MEMO JAVA

Rev : 17/01/2025

Quelques méthodes de la classe String

```
public String chaine = "Bonjour" ;
```

endsWith()

```
chaine.endsWith("our")
//recherche si le mot se termine par le(s) caractère(s) en paramètre
```

startWith()

Idem pour le début de chaîne

charAt()

caractère positionné à l'index spécifié entre parenthèses

indexOf()

Renvoie l'index positionnel d'un caractère ou d'un mot (-1 si l'élément recherché n'appartient pas au mot)

lastIndexOf()

Idem mais partir de la fin du mot

substring()

Extraction de sous-chaîne

```
if ( chaine.substring(1,5).equals("jour") ) { … }
```

1 : index du début de la sous-chaîne, qui se termine par l'index 5.

equals() permet la comparaison de chaînes

```
String subS = chaine.substring(3,chaine.lastIndexOf("r")+1);
System.out.println("subS : "+subS);
if (subS.equals("jour") ){
    System.out.println("contient jour");
}
```

Organisation projet avec Swing + BDD + ArrayList

Application principale : JFrame / Main() : App.java

```
package view;
import java.awt.*;
import javax.swing.*;
import controler.ControlLogin;
import tools.ControlConnection;


public class App extends JFrame{

	private App app;
	private Dimension dim;

	public App(String str) { // ============================== CONSTRUCTEUR
		super(str);

		this.setDefaultCloseOperation(EXIT_ON_CLOSE);
		this.setPreferredSize(new Dimension(800,640));
		this.setLocationRelativeTo(null);
		this.setResizable(true);
		this.setVisible(true);
	}

	public static void main(String[] args) { ====================== MAIN

		App appli = new App("Application principale");
		ViewConnexion cn = new ViewConnexion(appli);
		// appli en paramètre de l'instance de ViewConnexion (vue)
		appli.getContentPane().add(cn);
		appli.pack();
		appli.centre(); // centrage de la JFrame, après le pack()

	}

	public App getApp() {
		return app;
	}
	...
	public Dimension getDim() {
		return dim;
	}

	public void centre() {

    Dimension windowsSize = this.getSize( );
    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screenSize = tk.getScreenSize();
    int x = (screenSize.width - windowsSize.width) / 2;
    int y = (screenSize.height- windowsSize.height)/ 2;
    this.setLocation(x, y);

  }
}
```

Section vue (ViewConnexion.java) :

```
package view;

import java.awt.Color;
...
import javax.swing.JTextField;

import controler.ControlLogin;

@SuppressWarnings("serial")
public class ViewConnexion extends JPanel implements ActionListener {

	JLabel labelTitre, label, label2, message1,message;
	...
	private App appli;
	JPanel panel2;

	public ViewConnexion(App app_conn) {
	// En paramètre du constructeur de la vue : la vue de l'Application principale (JFrame)

		this.appli = app_conn;
		this.setLayout(null);

		// ==========================================

		setBackground(new Color(200,200,180));
		setPreferredSize(new Dimension(600,400));

		labelTitre = new JLabel("GEDEM");
		labelTitre.setFont(new Font("Verdana", Font.PLAIN, 38));
		labelTitre.setForeground(new Color(100,100,250));

		...

		dblogin = new JTextField(20);
		dblogin.setBounds(300, 150, 300, 40);
		dblogin.addActionListener(this);

		bouton_val = new JButton("Valider");
		bouton_val.setBounds(300, 300, 100, 40);
		bouton_val.addActionListener(this);

		add(labelTitre);
		...
		add(bouton_val);


	} // Fin constructeur

	// ========================================= ACCESSEURS

	public JTextField getDblogin() {
		return dblogin;
	}


	public void setDblogin(JTextField dblogin) {
		this.dblogin = dblogin;
	}


	public JPasswordField getDbpassword() {
		return dbpassword;
	}
	...
```

```
// ========================================== GESTION DES ACTIONS

	@Override
	public void actionPerformed(ActionEvent evt) {

		if(evt.getSource()==bouton_val) {
			new ControlLogin(this,appli);
	/*
	 Lorsque le bouton est actionné, on fait appel au constructeur du controleur ControlLogin,
	 en lui passant en paramètres cette Vue ViewConnexion (grâce au mot-clé this), et la Frame actuelle
	 avec this.appli (instance de App en cours)

	*/

	}
	if(evt.getSource()==dbpassword) {
	// si on appuie sur [ENTREE] dans le champ password du formulaire --> changement de couleur
	// ça sert à rien mais c'est tellement bien...
		label.setForeground(Color.RED);
		label.setFont(new Font("serif", Font.PLAIN, 42));
		setBackground(new Color(200,130,220));
		this.appli.getContentPane().add(getMessage2());

		}
}

}
```

Section Contrôleur (ControlLogin.java) :

```
package controler;

import java.sql.*;
import java.util.ArrayList;
...
import view.ViewListeFormation;

public class ControlLogin {

	ControlConnection cc = new ControlConnection();
	ArrayList<Login> listeLogin;
	private ViewListeStagiaire vls;
	private ViewListeFormation vlf;
	private ViewFormateur vform;

	private int flag;

	// /** CONSTRUCTEUR ========================
	/**
	 *
	 * @param cn1
	 * @param app2
	 */
	public ControlLogin(ViewConnexion cn,App appli) {
	// ViewConnexion et App en paramètres, pour pouvoir échanger
	// des infos entre les vues et le controleur -   cn : JPanel                    appli : JFrame
	//this.app = appli;

		JTextField loginres = cn.getDblogin();
	// recupère dblogin avec le getter de la classe ViewConnexion
```

```
                JTextField passwordres = cn.getDbpassword();
                String loginresStr = loginres.getText();
        // on transforme l'objet TextField en String, pour le passer en paramètre de la requête SQL
                String passwordresStr = passwordres.getText();
                this.listeLogin = new ArrayList<Login>();
                this.flag=0;

        try {


String req2="SELECT dblogin,dbpassword,dbrole FROM login WHERE dblogin='"+loginresStr+"' AND
dbpassword='"+passwordresStr+"' ;";
 //la requete récupère les infos dans la BDD en sélectionnant les champs qui correspondent
// aux logins, pasword et role de la vue "ViewConnexion"
//String req2="SELECT "
ResultSet rs2 = cc.getSt().executeQuery(req2);
// on fait un get pour recup le statement de ControlerConnection
if (rs2.next()) { // si rs2 contient une donnée, donc si login et mot de passe correspondent

        System.out.println("LOGIN ET MOT DE PASSE OK");
//================================================== Traitement des resultats
                if (rs2.getString(3).equals("Stagiaire")) {
                        System.out.println("Vous etes stagiaire.");
                        vls = new ViewListeStagiaire(appli);
                        appli.getContentPane().add(vls);
                }
                else if (rs2.getString(3).equals("Administrateur")) {
                        ...
                        }
                        appli.getContentPane().repaint();
                        appli.getContentPane().revalidate();

                                        }
                else { // Affichage console si login et password ne "match" pas avec la BDD
                        System.out.println("ACCESS REFUSE console");

                        appli.getContentPane().add(cn.getMessage2());
                        appli.getContentPane().repaint();
                        appli.getContentPane().revalidate();
                }
        // Affichage
                for (int i=0;i<listeLogin.size(); i++) {
                System.out.print(listeLogin.get(i).getDbLogin()+"\t");
                System.out.print(listeLogin.get(i).getDbPassword()+"\t");
                System.out.print(listeLogin.get(i).getDbRole()+"\n");

                        }

                }
                catch (SQLException efc){
                        System.out.println("Erreur ControlLogin");
                        efc.printStackTrace();
                        }
}

        public ViewListeStagiaire getVls() {
                return vls;
        }

        public void setVls(ViewListeStagiaire vls) {
                this.vls = vls;
        }


}
```

Généralités avec SWING

JWindow : pour les splash screen (pas de bordures / fermeture…)
JFrame : fenêtre principale, peut avoir un menu, titre, fermeture, redimensionnement…
JDialog : boîte de dialogue / fenêtre avec boutons fermeture,réduc,…

Basique

```
//creation de la fenêtre principale
        JFrame frame = new JFrame("vbasic");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //creation du conteneur
        JPanel conteneur = new JPanel();
        frame.add(conteneur);
        conteneur.setLayout(new BoxLayout(conteneur, BoxLayout.PAGE_AXIS));

        //creation du conteneur de champ et des boutons
        JPanel champ = new JPanel();
        conteneur.add(champ);
        champ.setLayout(new BoxLayout(champ, BoxLayout.PAGE_AXIS));
        JPanel boutonPane = new JPanel();
        conteneur.add(boutonPane);
        boutonPane.setLayout(new BoxLayout(boutonPane, BoxLayout.LINE_AXIS));
```

BoxLayout : chaine les blocs en les alignant soit horizontalement (X_AXIS), soit verticalement (Y_AXIS).

MENUBAR

```
private JMenuBar menu = new JMenuBar();
private JMenu fichier = new JMenu("Fichier");
private JMenu édition = new JMenu("Edition");
private JPanel panneau = new JPanel();
private JTextField positions = new JTextField(" Lignes : 1 Colonnes : 1");
private JTextField lireSélection = new JTextField(24);
private ZoneEdition éditeur = new ZoneEdition();

private void menu() {
     setJMenuBar(menu);
     menu.add(fichier);
     fichier.add(actionNouveau);
     fichier.add(actionOuvrir);
     fichier.add(actionEnregistrer);
     menu.add(édition);
     JMenuItem sélection = new JMenuItem("Tout sélectionner");
     édition.add(sélection);
     sélection.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            éditeur.selectAll();
        }
     });
     édition.add(actionCopier);
     édition.add(actionCouper);
     édition.add(actionColler);
     final JCheckBoxMenuItem lectureSeule = new JCheckBoxMenuItem("LectureSeule");
     édition.add(lectureSeule);
     lectureSeule.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          éditeur.setEditable(!lectureSeule.isSelected());
        }
     });
```

FONT

```
JTextArea txt = new JTextArea();
Font font = new Font("Verdana", Font.BOLD, 12);
txt.setFont(font);
txt.setForeground(Color.BLUE);
```

Well, once you have your font, you can invoke `deriveFont`. For example,

```
helvetica = helvetica.deriveFont(Font.BOLD, 12f);
```

Changes the font's style to bold and its size to 12 points.

IMAGES

Chargement d'une image (type ImageIcon) :

ex1 :

```
URL url = new URL("http://today.java.net/jag/bio/JagHeadshot-small.jpg");
BufferedImage image = ImageIO.read(url);
Icon icon = new ImageIcon(image);
JLabel label = new JLabel(icon);
```

ex2 :

```
String monFichierImage = "src/package_test/Images/peppa.png";
BufferedImage peppa = ImageIO.read(new File(monFichierImage));
ImageIcon imagePeppa = new ImageIcon(peppa);
JLabel labelShoot = new JLabel(imagePeppa);
labelShoot.setOpaque(true);
labelShoot.setBackground(new Color(120,90,90));
jp.add(labelShoot); // jp est une instance de JPanel
```

Modifier la taille d'une image (type ImageIcon) :

```
ImageIcon imageIcon = new ImageIcon("./img/imageName.png");
// load the image to a imageIcon

Image image = imageIcon.getImage(); // transform it

Image newimg = image.getScaledInstance(120, 120,  java.awt.Image.SCALE_SMOOTH);
// scale it the smooth way

imageIcon = new ImageIcon(newimg);  // transform it back
```

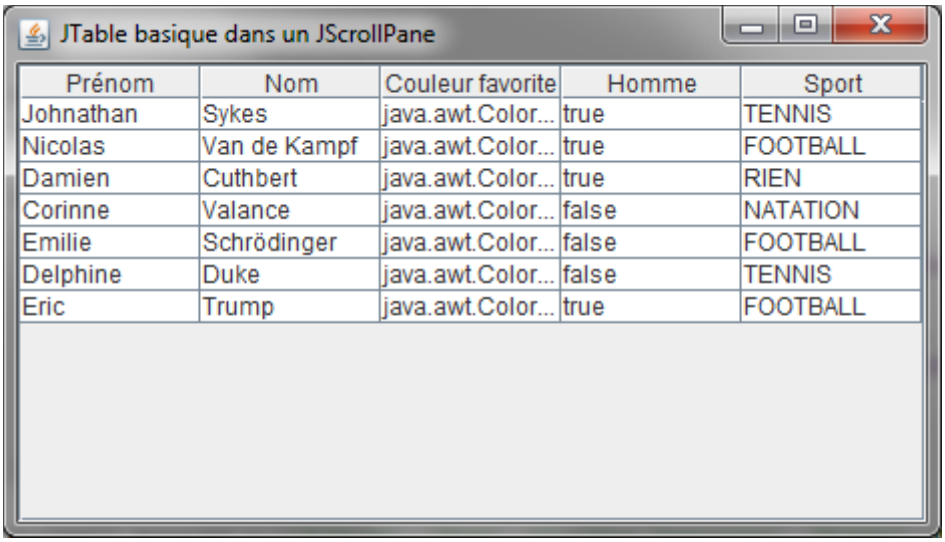
Obtenir les dimensions d'une image :

à partir de la classe BufferedImage => utilisation des méthodes getWidth() et getHeight()

```
BufferedImage img = ImageIO.read(new File("filename.jpg"));

        System.out.println("Width = " + img.getWidth());
        System.out.println("Height = " + img.getHeight());
```

Jtable :

```
public class JTableBasiqueAvecScrollPane extends JFrame {
    public JTableBasiqueAvecScrollPane() {
        super();

        setTitle("JTable basique dans un JScrollPane");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Object[][] donnees = {
                {"Johnathan", "Sykes", Color.red, true, Sport.TENNIS},
                {"Nicolas", "Van de Kampf", Color.black, true, Sport.FOOTBALL},

        };

        String[] entetes = {"Prénom", "Nom", "Couleur favorite", "Homme", "Sport"};
        JTable tableau = new JTable(donnees, entetes);
        getContentPane().add(new JScrollPane(tableau), BorderLayout.CENTER);
        pack();
    }

    public static void main(String[] args) {
        new JTableBasiqueAvecScrollPane().setVisible(true);
    }
}
```



Constructeurs :

– JTable(Object[][] données, Object[] nomsColonnes)
– JTable(Vector données, Vector nomsColonnes)
Données est un vecteur de vecteurs d'objets

JTable : exemple du cas de base

```
import java.util.Vector;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
public class Base extends JFrame {
public Base() {
super("table de base");
// Intitulés des colonnes
Vector<String> colonnes = new Vector<String>();
colonnes.add("Nom"); colonnes.add("Prénom"); colonnes.add("Age");
// Contenu de la table
Vector<Vector<String>> donnees = new Vector<Vector<String>>();
Vector<String> donnee1 = new Vector<String>();
donnee1.add("fournier"); donnee1.add("domique"); donnee1.add("35"); donnees.add(donnee1);
Vector<String> donnee2 = new Vector<String>();
donnee2.add("amanton");donnee2.add("laurent"); donnee2.add("20"); donnees.add(donnee2);
// Création de la table
JTable table = new JTable(donnees, colonnes);
JScrollPane panneau; panneau = new JScrollPane(table);
table.setFillsViewportHeight(true);
add(panneau);
setDefaultCloseOperation(EXIT_ON_CLOSE); pack(); setVisible(true);
}
public static void main(String [] args) {
Base fenetre = new Base();
}}
```

```
================================ AUTRE EXEMPLE

public class JtableVect1 extends JFrame{


 public static void main(String args[])
 {
 JtableVect1 jv1 = new JtableVect1();
 jv1.petitTableau();
 }

 public void petitTableau() {
          //crée un frame

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            //ligne 1
            Vector<String> row1 = new Vector<String>();
            row1.addElement("A");
            row1.addElement("B");
            row1.addElement("C");

            //ligne 2
            Vector<String> row2 = new Vector<String>();
            row2.addElement("X");
            row2.addElement("Y");
            row2.addElement("Zoo");

            //données pour JTable(ligne 1 + ligne 2)
            Vector<Vector> data = new Vector<Vector>();
            data.addElement(row1);
            data.addElement(row2);

         // my rajout

           ListIterator<Vector> lidata = data.listIterator();
           while (lidata.hasNext()) {
                    System.out.println(lidata.next());
           }

            //Header de JTable
            Vector<String> columns = new Vector<String>();
            columns.addElement("Colonne 1");
            columns.addElement("Colonne 2");
            columns.addElement("Colonne 3");
            JTable table = new JTable(data, columns);

            JScrollPane scroll = new JScrollPane(table);
            add(scroll, BorderLayout.CENTER);
            setSize(300, 150);
            setVisible(true);

 }
}
```

COLLECTIONS

Il y a des opérations réalisées sur un seul objet ou bien sur une collection (un ensemble d'objets).

add (remove) permet d'ajouter (resp. de retirer) un élément. Quand à **addAll** (**removeAll**) permet d'ajouter (resp. de retirer même si les éléments sont dupliqués dans la collection originale) une collection.

contains (**containsAll**) permet de vérifier si un objet (resp. les éléments d'une collection) est présent dans la collection.

size, **isEmpty** et **clear**, permettent respectivement de donner la taille de la collection, de vérifier si la collection est vide et finalement d'effacer le contenu de la collection.

retainsAll se comporte comme le résultat de l'intersection de deux ensembles. Si A={1,2,5,8} et B={3,8} alors A = {8}.

equals permet de tester si deux objets sont égaux.
hashCode retourne le code de hachage calculé pour la collection.
toArray retourne les éléments de la collection sous le format d'un tableau.
toArray(Object a[]) permet de préciser le type du tableau à retourner. Si le tableau est grand les éléments sont rangés dans ce tableau, sinon un nouveau tableau est crée pour recevoir les éléments de la collection.

L'interface collection est dotée d'une instance d'une classe qui implante l'interface Iterator. C'est l'outil utilisé pour parcourir une collection. L'interface Iterator contient ce qui suit:

```
public interface Iterator {
boolean hasNext();
Object next();
void remove(); // Optional
}
```

hasNext permet de vérifier s'il y a un élément qui suit.
next permet de pointer l'élément suivant.
remove permet de retirer l'élément courant.

```
Collection collection = ...;
Iterator iterator = collection.iterator();
while (iterator.hasNext()) {
Object element = iterator.next();
if (removalCheck(element)) {
iterator.remove();
}
}
```

Les collections vues comme des ensembles réalisent les 3 opérations mathématiques sur des ensembles:
union: add et addAll
intersection: retainAll
différence: remove et removeAll

LIST

Les méthodes de l'interface List permettent d'agir sur un élément se trouvant à un index donné ou bien un ensemble d'éléments à partir d'un index donné dans la liste.
get (**remove**) retourne (resp. retirer) l'élément se trouvant à la position index.
set (**add** & **addAll**) modifie (resp. ajouter) l'élément (resp. un seul ou une collection) se trouvant à la position index.
indexOf (**lastIndexOf**) recherche si un objet se trouve dans la liste et retourner son (resp. son dernier) index.
subList permet de créer un sous liste d'une liste.

Pour parcourir une liste, il a été défini un itérateur spécialement pour la liste.

```
public interface ListIterator extends Iterator {
boolean hasNext();
Object next();
boolean hasPrevious();
Object previous();
int nextIndex();
int previousIndex();
void remove(); // Optional
void set(Object o); // Optional
void add(Object o); // Optional
}
```

permet donc de parcourir la liste dans les deux directions et de modifier un élément (set) ou d'ajouter un nouveau élément.

```
List list = ...;
ListIterator iterator = list.listIterator(list.size());
while (iterator.hasPrevious()) {
Object element = iterator.previous();
// traitement d'un élément
}
```

hasNext permet de vérifier s'il y a un élément qui suit.
next permet de pointer l'élément courant.
nextIndex retourne l'index de l'élément courant.

Pour les sous listes, elles sont extraites des listes de fromIndex (inclus) à toIndex (non inclus). Tout changement sur les sous listes affecte la liste de base, et l'inverse provoque un état indéfini s'il y a accès à la sous liste.

VECTOR

Classe d'objets synchronisés, contrairement aux ArrayList (l'appel d'une méthode vector doit être terminée avant d'en invoquer une nouvelle) – **Tableaux dynamiques**

```
Vector vec1 = new Vector (10) ;
```
10 correspond à la capacité initiale du Tableau, soit 10 par défaut.

Méthodes associées :

```
int lastIndexOf Objectelem
```
Returns the index of the last occurrence of the specified object in this vector.

```
void addElementObjectobj
```
Adds the specified component to the end of this vector, increasing its size by one

Vector Example

```
import java.util.Enumeration;
import java.util.Vector;
// Store Strings in a vector
Vector<String> v = new Vector<String>();
v.add("Football");
v.add("Basketball");
v.add("Tennis");
// Get first and second elements in the vector
String firstSport = v.firstElement();
String secondSport = v.get(1);
// Print all contents of the vector
Enumeration<String> sports = v.elements();
while (sports.hasMoreElements()) {
String sport = sports.nextElement();
System.out.println(sport);
}
// Remove Tennis
v.remove(2);
// Remove all
v.removeAllElements();
```

Autre possibilité sans Enumeration :

```
for (int i=0; i<v.size(); i++) {
System.out.println(v.elementAt(i));
}
```

========== Mon exemple

```
public String afficheListe() {
            Integer[] aList = {1,2,3,4,5};
            Vector vec1 = new Vector();
            Collections.addAll(vec1,aList);
            String affListe="";
            /*
            Enumeration<Integer> enum1 = vec1.elements();
            while (enum1.hasMoreElements()) {
            affListe += enum1.nextElement().toString()+" ";
            System.out.println(affListe);
            }
            */
            for (int i=0; i<vec1.size();i++) {
                  affListe += vec1.get(i);
            }

            return affListe;
      }
```

utilisation de addAll() :

```
import java.util.*;
1.public class VectorAddAllExample2 {
2.    public static void main(String arg[]) {
3.        //Create a first empty vector
4.          Vector<String> vec1 = new Vector<>(4);
5.          //Add elements in the first vector
6.          vec1.add("E");
7.          vec1.add("F");
8.          vec1.add("G");
9.          vec1.add("H");
10.          //Create a second empty vector
11.          Vector<String> vec2 = new Vector<>(4);
12.          //Add elements in the second vector
13.          vec2.add("A");
14.          vec2.add("B");
15.          vec2.add("C");
16.          vec2.add("D");
17.          //Add elements of the vec2 at 1st element position in the vec1
18.          vec1.addAll(0, vec2);
19.          //Printing the final vector after appending
20.          System.out.println("Final vector list: "+vec1);
21.          }
22.}
```

CONVERSIONS DE TYPE

Variable hauteurFenetre déclarée en int :

```
// pour retourner le type de la variable : attention, comme int est un type primitif, il faut d'abord le convertir
// en un objet Integer à l'aide de Integer.valueOf(), et appliquer ensuite les méthodes getClass().getSimpleName()
// pour obtenir le type

System.out.println("Type de hauteurFenetre : "+Integer.valueOf(hauteurFenetre).getClass().getSimpleName());
```

Mais du coup ça renvoie « Integer »…

ArrayList

```java
// Java code to demonstrate the concept of
// array of ArrayList

import java.util.*;
public class Arraylist {
    public static void main(String[] args)
    {
        int n = 3;

        // Here aList is an ArrayList of ArrayLists
        ArrayList<ArrayList<Integer> > aList =
                  new ArrayList<ArrayList<Integer> >(n);

        // Create n lists one by one and append to the
        // master list (ArrayList of ArrayList)
        ArrayList<Integer> a1 = new ArrayList<Integer>();
        a1.add(1);
        a1.add(2);
        aList.add(a1);

        ArrayList<Integer> a2 = new ArrayList<Integer>();
        a2.add(5);
        aList.add(a2);

        ArrayList<Integer> a3 = new ArrayList<Integer>();
        a3.add(10);
        a3.add(20);
        a3.add(30);
        aList.add(a3);

        for (int i = 0; i < aList.size(); i++) {
            for (int j = 0; j < aList.get(i).size(); j++) {
                System.out.print(aList.get(i).get(j) + " ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
1 2
5
10 20 30
```

Gestion d'évènements

ActionListener

```
JButton b_ok = new JButton("OK");
b_ok.addActionListener(new ActionListener()
{
        @Override
        public void actionPerformed(ActionEvent e)
        {
                // ... Action du bouton OK ici
                System.out.println("Clic du bouton OK");
        }
});
```

```
bouton_val = new JButton("Valider");
            bouton_val.setBounds(300, 300, 100, 40);
            bouton_val.addActionListener(this);
bouton2 = new JButton("Annuler");
            bouton2.setBounds(300, 300, 100, 40);
            bouton2.addActionListener(this);
…

public void actionPerformed(ActionEvent evt) {

            if(evt.getSource()==bouton_val) {
                  new ControlLogin(this,appli);
            }

            if(evt.getSource()==bouton2) {
                  System.out.Println("bouton 2 appuyé");
            }
```

ItemListener

```
public class ItemListenerExample implements ItemListener{
    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
        label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        checkBox1 = new Checkbox("C++");
        checkBox1.setBounds(100,100, 50,50);
        checkBox2 = new Checkbox("Java");
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        checkBox1.addItemListener(this);
        checkBox2.addItemListener(this);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
    public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==checkBox1)
            label.setText("C++ Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
        if(e.getSource()==checkBox2)
        label.setText("Java Checkbox: "
        + (e.getStateChange()==1?"checked":"unchecked"));
     }
public static void main(String args[])
{
    new ItemListenerExample();
}
}
```

Possibilité d'activer ActionListener et ItemListener dans la même classe :

```
public class ViewConnexion extends JPanel implements ActionListener,ItemListener {
```

KeyListener

```
public class ViewConnexion extends JPanel implements ActionListener,ItemListener, KeyListener {
...
typingArea = new JTextField(20);
typingArea.addKeyListener(this);
...
/** Handle the key pressed event from the text field. */
    public void keyTyped (KeyEvent e) {
        System.out.println("Touche pressée");

        public void keyPressed (KeyEvent e) {
                System.out.println("Touche pressée");

        public void keyReleased (KeyEvent e) {
                System.out.println("Touche pressée");
        }
```

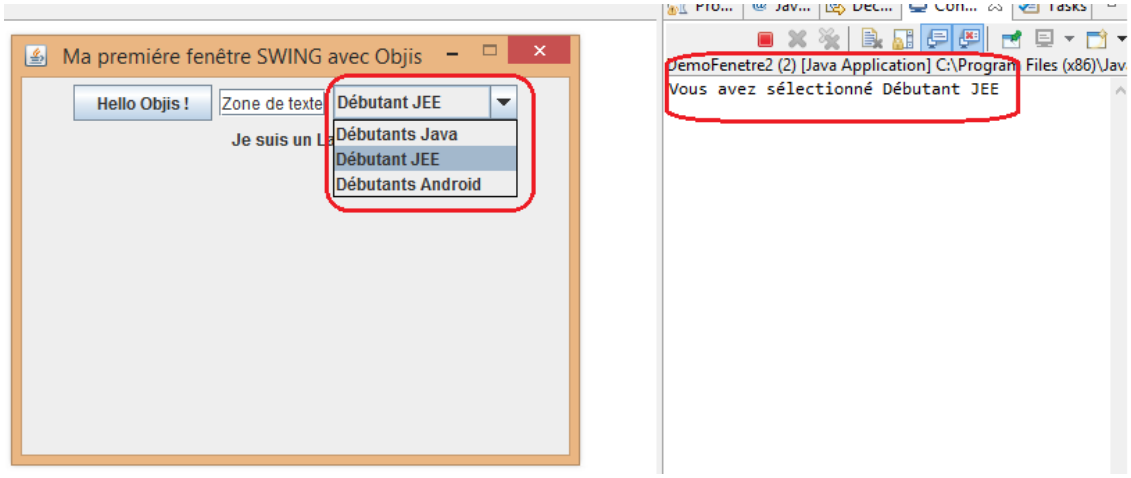
Fenetre2.java

```
56              @Override
57              public void keyReleased(KeyEvent e) {
58                  // TODO Auto-generated method stub
59                  JTextField textField = (JTextField) e.getSource();
60                  String text = textField.getText();
61                  textField.setText(text.toUpperCase());
62              }
63
64              @Override
65              public void keyPressed(KeyEvent e) {
66                  // TODO Auto-generated method stub
67
68              }
69          });
70          // ajoute zone de texte sur le conteneur
71          conteneur.add(texte);
72          // ajout d'éléments dans notre combobox
73          combo.addItem("Débutants Java");
74          combo.addItem("Débutant JEE");
75          combo.addItem("Débutants Android");
76          // ajout listener sur jcombobox
77          combo.addItemListener(new ItemListener() {
78
79              @Override
80              public void itemStateChanged(ItemEvent e) {
81                  // TODO Auto-generated method stub
82                  if (e.getStateChange() == ItemEvent.SELECTED) {
83                  System.out.println("Vous avez sélectionné " + e.getItem());
84                  }
85              }
86          });
87          // ajout combobox à notre conteneur
88          conteneur.add(combo);
89          // ajouter le label
90          conteneur.add(label);
91          setContentPane(conteneur);
92      }
93  }
```

Nous avons utilisé l'interface *ItemListener* avec sa méthode itemStateChanged qui nous permet d'intercepter les changements d'items au niveau du combobox .

```
@Override
public void itemStateChanged(ItemEvent e) {
    // TODO Auto-generated method stub
    if (e.getStateChange() == ItemEvent.SELECTED) {
    System.out.println("Vous avez sélectionné " + e.getItem());
    }
}
```

En sortie, nous affichons l'item sélectionné .
Le résultat donne l'écran ci-dessous .



Autre possibilité pour le KeyListener (évènement touche du clavier) :

importation

```
import java.awt.event.KeyListener;
```

Déclaration de la classe

```
public class ViewPanel2 extends JPanel implements ActionListener, KeyListener {
…
Méthode pour appliquer le Listener sur un objet :
UnJTextField.addKeyListener(this);
...
}
```

Nouvelle méthode pour la gestion de l'évènement :

```
public void keyPressed(KeyEvent e) {
		// TODO Auto-generated method stub
		if (e.getKeyCode() == KeyEvent.VK_ENTER) {
			Ccalc ctrl1 = new Ccalc(laVm, this);
			System.out.println("Evenement source [Enter] key pressed : calcul résultat");
		}
```

JDBC – Bases de données

Méthode statique **forName** de la classe **Class**

Pour Mysql / MariaDB :

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

Installation du pilote sous Eclipse :
Clic droit sur le nom du projet > Propriétés > Onglet Librairies > Add external JARs…

Imports nécessaires : `import java.sql.* ;`

Exceptions à gérer :

`ClassNotFoundException` : exception pour pilote introuvable
`SQLException` : exception pour les autres problèmes liés au driver. (erreurs de Login, requêtes SQL,…)

Etablissement d'une connexion :

```
try {
String strClassName = "com.mysql.cj.jdbc.Driver " ;
String dbName = "labase" ;
String login = "user_x" ;
String motDePasse = "user_pass" ;
String strUrl = "jdbc:mysql://localhost:3306/"+dbName+"?useSSL=false&timeZoneserver=UTC";

Class.forName(strClassName) ;
Connection conn = DriverManager.getConnection(strUrl, login, motDePasse) ;
// … Opérations
conn.close() ;
}
catch (ClassNotFoundException e) {
System.out.println("Erreur de driver JDBC") ;
e.printStackTrace() ;
}
```

Requête SQL : utilise l'interface **Statement**
si la requête renvoie une réponse (cas de insert, update, delete) : on utilise **executeUpdate()** sur le statement
Si elle renvoit un ensemble de résultat (tuple) : on utilise la méthode **executeQuery()** du statement

```
try {
String strClassName = "com.mysql.cj.jdbc.Driver" ;
String dbName = "labase" ; String login = "user_x" ;
String motDePasse = "user_pass" ;
String strUrl = "jdbc:mysql://localhost:3306/ "+dbName+"?useSSL=false&timeZoneserver=UTC";
Class.forName(strClassName);
Connection conn = DriverManager.getConnection(strUrl, login, motDePasse);
Statement st = conn.createStatement() ;
String requete = "INSERT INTO laTable (nom,prenom) VALUES ('Richard','Pierre');";
st.executeUpdate(requete);
… suite du code...
st.close(); conn.close();
}
catch (ClassNotFoundException e) {
System.out.println(" Erreur de driver JDBC ");
e.printStackTrace();
}
catch (SQLException e) {...}
```

```
try {
String strClassName = "com.mysql.cj.jdbc.Driver";
String dbName = "labase" ; String login = "user_x" ;
String motDePasse = "user_pass" ;
String strUrl = "jdbc:mysql://localhost:3306/ "+dbName+"?useSSL=false&timeZoneserver=UTC" ;
Class.forName(strClassName) ;
Connection conn = DriverManager.getConnection(strUrl, login, motDePasse) ;
Statement st = conn.createStatement() ;
String requete = " SELECT * FROM laTable WHERE user='toto'; " ;
ResultSet rs = st.executeQuery(requete) ;
while (rs.next()) {… suite du code...
st.close() ; conn.close() ;
}
catch (ClassNotFoundException e) {
System.out.println(" Erreur de driver JDBC ") ;
e.printStackTrace() ;
}
catch (SQLException e) {...}
```

MANIPULER LE RÉSULTAT

On peut identifier chaque colonne de la BDD, soit par son nom, soit par son résultat.

```
String strQuery= "SELECT * FROM T_users ;" ;
ResultSet rs = st.executeQuery(strQuery) ;
while (rs.next()) {

System.out.print("Id["+rs.getInt(1)+"]"+rs.getString(2)+"["+rs.getString("password")+"]")
;
                    }
rs.close() ;
```

Autres opérations

Requêtes préparées : `java.sql.PreparedStatement`
Procédures stockées : `java.sql.CallableStatement`
Accès aux métadonnées : `java.sql.DatabaseMetaData`

Metadata

```
try {
ResultSetMetaData rsmd = rs.getMetaData() ;
for (int i=1 ; i<= rsmd.getColumnCount() ; ++i) {
        enTeteColonne.addElement(rsmd.getColumnName(i)) ;
}
```

Requêtes préparées

```
String req="SELECT * FROM employe WHERE age > ?" ;
// création d'un PreparedStatement
PrepareStatement pst = conn.prepareStatement(req) ;
// requête SQL compilée par le SGBD, passage des paramètres par setXxx
pst.setInt(1,55) ; // => age>55
// execution de la requête
```

Après les try/catch d’une connexion JDBC, on peut aussi faire un "finally" pour fermer les objets ouverts :

```
finally {
    if ( connexion != null )
        try {
            /* Fermeture de la connexion */
            connexion.close();
        } catch ( SQLException ignore ) {
         /* Si une erreur survient lors de la fermeture, il suffit de l'ignorer. */
        }
      if ( statement != null )
        try {
            /* Fermeture de la connexion */
            statement.close();
        } catch ( SQLException ignore ) {

        }
      if ( resultset != null )
        try {
            /* Fermeture de la connexion */
            resultset.close();
        } catch ( SQLException ignore ) {

        }
}
```

REGEX – Expressions régulières

```
String bloc_recup_t = bloc_recup.replaceAll(" ",""); // suppression de tous les espaces
String valNum_temp="";
for (int i=0; i<bloc_recup_t.length(); i++) {
            if (String.valueOf(bloc_recup_t.charAt(i)).matches("[0-9\\.]")) {
                  valNum_temp += bloc_recup_t.charAt(i);
            }
// pour chaque caractère de la chaîne correspondant à une valeur numérique
// On concatène, le résultat étant affecté à valNum_temp.
```

matches() prend une expression régulière en paramètre.
Le double backslash permet d'échapper un caractère possédant une signification particulière, comme le point.

Autre exemple de manipulation d'expressions régulières faisant appel aux classes Pattern et Matcher :

```
import java.util.regex.*;
import java.util.Scanner;
import java.lang.Boolean;

public class MonRegex {

private String lachaine = "";
private boolean boolMatch;

// Constructeur
public MonRegex(String chaine){
      this.lachaine = chaine;
      Pattern pat = Pattern.compile("a.+");
      Matcher m = pat.matcher(this.lachaine);
      this.boolMatch = m.matches();
      }

public static void main(String [] args) {

      Scanner lectureC = new Scanner(System.in);
      System.out.println("Entrez le chaine : ");
      String varchaine = lectureC.nextLine();
      MonRegex mr1 = new MonRegex(varchaine);
      System.out.println("Valeur du booleen Match : " + mr1.getBoolMatch());

      }
}
//getter
public boolean getBoolMatch(){
return boolMatch;
}
```