Rappels fondamentaux Java

Définition : Langage de programmation orienté objet, portable et interprété.

# Procédé de création d'un programme Java

source --> NomClasse.java

compilation --> javac NomClasse.java => (byte-code)

interprétation --> java NomClasse == machine virtuelle ou JVM

Versions de java

# Structure d'un fichier Java :

**Règle : une seule classe publique par fichier**

**Attention : l'indexation des paramètres fournis en ligne de commande en java commence au premier paramètre saisi**

public class NomDeClasse

{

public type idf(...) {...}

public static type idf(...) {...}

public static void main(String [] args)

{

}

}

# Conventions d'écriture

classe -> NomDeClasse

variable -> nomDeVariable

fonction -> nomDeFonction

constante -> CONSTANTE

# Variables :

2 types :

* primitifs : boolean, char, byte, short, int, long, float, double
* objets : X o; o = new X();

Remarque : classes enveloppes (wrappers)

ex : Integer it = new Integer(10);

enveloppe ---------> primitif : constructeur

primitif -------------> enveloppe : méthode

auto-boxing / unboxing de Java 7 : Integer it = 12; int n; n = it;

Exemples :

* String, StringBuffer (thread), StringBuilder(pas de thread) :
  + length()
  + charAt()
  + indexOf(" "[,pos])
  + substring(i, j)
  + +
  + toUpperCase()
  + toLowerCase()
  + equals(str)
  + compareTo(str)

*Exemple (mettre en majuscule la première lettre):*

*String str;*

*Scanner sc = new Scanner(System.in);*

*str = sc.nextLine();*

*String converted = str.substring(0,1).toUpperCase() + str.substring(1);*

* Math
* Scanner
* Arrays
* System

**Tous les packages sont contenus dans un fichier compressé rt.jar**

# Opérateurs :

+ - / \* %

++ --

< <= == >= != && || !

<< >> & | ^ ~

# Instructions

if, switch

for, while, do while

break, continue

*Astuce :*

*label;*

*for (......)*

*{*

*for(...)*

*{*

*for(...)*

*{break label;}*

*}*

*}*

# Tableaux :

* construction tableau à une dimension :

int tab[];

tab = new int[2];

* initialisés à 0
* tab.length

*ex : palindrome Java*

*char tab[] = new char[5];*

*for (int i = 0; i < tab.length / 2; i++)*

*{*

*if(tab[i] == tab[tab.length – (i – 1)])*

*cpt++;*

*}*

* autre boucle for :

for(int elem : tab)

{ s.o.p(elem);}

**pas de modification de elem possible car on travaille sur une copie**

*Programme permettant de tirer 5 nombres entiers tous différent entre 1 et 10*

*int tab[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ;*

*int fin[] = new int[5];*

*Random r = new Random();*

*int inter, ind1, ind2;*

*for (int i = 0; i < 5; i++)*

*{*

*ind1 = r.nextInt()%10;*

*ind2 = r.nextInt()%10;*

*inter = tab[ind1];*

*tab[ind1] = tab[ind2]; tab[ind2] = inter;*

*}*

* tableaux à 2 dimensions

int [][]tab;

tab = new int [2] [];

tab[0] = new int[10];

tab[1] = new int[100];

...

# Fonctions

public type idfonc(type1 p1, type2 p2)

# Exceptions

outil POO, de correction d'erreurs à l'exécution.

try

{

inst,

inst,

inst,

...

}catch(XException1 e1){...}

catch(XEception2 e2){...}

...

catch(XEception en) {...}

finally {...}

*ex: IoException, FileNotFoundException,*

*try*

*{*

*FileReader f = new FileReader("fic.txt");*

*} catch(FileNotFoundException){s.o.p("Inexistant");*

# Emission et propagation des exceptions

cas1 = exception générées par la JVM

cas2 = émission d'exception par le programmeur

*ex : try{ if(c) throw new XException(); } catch(XException e){};*

**Propagation :**

*void f() throws XException*

*{*

*if(c) throw new XException();*

*}*

*void g()*

*{ f(); }*

# Arbre d'héritage

Rappels de programmation objet

# Classes

Syntaxe :

[modif] classX{

[modif] typei id[=vi];

[modif] typej idfonc(...){...}

}

Modificateurs de classes :

* droits :
* généraux :

Modificateurs de membres :

* droits :
* généraux :

Remarque :

* blocs d'initialisation (à utiliser pour initialiser des données membres de classes anonymes)

classX{

{ins}

stativ {ins}

* constructeurs

public classX{

X(..){};

}

3 types : défaut, initialisation, trivial

Remarque :

* toute classe hérite de la classe Object
  + toString --> public String toString()
  + equals --> public boolean equals(Object)

ex:

public class Etudiant{

String nom;

int class;

public boolean equals(Object a){

if(!(a instanceof Etudiant))

return false;

Etudiant e = (Etudiant)a;

return e.nom.equals(nom) && e.class == class;

}

}

* + clone

créé une copie de l'objet

ex:

**public** **class** Student **implements** Cloneable{

Adress adress;

String name;

**int** classe;

**public** Object clone()

{

**try**{

Etudiant e = (Etudiant)**super**.clone();

e.address = (Adress)address.clone();

**return** e;

}

**catch**(CloneNotSupportedException e){

**return** **null**;

}

}

}

# Héritage

Mise en place :

[modif] class Base{}

[modif] class Derivee extends Base{}

* private dans classe de base = inaccessible dans la classe dérivée
* super appelle le plus proche dans la hiérarchie de dérivation aux paramètres correspondant
* redéfinition
* compatibilité des types

## réflexion

Class : modélise un type de données

Construction :

X o = new X();

Class c = o.getClass()

ou

Class c = X.class

ou

Class c = Class.forName("nomtype"); -> FQDN : nompackage.nomclasse

Class getSuperclass()

Method[] getMethods()

Method[] getDeclaredMethods()

Field[] getFields()

Field[] getDeclaredFields()

Constructor[] getConstructors()

Method getMethod(String name, Class[])

Class getInterface()

Exemple de fonction affichant l'arborescence d'une classe :

public void printTree(String s){

Class class;

class = Class.forName(s);

while(class != null){

sysout(class.getName());

class = class.getSuperclass();

}

}

Fonction de test d'existence de méthode :

public boolean hasMethod(String classSt, String idfonc, String type1, String type2)

{

Class c = Class.forName(classSt);

Class types[] = { Class.forName(type1), Class.forName(type2)}

if (c.getMethod(idfonc, types) == null)

return false;

return true;

}

## Remarque : Interface

Une interface Java est une classe contenant uniquement des constantes statiques et des méthodes abstraites.

Syntaxe :

public interface I extends I1, I2, ..., Ik

{

public final static csi = vi;

[public abstract]typej fj(...);

}

[modif]class X extends Y implements I

Objectifs :

* énumérations
* "héritage multiple"
* marquage, ex : Clonable, Comparable

public int compareTo(Object o){

if (note < ((Etudiant)o).note) return -1;

if(note > ((Etudiant)o).note) return 1;

return 0;

}

# V. Les classes internes

2 objectifs :

4 types :

Classes membres :

[modif] class X {

[modif] class Y{

}

}

Ex :

public class Liste{

Class Maillon{

Int val ;

Maillon suiv ;

Public void Maillon(){

Val = 0 ;

Suiv = null ;

}

Public void Maillon(int val, Maillon suiv){

Val = val ;

Suiv = suiv ;

}

}

Maillon debut ;

Public Void ajout(int val) {

Maillon Nv\_mall = new Maillon(val, debut) ;

debut = nv\_mail ;

}

## Classes anonymes :

Classes anonymes = servent à créer 1 seul objet dans l’application

* par héritage

On peut utiliser que des méthodes redéfinies

ActionListener t = new ActionListener(){

Public void actionPerformed(ActionEvent e){

Sop(“hello”);

}

}

IHM avec Java

2 bibliothèques

# Mise en place d'une fenêtre d'application

ex :

Exemple

## JFrame

public class App extends JFrame{

//définition des composants

public App(){

setTitle("Exemple");

setSize(400, 300);

…

setVisible(true);

}

}

public class Program{

public static void main(String[] args){

new App();

}

}

**Propriétés :**

* Fonctions généralistes
  + setTitle
  + setSize ou pack()
  + setVisible
  + setLocation(x, y)
  + setBounds(x, y, largeur, hauteur)
  + setResizeable(true/false)
  + setDefaultCloseOperation(JFrame.)
* Toolkit
  + Toolkit tk = Toolkit.getDefaultToolkit();
    - Dimension dim = tk.getScreenSize();
      * dim
* Thread
  + main thread
  + SwingWorkers = thread de traitement en tâche de fond
  + EventDIspatchThread (EDT) =

Runnable o = new Runnable(){

public void run(){

new App();

}

};

SwingUtilities.invokeLater(o);

# Gestion événementielle

Schéma :

# 

Event

Listener

Composant

Programmation :

XXXListener (interfaces)

JComp

ex: y.addActionListener(x);

add XXXListener

objety

comp

objetx

Classe Gestion

WindowListener

windowsClosing(WindowEvent e)

windowClosed(WindowEvent e)

windowOpened(WindowEvent e)

windowActivated(WindowEvent e)

windowDeactivated(WindowEvent e)

windowIconified(WindowEvent e)

windowDeiconified(WindowEvent e)

Remarques :

autres possibilités de programmation :

* adapteur
  + WindowAdapter

on utilise donc extends WindowAdapter au lieu de implements WindowListener pour ne pas avoir à implémenter toutes les méthodes.

* classe interne

addWindowListener(new WindowAdapter(){ public void windowClosing(…)…});

* implementation du listener sur la classe héritant de JFrame

Propriété :

public void fevent(XXXEvent e){

object o = e.getSource()

if (o == comp1)

…

}

Sur ce qui est cliquable

String s = e.getActionCommand();

# Etude des composants Swing

## Famille de composants

3 groupes

Les fonctions communes

## Structure des conteneurs lourds

### Cas du JFrame

JFrame -> JRootPane

Placement d'un composant :

Container c = getContentPane() // ou getClassPane()

JComponent jp = new JComponent(…);

c.add(jp);

|  |  |
| --- | --- |
| **Par le Programmeur** | **Par algorithme** |
| c.setLayout(null) | XXXLayout = new XXXLayout(); |
| jp.setBounds(…) ou setLocation(…) | l.setXXX(…); |
| setSize | c.setLayout(l); |
| **JAMAIS CA** | c.add(jp); |

## Principaux Layouts

### BorderLayout

|  |  |  |
| --- | --- | --- |
| C1 NORTH | | |
| C2 WEST | C3 CENTER | C4 EAST |
| C5 SOUTH | | |

Container c = getContentPane();

BorderLayout bl = **new** BorderLayout();

bl.setVgap(10);

bl.setHgap(10);

c.setLayout(bl);

c.setBackground(Color.*black*);

JButton []jbuttons = **new** JButton[5];

jbuttons[0] = **new** JButton("OK");

jbuttons[1] = **new** JButton("OK2");

jbuttons[2] = **new** JButton("OK3");

jbuttons[3] = **new** JButton("OK4");

jbuttons[4] = **new** JButton("OK5");

c.add(jbuttons[0], BorderLayout.*NORTH*);

c.add(jbuttons[1], BorderLayout.*WEST*);

c.add(jbuttons[2], BorderLayout.*CENTER*);

c.add(jbuttons[3], BorderLayout.*EAST*);

c.add(jbuttons[4], BorderLayout.*SOUTH*);

### FlowLayout

Rempli ligne après ligne, du haut vers le bas

Container c = getContentPane();

FlowLayout fl = **new** FlowLayout();

fl.setVgap(10);

fl.setHgap(10);

BorderLayout bl = **new** BorderLayout();

c.setLayout(bl);

c.setBackground(Color.*black*);

JButton []buttons2 = **new** JButton[4];

buttons2[0] = **new** JButton("1");

buttons2[1] = **new** JButton("2");

buttons2[2] = **new** JButton("3");

buttons2[3] = **new** JButton("4");

c.add(buttons2[0], BorderLayout.*NORTH*);

c.add(buttons2[1], BorderLayout.*WEST*);

c.add(buttons2[2], BorderLayout.*CENTER*);

c.add(buttons2[3], BorderLayout.*EAST*);

JPanel pan = **new** JPanel();

pan.setLayout(fl);

JButton []jbuttons = **new** JButton[5];

jbuttons[0] = **new** JButton("OK");

jbuttons[1] = **new** JButton("OK2");

jbuttons[2] = **new** JButton("OK3");

jbuttons[3] = **new** JButton("OK4");

jbuttons[4] = **new** JButton("OK5");

PrintButton pb = **new** PrintButton();

**for** (JButton b : jbuttons){

b.addActionListener(pb);

pan.add(b);

}

c.add(pan, BorderLayout.*SOUTH*);

setVisible(**true**);

### GridLayout

GridLayout gl = new GridLayout(3, 4);

c.add(composant);

Container c = getContentPane();

c.add(**new** JLabel("My Calc"), BorderLayout.*NORTH*);

c.add(**new** JTextField("Resultat"), BorderLayout.*CENTER*);

JPanel southPan = **new** JPanel();

JPanel endPan = **new** JPanel();

GridLayout gl = **new** GridLayout(4, 3);

JButton []buttons = **new** JButton[17];

southPan.setLayout(gl);

**for** (**int** i = 0; i < 10; i++){

buttons[i] = **new** JButton(""+i);

}

buttons[10] = **new** JButton(".");

buttons[11] = **new** JButton("+/-");

**for** (**int** i = 0; i < 12; i++)

southPan.add(buttons[i]);

endPan.setLayout(**new** BorderLayout());

endPan.add(southPan, BorderLayout.*CENTER*);

JPanel southPan2 = **new** JPanel();

buttons[12] = **new** JButton("+");

buttons[13] = **new** JButton("-");

buttons[14] = **new** JButton("/");

buttons[15] = **new** JButton("\*");

buttons[16] = **new** JButton("c");

**for**(**int** i = 12; i < 17; i++){

southPan2.add(buttons[i]);

}

endPan.add(southPan2, BorderLayout.*SOUTH*);

c.add(endPan, BorderLayout.*SOUTH*);

### CardLayout

CardLayout cl = new CardLayout();

Container c = getContentPane();

c.setLayout(cl);

JPanel jp1 = new JPanel();

JPanel jp2 = new JPanel();

JPanel jp3 = new JPanel();

c.add(jp1, "p1");

c.add(jp2, "p2");

c.add(jp3, "p3");

cl.next(c);

cl.previous(c);

cl.first(c);

cl.last(c);

cl.show(c, "p1");

setSize(**new** Dimension(200,110));

setMinimumSize(**new** Dimension(200,110));

/\*setMinimumSize(new Dimension(250,230));\*/

setTitle("Calculatrice");

addWindowListener(**new** windowHandler());

setDefaultCloseOperation(*DO\_NOTHING\_ON\_CLOSE*);

*cl* = **new** CardLayout();

*c* = getContentPane();

*c*.setLayout(*cl*);

ActionListener actionNext = **new** ActionListener(){

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

// **TODO** Auto-generated method stub

*cl*.next(*c*);

}

};

ActionListener actionPrev = **new** ActionListener(){

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

// **TODO** Auto-generated method stub

*cl*.previous(*c*);

}

};

JButton []actionButtons = **new** JButton[8];

JPanel []jp = **new** JPanel[4];

**int** j = 0;

**for** (**int** i = 0; i < 4; i++){

jp[i] = **new** JPanel();

jp[i].setLayout(**new** BorderLayout());

jp[i].add(**new** JLabel("fiche "+(i+1)), BorderLayout.*NORTH*);

actionButtons[j] = **new** JButton("suiv");

actionButtons[j].addActionListener(actionNext);

jp[i].add(actionButtons[j], BorderLayout.*SOUTH*);

actionButtons[j+1] = **new** JButton("prec");

actionButtons[j+1].addActionListener(actionPrev);

jp[i].add(actionButtons[j+1], BorderLayout.*CENTER*);

j+=2;

*c*.add(jp[i], "p"+i);

}

### GridBagLayout

GridBagLayout gbl = new GridBagLayout();

GridBagConstraints gbc = new GridBagConstraints();

container c = getContentPane();

JButton jb = new JButton("b1");

gbc.gridx = 0;

gbc.gridy = 0;

gbc.gridwidth = 2;

gbc.gridheight = 1;

gbc.fill = GridBacgConstraints.BOTH;

c.add(jb, gbc);

gbc.anchor = GridBackConstraint.FIRST\_LINE\_START;

### BoxLayout (appartient à Swing)

Container c = getContentPane();

BoxLayout bl = new BoxLayout(c, BoxLayout.X\_AXIS ou BoxLayout.Y\_AXIS);

c.setLayout(bl);

c.add(composant);

Box b = Box.createHorizontalBox();

b.add(composant);

## d. Autres propriétés

### Menus

2 possibilités

JMenuBar jb = new JMenuBar();

setJMenuBar(jb);

JMenu menu1 = new JMenu("Menu 1");

JMenuItem jit1 = new JMenuItem("i1");

menu1.add(jit1);

jb.add(menu1);

JPopupMenu jp = new JPopupmenu();

JMenuITem jit1 = new JMenuItem("i");

jp.add(jit);

jp.show(contentPane, x, y) 🡪 à faire dans le MouseListener

### Type d'application

Simple Document Interface (SDI) =/= Multiple Document Interface (MDI)

Pour une MDI :

**JDesktopPane** : fenêtre container dans laquelle ouvrir les nouvelles fenêtres (**JInternalFrame**)

Container c = getContentPane();

JDesktopPane jdk = new JDesktopPane();

c.add(jdk);

JInternalFrame jf = new JInternalFrame("titre", bool1, bool2, bool3, bool4);

jf.setVisible(true);

jf.setLocation(x, y);

jf.setSize(w, h);

jdk.add(jf);

DesktopManager permet de gérer toutes les fenêtres internes en même temps : jdk.getDesktopManager().methode

### Autres Propriétés

Tous les composants de Swing sont fabriqués en respectant une architecture de type MVC (Modèle, Vue, Contrôleur).

Vue : apparence

Modèle : infos stockées

Contrôleur : gestionnaire d'interactions

ex : ListModel

AbstractListModel

DefaultListModel

Programmation d'une liste :

liste statique :

String []tab = {"a", "b", "c"};

JList jl = new JList(tab);

liste dynamique

DefaultListModel dlm = new DefaultListModel();

dlm.addElement("a");

dlm.addElement("b");

dlm.addElement("c");

JList jl = new JList(dlm);

dlm.addElement("d");

# Dessin

1. On ne dessine jamais sur un conteneur lourd, mais toujours sur un intermédiaire ou sur un simple
2. objet de tracé : Graphics

2 cas de dessins :

* au chargement -> fabrication par dérivation du composant sur lequel on dessine

exemple :

class MonBeauDessin extends JPanel{

public void paintComponent(Graphics g){

super.paintComponent(g);

g.drawLine(10, 20, 40, 50);

}

}

On ne fais jamais appel directement à paintComponents, si on veut forcer on utilise repaint sur le container

* en réponse à une action de l'utilisateur ->Graphics g = composant.getGraphics()

g.draw….

g.dispose();

# Flux Java (java.io)

## Présentation

### Définition :

Un ensemble d'informations (octets, caractères) échangés séquentiellement entre un objet source et un objet de destination

* 2 types de flux

## II. Fonctionnement des classes de flux

### Noms :

|  |  |
| --- | --- |
| préfixe | suffixe |
| * flux concret = type de l'objet avec lequel on discute (ex: File, Object, ByteArray, …) * flux filtre = algorithme de modification du comportement d'un flux concret (design pattern : décorateur) | sens + nature |

### Exemples :

cas1 : écriture de réels dans un fichier

FileOutputStream fos = new FileOutputStream("fic.bin");

DataOutputStream dos = new DataOutputStream(fos);

double x = 2.25;

dos.writeDouble(x);

dos.close();

cas2: écriture bufferisée

FileOutputStream fos = new FileOutputStream("fic.bin");

BufferedOutputStream bos = new BufferedOutputStream(fos)

DataOutputStream dos = new DataOutputStream(bos);

double x = 2.25;

dos.writeDouble(x);

dos.close();

cas 3 : lecture de plusieurs fichiers

ex: ventes janvier

FileInputStream fis1 = new FileInputStream("janv.bin");

FileInputStream fis2 = new FileInputStream("fev.bin");

SequenceInputStream sis = new SequenceInputStream(fis1, fis2);

BufferedInputStream bis = new BufferedInputStream(sis);

DataInputStream dis = new DataInputStream(bi1);

x += dis.readDouble();

dis.close();

cas 4 :

byte[] tab;

ByteArrayInputStream bis = new BiteArrayInputStream(tab);

DataInputStream dis = new DataInputStream(bis)

int r = dis.readInt();

## Gestion de fichiers Java

### fichiers d'octets :

|  |  |  |
| --- | --- | --- |
| action | Lecture | Ecriture |
| ouverture | FileInputStream | FileOutputStream |
| écriture | X | writeInt, writeFloat, writeChar, writeChars / writeUTF (gère la taille des chaine) |
| lecture | BufferInputStream, DataInputStream |  |
| fermeture | close() | close() |

### fichiers de caractère

|  |  |  |
| --- | --- | --- |
| action | Lecture | Ecriture |
| ouverture | FileReader | FileWriter |
| écriture | X | write ou BufferedWriter ou PrintWriter( print, println) |
| lecture | read ou BufferedReader (readline -> null en fin de fichier) | X |
| fermeture | close() | close() |

### fichiers à accès direct (RandomAccessFile) : travail dans le des deux sens, contrôle sur la position

ouverture :

RandomAccessFile raf = new RandomAccessFile("fic.bin", "rw");

écriture :

raf.writeInt(10)

lecture:

raf.readInt() -> EOFException

position:

raf.seek(nb d'octets)

long raf.getFilePointer();

raf.length();

raf.setLength(T);

### file : sert à modéliser une entrée (virtuelle) du système de fichier

File f = new File(String file)

new File(String directory, String file)

new File(File directory, String file)

f.exists();

f.isFile();

f.isDirectory();

f.canRead();

f.canWrite();

String[] list();

File[] listFiles();

ou

File[] listFiles(FilenameFilter new class implements FilenameFilter

{

public boolean accept(File f, String s){};

});

## Sérialisation

### Définition : écriture d'objet complet dans un flux (données + empreinte de la classe)

cas d'utilisations : variables de sessions, RMI, application graphiques

### Mise en place :

class X implements Serializable{}

En cas d'héritage si une classe n'a pas implémenté Serializable les données héritées ne sont pas écrites dans le flux il faut un constructeur par défaut dans la base;

Attention si une données agrégée n'implémente pas serializable alors n a une exception

### Ecriture :

FileOutputStream fos = new FileOutputStream("ficobj.bin");

ObjectOutputStream oos = new ObjectOutputStream(fos);

X o = new X();

oos.writeObject(o);

### Lecture :

FileInputStream fis = new FileInputStream("ficobj.bin");

ObjectInputStream ois = new ObjectInputStream(fis);

X p = (X)ois.readObject();

### Personnalisation :

class X {

transient int n; (ne pas sérialiser n)

}

class X extends Y implements Serializable {

private void writeObject(ObjectOutputStream o){

int val = getMbY();

o.writeInt(val);

o.defaultWriteObject();

}

}

# Base de données en Java

JDBC (Java Database Connectivity)

## Présentation

définition : API qui offre des outils de programmation permettant d'établir la communication entre un programme Java et 1 SGBD

2 conditions :

2 packages :

Les éditeurs offrent un "Driver", classe implémentant les interfaces de java.sql et javax.sql

4 types de driver JDBC :

1. drivers bases Microsoft (pont Jdbc-Odbc) : très long en traitement (à éviter avec Java)
2. drivers mixtes Java/C++ : rapide, souvent payant
3. drivers réseau : utilisé dans le web, permet de gérer plusieurs types de bases
4. drivers 100% Java : rapide, gratuit en général

## Programmation des accès en lecture

### Préambule:

Algo : PGM 🡪 DriverManager 🡪Connection 🡪 Statement 🡪 ResultSet

Driver -> SGBD

#### Le chargement du driver

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

try{

Connection c = DriverManager.getConnection(adresse, login, mdp);

} catch(SQLException e) {sysout(e.getMessage())}

format des addresses : pseudo url = jdbc:nomMoteurBDD://adresseMoteur[:port]/[nomDB]

* jdbc:mysql://127.0.0.1/myDb

#### fabrication des requêtes

Statement st = c.createStatement();

String query = "select \* from eleves";

ResultSet rs = st.executeQuery(query); -> select, describe, show

int nbline = st.executeUpdate(query); -> insert, delete, update, use

boolean b = st.execute(query); si b vaut true ResultSet rs = st.getResultSet() sinon int nb = st.getUpdateCount()

st.executeBatch() :

st.addBatch("insert …");

st.addBatch(" update …");

st.addBatch(insert …);

int[] tab = executeBatch();

## Exploitation des ResultSet

boolean b = rs.next();

while (rs.next())

{

rs.getXXX

}

#### Propriétés

* Situation problématique

Statement st = C.createStatement();

ResultSet rs1 = st.executeQuery("…");

ResultSet rs2 = st.executeQuery("…"); 🡪 Annulation de rs1

Il faut faire deux Statements

* Gestion du curseur

déplacement quelconque + modification possible

Statement st = c.createStatement()

ResultSet.TYPE\_FOWARD\_ONLY

ResultSet.TYPE\_SCROLL\_SENSITIVE => recoit les modifications dans la base

ResultSet.TYPE\_SCROLL\_INSENSITIVE => ignore les modifications en temps réel apportées dans la base

ResultSet.CONCUR\_READ\_ONLY

ResultSet.CONCUR\_UPDATABLE

Fonctions de déplacement :

rs.next

rs.previous

rs.first

rs.last

rs.beforeFirst

rs.afterLast

rs.absolute(n°)

rs.relative(n°)

Fonctions de modification :

modification de ligne :

rs.updateXXX(nom ou num, valeur)

rs.updateRow();

suppression de ligne

rs.deleteRow();

insertion de ligne :

rs.moveTorInsertRow();

rs.updateXXX(nom ou num, valeur);

rs.insertRow();

// rs.moveToCurrentRow();

Limitation : ces fonctions ne sont utilisables que sur une table possédant une clé primaire et sans jointure

## Fonctionnalités avancées

#### les MetaData

2 cas

pour les jeux d'enregistrement :

ResultMetaData rsm = rs.getMetaData();

int nb = rsm.getColumnCount()

String st = rsm.getColumnName(int index)

String st = rsm.getColumnType(int index)

String st = rsm.getColumnLabel(int index)

Pour les base de données :

DatabaseMetaData dbm = c.getMetaData();

String st = dbm.getDriverName();

ResultSet rs = dbm.getCatalogs();

ResultSet rs = dbm.getTables();

#### les autres formes de Statement

Prepared Statement = requêtes pré compilées par le moteur de base de données

PreparedStatement ps = c.prepareStatement("SELECT \* FROM joueur WHERE num = ?");

ps.setXXX(index du point d'interrogation (commence à 1), valeur);

ResultSet rs = ps.executeQuery();

Callable Statement = requêtes d'appel à des procédures stockées

CallableStatement cs = c.prepareCall("{? = call nomprocedure(?,?,?)}");

cs.setXXX(num, valeur);

cs.execute();

//Si out : cs.registerOutParameter(num, type( == java.lang.TYPES.CHAR, …));

cs.getXXX(numéro);

ex : avec la procedure :

create procedureaffich(in p int)

begin

select \* from joueur where num = p;

end;

CallableStatement cs = c.prepareCall("{call procedureaffich(?)}");

cs.setInt(1, 20);

ResultSet rs = cs.executeQuery();