

# **Java : Partie 4**

## **Persistance de Données**

Juin 2013

Auteur : Chouaïb LAGHLAM

Concepteur Développeur Informatique  
Module : Programmation Java  
Persistance de Données

---

**Persistance de Données : Sommaire**

<b>Préalabe.....</b>	<b>3</b>
<b>Qu'est que la Persistance de Donnée ? .....</b>	<b>4</b>
Volume des données à stocker .....	4
Sécurité et confidentialité des données .....	4
Persistance des données.....	5
Types de stockage.....	6
Fichiers Texte.....	6
Fichiers XML .....	6
Fichiers spécialisés.....	6
Bases de données .....	6
Autre type de stockage .....	7
La Sérialisation des Objets .....	7
Sérialiser des objets.....	7
DéSérialiser des objets.....	7
Exemple.....	7
Accès aux Fichiers Texte .....	11
Lire une Fichier texte .....	11
Ecrire dans un Fichier texte .....	12
Accès aux Bases de Données.....	13
Différentes manières d'accéder à une Base de Données .....	13
Accès à une Base de Données via JDBC .....	14
Paramétrage d'accès JDBC à une Base de Données .....	15
Classe ParametresBD .....	15
Classe EtatSQL .....	19
<b>Exemple d'accès à une BD.....</b>	<b>20</b>
Base de données et Table dans MYSQL .....	20
Base de données et Table dans SQL Server .....	21
Changez, dans la classe «ParametresBD» : le nom de la base et le type de SGBDR .....	21
Testez l'accès à la BD .....	21
Exemple de test : .....	21

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

---

## Préalabe

Avant de commencer à lire cette 4<sup>ème</sup> partie, vous êtes supposé avoir les connaissances de :

- ➔ La 1<sup>ère</sup> partie qui permet d'installer l'environnement de développement : Eclipse,
- ➔ La 2<sup>ème</sup> partie qui initie à la programmation de base en Java,
- ➔ La 3<sup>ème</sup> partie qui vous donne les connaissances nécessaires en Programmation Orientée Objets (POO).

Merci de créer dans votre Workspace, un projet :

- De type « **Java Project** »,
- Nommez le « **prj\_Java\_030\_Persistence**»,
- Créez un package « **pack\_Serialisation**» sous le dossier **src**,
- Créez un package « **pack\_accesFichiersTexte**» sous le dossier **src**,
- Créez un package « **pack\_BDDirect**» sous le dossier **src**.

## Qu'est que la Persistance de Donnée ?

### Volume des données à stocker

Chaque application informatique manipule des données, mais toutes n'ont pas les mêmes besoins pour le stockage de données. Prenons des exemples :

- ➔ Le jeu très connu «**Le solitaire**» où il faut ranger les cartes par catégorie (cœur, pique, carreau, trèfle) :
  - il y a besoin de stocker très peu d'informations : la position des cartes dans la dernière partie, Afin de vous la réafficher au prochain démarrage du jeu,
- ➔ un **site Internet de type forum** où les utilisateurs s'échangent des renseignements, des avis sur des sujets précis : Généralement : le site garde une trace de votre email, de la date de connexion, des questions posées, .... : Pas beaucoup d'informations,
- ➔ Le **site de votre banque** : les données stockées sont très nombreuses :
  - Les données sur votre identité,
  - Les données sur vos comptes,
  - Les données sur les mouvements financiers réalisés sur vos comptes : paie, retrait d'argent, remise de chèque, prélèvements, virements, ... ;
- ➔ Un **jeu d'échecs** digne de ce nom :
  - Stocke des milliers de combinaison de coups possibles pour faire de l'intelligence artificielle,
  - Stocke l'enchaînement des coups dans une partie commencée,
  - .....

### Sécurité et confidentialité des données

La sécurité des données consiste à ne pas les perdre,

La confidentialité des données consiste à mettre en place des autorisations pour : consulter ou mettre à jour les données.

Là aussi, selon le type d'applications, les besoins ne sont pas les mêmes.

Reprenons les exemples ci-dessus :

- ➔ Le jeu «**Le solitaire**» n'a besoin d'aucune sécurité ni confidentialité particulière :
  - On peut perdre les données de la dernière partie, quelqu'un peut lire les infos sur l'enchaînement des cartes : aucune importance,
- ➔ un **site Internet de type forum** :
  - Là déjà : il faut commencer à s'occuper de la confidentialité des identités des internautes mais tous les sites de ce genre ne mettent pas en place des stratégies très poussées et une simple sauvegarde suffit pour ne pas perdre les données,
- ➔ Le **site de votre banque** :

C'est une application très stratégique :

- Sécurité : la perte de données aura des conséquences très négatives :
  - Perte des flux financiers récents : donc décalage dans les calculs des soldes des clients, .....

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistance de Données

- Il faut sauvegarder sur des sites géographiques différents, sauvegarder presque en live, ....

- Confidentialité : la protection des identités des clients de la banque et de leurs opérations bancaires est très importante :
  - Liaisons sécurisées, cryptage de données, stratégies de mots de passe, ....

→ Un **jeu d'échecs** digne de ce nom :

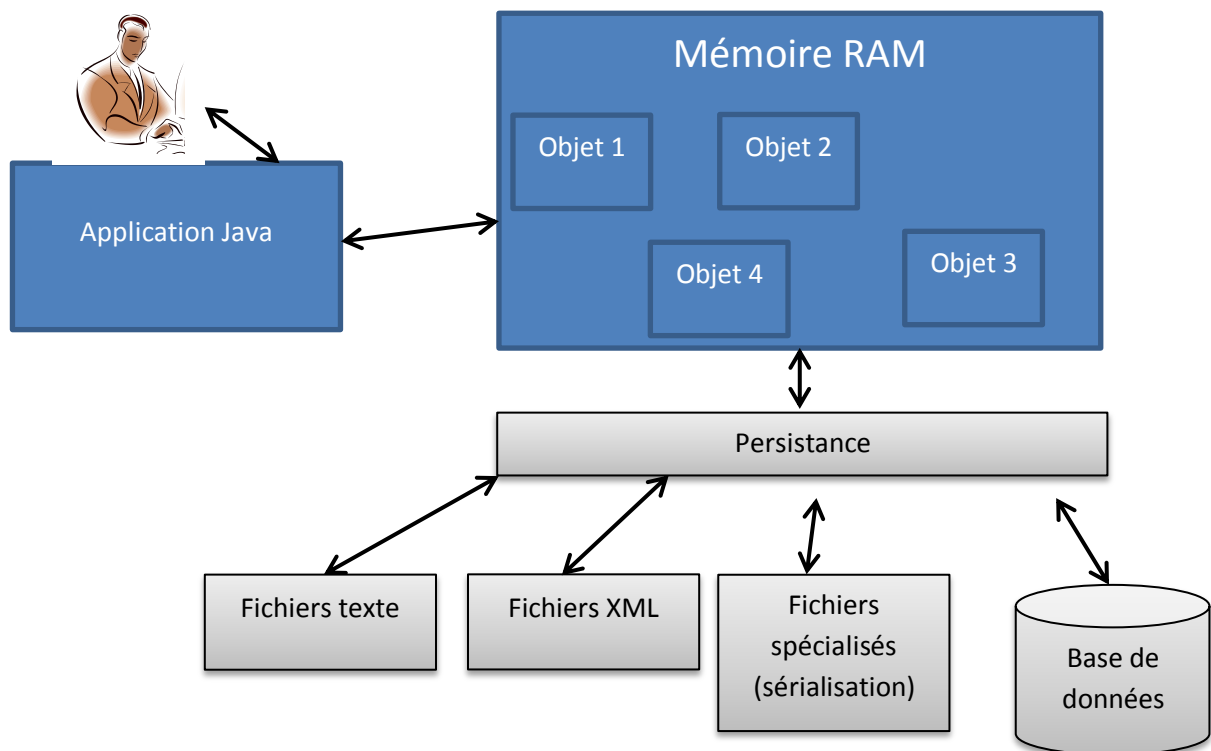
- La confidentialité du joueur est garantie d'office car ce dernier utilise un pseudo et ne donne aucune information réelle sur lui-même,
- La sécurité consiste en une simple sauvegarde.

### Persistance des données

On appelle la persistance des données l'opération qui consiste à stocker les données sur un support externe à la mémoire vive (la mémoire RAM),

En java (donc on programmation orientée objets), pendant l'exécution d'une application, les données sont stockés dans des objets. Ces objets sont supprimés de la mémoire après l'exécution de l'application :

Exemple :



Selon le type d'applications à faire : il faut choisir un type de stockage de données.

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

---

## Types de stockage

### Fichiers Texte

- ➔ Ce type de stockage de données est facile à mettre en place,
- ➔ Il ne garantit aucune confidentialité,
- ➔ L'application qui remplit le fichier et celle qui le lit doivent se mettre d'accord sur la structure des données dans
- ➔ Le fichier :
  - Comment une ligne du fichier est décomposée en informations élémentaires ?
  - Comment les informations sont séparées dans une ligne : le caractère séparateur ?
  - ....

### Fichiers XML

- ➔ Ce type de stockage est légèrement meilleur que les fichiers texte,
- ➔ Pas de confidentialité garantie,
- ➔ On stocke le nom de l'information (métadonnée) et la valeur de l'information (donnée) ensemble :
  - Par exemple : on stocke : nom client et 12345 ;
- ➔ On utilise un système de balises (comme en HTML) pour séparer les données dans le fichier,
- ➔ Ce type de fichiers est très répandu actuellement (en 2013) pour stocker les données de configuration :
  - Paramétrage d'une application : langue utilisée, chemins des ressources : documents, photos, ....
  - Paramétrage d'une BD : nom du serveur, nom de la BD, ....
  - ....

### Fichiers spécialisés

- ➔ Ici on crée des données dans un fichier lisible uniquement par un type particulier d'applications : autrement dit, On ne peut pas ouvrir le fichier par un simple «Bloc-notes». Par exemple : ouvrir un document Word par le Bloc-notes ne donnera rien de lisible,
- ➔ Exemples des fichiers spécialisés : les cookies, la base de registre de Windows, les fichiers .ini, .....
- ➔ Nous allons utiliser, ci-dessous, ce type de fichier dans ce que nous allons appeler «la sérialisation»,

### Bases de données

- ➔ C'est le meilleur support actuel pour la sécurité et la confidentialité des données,
  - Les technologies telles que le **mirroring** et la **réplication** permettent respectivement des sauvegardes en temps réel et sur plusieurs sites géographiques,
- ➔ Les bases de données permettent de réaliser aisément les **relations entre les données** : telle facture appartient à Tel client, tel client a passé telles commandes, ...
- ➔ Les Systèmes de Gestion de Bases de Données Relationnelles (SGBDR) permettent de garantir une **haute confidentialité**,
- ➔ Il convient de remarquer qu'il y ait deux types de BD actuellement :
  - **Bases de données relationnelles**,
  - **Bases de données Objets** : en émergence : pas encore très répandues à ce jour (en 2013),
  -

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

---

#### Autre type de stockage

Nous n'avons cité ci-dessus que les types les plus répandus : d'autres types existent :

Par exemple : des types anciens mais qui existent encore : fichiers indexés, ...etc.

### La Sérialisation des Objets

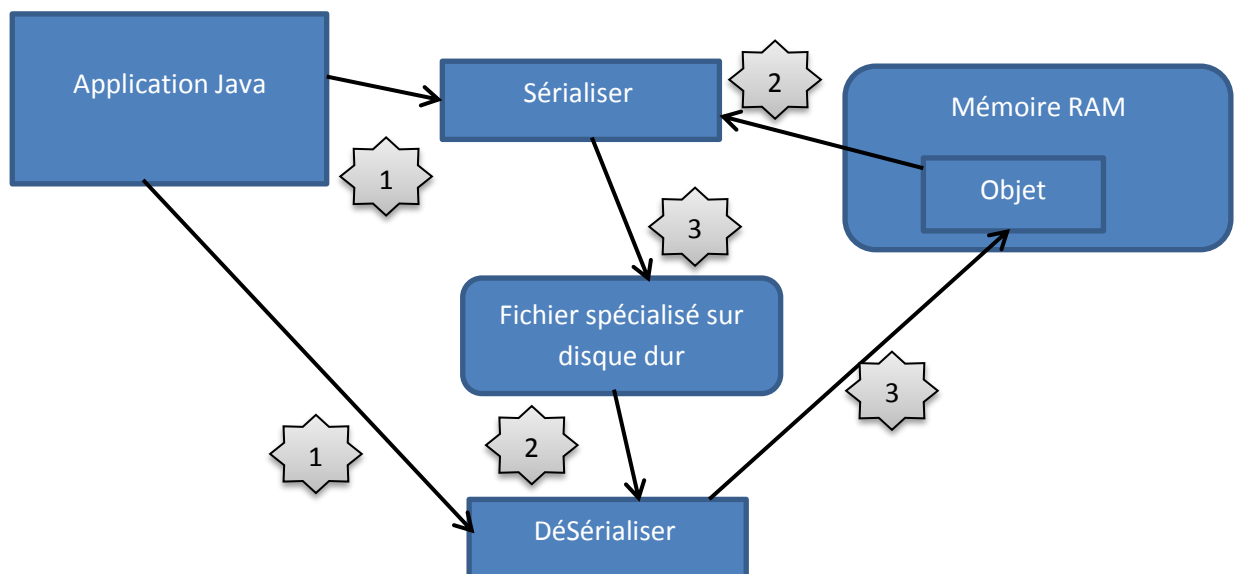
La première idée qui a émergé, pour stocker des objets créés en mémoire par une application Java, est de les stocker tels quels sur le disque dans des fichiers spécialisés et faire l'opération inverse pour les restituer en mémoire.

#### Sérialiser des objets

La sérialisation est l'opération qui va consister à stocker un objet java dans un fichier spécialisé stocké sur le disque dur. L'application peut s'arrêter. L'objet en mémoire sera perdu mais le fichier spécialisé garde une copie de l'objet.

#### DéSérialiser des objets

La DéSérialisation consiste à lire la copie de l'objet dans le fichier spécialisé et le restituer en mémoire pour que l'application java le réutilise.



#### Exemple

#### Classe de l'objet à créer et à sérialiser:

```
package pack_Serialisation;
// faire référencer à l'interface qui permet de sérialiser / désérialiser
import java.io.Serializable;

public class Sportif implements Serializable {
    // attributs
    private String nom;
    private String prenom;
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

---

```
private int taille;
// constructeur
public Sportif(String nom, String prenom, int taille) {
this.nom = nom;
this.taille = taille;
this.prenom = prenom;
}
// getteur
public String getNom() {
return nom;
}
// setteur
public void setNom(String nom) {
this.nom = nom;
}
// getteur
public int getTaille() {
return taille;
}
// setteur
public void setTaille(int taille) {
this.taille = taille;
}
// getteur
public String getPrenom() {
return prenom;
}
// setteur
public void setPrenom(String prenom) {
this.prenom = prenom;
}
}
```

Remarque :

➔ On utilise pour la classe des objets à sérialiser (ici la classe Sportif) l'interface «**Serializable**» ,

#### Code qui crée un objet et le sérialise :

```
package pack_Serialisation;
// faire référence au package de flux d'entrées/sorties
import java.io.*;
// la classe exemple pour sérialiser
public class SerialiseSportif {
```



# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

---

```
// méthode main
public static void main(String[] args) {
    // créer un objet sportif
    Sportif sp = new Sportif("Messi","Lionel",169);

    try {
        // on définit le fichier dans lequel on stockera
        // l'objet serialise : avec la classe FileOutputStream
        FileOutputStream fichier = new FileOutputStream("Ressources/sp.ser");
        // on crée un objet de type ObjectOutputStream
        // on lui fournissant le fichier dans lequel sérialiser
        ObjectOutputStream objASerialiser = new ObjectOutputStream(fichier);
        // on sérialise grace à la méthode writeObject
        objASerialiser.writeObject(sp);
        // on vide le tampon dans le fichier
        objASerialiser.flush();
        // on termine l'opération par close
        objASerialiser.close();
    } // fin try
    catch (java.io.IOException e) {
        e.printStackTrace();
    } // fin catch
} // fin main
} // fin classe
```

Remarques :

➔ Pour sérialiser, on utilise deux classes Java :

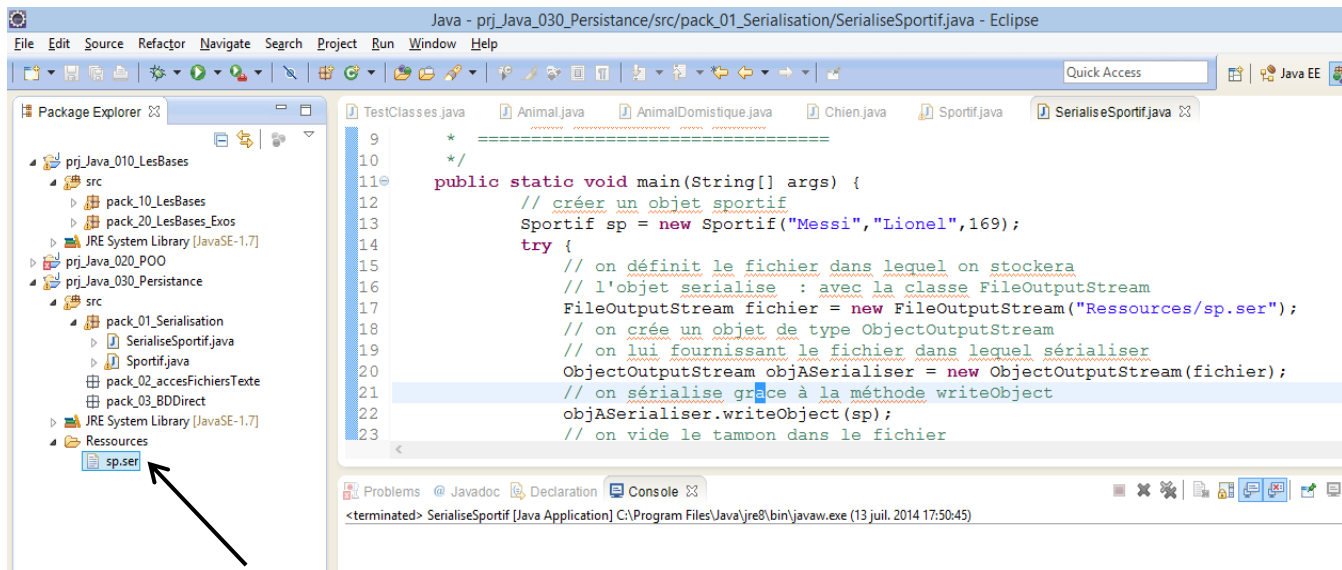
- **FileOutputStream** : qui crée le fichier dans lequel sera stocké l'objet,
- **ObjectOutputStream** avec la méthode **WriteObject** qui stocke notre objet dans le fichier.

Nous voyons le fichier sur le disque :

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistance de Données



**Maintenant le code qui déséréalise l'objet et le rend utilisable de nouveau dans le code :**

```
package pack_Serialisation;
import java.io.*;
public class DeSerialiseSportif {
    // méthode main
    public static void main(String[] args) {
        try {
            //
            // on définit le fichier dans lequel est stocké
            // l'objet serialise : avec la classe FileInputStream
            FileInputStream fichier = new FileInputStream("Ressources/sp.ser");
            // on crée un objet de type ObjectInputStream
            // on lui fournissant le fichier contenant l'objet
            // sérialisé
            ObjectInputStream objSerialise = new ObjectInputStream(fichier);
            // on appelle la méthode readObject pour déséréaliser
            // remarquer le cast
            Sportif spo = (Sportif) objSerialise.readObject();
            // on affiche les attributs de l'objet déséréalisé
            System.out.println("Sportif : ");
            System.out.println("nom : "+spo.getNom());
            System.out.println("prenom : "+spo.getPrenom());
            System.out.println("taille : "+spo.getTaille());
        } // fin try
        catch (java.io.IOException e) {
            e.printStackTrace();    // afficher le log de l'erreur(l'exception)
        } // fin catch
        catch (ClassNotFoundException e) {
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

```
        e.printStackTrace();
    } // fin catch
} // fin main
} // fin classe
```

Remarques :

➔ Pour Désérialiser, on utilise deux classes Java :

- **FileOutputStream** : qui crée le fichier dans lequel sera stocké l'objet,
- **ObjectInputStream** avec la méthode **readObject** qui récupère notre objet du fichier.

## Accès aux Fichiers Texte

Prenons le fichier texte «**source1.txt**» stocké dans le dossier «**Ressources**» du projet :

```
Hello
Bonjour
Salam
¡hola
Shalom
ciao
Hallo
```

### Lire une Fichier texte

Voici le code qui permet de lire ligne par ligne à partir d'un fichier texte :

```
package pack_accesFichiersTexte;
// faire référence au package de flux d'entrées/sorties
import java.io.*;

/*
   Classe :   exemple lecture d'un fichier texte
              sur disque dur
*/

public class LectureFichierTexte {

    /* *****
       Méthode main
       ***** */
    public static void main(String[] args) {
        try {
            String ligne ;
            // FileReader permet de référencer un fichier de données
            // BufferedReader permet de savoir quel flux (buffer) va être
            // utilisé pour utiliser ce fichier
            BufferedReader fichier =
                new BufferedReader(new FileReader("Ressources/source1.txt"));
            // readLine permet de lire une ligne du fichier
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistance de Données

---

```
        while ((ligne = fichier.readLine()) != null) {
            // on affiche la ligne sur la console
            System.out.println(ligne);
        } // fin while
        // on ferme le fichier de données
        fichier.close();
    } // fin try
    catch (Exception e) {
        e.printStackTrace();
    } // fin catch
} // fin main
// fin classe
}
```

Remarques :

➔ Nous utilisons ici deux classes Java :

- **FileReader** : qui précise le nom et le chemin du fichier à lire,
- **BufferedReader** : qui permet de lire le fichier, ligne par ligne, grâce à la méthode **readLine()**,

➔ Si la ligne lue contient plusieurs informations séparées par un caractère séparateur (par « ; » par exemple), Utilisez la méthode « **Split** » des chaînes de caractères pour extraire chaque information.

#### Ecrire dans un Fichier texte

Voici le code qui permet d'écrire dans un fichier texte :

```
package pack_accesFichiersTexte;
//faire référence au package de flux d'entrées/sorties
import java.io.*;
import java.util.*;
/*=====
Classe : exemple ecriture dans un fichier texte sur disque dur
=====
*/
public class EcritureDansFichierTexte {
    //
    // méthode main
    //
    public static void main(String[] args) {

        try {
            //String ligne ;
            int nombre = 3450;
            // classe FileWriter : référence le fichier texte de sortie
            // classe BufferedWriter : le buffer qui permet d'écrire dans le
            // le fichier
        }
    }
}
```

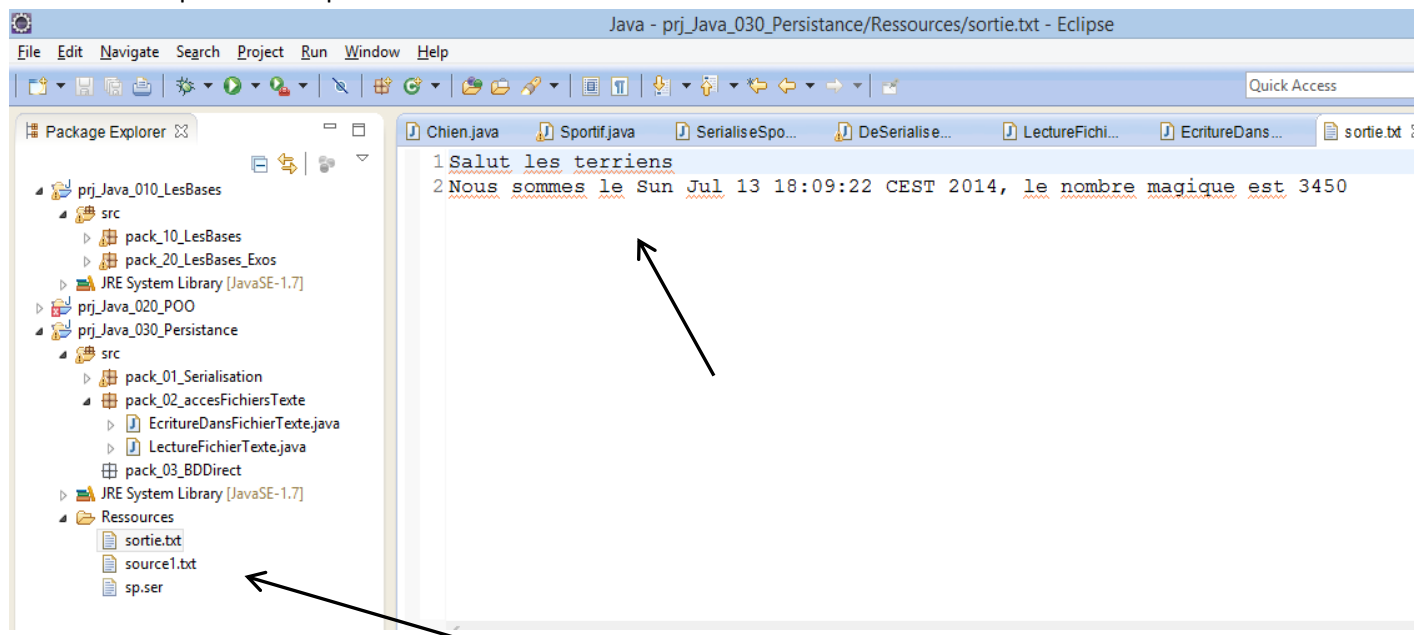
# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistance de Données

```
BufferedWriter fichier = new BufferedWriter(new FileWriter("Ressources/sortie.txt"));
// on écrit le buffer dans le fichier
Fichier.write("Salut les terriens");
// on fait un rc dans le fichier
fichier.newLine();
fichier.write("Nous sommes le " + new Date());
fichier.write(", le nombre magique est " + nombre);
// ne pas oublier de terminer le traitement
fichier.close();
}
catch (Exception e) {
    e.printStackTrace();
}
} // fin main
}
```

Voici le texte que l'on récupère en sortie :



## Accès aux Bases de Données

### Différentes manières d'accéder à une Base de Données

Il y a très longtemps, chaque langage de programmation avait ses fonctions propres pour accéder à une Base de données. Puis, il y eut l'apparition du standard **ODBC** (Open Data Base Connectivity) : une sorte de pilote qui permet d'envoyer des Instructions SQL à partir d'une application,

Des années plus tard, **JDBC** fut une version d'ODBC spécialisée pour le langage JAVA,

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistance de Données

---

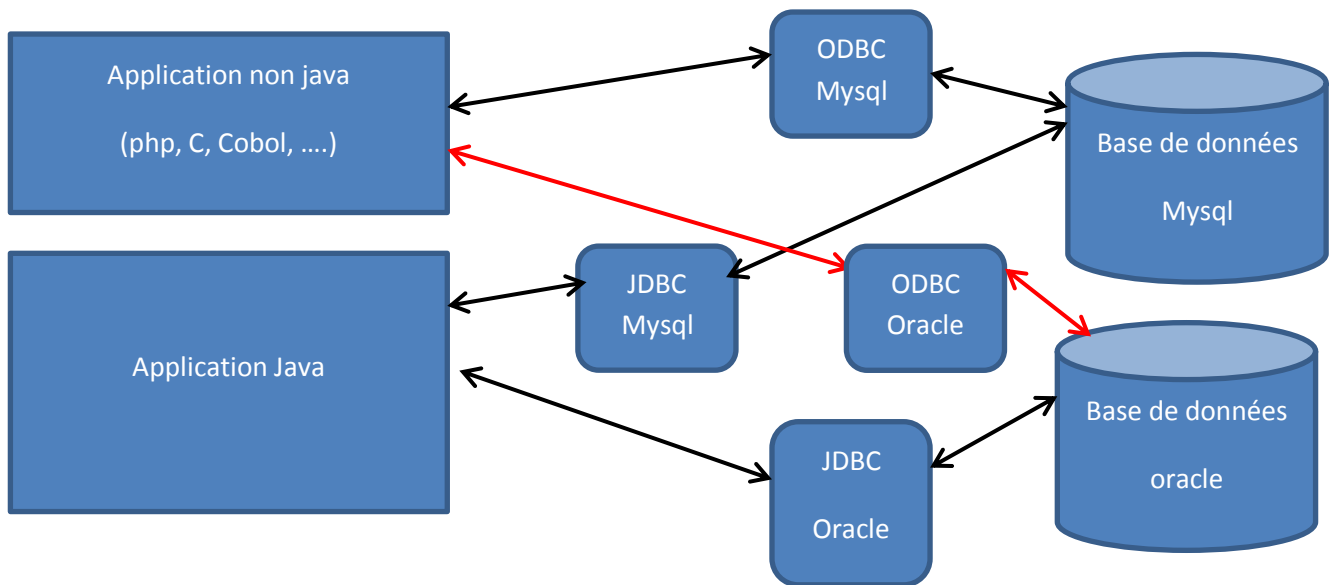
Ces dernières années, des outils sophistiqués (des Framework) ont vus le jour dans le monde java et permettent de masquer le langage SQL et d'accéder plus facilement aux BD.

Exemples de Framework ou de technologies :

- Technologie **JPA** : Java Persistence Api
- Framework **Spring**,
- Framework **Hibernate**, ....

Ici, nous allons utiliser l'accès par JDBC. Un accès par JPA est vu dans la partie consacrée aux applications Web.

Schéma :



#### Accès à une Base de Données via JDBC

Afin d'accéder à une BD relationnelle via le pilote JDBC, il faut se doter :

- ➔ Du **fichier** adéquat qui représente le **JDBC** : une archive java (fichier .jar) correspondant au type de BD à lire,
  - Par exemple : pour accéder à une BD de type Mysql : il existe le fichier «**mysql-connector-java-5.1.11-bin.jar**»
  - Par exemple : pour accéder à une BD de type SQL Server : il existe le fichier «**sqljdbc4.jar**»
- ➔ Des informations sur la BD :
  - La source de données :

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

---

- C'est une chaîne de caractères (qu'on appelle aussi la **chaîne de connexion**) et qui contient :  
Le nom du serveur de données, le nom de la BD, le nom user autorisé à accéder à la BD et le Mot de passe correspondant au user.

#### Paramétrage d'accès JDBC à une Base de Données

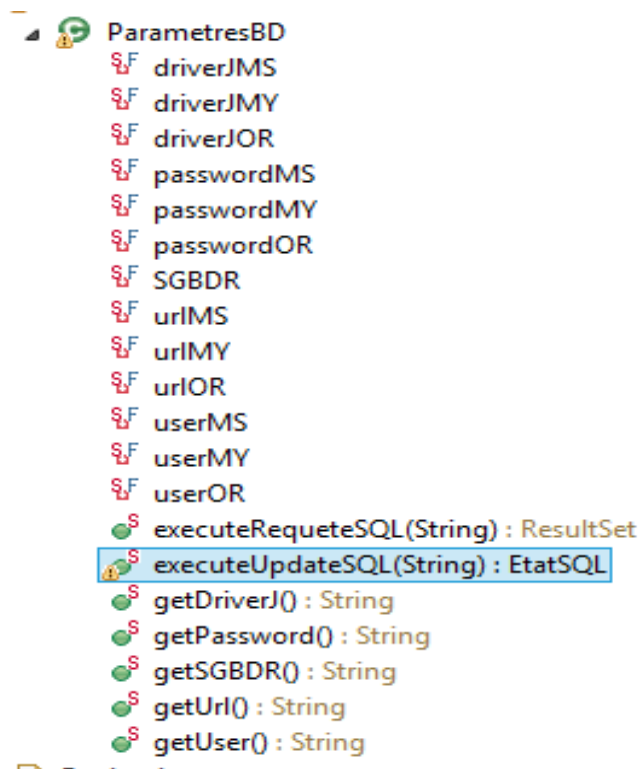
Dans une application informatique, il y a forcément de nombreux endroits différents où il faut accéder à la BD de Données. Il faut donc réfléchir à :

- Ne pas citer de nombreuses fois les noms constants : nom de serveur, nom de la BD, nom du user, mot De passe,
- Ne pas répéter les actions évidentes : se connecter à la base, envoyer la requête, récupérer le résultat de La requête, ... ;

Je vous propose ici de fabriquer une classe nommée «**ParametresBD**» qui :

- Stocke les *valeurs constantes* sur le serveur, la BD, le user et le mot de passe pour MySQL, SQL Server et mot de passe : si ces valeurs changent : c'est ici le seul endroit où il faut les changer,
- Contient deux méthodes :
  - Méthode « **executeRequeteSQL** » qui reçoit n'importe quelle requête SQL de type **SELECT**, l'envoie à la BD et renvoie le résultat au code appelant,
  - Méthode « **executeUpdateSQL** » qui reçoit n'importe quelle requête SQL de type **INSERT** ou **UPDATE** ou **DELETE**, l'envoie à la BD et renvoie true ou false au code appelant,
- Cette Classe permet l'accès aux différent SGBDR du marché : MySQL, SQL Server, Oracle,... moyennant un Un léger paramétrage.

#### Classe ParametresBD



# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

```
package com.gsb.modeles;
import java.sql.*;

/* =====
   Auteur      :      Chouaïb LAGHLAM
   Date        :      Janvier 2014
   =====
   Accès aux Données d'une BD via JDBC
   =====
*/

public class ParametresBD {
    // propriété      :      type SGBDR : Mysql, MS SQL Server, Oracle,..
    // changer ci-dessous l'attribut SGBDR en valeur MY ou MS ou OR pour spécialiser
    // l'accès à MySQL ou à Microsoft SQL Server ou à Oracle
    private static final String SGBDR="MS";
    /*
     *      =====
     *      paramètres Serveur Mysql
     *      =====
    */
    //      constante      :      nom de la source de données :
    //      :      Type d'accès + nom serveur + nom de la BD
    private static final String urlMY = "jdbc:mysql://localhost/BDFrais";
    //      constante      :      login
    private static final String userMY = "root";
    //      constante      :      mot de passe
    private static final String passwordMY = "";
    //      constante      :      driver jdbc
    private static final String driverJMY = "com.mysql.jdbc.Driver";
    /*
     *      =====
     *      paramètres Serveur SQLServer 2012 et plus
     *      =====
    */
    //      constante      :      nom de la source de données
    private static final String urlMS =
        "jdbc:sqlserver://HP_LAGHLAM\\SQLLAGHLAM;database=BDFrais";
    //      constante      :      login
    private static final String userMS = "sa";
    //      constante      :      mot de passe
    private static final String passwordMS = "I551225";
    //      constante      :      driver jdbc
    private static final String driverJMS =
        "com.microsoft.sqlserver.jdbc.SQLServerDriver";
    /*
     *      =====
     *      paramètres Serveur Oracle 11g et plus
     *      =====
    */
    //      constante      :      nom de la source de données
```



# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

---

```
private static final String urlOR = "";
//      constante      :      login
private static final String userOR = "";
//      constante      :      mot de passe
private static final String passwordOR = "";
//      constante      :      driver jdbc
private static final String driverJOR = "";
/*
 *      =====
 *      getteurs (pas de setteurs ici)
 *      =====
 */
public static String getSGBDR() {
    return SGBDR;
}
public static String getDriverJ() {
    switch(getSGBDR())
    {
        case "MY":                // Mysql
            return driverJMY;
        case "MS":
            return driverJMS;      // MS SQL Server
        case "OR":
            return driverJOR;      // Oracle
        default :
            return "";
    }
}
public static String getUrl() {
    switch(getSGBDR())
    {
        case "MY":                // Mysql
            return urlJMY;
        case "MS":
            return urlJMS;        // MS SQL Server
        case "OR":
            return urlJOR;        // Oracle
        default :
            return "";
    }
}
public static String getUser() {
    switch(getSGBDR())
    {
        case "MY":                // Mysql
            return userJMY;
        case "MS":
            return userJMS;      // MS SQL Server
        case "OR":
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

```
        return userOR;           // Oracle
    default :
        return "";
    }
}
public static String getPassword() {
    switch(getSGBDR())
    {
        case "MY":                // Mysql
            return passwordMY;
        case "MS":
            return passwordMS;     // MS SQL Server
        case "OR":
            return passwordOR;     // Oracle
        default :
            return "";
    }
}
//      méthode      :      pour insérer ou modifier ou supprimer
//      dans la BD
public static EtatSQL executeUpdateSQL(String requete) throws ClassNotFoundException{
    try {
        Class.forName(getDriverJ());
        Connection connexion = DriverManager.getConnection(getUrl(),getUser(),getPassword());
        Statement instruction = connexion.createStatement();
        int resultatTemp = instruction.executeUpdate(requete);
        EtatSQL resultatSQL=new EtatSQL("000","table regions","OK pour : "+requete);
        return resultatSQL;
    }
    catch (Exception e)
    {
        EtatSQL resultatSQL=new EtatSQL("-100","table regions","KO pour : "+requete);
        return resultatSQL;
    }
}
//      méthode      :      pour lire BD
public static ResultSet executeRequeteSQL(String requete) throws ClassNotFoundException{
    try {
        Class.forName(getDriverJ());
        ResultSet resultat = null;
        Connection connexion = DriverManager.getConnection(getUrl()
            ,getUser(),getPassword());
        Statement instruction = connexion.createStatement();
        ResultSet resultatTemp = instruction.executeQuery(requete);
        resultat = resultatTemp;
        return resultat;
    }
    catch (Exception e) {
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

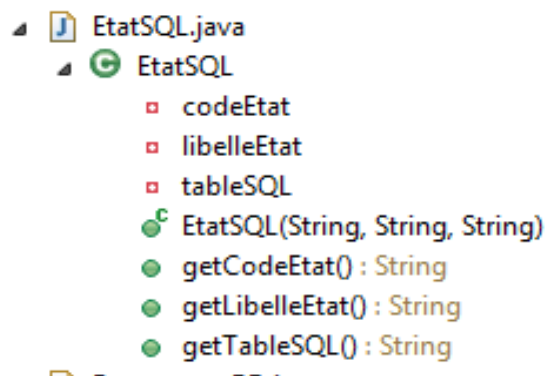
---

```
        return null;
    }
} // fin méthode
// fin classe
```

#### Classe EtatSQL

Lorsqu'une instruction SQL est exécutée par le code de la classe ci-dessus ne fonctionne pas (se plante) : erreur dans la syntaxe SQL, nom de la colonne n'existe pas, ....: il faut envoyer des informations au code appelant pour qu'il sache ce qu'il ne va pas.

Pour cela, je fournis la classe « **EtatSQL** » qui permet de renvoyer un objet contenant le code erreur, l'instruction qui plante au code appelant.



```
package pack_BDDirect;

/* =====
   Auteur      :      Chouaïb LAGHLAM
   Date        :      Janvier 2014
   =====
   Projet GSB
   Classe qui permet à une méthode accédant
   à la BD de renvoyer des infos sur
   l'exécution d'une instruction SQL
   =====
*/

public class EtatSQL {
    // propriétés
    private String codeEtat;
    private String tableSQL;
    private String libelleEtat;
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

```
// constructeur
public EtatSQL(String codeEtat, String tableSQL, String libelleEtat) {
    super();
    this.codeEtat = codeEtat;
    this.tableSQL = tableSQL;
    this.libelleEtat = libelleEtat;
}

// getteurs
}
public String getCodeEtat() {
    return codeEtat;
}
public String getTableSQL() {
    return tableSQL;
}
public String getLibelleEtat() {
    return libelleEtat;
}
}
```

## Exemple d'accès à une BD

### Base de données et Table dans MYSQL

BD : Rungis      Table : fruits

The screenshot shows a MySQL database management interface. On the left, a sidebar contains a tree view with the following structure:

- (Tables récentes) ...
- rungis
  - fruits
- Nouvelle table

An arrow points from the 'fruits' table in the sidebar to the table view. The main area displays the SQL query:

```
SELECT *
FROM 'fruits'
LIMIT 0, 30
```

Below the query, the 'Afficher' section shows the following settings:

- Ligne de départ: 0
- Nombre de lignes: 30
- En-têtes à intervalle de: 100 lignes

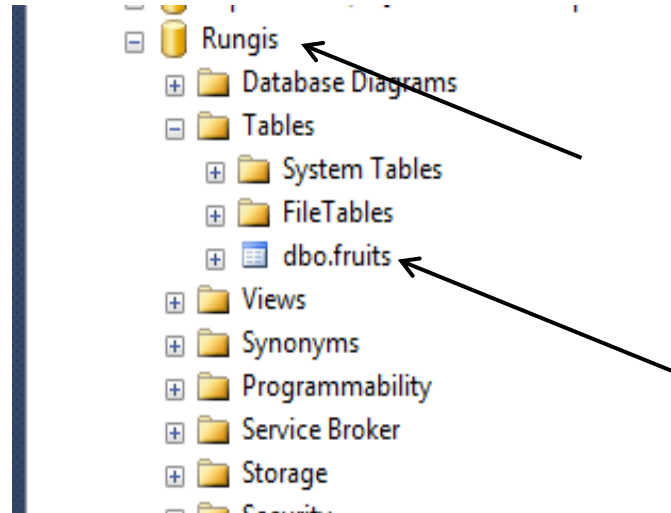
An arrow points from the 'Afficher' section to the table view. The table view shows the following data:

code	nom
Ban	Bananes
fra	Fraises
ora	Orange
pom	Pommes

## Base de données et Table dans SQL Server

BD : Rungis

Table : fruits



Changez, dans la classe «ParametresBD» : le nom de la base et le type de SGBDR

```
.....  
  
private static final String SGBDR="MS";  
  
....  
//          constante      :      nom de la source de données  
private static final String urlMS = "jdbc:sqlserver://HP_LAGHLAM\\SQLLAGHLAM;database=Rungis";  
//          constante      :      login  
private static final String userMS = "sa";  
.....
```

## Testez l'accès à la BD

Créez une classe de test avec la méthode « main » et tester des ajouts, modifications, suppression et lecture de fruits

### Exemple de test :

Lecture de tous les fruits + ajout d'un fruit dans la BD

```
package pack_BDDirect;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Persistence de Données

```
public class TestBDRungis {

    /* 1) NE PAS OUBLIER DE TELECHARGER LE JDBC
    * DU SGBDR UTILISE (MySQL ou SQL Server ou
    * Oracle ou ....
    * 2) DIRE AU PROJET JAVA D'UTILISER
    * LE JDBC : voir propriétés du projet
    */
    public static void main(String[] args) {
        try {
            /* =====
            requête de Sélection sur BD SQL Server : BD Rungis, Table fruits
            =====
            */
            String maRequete3="Select * from fruits";
            // je demande son exécution et je récupère le résultat
            ResultSet resultatRenvoye=ParametresBD.executeRequeteSQL(maRequete3);
            // j'exploite les lignes renvoyées par la BD
            while(resultatRenvoye.next()!=true)
            {
                System.out.println("Fruit : "+resultatRenvoye.getString(1)
                +" - "+resultatRenvoye.getString(2));
            }
            /* =====
            requête Insertion sur BD SQL Server : BD Rungis, Table fruits
            =====
            */

            String maRequete4="insert into fruits (nom) values('Banane')";
            // je demande son exécution et je récupère le résultat
            EtatSQL reponse=ParametresBD.executeUpdateSQL(maRequete4);
            if(reponse==null)
            {
                System.out.println("Ajout fruit dans SQL Server : KO");
            }
            else
            {
                System.out.println("Ajout fruit dans SQL Server : OK");
            }
        }
        catch (Exception e) {
            System.out.println("Problème SQL");
        }
    }
}
```

# Concepteur Développeur Informatique

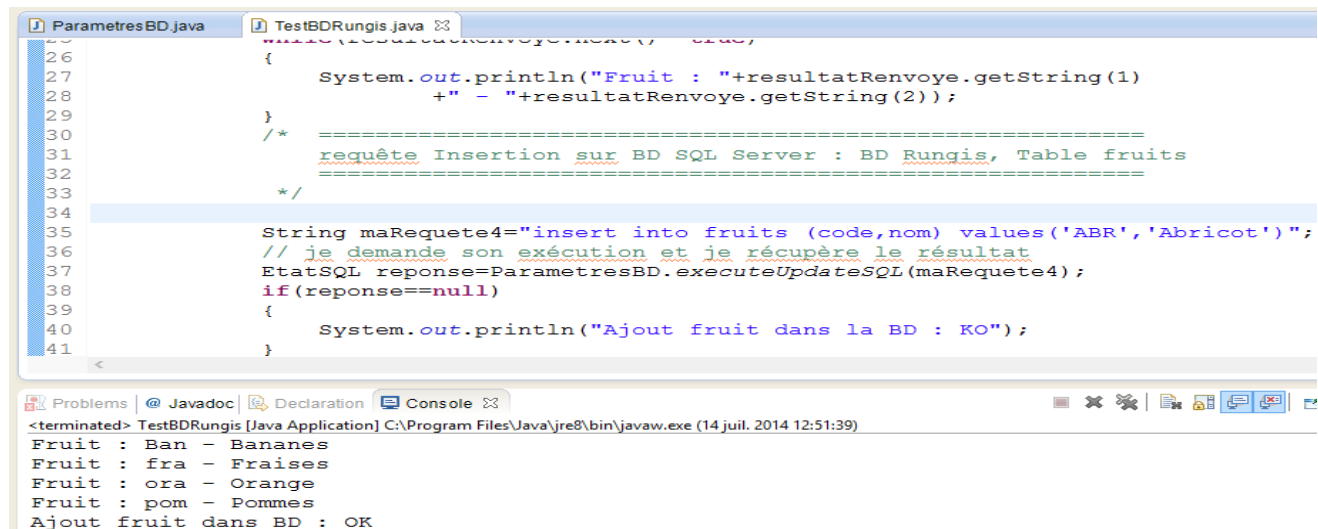
## Module : Programmation Java

### Persistence de Données

```
    }  
}  
    // fin main  
} // fin classe
```

On obtient pour MYSQL, le résultat suivant sur la console :

On affiche les fruits existants + on ajoute du fruit 'Abricot' :



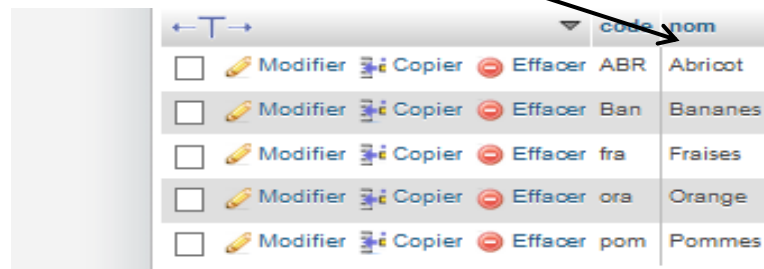
The screenshot shows an IDE with two tabs: 'ParametresBD.java' and 'TestBDRungis.java'. The 'TestBDRungis.java' tab is active, showing Java code that interacts with a database. The code includes comments in French and a SQL insert statement for 'Abricot'. Below the code, the console output is visible, showing the results of the database operations.

```
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41
```

```
String maRequete4="insert into fruits (code,nom) values('ABR','Abricot')";  
// je demande son exécution et je récupère le résultat  
EtatSQL reponse=ParametresBD.executeUpdateSQL(maRequete4);  
if(reponse==null)  
{  
    System.out.println("Ajout fruit dans la BD : KO");  
}
```

Console Output:

```
<terminated> TestBDRungis [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (14 juil. 2014 12:51:39)  
Fruit : Ban - Bananes  
Fruit : fra - Fraises  
Fruit : ora - Orange  
Fruit : pom - Pommes  
Ajout fruit dans BD : OK
```



The screenshot shows a table viewer with a table containing fruit data. Each row has a checkbox, a pencil icon for 'Modifier', a plus icon for 'Copier', and a red circle icon for 'Effacer'. An arrow points to the 'code' column header.

	code	nom
<input type="checkbox"/>	ABR	Abricot
<input type="checkbox"/>	Ban	Bananes
<input type="checkbox"/>	fra	Fraises
<input type="checkbox"/>	ora	Orange
<input type="checkbox"/>	pom	Pommes

A vous maintenant d'approfondir l'accès aux données en allant découvrir par exemple :

- ➔ La technologie JPA,
- ➔ Les mappages de BD selon les EJB,
- ➔ Les mappages de BD dans les Framework 's tels que Spring, Hibernate,....

Bonne continuation