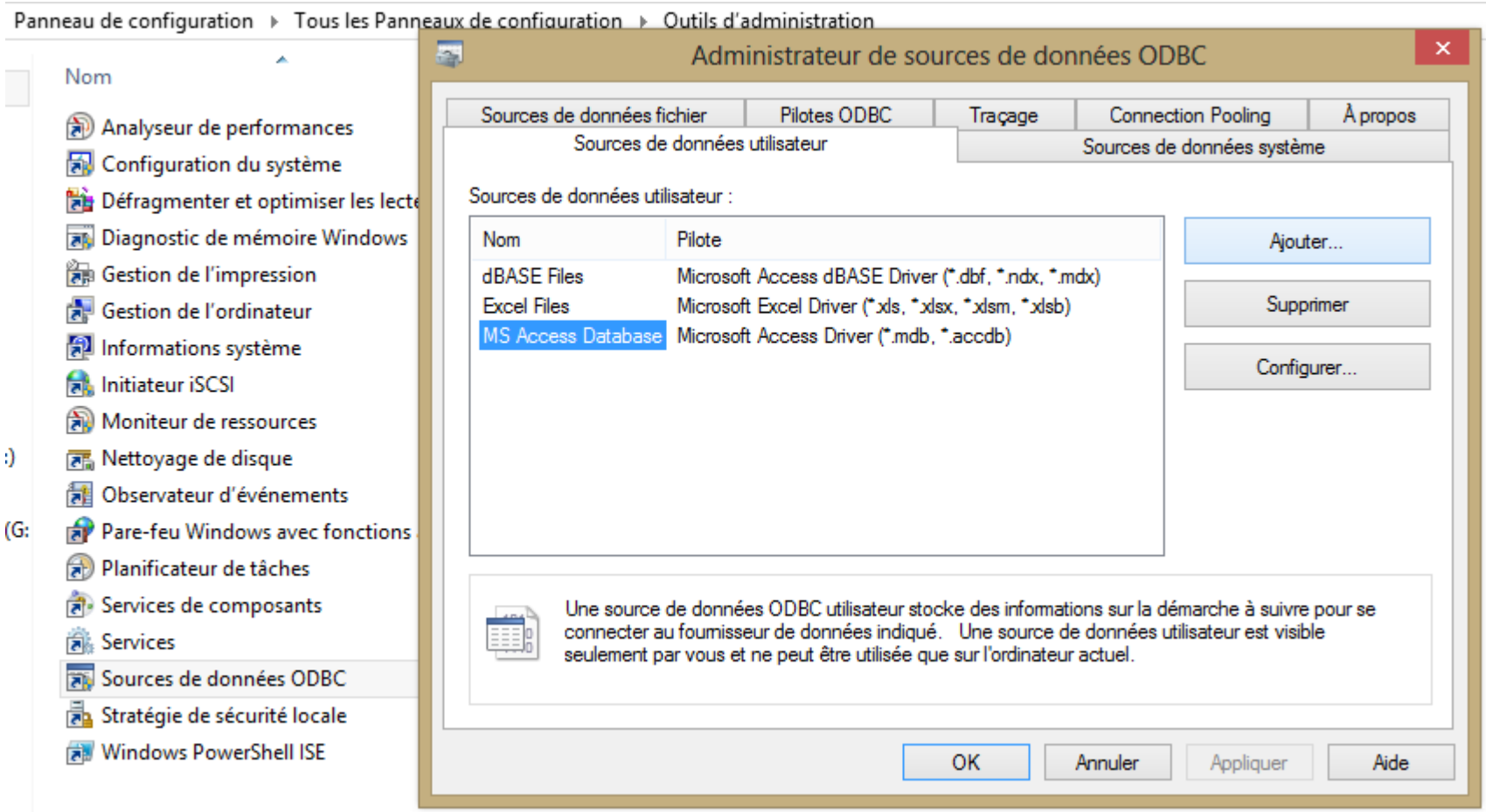
Avant Programme Java…

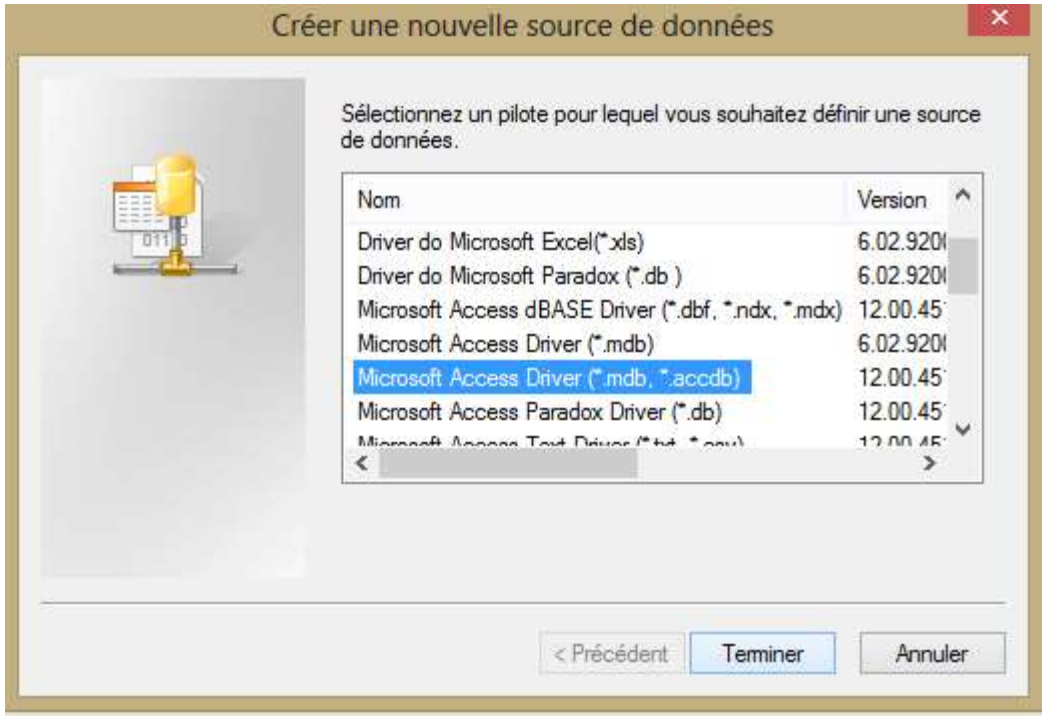
- Création de la base de données:
- ✓ Utilisation de MS Access pour BD Access (sous Windows)
- ✓ Utilisation de WampServer pour BD Mysql
- ✓ Utilisation de PostreSQL
- ✓ Utilisation de Oracle
- Ajout et Configuration du pilote adéquat:
- ✓ Pour Access l'ODBC est existant; il reste juste à configurer avec notre base
- ✓ Pour les autres il faut le télécharger en « .jar » ou « .zip », le placer le dossier lib/ext présent dans le dossier d'installation du JRE

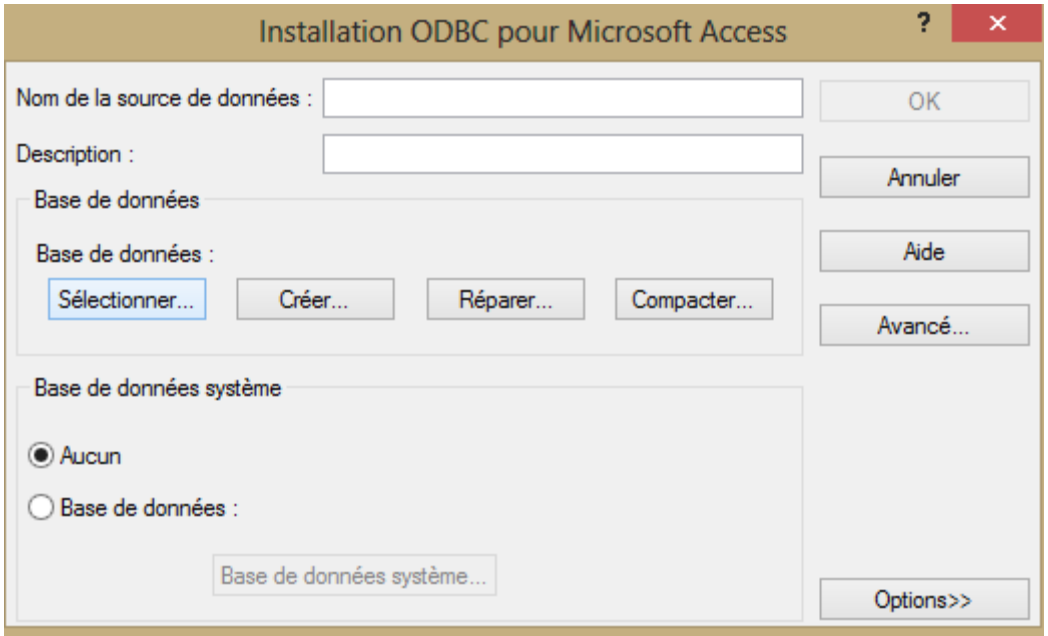
Exemple: configuration pour une base de données « Ecole » crée en MS Access

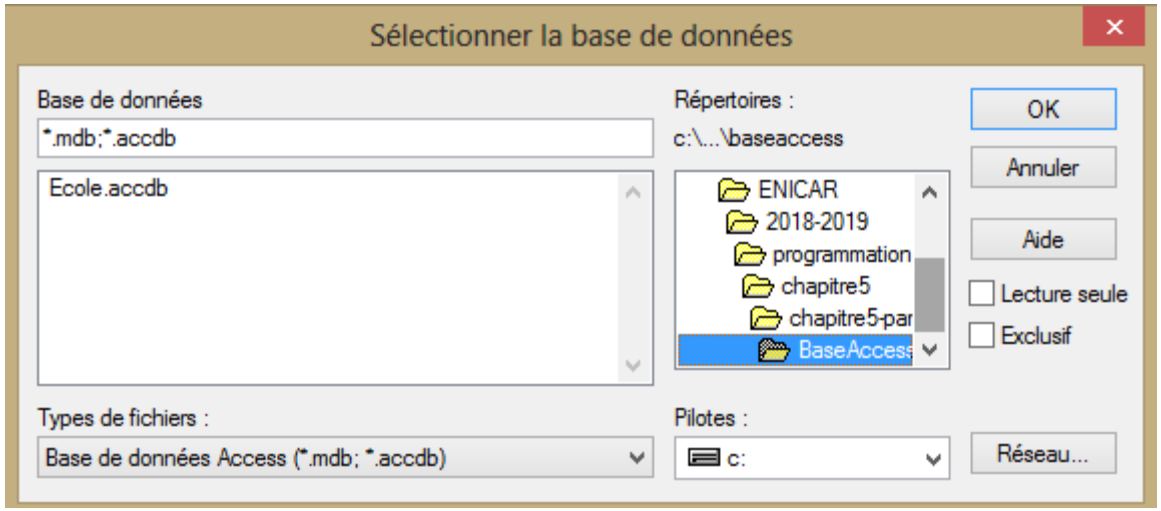





Panneau de configuration ▸ Tous les Panneaux de configuration ▸ Outils d'administration
Nom
Analyseur de performances
Configuration du système
Défragmenter et optimiser les lecte
Diagnostic de mémoire Windows
Gestion de l'impression
Gestion de l'ordinateur
Informations système
Initiateur iSCSI
Moniteur de ressources
Nettoyage de disque
Observateur d'événements
Pare-feu Windows avec fonctions
Planificateur de tâches
Services de composants
Services
Sources de données ODBC
Stratégie de sécurité locale
Windows PowerShell ISE
Administrateur de sources de données ODBC
Sources de données fichier
Pilotes ODBC
Traçage
Connection Pooling
À propos
Sources de données utilisateur
Sources de données système
Sources de données utilisateur :
Nom
Pilote
dBASE Files
Microsoft Access dBASE Driver (*.dbf, *.ndx, *.mdx)
Excel Files
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
MS Access Database
Microsoft Access Driver (*.mdb, *.accdb)
Ajouter...
Supprimer
Configurer...
Une source de données ODBC utilisateur stocke des informations sur la démarche à suivre pour se connecter au fournisseur de données indiqué. Une source de données utilisateur est visible seulement par vous et ne peut être utilisée que sur l'ordinateur actuel.
OK
Annuler
Appliquer
Aide


Créer une nouvelle source de données
Sélectionnez un pilote pour lequel vous souhaitez définir une source de données.
Nom
Version
Driver do Microsoft Excel(*.xls)
Driver do Microsoft Paradox (*.db)
Microsoft Access dBASE Driver (*.dbf, *.ndx, *.mdx)
Microsoft Access Driver (*.mdb)
Microsoft Access Driver (*.mdb, *.accdb)
Microsoft Access Paradox Driver (*.db)
< Précédent
Terminer
Annuler

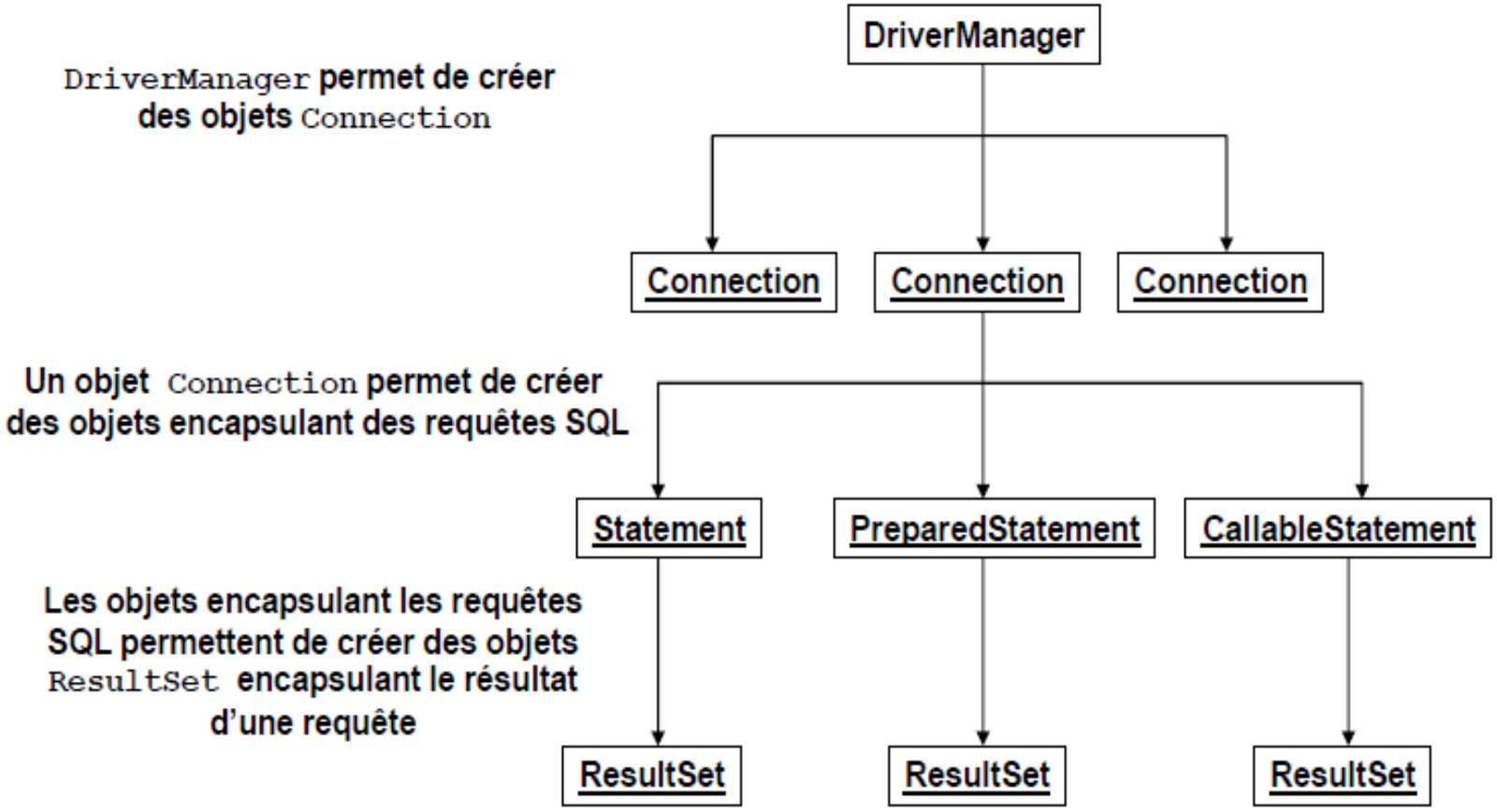

Installation ODBC pour Microsoft Access
Nom de la source de données :
Description :
Base de données
Base de données :
Sélectionner...
Créer...
Réparer...
Compacter...
Base de données système
Aucun
Base de données :
Base de données système...
OK
Annuler
Aide
Avancé...
Options>>



La base de données est prête, les tables sont créées, remplies et nous possédons le driver nécessaire ! Il ne nous reste plus qu'à nous connecter. Créons un nouveau projet Java et commençons notre code ...

DRIVE MANAGER

- DriverManager : classe java à laquelle s'adresse le code de l'application cliente.



- Au début on doit charger le driver (à partir de JDBC 4.0 on n'a plus besoin de charger explicitement le pilote JDBC. Tous les drivers compatibles JDBC 4.0 présent dans le classpath de l'application sont automatiquement chargés):

- ✓ Class.forName("odbc:sun.jdbc.odbc.JdbcOdbcDriver");
- ✓ Class.forName("org.postgresql.Driver");
- ✓ Class.forName("com.mysql.jdbc.Driver");
- ✓ Class.forName("oracle.jdbc.OracleDriver ").newInstance()

- Pour les différents drivers on doit prévoir pour celle méthode (forName()) la génération d'une exception: **ClassNotFoundException**
- Il faut qu'une connexion soit effective afin que le programme et la base de données puissent communiquer

- ✓ Connection conn = DriverManager.getConnection(url, user, pwd);
- ✓ Connection conn = DriverManager.getConnection(url,);

Exemples selon les drivers pour une BD intitulée « Ecole »:

- ✓ Connection conn = DriverManager.getConnection("jdbc:odbc:Ecole");
- ✓ Connection conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/Ecole");
- ✓ Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/Ecole");
- Pour Oracle la forme du String est: **jdbc:oracle:thin@serveur:port:base**

```
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@im2ag-oracle.e.enicar.tn:1521:Ecole" );
```

- Pour les différents drivers on doit prévoir pour celle méthode (getConnection()) la génération d'une exception: **SQLException**
- A la fin fermer la connexion: conn.close(); (dans finally)

Préparer et exécuter une requête

- Une fois une Connection créée on peut l'utiliser pour créer et exécuter des requêtes (**statements**) SQL.
- 3 types **(**interfaces) d'objets statement :
- ✓ **Statement** : requêtes simples (SQL statique)
- ✓ **PreparedStatement** : requêtes précompilées (SQL dynamique si supporté par SGBD) qui peuvent améliorer les performances
- ✓ **CallableStatement** : encapsule procédures SQL stockées dans le SGBD
- 3 formes (méthodes) d'exécutions :
- ✓ **executeQuery** : pour les requêtes qui retournent un résultat (SELECT); résultat accessible au travers d 'un objet **ResultSet**
- ✓ **executeUpdate** : pour les requêtes qui ne retournent pas de résultat (INSERT, UPDATE, DELETE, CREATE TABLE et DROP TABLE)
- ✓ **execute** : quand on ne sait pas si la requête retourne ou non un résultat, procédures stockées

- Exemple

```
Statement stmt = conn.createStatement();
String my = "SELECT prenom, nom, email " + "FROM employe
    " + "WHERE (nom='Dupont') AND (email IS NOT NULL) " +
    "ORDER BY nom";
ResultSet rs = stmt.executeQuery(my);
```

Lecture des résultats

Méthodes de ResultSet

- Méthodes de parcours

`first()`	Positionne sur la première ligne (1er enregistrement)
`last()`	Positionne sur la dernière ligne (dernier enregistrement)
`next()`	Passe à la ligne suivante
`previous()`	Passe à la ligne précédante
`beforeFirst()`	Positionne avant la première ligne
`afterLast()`	Positionne après la dernière ligne
`absolute(int)`	Positionne à une ligne donnée
`relative(int)`	Déplacement d'un nombre de lignes donné par rapport à ligne courante

- Méthodes de test de la position du curseur

`boolean isFirst()`	True si curseur positionné sur la première ligne
`boolean isBeforeFirst()`	True si curseur positionné avant la première ligne
`boolean isLast()`	True si curseur positionné sur la dernière ligne
`boolean isAfterLast()`	True si curseur positionné après la dernière ligne

Exemple

Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT nom, code_client FROM Clients");

Nom	Prénom	Code_client	Adresse
DUPONT	Jean	12345	135 rue du Lac
DUROND	Louise	12545	13 avenue de la Mer
... ...			
ZORG	Albert	45677	8 Blvd De la Montagne

Nom	Code_client
DUPONT	12345
DUROND	12545
... ...	
ZORG	45677

```
while (rs.next())
{
//En SQL les numéros de colonnes débutent à 1
String s = rs.getString("nom"); // rs.getString(1);
int i = rs.getInt("code_client"); // rs.getInt(2);
System.out.println("Ligne = " + s + " " + i );
}
```

Type SQL	Méthode	Type Java
CHAR	getString	String
VARCHAR	getString	String
NUMERIC	getBigDecimal	java.Math.BigDecimal
DECIMAL	getBigDecimal	java.Math.BigDecimal
BIT	getBoolean	boolean Boolean
TINYINT	getByte	byte Integer
SMALLINT	getShort	short Integer
INTEGER	getInt	int Integer
BIGINT	getLong	long Long
REAL	getFloat	float Float
FLOAT	getDouble	double Double
DOUBLE	getDouble	double Double
DATE	getDate	java.sql.Date
TIME	getTime	java.sql.Time
TIME STAMP	getTimestamp	java.sql.Timestamp

Peut être appelée sur n'importe quel type de valeur

`getObject` peut retourner n'importe quel type de donnée « packagé » dans un objet java (wrapper object)

Si une conversion de données invalide est effectuée (par ex DATE -> int), une `SQLException` est lancée

- Un objet Statement représente une simple (seule) requête SQL.
- Un appel à executeQuery() , executeUpdate() ou execute() ferme implicitement tout ResultSet courant associé avec l'objet Statement
- Avant d 'exécuter une autre requête avec un objet Statement il faut être sûr d'avoir exploité les résultats de la requête précédente.
- Si application nécessite d'effectuer plus d'une requête simultanément, nécessaire de créer et utiliser autant d'objets Statement.

Exemple complet

```
import java.sql.*;
public class TestJDBC {
public static void main(String[] args) throws Exception {
Class.forName("postgres95.pgDriver");
Connection conn =
DriverManager.getConnection("jdbc:pg95:mabase",
"enicar", "");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * from employe");
while (rs.next()) {
String nom = rs.getString("nom");
String prenom = rs.getString("prenom");
String email = rs.getString("email");
}
rs.close();
stmt.close();
conn.close();
}
}
```