

INTERFACE GRAPHIQUE AVEC SWING



Objectif du cours



A la fin de cours vous serez capable de:

- Construire vos propre GUIs
- Utiliser les composants Java: JTextField, Buttons...
- Gérer les évènements
- Dessiner vos objets customisés

Plan du cours

- Présentation, Portabilité,
- AWT/SWING
- Conteneurs
- Composants communs
- Layout Manager
- Evènements
- Menus
- Listing
- Painting



Présentation

L'interface est peut-être la partie la plus importante d'une application elle est aussi celle que l'on remarque le plus.

Pour les utilisateurs, elle représente l'application et certains ne doivent même pas savoir qu'un programme est exécuté en arrière-plan.

La création d'une interface utilisateur doit être considérée comme un processus itératif; il est rare de parvenir à la perfection à la première tentative.

Ce support présente les composants nécessaires à la conception d'une application Swing, ainsi que le processus de création d'une interface.

Portabilité

La particularité du langage Java est de permettre l'écriture de programmes portables avec des interfaces-utilisateur graphiques mélangeant dans un même fichier code source et design.

Dans la plupart des langages de telles interfaces sont réalisées au moyen de bibliothèques séparées du langage et dépendantes de la couche graphique appartenant à l'O.S installé.

Interface graphique avec swing

AWT/SWING

Java Foundation Class

- **AWT** Components
- **Swing** supporté par Oracle (autrefois Sun)
- Swing + AWT constituent la bibliothèque JFC (pour Java Foundation Classes)

Swing repose sur AWT mais Swing != AWT !

- **JButton != Button !**

Les concepteurs de Swing ont repris les mêmes noms que AWT, avec un J en début du nom pour les différencier: JComponent, JFrame, JDialog, JButton, Jpanel.

Bibliothèque AWT

- **AWT**, acronyme de Abstract Window Toolkit première bibliothèque pour réaliser des interfaces utilisateur graphiques (G.U.I)
- Les classes de AWT constituent le paquet **java.awt**
- Dépendante du système d'exploitation, donc pas de portabilité

Java utilise le système d'exploitation sous-jacent pour afficher les composants graphiques :

Conséquence: affichage de l'interface utilisateur d'une application peut diverger sensiblement.

AWT garantira que la fonctionnalité recherchée sera dans tous les cas fournie mais elle sera présentée différemment.

Or Java se veut être 100% indépendant de la plate-forme utilisée. Pour cette raison, une nouvelle API a été définie →

A.P.I SWING

- Entièrement écrite en Java, donc portable.
- **Swing** bibliothèque plus puissante, en complément d'A.W.T . Swing est venu apres AWT
- pour les composants graphiques.
- certains concepts A.W.T présents dans Swing.
- Swing A.P.I immense → 18 packages publiques
 - le plus utilisé : **javax.swing**
- distinction entre objets et modèles (design pattern)
 - si état de l'objet change tous les représentations graphiques de l'objet seront impactées

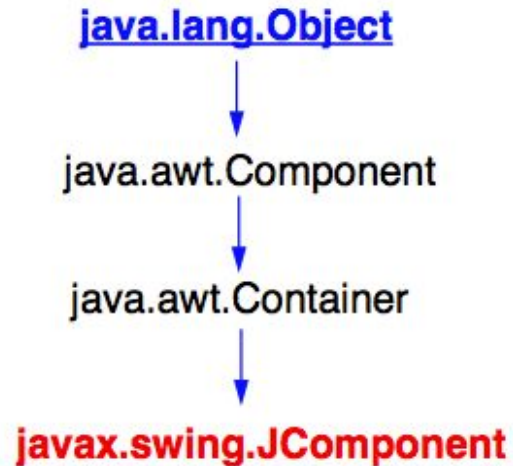
Objets Swings

Parmi les objets, on distingue :

- Les classes conteneurs
- Les classes composants
- Les classes de liaisons :
 événements, gestionnaire de disposition, ...

Ces objets sont organisés en une arborescence de classe

Sommet de la Hiérarchie



La classe **java.awt.Component** est la superclasse qui représente les objets graphiques, aussi bien AWT que Swing.

Ergonomie

Dans ce support le code est écrit au même endroit pour des raisons de présentation
→ programmation non modulaire

```
public class MainClass { // Un seul Fichier Mainclass.java
```

```
    public static void main(String[] args) {
```

```
        MyFen fen = new MyFen("titre");
```

```
        fen.setSize(50, 50);
```

```
        fen.setVisible(true);
```

```
    }
```

```
MyFen extends JFrame{
```

```
    private String titreFen;
```

```
    MyFen(String title ){
```

```
        this.titreFen = title;
```

```
        super(titreFen);
```

```
    }
```

```
}
```

Ergonomie prog modulaire

programmation modulaire avec une classe par fichier

```

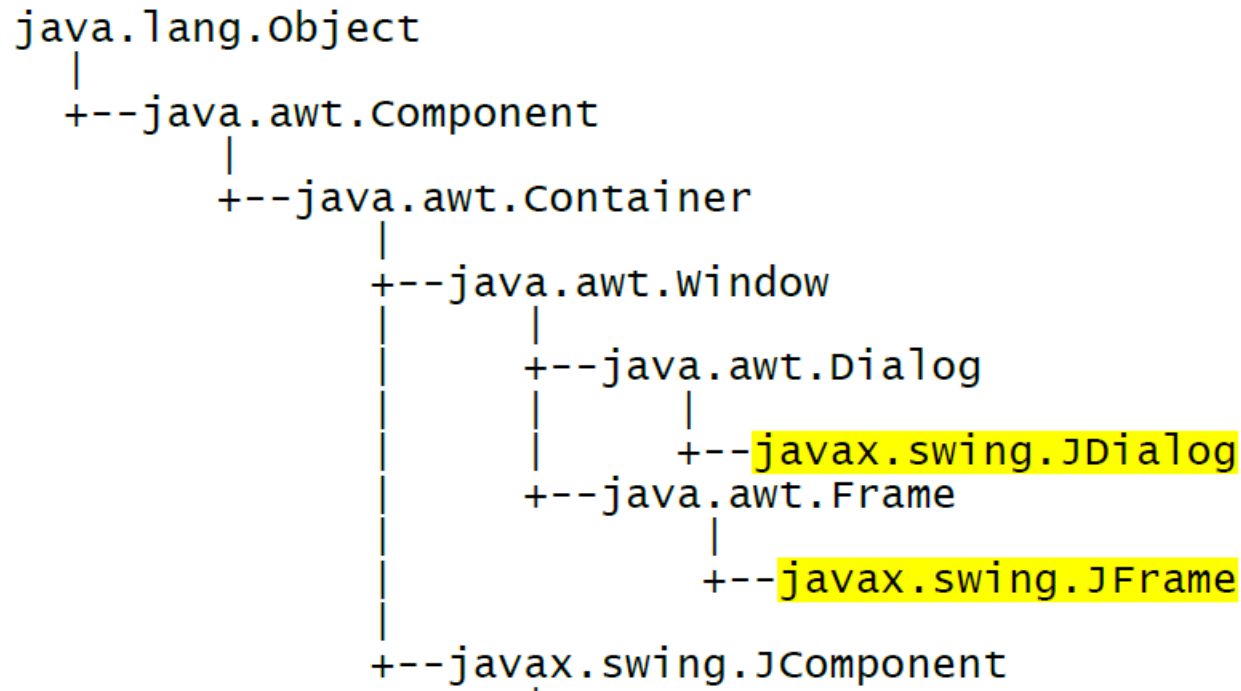
public class MainClass {    //Fichier Mainclass.java
    protected static void createAndShowGUI() {
        MyFen fen = new MyFen("titre");
        fen.setSize(50, 50);
        fen.setVisible(true);
    }
    public static void main(String[] args) {
        //thread création objet graphique et rend la main au thread principal ;
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() { createAndShowGUI() ; }
        });
    }
}

public class MyFen extends JFrame{    //Fichier MyFen.java
    private String titreFen;
    MyFen(String title ){
        super(title);
        this.titreFen = title;
    }
}
    
```

Interface graphique

CONTENEURS(Container)

Hiérarchie des conteneurs Swing



Conteneurs

La classe `java.awt.Container` est une sous-classe de `java.awt.Component`

→ un conteneur peut être vu comme un composant lorsqu'il est inclus dans un autre conteneur.

mais super classe de `javax.swing.JComponent`

→ Un conteneur contient des composants graphiques

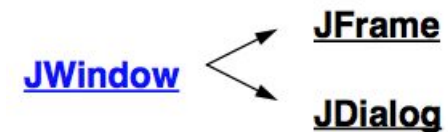
- **JWindow**: fenêtre de base sans titre
ni menu système ni bordure et non déplaçable
- **JFrame**: fenêtre principale de l'application...
- **JPanel** : regroupement de composant..
- **JDialog** : fenêtre d'interaction avec l'utilisateur

javax.swing.JFrame

Classe associée à la fenêtre principale de l'application.
Toutes les application I.H.M héritent de cette classe.

Elle fournit de centaines de méthodes pour la création d'une interface graphique contenant :

- Des composants(boutons, menus, titres, onglets)
- Une taille
- Une barre de titre
- Une opération de fermeture ...



JFrame: Les méthodes

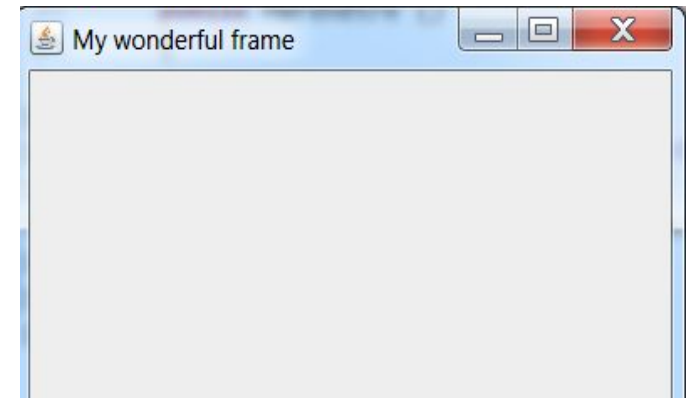
- `setSize(int width, int height)` : Fixe la taille de la fenêtre.
- `setTitle(String title)` : Définit le libellé de la barre de titre
- `setVisible(boolean visible)` : Définit si la fenêtre s'affiche .
- `setDefaultCloseOperation(int op)` :
`setDefaultCloseOperation(EXIT_ON_CLOSE);`
- `setResizable(boolean resizable)` :
 - Autorise ou non à retailer la fenêtre

JFrame: exemple

```
import javax.swing.* ;

class MaFenetre extends JFrame
{
    public MaFenetre () {
        this.setSize(400, 400);
        this.setTitle("My wonderful frame");
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(true);
    }
}

public static void main(String[] args)
{
    MaFenetre uneFenetre = new MaFenetre();
}
```



visibilité et fermeture de la fenêtre

La création d'une instance de la classe « MaFenetre » stocke en mémoire un objet fenetre.

Cette fenêtre n'est pas visible.

La méthode setVisible(true) la rendra visible.

EXIT_ON_CLOSE :

Par défaut, une demande de fermeture de la fenêtre (clique sur la case de fermeture) rend la fenêtre invisible.

plus aucun élément ne sera visible mais l'application continue à tourner et à consommer des ressources...

Solution:

```
myJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

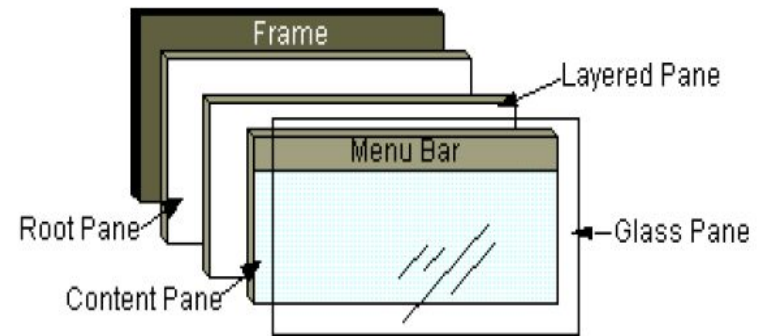
javax.swing.JPanel

Un panneau est un conteneur (ou containers), en général transparent (possibilité de changer sa couleur de fond), avec pour but de

- regrouper des contrôles pour mieux les positionner dans une fenêtre
- regrouper visuellement des contrôles
- ajouter des barres de défilement aux composants "scrollables"

La fenêtre principale est elle-même un conteneur.

Elle est composée de plusieurs panneaux superposés.



- un panneau de classe **JLayeredPane** recouvrant toute la surface de la fenêtre. Sur ce panneau sont posés 2 autres panneaux: la barre de menu (JMenuBar) et la zone client (Content Pane) sur laquelle sont posés les contrôles.

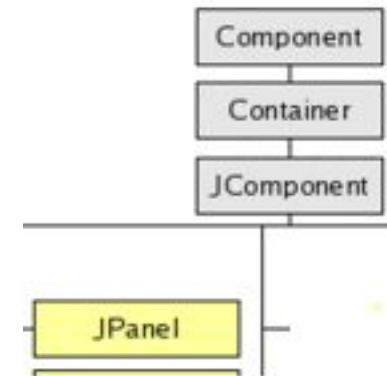
Utilisation du JPanel

Un panel est le conteneur de tous les éléments d'une fenêtre.

Un panel est ajouté à la fenêtre.

L'ordre n'est pas important pour:

- le définir un panel,
- l'ajouter au cadre
- lui affecter un gestionnaire de disposition (plus loin dans le cours)
- lui ajouter des contrôles graphiques (component) → méthode add()
- fixer les dimensions et rendre visible le cadre.
 - `frame.getContentPane().add(myButton);`



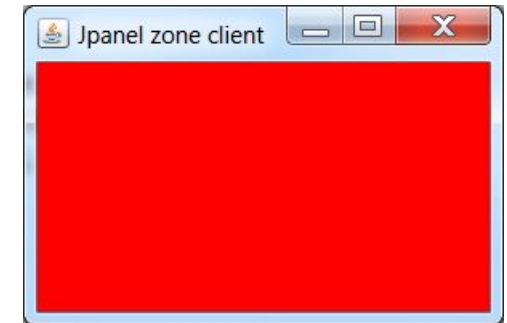
JPanel

La zone client est occupée par un JPanel (zone rectangulaire dans laquelle il est possible de dessiner, posé des composants)

La méthode **getContentPane()** fournit une référence sur la zone client. C'est un objet instance de la classe JPanel.

Exemple : couleur de fond rouge, occupant toute la partie utile (zone client) de la fenêtre principale.

```
public class MainCoursJPanel {  
    public static void main(String[] args)  
    {  
        JFrame fen = new JFrame("Jpanel zone client");  
        fen.setSize(300,200);  
        fen.setVisible(true);  
        fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        JPanel zoneClient = (JPanel)fen.getContentPane();  
        zoneClient.setBackground(Color.red);  
    }  
}
```



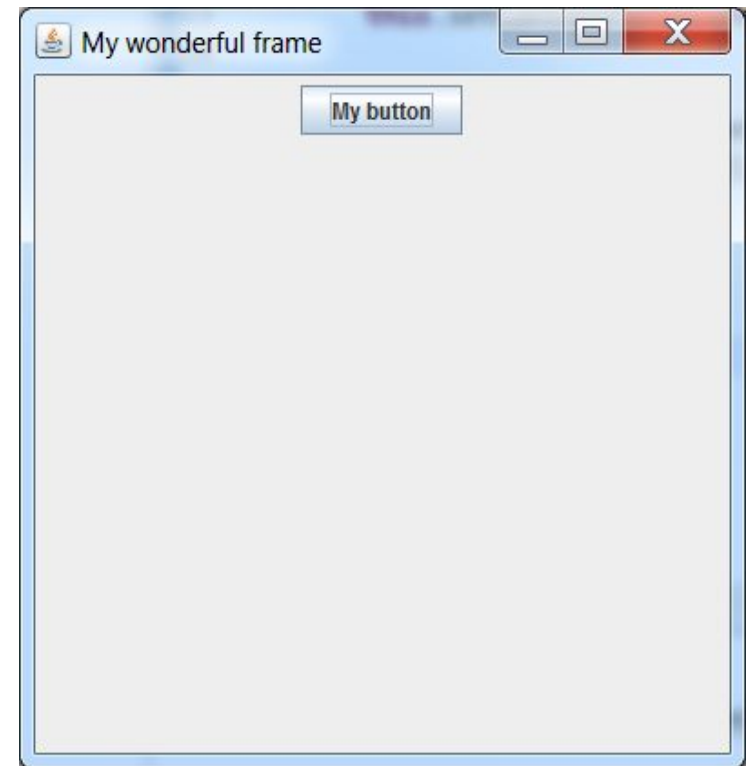
Bien qu'il soit possible de récupérer la zone cliente (sous forme de panneau) d'un jframe , il est préférable de créer son JPanel qui sera ajouté au cadre (JFrame) → (voire diapo suivante)

ajouter des contrôles

étapes:

- 1) ajouter des contrôles aux panneaux
- 2) ajouter les panneaux à la zone client.

```
public class MyFenetre extends JFrame{  
  
    public MyFenetre () {  
        .....  
        JPanel Drawpanel = new JPanel();  
        //etape 1  
        DrawPanel.add(new JButton("My button"));  
        //etape 2  
  
        this.add(DrawPanel);  
  
        this.setVisible(true);  
    }  
}  
  
public class MainClass {  
    public static void main(String[] args) {  
        JFrame cadre = new MyFenetre("My wonderful frame");  
    }  
}
```



Conteneur:dialog

javax.swing.JOptionPane

Une fenêtre Dialog est un conteneur

- non contenu dans un autre conteneur
- dépend d'un autre cadre (JFrame, JWindow)

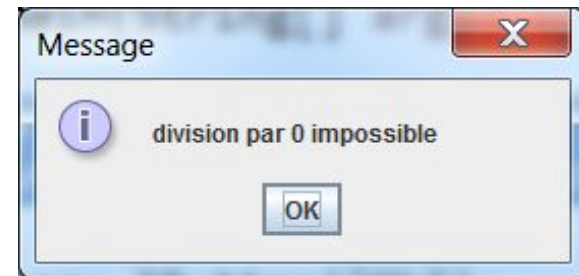
Fonction : afficher un message d'erreur ou warning à l'utilisateur.(modale)

La classe javax.swing.**JOptionPane**

JOptionPane est un simple container qui crée automatiquement une JDialog and add itself to the JDialog's content pane.

Code minimal → dialogue standard

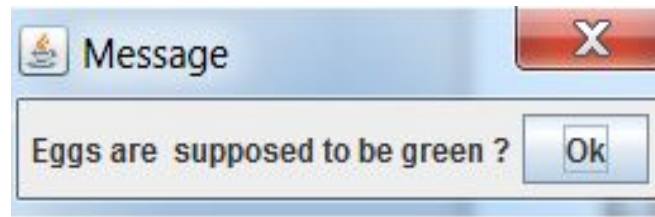
```
JWindow fen = new JWindow();  
JOptionPane.showMessageDialog(fen, "la division par 0 impossible");
```



Javax.swing.JDialog

Elle peut contenir des contrôles tous dérivés directement ou indirectement de **JComponent** (liste, bouton, label, text...).

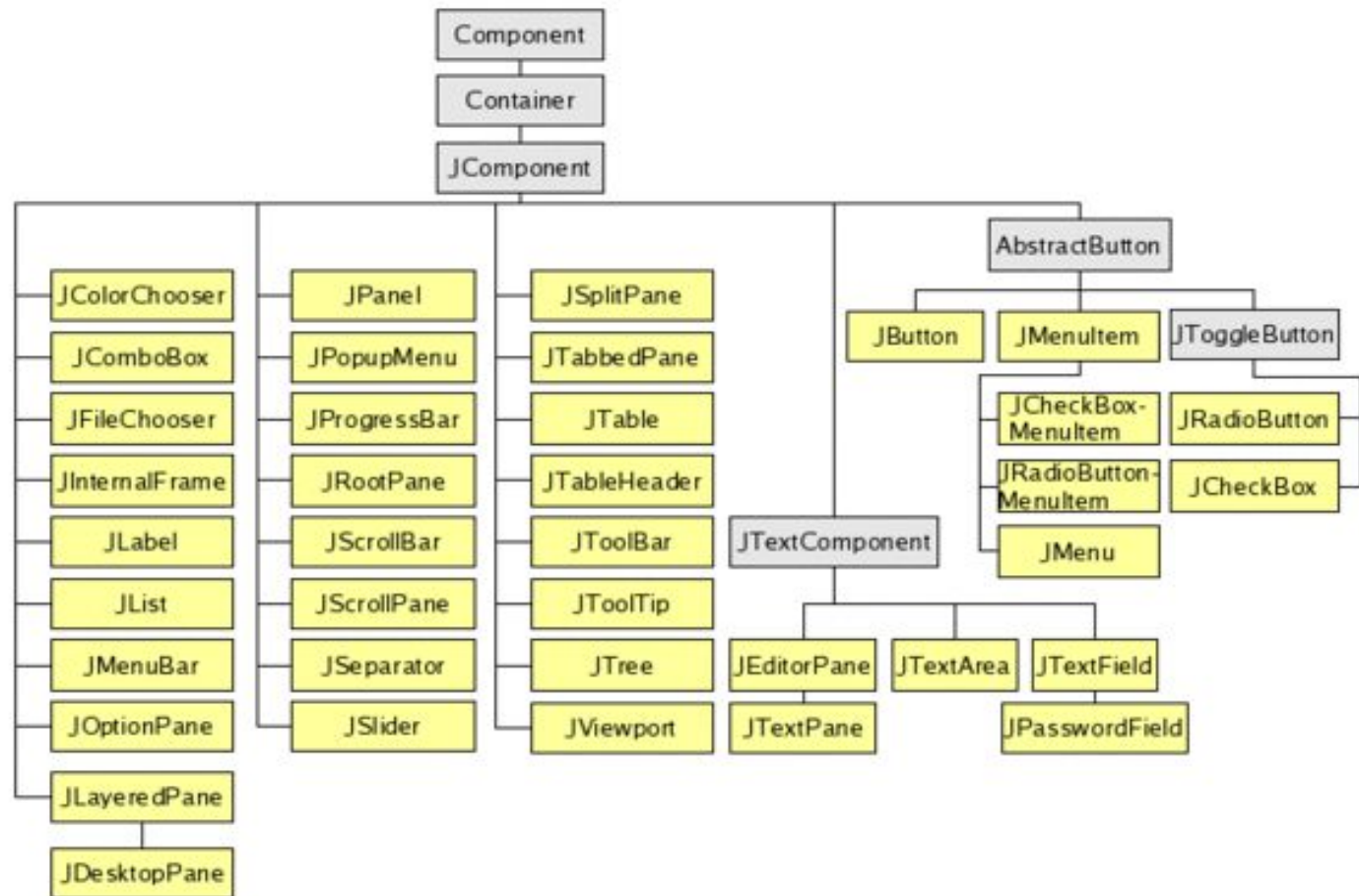
```
JButton btnOk = new JButton("Ok");
JLabel myLabel = new JLabel("Eggs are supposed to be green ?");
JFrame frame = new JFrame("DialogDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JDialog dialog = new JDialog(frame, "Message", true);
JPanel contentPane = new JPanel();
contentPane.add(myLabel);
contentPane.add(btnOk);
dialog.setContentPane(contentPane);
dialog.pack();
dialog.setVisible(true);
//pas de exit on close pour une JDialog
System.exit(0);
```



LES COMPOSANTS COMMUNS

JButton, JLabel....

Hiérarchie des composants



SWING utilise intensément la programmation orientée objet.

7 niveaux de dérivation entre la classe Object et la classe JRadioButton

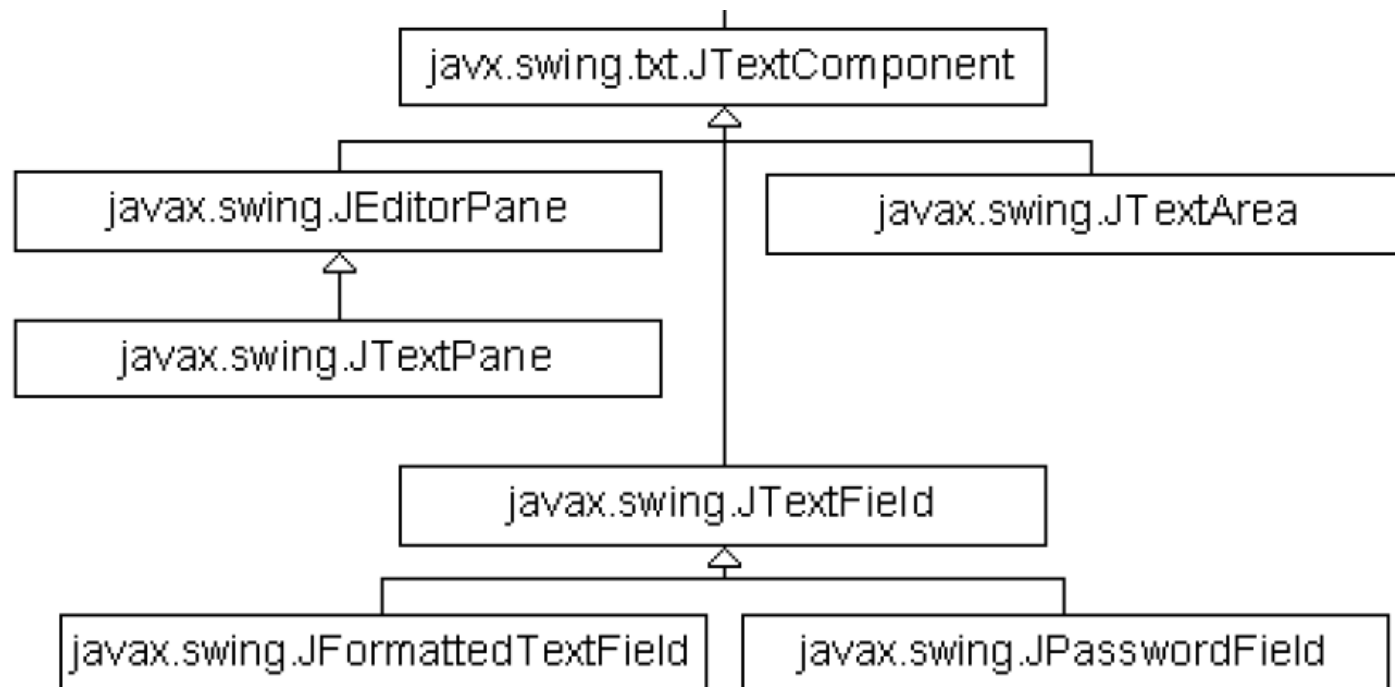
Object → Component → Container → JComponent → AbstractButton → JToggleButton → JRadioButton

JLabel

- Un label appartient a la classe JLabel.
- Un label sert à afficher un texte simple, une image ou les 2 :
 - Pour un texte : **setText**(String Text)
 - Pour une image : **setIcon**(Icon icon)
- Plusieurs constructeurs disponibles:
 - JLabel()
 - JLabel(String text)
 - JLabel(Icon icon)

Texte

Swing possède plusieurs composants pour permettre la saisie de texte.



Texte

City:

JLabel JTextField

Enter the password:

JPasswordField

This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

JTextArea :
texte simple multilignes

Ascenseur :
cf. **JScrollPane**



JEditorPane : texte avec styles compatible HTML et RTF

Composants communs : texte

JTextField



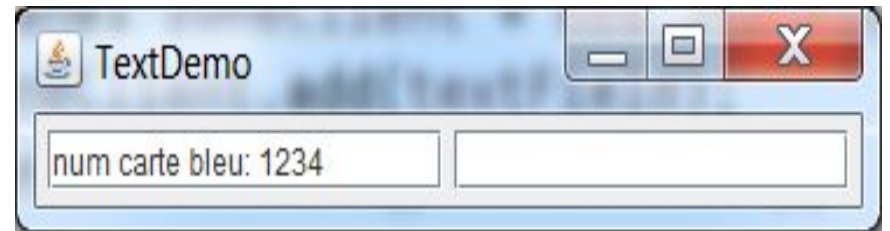
- La classe **JTextField** sert à définir les champs
- Un champ de texte sert à saisir des données
- Plusieurs constructeurs disponibles:
 - **JTextField**(int columns)
 - **JTextField**(String defaultText)
 - **JTextField**(String defaultText, int columns)
- Méthodes utiles :
 - **void** setText(String text) Set the text in the text field.
 - String getText() Get the text typed in the text field.

JTextArea

- Un aire de texte avec un nombre de lignes et colonnes.
- Plusieurs constructeurs disponibles:
 - **JTextArea** (int rows, int columns)
 - **JTextArea** (String defaultText)
 - **JTextArea** (String defaultText, int rows, int columns)
- Méthodes :
 - String getText() / **void** setText(String text)
 - Get ou set le text dans un aire de texte .
 - **int** getRows() / **void** setRows(int nbOfRows)
 - **int** getColumns() / **void** setColumns(int nbOfCol)

Exemple

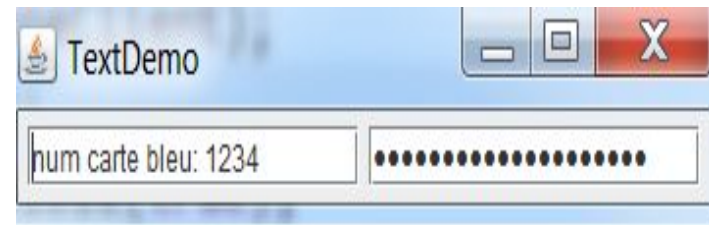
```
JFrame frame = new JFrame("TextDemo");  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
JTextField textField = new JTextField(15);  
textField.setText("num carte bleu: 1234");  
JPasswordField passwordText = new JPasswordField(15);  
JPanel zoneClient = new JPanel();  
zoneClient.add(textField);  
zoneClient.add(passwordText);  
frame.add(zoneClient);
```



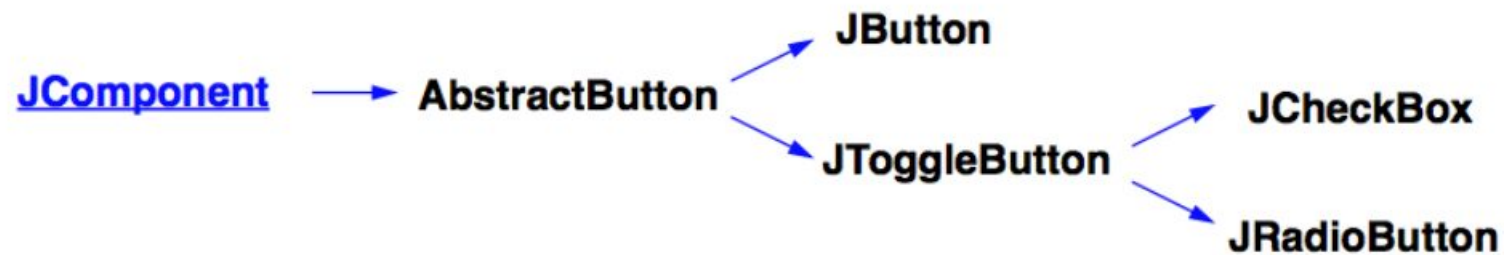
//Taille de la Window s'adapte au contenu (size and layouts)

```
frame.pack();  
frame.setVisible(true);
```

```
String text = textField.getText();  
passwordText.setText(text);
```



La classe AbstractBouton



JButton




JCheckbox :
choix indépendants



JRadioButton :
choix exclusif : cf. **ButtonGroup**

JButton

- Un bouton classique : 
- Il faut définir une action sur un click bouton avec la méthode ***addActionListener(ActionListener event)*** (plus loin dans le cours)
- Plusieurs constructeurs disponibles :
 - **JButton()**
 - **JButton(String text)**
 - **JButton(Icon image)**

JCheckBox

- Représente une case à cocher.



- Constructeurs :

- *JCheckBox(String text)*
- *JCheckBox(String text, boolean selected)*

- Méthodes Utiles:

- *boolean isSelected()* :

Retourne true si la case est cochée.

- *void setSelected(boolean selected)* :

Set if the check box must be selected.

- *String getText()* / *void setText(String label)* :

Get or set the label for the check box.



JRadioButton

Un bouton radio.



■ constructeurs :

■ *JRadioButton(String text)*

■ *JRadioButton(String text, boolean selected)*

■ méthodes :

■ *boolean isSelected() / void setSelected(boolean)*

ButtonGroup

Un bouton radio n'est jamais seul. Une application dispose toujours d'un groupe de N boutons radio permettant de choisir 1 option parmi N (un seul peut être sélectionner à la fois).

Les objets boutons radio (JRadioButton) sont toujours ajoutés à un groupe de boutons (ButtonGroup) pour assurer la dynamique de sélection. Les boutons sont visuellement regroupés, c'est à dire qu'ils sont souvent placés dans un panneau avec titre et bordure.

Méthodes utiles :

Ajouter ou supprimer un bouton dans un groupe de bouton:

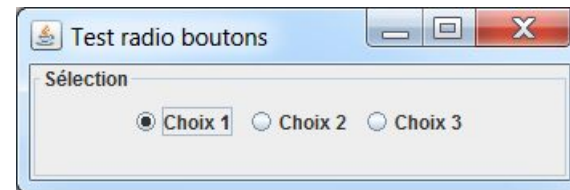
- *void add(AbstractButton btn)*
- *void remove(AbstractButton btn)*

Composants communs : button

ButtonGroup:exemple

```
JButtonGroup group = new ButtonGroup();
JRadioButton radio1 = new JRadioButton("Choix 1", true);
JRadioButton radio2 = new JRadioButton("Choix 2");
JRadioButton radio3 = new JRadioButton("Choix 3");
group.add(radio1);
group.add(radio2);
group.add(radio3);
JPanel panel = new JPanel();
panel.setBorder(BorderFactory.createTitledBorder("Sélection"));
panel.add(radio1);
panel.add(radio2);
panel.add(radio3);
```

```
JFrame f = new JFrame();
f.getContentPane().add(panel);
f.pack();
f.setVisible(true);
```





JProgressBar

- Une barre de progression.



- Constructeur :

JProgressBar(int min_value, int max_value).

- Méthodes :

- *int getValue() / void setValue() :*

Get ou set la valeur de progression .

- *void setString(String text) :*

Texte écrit sur la barre.

- *void setStringPainted(boolean painted) :*

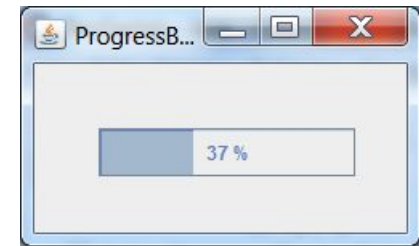
- Indique si le texte est écrit sur la barre.

- **false** par défaut.

ProgressBar Demo

```
public class ProgressBarDemo extends JPanel
{
    JProgressBar jb;
    CoursProgressBarDemo()
    {
        jb = new JProgressBar(0,100);
        jb.setBounds(40,40,160,30);
        jb.setValue(0);
        jb.setStringPainted(true);
        this.add(jb);
    }
    public void iterate()
    {
        Random random = new Random(); int i= 0;
        while(i <= 100){
            jb.setValue(i); i++;
            try { Thread.sleep(random.nextInt(200)); }
            catch (InterruptedException e) {}
        }
    }
}

public static void main(String[] args) {
    JFrame frame = new JFrame("ProgressBarDemo");
    ProgressBarDemo MonContentPane = new ProgressBarDemo();
    MonContentPane.setOpaque(true);
    frame.setContentPane(MonContentPane);
    MonContentPane.iterate();
}
```



ActionCommand

- Disponible sur de nombreux composant.
- Permet d'identifier des composants.
- Set: *void setActionCommand(String command).*
- Get: *String getActionCommand().*

coursActionCommand ▶ bin ▶ coursActionCommand ▶ images



```
public ActionCommand() {  
    JRadioButton birdButton = new JRadioButton("Bird");  
    birdButton.setActionCommand("Bird");  
    URL location = ActionCommand.class.getResource(  
        "images/" + birdBtn.getActionCommand() + ".gif");  
    Icon img = new ImageIcon(location);  
    JLabel LblPicture = new JLabel(img);  
    this.add(LblPicture);  
    LblPicture.setIcon(new ImageIcon(location));  
}
```

ActionCommand &(ActionEvent)

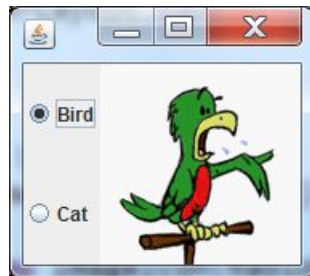
Dans l'exemple précédent peu d'intérêt de coder :

```
"images/" + birdBtn.getActionCommand() + ".gif";
```

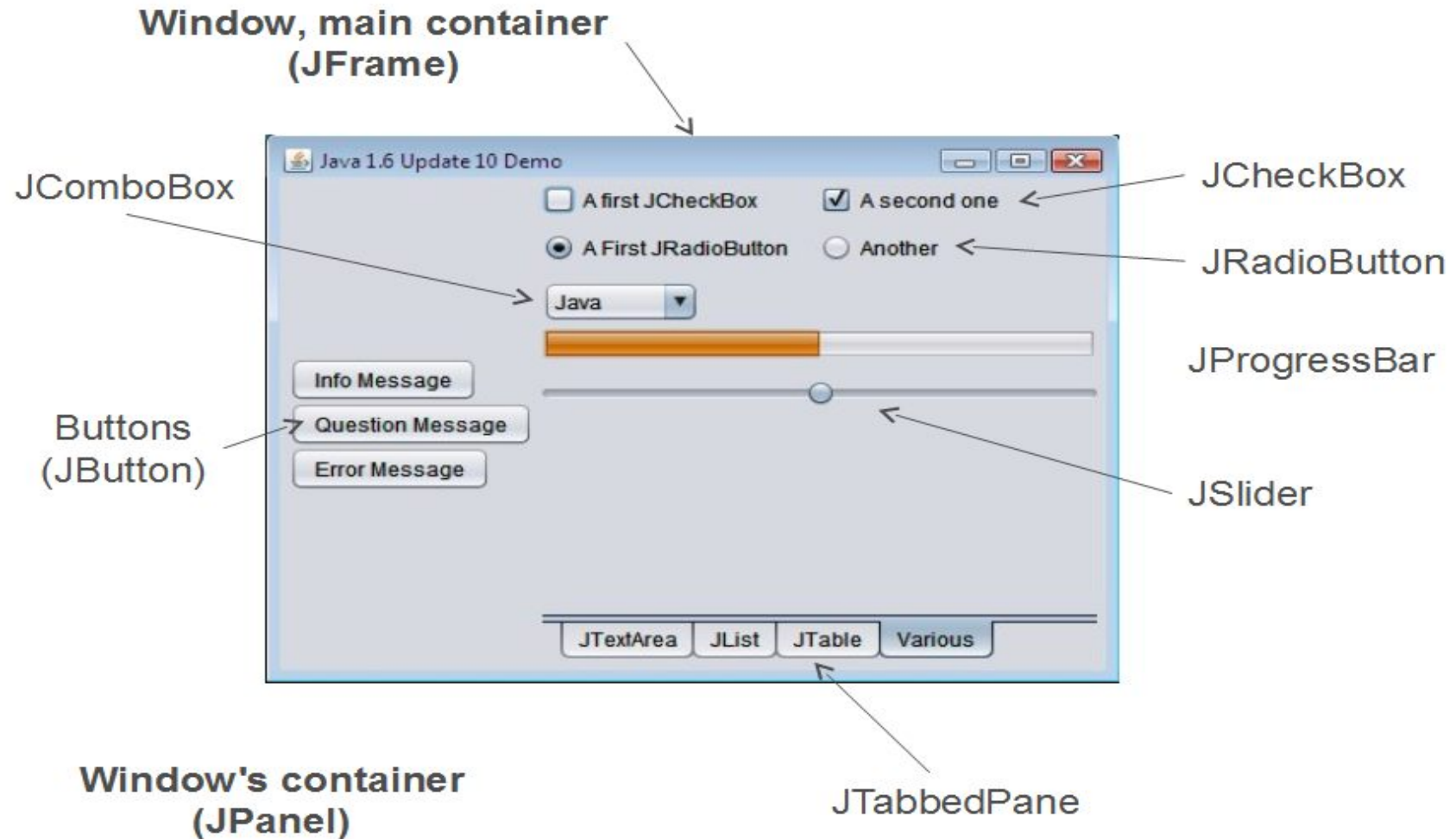
Avec une seule image :

```
"images/Bird.gif"
```

Mais l'idée est de charger l'image qui correspond au bouton cliqué nous verrons comment dans la section evenement.



JFrame: fenêtre complète fonctionnelle



Interface graphique

LAYOUT MANAGER

Agencement des contrôles

Presentation

Java propose une mise en forme automatique (utilisé jusqu'à présent) mais il possible d'utiliser un gestionnaire de mise en forme personnalisable: **Layout manager**

- Associé aux conteneurs, assure une stratégie de positionnement prenant en compte la taille et la position des composants qu'il lui sont confiées pour tenter de fixer la taille réelle (size) du composant à une valeur la plus proche possible de sa taille idéale (preferredSize)

- Plusieurs sont disponibles :

-BorderLayout, FlowLayout, GridLayout, BoxLayout, GridBagLayout.

A vous de choisir le plus adapté à la situation

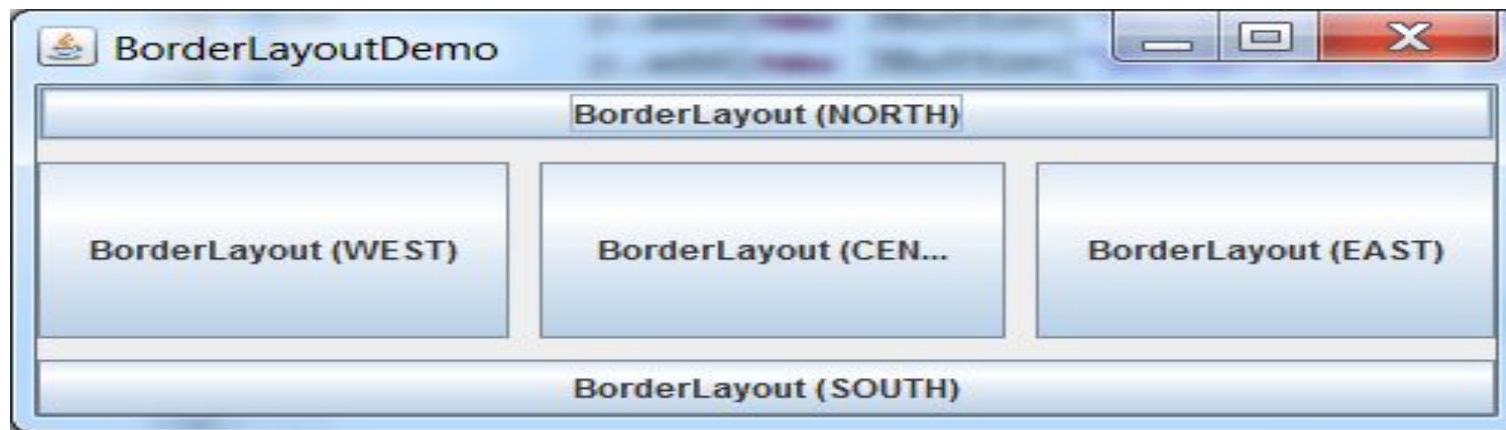
- Associer conteneur & layout manager

avec la Méthode *setLayout(LayoutManager layout)*

`objetContainer.setLayout (new xxxLayout(. . .));`

BorderLayout

- Chaque composant est placée dans un conteneur (Container) séparé en 5 régions.
- La région par défaut est : BorderLayout.Center



- La classe BorderLayout a deux constructeurs:
 - BorderLayout()
 - BorderLayout(int hspace, vspace): espace entre les composants.

BorderLayout: Example

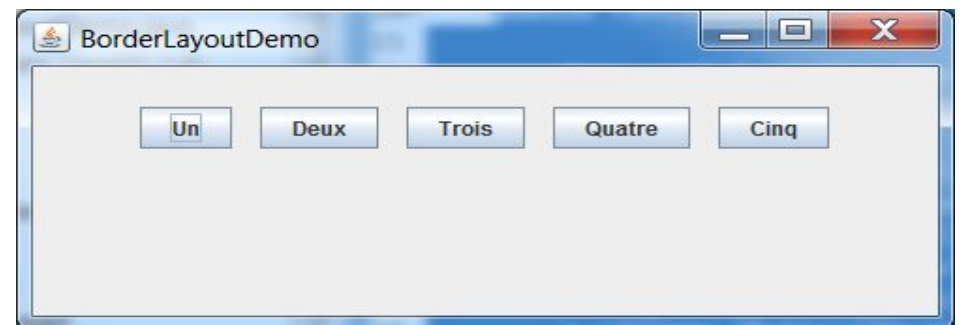
```
public static void main(String[] args) {  
  
    JFrame cadre = new JFrame("BorderLayoutDemo");  
    JPanel p = new JPanel();  
    p.setLayout(new BorderLayout(10,10));  
    p.add(new JButton("BorderLayout(CENTER)"), BorderLayout.CENTER);  
    p.add(new JButton("BorderLayout (NORTH)"), BorderLayout.NORTH);  
    p.add(new JButton("BorderLayout (SOUTH)"), BorderLayout.SOUTH);  
    p.add(new JButton("BorderLayout (WEST)"), BorderLayout.WEST);  
    p.add(new JButton("BorderLayout (EAST)"), BorderLayout.EAST);  
    cadre.getContentPane().add(p);  
    cadre.setSize(500, 200);  
    cadre.setVisible(true);  
}
```

FlowLayout

- Le layout par défaut de JPanel.
- Un gestionnaire FlowLayout place les composants de la gauche vers la droite :
 - Sur la même ligne.
 - Crée une "nouvelle ligne" si nécessaire.
 - Chaque ligne est centrée.

FlowLayout: Exemple

```
public FenLayoutManager(String titre) {  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    this.setSize(500, 200);  
    this.setTitle(titre);  
    JPanel p = new JPanel();  
    p.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 25 ));  
    p.add(new JButton("Un"), BorderLayout.CENTER);  
    p.add(new JButton("Deux"), BorderLayout.NORTH);  
    p.add(new JButton("Trois"), BorderLayout.SOUTH);  
    p.add(new JButton("Quatre"), BorderLayout.WEST);  
    p.add(new JButton("Cinq"), BorderLayout.EAST);  
    this.getContentPane().add(p);  
    this.setVisible(true);  
}
```



GridLayout

- Un gestionnaire GridLayout organise les composants selon une grille rectangulaire ayant un nombre de lignes et de colonnes convenus lors de la création du gestionnaire.
- Toutes les cases de cette grille ont les mêmes dimensions.
- Les composants occupent tout l'espace disponible dans la case.
- La distance par défaut inter-composant est modifiable (Hgap et Vgap) ...

GridLayout: Exemple

```
JFrame fen = new JFrame("LayoutManager-Grid");
fen.setSize(300,200);
fen.setVisible(true);
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel p = new JPanel();
p.setLayout(new GridLayout(4,2,5,10 ));
p.add(new JButton("Un"));
p.add(new JButton("Deux"));
p.add(new JButton("Trois"));
p.add(new JButton("Quatre"));
p.add(new JButton("Cinq"));
p.add(new JButton("Six"));
p.add(new JButton("Sept"));
fen.getContentPane().add(p);
```



GridBagLayout

L'un des gestionnaire le plus flexible mais bien plus complexe que les précédents.

Place les composants dans un grid de lignes et colonnes comme un simple GridLayout mais :

- permet que l'espace occupé par chaque composant soit une région rectangulaire formée par la fusion d'un certain nombre de cases de ce quadrillage.
- Lignes et colonnes n'ont pas nécessairement les mêmes dimensions.



GridBagLayout

- Lors de son ajout au conteneur, chaque composant est associé à une contrainte spécifiant:
 - les cases du quadrillage occupé par le composant
 - comment il les occupe.
 - Les contraintes instances de GridBagConstraints:
 - **int gridx, gridy** : position du composant dans le grid.
 - **int gridwidth, gridheight** : nombre de cellules occupées par le composant.
 - **int fill** : Indique comment le composant remplit sa zone si la taille de la zone est plus grande que la taille du composant.
 - **int anchor** : position du composant dans sa zone, s'il ne la remplit pas entièrement.
- Les valeurs possibles sont : GridBagConstraints.NORTH, GridBagConstraints.EAST
GridBagConstraints.NORTHEA, GridBagConstraints.SOUTHEAST

GridBagLayout

```
// dans le constructeur de la classe extendant un JPanel
this.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0; c.gridy = 0;
c.weightx = 0.5; //gere espace horizontale entre les btns
this.add(new JButton("Un"),c);
c.gridx= 1;
this.add(new JButton("Deux"),c);
c.gridx= 2;
this.add(new JButton("Trois"),c);
c.gridx = 0; c.gridy = 1;
c.ipady= 60;
c.gridwidth = 2;
this.add(new JButton("Long-Named Button 4"), c);
```



Positionnement absolu

- Il est possible de n'associer aucun gestionnaire de mise en forme à un conteneur.
- Méthodologie :
 - Initialiser the Layout container à **null** :
`contenu.setLayout(null)`
 - Ajouter votre composant au container :
`contenu.add(bouton1);`
 - Spécifier dimensions et positions des composants ajoutés:
`bouton1.setBounds(40, 40, 80, 30) ;`

Positionnement absolu

- Les composants sont placés dans le conteneur en indiquant la position précise exprimées en pixels dans un repère dont l'origine (0,0) est placée en haut et à gauche (relativement au conteneur)



Positionnement absolu

à l'aide de la méthode `setBounds` de la classe `Component`.

Composé de:

Une coordonnée X.

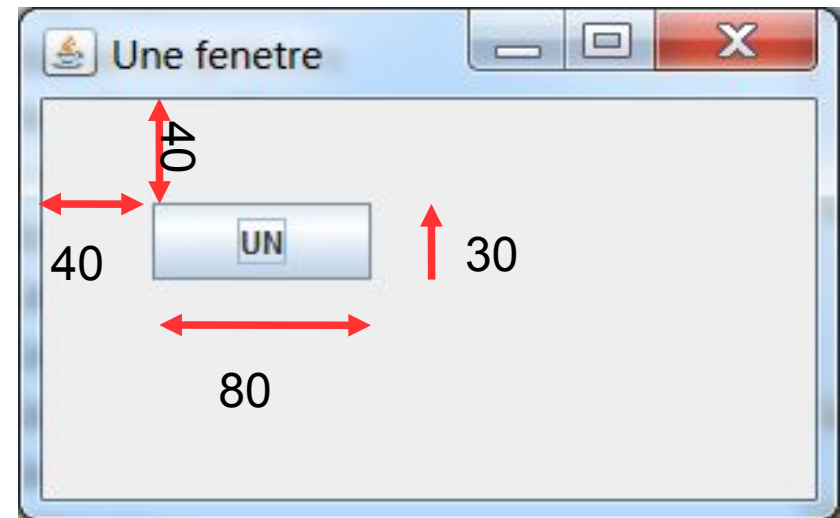
Une coordonnée Y.

La largeur du composant.

La hauteur du composant.

`setBounds(int x, int y, int width, int height)`

`bouton1.setBounds(40, 40, 80, 30) ;`



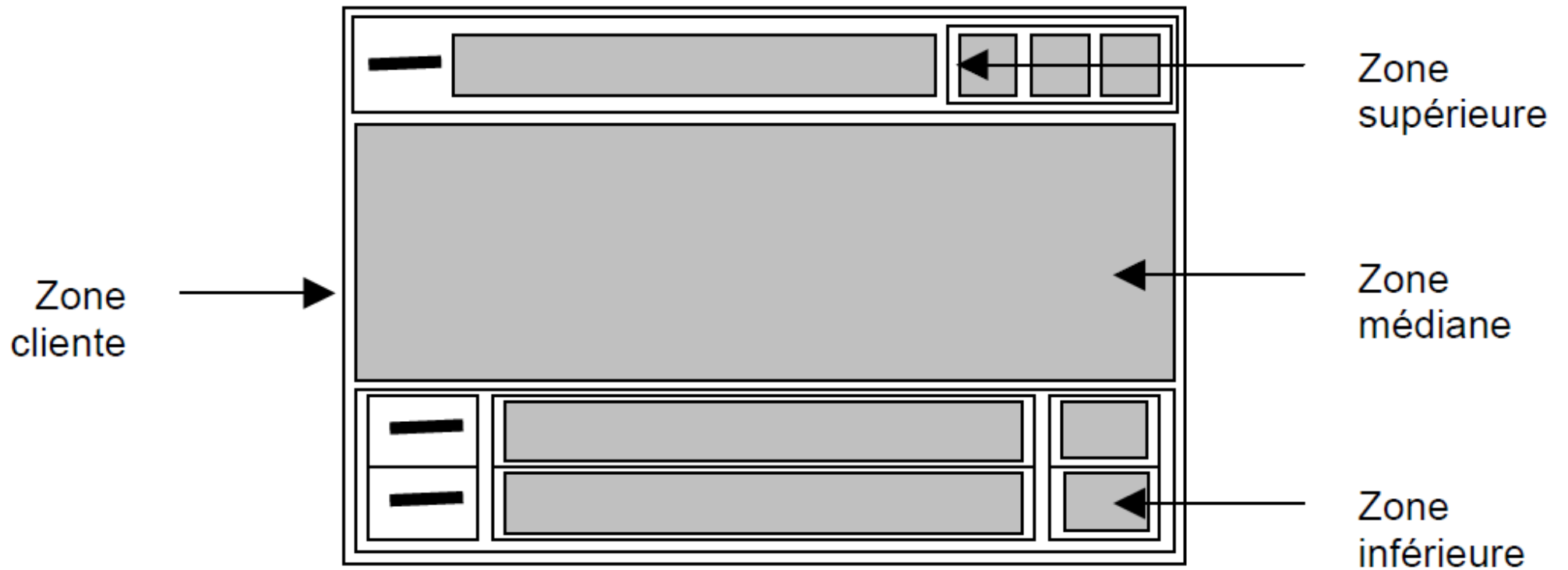
Critères de choix

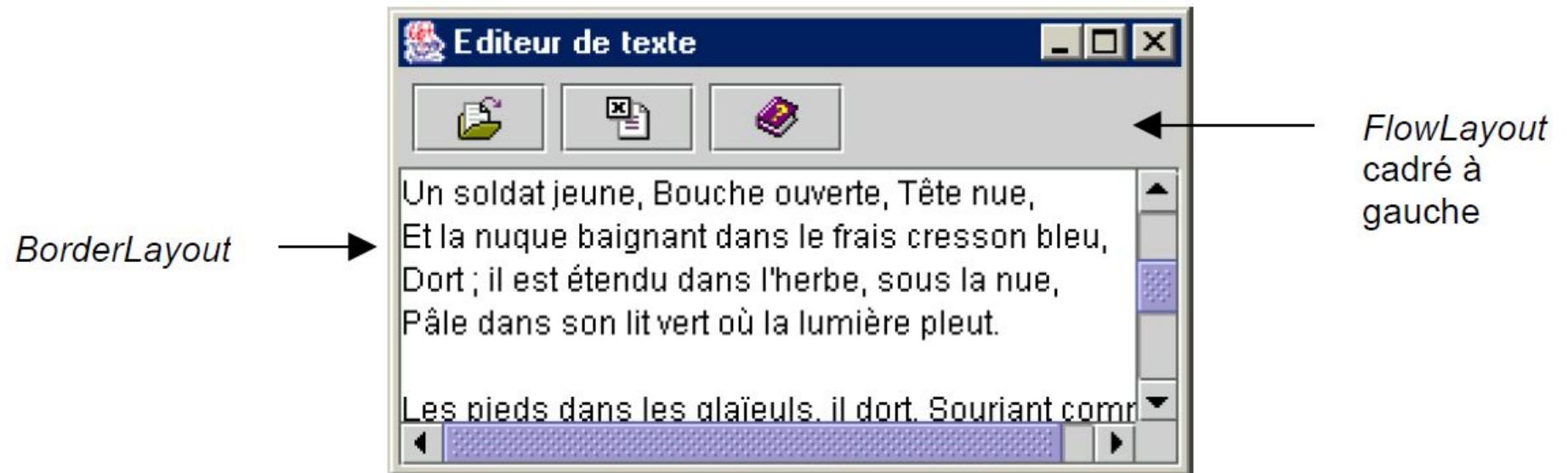
La quasi-totalité des interfaces utilisateur peuvent être obtenues en posant les contrôles dans des panneaux,

Chaque panneau ayant sa propre stratégie d'organisation (une des quatre précédemment décrites); ces panneaux pouvant à leur tour être posés dans d'autres panneaux ... etc.

La seule façon rationnelle de définir rapidement une interface utilisateur avec Swing est de:

1. dessiner sur une feuille de papier l'interface souhaitée
2. identifier les zones géographiques de contrôles
3. déduire les panneaux et leur stratégie d'organisation





EVENEMENTS

Gestion des actions de l'utilisateur

Etre source d'évènement

Les événements qui intéressent une application traduisent des actions de l'utilisateur sur les organes d'entrée (souris, clavier).

Certains sont de bas niveau → "l'utilisateur a bougé la souris", d'autres sont très élaborés, comme "l'utilisateur a choisi la commande Enregistrer dans le menu Fichier".

Un interface graphique (essentiellement les composants) doit pour interagir avec l'utilisateur réagir à certains événements

Chaque action sur un éléments graphique: boutons, windows, ... déclenche un événement.

L'élément graphique est source de l'évènement.

Classes d'événements

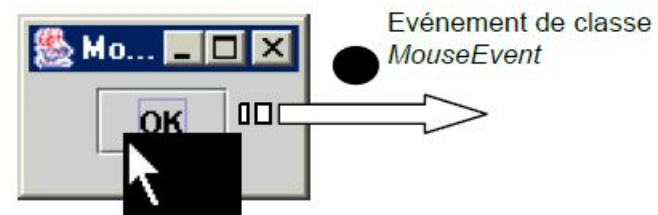
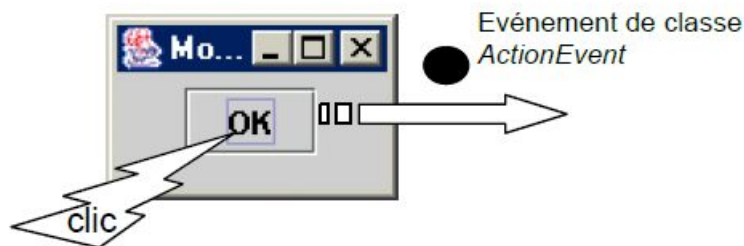
Les événements sont rangés en classes d'événements.

Événement émis	Causes possible
FocusEvent	Changement de focus sur un composant
MouseEvent	Clic sur un panneau
KeyEvent	Frappe d'une touche dans une zone de texte
WindowEvent	Agrandissement d'une fenêtre
ActionEvent	Clic sur un bouton
ListSelectionEvent	Sélection d'un élément d'une liste

Pour un même composant,

La classe de l'objet sera différente selon l'évènement :

Exemple: clic ou survol d'un bouton.



Listener pour traiter un evt

La source de l'évènement ne traite pas les événements qu'il génère. Il délègue ce traitement à des objets particuliers appelés **listener**. Un **listener** doit implémenter une interface particulière dérivée de l'interface **EventListener**.

Pour traiter un événement de type XXXEvent, un listener doit implémenter l'interface **XXXListener** :

- MouseEvent → **MouseMotionListener** : événement de souris
- MouseEvent → **MouseListener** : événement de souris
- ActionEvent → **ActionListener** : clic souris

L'interface **ActionListener** définit un listener capable d'intercepter un **ActionEvent** (click bouton)

Règle de souscription

Pour être prévenu lorsque l'évènement se produit les objets souscrivent à un interface listener en appelant la méthode :

`addXXXListener(XXXListener objetEcouleur)`

avec `a.addXXXListener(b);`

b souscrit aux évènements qui seront produits par a

Exemple :

La classe JButton fournit la méthode

addActionListener(ActionListener listener)

Avec l'appel de cette méthode sur un objet de type JButton :

`btnOk.addActionListener(new MyAppTraitementClickBtn ());`

un objet de la classe AppActionListener **souscrit** à l'évènement

Il est à l'écoute d'un click utilisateur sur le bouton Ok

Du côté des souscripteurs

Implémenter l'interface `ActionListener` :

`class` `MyAppTraitementClickBtn` `implements` `ActionListener`
re-définir la méthode `actionPerformed(ActionEvent e)`
(automatiquement appelé sur un click bouton..)
en codant un comportement particulier

```
class MyAppTraitementClickBtn implements ActionListener  
{  
    public void actionPerformed(ActionEvent e){  
        if (e.getSource() == BtnOk)  
            System.out.println("click bouton ok") ;  
    }  
}
```

composant cible peut être son propre auditeur

Exemple : fenêtre de détection des clics souris.

```
public class MaFenetre extends JFrame implements MouseListener
{
    public MaFenetre (){
        this.addMouseListener(this);
    }

    public void mouseClicked(MouseEvent e)
    {
        int x = e.getX() ; int y = e.getY() ;
        //coordonnées du curseur de la souris au moment du clic
        System.out.println("coordonnees duclick " + x + ", " + y);
    }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
}
```

Interface MouseListener

L'interface MouseListener comporte 5 méthodes correspondant chacune à un événement souris particulier qu'il faut obligatoirement Implémenter.

```
public interface MouseListener extends EventListener
{
    public void mousePressed(MouseEvent e) ;
    public void mouseReleased(MouseEvent e) ;
    public void mouseClicked(MouseEvent e) ;
    //appelé lors d'un clic souris sur un composant
    public void mouseEntered(MouseEvent e) ;
    public void mouseExited(MouseEvent e) ;
}
```

Adaptateur XXXAdapter

Comme seule la méthode `mouseClicked` nous intéresse, un `mouseAdapter` nous permet d'implémenter cette seule méthode.

```
public class MainClass extends JFrame
{
    public static void main(String args[]) {
        JFrame fen = new JFrame();
        fen.getContentPane().setBackground(Color.white);
        fen.setSize(500,500); fen.setVisible(true) ;

        fen.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e)
            {
                int x = e.getX() ; int y = e.getY() ;
                System.out.println(" pt de coordonnees " + x + ", " + y);
            }
        });
    }
}
```

Définir listener/souscripteur

- 3 possibilités :
 - La classe courante.
 - Créer une nouvelle classe, en l'implémentant.
 - Créer une classe anonyme.

classe courante implémente le listener

```
public class MyFrame extends JFrame implements ActionListener
{
    private JButton button;
    public MyFrame() {
        button = new JButton("My button");
        button.addActionListener(this);
    }
    @Override
    public void actionPerformed(ActionEvent ae) {
        System.out.println("click sur my bouton !!");
    }
}
```

une nouvelle classe définit un listener

classe indépendante qui détermine la frontière entre l'interface graphique (émission d'événements) et celle qui représente la logique de l'application (traitement des événements) .

```
public class MyFrame extends JFrame
{
    private JButton button;
    public MyFrame() {
        button = new JButton("My button");
        button.addActionListener(new MyListener());
    }
}

public class MyListener extends ActionListener
{
    @Override
    public void actionPerformed(ActionEvent ae) {
        System.out.println("I performed a click ! It works !!");
    }
}
```

classe anonyme définit un listener

```
public class MyFrame extends JFrame
{
    private JButton button;
    public MyFrame() {
        button = new JButton("My button");
        button.addActionListener(

            new ActionListener()
            {
                public void actionPerformed(ActionEvent e){
                    System.out.println("click ! It works !!");
                }
            }
        );
        add(button);
    }
}
```

voire code source [Swing/ListenerAnonymousClass](#)

ActionEvent

```
public class MyFrame extends JFrame implements ActionListener
{
    private JButton br, bj ;
    public MyFrame() {
        br = new JButton("Red");
        br.addActionListener(this);
        bj = new JButton("Yellow");
        bj.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae) {
        System.out.println("click sur bouton !!");
    }
}
```

Lorsque vous cliquez sur bouton1 et bouton2, le même message apparaît.
Les 2 boutons partagent le même listener et le même gestionnaire d'événements.

Object getSource() , String getActionCommand()

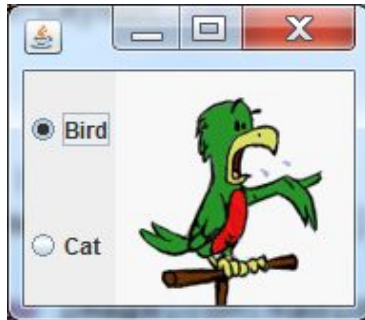
l'objet de type `ActionEvent` transmis en paramètre de `actionPerformed` et l'une des méthode de récupération du composant permet de distinguer des comportements :

```
br.setActionCommand("Rouge");  
br.addActionListener(this);  
bj.setActionCommand("Jaune");  
bj.addActionListener(this);
```

```
@Override  
public void actionPerformed(ActionEvent e)  
{  
    if ( e.getSource() == br )  
        panel.setBackground(Color.red);  
  
    if(e.getActionCommand()== "Jaune" )  
        panel.setBackground(Color.yellow);  
}
```

Exercice

- Modifier l'image avec un boutonradio

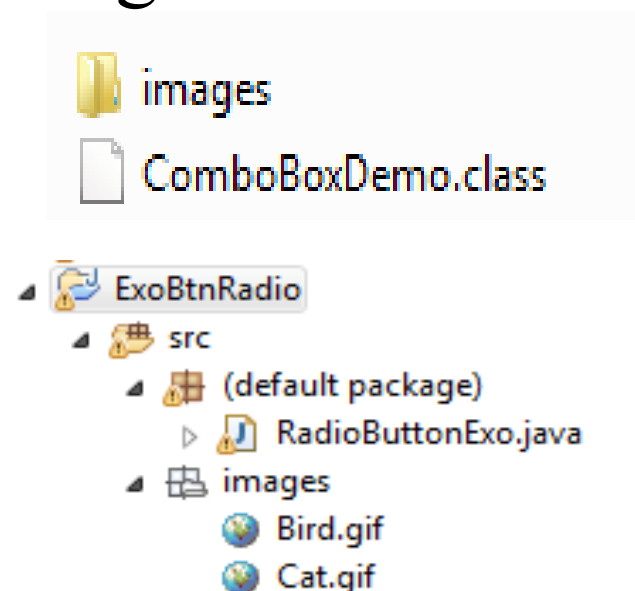


- 2 répertoires courants pour les images :

- **bin** (à côté du fichier .class)

(ce dossier se reconstruit à chaque exécution)

- **src** (à côté du fichier .java)



D'autres Listeners

- ItemListener:

Fournit la méthode **itemStateChanged** pour les objets à 2 états (coché , décoché ex : checkbox)

- WindowListener :

-Quand la fenêtre est activée, iconifiée, ...

- MouseListener :

-Click souris, ...

- KeyListener :

-Appui relâchement d'une touche du clavier ...

- FocusListener :

-prendre, perdre le focus...

classe javax.swing.Timer

L'utilisation des événements, ne se limite pas aux interfaces graphiques.

Un objet `Timer(int delay, ActionListener listener)`

- fait partie du package swing mais n'est pas un composant graphique.
- peut envoyer un événement `ActionEvent` à intervalle régulier

```
public class Metronome implements ActionListener {
    Timer timer;

    public Metronome () {
        timer = new Timer(1000, this);
        timer.start();
        try { Thread.sleep(10000); }
        catch (InterruptedException e) { timer.stop(); }
    }

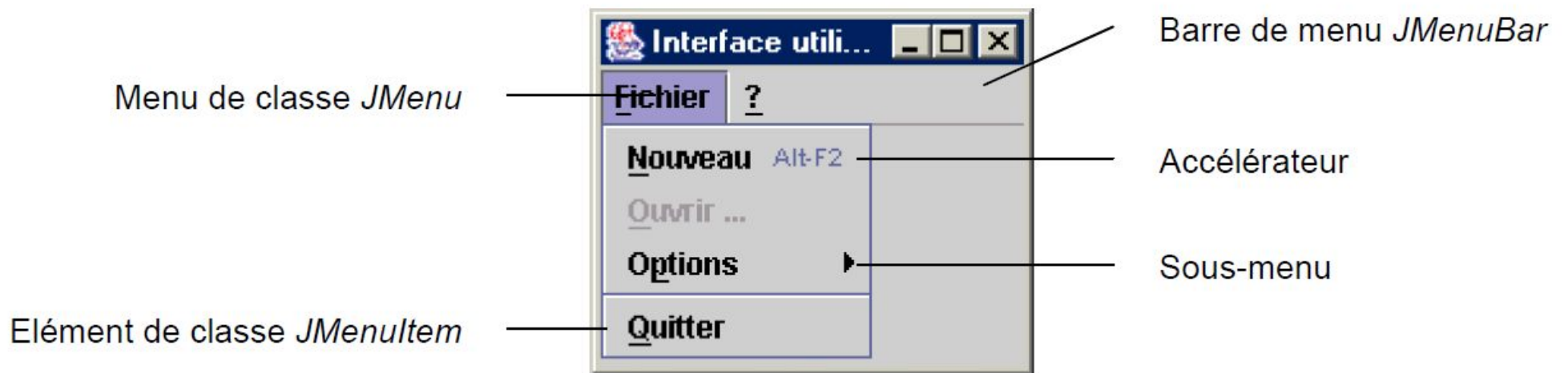
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == timer ) System.out.println("Heard It");
    }

    public static void main(String[] args) { new Metronome(); }
}
```




Menus

Créer des menus pour vos applications





Exemple

La JMenuBar est liée à la JFrame, elle contient un JMenu.
L'objet JMenu contient des JMenuItem.

```
public class MainCoursJMenu extends JFrame {
    public MainCoursJMenu(){
        MenuBar mb = new MenuBar();
        this.setMenuBar(mb); //barre de menu s'ajoute à la JFrame
        Menu m = new Menu("liste des elements");
        m.addActionListener(afficherMenuListener); //abonnement
        //le menu est contenu dans la barre de menu
        mb.add(m); //ajout du menu à la barre de menu
        //les items (type JMenuItem) du menu sont contenu dans le menu
        m.add(new JMenuItem("element 1")); //ajout des items au menu
        m.addSeparator();
        m.add(new JMenuItem(" 2eme element "));
        //Un JMenuItem peut s'associer à ActionCommand et utiliser un
        //ActionListener pour détecter et traiter un click dessus (comme
        un bouton)
        ActionListener afficherMenuListener = new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                System.out.println("Elmnt menu :"+ event.getActionCommand());
            }
        };
    }
};
```



Menus

JPopupMenu

- Un menu contextuel
- Quand l'utilisateur utilise un right-click sur la zone client.
- Comme un Jmenu, se remplit avec JMenuitem.



Listing

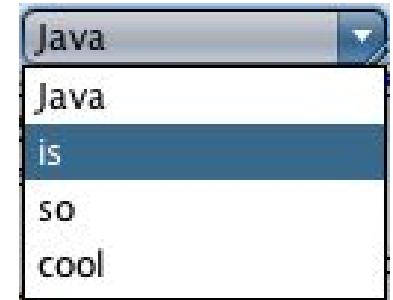
Lister vos données



list

JComboBox

- Une drop down list.
- Constructeurs :
 - *JComboBox(Object[] items)*
 - *JComboBox(Vector<?> items)*
- Méthodes :
 - *int getSelectedIndex()*
 - *Object getSelectedItem()*



Les items dans le JComboBox sont de type strings.

list

JComboBox(Object[] items)

```
public class Person { private String prenom,nom;}
```

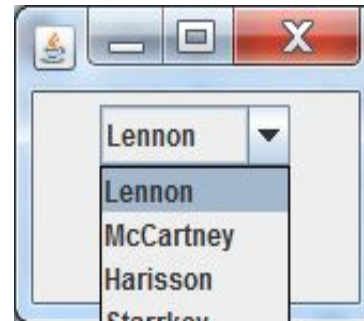
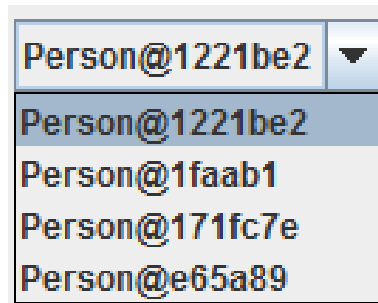
```
Person person1 = new Person("John", "Lennon");
```

```
Person person2 = new Person("Paul", "McCartney");
```

```
Person person3 = new Person("Georges", "Harisson");
```

```
Person person4 = new Person("Richard", "Starrkey");
```

```
JComboBox<Person> comboBoxPersonne = new JComboBox<Person> (  
    new Person[] { person1, person2, person3, person4 }  
);
```



Note: Le texte affiché dans le JComboBox est le **toString()** des items.

```
public String toString() { return (nom); }
```

Jcombox exercice

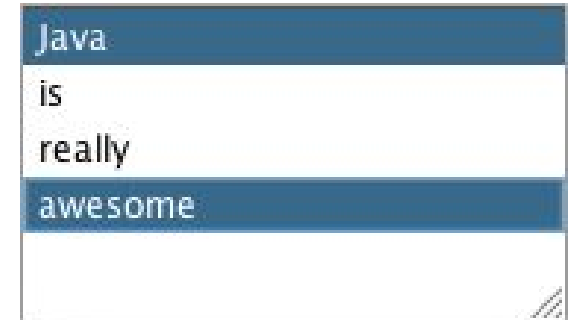
sélectionner un item dans la combo box
il sera ajouté dans un tableau



Listing

JList

- Une simple list.
- multi selection possible.
- Modèle implicite par défaut immuable



```
public CoursJList()  
{  
    String [] noms = {"Java", "is", "really", "awesome"};  
    JList lstNom = new JList(noms);  
}
```

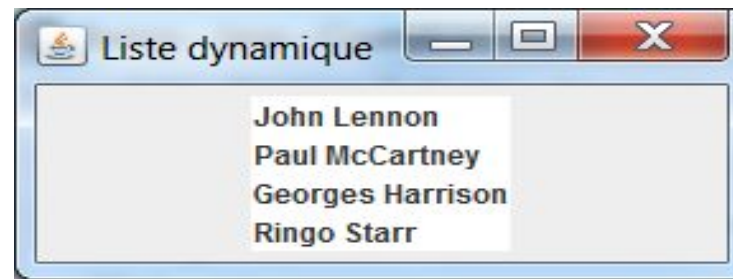
Le tableau noms est le "**modèle**" de la liste.
La liste est une **vue** du modèle.



JList: ajout / suppression d'éléments

Pour modifier le contenu de la liste
associer à la liste un "modèle par défaut" instance de la
classe **DefaultListModel**.

```
DefaultListModel<String> listModel = new DefaultListModel<String>();  
listModel.addElement("Pete Best");  
listModel.addElement("John Lennon");  
listModel.addElement("Paul McCartney");  
listModel.addElement("Ringo Starr");  
JList<String> Malist = new JList<String>(listModel);  
this.add(Malist);  
listModel.remove(0);  
listModel.insertElementAt("Georges Harrison", 2);
```





Une table ressemble à une liste avec plusieurs colonnes.

- Objet instance de la classe JTable
- S'associe à un modèle
- Instance de classe DefaultTableModel

Prenom	Nom
John	Lennon
Paul	McCartney
Georges	Harisson
Richard	Starrkey



Modèle pour les tableaux

■ interface : *TableModel*.

■ Classe: *AbstractTableModel*, *DefaultTableModel*.

Définition de votre modèle en héritant de la classe *AbstractTableModel* ou par implémentation de l'interface *TableModel*

méthodes présentes dans la classe *AbstractTableModel* :

- **int getColumnCount()** : retourne le nombre de colonne à afficher.
- **int getRowCount()** : retourne le nombre de ligne à afficher
- **String getColumnName(int col)**: retourne le nom de la colonne col
- **Object getValueAt(int row, int col)**:

retourne l'attribut de l'objet présent dans la case (row , col)

- **void setValueAt(Object value, int row, int col)** :

modifie le contenu de la case (row, col) pour les cases en mode édition.

boolean isCellEditable(int row, int col) : case en mode édition **return true;**

Modèle pour les tableaux

```
public class Controleur extends AbstractTableModel
private final String[] entetes = {"Prenom", "Nom"};
private ArrayList<Person> TabPers ;

public Controleur() {
    super();
    TabPers = new ArrayList<Person>();
    TabPers.add(new Person("John", "Lennon"));
    ...
}

public int getRowCount(){return TabPers.size();    }
public int getColumnCount(){return entetes.length;    }
public String getColumnName(int columnIndex){
    return entetes[columnIndex];
}

public Object getValueAt(int row, int col) {
    Person p = TabPers.get(row);
    if(col == 0) return p.getPrenom();
    if(col == 1) return p.getNom();
    return null;
}
```

fireTableDataChanged

Pour la mise à jour de la vue ...

Ajout /suppression /modification

appeler la méthode **fireTableDataChanged()**

The view will refresh itself 😊

```
public void removePersonne(int rowIndex) {  
    TabPers.remove(rowIndex);  
    fireTableDataChanged();  
}
```



Prenom	Nom
John	Lennon
Paul	McCartney
Georges	Harisson
Richard	Starrkey
Johnny	Guitar

supprimer



Prenom	Nom
John	Lennon
Paul	McCartney
Georges	Harisson
Richard	Starrkey
Johnny	Guitar

supprimer

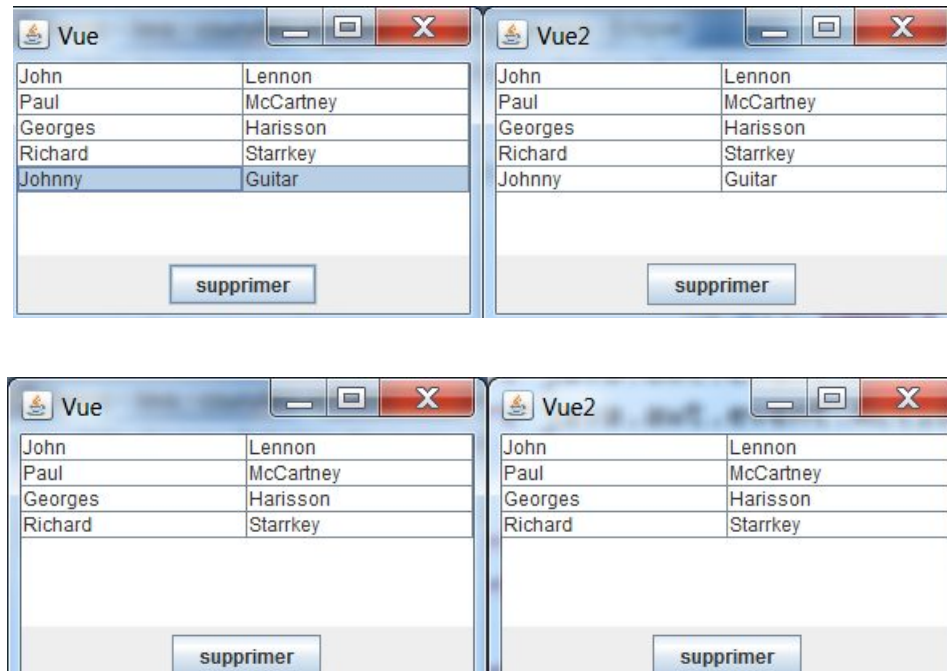


Prenom	Nom
John	Lennon
Paul	McCartney
Georges	Harisson
Richard	Starrkey

supprimer

Using Models

le modèle propage automatiquement les modifications aux listeners interested, making it easy for the GUI to stay in sync with the data
 JTable → AbstractTableModel



Interface graphique

PEINDRE ET REPEINDRE



Présentation

- Les composants graphiques sont peints (sur l'écran) : une première fois dès leur affichage initial, ensuite chaque fois qu'ils ont été totalement ou partiellement modifiés, effacés, agrandis...
- Aussi longtemps que l'apparence standard d'un composant vous convient, vous n'avez pas à vous préoccuper de son affichage : le composant prend soin de lui même.
- Pour dessiner des composants personnalisés avec une apparence graphique propre:
 - Dans un panneau objet de la classe `Jpanel`, vous redéfinissez la méthode `void paintComponent(Graphics g)` de la classe `JComponent`

Les classes Graphics

- `paintComponent` fournit un objet de type `Graphics`.
- Un objet `graphic` se récupère implicitement comme paramètre de la méthode: *`void paintComponent(Graphics g)`*
- Un contexte graphique est associé à cet objet. Il encapsule l'ensemble des informations et des outils nécessaires pour réaliser des opérations graphiques à travers un ensemble de méthodes `get/set` dont les plus importantes sont ...

Méthode de la classe Graphics

- *GetColor()* , *setColor(Color c)* :
 - Recupere ou fixe la couleur courante sur le contexte graphique .
- *drawLine(int x1, int y1, int x2, int y2)* :
 - Dessine une ligne, utilisant la couleur courante, entre les points (x1, y1) et (x2, y2).
- *drawRect(int x, int y, int width, int height)* :
 - Dessine le contour du rectangle spécifié.
- *FillRect(int x, int y, int width, int height)* :
 - Remplit le rectangle spécifié.
- *drawOval(int x, int y, int width, int height)* :
 - Dessine le contour d'un ovale.
- *fillOval(int x, int y, int width, int height)* :
 - Remplit un oval définit par les dimensions du rectangle specifié avec the

Repaint

- `paintComponent (Graphics)` ne peut pas être appelée directement pour :
 - redessiner
 - rafraîcher vos composants.
- Pour l'appeler utiliser l'une des méthodes :
 - `repaint()` : demande à repeindre le composant.
 - `repaint (Rectangle r)` : demande à repeindre la region spécifiée du composant.

Exemple

```
public class MyPanel extends JPanel {
```

```
    @Override
```

```
    protected void paintComponent(Graphics g) {
```

```
        super.paintComponent(g);
```

```
        g.setColor(Color.RED);
```

```
        g.fillOval(0, 0, getWidth(), getHeight());
```

```
        g.setColor(Color.BLACK);
```

```
        g.drawOval(0, 0, getWidth(), getHeight());
```

```
        g.drawString("Hello World !", getWidth() / 3, getHeight() / 2);
```

```
    }
```

```
}
```

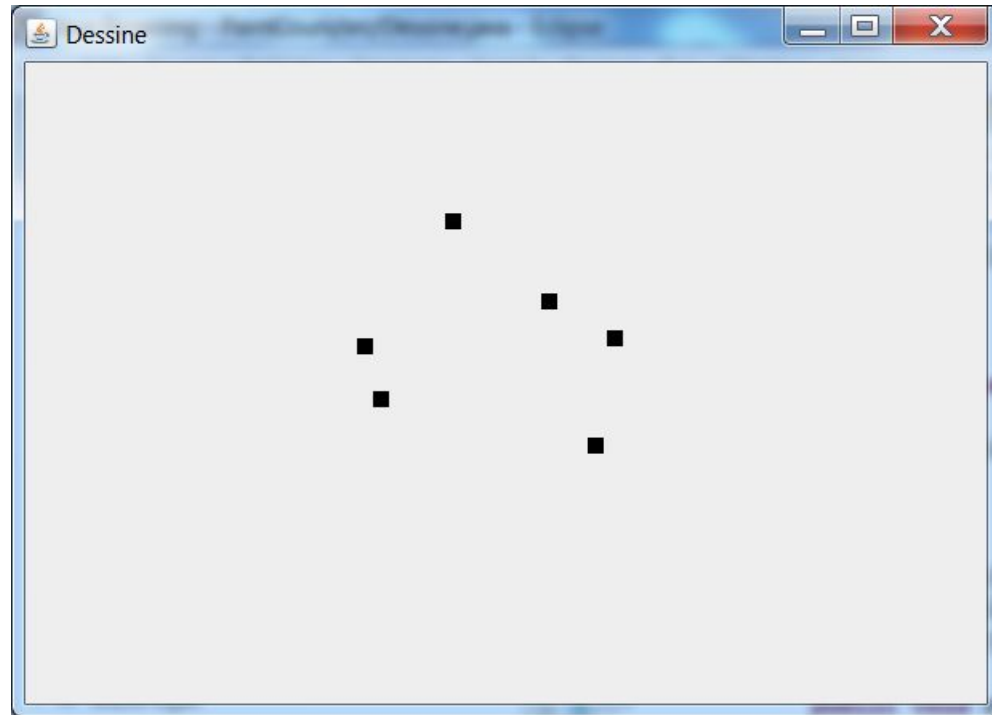


Peindre & repeindre

Exemple avec la methode repaint

Sur chaque click souris, un point se dessine.

Avec la méthode **repaint**, le panneau se redessinne pour ajouter le nouveau point .



Exemple avec la methode repaint

```
public class Dessine extends JPanel
{
    private ArrayList<Point> points = new ArrayList<Point>();

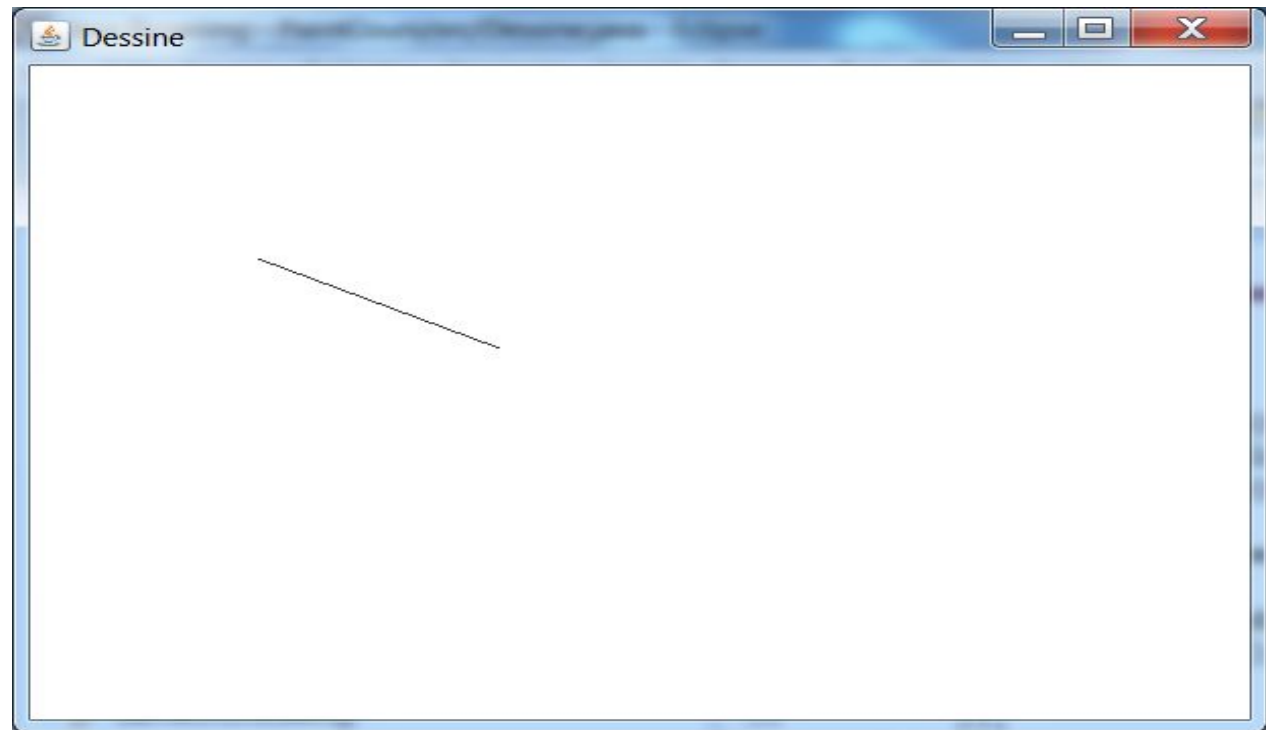
    Dessine(){
        addMouseListener(new MouseAdapter()
        {
            public void mouseClicked(MouseEvent e)
            {
                points.add(new Point(e.getX(), e.getY()));
                repaint();
            }
        });
    }

    public void paintComponent(Graphics g){
        for(Point p : this.points)
            g.fillRect(p.x, p.y, 10,10);
    }
}
```

Exercice

Code source: `exoDessinLignePtClick`

Dessiner un polygone que l'utilisateur définit en cliquant dans la fenêtre pour définir les points qu'il souhaite placer comme sommets des segments de la figure.



Pour aller plus loin

2 références indispensables pour programmer des interfaces graphiques en Java sont:

- la documentation en ligne de l'API Java
<http://docs.oracle.com/javase/6/docs/api/>
notamment tout ce qui concerne les paquets dont les noms commencent par `java.awt` et `javax.swing`,
- le tutoriel en ligne :

<https://docs.oracle.com/javase/tutorial/uiswing/index.html>

INTERFACE GRAPHIQUE

