جامعة الحسن الأول
UNIVERSITÉ HASSAN 1ER

UNIVERSITE HASSAN 1ER
ÉCOLE NATIONALE DES SCIENCES APPLIQUEES – BERRECHID

ENSA Ecole Nationale des Sciences Appliquées BERRECHID

DIAPORAMAS

(SUPPORT ELECTRONIQUE DE COURS)

INTERFACES HOMME-MACHINE EN JAVA

ROFESSEUR : LAHCEN MOUMOUN

DEPARTEMENT GENIE INFORMATIQUE & MATHEMATIQUES

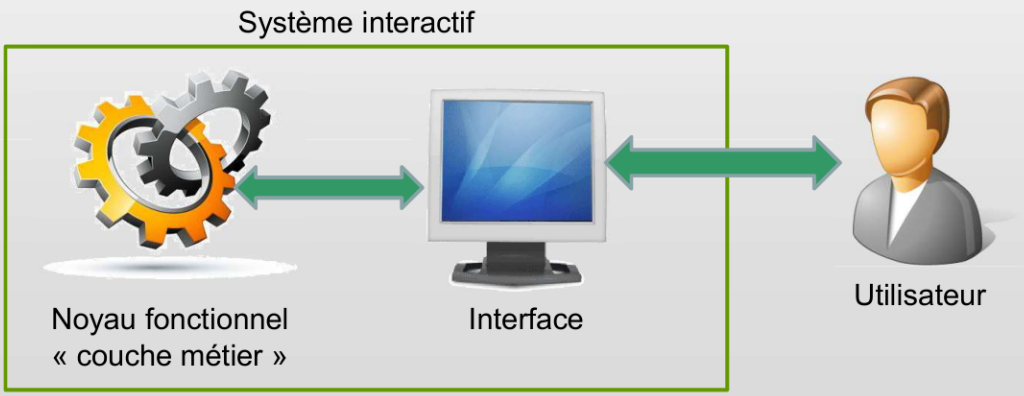
Interface Homme-Machine en JAVA

Chapitre 1 : Concept de base des IHMs en JAVA

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

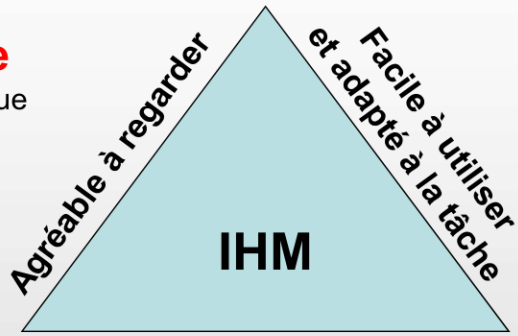
Interface Homme Machine (IHM)

- L'Interface (ou Interaction) Homme Machine est un ensemble des aspects de la conception, de l'implémentation et de l'évaluation des systèmes informatiques interactifs.
- Une Interface Utilisateur Graphique (GUI: Graphical User Interface) est une représentation picturale qui utilise généralement un mode d'interaction gestuel.



IHM: Ergonomie, Graphisme et Technologie

Graphisme
Dimension artistique



Ergonomie
Prise en compte du facteur humain

Facile à implémenter et à maintenir

Technologie
Contraintes techniques

Architecture des interfaces graphiques

Une interface graphique est basée sur une architecture comportant :

- Un système de fenêtrage (windowing system) qui fournit aux applications un espace (graphique) d'accueil propre.
- Un gestionnaire de fenêtres (window manager) qui gère l'ensemble des fenêtres et uniformise leur manipulation
- Des outils et bibliothèques d'objets (tools & API) qui uniformisent la représentation visuelle des objets et permettent de rendre la plupart des services généraux attendus.

IHM en JAVA

Java permet de programmer des interfaces graphiques riches et portables. il existe 2 grand grandes bibliothèques de classes de composants graphiques utilisables en Java :

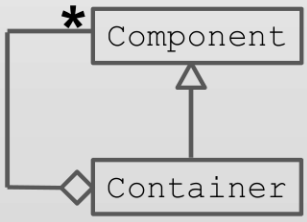
- **AWT** (abréviation de Abstract Window ToolKit) : la bibliothèque la plus ancienne qui s'appuie sur le toolkit (boîte à outils) graphique de l'OS.
- **Swing** (mot anglais qui décrit un mouvement de balancement) : le remplaçant officiel de AWT, qui comporte des composants écrit complètement avec java et sont indépendant de l'OS.

Les classes AWT et Swing font partie d'un ensemble plus vaste de classes JAVA appelé Java Fondation Classes (JFC).

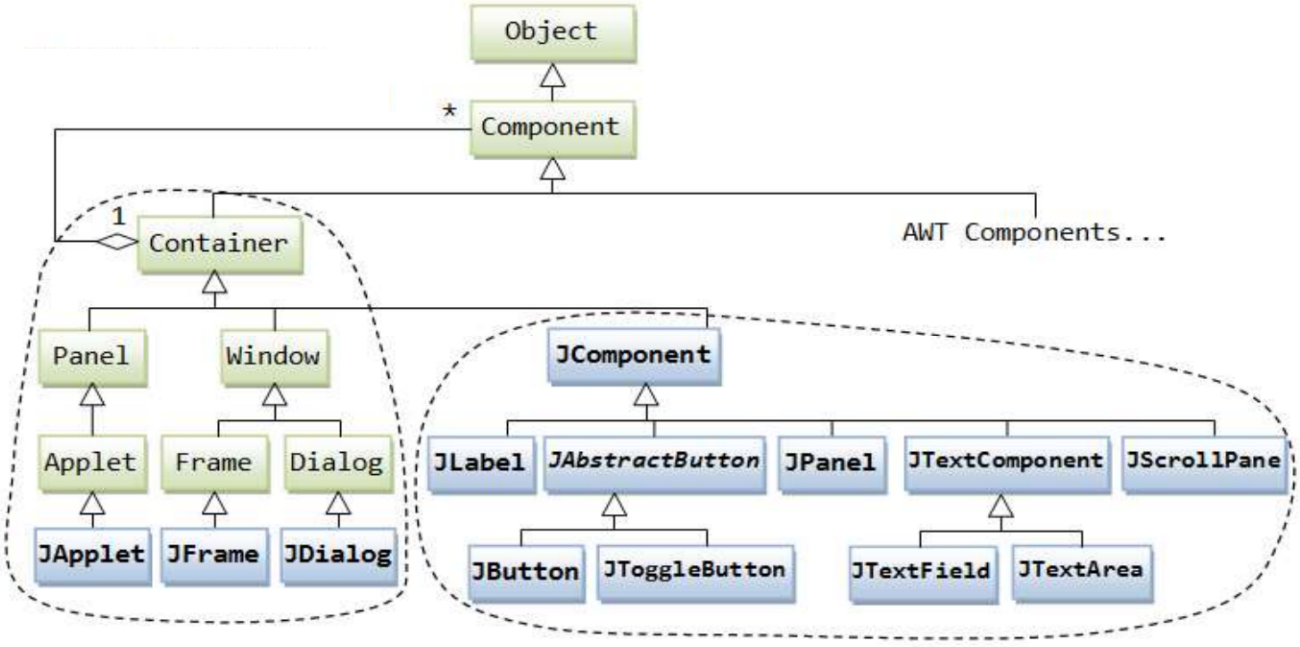
Notion de composants et de conteneurs

Il existe deux principaux types de composants susceptibles d'intervenir dans une interface graphique :

- Les conteneurs qui sont destinés à contenir d'autres composants, comme par exemple les fenêtres;
- Les composants atomiques qui sont des composants qui ne peuvent pas en contenir d'autres (les boutons par exemple).

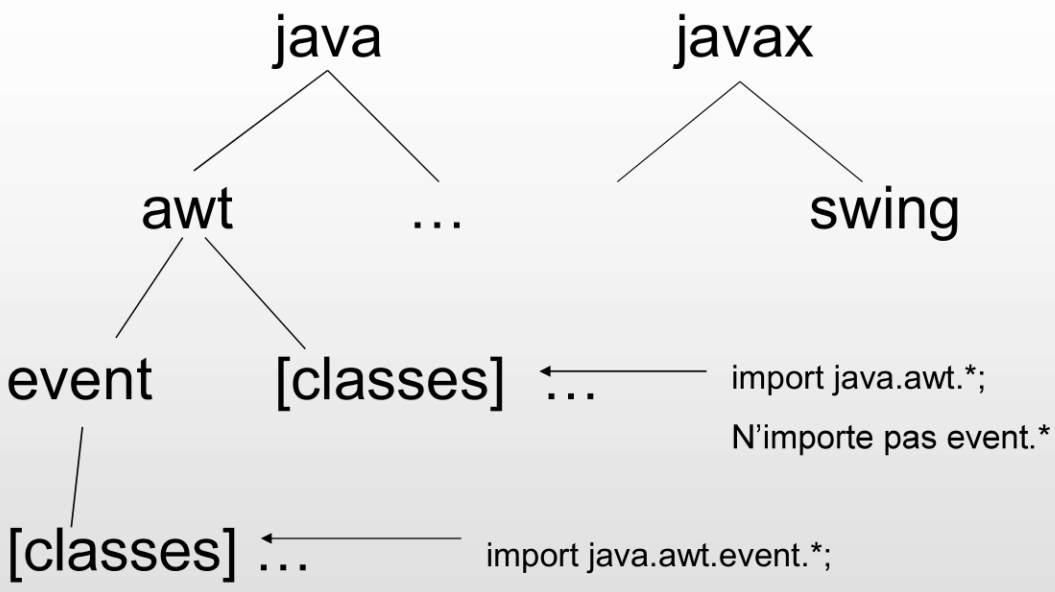


Hiérarchie des composants AWT et Swing



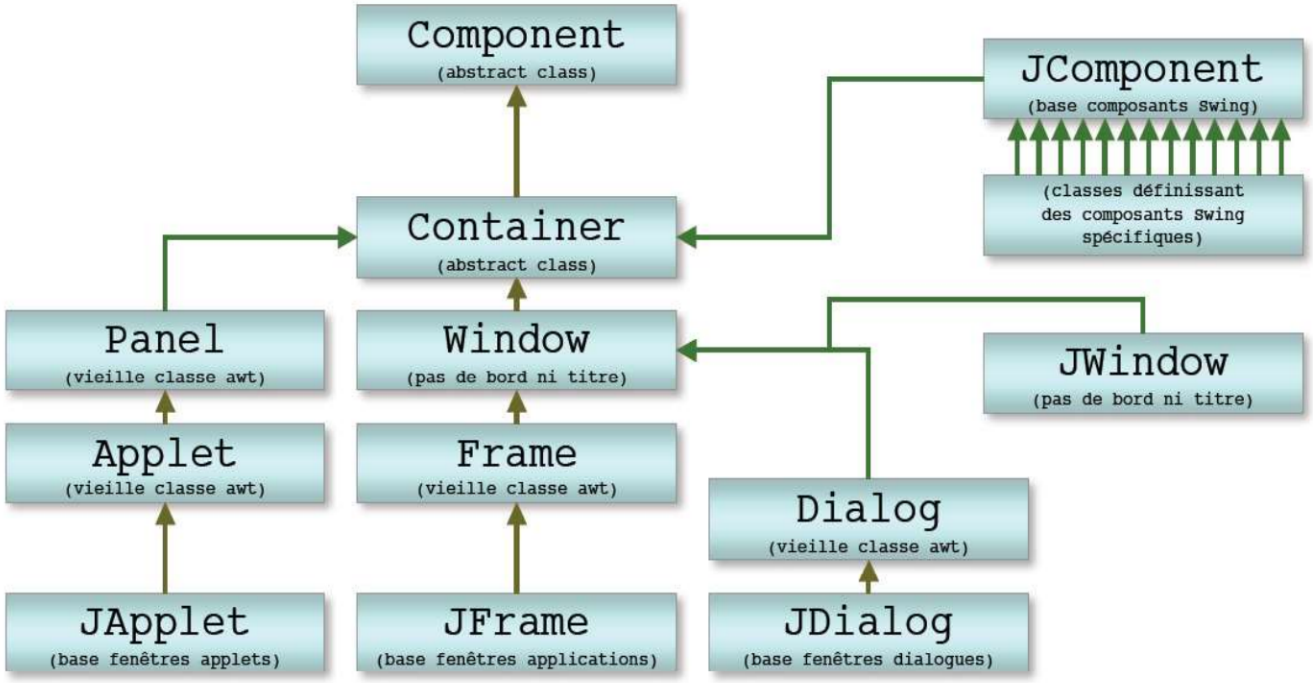
Par héritage, les composants Swing sont également des composants AWT : Les composant Swing se distinguent des composants AWT par la lettre J: JButton (Swing), Button (AWT).

Hiérarchie des packages AWT et Swing



En pratique, nous importerons systématiquement toutes les classes des paquetages java.awt , javax.swing et java.awt.event

Les composant et conteneur



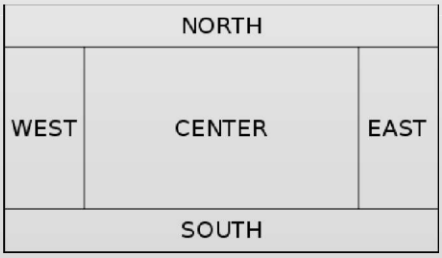
Un Container peut contenir d'autres Container à l'exception de Window et ses sous-classes (dialogues ou fenêtres)

les containers

Pour la gestion de ses composants, les containers possèdent principalement :

- Des méthodes de gestion de la relation d'agrégation Container/Composant :
  - add/remove
  - getComponentCount/getComponentAt
  - getComponents…
- Des méthodes de gestion de la disposition des composants (layout)

Exemple : BorderLayout →



La classe Component

La classe Component comporte principalement les méthodes suivantes

- Montrer et cacher un composant « **void setVisible(boolean b) »**
- Activer et désactiver un composant « **void setEnabled(boolean b) »**
- Connaître l'état d'un composant « **boolean isEnabled() »**
- Modifier la couleur du premier et celle de fond :

  void setBackground(Color c) , **void setForeground(Color c)**
- Modifier la position et la taille d'un composant :

  void setSize(int largeur, int hauteur)

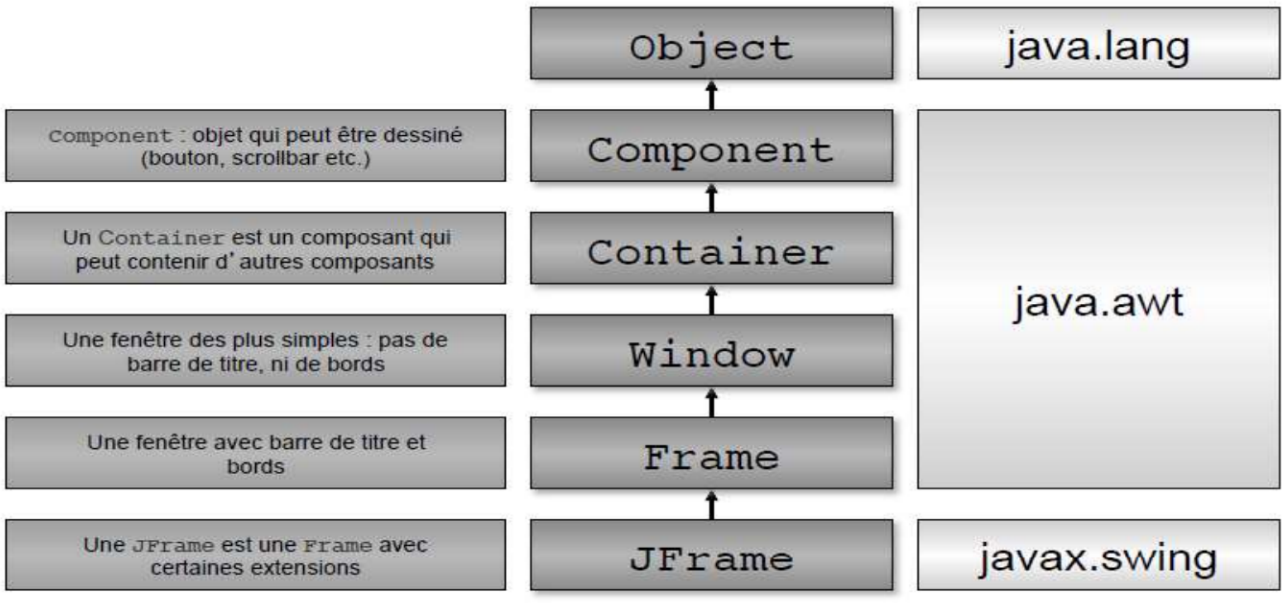
  void setBounds(int x, int y, int largeur, int hauteur)
- Gérer les dimensions d'un composant :

  Dimension getSize()

  void setSize(Dimension dim)

  La classe Dimension du paquetage java.awt contient deux champs publics : height (hauteur) et width (largeur).

La classe JFrame (1/4)

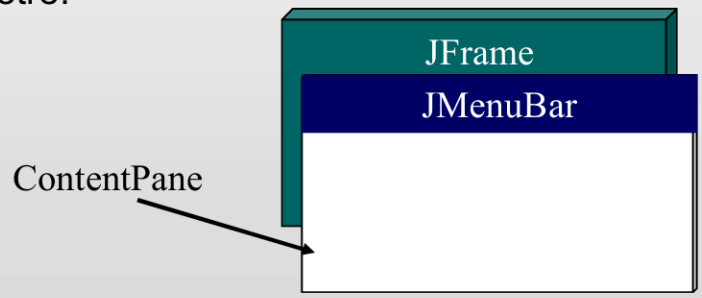


Un JFrame est l'objet qui représente une fenêtre à l'écran. C'est dans cette fenêtre qu'on place tous les composants de l'interface : boutons, cases à cocher, champs de texte, etc.

La classe JFrame (2/4)

La classe JFrame hérite de Container, Un objet JFrame peut contenir des Composants (une fenêtre ne peut être placée dans un Container).

- La méthode getContentPane de la classe JFrame fournit une référence, de type Container, au contenu de la fenêtre.
- Les méthodes add et remove de la classe Container permettent respectivement d'ajouter et de supprimer un composant d'une fenêtre.



La classe JFrame (3/4)

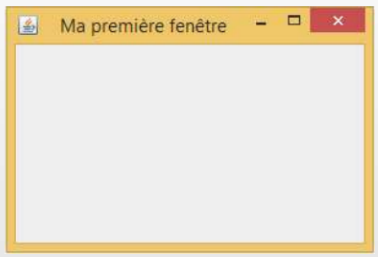
Les arguments de la méthode setDefaultCloseOperation de JFrame permet d'imposer le comportement lors de la fermeture de la fenêtre :

- DO_NOTHING_ON_CLOSE : ne rien faire,
- HIDE_ON_CLOSE : cacher la fenêtre (comportement par défaut).
- DISPOSE_ON_CLOSE : détruire l'objet fenêtre.

Mais, en aucun cas, la fermeture de la fenêtre ne mettrait fin à l'application.

La classe JFrame (4/4)

```
import javax.swing.*;
public class MaFenetre1 {
  public static void main(String[ ] args) {
    JFrame fn= new JFrame("Ma première fenêtre");
    fn.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fn.setSize(300, 200); fn.setLocation(250, 150);
    fn.setVisible(true); }
}
```




```
import java.awt.*;
import javax.swing.*;
public class MaFenetre2 extends JFrame {
  MaFenetre2(String title ) {
    setTitle(title); setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Dimension screenSize = Toolkit.getDefaultToolkit( ).getScreenSize( );
    int w = screenSize.width; int h = screenSize.height;
    setBounds(w/2, h/2, w/4, h/4);}
  public static void main(String[ ] args) {
    MaFenetre2 fn= new MaFenetre2("Ma première fenêtre"); fn.setVisible(true); }
}
```


Les classes JLabel et JButton

- Un composant de type JLabel (Une étiquette) permet d'afficher dans un conteneur un texte (d'une seule ligne) non modifiable par l'utilisateur, mais que le programme peut faire évoluer.
  - Le constructeur de JLabel précise le texte initial :

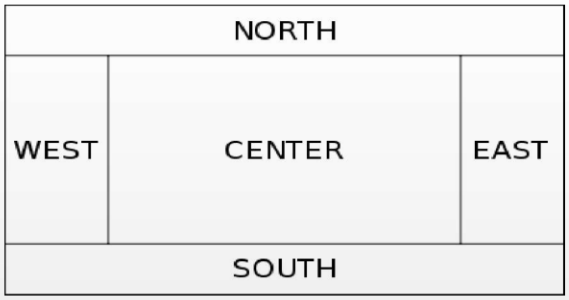
    JLabel JLabel1 = new JLabel ("texte initial") ;
  - Une étiquette n'a ni bordure, ni couleur de fond (la méthode setBackground est sans effet).
  - Le méthode setText (nouveau texte) permet de modifier le texte d'une étiquette.
- JButton correspond à une classe dessinant un bouton qui contient un libellé et qui est capable de détecter les clics de souris.
  - Le constructeur de JButton précise le texte qui figure à l'intérieur : JButton monBouton = new JButtton ("ESSAI") ;

Gestionnaire des agencements

- Un gestionnaire d'agencement (mise en page ou disposition) est un objet (classe java.awt.LayoutManager) associé à un conteneur en vue d'organiser et de structurer (taille et emplacement) ses composants.
- L'association du gestionnaire d'agencement avec le container se fait par la méthode setLayout de l'objet container.
- Les principals gestionnaires d'agencement sont BorderLayout FlowLayout et GridLayout.

Gestionnaire BorderLayout

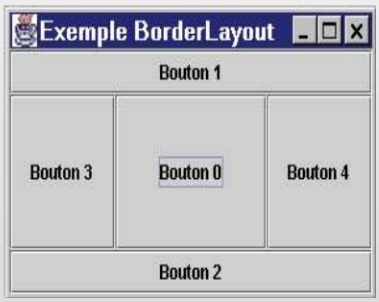
- BorderLayout divise l'espace du conteneur qui lui est associé en 5 régions:



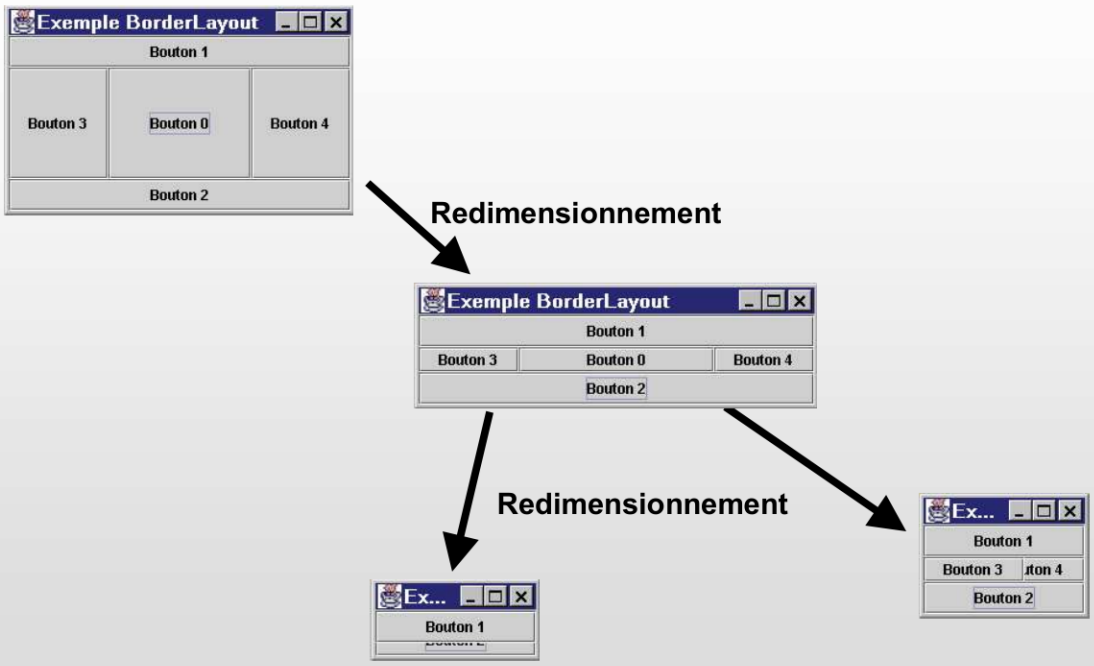
- Le Layout Manager par défaut du contentPane d'un JFrame est un BorderLayout
- Les constantes d'agencement utilisées sont: **BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.WEST, BorderLayout.EAST et BorderLayout.CENTER.**
- Un composant est ajouté à une région de son conteneur (un seul élément par région) avec la méthode add(…) prenant en second paramètre la constante d'agencement appropriée.

Gestionnaire des agencements BorderLayout

```
import javax.swing.* ; import java.awt.* ;
class MaFenetre extends JFrame {
    final int NBOUTONS = 5 ;
    public MaFenetre () {
        setTitle ("Exemple BorderLayout") ; setSize (300, 180) ;
        Container contenu = getContentPane() ;
        boutons = new JButton[NBOUTONS] ;
        for (int i=0 ; i<NBOUTONS ; i++) boutons[i] = new JButton ("Bouton " + i) ;
        contenu.add(boutons[0]) ; // au centre par defaut
        contenu.add(boutons[1], BorderLayout.NORTH) ;
        contenu.add(boutons[2], BorderLayout.SOUTH) ;
        contenu.add(boutons[3], BorderLayout.WEST) ;
        contenu.add(boutons[4], BorderLayout.EAST) ;
    }
    private JButton boutons[] ;
}
public class Layout1 {
    public static void main (String args[]) {
    MaFenetre fen = new MaFenetre() ; fen.setVisible(true) ; }
}
```

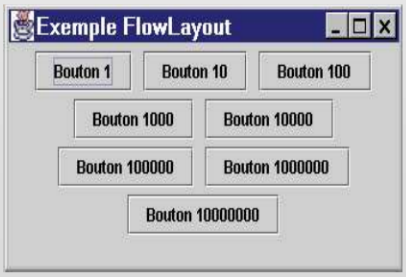


Gestionnaire des agencements BorderLayout



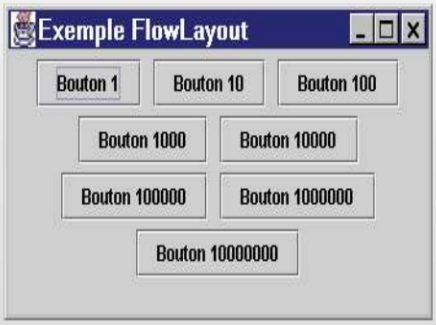
Gestionnaire des agencements FlowLayout

- Avec FlowLayout, les composants sont agencés (les uns derrière les autres) de gauche à droite, avec le retour à la ligne activé. Quand un composant (la taille des composants est respectée) ne tient pas dans le sens horizontal, il passe à la ligne suivante.
- Lors de la construction d'un gestionnaire FlowLayout, on peut spécifier un paramètre d'alignement des composants au sein du conteneur. Pour cela, on utilise l'une des constantes suivantes: **FlowLayout.LEFT, FlowLayout.RIGHT ou FlowLayout.CENTER.**

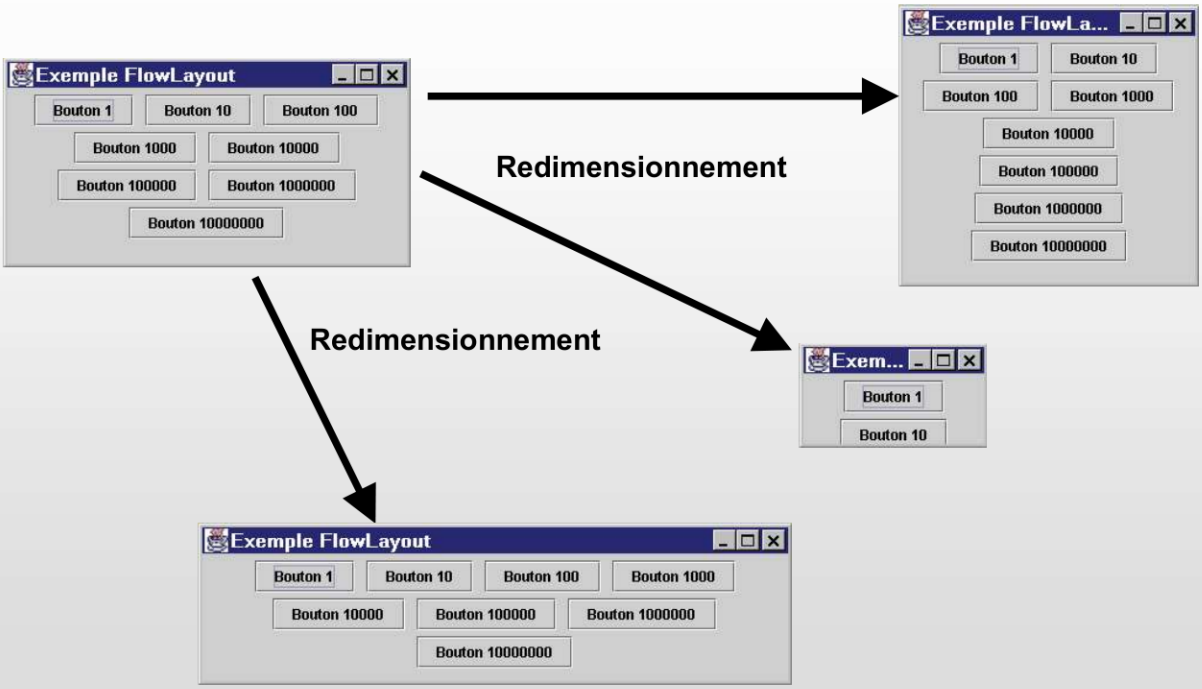


Gestionnaire des agencements FlowLayout

```
import javax.swing.* ; import java.awt.* ;
class MaFenetre extends Jframe {
   final int NBOUTONS = 8 ;
   public MaFenetre () {
     setTitle ("Exemple FlowLayout") ; setSize (350, 180) ;
     Container contenu = getContentPane() ;
     contenu.setLayout (new FlowLayout(FlowLayout.CENTER, 10, 5)) ;
     boutons = new JButton[NBOUTONS] ; int n = 1 ;
     for (int i=0 ; i<NBOUTONS ; i++) {
         boutons[i] = new JButton ("Bouton " + n) ;
         n *= 10 ; contenu.add(boutons[i]) ;
     }
   }
   private JButton boutons[] ;
}
public class Layout2 {
   public static void main (String args[]) {
         MaFenetre fen = new MaFenetre() ; fen.setVisible(true) ; }
   }
```

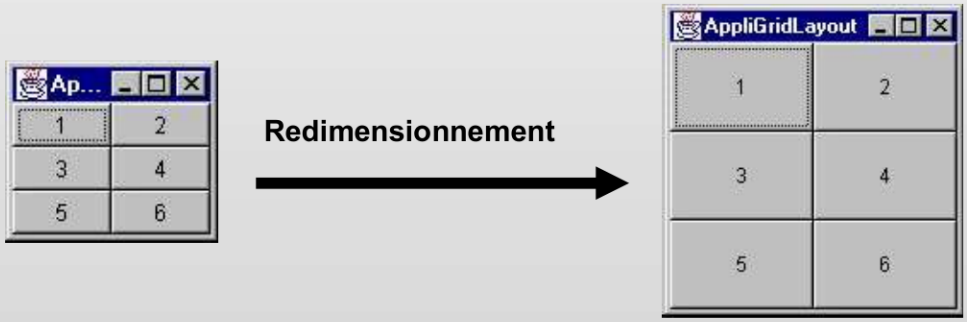


Gestionnaire des agencements FlowLayout



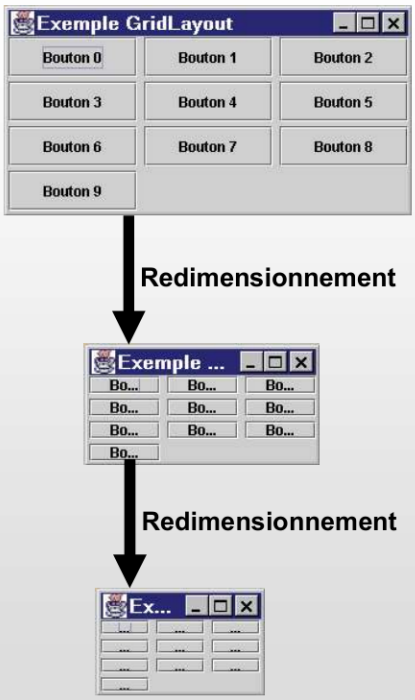
Gestionnaire des agencements GridLayout

- Le gestionnaire GridLayout permet de disposer les différents composants suivant une grille régulière, chaque composant occupant une cellule.
- À la construction, on choisit le nombre de lignes et de colonnes de la grille et, éventuellement, des intervalles entre les composants
- Chaque composant est ajouté à son conteneur avec la méthode add( ) qui remplit la grille ligne par ligne.



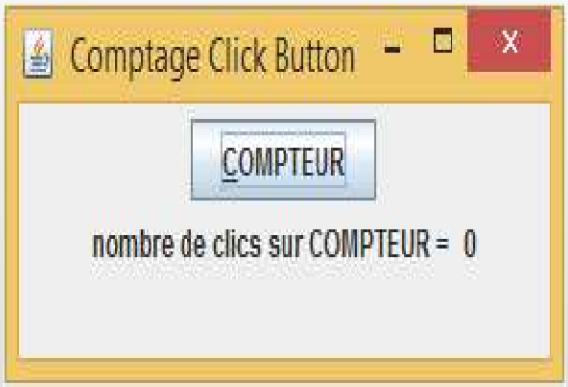
Gestionnaire des agencements GridLayout

```
import javax.swing.* ; import java.awt.* ;
class MaFenetre extends Jframe {
    public static int NBOUTONS = 10;
    public MaFenetre () {
        setTitle ("Exemple GridLayout") ;
        setSize (350, 180) ;
        Container contenu = getContentPane() ;
        contenu.setLayout (new GridLayout(4, 3, 6, 4)) ;
        boutons = new JButton[NBOUTONS] ;
        for (int i=0 ; i<NBOUTONS ; i++) {
            boutons[i] = new JButton ("Bouton " + i) ;
            contenu.add(boutons[i]) ;
        }
    }
    private JButton boutons[] ;
    }
    public class Layout3 {
        public static void main (String args[]) {
            MaFenetre fen = new MaFenetre() ; fen.setVisible(true) ; }
    }
```



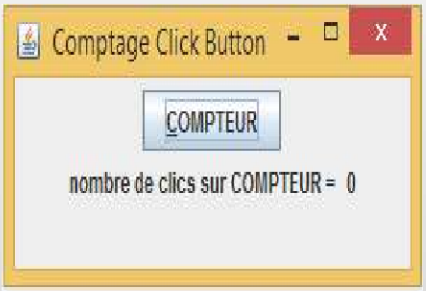
Exemple 1

Interface comportant un JButton « COMPTEUR » et deux JLabel. L'une des étiquettes doit contenir le nombre des cliques sur le bouton.



Exemple 1 (sans événements)

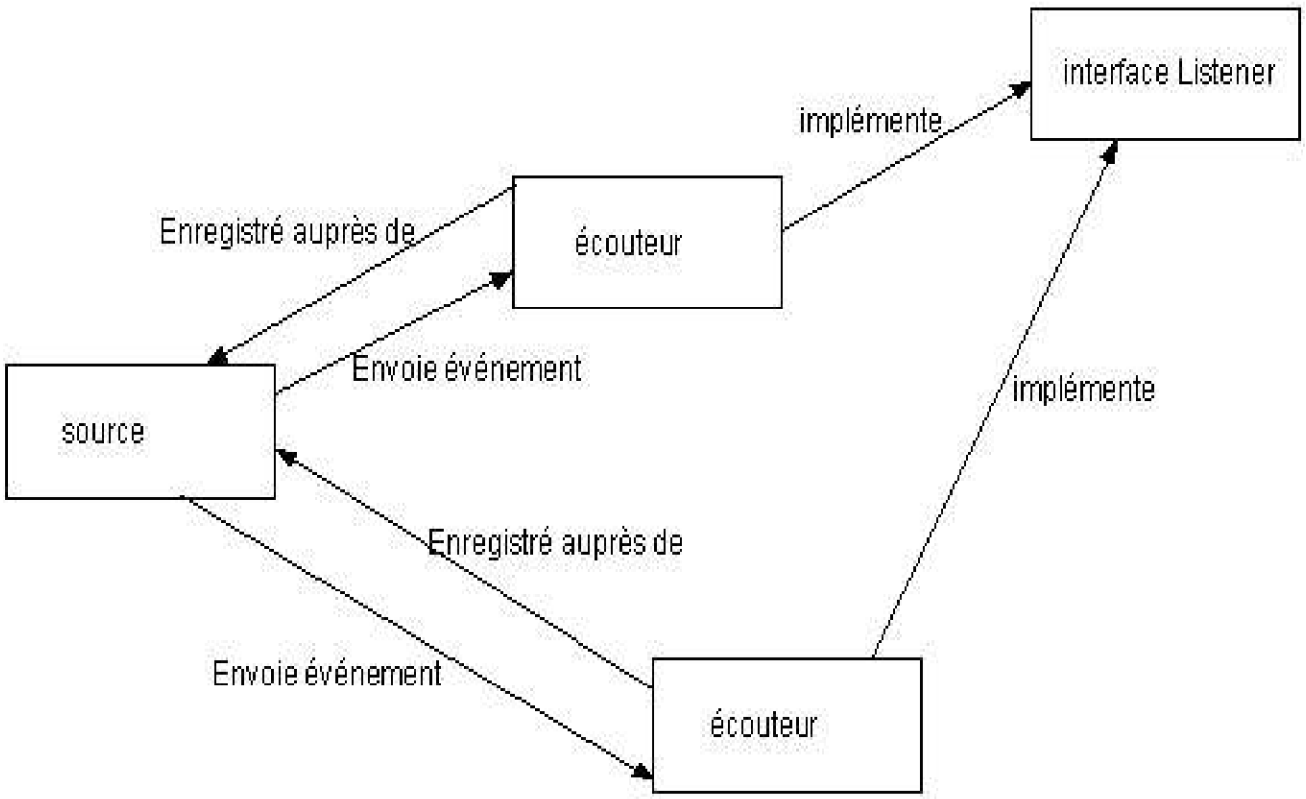
```
import java.awt.* ;  import javax.swing.* ; import java.awt.event.KeyEvent;
class Fenetre extends JFrame {
    public Fenetre () {
       setTitle ("Comptage Click Button") ; setSize (300, 120) ;

       buttonCompt = new JButton ("COMPTEUR") ;
       buttonCompt.setMnemonic(KeyEvent.VK_C);// Alt + C
       JLabel LabelMessage = new JLabel ("Nombre de clics sur COMPTEUR = ") ;
       LabelCompt = new JLabel ("0") ;

       Container contenu = getContentPane() ;
       contenu.setLayout (new FlowLayout() ) ;

       contenu.add(buttonCompt) ;
       contenu.add(LabelMessage) ;
       contenu.add(LabelCompt) ;
    }
    private JLabel LabelCompt ;
    private JButton buttonCompt;
}
public class Exemple1 {
     public static void main (String args[]) {
       Fenetre fen = new Fenetre() ; fen.setVisible(true) ;  }
}
```



Gestion des événements

- En java, pour qu'un objet puisse répondre à un événement, il faut lui attacher un écouteur d'événement (**Listener**). Lorsqu'un événement se produit:
  - Il est reçu par le composant avec lequel l'utilisateur interagit (objet source).
  - L'objet **source** transmet l'événement à l'écouteur qui possède une méthode pour traiter l'événement.
  - La méthode de traitement de l'événement reçoit l'objet événement généré de façon à traiter l'interaction de l'utilisateur.
- Il existe différents types de Listeners (représentées comme interfaces):
  - **WindowsListener** : pour gérer les événement sur la fenêtre
  - **ActionListener** :pour gérer les événements produits sur les composants graphiques.
  - **KeyListener** : pour gérer les événements du clavier
  - **MouseListener** : pour gérer les événements de la souris.
  - ….

Gestion des événements



Catégorie d'événements AcltionListener

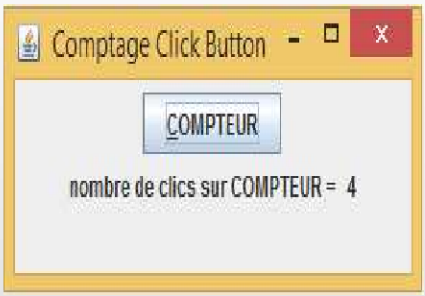
- **AcltionListener** est une interface qui définit une seule méthode:

  public void **actionPerformed**(ActionEvent e);

- L'événement actionPerformed est produit quand on valide une action par un clique ou par la touche de validation du clavier.
- Pour gérer cet événement, il faut créer une classe implémentant l'interface ActionListener et redéfinir la réponse aux événements utilisateur dans la méthode actionPerformed.
- L'objet instancié de la classe implémentant cette interface, s'appelle un listener (écouteur) ou un gestionnaire d'événements.
- L'association d'un objet source à un objet écouteur est obtenu par une méthode nommée **addActionListener**.

Exemple 1 (avec l'événement Action)

```
import java.awt.* ;
import javax.swing.* ;
import java.awt.event.*; // utilisation de KeyEvent, ActionEvent et ActionListener
class Fenetre extends JFrame implements ActionListener {
    public Fenetre () {
        … // création de l'aspect visuel de l'interface
        buttonCompt.addActionListener (this);
    }
    public void actionPerformed(ActionEvent e) {
         nbClics++ ;
         LabelCompt.setText(""+ nbClics ) ;
    }
    private JLabel LabelCompt ; private JButton buttonCompt;
    private int nbClics ;
}
public class Exemple1 {
     public static void main (String args[]) {
        Fenetre fen = new Fenetre() ; fen.setVisible(true) ;  }
}
```

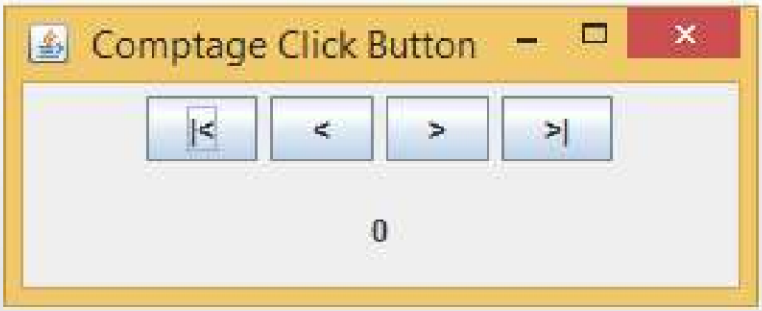


Méthode associées aux événements

- Pour un évènement donné, la méthode getSource permet d'identifier la source de l'événement (elle s'applique aux événements générés par tous les composants).
- La méthode getActionCommand (présenté uniquement dans ActionEvent) permet d'obtenir la chaîne de commande associée à l'evenement (étiquette associée à un bouton par exemple)
- La méthode setActionCommand permet d'imposer une chaîne de commande donnée.
- La méthode getComponent fournit la même référence que getSource, mais du type Component. Ainsi, e.getComponent() est équivalent à (Component)e.getSource().

Exemple 2

Interface comportant des JButton et une étiquette qui contient la valeur obtenue par un clique de chacune des boutons.



L'interface WindowListener

- Les fenêtres génèrent des événements de la catégorie WindowEvent (actions d'ouverture, de fermeture ou de réduction en icône).
- L'interface WindowListener comporte plusieurs méthodes qui **doivent toutes être redéfinies** (définitions vides des méthodes dont on a pas besoin).

Exemple : Arrêt d'un programme

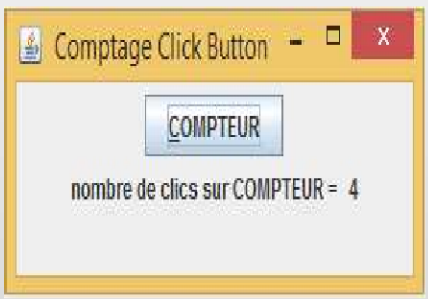
```
class … implements WindowListener {
…
public void windowClosing (WindowEvent e) {// en cours de fermeture
        System.exit(0) ;}
public void windowOpened (WindowEvent e) { } // ouverture fenetre
public void windowIconified (WindowEvent e) { } // fenetre en icone
public void windowDeiconified (WindowEvent e) { } // icone en fenetre
public void windowClosed (WindowEvent e) { } // fenetre fermee
public void windowActivated (WindowEvent e) { } // fenetre activee
public void windowDeactivated (WindowEvent e) { } // fenetre desactivee
…}
```

Notion d'adaptateur

- Pour éviter la redéfinition de toutes les méthodes de l'interface WindowListener , Java dispose d'une classe particulière WindowAdapter qui implémente toutes ces méthodes avec un corps vide. Ainsi, il suffit que la classe écouteur hérite de WindowAdapter pour redéfinir uniquement les méthodes nécessaires.
- En général, à une catégorie d'évènements donnée Xxx (exemple Action ou Window), on associera toujours à une source un objet écouteur (de type XxxListener), par la méthode addXxxListener. Lorsque cette catégorie dispose de plusieurs méthodes, on pourra :
  - Soit redéfinir toutes les méthodes de l'interface XxxListener (la clause implements XxxListener doit figurer dans l'en-tête de classe de l'écouteur), certaines méthodes pouvant avoir un corps vide ;
  - Soit faire appel à une classe dérivée d'une classe adaptateur XxxAdapter et ne fournir que les méthodes qui nous intéressent.

Exemple 1 (avec WindowAdapter)

```
import java.awt.* ; import javax.swing.* ; import java.awt.event.*;
class EcouteurWindow  extends  WindowAdapter  {
   public void windowClosing (WindowEvent e) { System.exit(0) ; }
}
class Fenetre extends JFrame implements ActionListener {
   public Fenetre () {
      … // création de l'aspect visuel de l'interface
      buttonCompt.addActionListener (this);
      this.addWindowListener(new EcouteurWindow() );
   }
   public void actionPerformed(ActionEvent e) {
        nbClics++; LabelCompt.setText(""+ nbClics ) ;
   }
   private JLabel LabelCompt ;
   private JButton buttonCompt; private int nbClics ;
}
public class Exemple1 {
    public static void main (String args[]) {
       Fenetre fen = new Fenetre() ; fen.setVisible(true) ;  }
}
```

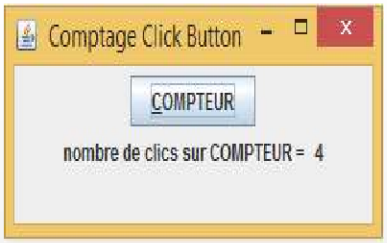


Classe interne

- Une classe interne est une classe imbriquer dans une autre classe externe.
  - Un objet d'une classe interne est toujours associé (à l'instanciation), à un objet d'une classe externe (on dit qu'il lui a donné naissance).
  - Un objet d'une classe interne a toujours accès aux champs et méthodes (même privés) de l'objet externe lui ayant donné naissance (pas aux autres objets de cette classe).
  - Un objet de classe externe a toujours accès aux champs et méthodes (même privés) d'un objet d'une classe interne auquel il a donné naissance.
- Les classes internes sont surtout utilisées pour la gestion des événements puisque leur utilisation permet d'établir facilement le lien entre l'objet source d'événements (objet de la classe externe) et l'objet écouteur d'événements (objet de la classe interne).

Exemple 1 (avec classes internes)

```
import java.awt.* ; import javax.swing.* ; import java.awt.event.*;
class Fenetre extends JFrame {
   class EcouteurAction implements ActionListener  {
      public void actionPerformed(ActionEvent e) {
            nbClics++ ; LabelCompt.setText(""+ nbClics ) ;  }
   }
   class EcouteurWindow  extends  WindowAdapter  {
      public void windowClosing (WindowEvent e) {  System.exit(0) ; }
   }
   public Fenetre () {
      … // création de l'aspect visuel de l'interface
      buttonCompt.addActionListener(new EcouteurAction());
      this.addWindowListener(new EcouteurWindow());
    }
   private JLabel LabelCompt ; private JButton buttonCompt; private int nbClics ;
}
public class Exemple1 {
    public static void main (String args[]) {
       Fenetre fen = new Fenetre() ; fen.setVisible(true) ;  }
}
```

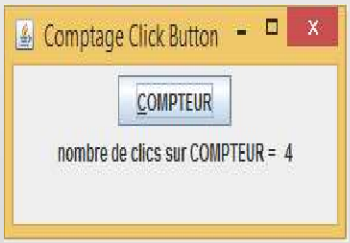


Classe anonyme

- Une classe anonyme est une classe qui ne possède pas de nom. Une telle classe ne peut donc être instanciée qu'à l'endroit où elle est définie(elle ne possède pas de constructeur).
- L'utilisation d'une classe anonyme ne s'applique que dans deux cas:
  - Classe anonyme dérivée d'une autre classe.
  - Classe anonyme implémentant une interface.
- Ce type de classe est très pratique lorsqu'une classe doit être utilisée une seule fois. Elle peut , ainsi, être utilisée comme écouteur d'événements pour une source déclarée dans la classe externe.

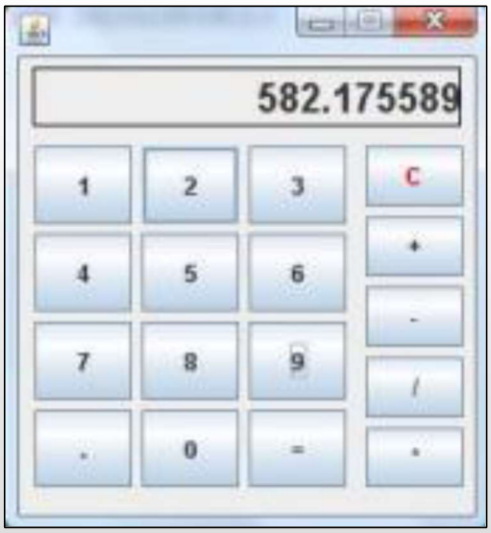
Exemple 1 (avec classes anonymes)

```
import java.awt.* ; import javax.swing.* ; import java.awt.event.*;
class Fenetre extends JFrame {
    public Fenetre () {
        … // création de l'aspect visuel de l'interface
        buttonCompt.addActionListener( new ActionListener( ) {
            public void actionPerformed(ActionEvent e) {
                nbClics++ ; LabelCompt.setText(""+ nbClics ) ;
            }
        } );
        this.addWindowListener(new WindowAdapter( ) {
            public void windowClosing (WindowEvent e) { System.exit(0) ; }
        } );
    }
    private JLabel LabelCompt ; private int nbClics ;
}
public class Exemple1 {
     public static void main (String args[]) {
          Fenetre fen = new Fenetre() ; fen.setVisible(true) ;  }
}
```



Exercice

Réaliser une calculatrice dont l'interface est la suivante (un JLabel pour l'affichage et autant de boutons qu'il en faut) :

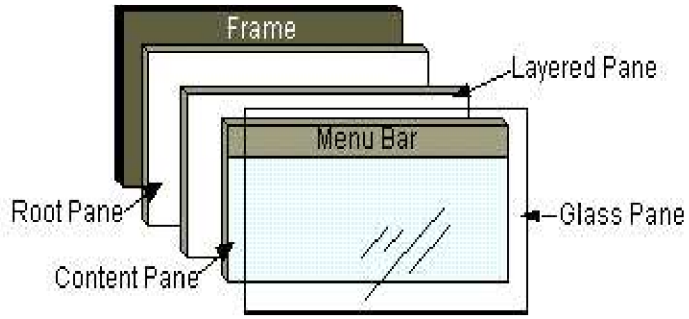


Interface Homme-Machine en JAVA

Chapitre 2: Les contrôles usuels des IHMs

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Structure d'un objet JFrame



- La zone d'une fenêtre située sous la barre de titre est un objet JRootPane.
- La zone où sont placés les menus d'une fenêtre est un Objet type JLayeredPane.
- La zone contenu comportant les composants d'une fenêtre est un objet contentPane de type JInternalPane.
- Le panneau GlassPane est transparent à toute la zone du JRootPane. (utilisé pour des pop-up menus ou des animations).
- La classe JFrame contient les méthodes d'accès aux panneaux : getRootPane(), getLayeredPane(), getContentPane() et getGlassPane().

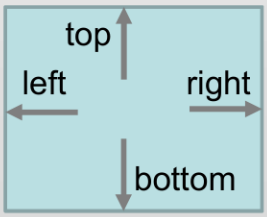
La classe JPanel

- Un **JPanel** est un conteneur intermédiaire destiné à contenir d'autres composants. Il est muni d'un gestionnaire d'agencement (le mode FlowLayout est le gestionnaire par défaut des conteneurs JPanel).
- Généralement, un composant est ajouté à un conteneur JPanel qui est, lui-même, ajouté au panneau de contenu de la fenêtre.
  - Constructeur : JPanel pane = new JPanel();
  - Méthode de mise de l'espace entre un panneau et son contenu:

    setBorder(BorderFactory.createEmptyBorder(top, left, bottom, right);
  - Méthode de gestion de la disposition:

    setLayout(new …);
  - Méthode d'ajout des composants :

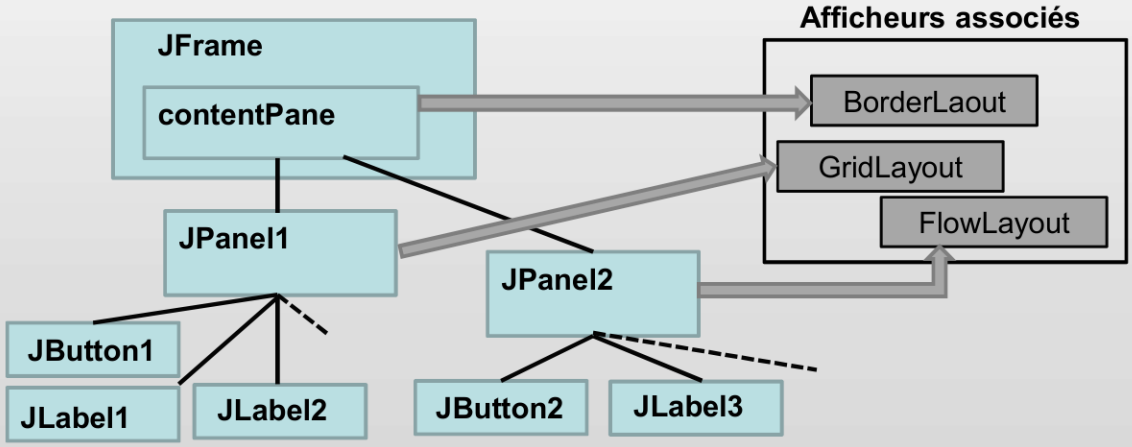
    pane.add(button);



Arbre des composants

- Un Container (objet JPanel par exemple) regroupe géométriquement les composants qui lui sont ajoutés. Ce Container peut également contenir d'autres Container (On a donc une arborescence de composants).
- La méthode getParent() sur un composant retourne le parent (de type Container) qui le contient.

Exemple d'arbre de composants



Les cases à cocher JCheckBox

- Le constructeur d'une case à cocher requiert un libellé qui sera affiché à côté de la case:

```
JCheckBox coche = new JCheckBox ("CASE") ; // case non  cochée
JCheckBox coche = new JCheckBox ("CASE", true) ;
```

- La méthode isSelected() retourne l'état d'une case.
- La méthode setSelected (boolean etat) permet d'imposer un état donné à une case.
- Chaque action de l'utilisateur sur une case à cocher génère à la fois un événement Action (méthode actionPerformed de l'interface ActionListener) et un événement Item.
- La méthode getActionCommand() de l'événement Action retourne le libellé de la case correspondante.
- Un écouteur de la catégorie Item doit implémenter l'interface ItemListener pour redéfinir sa seule méthode itemStateChanged:

```
public void itemStateChanged (ItemEvent ev)
```


Les boutons radio JRadioButton

- Un objet radio est instancié par la classe JRadioButton. son constructeur requiert le libellé de l'objet.

```
new JRadioButton ("…") ;   new JRadioButton ("…", true) ;
```

- L'état d'un bouton radio s'obtient par la méthode isSelected()
- La méthode setSelected(boolean etat), impose un état à un bouton
- Pour obtenir la désactivation automatique d'un ensemble de boutons radio, il faut les associer (méthode add) à un groupe ButtonGroup (un objet ButtonGroup n'est pas un composant).
- Pour mettre en évidence les groupes de boutons, il faut utiliser un panneau différent pour chaque groupe . chaque panneau est entouré d'une bordure comportant un titre indiquant le rôle du groupe.

```
panneau.setBorder (new TitledBorder ("…")) ;
```

- Pour un bouton radio, on utilise généralement l'événement Item (la sélection d'un bouton radio appartenant à un groupe génère les événements Action et Item pour ce bouton, ainsi qu'un événement Item pour un autre bouton du groupe).

Le champs de texte : JTextField (1/2)

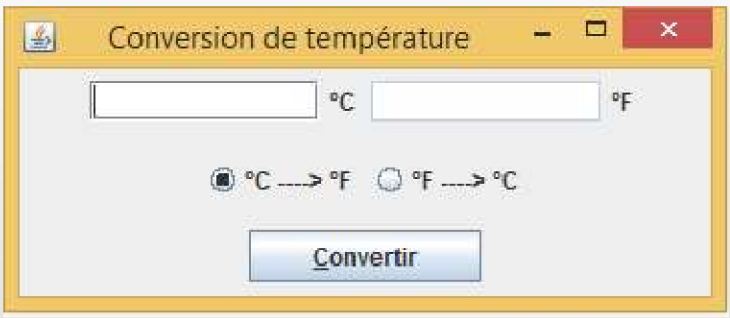
- Un champ de texte s'obtient en instanciant un objet de type JTextField. Son constructeur doit obligatoirement indiquer une taille (nb de caractères) : new JTextField (20) ; // champ initialement vide.
- La méthode getText() renvoie la chaine de caractère figurant dans un champ de texte. La méthode setText(String s) met la chaîne s dans un champ de texte.
- La méthode boolean isEditable(): teste si un champ de texte est éditable (modifiable par l'utilisateur) ou non. La méthode setEditable(boolean b): définit la propriété éditable pour le champ de texte.
- Lors de l'exécution on peut modifier la taille d'un champ de texte avec l'utilisation des méthodes setColumns et revalidate (ou la méthode validate à son conteneur) : setColumns (…) ; revalidate() ;

Le champs de texte : JTextField (2/2)

- Les événements générés par un champ de texte sont:
  - un événement Action provoqué par l'appui de l'utilisateur sur la touche de validation (le champ de texte étant sélectionné);
  - un événement de perte (ou de gain) de focus au moment où le champ de texte perd (ou gain) le focus.
- L'interface FocusListener comporte deux méthodes d'en-têtes :
  - public void focusGained (FocusEvent e) // prendre le focus
  - public void focusLost (FocusEvent e) // perdre le focus

Exercice 1

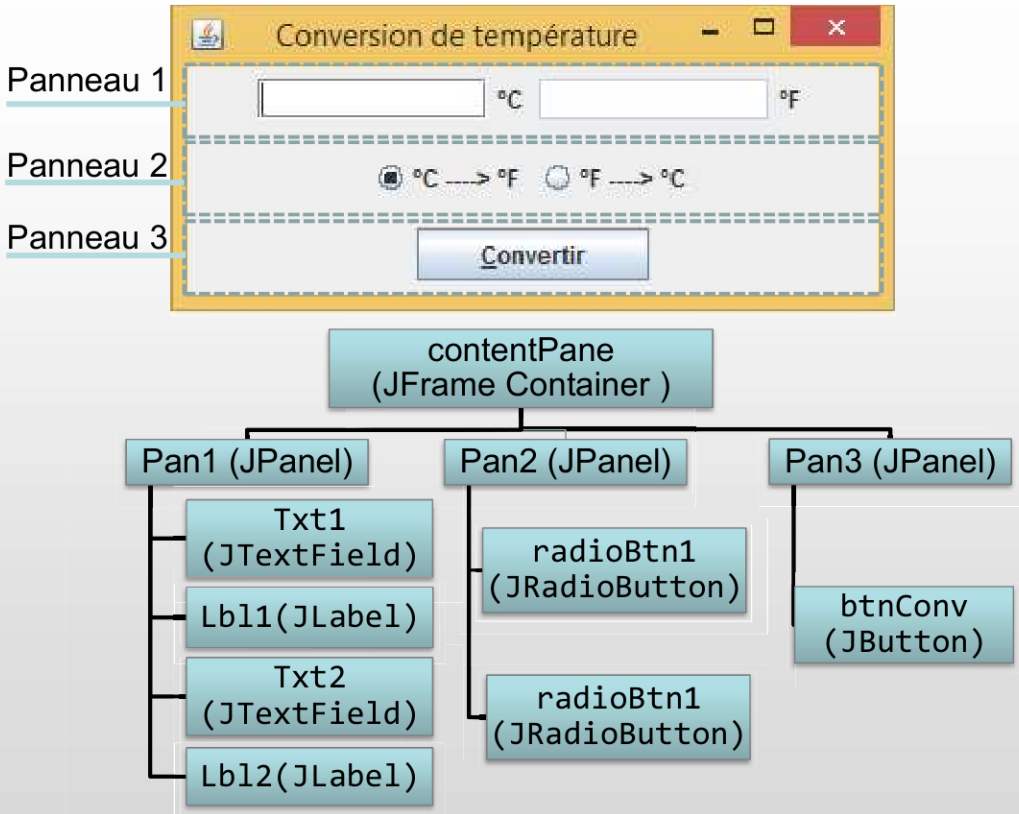
Application de conversion de température exprimée en Fahrenheit ou en Celsius.



Les formules de conversion à utiliser sont :

$$°C = \frac{°F - 32}{1.8000} \quad , \quad °F = °C * 1.8000 + 32$$

Exercice 1 (arbre des composants)



Exercice 1

```
import java.awt.*; import java.awt.event.*; import javax.swing.*;
class FenConv extends JFrame {
  public FenConv() {
    //création de l'aspect graphique
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 350, 150); setTitle("Conversion de température" );
    txt1=new JTextField("0", 10); txt1.setHorizontalAlignment(SwingConstants.CENTER);
    lbl1 = new JLabel ("°C ") ; lbl2 = new JLabel ("°F ") ; txt2=new JTextField("0", 10);
    txt2.setHorizontalAlignment(SwingConstants.CENTER);  txt2.setEnabled(false);
    JPanel pan1 = new JPanel(); pan1.setLayout(new FlowLayout());
    pan1.add(txt1);pan1.add(lbl1);pan1.add(txt2);pan1.add(lbl2);
    radioBtn1 =new JRadioButton("°C ----> °F"); radioBtn2 =new JRadioButton("°F ----> °C");
    ButtonGroup grRB=new ButtonGroup(); grRB.add(radioBtn1); grRB.add(radioBtn2);
    radioBtn1.setSelected(true);
    JPanel pan2= new JPanel(); pan2.setLayout(new FlowLayout());
    pan2.add(radioBtn1);pan2.add(radioBtn2);
    btnConv = new JButton ("     Convertir     "); btnConv.setMnemonic(KeyEvent.VK_C);
    JPanel pan3 = new JPanel(); pan3.setLayout(new FlowLayout()); pan3.add(btnConv);
    Container contentPane = getContentPane(); contentPane.setLayout(new GridLayout(3,1));
    contentPane.add(pan1);  contentPane.add(pan2); contentPane.add(pan3);
    pack();
```


Exercice 1

```
    radioBtn1.addItemListener( new ItemListener() {
        public void itemStateChanged(ItemEvent arg0) {
                if (radioBtn1.isSelected()) {etat=true; lbl1.setText("°C "); lbl2.setText("°F ");}
                        else {etat=false; lbl1.setText("°F "); lbl2.setText("°C ");}
                convertir(); }
    });
    this.addWindowListener(new WindowAdapter( )  {
         public void windowClosing (WindowEvent e) { System.exit(0) ; }
    } );
    btnConv.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) { convertir();}
    });
  }
  private void convertir(){
    Float val1= Float.parseFloat(txt1.getText()); Float val2;
    if (etat) val2= (float) (val1 * 1.8 + 32) ;  else val2= (float) ( (val1 -32 ) / 1.8 ) ;
    txt2.setText(val2.toString()); }
  }
  private boolean etat=true; // true pour °C --> °F et flase sinon
  JTextField txt1, txt2;
  JLabel lbl1, lbl2;
  JButton  btnConv; JRadioButton radioBtn1 , radioBtn2 ;
}
```

La MVC JTextField

- Le composant JTextField est implémenté (comme tous les composants Swing), en utilisant une architecture modèle-vue-contrôleur. Ainsi, un objet de type Document permet de conserver le contenu du composant, tandis qu'un autre objet nommé "vue" est utilisé pour fournir la représentation de ce contenu.
- Toute modification de l'objet de type Document génère un des trois événements de la catégorie Document. L'écouteur correspondant doit implémenter l'interface DocumentListener, laquelle comporte trois méthodes d'en-têtes (Il n'existe pas de classe adaptateur) :
  - public void insertUpdate(DocumentEvent e) // insertion des caractères
  - public void removeUpdate(DocumentEvent e) //suppression
  - public void changedUpdate (DocumentEvent e) // pas pour JTextField
- L'objet document associé à un composant s'obtient par la méthode getDocument.

Exercice 2

Application de conversion de température exprimée en Fahrenheit ou en Celsius.

Les formules de conversion à utiliser sont :

$$°C = \frac{°F - 32}{1.8000} \quad , \quad °F = °C * 1.8000 + 32$$

Exercice 2

```
import java.awt.*; import java.awt.event.*; import javax.swing.*;
class FenConv extends JFrame {
  public FenConv() {
    … //création de l'aspect graphique
    // Aspect événementiel
    radioBtn1.addItemListener( new ItemListener() {
      public void itemStateChanged(ItemEvent arg0) {
        if (radioBtn1.isSelected()) {etat=true; lbl1.setText("°C "); lbl2.setText("°F ");}
        else {etat=false; lbl1.setText("°F "); lbl2.setText("°C ");}
        convertir(); }
    });
    this.addWindowListener(new WindowAdapter( )  {
        public void windowClosing (WindowEvent e) { System.exit(0) ; }
    } );
    txt1.getDocument().addDocumentListener(new DocumentListener() {
        public void removeUpdate(DocumentEvent e) { convertir(); }
        public void insertUpdate(DocumentEvent e) { convertir(); }
        public void changedUpdate(DocumentEvent e) { }
    });
  }
```


La boîte de liste JList (1/3)

- La boîte de liste JList permet de sélectionner une ou plusieurs valeurs dans une liste prédéfinie (initialement, aucune valeur n'est sélectionnée).
- Une boîte de liste peut être crée en fournissant un tableau de chaînes à son constructeur.

  Exemple :

  ```
  String[] couleurs = {"rouge", "bleu", "gris", "vert", "jaune", "noir" } ;
  JList liste = new JList (couleurs) ;
  ```

- Un panneau de défilement de type JScrollPane permet d'ajouter une barre de défilement à une boîte JList (l'objet JScrollPane est le seul à être ajouté au conteneur concerné).

  Exemple :

  ```
  JScrollPane defil = new JScrollPane (liste) ; // introduit liste dans defil
  ```

La boîte de liste JList (2/3)

- Modifier le type de sélection d'une boîte JList (paramètre : SINGLE_SELECTION,SINGLE_INTERVAL_SELECTION, MULTIPLE_INTERVAL_SELECTION).

  void setSelectionMode (int modeDeSelection)

- Fixer le nombre de valeurs à afficher avant de défiler d'une liste: void

  setVisibleRowCount(int NbValeur) ;

- Récupérer la (ou les) valeur (s) sélectionnée (s). Une conversion explicite est obligatoire.

  Object getSelectedValue()

  Object[ ] getSelectedValues()

- Connaître la (ou les) valeur (s) sélectionnée(s), et sa (ou leur) position (s) dans la liste.

  int getSelectedIndex() // position 1ière valeur sélectionnée

  int[ ] getSelectedIndices() // tableau des positions sélectionnées

La boîte de liste JList (3/3)

- Les événements générés par la liste (ne génère pas d'événement Action) sont de la catégorie ListSelection, laquelle correspond à l'interface ListSelectionListener (paquetage swing.event) et comporte une seule méthode valueChanged.

  public void valueChanged (ListSelectionEvent e)

- Pour pallier à la redondance de l'événement généré(désélection de la valeur précédente par appui sur le bouton de la souris et sélection d'une nouvelle par relâchement), on dispose d'une méthode getValueIsAdjusting permettant de savoir si l'on est ou non en phase de transition.

```
public void valueChanged (ListSelectionEvent e){
   if (!e.getValueIsAdjusting()) {
      // accès aux informations sélectionnées et traitement
   }
}
```

Exercice 3

On considère une application de conversion de devises. Les taux de change sont indiques sur la base du Dollar(USD) selon les barèmes suivant (on se limite à ces devises):

- 1 Dirham (MAD)= 0.10072 Dollar (USD)
- 1 Euro (EUR) = 1.08461 USD
- 1 Livre (GBP) =1.45700 USD
- 1 Yen (JPY)= 0.00847 USD

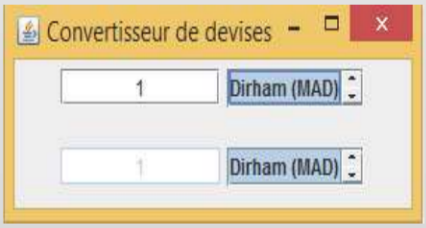
La conversion se fera en fonction des éléments sélectionnes dans les 2 listes (objet JList).

Exercice 3

```
import java.awt.*; import java.awt.event.*;
import javax.swing.*; import javax.swing.event.*;
public class Convertisseur extends JFrame{
  private JTextField txt1, txt2;  private JList list1,list2;
  String[] moneys={"Dirham(MAD)","Dollar(USD)","Euro(eur)","Livre(GBP)","Yen (JPY)"};
  float[] changes = {0.10072f,  1, 1.08461f, 1.45700f, 0.00847f };
  int indiceDevise1, indiceDevise2;
  public Convertisseur() {
    txt1=new JTextField("1", 10); txt2= new JTextField("1", 10);
    txt1.setHorizontalAlignment(SwingConstants.CENTER);
    txt2.setHorizontalAlignment(SwingConstants.CENTER); txt2.setEnabled(false);
    Container cont = getContentPane();cont.setLayout(new GridLayout(2,1,5,5));
    JPanel panDevise1 = new JPanel(new FlowLayout());
    JPanel panDevise2 = new JPanel(new FlowLayout());
    list1 = new JList(moneys);JScrollPane defilDevis1 = new JScrollPane (list1);
    list1.setVisibleRowCount(1);list1.setSelectedIndex(0);
    list2 = new JList(moneys); JScrollPane defilDevis2 = new JScrollPane (list2);
    list2.setVisibleRowCount(1);list2.setSelectedIndex(0);
    panDevise1.add(txt1); panDevise1.add(defilDevis1);
    panDevise2.add(txt2); panDevise2.add(defilDevis2);
    cont.add(panDevise1);cont.add(panDevise2);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); ... }
```

Exercice 3

```
public class Convertisseur extends JFrame{
  class ecouteurList implements ListSelectionListener{
     public void valueChanged(ListSelectionEvent e) {
      if (!e.getValueIsAdjusting()) {
        indiceDevise1=list1.getSelectedIndex();indiceDevise2=list2.getSelectedIndex();
        currentChange(indiceDevise1, indiceDevise2) ; }
    } }
  public void currentChange(int ind1, int ind2) {
    Float output= Float.parseFloat(txt1.getText() )* (changes[ind1]/ changes[ind2]) ;
    txt2.setText( output.toString() );   }
  public Convertisseur() {
    ...
    list1.addListSelectionListener( new ecouteurList());
    list2.addListSelectionListener( new ecouteurList());
    txt1.getDocument().addDocumentListener(new DocumentListener() {
       public void removeUpdate(DocumentEvent e) {
            currentChange(indiceDevise1, indiceDevise2) ;
       public void insertUpdate(DocumentEvent e) {
            currentChange(indiceDevise1, indiceDevise2) ; }
       public void changedUpdate(DocumentEvent e) { }
     });
  }
```



Les boîtes combo « JComboBox » (1/3)

- La boîte de liste combinée associe un champ de texte (par défaut non éditable) et une boîte de liste à sélection simple.
  - L'utilisateur peut choisir une valeur dans la liste, qui s'affiche alors dans le champ de texte.
  - Par validation (ou perte de focus), l'utilisateur peut également saisir dans le champ de texte (il faut rendre le champ éditable) une valeur de son choix (la nouvelle valeur n'est pas ajoutée automatiquement à la liste).
- Une boîte combo peut être construite en fournissant à son constructeur de la clase JComboBox un tableau de chaine de caractères

  Exemple :

```
String[] couleurs = {"rouge", "bleu", "gris", "vert", "jaune", "noir" } ;
JComboBox combo = new JComboBox(couleurs) ;
```

Les boîtes combo « JComboBox » (2/3)

- Pour rendre une boîte combo éditable, on utilisera la méthode setEditable(boolean etat)
- Une boîte combo est dotée d'un ascenseur dès que son nombre de valeurs sera supérieur au nombre de valeurs visibles (8 par défaut , modifiable par la méthode setMaximumRowCount)
- La méthode setSelectedIndex(int rang) permet forcer la sélection d'un élément d'un rang donné.
- La méthode « Objet getSelectedItem() » fournit la valeur sélectionnée (ou saisie dans le champ texte).
- La méthode « int getSelectedIndex() » fournit le rang de la valeur sélectionnée. Elle fournit la valeur -1 lorsque l'utilisateur saisie une valeur dans le champ de texte.

Les boîtes combo « JComboBox » (3/3)

- La boîte combo dispose de méthodes appropriées à sa modification.
  - La méthode « addItem(valeur) » permet d'ajouter une nouvelle valeur à la fin de la liste.
  - La méthode « insertItemAt (valeur, rang) » permet d'insérer une nouvelle valeur à un rang donné.
  - La méthode « removeItem(valeur) » permet de supprimer une valeur existante.
- La boîte combo génère les événements suivants:
  - Un évènement Action généré lors d'une sélection dans la liste ou lors de la validation du champ texte (lorsqu'il est éditable). Aucun événement Action n'est généré en cas de perte de focus.
  - Un événements Item (et non plus ListSelection comme la boîte de liste) à chaque modification de la sélection.

Exercice 4

On considère une application de conversion de devises. Les taux de change sont indiques sur la base du Dollar(USD) selon les barèmes suivant (on se limite à ces devises):

- 1 Dirham (MAD)= 0.10072 Dollar (USD)
- 1 Euro (EUR) = 1.08461 USD
- 1 Livre (GBP) =1.45700 USD
- 1 Yen (JPY)= 0.00847 USD

La conversion se fera en fonction des éléments sélectionnes dans les 2 listes (objet JComboBox).

La classe JTextArea

- La classe JTextArea permet de saisir et d'afficher une zone de texte sur plusieurs lignes.
- Un objet JTextArea peut être défini en précisant le nombre de lignes ainsi que la largeur de chaque ligne:

  JTextArea zoneTexte1 = new JTextArea(10,50); //10 lignes de largeur 50

  JTextArea zoneTexte2 = new JTextArea("Texte"); // contenu initialisé
- La méthode « void setRows(int lignes) » permet de modifier le nombre de lignes de la zone de texte.
- La méthode « void setColumns(int colonnes) » permet de modifier le nombre de colonnes de la zone de texte.
- La méthode « void setLineWrap(true) » permet d'activer le retour automatique à la ligne pour éviter que des longues lignes ne soient tronquées.
- La méthode « void setWrapStyleWord(true) » permet d'éviter que les mots ne soient coupés en fin de ligne.
- Pour gérer le défilement du texte au sein d'une zone textuelle, on le met dans un conteneur de type JScrollPane.

Les événements liés à la focalisation

- La focalisation (prise/perte du focus) d'un composant génère un événement de la catégorie FocusEvent. L'interface FocusListener dispose de méthodes suivantes (On peut savoir si un composant donné possède le focus en appelant sa méthode hasFocus):
  - La méthode focusGained utiliser pour traiter la prise du focus.
  - La méthode focusLost permet de traiter la perte du focus.
- On peut forcer un composant à recevoir le focus par la méthode requestFocus de la classe JComponent :

```
compo.requestFocus() ;   // force le focus sur le composant compo
```

- L'ordre dans lequel les différents composants d'un conteneur sont parcourus lorsqu'on déplace le focus par le clavier est fixé par l'ordre dans lequel ils ont été ajoutés au conteneur.
- Pour imposer un ordre particulier de parcourt clavier (déplacement du focus) des composants d'un conteneur, on précise le rang correspondant lors de l'appel de méthode add :

```
conteneur.add (composant , rang ) ;
```


La classe JScrollBar

La classe JScrollBar permet de définir des ascenseurs horizontaux ou verticaux. Ses principales méthodes sont :

- JScrollbar(int orientation, int valiniale ,int pasPage, int valMax, int valMin) . La valeur de l'orientation est : JScrollbar.HORIZONTAL ou JScrollbar.VERTICAL
- « int getValue() » et « void setValue(int) » qui permettent respectivement de retourner ou de définir la positon de l'ascenseur
- « int getBlockIncrement() » et « void setBlockIncrement (int) » qui permettent respectivement de retourner ou de définir la valeur utilisée pour le pas en mode page à page
- « int getUnitIncrement() » et « void setUnitIncrement (int) » qui permettent respectivement de retourner ou de définir la valeur utilisée pour le pas unitaire.
- « int getMaximum() » , « int getMinimum()», « void setMaximum (int) » et « void setMinimum (int) » qui permettent de retourner ou de définir la valeur maximale ou la valeur minimale.

La classe JScrollBar

- La méthode adjustmentValueChanged de l'interface AdjustmentListener permet de traiter le changement de l'état de la barre de défilement
- Pour pallier à la redondance de l'événement généré, on dispose d'une méthode getValueIsAdjusting permettant de savoir si l'on est ou non en phase de transition.

```
public void adjustmentValueChanged(AdjustmentEvent ae) {
    if (ae.getValueIsAdjusting()) return;
    valeur=ae.getValue();
    …
  }
});
```


Exercice 5

On considère une application dont l'aspect graphique est le suivant:

Assurer la synchronisation entre la zone de texte, l' ascenseur et le bouton « initialiser »

Exercice 5

```
class Fenetre extends JFrame{
   private JTextField txt1;  private JButton btnInit; private   JScrollBar sBar;
   private     Integer valeur;
   public Fenetre() {
        …
        sBar= new JScrollBar(JScrollBar.HORIZONTAL, 0, 10, 0, 1000);
        sBar.setPreferredSize(new Dimension(120, 20));
        …
        sBar.addAdjustmentListener(new AdjustmentListener() {
            public void adjustmentValueChanged(AdjustmentEvent ae) {
                if (sBar.getValueIsAdjusting()) return;
                valeur=ae.getValue();  txt1.setText(valeur.toString());  });
        btnInit.addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent arg0) {
                valeur=0; sBar.setValue(0); txt1.requestFocus(); }    }   );
        txt1.getDocument().addDocumentListener(new DocumentListener() {
            public void removeUpdate(DocumentEvent e) {
                valeur=Integer.parseInt(txt1.getText()  ); sBar.setValue(valeur); }
            public void insertUpdate(DocumentEvent e) {
                valeur=Integer.parseInt(txt1.getText()  ); sBar.setValue(valeur);; }
            public void changedUpdate(DocumentEvent e) { }     });
}
```


Les événements liés au clavier

- Les événements générés par le clavier appartiennent à la catégorie KeyEvent. Ils sont gérés par un écouteur implémentant l'interface KeyListener qui comporte trois méthodes :
  - keyPressed, appelée lorsqu'une touche a été enfoncée,
  - keyReleased, appelée lorsqu'une touche a été relâchée,
  - keyTyped, appelée (en plus des deux précédentes), lors d'une succession d'actions correspond à un caractère Unicode(Avec une touche telle que Alt on obtient seulement un appel de keyPressed, suivi d'un appel de keyReleased, sans aucun appel de keyTyped).
- Un objet keyEvent contient les informations nécessaires à l'identification de la touche ou du caractère concerné:
  - La méthode getKeyChar fournit le caractère concerné (sous la forme d'une valeur de type char).
  - La méthode getKeyCode fournit un entier nommé code de touche virtuelle permettant d'identifier la touche concernée.
  - La méthode statique getKeyText permet d'obtenir, sous la forme d'une chaîne, un bref texte expliquant le rôle d'une touche de code donné.

Les événements liés au clavier

Code (KeyEvent)	Touche correspondante
VK_0 à VK_9	0 à 9 (pavé alphabétique)
VK_NUMPAD0 à VK_NUMPAD9	0 à 9 (pavé numérique)
VK_A à VK_Z	A à Z
VK_F1 à VK_F24	fonction F1 à F24
VK_ALT	modificatrice Alt
VK_ALT_GRAPH	modificatrice Alt graphique
VK_CAPS_LOCK	verrouilllage majuscules
VK_CONTROL	Ctrl
VK_DELETE	Suppr
VK_DOWN	flèche bas
VK_END	Fin
VK_ENTER	de validation

Code	Touche correspondante
VK_ESCAPE	Echap
VK_HOME	Home
VK_INSERT	Insert
VK_LEFT	flèche gauche
VK_NUM_LOCK	verrouillage numérique
VK_PAGE_DOWN	Page suivante
VK_PAGE_UP	Page précédente
VK_PRINTSCREEN	Impression écran
VK_RIGHT	flèche droite
VK_SCROLL_LOCK	arrêt défilement
VK_SHIFT	majusucles temporaire
VK_SPACE	espace
VK_TAB	de tabulation

Les événements liés au clavier

- il est possible de connaître l'état des touches modificatrices (touches Shift, Alt, Alt graphic et Cntrl) au moment où il a été généré. Pour ce faire, on dispose des méthodes suivantes, qui fournissent la valeur true si la touche correspondante est pressée, la valeur false dans le cas contraire :

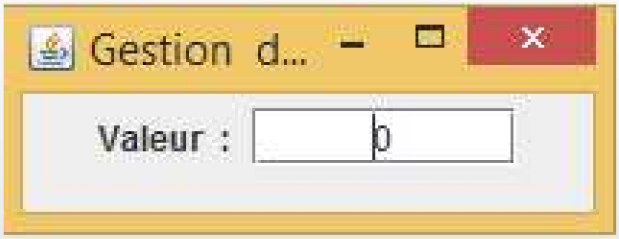
Méthode	Touche correspondante
isAltDown	Alt
isAltGraphDown	Alt graphique
isControlDown	Cntrl
isShiftDown	Shift
isMetaDown	L'une des quatres précédentes

- La méthode getModifiers fournit un entier qui indique l'état de ces touches et qu'on peut tester à l'aide de "masques binaires" définis dans la classe InputEvent : ALT_MASK, CONTROL_MASK, SHIFT_MASK et META_MASK.

```
void keyPressed (KeyEvent e) {
  if ((e.getKeyCode() == KeyEvent.VK_E) && ((e.getModifiers() &
       InputEvent.CTRL_MASK) != 0 )) // combinaison Cntrl/e
  {…}
}
```

Exercice 6

On considère une application dont l'aspect graphique est le suivant:



Assurer le contrôle de la zone de texte pour limiter la saisie aux chiffres. Prévoir une limite maximum du nombre de ces chiffres (5 par exemple).

Exercice 3

```
import java.awt.*; import java.awt.event.*;import javax.swing.*;
public class TextFieldFilter extends javax.swing.JFrame {
    JTextField jTextField_numeric;
    public TextFieldFilter() {
        this.setTitle("Gestion  du clavier"); setBounds(100,100,300,120);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel lbl = new JLabel("Valeur  : "); jTextField_numeric= new JTextField("0", 8);
        jTextField_numeric.setHorizontalAlignment(SwingConstants.CENTER);
        Container cont = getContentPane();
        JPanel pan1 = new JPanel(new FlowLayout());
        pan1.add(lbl);pan1.add(jTextField_numeric); cont.add(pan1);
        //Ajouter le listener pour avoir que des chiffres
        jTextField_numeric.addKeyListener(new KeyAdapter() {
              public void keyTyped(KeyEvent evt) {
                 char c = evt.getKeyChar();
                 int len=jTextField_numeric.getText().length();
                 if ((c < '0' || c > '9') || len>5) { evt.consume(); }  //suppression du caractère
              } });
    }
    public static void main(String args[]) {
         TextFieldFilter frame = new TextFieldFilter(); frame.setVisible(true);      }
}
```

Les dessins en java

- Pour obtenir un dessin permanent dans un composant, il faut le réaliser en redéfinissant la méthode paintComponent du composant :

  void paintComponent (Graphics g)

  L'argument g est un contexte graphique qui sert d'intermédiaire entre les demandes de dessins et leur réalisation effective.
- La méthode paintComponent est automatiquement appelée par Java chaque fois que le composant a besoin d'être dessiné ou redessiné (cas de modification de sa taille, de déplacement, de restauration après une réduction en icône...).
- La démarche la plus raisonnable consiste à dessiner dans un panneau (un objet dérivée de la de classe *JPanel*).
- Dans la méthode paintComponent à redéfinir, il faut appeler explicitement la méthode paintComponent de la classe ascendante JPanel et ceci avant de réaliser le dessin (super.paintComponent(g));
- Pour forcer l'appel prématuré de *paintComponent* on peut faire appel à la méthode *repaint*.

Les dessins en java

- La méthode drawOval(int x1, int y1, int width, int height) permet de dessiner un rond vide.
- La méthode drawRect(int x1, int y1, int width, int height) dessine un rectangle vide.
- La méthode drawRoundRect(int x1, int y1, int width, int height, int arcWidth, int arcHeight) permet de dessiner un rectangle qui sera arrondi (l'arrondi est défini par la valeur des deux derniers paramètres).
- La méthode drawLine(int x1, int y1, int x2, int y2) permet de tracer des lignes droites.
- La méthode drawPolygon(int[] x, int[] y, int nbrePoints) permet de dessiner des polygones en définissant les coordonnées de tous les points ainsi que le nombre de points du polygone.
- La méthode drawPolyline(int[] x, int[] y, int nbrePoints) permet de tracer plusieurs lignes (elle ne fait pas le lien entre la première et la dernière valeur des tableaux).
- La méthode drawString(String str, int x, int y) permet d'écrire du texte à partir d'une position donnée.
- Certaines de ces méthodes possèdent également leur homologue pour dessiner des formes remplis (fillOval, fillRect, fillPolygon).

Exercice 7

On propose une interface permettant d'afficher les formes suivantes :

- Un rond, forme par défaut ;
- Un carré ;
- Un triangle ;
- Un polygone
- Une étoile.

La liste JComboBox contiendra les cinq choix des formes

Exercice 7

```
import java.awt.*; import javax.swing.*; import java.awt.event.*;
public class Panneau extends JPanel {
  private int posX = 125;   private int posY = 10; private String forme = "ROND";
  public void paintComponent(Graphics g){
      g.setColor(Color.white);g.fillRect(0, 0, this.getWidth(), this.getHeight());
      g.setColor(Color.red);    draw(g);   }
  private void draw(Graphics g){
      if(this.forme.equals("ROND")){ g.fillOval(posX, posY, 50, 50); }
      if(this.forme.equals("CARRE")){ g.fillRect(posX, posY, 50, 50); }
      if(this.forme.equals("TRIANGLE")){
         int[] ptsX={posX + 25, posX + 50, posX}; int[] ptsY={posY, posY + 50, posY + 50};
        g.fillPolygon(ptsX, ptsY, 3);  }
      if(this.forme.equals("POLYGONE")){
         int[] ptsX={posX, posX+13, posX+37, posX+50, posX+50, posX+37, posX+13, posX  };
         int[] ptsY={ posY+13, posY, posY, posY+13, posY+37, posY+50, posY+50, posY+37 };
         g.fillPolygon(ptsX, ptsY,8) ;     }
      if(this.forme.equals("ETOILE")){
         int[] ptsX={ posX, posX+25, posX+50, posX+50, posX+50, posX+25, posX , posX };
         int[] ptsY={ posY, posY, posY, posY+25, posY+50, posY+50, posY+50 , posY+25 };
         g.drawLine(ptsX[0],ptsY[0],ptsX[4],ptsY[4]);g.drawLine(ptsX[1],ptsY[1],ptsX[5], ptsY[5]);
         g.drawLine(ptsX[2],ptsY[2],ptsX[6],ptsY[6]);g.drawLine(ptsX[3],ptsY[3],ptsX[7],ptsY[7]); }
  }
  public void setForme(String form){ this.forme = form; }
}
```

Exercice 7

```
class Fenetre extends JFrame{
  private Panneau pan = new Panneau(); private JPanel container = new JPanel();
  private JLabel label = new JLabel("Choix de la forme");
  private JComboBox combo = new JComboBox();
  public Fenetre(){
    this.setTitle("Dessin de formes"); this.setSize(300, 150);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    container.setBackground(Color.white);
    container.setLayout(new BorderLayout());container.add(pan, BorderLayout.CENTER);
    combo.addItem("ROND"); combo.addItem("CARRE"); combo.addItem("TRIANGLE");
    combo.addItem("POLYGONE"); combo.addItem("ETOILE");
    combo.addActionListener(new FormeListener());
    JPanel top = new JPanel(); top.add(label); top.add(combo);
    container.add(top, BorderLayout.NORTH); this.setContentPane(container);
    this.setVisible(true);
  }
  class FormeListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        pan.setForme(combo.getSelectedItem().toString());  pan.repaint(); }
  }
}
public class TestDesin { public static void main(String[] args){ Fenetre fen =new Fenetre();}
```


Les événements liés à la souris

- L'interface MouseListener comporte cinq méthodes correspondant chacune à un événement particulier : mousePressed, mouseReleased, mouseEntered, mouseExited et mouseClicked ("clic complet" lors d'appui et relâchement d'un bouton à condition que la souris n'ait pas été déplacée entre temps).
- La méthode getModifiers de la classe MouseEvent permet de connaître le(s) bouton(s) de la souris concerné(s) par l'événement. Elle fournit un entier dans lequel un bit de rang donné est associé à chacun des boutons et prend la valeur 1 pour indiquer un appui. La classe InputEvent contient des "masques" afin de tester le bouton concerné.

Masque	Bouton correspondant
InputEvent.BUTTON1_MASK	gauche
InputEvent.BUTTON2_MASK	central (s'il existe)
InputEvent.BUTTON3_MASK	droite

Exemple : tester si le bouton droit est enfoncé:

```
if ((e.getModifiers() & InputEvent.BUTTON3_MASK) != 0) ...
```

- Lorsqu'on s'intéresse uniquement au relâchement du bouton droit on peut se contenter d'utiliser la méthode isPopupTrigger.

Les événements liés à la souris

- La méthode getClickCount (de la classe MouseEvent) permet de gérer doubles clics. Elle fournit un compteur de clics successifs en un même point. Ce compteur est remis à zéro lors d'un déplacement de la souris ou lorsqu'un certain temps (paramétre de l'environnement) s'est écoulé après un clic.
- Lors du déplacement de la souris (même sans cliquer sur un de ses boutons), plusieurs événements peuvent être générés:
  - Un événement mouseEntered chaque fois que la souris passe de l'extérieur à l'intérieur d'un composant.
  - Un événement mouseExited chaque fois que la souris passe de l'intérieur à l'extérieur d'un composant.
  - Un événement mouseMoved (de l'interface MouseMotionListener) lors du déplacement sur un composant sans qu'un bouton ne soit enfoncé.
  - Un événement mouseDragged (de l'interface MouseMotionListener) si un bouton est resté enfoncé pendant le déplacement sur un composant.

Exemple

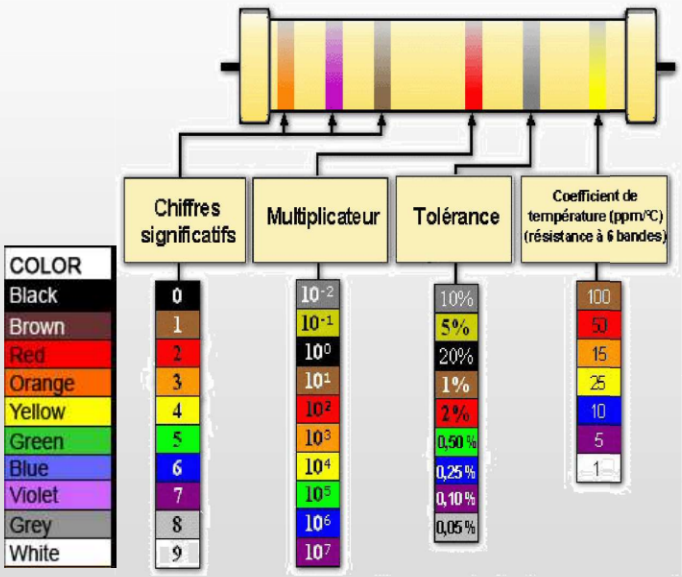
```
import java.awt.*; import javax.swing.*; import java.awt.event.*; import javax.swing.event.* ;
class MaFenetre extends JFrame {
   public MaFenetre () { setTitle ("Essais drag souris") ; setSize (300, 200) ;
      paneau = new Paneau() ; getContentPane().add(paneau) ; }
   private JPanel paneau ; }
class Paneau extends JPanel implements MouseMotionListener {
   Paneau() {
      addMouseMotionListener(this) ;
      addMouseListener (new MouseAdapter() {
           public void mouseReleased (MouseEvent e) { enCours = false ;}    }) ;
      repaint() ; }
   public void mouseDragged (MouseEvent e) {
      if (!enCours) { xDeb=e.getX(); yDeb=e.getY(); xFin=xDeb; yFin=yDeb; enCours=true; }
           else { xFin = e.getX() ; yFin = e.getY() ; }
      repaint() ; }
   public void mouseMoved (MouseEvent e) { }
   public void paintComponent (Graphics g) {
      super.paintComponent(g) ;  int xd, xf, yd, yf ;
      xd = Math.min (xDeb, xFin) ; xf = Math.max (xDeb, xFin) ;
      yd= Math.min(yDeb, yFin); yf= Math.max(yDeb, yFin); g.drawRect(xd, yd, xf-xd, yf-yd); }
   private boolean enCours = false ; private int xDeb, yDeb, xFin, yFin ;   }
public class Drag1 {
   public static void main (String args[]){MaFenetre fen=new MaFenetre(); fen.setVisible(true);}}
```

Récapitulatif des catégories d'événements

Catégorie	Nom de l'interface	Méthodes	
Action	ActionListener	actionPerformed	(ActionEvent)
Item	ItemListener	itemStateChanged	(ItemEvent)
Mouse	MouseMotionListener	mouseDragged - mouseMoved	(MouseEvent)
Mouse	MouseListener	mousePressed – mouseReleased – mouseEntered - mouseExited - mouseClicked	(MouseEvent)
Key	KeyListener	keyPressed - keyReleased - keyTyped	(KeyEvent)
Focus	FocusListener	focusGained - focusLost	(FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged	(AdjustmentEvent)
Component	ComponentListener	componentMoved - componentHiddent – componentResize - componentShown	(ComponentEvent)
Window	WindowListener	windowClosing - windowOpened - windowIconified - windowDeiconified windowClosed - windowActivated - windowDeactivated	(WindowEvent)
Container	ContainerListener	componentAdded - componentRemoved	(ContainerEvent)
Text	TextListener	textValueChanged	(TextEvent)

Exercice 8

Réaliser une application JAVA permettant de simuler le calcul des codes couleurs d'une résistance.



Exemple

- 1 chiffre significatif : jaune → 4
- 2 chiffre significatif : vert → 5
- Multiplicateur: orange→ 3
- Tolérance : argent : 10 %

Donc la valeur de cette résistance est :
45 x 10^3 à 10 % soit 45 k à 10 %.

Interface Homme-Machine en JAVA

Chapitre 3: Les boîtes de dialogue et les menus déroulants

Pr LAHCEN MOUMOUN
estblearn@gmail.com

Les boîtes de dialogue

La classe JOptionPane dispose d'un certain nombre de boîtes de dialogue simples qui sont utilisées très fréquemment dans toutes sortes d'applications :

- Les boîtes de message
- Les boîtes de confirmation
- Les boîtes de saisie
- Les boîtes d'options

Les boîtes de message

- La classe JOptionPane du paquetage javax.swing dispose d'une méthode statique showMessageDialog permettant d'afficher des messages à l'aide d' une boîte de dialogue modale.
  - static void showMessageDialog(Component parent, Object message);
  - static void showMessageDialog(Component parent, Object message, String titre, int messageType);
- L'argument « parent » représente la fenêtre propriétaire de la boîte de message, c'est-à-dire celle dans laquelle elle va s'afficher.
- L'argument messageType est une constante choisi parmi la liste suivante :

Paramètre	Type
JOptionPane.ERROR_MESSAGE	Erreur
JOptionPane.INFORMATION_MESSAGE	Information
JOptionPane.WARNING_MESSAGE	Avertissement
JOptionPane.QUESTION_MESSAGE	Question
JOptionPane.PLAIN_MESSAGE	Aucune icône

Les boîtes de message

Exemple 1 (boîte de message usuelle):

```
JOptionPane.showMessageDialog(fen, "Hello"); // fen est un objet JFrame
```

Exemple 2:

```
JOptionPane.showMessageDialog(fen,"Mauvais choix","Message d'avertissement",
                              JOptionPane.ERROR_MESSAGE);
```

Les boîtes de confirmation

- La méthode (statique) showConfirmDialog de la classe JOptionPane permet d'afficher des boîtes modales dites "de confirmation" offrant à l'utilisateur un choix de type oui/non.
  - static int showConfirmDialog(Component parent, Object message);
  - static int showConfirmDialog(Component parent, Object message, String title, int optionType);
  - static int showConfirmDialog(Component parent, Object message, String title, int optionType, int messageType);
- Les boutons de la méthode showConfirmDialog sont définis par le paramètre entier optionType dont la valeur est choisie parmi les constantes suivantes :

Paramètre	Valeur	Type de boîte de confirmation
JOptionPane.DEFAULT_OPTION	-1	boîte usuelle
JOptionPane.YES_NO_OPTION	0	boutons YES et NO
JOptionPane.YES_NO_CANCEL_OPTION	1	boutons YES, NO et CANCEL
JOptionPane.OK_CANCEL_OPTION	2	boutons OK et CANCEL

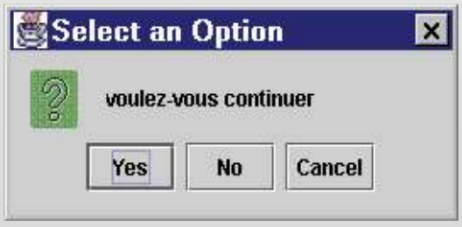
Les boîtes de confirmation

La valeur de retour de la méthode showConfirmDialog précise l'action effectuée par l'utilisateur, sous la forme d'un entier dont la valeur est la suivante :

Constante symbolique	Valeur	Signification
JoptionPane.OK_OPTION	0	action sur OK
JoptionPane.YES_OPTION	0	action sur YES
JoptionPane.NO_OPTION	1	action sur NO
JoptionPane.CANCEL_OPTION	2	action sur CANCEL
JoptionPane.CLOSED_OPTION	-1	fermeture de la boîte

Exemple 1 (boîte de confirmation usuelle):

```
int rep=JOptionPane.showConfirmDialog(fen, "voulez-vous continuer") ;
```

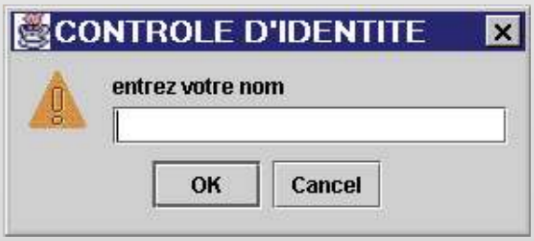


Les boîtes de saisie

- La méthode showInputDialog de la classe JOptionPane permet de saisir une information sous la forme d'une chaîne de caractères.
  - static Object showInputDialog(Component parent, Object message);
  - static Object showInputDialog(Component parent, Object message, String title, int messageType);
- La méthode showInputDialog fournit en retour :
  - soit un objet de type String contenant le texte fourni par l'utilisateur,
  - soit la valeur null si l'utilisateur n'a pas confirmé sa saisie par OK (action sur le bouton Cancel ou fermeture de la boîte).

Exemple :

```
String txt = JOptionPane.showInputDialog (null, "entrez votre nom",
"CONTROLE D'IDENTITE", JOptionPane.WARNING_MESSAGE) ;
```



Les boîtes d'options

- Java vous permet d'afficher une boîte d'options permettant un choix d'une valeur parmi une liste, par l'intermédiaire d'une boîte combo.
  - static Object showInputDialog(Component parent, Object message, Object[ ] options, Object default);
  - static Object showInputDialog(Component parent, Object message, String title, int messageType, Icon icon, Object[ ] options, Object default);
- La valeur de retour de la méthode showInputDialog, de type Object, correspond à l'option sélectionnée s'il y en a une, sinon à la valeur null. Sa conversion en String sera généralement nécessaire.

Exemple :

```
String[] couleurs = { "rouge", "vert", "bleu", "jaune",
    "orange", "blanc"} ;
String txt = (String)JOptionPane.showInputDialog (fen,
    "choisissez une couleur", "BOITE D'OPTIONS",
   JOptionPane.QUESTION_MESSAGE,  // type d'icône
   null, // icône supplémentaire
   couleurs, // tableau de chaînes présentées dans la boîte combo
   couleurs[1]) ; // chaîne sélectionnée par défaut
```

Les boîtes d'options

- Il existe également une méthode nommée showOptionDialog, affichant les options voulues, non plus sous la forme d'une boîte combo, mais à raison d'un bouton pour chacune des options.
- Le résultat fourni par showOptionDialog est un entier correspondant au rang de l'option choisie (ou à une valeur négative en cas d'absence de choix).
  - static int showOptionDialog(Component parent, Object message, Object[] options, Object default);
  - static int showOptionDialog(Component parent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object default);

Exemple :

```
int rang = JOptionPane.showOptionDialog (this,
"choisissez une couleur", "BOITE D'OPTIONS",
JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.QUESTION_MESSAGE, null,
couleurs, couleurs[1]) ;
```

Les boîtes de dialogue personnalisées

- Java dispose d'une classe JDialog qui permettra de créer des boîtes de dialogue personnalisées.

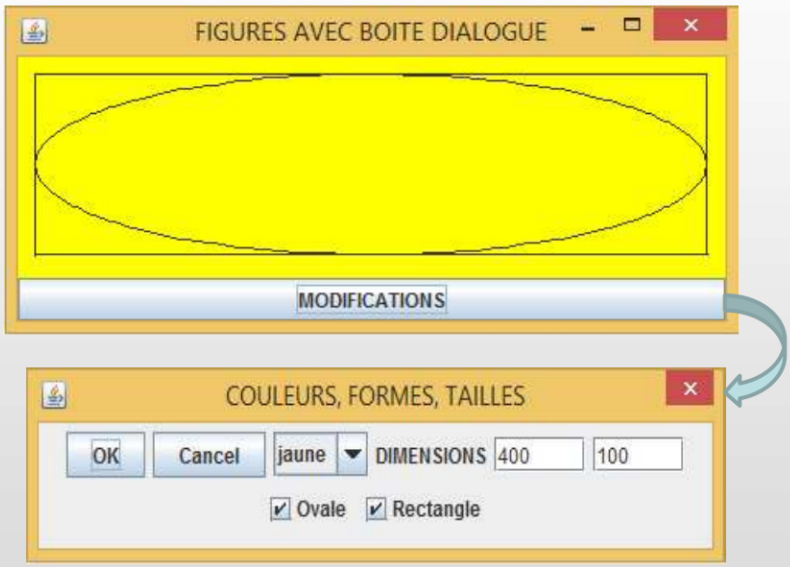
  Exemple:

```
JDialog bd = new JDialog (fen,          // fenêtre propriétaire (parent)
      "Ma boite de dialogue",     // titre de la boîte
       true) ;     // boîte modale
```

- Généralement, on est amené à créer une classe dérivée de JDialog, qu'on pourra spécialiser en introduisant les champs et les fonctionnalités dont on aura besoin.
- Les méthodes setSize et setBounds permettent de définir la taille d'une boîte de dialogue alors que l'appel de setVisible assure son affichage. La méthode dispose permet de libérer une boîte de dialogue, ainsi que les objets qui lui sont associés.
- Avec une boîte de dialogue modale, l'utilisateur ne peut pas agir sur d'autres composants que ceux de la boîte tant qu'il n'a pas mis fin au dialogue (l'instruction suivant l'appel de setVisible d'une boite ne sera exécutée tant que l'utilisateur n'aura pas mis fin au dialogue).

Exemple de boîtes de dialogue personnalisées

On considère une application d'affichage des formes graphiques (ovale et rectangle). La modification des paramètres de ces formes est obtenu par une boite de dialogue.



Exemple de boîtes de dialogue personnalisées

```
class Infos // pour les informations à échanger avec la boîte de dialogue
   { public boolean ovale,rectangle; public int largeur,hauteur; public String nomCouleur;}
class MaFenetre extends JFrame implements ActionListener {
  public static final String[] nomCouleurs = {"rouge",      "vert",     "jaune",   "bleu"} ;
  public static final Color[] couleurs= {Color.red, Color.green, Color.yellow, Color.blue};
  public MaFenetre () {
        setTitle ("FIGURES AVEC BOITE DIALOGUE") ; setSize (450, 200) ;
        Container contenu = getContentPane() ;
        pan = new PaneauDessin() ; contenu.add(pan) ; // panneau pour les dessins
        lanceDial = new JButton ("MODIFICATIONS") ; // bouton pour lancer la boite
        contenu.add (lanceDial, "South") ; lanceDial.addActionListener(this);}
  public void actionPerformed (ActionEvent ev) {
        if ( dialogue == null ) { dialogue = new Dialogue(this); infos = new Infos ();  }
        infos.largeur=pan.getLargeur(); // recup informations courantes dans l'objet infos
        infos.hauteur=pan.getHauteur(); infos.rectangle = pan.getRectangle() ;
         infos.ovale = pan.getOvale() ; infos.nomCouleur = pan.getNomCouleur() ;
        dialogue.lanceDial(infos) ; // lancement dialogue
        pan.setLargeur (infos.largeur) ; //prise en compte nouvelles informations
        pan.setHauteur (infos.hauteur) ; pan.setRectangle (infos.rectangle) ;
        pan.setOvale (infos.ovale) ; pan.setCouleur (infos.nomCouleur) ; pan.repaint() ;}
  private PaneauDessin pan ; private JButton lanceDial ; private Dialogue dialogue ;
  private Infos infos ; }
```

Exemple de boîtes de dialogue personnalisées

```
class Dialogue extends JDialog implements ActionListener {
    private JButton okBouton, cancelBouton ;  private boolean ok = false ;
    private JComboBox comboCoulFond ; private JTextField txtLargeur, txtHauteur ;
    private JCheckBox cOvale, cRectangle ;

    public Dialogue(JFrame parent) {
        super (parent, "COULEURS, FORMES, TAILLES", true) ; setSize (420, 100) ;
        Container contenu = getContentPane() ; okBouton = new JButton ("OK") ;
        contenu.add(okBouton) ; contenu.setLayout(new FlowLayout());
        okBouton.addActionListener(this) ; cancelBouton = new JButton ("Cancel") ;
        contenu.add(cancelBouton) ; cancelBouton.addActionListener(this) ;
        comboCoulFond = new JComboBox (MaFenetre.nomCouleurs) ; // choix couleur
        contenu.add(comboCoulFond); JLabel dim = new JLabel ("DIMENSIONS") ;
        contenu.add (dim) ; /* choix dimensions */
        txtLargeur = new JTextField (5) ; contenu.add (txtLargeur) ;
        txtHauteur = new JTextField (5) ; contenu.add (txtHauteur) ;
        cOvale = new JCheckBox ("Ovale") ; contenu.add (cOvale) ; /* choix formes */
        cRectangle = new JCheckBox ("Rectangle") ; contenu.add (cRectangle) ;  }
    …
    }
```


Exemple de boîtes de dialogue personnalisées

```
class Dialogue extends JDialog implements ActionListener {
    private JButton okBouton, cancelBouton ;  private boolean ok = false ;
    private JComboBox comboCoulFond ; private JTextField txtLargeur, txtHauteur ;
    private JCheckBox cOvale, cRectangle ;
    public Dialogue(JFrame parent) { …}
    public void lanceDial(Infos infos) {
        txtLargeur.setText(""+infos.largeur) ; /* placer infos dans controles */
        txtHauteur.setText (""+infos.hauteur) ; cOvale.setSelected (infos.ovale) ;
        cRectangle.setSelected (infos.rectangle) ;
        comboCoulFond.setSelectedItem (infos.nomCouleur) ;
        ok=false; setVisible(true); /*lancer le dialogue et on récupère les informations si ok
        if (ok) {
            infos.largeur = Integer.parseInt(txtLargeur.getText()) ;
            infos.hauteur = Integer.parseInt(txtHauteur.getText()) ;
            infos.rectangle = cRectangle.isSelected() ; infos.ovale = cOvale.isSelected() ;
            infos.nomCouleur = (String)comboCoulFond.getSelectedItem() ;  }
    }
    public void actionPerformed (ActionEvent e) {
        if (e.getSource() == okBouton) { ok = true ; setVisible(false) ; }
        if (e.getSource() == cancelBouton) setVisible(false) ;  }
    }
```

Exemple de boîtes de dialogue personnalisées

```
class PaneauDessin extends Jpanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g) ;
        if (ovale)      g.drawOval (10, 10, 10 + largeur, 10 + hauteur) ;
        if (rectangle) g.drawRect (10, 10, 10 + largeur, 10 + hauteur) ;    }
    public void setRectangle(boolean b) { rectangle = b ; }
    public boolean getRectangle () { return rectangle ; }
    public void setOvale(boolean b) { ovale = b ; }
    public boolean getOvale () { return ovale ; }
    public void setLargeur (int l){largeur = l; } public int getLargeur(){ return largeur; }
    public void setHauteur(int h){ hauteur=h;} public int getHauteur(){ return hauteur; }
    public void setCouleur (String c) {
        for (int i = 0 ; i<MaFenetre.nomCouleurs.length ; i++)
            if (c == MaFenetre.nomCouleurs[i]) setBackground (MaFenetre.couleurs[i]) ;
                nomCouleur = c ; }
    public String getNomCouleur ()      { return nomCouleur ; }
    private boolean rectangle = false, ovale = false ; private int largeur=50, hauteur=50 ;
    private Color couleur ; private String nomCouleur = MaFenetre.nomCouleurs[0] ;
}
public class ExDial {
    public static void main (String args[]) {
        MaFenetre fen = new MaFenetre() ; fen.setVisible(true) ; }
}
```


Les menus déroulants

- Les menus déroulants usuels(rattachés à une barre de menus) font intervenir trois sortes d'objets : un objet barre de menus (**JMenuBar**), différents objets menu (**JMenu**) qui seront visibles dans la barre de menus et pour chaque menu, les différentes options(**JMenuItem**), qui le constituent.

  JMenuBar barreMenus=new **JMenuBar**(); //création d'une barre de menus
  fen.**setJMenuBar**(barreMenus); // rattache l'objet barreMenus à la fenêtre fen
  JMenu menu1 = new **JMenu** ("nomMenu1") ; // cree un menu
  barreMenus.**add**(menu1) ; // ajouter le menu à barreMenus
  JMenuItem option1 = new **JMenuItem** ("nomOption1") ; // crée une option
  menu1.**add**(option1) ; // l'ajoute l'option au menu
- Dans un menu, il est possible d'introduire une barre séparatrice entre deux options, en recourant à la méthode **addSeparator()** de la classe **JMenu**
- Chaque action sur une option (**JMenuItem**) génère un événement Action qu'on peut traiter en associant un écouteur à l'objet correspondant

Les menus déroulants

- Autre que les options **JMenuItem** qui sont les plus usuelles, on peut utiliser dans un menu : des options cases à cocher (**JCheckBoxMenuItem**) et des options boutons radio (**JRadioButtonMenuItem**)
- Les options boutons radio peuvent être placées dans un groupe (objet de type **ButtonGroup**) de manière à assurer l'unicité de la sélection à l'intérieur du groupe.
- Les événements générés par ces nouvelles options sont les événements **Action** et **Item** (les événements *Item* sont traités par la méthode ***itemStateChanged***)
- La méthode **isSelected** (de la classe JRadioButtonMenuItem ou JCheckBoxMenuItem) pour savoir si une option est sélectionnée

Les menus déroulants

- La méthode **setMnemonic** de la classe **AbstractButton** (dont dérivent, entre autre, les classes menus et options de menus) permet d'associer un caractère mnémonique à un menu (combinaison Alt/Caractère) ou à une option (frappe simplement sur le caractère lorsque le menu de l'option est affiché)

```
menu1.setMnemonic ('M') ; // M = caractere mnemonique du menu menu1
```

- Un accélérateur est une combinaison de touches qu'on associe à une option (jamais à un menu) et qui s'affiche à droite de son nom

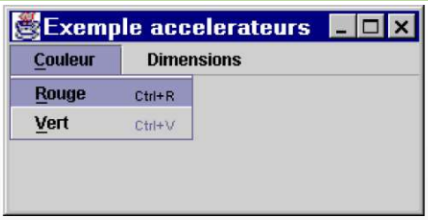
```
JMenuItem option1 = new JMenuItem ("nomOption1") ;
Option1.setAccelerator(
   KeyStroke.getKeyStroke( // Construire la combinaison de touches
   KeyEvent.VK_O,   // Touche O
   InputEvent.CTRL_MASK)) ; // + touche CTRL
```

Les menus déroulants

- La méthode setEnabled permet d'activer ou de désactiver un menu ou une option.
- Java permet d'obtenir un affichage d'infos bulle pour n'importe composant. Il suffit pour cela de lui associer le texte voulu à l'aide de la méthode **setToolTipText**

  Option1.setToolTipText ("TexteInfoBulle") ;
- Pour obtenir une hiérarchie d'option (une option qui fait faire apparaître une liste de sous options), il suffit d'utiliser dans un menu une option qui soit non plus de type JMenuItem, mais de type JMenu

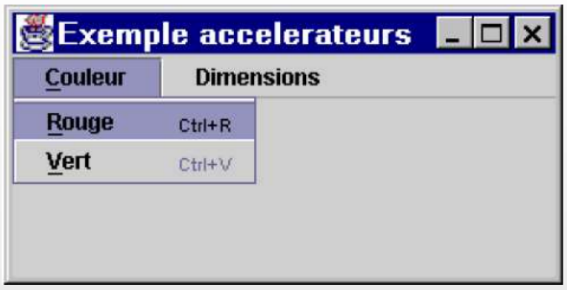
Exemple de menus déroulants




```
import java.awt.*; import java.awt.event.*;
import javax.swing.* ; import javax.swing.event.*;
class FenMenu extends JFrame implements ActionListener {
  public FenMenu () {
    setTitle ("Exemple de menu") ; setSize (300, 150) ;
    barreMenus = new JMenuBar(); setJMenuBar(barreMenus) ; //Barre des menus
    couleur = new JMenu ("Couleur") ; couleur.setMnemonic ('C') ; // Menu Couleur
    barreMenus.add(couleur);
    rouge=new JMenuItem ("Rouge");  rouge.setMnemonic ('R') ;
    rouge.setToolTipText ("Fond Rouge") ;
    rouge.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R,InputEvent.CTRL_MASK)) ;
    couleur.add(rouge) ;rouge.addActionListener (this) ;
    vert = new JMenuItem ("Vert") ; vert.setMnemonic ('V') ;
    vert.setToolTipText ("Fond Vert ") ;
    rouge.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_V, InputEvent.CTRL_MASK));
    couleur.add(vert) ;vert.addActionListener (this) ;
    dimensions = new JMenu ("Dimensions") ;barreMenus.add(dimensions) ;
    largeur = new JMenuItem ("Largeur") ;dimensions.add(largeur) ;
    largeur.addActionListener (this) ;hauteur = new JMenuItem ("Hauteur") ;
    dimensions.add(hauteur) ;hauteur.addActionListener (this) ;
  }
```

Exemple de menus déroulants

```
class FenMenu extends JFrame implements ActionListener {
  public FenMenu () { …}
    public void actionPerformed (ActionEvent e) {
        Object source = e.getSource() ;
        if (source == rouge) {…}
        if (source == vert) {…}
        if (source == largeur) {…}
        if (source == hauteur) {…}
    }
    private JMenuBar barreMenus ;
        private JMenu couleur, dimensions ;
        private JMenuItem rouge, vert, largeur, hauteur ;
    }
public class Menu1{
    public static void main (String args[]){
        FenMenu fen = new FenMenu() ;
        fen.setVisible(true) ;
    }
}
```



Les menus surgissants

- Un objet de type JPopupMenu, auquel on rattache des objets de type JMenuItem permet d'obtenir un menu surgissant (liste d'options apparaît suite à une certaine action de l'utilisateur, en général un clic sur le bouton droit de la souris)
- Un menu surgissant doit être affiché explicitement en utilisant la méthode show de la classe JPopupMenu. Celle-ci nécessite qu'on lui précise 2 arguments : le composant concerné (fenêtre en général) et les coordonnées du coin supérieur gauche du menu.
- Les événements générés par les options d'un menu surgissant restent les événements Action et éventuellement Item.
- la classe MouseEvent dispose d'une méthode isPopupTrigger qui fournit la valeur true si le bouton concerné est celui traditionnellement réservé aux menus surgissants.

Exemple de menus surgissants

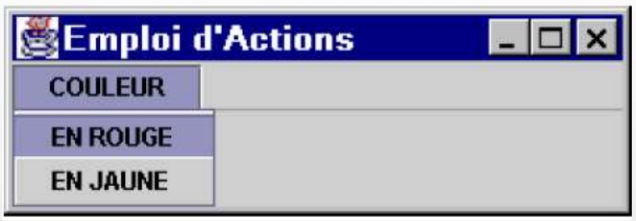
```
class FenMenu extends JFrame implements ActionListener{
  public FenMenu (){
    setTitle ("Exemple de menu surgissant") ; setSize (400, 120) ;
    /* creation menu surgissant Couleur et ses options Rouge et Vert */
    couleur= new JPopupMenu (); rouge= new JMenuItem ("Rouge") ;couleur.add(rouge);
    rouge.addActionListener (this) ;vert = new JMenuItem ("Vert") ;couleur.add(vert) ;
    vert.addActionListener (this) ;
    addMouseListener (new MouseAdapter() {
      public void mouseReleased (MouseEvent e) {
        if (e.isPopupTrigger()) couleur.show (e.getComponent(), e.getX(), e.getY()) ;}
    } ) ;
  }
  public void actionPerformed (ActionEvent e){
    Object source = e.getSource() ;
    if (source == rouge) {…}
    if (source == vert) {…}
  }
  private JPopupMenu couleur ; private JMenuItem rouge, vert ;
}
```

Les menus surgissants

- Lors de l'utilisation d'un menu déroulant et d'un menu réagissant, le traitement d'une action donnée ne doit être réalisé qu'en un seul point du code. Une telle action peut être représente par AbstractAction
- La classe AbstractAction dispose d'une méthode actionPerformed qu'on peut redéfinir à volonté dans n'importe quelle classe dérivée. Un objet action est obligatoirement un écouteur des événements Action, et cet écouteur se trouve automatiquement associé au composant correspondant lors de l'exécution de la méthode add. Ainsi, un objet de type JMenuItem, ayant pour libellé le nom de l'action correspondante

Exemple de menus déroulants

```
class MonAction extends AbstractAction {
   public MonAction (String nom, Color couleur) { super (nom) ; this.couleur = couleur ;}
   public void actionPerformed (ActionEvent e){ if (couleur == Color.red) {…} …}
   private Color couleur ;
}
class MaFenetre extends JFrame{
   public MaFenetre (){
      setTitle ("Emploi d'Actions ") ;
      setSize (300, 100) ;
      menu = new JMenuBar() ; setJMenuBar (menu) ;
      menuCouleur = new JMenu("COULEUR") ;
      actionRouge = new MonAction ("EN ROUGE", Color.red) ;
      actionJaune = new MonAction ("EN JAUNE", Color.yellow) ;
      menuCouleur.add(actionRouge) ; menuCouleur.add(actionJaune) ;
      menu.add(menuCouleur) ;
   }
   private MonAction actionRouge, actionJaune ;
   private JMenuBar menu ; private JMenu menuCouleur ;
   private JMenuItem optionRouge, optionJaune ;
   }
```



Interface Homme-Machine en JAVA

Chapitre 4: JDBC

Pr LAHCEN MOUMOUN
estblearn@gmail.com

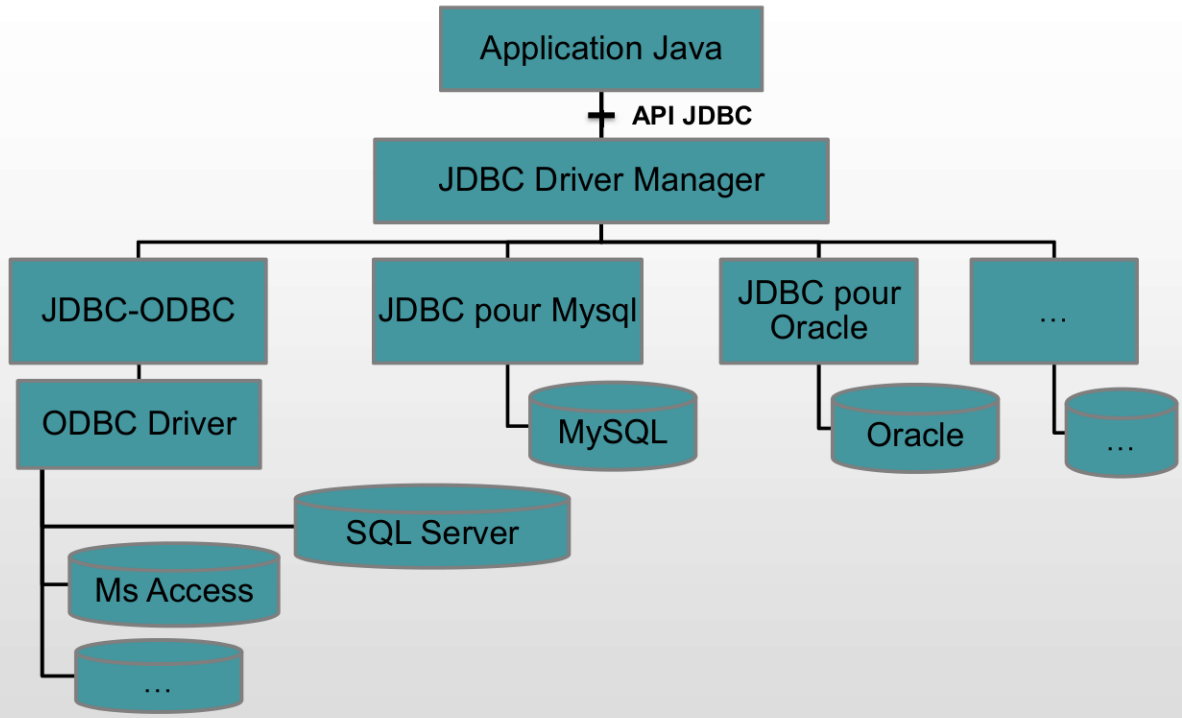
Java et JDBC

- JDBC (Java Data Base Connectivity) est une API (Application Programming Interface) Java permettant un programme d'application en Java d'accéder et manipuler une base de données en exécutant des instructions SQL.
- JDBC offre un accès uniforme une base de données:
  - Portable sur la plupart des OS
  - Indépendant du SGBD (seule la phase de connexion est spécifique – driver)
  - Compatible avec la plus part des SGBDR (Oracle, Postgres, MySQL, Informix, MS SQL Server…)

Pilotes JDBC

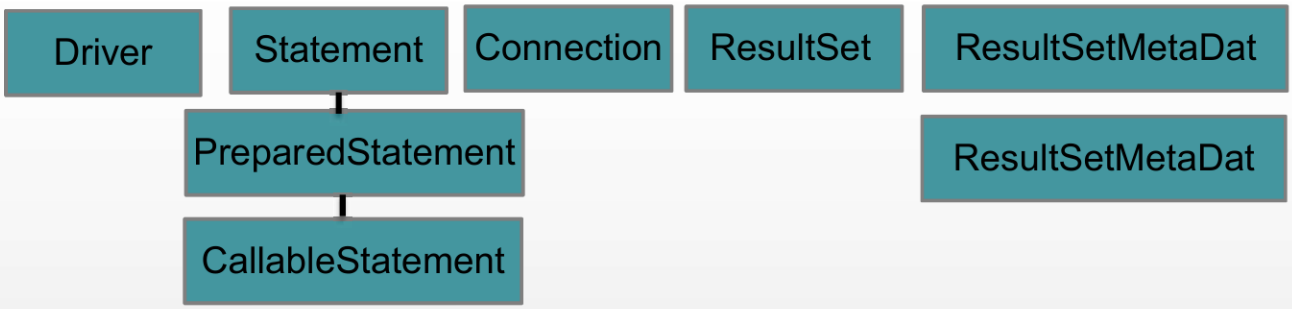
- L'API JDBC fait partie du JDK (Java Devlopment Kit). Elle est représentée par le paquetage java.sql.
- Pour travailler avec un SGBD, il faut disposer de classe qui implémente les interfaces de JDBC. Un ensemble de telles classes est désigné sous le nom de driver JDBC.
- Il existe un pilote particulier « JdbcOdbcDriver » qui permet à une application java communiquer avec n'importe quelle source de données via les pilotes ODBC (Open Data Base Connectivity)

Pilotes JDBC



- Driver Manager = Gestionnaire de tous les divers chargés par une application Java.
- Un gestionnaire JDBC prend la forme d'un fichier JAR, il faut le référencier et l'ajouter au projet JAVA

Les interfaces du package Java.SQL



- Driver : renvoie une instance de Connection
- Connection : connexion à une base
- Statement : instruction SQL
- PreparedStatement : instruction SQL paramétrée
- CallableStatement : procédure stockée dans la base
- ResultSet : enregistrements récupérés par une instruction SQL
- ResultSetMetaData : description des enregistrements récupérés
- DatabaseMetaData : informations sur la base de données

Exceptions liées aux bases de données

La classe SQLException du paquetage java.sql enrichit la classe Exception pour permettre de mieux gérer les éventuelles erreurs émise par une BD:

- String getMessage(): Envoie un message décrivant l'erreur.
- String getSQLState(): Envoie un code d'erreur associé à une requête SQL.
- int getErrorCode(): Envoie un code d'erreur spécifique au fournisseur du pilote.

Exceptions liées aux bases de données

On procède à la connexion à une base de données en deux étapes :

1. **Chargement du driver par la JVM** : il s'agit d'une implémentation de l'interface **Java.sql.Driver,** le driver peut être chargée en appelant la méthode **forName** de la classe **java.long.Class**

   Exemple

   ```
   Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" ) ;
   Class.forName("com.mysql.jdbc.Driver");
   Class.forName("oracle.jdbc.driver.OracleDriver");
   ```

2. **Connexion :** elle s'effectue en appelant la méthode getConnection de la classe **java.sql.DriverManager.**

   Pour un pilote com.mysql.jdbc.Driver :

   ```
   Connection conn=DriverManager.getConnection("jdbc:mysql:
               //localhost:3306/ DB", "user", "pass" );
   ```

   Le premier paramètre décrit le driver JDBC, la machine où tourne le SGBDR et le nom de la base de données.

Exemple

```
Import java.sql.*; import javax.swing.JOptionPane;
Public class testConecionJDBC{ // BD de test installée par défaut avec MYSQL
  public static void main ( String [] args){
    Connection conn=null;
    try {
      Class.forName("com.mysql.jdbc.Driver"); //Chargement du driver JDBC/MYSQL
      conn=DriverManager.getConnection("jdbc:mysql: //localhost:3306/test");
      JOptionPane.showMessageDialog(null, "Connexion OK" );}
    catch (ClassNotFoundException ex) { // si le driver n'est pas charger par forName
      JOptionPane.showMessageDialog(null, "Classe introuvable"   + ex.getMessage());}
     catch (SQLException ex) {
      JOptionPane.showMessageDialog(null, " Connexion Impossible"  +ex.getMessage());}
    finally {
      try {
         if (conn!=null) conn.close();} //Fermeture de la connexion
      Catch(SQLExeption ex) {ex.printStackTrace();}
    }
    System.exit(0); //arrêt de la JVM;
    }
}
```

Connexion JDBC (java.sql.Connection)

La connexion renvoyée par la méthode getConnction de la classe java.sql.DriverManager permet d'effectuer des opérations sur la base de données. Ces principales méthodes sont:

- createStatement : cette méthode renvoient une instance de java.sql.Statement utilisée pour exécuter une instruction SQL.
- prepareStatement : cette méthode précompilent des instructions SQL paramétrées et renvoient une instance de java.sql.PreparedStatement .
- prepareCall : cette méthode préparent l'appel aux procédures stockées de la BD et renvoient une instance de CallableStatement .
- setAutoCommit, commit, rollback : gèrent les transactions sur la BD.
- getMetaData : cette méthode renvoie une instance de java.sql.DatabaseMetaData pour obtenir des informations sur la base de données et sur ses possibilités.
- Close et isclosed : gèrent la fermeture d'une connexion.

Exécuter des instructions SQL (java.sql.Statement)

Les méthodes **createStatement** d'une connexion renvoient un objet dont la classe implémente l'interface **java.sql.Statement**. Les méthodes de cette interface les plus utilisées sont :

- executeUpdate : elle exécute l'instruction SQL en paramètre sur la BD, pour mettre à jour ses tables ou les enregistrements d'une table.
- executeQuery : elle exécute une sélection SQL se servant de l'instruction Select en paramètre et renvoient une instance de java.sql.ResultSet utilisée pour énumérer les enregistrements recherchés ligne par ligne.
- execute() : elle permet d'exécuter n'importe quelle instruction SQL. Cette méthode retourne un booléen valant true si l'instruction renvoie un objet ResultSet et false sinon (le résultat d'une requête de sélection pourra être obtenu par la méthode getResultSet).

Exploiter les résultats d'une sélection SQL

L'interface java.sql.ResultSet est dotée de deux catégories de méthodes, appelées tour à tour :

- Les méthodes next, first, last, beforeFirst, afterLast, relative, absolute (déplacement sur un enregistrement) sont utilisées pour changer de ligne : ces méthodes renvoient true s'il existe une ligne à la position choisie.
- Les méthodes get… comme getString, getInt, getDate, getObject …, renvoient la valeur d'un des champs d'une ligne. Chacune de ses méthodes existe sous deux formes qui prennent en paramètre l'identificateur ou le numéro d'ordre d'un champ de l'instruction Select (le numéro du premier champ est 1).

Types SQL et Types Java

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC Type	Java Type
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

Utilisation du JDBC

- Créer un objet Statement

  Statement statement = conn.createStatement();
- Exécuter une requête (l'objet ResultSet retourné contient les lignes du résultat de la requête)

  String query = "SELECT col1, col2, col3 FROM sometable";

  ResultSet resultSet = statement.executeQuery(query);
- Gérer le résultat

```
while(resultSet.next()) {
    System.out.println(resultSet.getString(1) + " " +
            resultSet.getString(2) + " " + resultSet.getString(3));
}
```

- Fermer la connexion

  connection.close();

Exemple

```
import java.sql.*;
import javax.swing.JOptionPane;

class CalculTotalFactures
{
  public static void main(String[] args)
  {
    Connection connexion  = null;
    Statement  instruction = null;
    ResultSet  resultat  = null;
    try
    {
      Class.forName ("com.mysql.jdbc.Driver");
      connexion = DriverManager.getConnection(
                                    "jdbc:mysql:///test"); 1
      instruction = connexion.createStatement();
      instruction.executeUpdate(
          "CREATE TABLE FACTURE(CLIENT CHAR(50),"
          + " ARTICLE VARCHAR(255), MONTANT DECIMAL(9,2))"); 2
```

Exemple

```
    instruction.executeUpdate(
        "INSERT INTO FACTURE (CLIENT, ARTICLE, MONTANT)"
        + " VALUES ('Thomas Durand', 'CDRx10', '6.35')"); 3
    instruction.executeUpdate(
        "INSERT INTO FACTURE (CLIENT, ARTICLE, MONTANT)"
        + " VALUES ('Sophie Martin', 'PC', '1500')");
    instruction.executeUpdate(
        "INSERT INTO FACTURE (CLIENT, ARTICLE, MONTANT)"
        + " VALUES ('Sophie Martin', 'Imprimante', '120.5')");

    resultat = instruction.executeQuery(
        "SELECT ARTICLE, MONTANT FROM FACTURE"
        + " WHERE CLIENT='Sophie Martin'"); 4
    String articles = "";
    double montantTotal = 0;

    while (resultat.next())
    {
      articles += resultat.getString ("ARTICLE") + " ";
      montantTotal += resultat.getDouble ("MONTANT"); 5
    }
    JOptionPane.showMessageDialog(null,
        "Articles : " + articles
        + "\nMontant total : " + montantTotal + " \u20ac");

    instruction.executeUpdate("DROP TABLE FACTURE");
}
```


Exemple

```
    catch (ClassNotFoundException ex)
    {
      JOptionPane.showMessageDialog(null,
            "Classe introuvable " + ex.getMessage ());
    }
    catch (SQLException ex)
    {
      JOptionPane.showMessageDialog(null,
            "Erreur JDBC : " + ex.getMessage ());
    }
    finally
    {
      try
      {
        if (resultat != null)
          resultat.close();
        if (instruction != null)
          instruction.close();
        if (connexion != null)
          connexion.close();
      }
      catch (SQLException ex)
      {
        ex.printStackTrace ();
      }
    }
    System.exit (0);
  }
}
```

Informations sur la BD (java.sql.DatabaseMetaData)

- La méthode **getMetaData** d'une connexion renvoie un objet dont la classe implémente l'interface **java.sql.DatabaseMetaData**. Les méthodes de cette interface permettent d'obtenir des informations complètes sur la base de données : tables, index, procédures stockées, champs, droits d'utilisation…
- La plupart de informations d'une BD sont renvoyées sous forme d'une instance de **java.sql.ResultSet** utilisée pour énumérer principalement les réponses suivantes:
  - **ResultSet getCatalogs() :** Retourner la liste du catalogue d'informations (avec JDBC-ODBC, on obtient la liste des BDs)
  - **ResultSet getTables**(String catalog, String schema, String tablemask, String types[]): Renvoie la liste des tables (utiliser le mask '%' pour retrouver toutes les tables).
  - **ResultSet getColumns** (catalog, schema, tableNames, columnNames): Retourner une description des colonnes correspondant au TableNames et aux columnNames.

Informations sur la BD (java.sql.DatabaseMetaData)

```
Connection connexion = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
    connexion = DriverManager.getConnection ("jdbc:mysql://localhost/test");
    DatabaseMetaData dmd = connexion.getMetaData();
    ResultSet tables = dmd.getTables(connexion.getCatalog(),null,"%",null);
    System.out.println("TABLE_NAME");
    while(tables.next()){
        String TypeTable = (String) tables.getObject("TABLE_TYPE");
        if (TypeTable.equals("TABLE")){
            System.out.println(tables.getString("TABLE_NAME"));
        }
        }
}
```

Informations sur une requête SQL

- L'objet **ResultsetMetaData** renvoyée par **getMetaData** d'un ResultSet permet de déterminer sa structure (nombre de champs, nom, type et taille de chaque champ)
- Les méthodes permettent de retrouver des informations concernant une table :

Méthodes	Résultat	Type
getColumnCount( )	Nombre de colonne constituant la table	int
getTableName(i)	Nom de la table	String
getColumnName(i)	Renvoie le nom de la colonne	String
getColumnLabel(i)	Fournit le titre suggéré de la colonne	String
getColumnTypeName(i)	Type de la colonne au sens base de données	String
getColumnDisplaySize(i)	Taille de la colonne	int
isNullable(i)	Si la colonne accepte des valeurs nulles	int
isAutoIncrement(i)	Si la colonne s'auto-incrémente	boolean

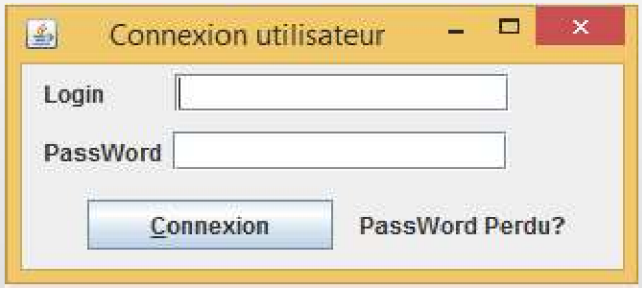
Informations sur une requête SQL

```
Statement instruction = connexion.createStatement();
ResultSet table = instruction.executeQuery("select * from emp");
ResultSetMetaData infoTable = table.getMetaData();
System.out.println("Nom de la table : " + infoTable.getTableName(1));
System.out.println("Nombre de colonnes : "+infoTable.getColumnCount());
for (int i=1; i<=infoTable.getColumnCount(); i++) {
    System.out.print("------------------------ : ");
    System.out.println("Colonne n°"+i);
    System.out.println("Label : "+infoTable.getColumnLabel(i));
    System.out.println("Nom : "+infoTable.getColumnName(i));
    System.out.println("Type : "+infoTable.getColumnTypeName(i));
    System.out.println("Taille : "+infoTable.getColumnDisplaySize(i));
    System.out.println("Null ? "+infoTable.isNullable(i));
    System.out.println("Auto-incrément ? "+infoTable.isAutoIncrement(i));
}
```

Exercice

Considérons une base de donnée comportant une table utilisateur . Cette table contient pour chaque utilisateur son login (clé) et son mot de passe :

Utilisateurs (login, password)



Paramétrer une requête SQL

- Les méthodes **PrepareStatement** d'une connexion précompilent une instruction SQL paramétrée. Ainsi via un objet **PreparedStatement** on optimise les traitements (le SGBD analyse une seule fois la requête).
- Chaque paramètre d'une requête paramétrée est symbolisé par un point d'interrogation (?) remplacé par une valeur lors de l'exécution de l'instruction.
- L'interface **java.sql.PreparedStatement** définit (en plus des méthode de **java.sql.Statement**) les méthodes:
  - Les méthodes void **set[type]** (int ordre, [type] val), comme setString, set Int, …, utilisées pour spécifier la valeur de chaque paramètre de l'instruction SQL précompilée (le premier numéro d'ordre 1).
  - Les deux méthodes sans paramètre **executeUpdate** et **executeQuery**, appelées pour exécuter l'instruction SQL précompilée en remplaçant les ? Par la valeur des paramètres.

Informations sur une requête SQL

```
PreparedStatement stmt=connexion.prepareStatement
                                        ("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 spécifies le premier paramètre de la requête
stmt.setString(2,"Ratan");
int i=stmt.executeUpdate();
System.out.println(i+ " enregistrement inséré");
stmt=connexion.prepareStatement ("select * from emp , dept where
                                        emp.deptno=dept.deptno and dname like ?");
stmt.setString(1,"SALES");
ResultSet rs =stmt.executeQuery();
while(rs.next()){  System.out.println(rs.getString("Ename"));   }
stmt.setString(1,"ESTB");
rs =stmt.executeQuery();
while(rs.next()){  System.out.println(rs.getString("Ename")); }
```


Affichage des donnes d'une requête: JTable

- JTable est un composant Swing utilisé pour présenter et éditer des données sous forme d'une table.

  JTable(Object [ ][ ] rowData, Object [] ColumnNames);

  // rowData : valeurs des cellules; ColumnNames: étiquettes des colonnes
- Il est généralement conseiller d'implémenter un modèle de table (objet TableModel) au lieu de placer les données dans un tableau bidimensionnel. Ainsi, la construction d'un objet JTable peut faire appel aux classes et interfaces intermédiaires (définies dans javax.swing.table) suivants:

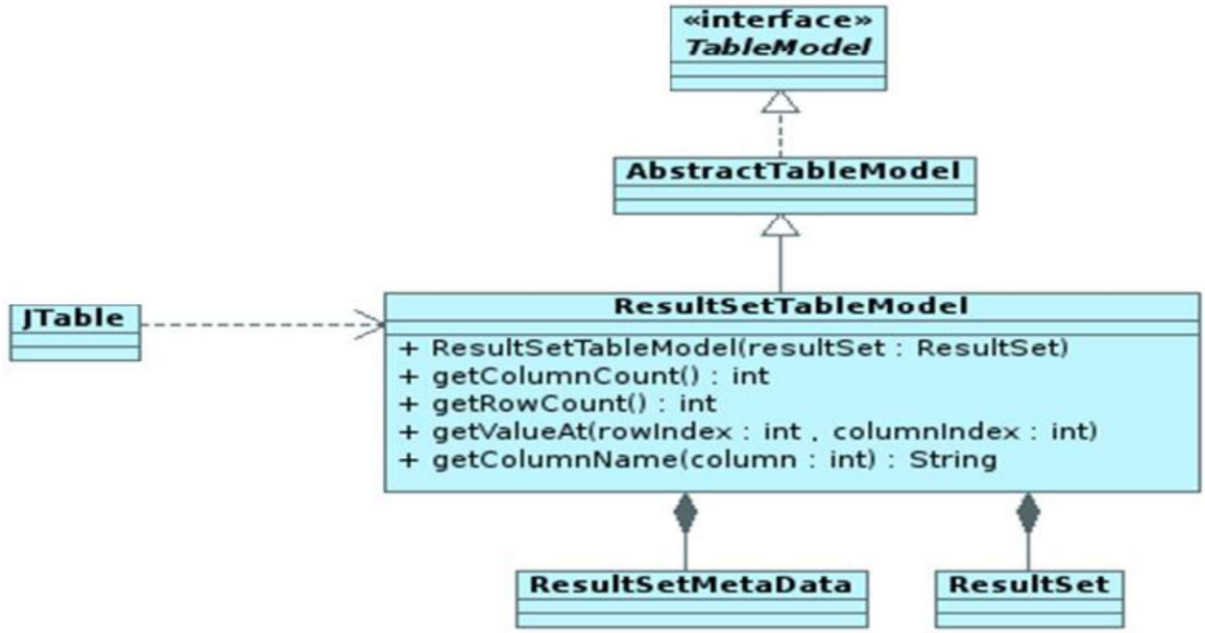
  JTable (TableModel dm, TableColumnModel cm);
  - TableModel : indique les données figurant dans la table
  - TableColumnModel : modèle pour les colonnes de la tables
  - JTableHeader : composant graphique gérant les titres des colonnes

Affichage des donnes d'une requête: JTable

En pratique, on implémente rarement TableModel d'une façon directe, mais on hérite plutôt de AbstractTableModel et on redéfinit, ainsi, les méthodes nécessaires :

- int getRowCount() : doit retourner le nombre de lignes du tableau
- int getColumnCount() : doit retourner le nombre de colonnes du tableau
- Object getValueAt(int rowIndex, int columnIndex) : doit retourner la valeur du tableau à la colonne et la ligne spécifiées
- String getColumnName(int columnIndex) : doit retourner l'en-tête de la colonne spécifiée

Affichage des donnes d'une requête: JTable



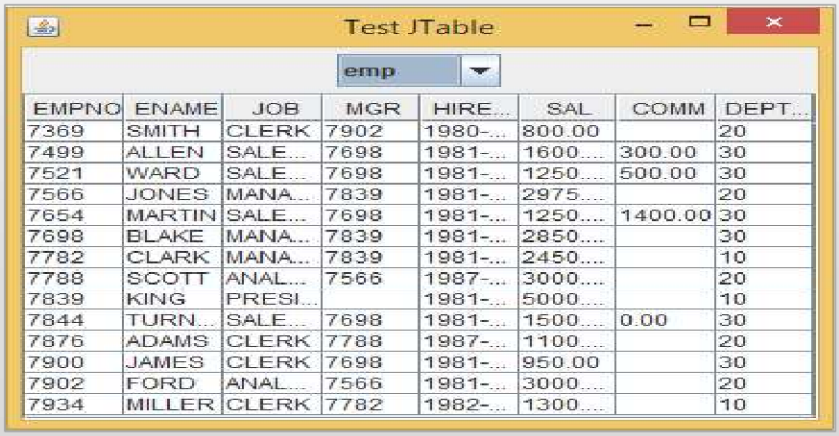
La classe *ResultSetTableModel* qui est chargée de fournir les données utiles au *JTable* en redéfinissant les méthodes necessaires

Affichage des donnes d'une requête: JTable

Soit une BD MYSQL de gestion des employés. le schéma de cette base est le suivant:

- dept (deptno, dname , loc)
- Emp(Empno, Ename, job, Mgr#, Hiredate, Sal ,Comm , Deptno#)
- Salgrade (Grade, Losal, Hisal)

Soit une application JAVA qui permet d'afficher l'ensemble des enregistrements d'une table



EMPNO	ENAME	JOB	MGR	HIRE...	SAL	COMM	DEPT...
7369	SMITH	CLERK	7902	1980-...	800.00		20
7499	ALLEN	SALE...	7698	1981-...	1600....	300.00	30
7521	WARD	SALE...	7698	1981-...	1250....	500.00	30
7566	JONES	MANA...	7839	1981-...	2975....		20
7654	MARTIN	SALE...	7698	1981-...	1250....	1400.00	30
7698	BLAKE	MANA...	7839	1981-...	2850....		30
7782	CLARK	MANA...	7839	1981-...	2450....		10
7788	SCOTT	ANAL...	7566	1987-...	3000....		20
7839	KING	PRESI...		1981-...	5000....		10
7844	TURN...	SALE...	7698	1981-...	1500....	0.00	30
7876	ADAMS	CLERK	7788	1987-...	1100....		20
7900	JAMES	CLERK	7698	1981-...	950.00		30
7902	FORD	ANAL...	7566	1981-...	3000....		20
7934	MILLER	CLERK	7782	1982-...	1300....		10

Informations sur une requête SQL

```
import java.awt.*; import javax.swing.*; import java.awt.event.*; import java.sql.*;
import javax.swing.table.*;
class ResultSetFrame extends JFrame {
    public  Connection getConnection() throws SQLException, ClassNotFoundException {
      String drivers = "com.mysql.jdbc.Driver"; String url = "jdbc:mysql:///test";
      Class.forName (drivers); return DriverManager.getConnection(url); }
    public ResultSetFrame() {
      setTitle("Test JTable"); setSize(400, 300); tableNames = new JComboBox();
      JPanel p = new JPanel(); p.add(tableNames);add(p, BorderLayout.NORTH);
      try {
            conn = getConnection(); DatabaseMetaData meta = conn.getMetaData();
            stat = conn.createStatement();
            ResultSet tables = meta.getTables(null, null, null,new String[ ] { "TABLE" });
            while (tables.next())  tableNames.addItem(tables.getString(3)); tables.close();
      }
      catch (ClassNotFoundException e){ e.printStackTrace(); }
      catch (SQLException e){e.printStackTrace();}
      …
    }
    private JScrollPane scrollPane; private ResultSetTableModel model;
    private JComboBox tableNames; private Connection conn;private ResultSet rs;
    private Statement stat;
    }
```

Informations sur une requête SQL

```
class ResultSetFrame extends JFrame {
  public ResultSetFrame() {
    ...
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent event) {
        try{ if (conn != null) conn.close();} catch (SQLException e){e.printStackTrace();}
      } });
    tableNames.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent event){
        try{
          if (scrollPane != null) remove(scrollPane);
          String tableName = (String)tableNames.getSelectedItem();
          if (rs != null) rs.close();
          String query = "SELECT * FROM " + tableName; rs = stat.executeQuery(query);
          model = new ResultSetTableModel(rs); JTable table = new JTable(model);
          scrollPane = new JScrollPane(table); add(scrollPane, BorderLayout.CENTER);
          validate();
          }
        catch (SQLException e){ e.printStackTrace(); } } });
  }
   private JScrollPane scrollPane; private ResultSetTableModel model;
   private JComboBox tableNames; private Connection conn;private ResultSet rs;
   private Statement stat;
} }
```


Informations sur une requête SQL

```
class ResultSetTableModel extends AbstractTableModel {
  public ResultSetTableModel(ResultSet aResultSet){
    rs = aResultSet;
    try { rsmd = rs.getMetaData();} catch (SQLException e){e.printStackTrace();}
  }
  public String getColumnName(int c){
    try{return rsmd.getColumnName(c + 1);}
    catch (SQLException e){e.printStackTrace();return "";} }
  public int getColumnCount(){
    try{ return rsmd.getColumnCount();}
    catch (SQLException e){ e.printStackTrace();return 0;} }
  public Object getValueAt(int r, int c){
    try { rs.absolute(r + 1);return rs.getObject(c + 1); }
    catch(SQLException e){ e.printStackTrace(); return null;} }
  public int getRowCount(){
    try { rs.last();  return rs.getRow(); }
    catch(SQLException e){e.printStackTrace();return 0;}  }
  private ResultSet rs;   private ResultSetMetaData rsmd;
}
```

Mapping objet relationnel

- En pratique, on cherche toujours séparer la logique de métier de la logique de présentation. Ainsi, une application est généralement devisée en 3 couches:
  - La couche d'accès au données DAO: partie de l'application qui permet d'accéder aux données qui sont souvent stockées dans une base de données relationnelles.
  - La couche métier : regroupe l'ensemble des traitements que l'application doit effectuée.
  - La couche présentation : s'occupe de la saisie et l'affichage des données.
- Le mapping Objet relationnel consiste faire correspondre un enregistrement d'une table de la BD un objet d'une classe correspondante (classe appelé persistante)

Affichage des donnes d'une requête: JTable

Soit une BD MYSQL de gestion des employés. le schéma de cette base est le suivant:

- dept (deptno, dname , loc)
- Emp(Empno, Ename, job, Mgr#, Hiredate, Sal ,Comm , Deptno#)
- Salgrade (Grade, Losal, Hisal)

Soit une application JAVA qui permet d'afficher les employés d'un département donnée (selection depuis un comboBox) proposer une structure en 3 couches (DAO, Metier, Présentation)