Construire une interface graphique en Java

Introduction à Java Swing

Avertissement

L'objectif de ce cours est de présenter les différentes techniques concernant la construction d'une interface graphique en JAVA.

La conception d 'interfaces graphiques étant loin d'être triviale, et les packages proposés pour cela par le JDK étant parmi les plus « complexes » (29 packages) nous n'avons pas la prétention d'être exhaustif. Il s'agit de donner les éléments de bases qui vous permettront ensuite d'aller plus loin dans la création d'interfaces utilisateur.

Une partie de ce cours est basée sur un exemple tiré du chapitre 4 du livre « *Java La Synthèse , 2ème edition* » de G. Clavel, N. Mirouze, S. Moucherot, E. Pichon et M. Soukal (InterEditions).

Certains éléments concernant la gestion des événements sont directement inspirés du cours « *Swing Crash course* » de Matthias Hauswirth http://www.isbiel.ch/I/Resources/Comp/Sun/Java/Swing/

D'autres sont inspiré du cours « *Graphical User Interface in Java* » de Jonas Kvarnström

On pourra également se référer à

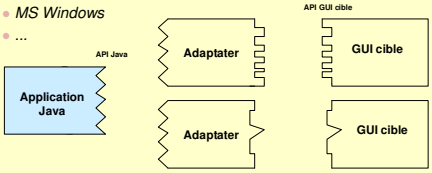
- « *JAVA in a Nutshell* » , chapitre 7, David Flanagan, 2nd Ed. O'Reilly 1997
- « *Java Foundation Classes in a nutshell* » , David Flanagan, Ed. O 'Reilly 1999
- « *Swing la synthèse* » , Valérie Berthié, Jean-Baptiste Briaud, Ed. Dunod 2001

API graphique pour les applications Java

- Java : un langage indépendant des plateformes (cross-platform language)
  - *une même programme doit pouvoir être utilisé dans des environnements (matériel et OS) différents sans recompilation.*
- Nécessité d'offrir une API pour les interfaces graphiques indépendante elle aussi des plateformes
  - *Classes et interfaces java*
  - *Modèle de gestion des événements*
  - *exemple : une classe TextField pour définir un champ de saisie de texte*
    - `TextField(String content)`
    - `TextField()`
    - `void setText(String content)`
    - `String getText()`
    - `...`

API Java et systèmes de fenêtrage

- Problème : les systèmes de gestion d'interface utilisateur (GUI Graphical User Interface systems) sont très différents les uns des autres :
  - *X Window + motif*
  - *X Window + gtk*
  - *MacOS X*
  - *MS Windows*
  - *...*



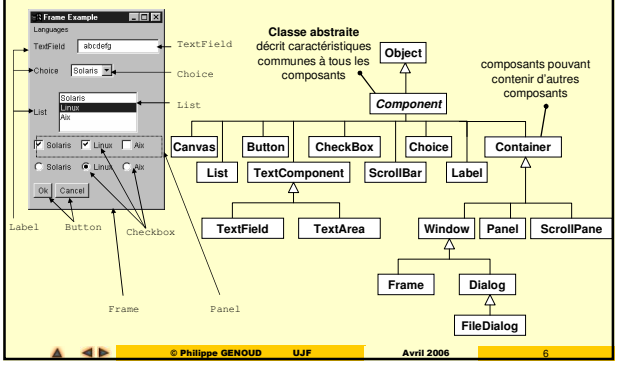
- Deux stratégies possibles :
  - *faire une utilisation maximale du système graphique cible*
  - *faire une utilisation minimale du système graphique cible*

API Java pour GUI

- Utilisation maximale du système graphique sous-jacent
  - *L'objet TextField délègue la plupart de ses tâches à un composant natif.*
    - *Le programmeur java utilise un objet TextField*
    - *Objet TextField délègue à une classe adaptateur dépendant de l'OS : MotifTextField, GTKTextField, WindowsTextField, MacOSTextField ....*
    - *Le système graghique natif réalise le plus gros du travail*
  - *Avantages / désavantages*
    - *(+) apparence et le comportement (look and feel) des interfaces Java identique à celui d'applications "ordinaires"*
    - *(+) pas besoin de réimplémenter des composants existants*
    - *(-) ne permet de ne fournir que les composants disponibles sur toutes les plateformes*
    - *(-) difficile de garantir un comportement identique sur toutes les plateformes*
- Choix adopté pour JAVA AWT
  - *AWT Abstract Window Toolkit*
  - *packages* **`java.awt.*`** *présents dans Java depuis version 1.0.*
  - *conception pas toujours très judicieuse (cf les évolutions de awt entre version 1.0 et 1.1 de Java)*

Composants graphiques de AWT

- palette de composants fournies par awt ne contient que des composants simples
  seuls les composants standard existants dans tous les systèmes d'exploitation peuvent être présents dans awt

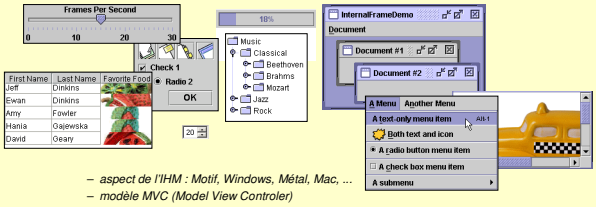


API Java pour GUI

- Utilisation minimale du système graphique sous-jacent
  - *Utiliser des éléments natifs uniquement pour opérations de base*
    - *Ouvrir une fenêtre, dessiner des lignes/du texte, gestion primitive des événements*
  - *Réaliser tout le reste en Java*
    - *L'objet TextField s'affiche en dessinant des lignes,...*
  - *Avantages / désavantages*
    - *(+) facilité d'éviter les différences entre plateformes*
    - *(+) n'importe quel nouveau composant d'interface est immédiatement disponible sur **toutes** les plateformes*
    - *(-) besoin de réimplémenter tous les composants d'interface*
    - *(-) les applications java n'ont pas le même look and feel que les applications "ordinaires"*
    - *(-) lenteur ?*
  - *C'est le choix adopté par SWING*
    - *packages `javax.swing.*` dans JDK depuis version 1.2*

Swing et JFC

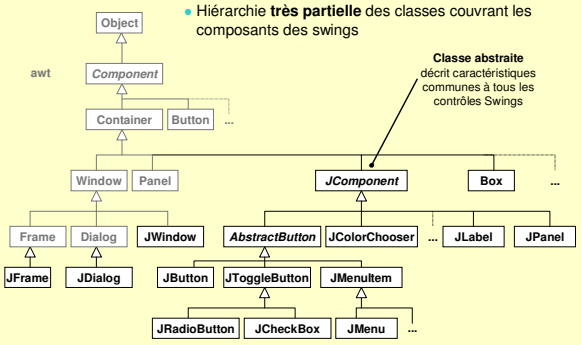
- Swing s'intègre aux JFC (**Java Fundation Classes** lancéés par SUN en 97 pour la création d'interfaces graphiques plus élaborées que AWT et intégré depuis version 2 de Java (1.2))
  - *JFC = Java 2D API + copier coller inter-applications + Swing + Accessibilité*
  - *Swing*
    - *Composants légers (lightweight) 100% java*
      - *Prennent en charge leur affichage sans passer par des objets « Peers » gérés par le système*
      - *multiplication des composants plus riches en fonctionnalités (listes arborescentes, grilles….)*



  - *aspect de l'IHM : Motif, Windows, Métal, Mac, ...*
  - *modèle MVC (Model View Controler)*

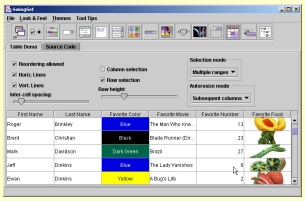
Composants graphiques Swing

- Hiérarchie **très partielle** des classes couvrant les composants des swings



Swing : démos et tutoriaux

- jdk1.5.0_06/demo/jfc/SwingSet2



- Java tutorial consacré aux swing
  - *http://java.sun.com/docs/books/tutorial/uiswing/*
- En particulier l'index visuel des composants swing
  - *http://java.sun.com/docs/books/tutorial/uiswing/components/components.html*

Penser à étudier des programmes sources pour apprendre à se servir des swing

Java Fundation Classes

- **packages awt et swing tous les deux présents dans la plateforme Java**
- **Quel package utiliser : AWT ou Swings ???**

> « Swing provides many benefits to programmers and end users. Unless you have a good reason not to convert (and use AWT), **we urge you to convert to Swing as soon as possible**. »
>
> M. Campione - K. Walrath *« the Java Tutorial »*

Puisqu'elles le disent si gentiment, allons-y…

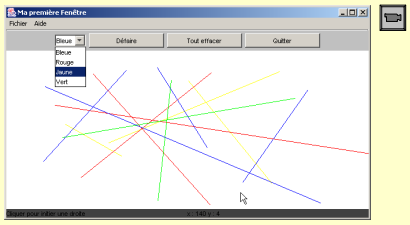
- **Si Swing et AWT ne sont plus vraiment en concurrence, un troisième larron a fait son apparition : SWT (Standard Window ToolKit) développé par IBM – Eclipse**
  - *approche similaire à AWT,*
  - *beaucoup plus ambitieux, nombreux composants, modèle MVC (JFace)*
  - *windows, motif, gtk, Mac OS X*

Introduction à Swing

- *composants de base*
- *conteneurs*
  - *fenêtres*
- *placement des composants (layout managers)*
- *gestion de l'interaction (événements)*

Exemple développé*

- Construction d'un éditeur graphique simple
  - *Créer une fenêtre*
  - *Gérer un menu*
  - *Gérer différents composants d'interaction simples (boutons, champ de saisie de texte…)*
  - *Dessiner dans une fenêtre*



*d'après « *Java La Synthèse , 2ème edition* » de G. Clavel, N. Mirouze, S. Moucherot, E. Pichon et M. Soukal (InterEditions).

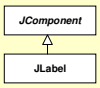
Structure de l'interface graphique

- Création de l'interface graphique passe **forcément** par une instance de la classe **`JFrame`**
- Du point de vue du système d 'exploitation cette fenêtre représente l'application
- La fenêtre joue le rôle de « **conteneur** » dans lequel vont être disposés les différents éléments constitutifs (**composants**) de l'interface graphique de l'application (boutons, listes déroulantes, zone de saisie…)
  - *ces éléments désignés sous les termes de*
    - *contrôles (IHM)*
    - *composants (components en JAVA)*
    - *widgets (Xwindow-Motif)*

Composants graphiques Swings — JLabel

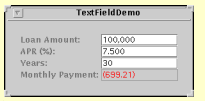
- Usage : afficher du texte statique et/ou une image
  - *Ne réagit pas aux interactions de l'utilisateur*
- Définition du texte du label
  - *Dans le constructeur* `JLabel lb = new JLabel("un label");`
  - *Par la méthode setText*
    `lb.setText("un autre texte pour le label");`
  - *Méthode symétrique* `lb.getText()`
- Définition d'une icône :
  - *Par défaut pas d'image*
  - *Dans le constructeur*
  - *Par la méthode setIcon*
    `lb.setIcon(new ImageIcon("info.gif"));`
  - *Spécification de la position du texte par rapport à l'icône*
    `lb.setVerticalTextPosition(SwingConstants.BOTTOM);`
    `lb.setHorizontalTextPosition(SwingConstants.CENTER);`
- Mise en forme du texte
  - *setText supporte HTML*
    `lb.setText("<html>ce texte est en <B>gras</B></HTML>")`

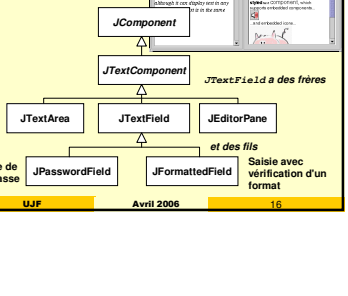
  attention : si texte HTML incorrect, exception levée à l'exécution



Composants graphiques Swing — JTextField

- Usage : présente un champ de saisie de texte
- Création d'un **`JTextField`**
  - `JTextField jtf = new JTextField();`
  - `JTextField jtf = new JTextField("un texte");`
  - `JTextField jtf = new JTextField(20);`
- Modification du texte
  - *Par interaction de l'utilisateur*
  - `jtf.setText("le texte");`
- Récupérer le texte
  - `jtf.getText();`
- Copier / Coller
  - `jtf.copy()` *ou* `jtf.cut()`
  - `jtf.paste();`
- Interdire saisie
  - `jtf.setEditable(false);`

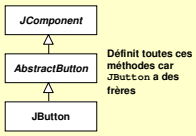




Composants graphiques Swing — JButton

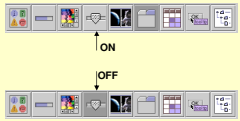
- Usage : un bouton permettant de déclancher une action

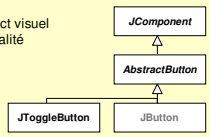
- Création d'un **`JButton`**
  - `JButton jbt = new JButton("Titre");`
  - `JButton jbt = new JButton("Titre", icon);`
- Association d'une icône
  - *à l'instanciation*
  - *méthodes* `setIcon()`, `setRollOverIcon()`, `setPressedIcon()`, `setDisabledIcon()`
- Association d'un raccourci clavier
  - `jbt.setMnemonic('b'); // Alt + b`
- Enlever / ajouter la bordure
  - `jbt.setBorder(false);`
- Enlever le cadre indiquant le focus
  - `jbt.setFocusPainted(false);`
- Simuler un clic utilisateur
  - `jbt.doClick();`



Composants graphiques Swing — JToggleButton

- Usage : un bouton à deux états. Utilisé lorsque l'aspect visuel du bouton doit refléter état (ON/OFF) d'une fonctionnalité (exemple boutons dans une barre d'outils).

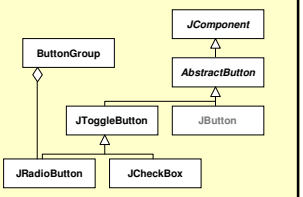




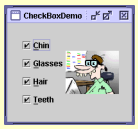
- Association d'une icône aux états
  - `jtbt.setIcon(icon1)` état "non sélectionné"
  - `jtbt.setSelectedIcon(icon2)` état "sélectionné"
- Forcer l'état
  - `jtbt.setSelected(true)`
- Consulter l'état
  - `jtbt.isSelected()` → `true ou false`

Composants graphiques Swing — JCheckBox et JRadioButton

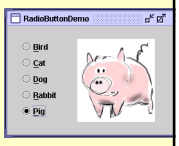
- Usage :
  - *cases à cocher (états indépendants)*
  - *et boutons radio (états dépendant au sein d'un groupe de boutons)*




```
JCheckBox cb1 = new JCheckBox("Chin");
JCheckBox cb2 = new JCheckBox("Glasses");
JCheckBox cb3 = new JCheckBox("Hair");
JCheckBox cb4 = new JCheckBox("Teeth");
```




```
ButtonGroup groupe = new ButtonGroup();
JRadioButton b1 = new JRadioButton("Bird");
JRadioButton b2 = new JRadioButton("Cat");
...
JRadioButton b5 = new JRadioButton("Pig");
b5.setSelected(true);
groupe.add(b1);
groupe.add(b2);
...
groupe.add(b5);
```



Capacités communes à tous les composants

- activation / désactivation du composant (le composant réagit ou non aux interactions avec l'utilisateur)
  - *setEnabled(boolean);*
  - *isEnabled() → boolean*

- visibilité / invisibilité d'un composant
  - *setVisible(boolean);*
  - *isVisible() → boolean*
- apparence du curseur (changement de l'apparence du curseur en fonction du composant qu'il survole)
  - *setCursor(java.awt.Cursor)*

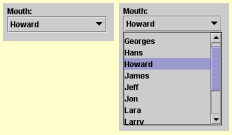
```
bt.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
```

- bulles d'aide ("tooltips")
  - *setToolTopText(String)*
    possibilité d'utiliser HTML pour formater le texte d'aide
- couleur
  - *setBackGroundColor(java.awt.Color)*
  - *setForeGroundColor(java.awt.Color)*
- …

Composants graphiques Swing — JComboBox

- Usage : liste déroulante dans laquelle l'utilisateur peut choisir un item
- indiquer les items à afficher
  - *en passant en paramètre du constructeur un tableau d'objets ou un java.util.Vector*

```
Object[] items = { "lundi", "mardi",
    "mercredi", "jeudi",
    "vendredi", "samedi" };
JComboBox cb = new JComboBox(items);
```



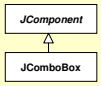
  - *en passant par* `addItem(Object)`

```
cb.addItem("dimanche");
```

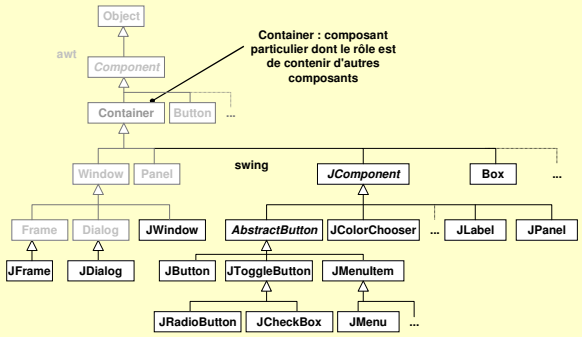
- combo box éditable
  - *le champ qui représente le texte sélectionné est éditable (c'est un JTextField)*

```
Vector pays = new Vector();
pays.add("Angleterre");
...
JComboBox cb = new JComboBox(pays);
cb.setEditable(true);
```

- récupération de l'item sélectionné
  - `getSelectedIndex()` → *int*
  - `getSelectedItem()` → *Object*



Container

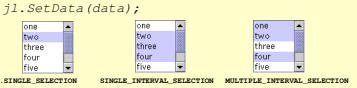


Composants graphiques Swing — JList

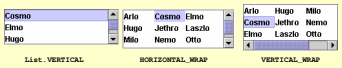
- Usage : liste déroulante dans laquelle l'utilisateur peut choisir un item
  - *à préférer à une combo box lorsque le nombre d'éléments de la liste est grand*
  - *permet une sélection multiple*
- indiquer les éléments à afficher
  - *en passant en paramètre un tableau d'objets ou un java.util.Vector*

```
Object[] data = { "lundi", "mardi",
    "mercredi", "jeudi","vendredi", "samedi" };
```

  - *dans le constructeur :* `JList jl = new JList(data);`
  - *par la méthode* **`setData`** *:* `jl.SetData(data);`
- définir le mode de sélection
  - `setSelectionMode(int)`

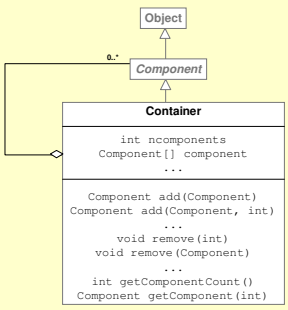


- définir l'arrangement des éléments
  - `setLayoutOrientation(int)`



- récupérer la sélection
  - `getSelectedIndex()` → *int*, `getSelectedIndices()` → *int[]*
  - `getSelectedValue()` → *Object*, `getSelectedValues()` → *Object[]*

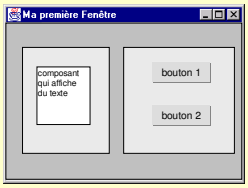
Container

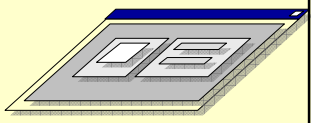


- Container est un composant
  - *un container peut contenir d'autres containers*
  - *permet de créer des arborescences de composants*

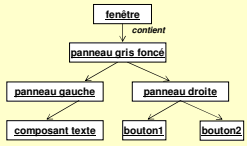
arborescence de containers

- construction d'une interface : élaborer une arborescence de composants à l'aide de containers jusqu'à obtenir l'interface souhaitée





```
panneauGauche.add(composantTexte);

panneauDroite.add(bouton1);
panneauDroite.add(bouton2);

panneauGris.add(panneauGauche);
panneauGris.add(panneauDroite);

fenêtre.add(panneauGris);
```

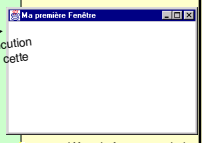


Fenêtre d'application

- Toute fenêtre d'application est représentée par une classe dérivant ou utilisant la classe **`JFrame`** du package **`javax.swing`**

```
import javax.swing.JFrame;

public class MyFrame extends JFrame {

   final static int HAUTEUR = 200;
   final static int LARGEUR = 300;

   public MyFrame() {
      setTitle("Ma première Fenêtre");
      setSize(LARGEUR,HAUTEUR);
      setVisible(true);
   }

 public static void main(String[] args) {
   new MyFrame();
   do {
     System.out.println("dans le thread principal");
     System.out.println("voulez vous poursuivre le thread
                                          principal ?");
   } while (LectureClavier.lireOuiNon());
   System.out.println("fin thread principal");
 }
}
```

- affiche la fenêtre à l'écran
- lance un « thread » d'exécution pour gérer interactions sur cette fenêtre



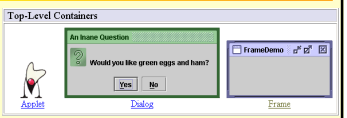
- se comporte comme toute fenêtre du système d'exploitation :
- peut être redimensionnée, déplacée, …

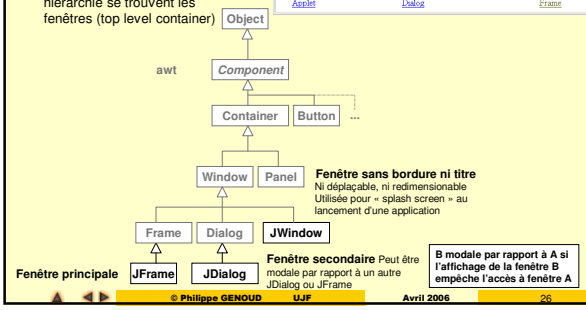
- par défaut la fermeture de la fenêtre la rend simplement invisible elle ne termine pas son thread d'exécution !

- l 'exécution du programme principal ne se termine que lorsque le « thread » lié à la fenêtre se termine lui aussi

Top Level Containers

- Pour apparaître sur l'écran, tout composant doit se trouver dans une hiérarchie de "conteneurs"
- A la racine de cette hiérarchie se trouvent les fenêtres (top level container)





Fenêtre sans bordure ni titre Ni déplaçable, ni redimensionable Utilisée pour « splash screen » au lancement d'une application

Fenêtre secondaire Peut être modale par rapport à un autre JDialog ou JFrame

B modale par rapport à A si l'affichage de la fenêtre B empêche l'accès à fenêtre A

Fenêtres — Comportement

- `setSize(int largeur, int hauteur)` fixe la taille de la fenêtre
- `setLocation(int x, int y)` fixe la position de la fenêtre (son coin supérieur gauche) sur l'écran
  - *java.awt.Toolkit permet d'obtenir la taille de l'écran*

```
Toolkit tk = Toolkit.getDefaultToolKit();
Dimension dimEcran = tk.getScreenSize();
```

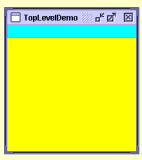
- `setResizable(boolean)` autorise ou non le redimensionnement de la fenêtre par l'utilisateur
- `setTilte(String)` définit le contenu de la barre de titre de la fenêtre
- `toFront()` `toBack()` fait passer la fenêtre au premier plan ou à l'arrière plan

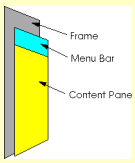
Fenêtres — Comportement

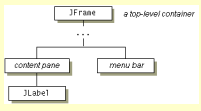
- `new JFrame(...)` ou `new JDialog(...)` crée un objet fenêtre mais ne provoque pas son affichage
- `setVisible(true)` (`show()`) affiche la fenêtre
- `setVisible(false)` (`hide()`) cache la fenêtre , mais ne la détruit pas.
  - *l'objet fenêtre est son contenu existent toujours en mémoire*
  - *la fenêtre peut être réaffichée ultérieurement*
- `dispose()` permet de libérer les ressources natives utilisées par le système pour afficher la fenêtre et les composants qu'elle contient
- par défaut à la fermeture d'une fenêtre, elle est simplement rendue invisible
  - *on verra plus tard comment définir nos propres comportements*
  - *possibilité d'associer un comportement choisi parmi un ensemble de comportements prédéfinis*
    - *`setDefaultCloseOperation(int operation)`*

```
WindowConstants.HIDE_ON_CLOSE
WindowConstants.DO_NOTHING_ON_CLOSE
WindowConstants.EXIT_ON_CLOSE
WindowConstants.HIDE_ON_CLOSE
```


Fenêtres







- les composants qui seront visibles dans la fenêtre seront placés dans un conteneur particulier associé à celle-ci : Content Pane
  - *pour récupérer ce conteneur :* `getContentPane()` → `Container`
- la fenêtre peut contenir de manière optionnelle une barre de menus (qui n'est pas dans le content pane)

Ajout d'un composant à une fenêtre

Avec AWT

```
import java.awt.*;

public class MyFrame extends  Frame {

   final static int HAUTEUR = 200;
   final static int LARGEUR = 300;

   public MyFrame() {
      setTitle("Ma première Fenêtre");
      setSize(LARGEUR,HAUTEUR);

      Button b =
         new Button("Mon 1er composant");
      add(b);
   }
}
```

1) Création du composant
2) ajout au « conteneur »

Avec les Swings

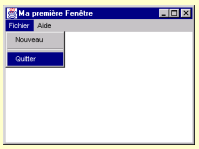
```
import javax.swing.*;

public class MyFrame extends  JFrame {

   final static int HAUTEUR = 200;
   final static int LARGEUR = 300;

   public MyFrame() {
      setTitle("Ma première Fenêtre");
      setSize(LARGEUR,HAUTEUR);

      JButton b =
         new JButton("Mon 1er composant");
      this.getContentPane().add(b);

      setVisible(true);
   }
```

Un composant ne peut être directement inséré dans une JFrame, mais à son « content pane » qui doit être récupéré au préalable

Fenêtres — Création d'un menu

- Seule une instance de la classe JFrame (ou JDialog) peut héberger un menu

```
import java.swing.*;

public class MyFrame extends JFrame {

   final static int HAUTEUR = 200;
   final static int LARGEUR = 300;

   public MyFrame() {
      setTitle("Ma première Fenêtre");
      setSize(LARGEUR,HAUTEUR);

      setJMenuBar(new MenuEditeur());

      setVisible(true);
   }


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



`setJMenuBar` prend en paramètre une instance de la classe `JMenuBar` :
- soit une instance directe de `JMenuBar` qui aura été modifiée grâce aux méthodes add(...)
- soit une instance d'une classe dérivée de `JMenuBar` comme dans le cas présent

Fenêtres — Création d'un menu

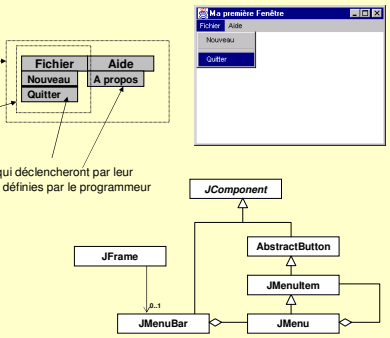
- Ajouter à l'application un menu rudimentaire

Classes utilisées :

JMenuBar : représente la barre de menu d'une fenêtre

JMenu : options visibles dans la barre de menu

JMenuItem :Items qui déclencheront par leur sélection des actions définies par le programmeur



Autres types de containers

- JFrame et JDialog containers racines (top level containers)
- Containers de type noeud

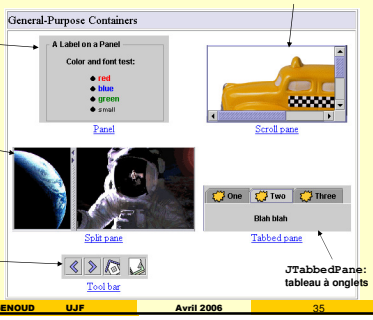
JScrollPane: lorsque le composant qu'il contient n'est pas affichable dans sa totalité

JPanel : Aspect réduit au minimum : rectangle invisible dont on peut fixer la couleur de fond Utilisé pour regrouper des composants dans une fenêtre

JSplitPane: permet de séparer son contenu en deux zones distinctes dont les surfaces respectives peuvent varier dynamiquement

JToolBar: barre d'outils (regroupe des boutons)

JTabbedPane: tableau à onglets



Fenêtres — Création d'un menu

Classes utilisées :

JMenuBar : représente la barre de menu d'une fenêtre

JMenu : options visibles dans la barre de menu

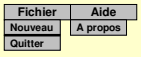
JMenuItem :Items qui déclencheront par leur sélection des actions définies par le programmeur

```
import java.swing.*;

public class MenuEditeur extends JMenuBar {
   JMenuItem quitter, nouveau, aPropos;

   public MenuEditeur() {

      JMenu menuFichier = new JMenu("Fichier");
      nouveau = new JMenuItem("Nouveau");
      quitter = new JMenuItem("Quitter");
      menuFichier.add(nouveau);
      menuFichier.addSeparator();
      menuFichier.add(quitter);

      JMenu menuAide = new JMenu("Aide");
      aPropos = new JMenuItem("A propos");
      menuAide.add(aPropos);

      this.add(menuFichier);
      this.add(menuAide);
   }
} // MenuEditeur
```

1) création des éléments du menu

2) association des objets JMenuItem dans un objet Menu

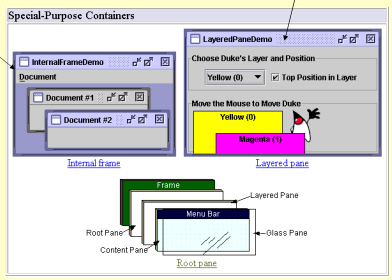
3) ajout des objets JMenu dans l'objet JMenuBar (this)



Encore d'autres containers

JLayeredPane: fournit une troisième dimension (z : profondeur) pour positionner les composants qu'il contient

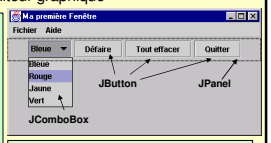
JDesktopPane: permet de définir des fenêtres internes dans une fenêtre



Imbrication des composants

- Pour structurer l'interface graphique utilisation de **`JPanels`**
- exemple : ajout d 'une barre d'outils à l'éditeur graphique

```
import java.awt.*;
import javax.swing.*;

public class BarreOutils extends JPanel {

  public BarreOutils() {

     String[] libelleCouleurs = {"Bleue",
                    "Rouge", "Jaune", "Vert"};

     Color[] couleurs = { Color.blue,
           Color.red, Color.yellow, Color.green };

     this.setBackground(Color.lightGray);

     JComboBox listeCouleurs = new JComboBox();
     for (int i = 0; i < libelleCouleurs.length; i++)
       listeCouleurs.addItem(libelleCouleurs[i]);

     this.add(listeCouleurs);
     this.add(new JButton("Défaire"));
     this.add(new JButton("Tout effacer"));
     this.add(new JButton("Quitter"));
   }

} // BarreOutils
```

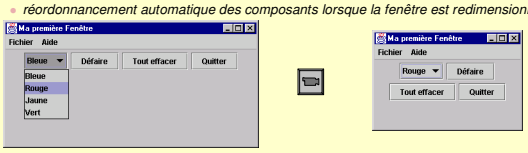



```
import javax.swing.*;

public class MyFrame extends  JFrame {

   final static int HAUTEUR = 200;
   final static int LARGEUR = 300;

   public MyFrame() {
      setTitle("Ma première Fenêtre");
      setSize(LARGEUR,HAUTEUR);
      setJMenuBar(new MenuEditeur());

      Container content=this.getContentPane();
      content.add(new BarreOutils());

      setVisible(true);
   }
}
```


Placement des composants

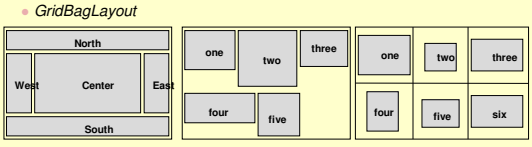
- Dans les exemples précédents les composants se sont placés automatiquement
  - *Pour la barre d'outils au centre du* **`JPanel`**
- Mécanismes par défaut
- Possibilité de les adapter aux besoins particulier de l'application

- **`LayoutManager`** objet associé à un **`Container`**
  - *se charge de gérer la disposition des composant appartenant à celui-ci*
    - *Par exemple Layout par défaut pour les* **`JPanels`**
      - *composants placés les uns après les autres dans leur ordre d'ajout*
      - *le tout doit être centré*
  - *réordonnancement automatique des composants lorsque la fenêtre est redimensionnée*



Placement des composants

- 5 gestionnaires de mise en forme implémentant l 'interface **`LayoutManager`** sont prédéfinis dans awt :
  - *BorderLayout*
  - *FlowLayout*
  - *GridLayout*
  - *CardLayout*
  - *GridBagLayout*



java.awt.BorderLayout — `add(Component,String)`

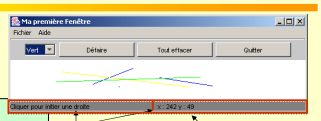
java.awt.FlowLayout — `add(Component)`

java.awt.GridLayout — `add(Component, index)`

- D'autres sont définis dans les swings (**`BoxLayout`**, **`SpringLayout`**….)
- Il est aussi possible de définir ses propres gestionnaires

Placement des composants : exemple GridLayout

- Définition de la barre d'état de l'application



JLabel — JPanel

GrigLayout : 1 ligne, 2 colonnes

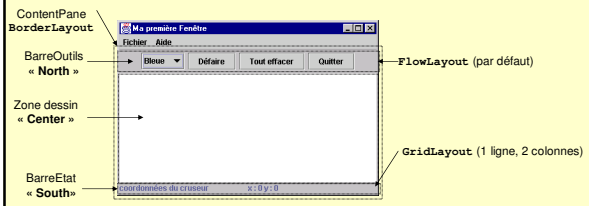
```
import javax.swing.*;
import java.awt.*;

public class BarreEtat extends JPanel {
   private JLabel coord, info;

   public BarreEtat() {
      this.setBackground(Color.darkGray);
      this.setLayout(new GridLayout(1,2));
      this.add(info = new JLabel());
      this.add(coord = new JLabel());
   }

   public void afficheCoord(int x, int y)
   {
      coord.setText("x : " + x + " y : " + y);
   }

   public void afficheInfo(String message)
   {
     info.setText(message);
   }
}
```

Associe un GridLayout en spécifiant le nombre de lignes et colonnes

Ajout des composants dans les différentes cellules définies par le layout

Placement des composants : exemple BorderLayout

- Ajout d'une zone de dessin et d'une barre d'état à la fenêtre de l'application



Associe un BorderLayout en spécifiant espacement entre les composants

Ajout des composants dans les différentes régions définies par le layout

```
public MyFrame () {
    BarreEtat barreEtat;
    setTitle("Ma première Fenêtre");
    setSize(LARGEUR,HAUTEUR);
    setJMenuBar(new MenuEditeur());
    this.getContentPane().setLayout(new BorderLayout(2,2));
    this.getContentPane().add(new BarreOutils(),"North");
    this.getContentPane().add(new ZoneGraphique(),"Center");
    this.getContentPane().add(barreEtat = new BarreEtat(),"South");
    barreEtat.afficheCoord(0,0);
    barreEtat.afficheInfo("coordonnées du cruseur");

    setVisible(true);
}
```


Comment dimensionner les composants ?

- Jusqu'à présent nous ne nous sommes pas préoccupés de la taille des composants
  - *Dans la barre d'outils les boutons n'ont pas tous la même taille (fixée automatiquement par le layout manager).*
- Possibilité d 'indiquer une taille avec les méthodes **`setPreferredSize`**, **`setMinimumSize`**.

```
public BarreOutils() {

     JComboBox listeCouleurs;
     String[] libelleCouleurs = {"Bleue", "Rouge", "Jaune", "Vert"};

     Color[] couleurs = { Color.blue, Color.red, Color.yellow, Color.green };

     this.setBackground(Color.lightGray);

     listeCouleurs = new JComboBox();
     for (int i = 0; i < libelleCouleurs.length; i++)
        listeCouleurs.addItem(libelleCouleurs[i]);

     this.add(listeCouleurs);
     JButton b;
     this.add(b= new JButton("Défaire"));
     b.setPreferredSize(new Dimension(130,25));
     this.add(b = new JButton("Tout effacer"));
     b.setPreferredSize(new Dimension(130,25));
     this.add(b = new JButton("Quitter"));
     b.setPreferredSize(new Dimension(130,25));
}
```

Indique les dimensions souhaitées, elles ne pourront pas toujours être respectées selon les contraintes et la politique de placement du `LayoutManager`

Comment dimensionner les composants ?

- Prise en compte des dimensions souhaitées selon le layout

Layout	*Hauteur*	*Largeur*
FlowLayout	**oui**	**oui**
BorderLayout (East, West)	**non**	**oui**
BorderLayout (North, South)	**oui**	**non**
BorderLayout(Center)	**non**	**non**
GridLayout	**non**	**non**

- Possibilité de se passer des services des LayoutManager et de placer les composants « à la main » en indiquant des positions et dimensions exprimées en pixels (`setBounds`, `setSize`)
  - *plus de souplesse*
  - *mais attention lorsque redimensionnement des conteneurs (attention à la portabilité de l'interface graphique)*

Gérer les événements

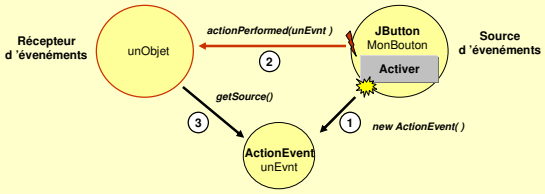
- Pour le moment les interactions de l'utilisateur avec les différents composants de l 'interface graphique ne provoquent aucune action
- Les applications comportant une interface graphique sont dirigées par les événements (event-driven)
  - *elles ne font rien jusqu'à ce que l'utilisateur bouge la souris, clique un bouton ou appuie sur une touche…*
- Le coeur de toute application comportant une interface graphique est le code de traitement des événements.
  - *un programme dirigé par les événements est structuré autours d'un modèle de traitement des événements. Bien comprendre ce modèle est important pour une bonne programmation.*
  - *dans JAVA , le modèle de gestion des événements par délégation*
    - *a radicalement (et heureusement) changé entre la version 1.0 et la version 1.1 (nous n'aborderons pas ici le modèle du JDK 1.0)*

Modèle événementiel du JDK 1.1

- Objectifs de conception
  - *simple et facile à apprendre*
  - *séparation nette entre code applicatif et code de l'interface utilisateur*
  - *faciliter l'écriture de code robuste pour la gestion des événements ("strong compile time checking")*
  - *suffisamment flexible pour autoriser selon les applications des modèles différents pour le flot et la propagation des événements*
  - *pour les concepteurs d'outils visuels permettre de découvrir à l'exécution*
    - *les événements qu'un composant peut générer*
    - *les événements qu'il peut observer*
  - *supporter une compatibilité binaire avec l'ancien modèle*
- Ce nouveau modèle est utilisé par AWT, Swing et sert également dans de nombreuses API Java (servlets, SAX, java BEANS …)

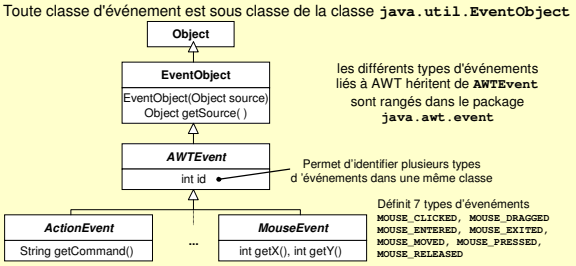
Modèle événementiel du JDK 1.1

- le modèle événementiel de JDK 1.1 se compose :
  - *d'objets sources d'événements*
  - *d'objets événements*
  - *d'objets récepteurs d'événements*
- Ces objets **interagissent de façon standard** en invoquant des méthodes pour permettre le déclenchement et la gestion des événements



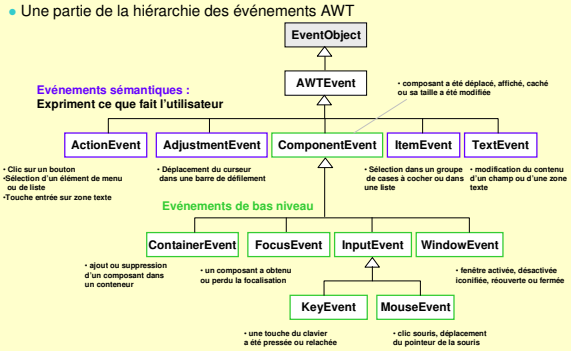
Objets Evénement

- Un objet événement encapsule une information spécifique à une instance d'événement
  - *exemple : un événement représentant un clic souris contient la position du pointeur souris*
- Les différents types d'événements sont représentés par des classes différentes
  - *ActionEvent, MouseEvent …*
- Toute classe d'événement est sous classe de la classe **`java.util.EventObject`**



La hiérarchie des événements

- Une partie de la hiérarchie des événements AWT



Récepteurs d'événements

- un récepteur d'événements est un objet qui doit être prévenu ("notified") par la source lorsque certains événements se produisent
- les notifications d'événements se font en invoquant des méthodes de l'objet à l'écoute, l'objet événement étant transmis en paramètre
- la source d'événements doit savoir quelle méthode du récepteur doit être appelée

→ pour chaque classe d'événements une **interface spécifique** définit les méthodes à appeler pour notifier les événements de cette classe

- *exemple : interface* ***ActionListener*** *pour les* ***ActionEvent***

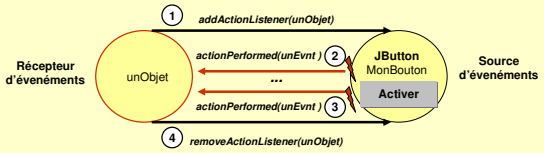
```
package java.awt.event;
import java.util.EventListener;
public interface ActionListener extends EventListener {
    /** Invoked when an action occurs.*/
    public void actionPerformed(ActionEvent e);
}
```

→ toute classe désirant recevoir des notifications d'un événement donné **devra implémenter cette interface**

- *un récepteur d'* ***ActionEvent*** *doit implémenter l'interface* ***ActionListener***

Sources d'événements

- événements générés par des sources d'événements (event sources)
- source d'événements, un objet capable de:
  - *déterminer quand un événement "intéressant" s'est produit*
  - *d'avertir (notify) des objets récepteurs (event listeners) de l'occurrence de cet événement*
- pour être averti des événements produits par une source un récepteur doit **se faire enregistrer auprès** de la source



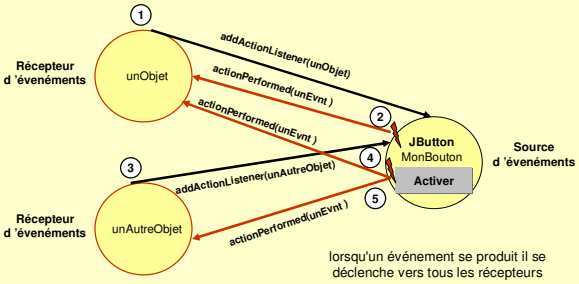
Interfaces d 'écoute d 'événements

- toutes les interfaces d'écoute d'événements héritent de **`java.util.EventListener`**
- par convention toutes les interfaces des récepteurs d'événements ont des noms de la forme ***`<EventType>`*****`Listener`**
  - *exemple : les événements de AWT et les interfaces correspondantes pour les récepteurs*

Classe d 'événement	Interface d 'écoute
`ActionEvent`	`ActionListener`
`AdjustmentEvent`	`AdjustmentListener`
`ComponentEvent`	`ComponentListener`
`ContainerEvent`	`ContainerListener`
`FocusEvent`	`FocusListener`
`ItemEvent`	`ItemListener`
`KeyEvent`	`KeyListener`
`MouseEvent`	`MouseListener`
	`MouseMotionListener`
`TextEvent`	`TextListener`
`WindowEvent`	`WindowListener`

Sources d 'événements

- Plusieurs récepteurs peuvent être à l'écoute d 'une même source d'événements



lorsqu'un événement se produit il se déclenche vers tous les récepteurs d'événements (multicast)

Interfaces d'écoute d'événements

- Une interface d'écoute d'événements peut contenir un nombre quelconque de méthodes, chacune correspondant à un événement différent

MouseEvent définit 7 types d'événements

`MOUSE_CLICKED`
`MOUSE_ENTERED`
`MOUSE_EXITED`
`MOUSE_PRESSED`
`MOUSE_RELEASED`

< interface > MouseListener

`void mouseClicked(MouseEvent)`
`void mouseEntered(MouseEvent)`
`void mouseExited(MouseEvent)`
`void mousePressed(MouseEvent)`
`void mouseReleased(MouseEvent)`

`MOUSE_DRAGGED`
`MOUSE_MOVED`

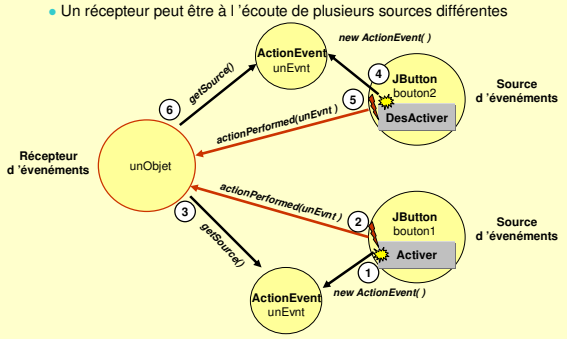
< interface > MouseMotionListener

`void mouseMoved(MouseEvent)`
`void mouseDragged(MouseEvent)`

- Les méthodes définies dans les interfaces d'écoute doivent se conformer au schéma standard : **`void <eventOccurenceMethodName>(<EventObjectType> evt);`**
- où
  - ***`eventOccurenceMethodName`*** *décrit clairement l'événement qui sera déclenché*
  - ***`EventObjectType`*** *est le type de l'événement déclenché et dérive obligatoirement de java.util.EventObject*

Sources d'événements

- Un récepteur peut être à l 'écoute de plusieurs sources différentes



Sources d'événements

- Une source d'événements pour une interface d'écoute d'événements propose une méthode d'enregistrement dont la signature a la forme suivante :

```
public void add<ListenerType>(<ListenerType> listener)
```

- A tout moment un objet récepteur d'événements peut annuler sa demande de notification
  - *Une source d'événements pour une interface d'écoute d'événements propose une méthode d'annulation de notification dont la signature a la forme suivante :*

```
public void remove<ListenerType>(<ListenerType> listener)
```


Exemple de gestion des événements

- Code de la classe lançant l'application

```
import javax.swing.*;

public class MyFrame extends JFrame {
   final static int HAUTEUR = 450;
   final static int LARGEUR = 750;
   public MyFrame7() {

      BarreEtat barreEtat = new BarreEtat();

      setTitle("Ma première Fenêtre");
      setSize(LARGEUR,HAUTEUR);
      setJMenuBar(new MenuEditeur());

      this.getContentPane().add(new BarreOutils(),"North");
      this.getContentPane().add(new ZoneDessin(barreEtat),"Center");
      this.getContentPane().add(barreEtat,"South");
      barreEtat.afficheInfo("coordonnées du cruseur");

      setVisible(true);
   }


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

} // MyFrame
```

L'objet zone de dessin a connaissance de l 'objet barre d 'état pour pouvoir agir sur lui lors de la réception des MouseEvent

Exemple de gestion des événements

- Afficher les coordonnées de la souris lorsqu'elle se déplace sur la zone de dessin.
- Type d 'événements : **`MouseEvent`** (**`MOUSE_MOVED`**, **`MOUSE_DRAGGED`**)
- Source d 'événements : la zone de dessin
- Interface d'écoute : **`MouseMotionListener`**
- Recepteur d 'événements ?
  - *Devra modifier l'affichage de la barre d 'état*
  - *Plusieurs solutions sont possibles. Prenons le cas où le récepteur est la zone graphique elle même*
    - *elle devra s'enregistrer auprès d'elle même pour être à l'écoute des* **`MouseMotionEvent`** *générés sur elle*
    - *elle devra avoir connaissance de l'objet gérant la barre d'état (la référence de celui-ci sera passée en paramètre du constructeur de la zône graphique)*

Pour conclure (momentanément)

- Pour gérer les événements il faut :
  - *identifier l'objet à l'origine des événements (souvent un composant)*
  - *identifier le type de l 'événement que l 'on veut intercepter.*
    - *Pour découvrir les types d'événements qu'est capable d'émettre un composant lister dans sa classe toutes les méthodes de type* **`addXXXListener`**
  - *créer une classe qui implémente l'interface associée à l'événement que l'on veut gérer*
    - *le choix de cette classe n'est pas neutre*
      - *celle du composant (ou du conteneur du composant) à l'origine de l 'événement ("facilité" d 'implémentation)*
      - *une classe indépendante qui détermine la frontière entre l 'interface graphique (émission des événements) et ce qui représente la logique de l 'application (traitement des événements). Une bonne séparation permet de faciliter l'évolution du logiciel.*
  - *Implémenter dans cette classe la (les) méthode(s) associées à l'événement. L'événement passé en paramètre contient des informations qui peuvent être utiles (position du curseur, état du clavier, objet source de l'événement).*

Exemple de gestion des événements

- Code de la classe représentant la zone de dessin

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class ZoneDessin extends JPanel implements MouseMotionListener {
   private BarreEtat be;

   public ZoneDessin(BarreEtat be) {
      setBackground(Color.white);
      setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
      this.be = be;
      this.addMouseMotionListener(this);
   }

   public void mouseMoved(MouseEvent e) {
      be.afficheCoord(e.getX(),e.getY());
   }

   public void mouseDragged(MouseEvent e) {
      be.afficheCoord(e.getX(),e.getY());
   }

} // ZoneGraphique
```

① L 'objet zone graphique va être à l 'écoute des événements MouseEvent de type MouseMotion

③ L 'objet zone graphique s 'enregistre lui même comme récepteur des événements MouseEvent de type MouseMotion qu'il est susceptible de générer

② L 'objet zone graphique utilise les informations contenues dans l 'objet MouseEvent qui lui est transmis pour mettre à jour la barre d'état.