Java

License Profesionnelle CISI 2009-2010

Cours 11 : Interface graphique-
GUI

Introduction

- Qu'est ce qu'une interface utilisateur ?
  - (Graphical User Interface GUI) ?
    - C'est une façade du programme qui le lie avec l'extérieur et qui va donc faciliter la tâche de l'utilisateur
    - La plupart des logiciels actuels disposent d'une interface car c'est un mode d'utilisation intuitif, relativement universel et efficace
    - La manière de réaliser une GUI en JAVA consiste à employer des composants graphiques

Introduction

- Composants graphiques
  - Ces composants graphiques sont accessibles grâce à différentes bibliothèques :
  - **SWING**
    - Bibliothèque proposant un grand nombre de classe GUI appelées « classe fondation Java » qui est la plus riches des bibliothèques de classes graphiques
    - On l'appelle par :

      import javax.swing.* ;
  - **AWT**
    - Abstract Windows Toolkit est une bibliothèque de classes graphiques Windows
    - On l'appelle par :

      import java.awt.* ;

Introduction

- Interface graphique
  - Dans une interface graphique, on imbrique généralement trois niveaux d'objets graphiques
    - Le conteneur principal :
      - permet d'encapsuler toutes les entités des deux autres niveaux (par exemple une fenêtre JFrame)
    - Un composant-conteneur intermédiaire :
      - aussi un conteneur qui se charge de regrouper en son sein des composants atomiques (par exemple un panneau JPanel)
    - Un composant atomique : appelé aussi widgets
      - Il s'agit d'éléments de base :
        - boutons, zones de saisie, liste de choix déroulant…
      - Ces composants sont ajoutés dans le conteneur courant en invoquant une méthode add

La fenêtre principale

- Le composant JFrame
  - La base de l'interface est de créer une fenêtre qui peut être agrandie, diminuée ou redimensionnée
  - Pour cela on utilise une classe de fenêtre, dérivée de la classe JFrame, définissant toutes les méthodes nécessaires à la construction d'une fenêtre de programme classique
  - Par exemple, avec

    JFrame fen = new JFrame

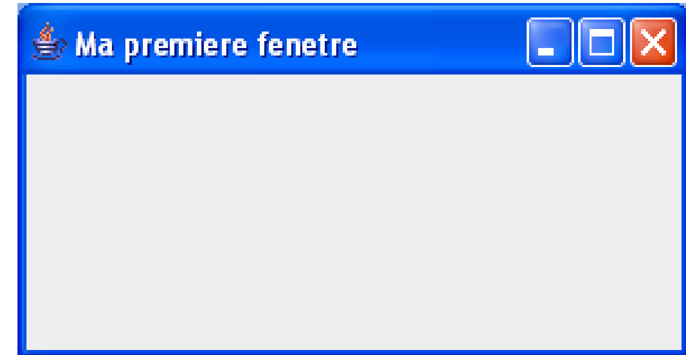
    - on crée un objet de type JFrame et on place sa référence dans fen

La fenêtre principale

Le composant JFrame

- Mais, on ne se limite pas à ça car rien n’apparaîtra à l’écran
- On utilisera alors :
  - fen.setVisible (true);
    - pour rendre visible la fenêtre (true pour visible et false pour caché)
  - fen.setSize(x, y);
    - pour fixer la taille de la fenêtre à son ouverture
      - x : taille horizontale en pixel
      - y : taille verticale en pixel
  - fen.setTitle("Ma première fenêtre");
    - pour donner un titre à la fenêtre

- Exemple : Premfen0.java




```
import javax.swing.* ;
public class Premfen0{
   public static void main (String args[]){
            JFrame fen = new JFrame() ;
            fen.setSize (300, 150) ;
            fen.setTitle ("Ma premiere fenetre") ;
            fen.setVisible (true) ;
   }
}
```

- On peut la retailler, la déplacer ou la réduire à une icône

- On peut aussi
  - en faire une classe et l'utiliser pour créer des objets :
    Premfen1.java

```
import javax.swing.* ;
class MaFenetre extends JFrame{
   public MaFenetre (){
   setTitle ("Ma premiere fenetre") ;
   setSize (300, 150) ;
  }
}
public class Premfen1{
   public static void main (String args[]){
            JFrame fen = new MaFenetre() ; // créer un cadre
                        fen.setVisible (true) ; // rendre visible la fenêtre
   }
}
```

La fenêtre principale

- Actions sur les caractéristiques d'une fenêtre
  - fen.setBounds(x,y,lg,ht)
    - place le coin supérieur de la fenêtre en x,y et donne à la fenêtre la taille lg*ht
  - fen.setDefaultCloseOperation(JFrame.xxx)
    - impose un comportement lors de la fermeture de la fenêtre
      - JFrame.EXIT_ON_CLOSE // ferme l'application à la fermeture de la fenêtre
      - JFrame.DO_NOTHING_ON_CLOSE // ne rien faire (inhiber l'icône détruire)

- Exemple : Premfen2.java

```
import javax.swing.* ;
class MaFenetre extends JFrame
{ public MaFenetre ()   // constructeur
  { setTitle ("Ma premiere fenetre") ;
    setBounds(500,40,300,150);
  }
}
public class Premfen2
{ public static void main (String args[])
  {
JFrame fen = new MaFenetre() ;
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //
    cette ligne termine le prog quand la fenêtre se ferme
    fen.setVisible (true) ;
  }
}
```

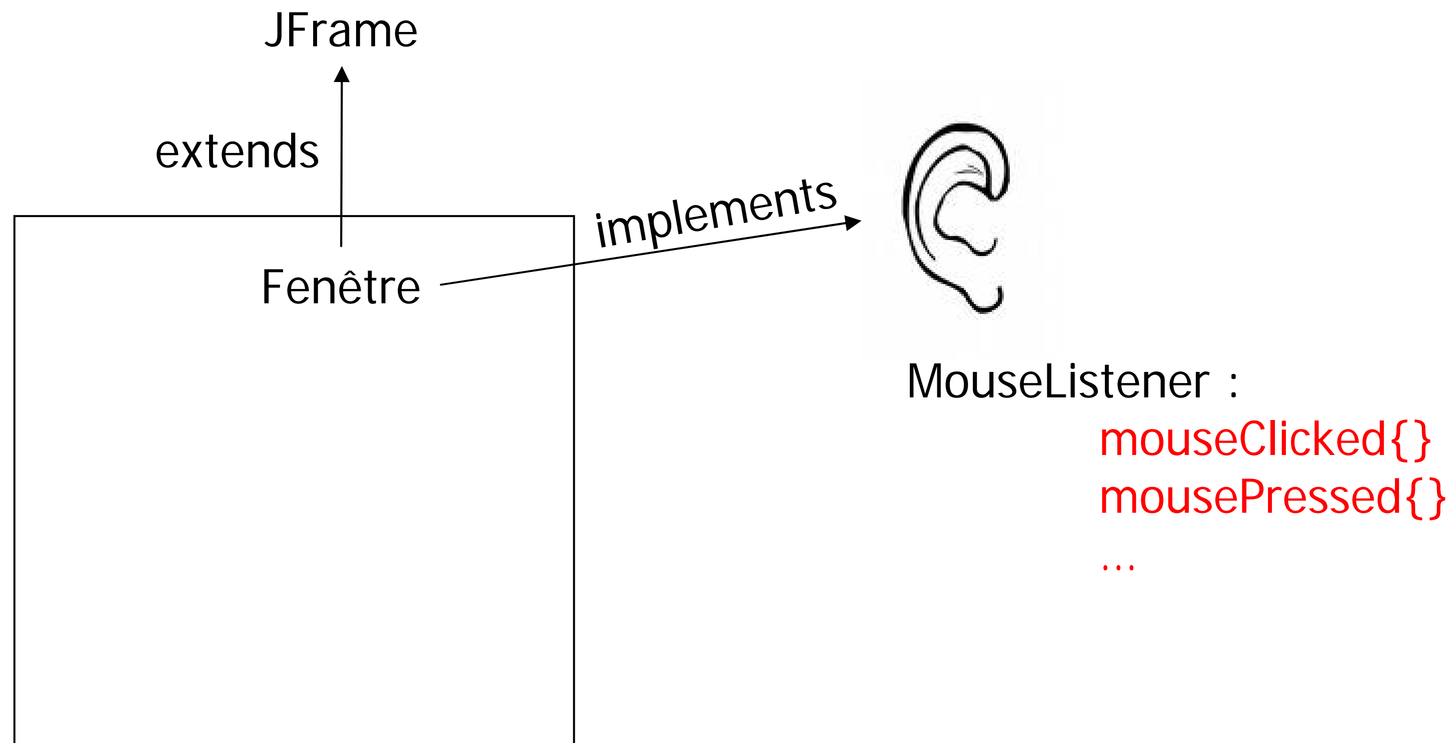
Gestion d'un clic dans la fenêtre

- Programmation événementielle
  - La plupart des événements sont créés par des composants qu'on aura introduits dans la fenêtre
    - Menus, boutons, boîtes de dialogues…
  - Avant de voir comment créer ces composants, nous allons voir comment traiter les événements

Gestion d'un clic dans la fenêtre

- Implémentation de l'interface MouseListener
  - En Java, tout événement possède une source
    - C'est l'objet qui l'a créé, ici la fenêtre principale
  - Pour traiter l'événement, on associe à la source un écouteur
    - un objet dont la classe implémente une interface particulière correspondant à une catégorie d'événements
  - Exemple
    - Il existe un écouteur souris qui correspond à la catégorie d'événements souris, un objet dont la classe est MouseListener
    - MouseListener comprend 5 méthodes à redéfinir correspondant à un événement particulier
      - mousePressed, mouseReleased, mouseEntered, mouseExited et mouseCliked

Gestion d'un clic dans la fenêtre



Gestion d'un clic dans la fenêtre

- L'instanciation de MouseListener consiste à définir une classe qui l'«implémente» :

```
Class EcouteurSouris implements MouseListener
{
    public void mouseClicked(MouseEvent ev) {}
    public void mousePressed (MouseEvent ev) {}
    public void mouseReleased(MouseEvent ev) {}
    public void mouseEntered (MouseEvent ev) {}
    public void mouseExited  (MouseEvent ev) {}
    // autres méthodes et champs de la classe
}
```

- L'implémentation consiste à :
  - ☛ redéfinir toutes ces méthodes
  - ☛ ou ne rien faire et les laisser vides

Exemple : Clic1.java

```
import javax.swing.* ;
import java.awt.event.* ;   // pour
    MouseEvent et MouseListener

class MaFenetre extends JFrame
    implements MouseListener{
    public MaFenetre (){
      setTitle ("Gestion de clics") ;
      setBounds (10, 20, 300, 200) ;
      addMouseListener (this) ;
    }
    public void  mouseClicked (
      MouseEvent ev)
   {
        System.out.println ("clic
        dans fenetre") ;
   }

public void mousePressed (MouseEvent
      ev) {
      System.out.println ("mousePressed");
}

public void mouseReleased(MouseEvent
      ev) {
}

public void mouseEntered (MouseEvent
      ev) {
      System.out.println ("mouseEntered") ;
}

public void mouseExited  (MouseEvent
      ev) {}
}
}

public class Clic1{
      public static void main (String args[]){
        MaFenetre fen = new MaFenetre() ;
        fen.setVisible(true) ;
      }
}
```

Gestion d'un clic dans la fenêtre

- Commentaires
  - Noter la présence de : java.awt.event.*;
    - La gestion des événements fait appel au paquetage java.awt.event
  - Les méthodes telles que mouseClicked doivent être déclarées publiques car une classe ne peut pas restreindre les droits d'accès d'une méthode qu'elle implémente
  - L'absence dans MaFenetre de addMouseListener(this) ne conduit pas à une erreur, mais aucune réponse ne sera apportée au clic

Gestion d'un clic dans la fenêtre

- Utilisation de l'information associée à un événement
  - On peut utiliser l'objet, MouseEvent ev, passé en paramètre des méthodes de gestion de l'événement pour traiter l'information contenue
  - Exemple : récupérer les coordonnées de la souris : Clic2.java

```
…
public void mouseClicked(MouseEvent ev)
  { int x = ev.getX() ;
    int y = ev.getY() ;
    System.out.println ("clic au point de coordonnées " + x + ", " + y
   ) ;
  }
```

Gestion d’un clic dans la fenêtre

- La notion d’adaptateur
  - Pour éviter de redéfinir toutes les méthodes, Java propose une classe MouseAdapter qui implémente toutes les méthodes avec un corps vide

```
Classe MouseAdapter implements MouseListener
{
    public void mouseClicked (mouseEvent ev){}
    public void mousePressed (mouseEvent ev){}
    public void mouseReleased (mouseEvent ev){}
    …
}
```

  - Dans ces conditions, on peut facilement définir une classe écouteur des événements souris ne comportant qu’une méthode, en procédant ainsi :

Gestion d'un clic dans la fenêtre

```
Classe EcouteurSouris extends MouseAdapter
{
   public void mouseClicked (mouseEvent ev){ //ici, on ne
   redéfinit que la méthode qui nous intéresse
            …
   }
}
```

- En résumé, voici comment il faut gérer les événements : Clic3.java

```
import javax.swing.* ;
import java.awt.event.* ;
class MaFenetre extends JFrame
{ MaFenetre ()    // constructeur
  { setTitle ("Gestion de clics") ;
    setBounds (10, 20, 300, 200) ;
    addMouseListener ( new MouseAdapter()
     { public void mouseClicked(MouseEvent ev)
       { int x = ev.getX() ;
         int y = ev.getY() ;
         System.out.println ("clic au point de coordonnees " + x + ", " + y ) ;
       }
     } ) ;
  }
}
public class Clic3
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible(true) ;
  }
}
```

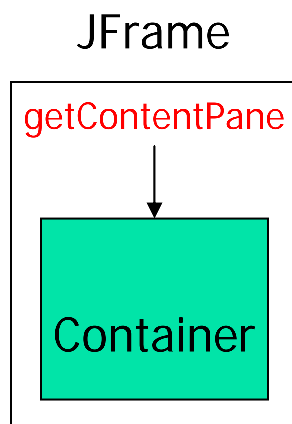
La fenêtre principale

- Le composant Container
  - Pour pouvoir placer des composants dans une fenêtre, il faut tout d'abord créer le conteneur qui les accueillera, puis placer ce conteneur dans la fenêtre
  - Pour accéder à la classe container, il faut importer la bibliothèque AWT
  - Pour cela, la classe JFrame dispose d'une méthode :

  JFrame.getContentPane () ;

  - Exemple :

  Container monConteneur = JFrame.getContentPane () ;



Premier composant : un bouton

- Création d'un bouton et ajout dans la fenêtre
  - On crée le bouton en utilisant la classe JButton

```
JButton monBouton;
…
monBouton = new JButton("Essai");
```

  - Ajout dans la fenêtre :
    - On nomme le contenu (de type Container), par ex :

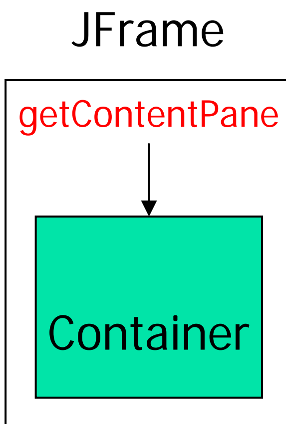
      ```
      Container c = getContentPane();
      ```

    - On ajoute le bouton à ce contenu

      ```
      c.add(MonBouton)
      ```

    - On aurait pu aussi faire directement :

      ```
      getContentPane().add(MonBouton)
      ```



Premier composant : un bouton

- Ajout dans la fenêtre :
  - On nomme ce contenu (de type Container), par ex :

    ```
    Container c = getContentPane();
    ```

  - On ajoute le bouton à ce contenu

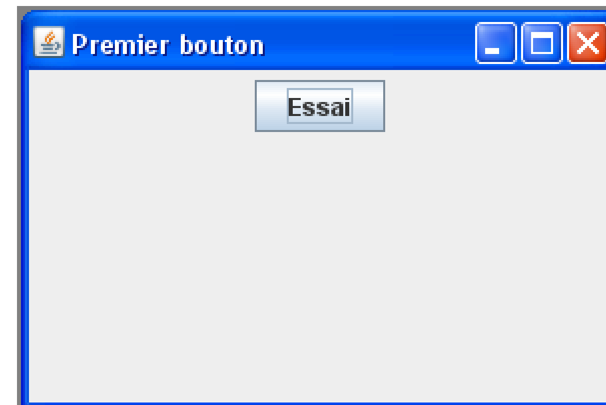
    ```
    c.add(MonBouton)
    ```

  - On aurait pu aussi faire directement :

    ```
    getContentPane().add(MonBouton)
    ```

Exemple complet: Bouton1.java

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

class Fen1Bouton extends JFrame
{ public Fen1Bouton ()
  {
    setTitle ("Premier bouton") ;
    setSize (300, 200) ;
    monBouton =new JButton ("Essai");
    getContentPane().add(monBouton);
  }
    private JButton monBouton ;
 }
public class Bouton1
{ public static void main (String args[])
  { Fen1Bouton fen = new Fen1Bouton() ;
  fen.setVisible(true);
  }
}
```



Premier composant : un bouton

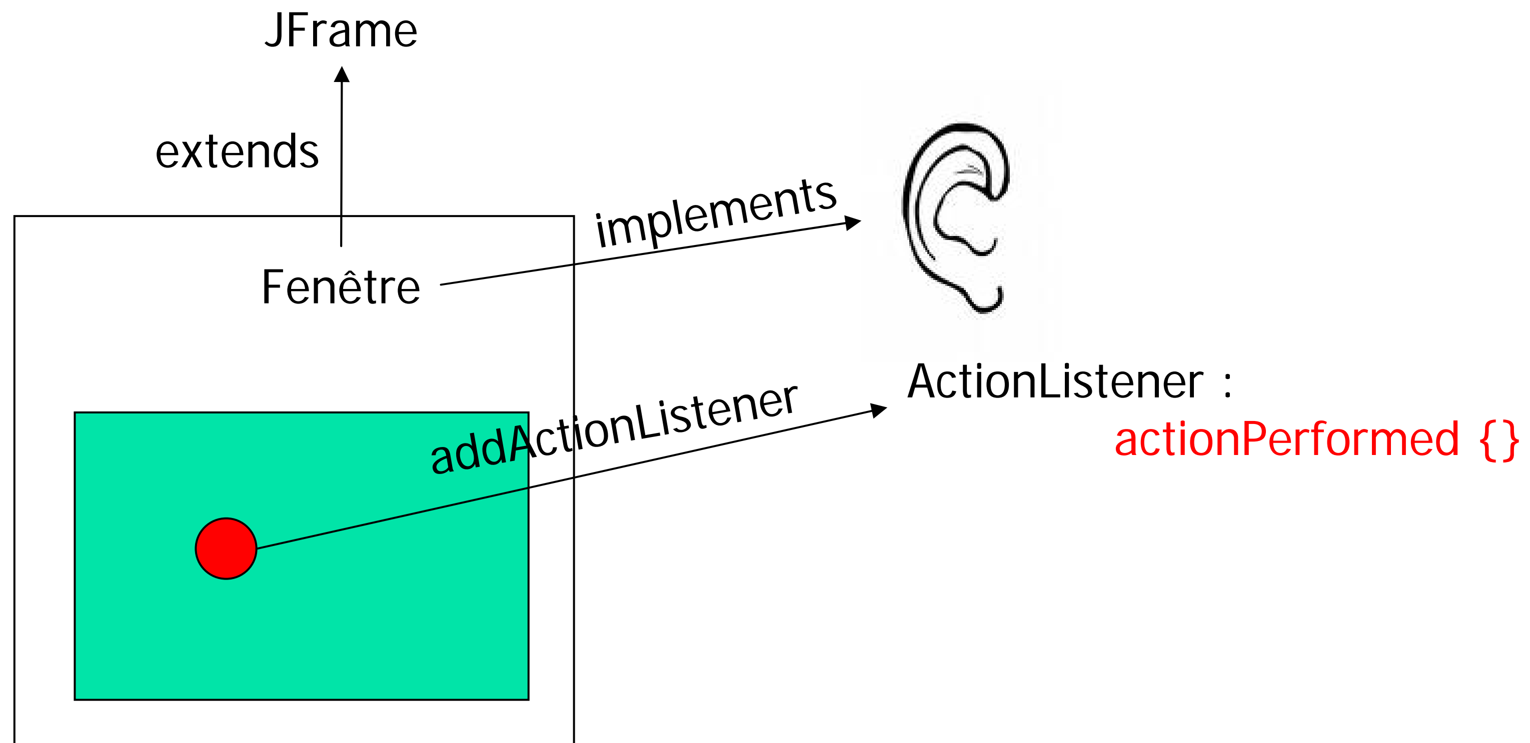
- Gestion du bouton avec un écouteur
  - Créer un écouteur : objet d'une classe qui implémente l'interface ActionListener
    - Cette classe ne contient qu'une méthode : actionPerformed
  - Associer cet écouteur au bouton par la méthode addActionListener

Premier composant : un bouton

- Exemple : Bouton2.java

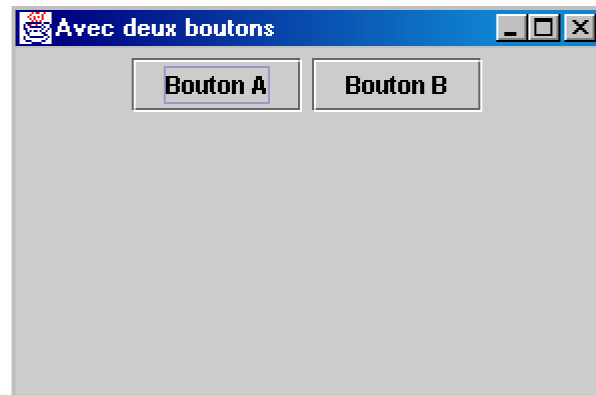
```
class Fen1Bouton extends JFrame implements
    ActionListener
{ public Fen1Bouton ()
  { …
    private JButton monBouton ;
    getContentPane().add(monBouton) ;
    monBouton.addActionListener(this);}

    public void actionPerformed (ActionEvent ev)
    { System.out.println ("action sur bouton ESSAI") ;}
    }
```

Gestion d'un clic dans la fenêtre



Gestion de plusieurs composants

- Entrée de plusieurs boutons
  - Déclaration des boutons
  - Ajout à la même fenêtre par add
  - Exemple : Boutons1.java



Gestion de plusieurs composants

- La fenêtre écoute les boutons
  - On peut faire de la fenêtre l'objet écouteur de tous les boutons :
  - Dans ce cas :
    - Prévoir exactement la même réponse
    - Prévoir une réponse dépendant du bouton concerné, ce qui nécessite de l'identifier
- Tous les boutons déclenchent la même réponse : Boutons1

```
…
monBouton1.addActionListener(this);  // la fenêtre écoute monBouton1
monBouton2.addActionListener(this);  // la fenêtre écoute monBouton2
}
public void actionPerformed (ActionEvent ev)   // gestion commune
{ System.out.println ("action sur un bouton") ;  // de tous les boutons
}
```

Gestion de plusieurs composants

- Une seule méthode pour les deux boutons
  - Mais action différente : grâce à la méthode getSource
    - Fournit une référence (de type Object) sur l’objet ayant déclenché l’événement
    - Exemple : Boutons2.java

```
public void actionPerformed (ActionEvent ev)
 { if (ev.getSource() == monBouton1)
     System.out.println ("action sur bouton numéro 1") ;
   if (ev.getSource() == monBouton2)
     System.out.println ("action sur bouton numéro 2") ;
 }
```

Gestion de plusieurs composants

- Classe écouteur différente de la fenêtre
  - Dans les exemples précédents, on a fait de la fenêtre l'objet écouteur des boutons
  - Voyons des situations où l'écouteur est différent de la fenêtre
    - Deux possibilités :
      - Une classe écouteur par bouton
      - Une seule classe écouteur pour tous les boutons

Classe écouteur différente de la fenêtre : Boutons4.java

```
import javax.swing.* ;
import java.awt.* ;
import java.awt.event.* ;
class Fen2Boutons extends JFrame
{
   private JButton monBouton1, monBouton2 ;
   public Fen2Boutons (){
         setTitle ("Avec deux boutons") ;
         setSize (300, 200) ;
         monBouton1 = new JButton ("Bouton A") ;
         monBouton2 = new JButton ("Bouton B") ;
         Container contenu = getContentPane() ;
         contenu.add(monBouton1) ;
         contenu.add(monBouton2) ;
         EcouteBouton1 ecout1 = new EcouteBouton1() ;
         EcouteBouton2 ecout2 = new EcouteBouton2() ;
         monBouton1.addActionListener(ecout1);
         monBouton2.addActionListener(ecout2);
   }
}
```

```
class EcouteBouton1 implements ActionListener{
   public void actionPerformed (ActionEvent ev){
            System.out.println ("action sur bouton 1") ; }
}
class EcouteBouton2 implements ActionListener{
   public void actionPerformed (ActionEvent ev){
            System.out.println ("action sur bouton 2") ;
  }
}
public class Boutons4{
   public static void main (String args[]){
            Fen2Boutons fen = new Fen2Boutons() ;
            fen.setVisible(true) ;
   }
}
```

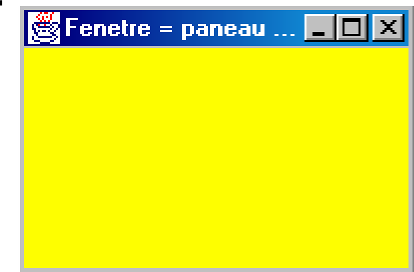
Premier dessin

- Remarques :
  - Java permet de dessiner sur n’importe quel composant en utilisant des méthodes de dessin
  - Cependant, si on utilise directement ces méthodes, le dessin disparaît au premier changement de taille…
  - Pour obtenir une permanence complète des dessins,
    - Il est nécessaire de placer les instructions de dessin dans une méthode particulière du composant concerné, nommée paintComponent
    - Cette méthode est automatiquement appelée par Java chaque fois que le composant a besoin d’être dessiné ou redessiné
  - On ne dessine pas dans la fenêtre, mais dans un panneau

Premier dessin

- Création d'un panneau : Paneau.java
  - Objet de la classe JPanel
    - Sorte de sous-fenêtre sans titre ni bordure
    - Rectangle qui reste invisible tant qu'on lui a pas donné de couleur
  - Contrairement à la fenêtre :
    - Un panneau ne peut pas exister de manière autonome
  - Voici comment le créer et l'ajouter à la fenêtre :

```
{ MaFenetre ()
  { …
    paneau = new JPanel() ;
    getContentPane().add(paneau) ;
        // on le définit ici car il est associé à la fenêtre
  }
  private JPanel paneau ;}
```



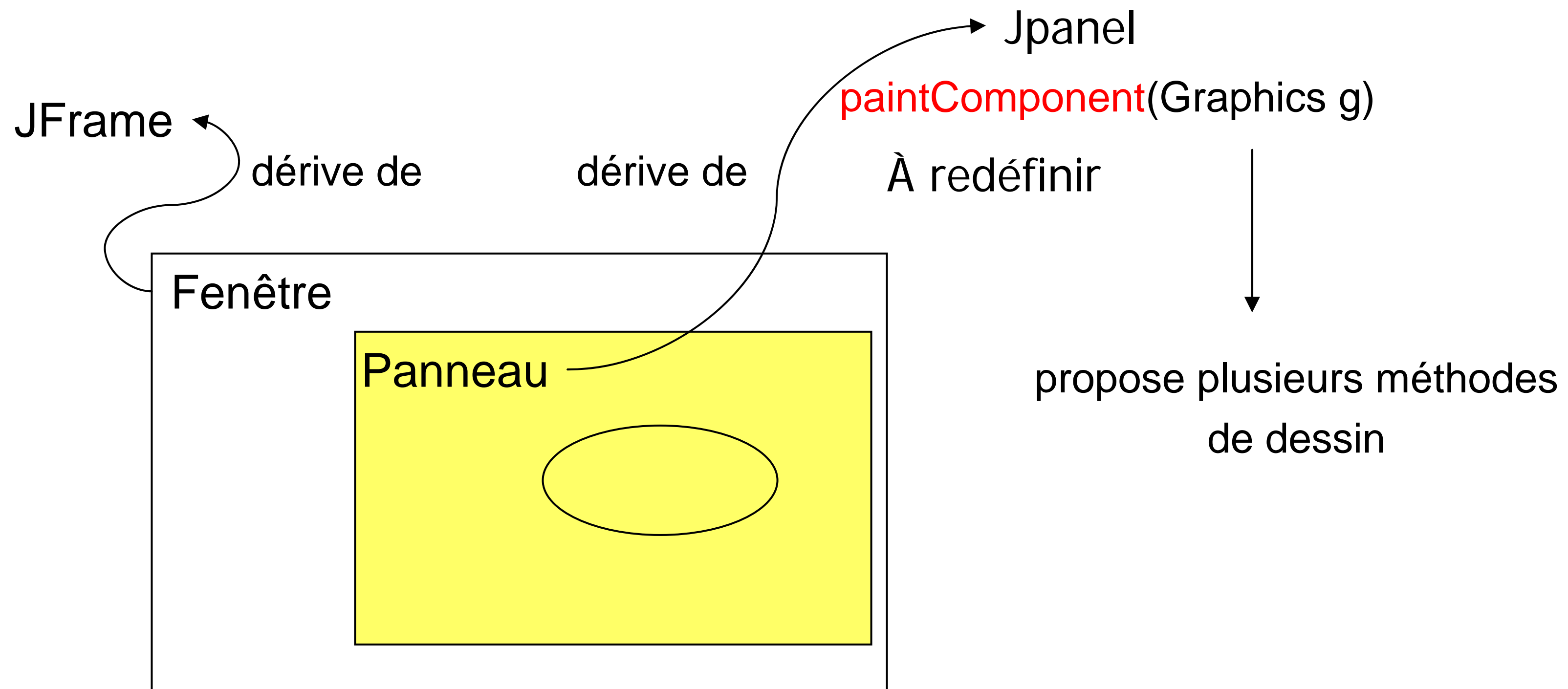
  - On peut le colorier : setBackground(Color.yellow);

Premier dessin

- Dessin dans le panneau : PremDes.java
  - Comme déjà dit auparavant :
    - Pour obtenir un dessin permanent dans un composant, il faut redéfinir sa méthode paintComponent dont on sait qu'elle sera appelée chaque fois que le composant aura besoin d'être redessiné
  - Comme il s'agit de redéfinir une méthode de la classe JPanel
    - Il faut faire obligatoirement que notre panneau soit un objet dérivé de la classe JPanel
  - paintComponent possède cet entête:

    Void paintComponent (Graphics g)
    - Graphics :
      - Classe qui contient toutes méthodes pour dessiner, colorier
      - Gère également des paramètres courants comme : couleur de fond, couleur de trait, style de trait, police, …

Premier dessin

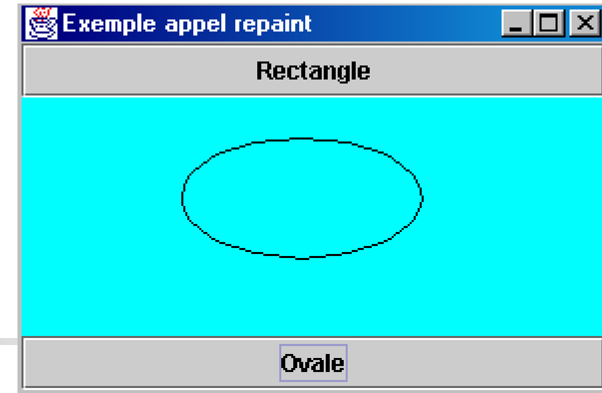


Premier dessin

- Dessin dans le panneau : PremDes.java

```
     …
     pan = new Paneau() ;
    getContentPane().add(pan) ;
    pan.setBackground(Color.yellow) ;  // couleur de fond = jaune
  }
  private JPanel pan ;
}
class Paneau extends JPanel
{ public void paintComponent(Graphics g)
  {  super.paintComponent(g) ; // initialise l'objet g dans sa classe
     g.drawLine (15, 10, 100, 50) ;
     g.drawRect(30, 30, 100, 50) ;
     g.drawOval(40, 10, 100, 50) ;
  }
}…
```

Premier dessin



- Forcer le dessin : Repaint.java
  - On cherche à modifier le dessin en cours d'exécution
  - On dispose de deux boutons :
    - Rectangle, Ovale
  - Le panneau doit s'effacer à chaque nouveau clic
    - On utilise dans ce cas la méthode repaint qui va faire dessiner tout le panneau dans la couleur de son "background" puis faire appel à la méthode paint du panneau (une instance de Jpanel) pour effacer le dessin
  - On utilise le gestionnaire de position par défaut « BorderLayout »
    - Dispose les éléments au centre et aux quatre bords
    - Fournir pour cela à la méthode add un argument : "North", "South", …

```
import javax.swing.* ;
import java.awt.* ;
import java.awt.event.* ;
class MaFenetre extends JFrame
    implements ActionListener
{ MaFenetre ()
  { setTitle ("Exemple appel repaint") ;
    setSize (300, 200) ;
    Container contenu =
      getContentPane() ;
       // creation paneau pour le dessin
    pan = new Paneau() ;
    pan.setBackground (Color.cyan) ;
    contenu.add(pan) ;
       // creation bouton "rectangle"
    rectangle = new JButton
      ("Rectangle") ;
    contenu.add(rectangle, "North") ;
    rectangle.addActionListener (this) ;
       // création bouton "ovale"
    ovale = new JButton ("Ovale") ;
    contenu.add(ovale, "South") ;
    ovale.addActionListener (this) ;
  }
public void actionPerformed (ActionEvent ev)
  { if (ev.getSource() == rectangle) pan.setRectangle() ;
    if (ev.getSource() == ovale) pan.setOvale() ;
    pan.repaint() ;   // pour forcer la peinture du paneau des
        maintenant
  }
  private Paneau pan ;
  private JButton rectangle, ovale ;
}
class Paneau extends JPanel
{ public void paintComponent(Graphics g)
  { super.paintComponent(g) ;
    if (ovale)g.drawOval (80, 20, 120, 60) ;
    if (rectangle) g.drawRect (80, 20, 120, 60) ;
  }
  public void setRectangle() {rectangle = true ;
      ovale = false ; }
  public void setOvale()     {rectangle = false ;
      ovale = true ;  }
  private boolean rectangle = false,
      ovale = false ;
}
public class Repaint
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible(true) ;
  }
}
```

Les contrôles usuels

- Nous avons vu les conteneurs :
  - Jframe et Jpanel
- Nous allons voir les composants atomiques, nommés aussi contrôles :
  - Cases à cocher
  - Boutons radio
  - Étiquettes
  - Champs de texte
  - Boîtes de listes
  - Boîtes combinées

Les contrôles usuels

- Case à cocher : Cases1.java
  - Permet de faire un choix de type : oui/non
  - Création :

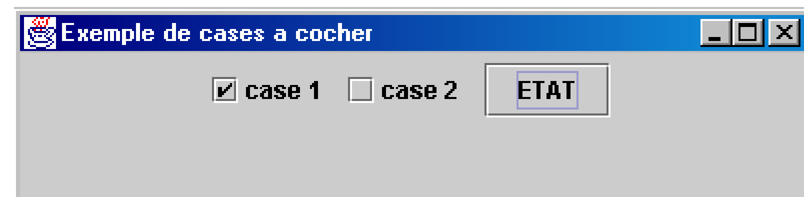
    JChekBox coche = new JChekBox ("CASE");
    - Par défaut la case est créée dans l'état non coché :
    - On peut lui imposer l'état coché en écrivant :

    JChekBox coche = new JChekBox ("CASE",true);
  - Ajout à la fenêtre

    getContentPane().add(coche) ;
  - Connaître l'état d'une case

    If (coche.isSelected()) …
  - Forcer l'état d'une case

    coche.isSelected(true);



Les contrôles usuels

Bouton radio

- Permet de faire un choix de type : oui/non, mais sa vocation est de faire partie d'un groupe de boutons dans lequel une seule option peut être sélectionnée à la fois :
- Création :

```
JRadioButton bRouge =  new JRadioButton ("Rouge");
JRadioButton bVert =  new JRadioButton ("Vert");
```

  - Par défaut la case est créée dans l'état non coché
  - On peut lui imposer l'état coché en écrivant :

```
JChekBox coche = new JChekBox ("CASE",true);
```

- Ajout à la fenêtre
  - Si on se contente de les ajouter par add à un conteneur, on obtient quelque chose d'équivalent aux cases à cocher

Les contrôles usuels

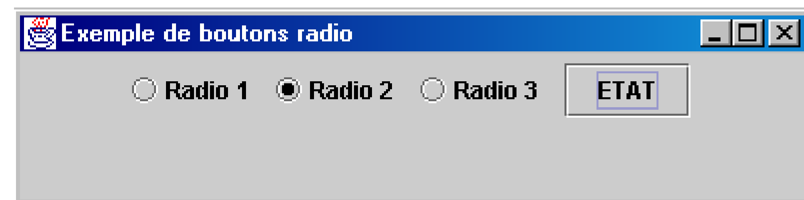
Boutons radio : Radios1.java

- Groupement
  - Pour obtenir la désactivation automatique d'autres boutons radio d'un même groupe, il faut de plus :
  - Créer un objet de type ButtonGroup : ex :

    ButtonGroup groupe = new ButtonGroup() ;
  - Associer chacun des boutons voulus à ce groupe à l'aide de la méthode add :

    groupe.add(bRouge);

    groupe.add(bVert);



- Exploitation
  - Comme les cases à cocher

Les contrôles usuels

- Les étiquettes : Label1.java
  - Permet d'afficher un texte dans un conteneur
  - Constructeur :

    JLabel texte = new Jlabel("texte initial")
  - Modification : possible par :

    texte.setText("Nouveau texte")

Les contrôles usuels

- Les champs de texte : Text1.java
  - Zone rectangulaire (avec bordure) dans laquelle on peut entrer ou modifier un texte
  - Construction :

    JTextField entree1, entree2;

    entree1= new JTextField(20) // champ de taille 20, initialement vide

    entree2= new JTextField("texte initial",20) // champ de taille 15, contenant au départ le texte "texte initial"

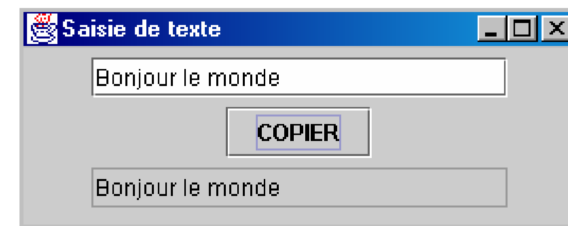
  - Exploitation : getText();

    String ch = entree1.getText()

  - Modification ou non :

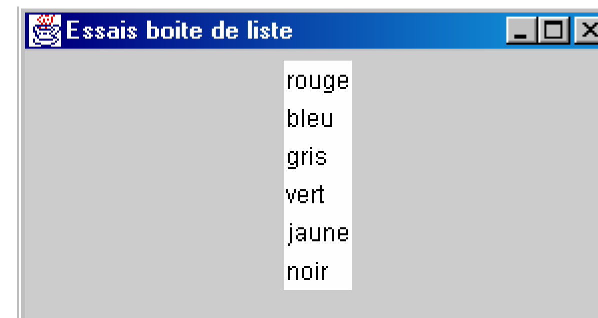
    entree1.setEditable (false) // champ entree1 n'est plus modifiable

    entree1.setEditable (true) // champ entree1 à nouveau modifiable



Les contrôles usuels

- Les boîtes de liste : Liste0.java



  - Création :
    - En fournissant un tableau de chaînes à son constructeur
    - Exemple :

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

  - Sélection
    - Initialement, aucune valeur n'est sélectionnée
    - On peut forcer la sélection d'un élément par :

```
Liste.setSelectedIndex(2); // sélection préalable de l'élément de rang 2
```

Les contrôles usuels

- Les boîtes de liste : Liste1.java
  - Barre de défilement :
    - Pour ajouter une barre de défilement :

      JScrollPane defil = new JScrollPane(liste)
    - Il faut ajouter au conteneur non plus la liste elle même mais le panneau de défilement, par ex au conteneur de type JFrame:

      getContentPane().add(defil)
    - Par défaut, la liste affichera huit valeurs. On peut modifier ce nombre par :

      Liste.setVisibleRowCount(3)

Les contrôles usuels

Les boîtes de liste

- Accès aux informations sélectionnées
  - Mettre les éléments dans une liste de chaînes

```
String ch = (String) liste.getSelectedValue();
```

  - Accès aux éléments

```
Object[] valeurs = liste.getSelectedValues();
For (int i=0;i<valeurs.length;i++)
System.out.println((String) valeurs[i]);
```

  - Accès à la position dans la liste

```
Int getSelectedIndex(); // position de la première valeur sélectionnée
Int [] getSelectedIndices(); // tableau donnant la position de tous les éléments
    sélectionnés
```

Les contrôles usuels

Les boîtes de liste

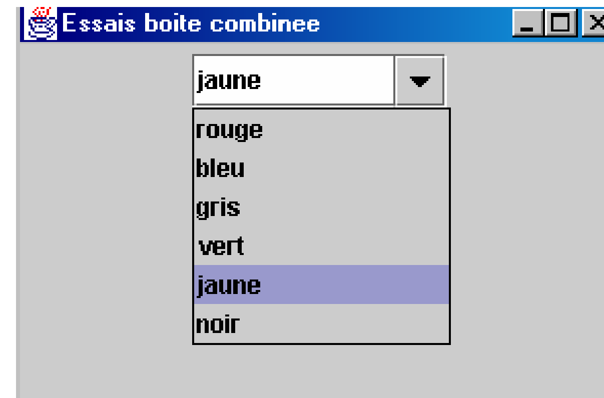
- Gestion des événements : Liste.java
  - Événements gérés par :
    ListSelectionListener
  - Ajout se fait par :
    liste.addListSelectionListener (this) ;
  - Gestion se fait par :

```
public void valueChanged (ListSelectionEvent e)
  { if (!e.getValueIsAdjusting())
    { System.out.println ("**Action Liste - valeurs selectionnees :") ;
      Object[] valeurs = liste.getSelectedValues() ;
      for (int i = 0 ; i<valeurs.length ; i++)
        System.out.println ((String) valeurs[i]) ;
    }
```

Les contrôles usuels

- La boîte combo : Combo.java
  - Rôle :
    - Associe un champ de texte et une boîte de liste à sélection simple
  - Tant que le composant n'est pas sélectionné, seul le champ de texte s'affiche :

Avant sélection



Après sélection

Les contrôles usuels

- Construction

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

- Pour rendre une boîte combo éditable :

```
combo.setEditable (true) ;
```

- Pour modifier le nombre d'éléments visibles (limité à 8) :

```
combo.setMaximumRowCount (nombre-souhaité) ;
```

- Pour forcer la sélection d'un élément :

```
Liste.setSelectedIndex(2);
```

Les contrôles usuels

- Exploitation
  - Différente de la liste
  - Accès à l'information sélectionnée ou saisie : getSelectedItem()
    - Fournit la valeur sélectionnée

      Object valeur = combo.getSelectedItem()
    - Elle fournit également le rang de la valeur sélectionnée

      int rang = combo.getSelectedIndex() ;
- Gestion des événements
  - Les événements sont de deux types : Action et Item
  - On les traite par les écouteurs ActionListener et ItemListener.
  - Pour le dernier, on utilise la méthode :

    Public void itemStateChanged (ItemEvent e)

Les contrôles usuels

La boîte combo : Combo2.java

- Évolution dynamique de la liste :
  - Ajout d’un item par addItem
    - Combo.addItem ("orange"); //permet d’ajouter "orange" en fin de liste
  - Insertion par insertItemAt
    - Combo. insertItemAt ("rose",2); // ajouter "rose" en position 2
  - Suppression par removeItem
    - Combo. removeItem ("gris"); // supprime "gris" de la liste

Exercice

- Énoncé : Formes.java
  - Réalisez une petite interface permettant de choisir des formes à dessiner dans une fenêtre, leurs dimensions et la couleur de fond
  - Pour ne pas charger le code, les formes proposées se limitent à l'ovale et au rectangle; indiquer le choix par des cases à cocher
  - Les dimensions sont choisies dans deux champs de texte et sont communes aux différentes formes

Les gestionnaires de mise en forme

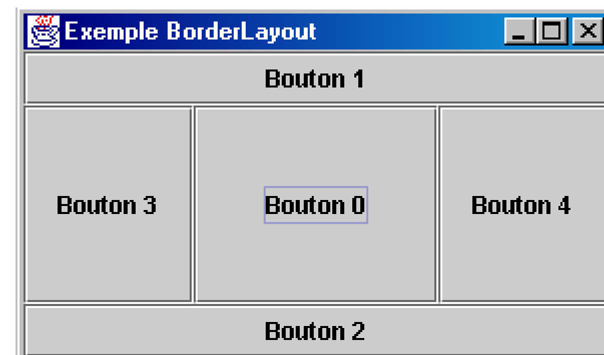
Fonctionnement

- Pour chaque conteneur (fenêtre, panneau, etc.) Java permet de choisir un gestionnaire de mise en forme responsable de la disposition des composants, parmi :
  - BorderLayout : suivant l'un des quatre bords et le centre
  - FlowLayout : suivant une même ligne
  - CardLayout : suivant une pile
  - GridLayout : suivant une grille
  - BoxLayout : suivant une ligne ou une colonne
  - GridBagLayout : suivant une grille, mais ceux-ci peuvent occuper plusieurs cellules, avec des contraintes sur les cellules

Les gestionnaires de mise en forme

- BorderLayout : Layout1.java
  - Dispose les composants suivant l'un des quatre bords du conteneur, ou au centre
  - Pour préciser l'emplacement :
    - on indique le bord : Nord, Sud, Ouest, Est au paramètre BorderLayout de la méthode add
    - Si on indique rien, le composant est placé au centre
  - Exemple :

```
conteneur.add(bouton1) ; // au centre par defaut
conteneur.add(bouton2, BorderLayout.NORTH) ;
conteneur.add(bouton3, BorderLayout.SOUTH) ;
conteneur.add(bouton4, BorderLayout.WEST) ;
conteneur.add(bouton5, BorderLayout.EAST) ;
```



Les gestionnaires de mise en forme

- BorderLayout
  - Espacement des composants :
    - Par défaut, ils sont espacés de 5 pixels en hauteur et en largeur
    - Pour les changer, le faire à la construction du conteneur :

      ```
      conteneur.setLayout (new BorderLayout (espace-hor, espace-vert))
      ```

    - On peut le faire également après :

      ```
      BorderLayout g = new BorderLayout();
      …
      g.setHgap(15);
      g.setVgap(8);
      …
      conteneur.setLayout(g)
      ```

Les gestionnaires de mise en forme

- FlowLayout
  - Dispose les composants les uns à la suite des autres, sur une même ligne, en passant à la ligne suivante si plus de place
  - Contrairement à BorderLayout, la taille des composants est respectée
  - On peut imposer une taille à un composant par la méthode setPreferredSize
  - Lors de la construction, on peut ajuster une ligne par rapport aux bords :

```
contenu.setLayout (new FlowLayout(FlowLayout.CENTER) ;
        //Les composants seront centrés sur les différentes lignes
```

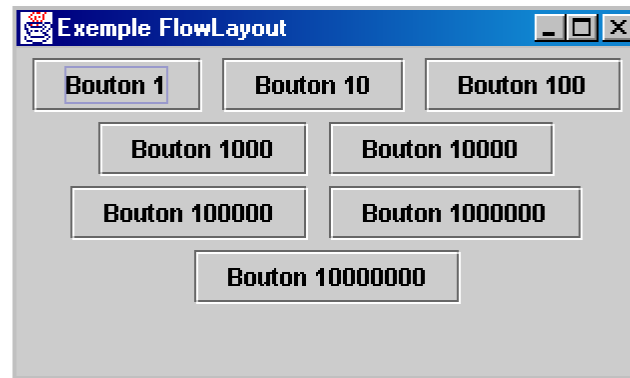
    Les autres paramètres sont : LEFT et .RIGHT

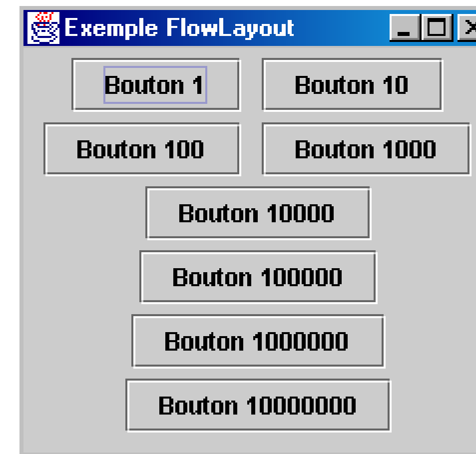
  - On peut également espacer les composants en hauteur et en largeur :

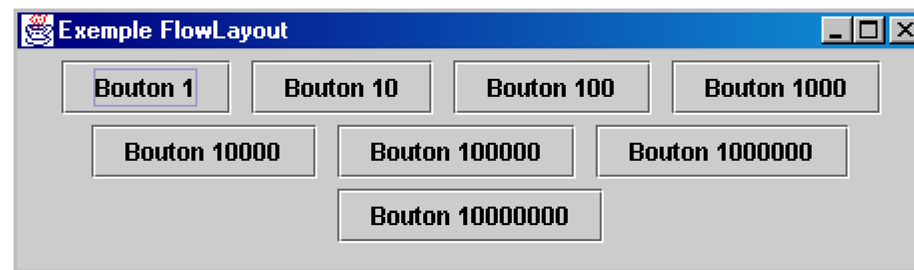
```
contenu.setLayout (new FlowLayout(FlowLayout.LEFT, 10, 15)) ;
        // Espace-larg=10, Espace_haut=15
```

Les gestionnaires de mise en forme

- FlowLayout : Layout2.java
  - Différents affichages obtenus en étirant et rétrécissant la fenêtre







Les gestionnaires de mise en forme

- GridLayout : Layout4.java
  - Permet de disposer les composants suivant une grille régulière, chaque composant occupant une cellule
  - A la construction, on choisit :

```
Conteneur.setLayout (new GridLayout (5,4)); // 5 lignes, 4 colonnes
Conteneur.setLayout (new GridLayout (5,4,15,10)); // 5 lignes, 4 colonnes, intervalle horizontal de 15, intervalle vertical de 10
```

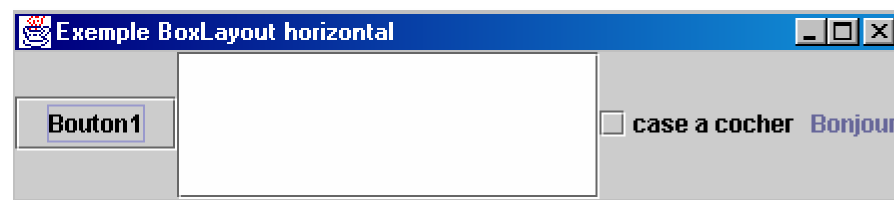
Les gestionnaires de mise en forme

- BoxLayout :
  - Permet de disposer les composants suivant une seule ligne ou une seule colonne, avec cependant certaines souplesses
  - Création par la méthode create…Box:

    Box ligne = Box. createHorizontalBox(); //box horizontal

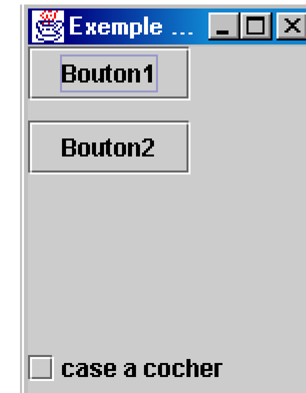
    Box ligne = Box. createVerticalBox(); //box horizontal

    - Un tel conteneur est doté par défaut d'un gestionnaire de type BoxLayout
  - Exemple : Layout5.java
    - Crée un Box horizontal dans lequel il place : un bouton, un champ de texte (de longueur 20), une case à cocher et une étiquette



Les gestionnaires de mise en forme

- Espacement des composants : Layout7.java
  - Java permet d'espacer les composants en insérant des objets virtuels appelés Strut (pour la verticale) et Glue pour l'horizontale
  - Comment
    - Après l'ajout d'un composant par add :
      bVert.add(b1)
    - Ajouter le même élément en mettant comme paramètre :
      - bVert.add(box.createVerticalStrut(10));
        //espace de 10 pixels



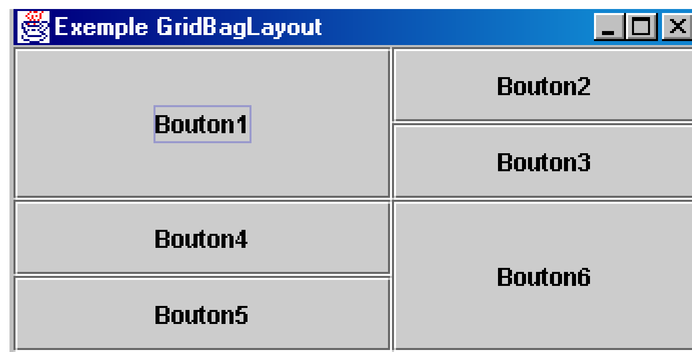
Les gestionnaires de mise en forme

- GridBagLayout
  - C'est le gestionnaire le plus souple, mais le plus difficile à employer, permet de disposer les composants en grille, mais ceux-ci peuvent occuper plusieurs cellules
  - Construction :

    GridLayout g = new GridLayout (); // création sans aucune information
  - Ajout des composants :
    - On fournit à la méthode add un 2ème argument de type GridBagConstraints dans lequel on précise les paramètres du composant :
      - Gridx(abscisse du coin sup gauche), gridy(idem pour y), gridwith(largeur), gridheight, fill(manière dont le composant occupe l'espace disponible : i.e. ajustage horizontal…)

Les gestionnaires de mise en forme

- Exemple : GridBag.java



Texte et graphiques

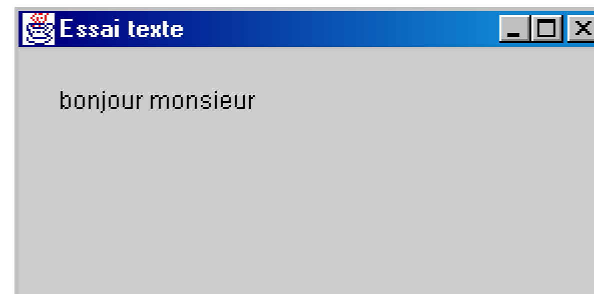
- Déterminer la position du texte
  - Affichage d'un seul texte
    - La classe Graphics offre la méthode drawString

      drawString("Bonjour", 50, 100); //affiche "Bonjour" en (50,100)
  - Affichage de deux textes consécutifs sur la même ligne
    - Comme nous ne connaissons pas la position de la fin de la première chaîne, nous utilisons deux méthodes :
      - getFontMetrics :
        - qui donne les caractéristiques de la fonte courante dans le panel créé
      - stringWith :
        - qui calcule la largeur en pixel de la chaîne dans la fonte

Texte et graphiques

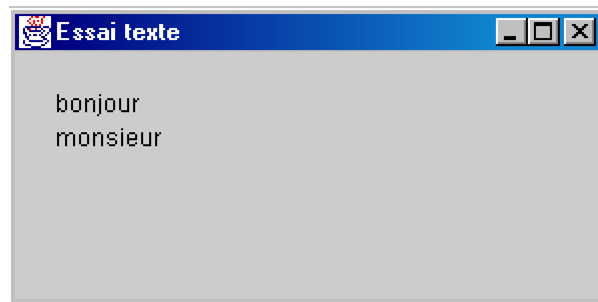
- Exemple : PremTxt1.java

```
String ch1 = "bonjour";
String ch2 = " monsieur"; //espace au début
g.drawString(ch1, x, y);
FontMetrics fm = g.getFontMetrics() ;
x += fm.stringWidth(ch1);
g.drawString(ch2, x, y);
```



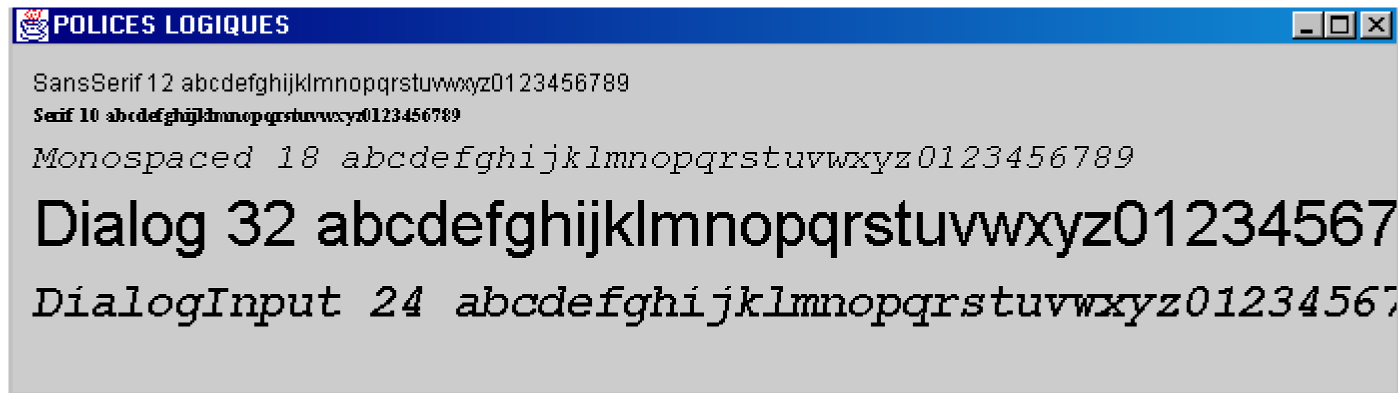
Texte et graphiques

- Affichage de deux lignes consécutives : PremTxt2.java
  - Calcul identique pour la hauteur



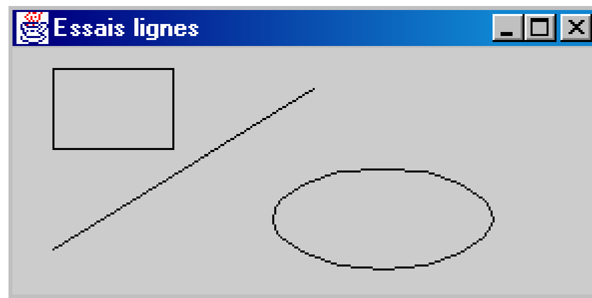
Texte et graphiques

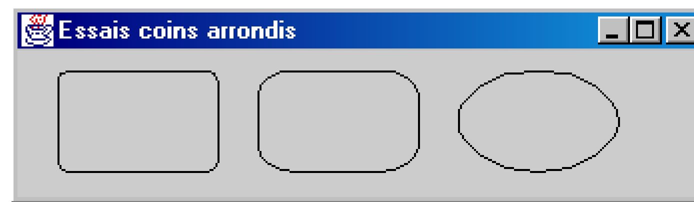
- Choix des fontes : PolLOG.java
  - Choisir une fonte courante pour un composant :
    - Utiliser la méthode setFont(fontevoulue)
    - Où fontevoulue : objet de la classe Font défini par 3 paramètres :
      - **Nom** : nom de la fonte (chaîne de caractères quelconque)
      - **Style** : Font.PLAIN, Font.BOLD, Font.ITALIC, Font.BOLD+Font.ITALIC
      - **Corps** : mesure en pixels

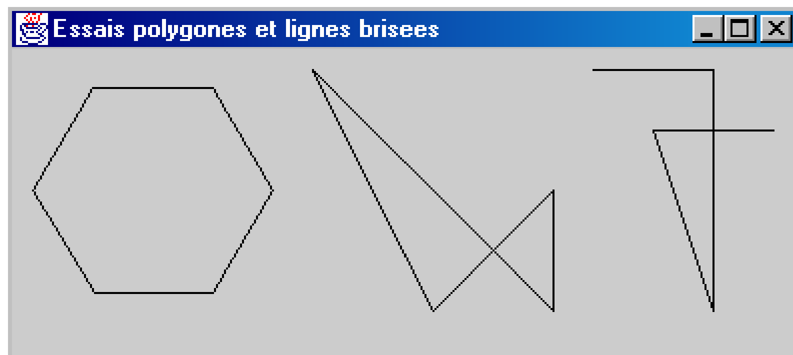


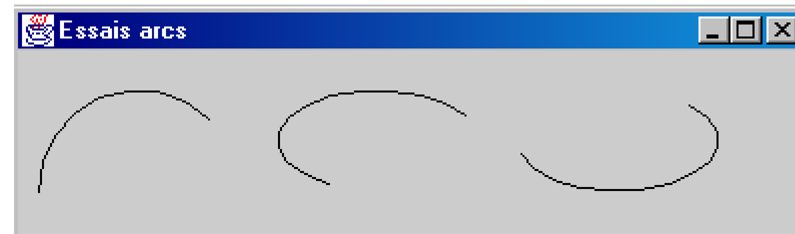
Texte et graphiques

- Tracés de lignes et formes graphiques :
  - Rien de nouveau : Lignes1.java, Lignes2.java, Polys.java, Arcs.java



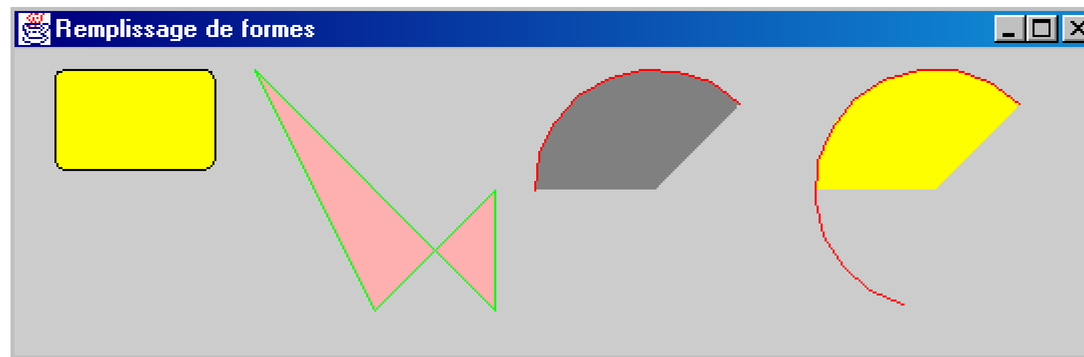






Texte et graphiques

- Remplissage de formes
  - La classe Graphics fournit une méthode pour remplir des formes : fillXXXX (ex: fillRect)
  - Exemple : Formes1.java



Texte et graphiques

- Affichage d'images : Images1.java
  - Création d'un objet image à partir d'une image existante
  - Utiliser la classe ImageIcon
    - Chargement de l'image dans rouge :

      ImageIcon rouge = new ImageIcon("rouge.gif"); //"rouge.gif" est une image gif existante dans le répertoire
    - Construction de la référence

      Image im = rouge.getImage()
    - Affichage dans un contexte graphique g

      g.drawImage(im,x,y,null); // x,y précisent le lieu d'affichage

```
import javax.swing.* ;
import java.awt.* ;
class MaFenetre extends JFrame
{ MaFenetre ()
  { setTitle ("IMAGES") ;
    setSize (300, 100) ;
    pan = new Paneau() ;
    getContentPane().add(pan) ;
  }
  private JPanel pan ;
}
class Paneau extends JPanel
{ public Paneau()
  { rouge = new ImageIcon ("rouge.gif") ;
  }
  public void paintComponent(Graphics g)
  { super.paintComponent(g) ;
    g.drawImage (rouge.getImage(), 20,30,null) ;
  }
  private ImageIcon rouge ;

}

public class Images1
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible(true) ;
  }
}
```

Texte et graphiques

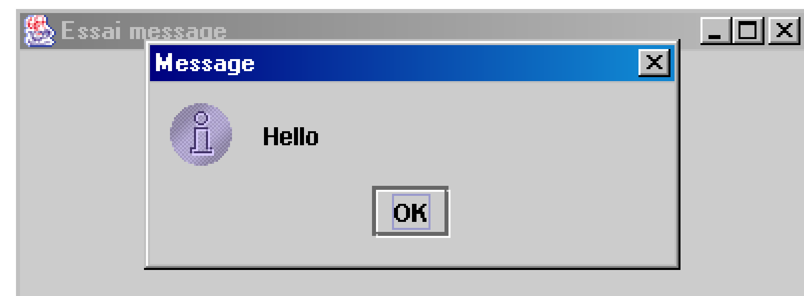
Fonctionnement

- Java dispose de méthodes standards permettant :
  - de fournir à l'utilisateur un message qui reste affiché tant qu'on n'agit pas sur un bouton ok
- Plus précisément :
  - La classe JoptionPane dispose d'une méthode statique showMessageDialog permettant d'afficher et de gérer automatiquement une telle boîte
  - Cette méthode dispose de plusieurs variantes que nous allons étudier

Les boîtes de dialogue

- La boîte de message usuelle : Mess1.java
  - L'appel suivant affiche dans la fenêtre fen la boîte de message suivante :

    JOptionPane.showMessageDialog(fen,"Hello");
  - fen est appelée fenêtre parent ou propriétaire de la boîte de message
  - Si on veut que l'affichage soit indépendant de toute fenêtre, remplacer fen par null

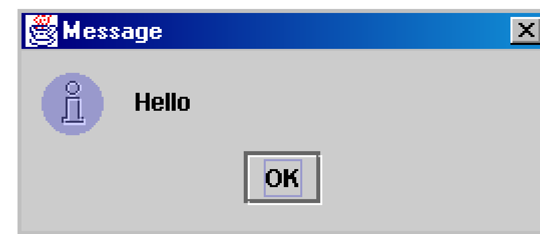


  - JOptionPane.showMessageDialog(null, "Hello");

Les boîtes de dialogue

- La boîte de message usuelle : Mess2.java
  - Cette possibilité peut très bien être exploitée dans un programme en mode console, comme suit :

```
import javax.swing.* ;
import javax.swing.* ;
public class Mess2
{ public static void main (String args[])
  { System.out.println ("avant message") ;
    JOptionPane.showMessageDialog(null, "Hello");
    System.out.println ("apres message") ;
  }
}
```



  - Dans ce cas, le programme ne crée pas de fenêtre graphique, mais il affiche quand même la boîte de message

Les boîtes de dialogue

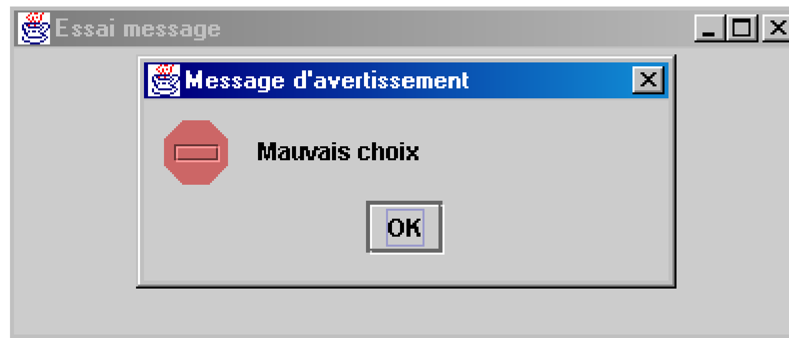
- Autres possibilités :
  - Il existe une autre variante de la méthode showMessageDialog qui permet de choisir :
    - Le contenu du message,
    - Le titre de la boîte,
    - Le type d'icône parmi la liste suivante :

Paramètre	Type d'icône
JoptionPane.ERROR_MESSAGE	Erreur
JoptionPane.INFORMATION_MESSAGE	Information
JoptionPane.WARNING_MESSAGE	Avertissement
JoptionPane.QUESTION_MESSAGE	Question
JoptionPane.PLAIN_MESSAGE	Aucune icône

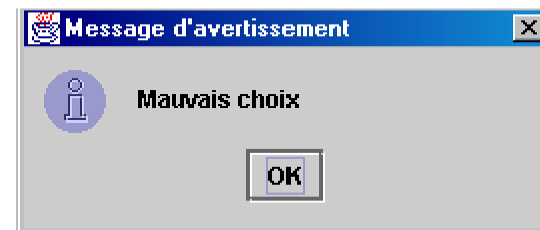
Les boîtes de dialogue

- Exemple : Mess3.java

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



Type Error



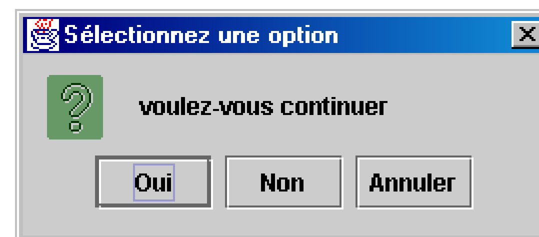
Type Information

Les boîtes de dialogue

- Les boîtes de confirmation
  - Java permet d'afficher des boîtes dites "de confirmation" offrant à l'utilisateur un choix de type oui/non
  - On utilise pour cela l'une des variantes de la méthode showConfirmDialog de la classe JOptionPane
- La boîte de confirmation la plus usuelle : Confirm1.java

  JOptionPane.showConfirmDialog(null, "voulez-vous continuer")

  - Affiche la boîte suivante :
    - La valeur de retour de la méthode est :
    - 0 (Yes), 1 (No), 2 (Cancel) et –1(fermé)



Les boîtes de dialogue

- Autres possibilités
  - Possibilités d'introduire d'autres commentaires et d'avoir d'autres types de boutons :

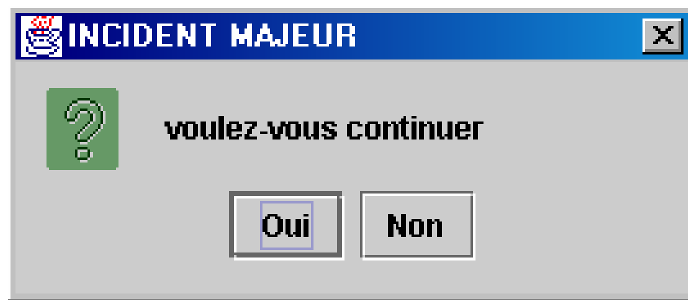
Paramètre	Valeur	Type d'icône
JoptionPane.DEFAULT_OPTION	-1	Boîte usuelle
JoptionPane.YES_NO_OPTION	0	Boutons YES et NO
JoptionPane.YES_NO_CANCEL_OPTION	1	Boutons YES, NO et CANCEL
JoptionPane.OK_CANCEL_OPTION	2	Boutons OK et CANCEL

Exemple avec cet appel :

```
JOptionPane.showConfirmDialog(null, "voulez-vous continuer",
  "INCIDENT MAJEUR", JOptionPane.YES_NO_OPTION);)
```

Les boîtes de dialogue

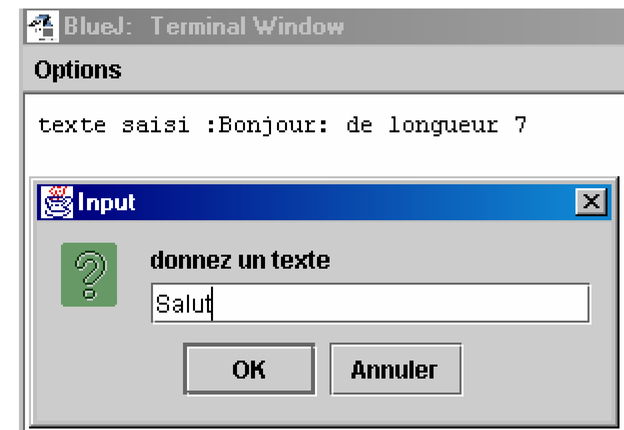
- Résultat : Mess4.java



Les boîtes de saisie

- Définition
  - La boîte de saisie permet à l'utilisateur de fournir une information sous la forme d'une chaîne de caractères
  - La méthode showInputDialog de la classe JOptionPane permet de gérer automatiquement le dialogue avec l'utilisateur
  - Plusieurs variantes :
- La boîte de saisie usuelle : Input1.java

JoptionPane.showInputDialog(fen,"donnez un texte")



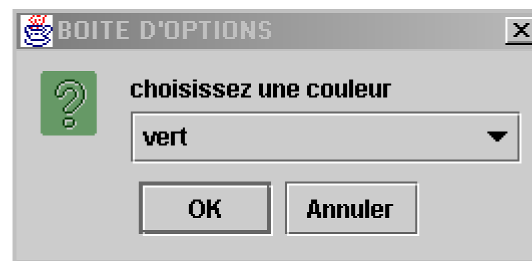
- Cette méthode fournit :
  - Soit un objet de type String
  - Soit la valeur null si l'utilisateur n'a pas confirmé sa saisie par Ok

Les boîtes d'options

- Définition : Options1.java
  - Java permet d'afficher une boîte d'options, par l'intermédiaire d'une liste combo
  - Supposons que l'on dispose d'un tableau de chaînes couleurs :

    couleurs = { "rouge", "vert", "bleu", "jaune", "orange", "blanc"} ;
  - Si fen est une fenêtre, l'appel suivant provoquera l'affichage suivant :

```
JOptionPane.showInputDialog (fen, "choisissez une couleur", "BOITE
    D'OPTIONS", JOptionPane.QUESTION_MESSAGE, /* type d'icône
    null, /* icône supplémentaire (ici aucune)
    couleurs, /* tableau de chaînes présentées dans la boîte combo
    couleurs[1]) ; /* rang de la chaîne sélectionnée par défaut
```



Les menus, les actions et les barres d'outils

- Menus déroulants
  - Java permet de doter une fenêtre de menus déroulants
- Deux possibilités
  - Créer une barre de menus qui s'affiche en haut de la fenêtre, et dans laquelle chaque menu pourra faire apparaître une liste d'options
  - Faire apparaître à un moment donné ce qu'on nomme un menu surgissant, formé quant à lui d'une liste d'options (ne sera pas étudié ici)

Les menus, les actions et les barres d'outils

Principe des menus déroulants

- Création
  - Les menus déroulants font intervenir trois sortes d'objets :
    - Un objet barre de menus (JMenuBar)
    - Différents objets menu (JMenu) qui seront visibles dans la barre de menus
    - Pour chaque menu, les différentes options, de type JMenuItem qui le constituent
  - La création d'un objet barre de menus se fait ainsi :

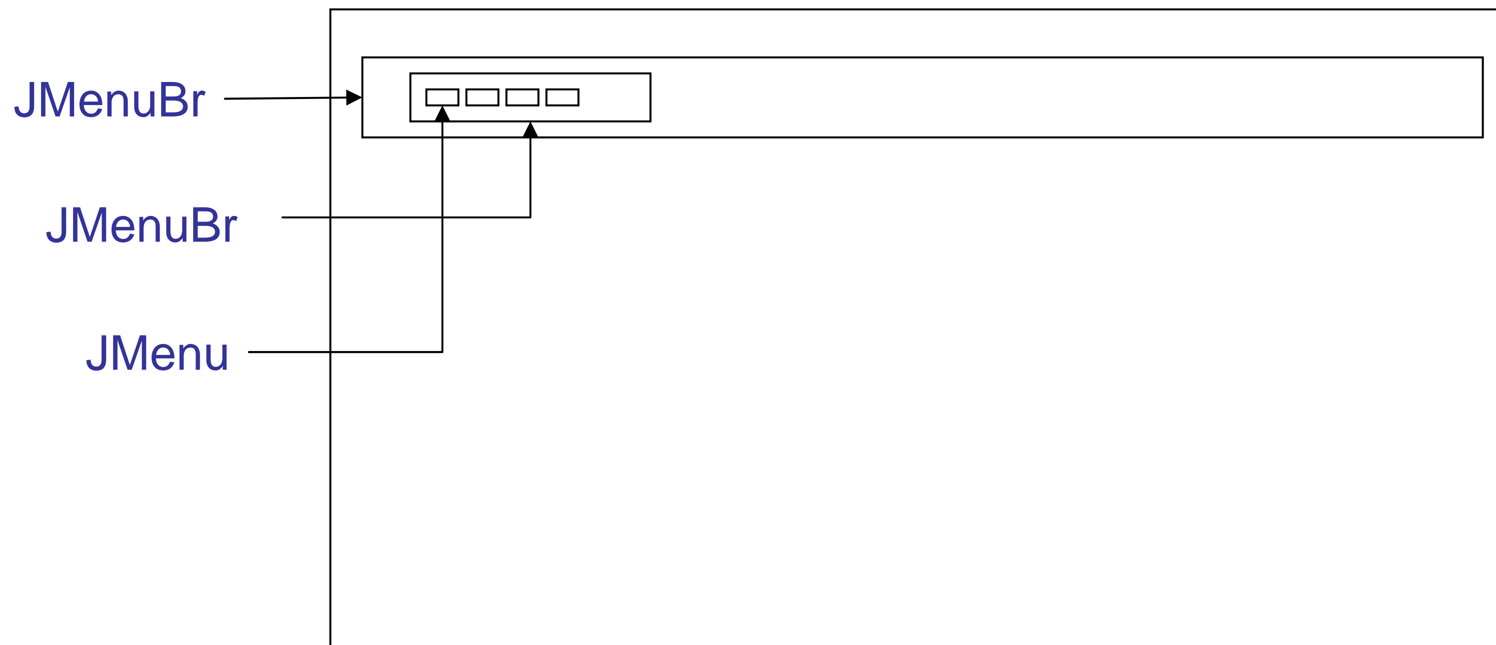
```
JMenuBar barreMenus = new JMenuBar();
```

  - Cette barre sera rattachée à une fenêtre fen par :

```
fen.setMenuBar(barreMenus); // rattache l'objet barreMenus à la fenêtre fen
```

Les menus, les actions et les barres d'outils

- Principe des menus déroulants



Les menus, les actions et les barres d'outils

Création d'un objet menu : Menu1.java

- Déclaration par le constructeur JMenu
- Ajout à la barre par add

```
JMenu couleur = new JMenu ("Couleur") ; // crée un menu de nom couleur
barreMenus.add(couleur) ; // l'ajoute à barre
```

Création d'une option

- Déclaration par appel au constructeur JMenuItem
- Ajout au menu par add :

```
Jmenu rouge = new JMenuItem ("Rouge") ; // crée une option de nom rouge
couleur.add(rouge) ; //l'ajoute au menu couleur
```

Les menus, les actions et les barres d'outils

- **Raccourcis Clavier :**
  - Il existe deux sortes de raccourcis clavier :
    - Les caractères mnémoniques
    - Les accélérateurs
  - Les caractères mnémoniques
    - Pour associer un caractère mnémonique à un menu ou à une option, on utilise la méthode setMnemonic de la classe AbstractButton
    - Exemple :

```
JMenu couleur = new JMenu ("couleur")
Couleur.setMnemonic ('c'); // C =caractère mnémonique du menu Couleur
JMenuItem rouge = new JMenuItem ("Rouge")
rouge.setMnemonic ('R'); // R =caractère mnémonique de l'option Rouge
```

    - On peut aussi préciser le mnémonique à la construction

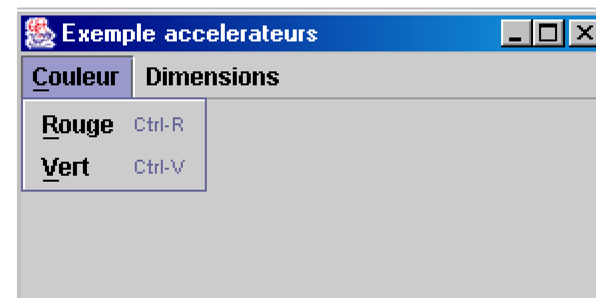
```
JMenu couleur = new JMenu ("couleur", 'C')
```

Les menus, les actions et les barres d'outils

- Raccourcis Clavier :
  - Les accélérateurs
    - Il s'agit là d'une combinaison de touches qu'on associe à une option et qui s'affiche à droite de son nom
    - Pour associer une telle combinaison, on utilise la méthode setAccelerator de la classe JMenuItem, à laquelle on fournit en argument, la combinaison de touches voulues
    - Pour ce faire, on utilise la méthode statique getKeyStroke de KeyStroke
    - Exemple : associer CTR/R à une option rouge

```
JMenuItem rouge = new JMenuItem ("Rouge")
rouge.setAccelerator(Keystroke.getKeyStroke(KeyEvent.VK_R,InputEvent.CTRL_
   MASK)
```

Exemple : Accel1



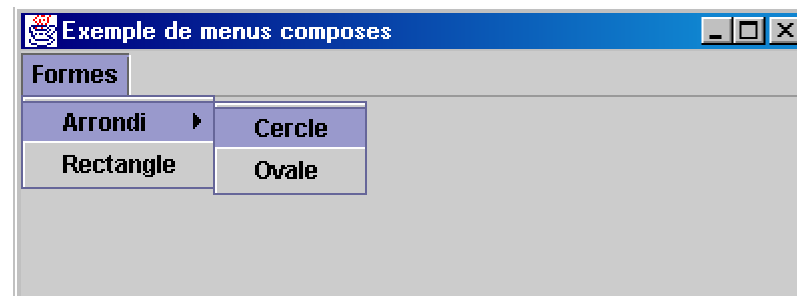
Les menus, les actions et les barres d'outils

- La bulle d'aide : ToolTip.java
  - Message affiché qui apparaît quand on laisse la souris appuyée sur certains composants
  - Obtention :
    - L'associer au composant, par ex : ici rouge

```
rouge.setToolTipText ("fond rouge");
```

Les menus, les actions et les barres d'outils

- Composition des options :
  - Une option peut à son tour faire apparaître une liste de sous-options
  - Pour obtenir ce résultat, il suffit d'utiliser dans un menu une option qui soit non plus de type JMenuItem, mais de type Jmenu
  - Après, on peut rattacher à ce sous-menu les options de notre choix
- Exemple : Compos



Les menus, les actions et les barres d'outils

- Les barres d'outils
  - Une barre d'outils est un ensemble de boutons disposés linéairement sur un des bords de la fenêtre
  - En général, ces boutons comportent des icônes plutôt que des libellés
  - Parfois, ces barres sont flottantes, i.e. on peut les déplacer d'un bord à l'autre de la fenêtre
  - Java permet de réaliser facilement de telles barres d'outils

Les menus, les actions et les barres d'outils

- Les barres d'outils : Outil1.java
  - Création d'une barre d'outils

    ```
    JToolBar barreCouleurs = new JToolBar();
    ```

  - Introduction de boutons dans la barre

    ```
    JButton boutonRouge = new Button ("Rouge");
    barreCouleurs.add(boutonRouge);
    JButton boutonVert = new Button ("Vert");
    barreCouleurs.add(boutonVert);
    ```

  - Ajout de la barre à la fenêtre fen

    ```
    Fen.getContentPane().add(barreCouleurs);
    ```

  - Gestion de la barre
    - Identique à celle des boutons

Les menus, les actions et les barres d'outils

- Utilisation d'icônes dans une barre d'outils
  - On sait qu'un bouton (JButton) peut être créé avec une icône au lieu d'un texte
  - Si on dispose d'un fichier nommé rouge.gif et contenant un dessin d'un carré de couleur rouge, on peut créer un objet icône de cette façon :

    ImageIcon iconRouge = ImageIcon("rouge.gif")