Interfaces Utilisateurs Graphiques en Java

Bibliographies

Voir le chapitre "Interfaces graphiques en Java - Introduction"
http://deptinfo.unice.fr/~grin/messupports/

L'ensemble des exemples ont été pris du lien qui suit. À l'époque Philippe les a développés pour une interface à base de AWT. J'ai adapté ces exemples pour le paquetage Swing. Intéressant de voir les différences entre les deux adaptations.
http://www.iro.umontreal.ca/~felipe/IFT1170-Ete2000/exemples.html#cours6

Difficile de passer sous silence la bible en la matière:
http://java.sun.com/docs/books/tutorial/uiswing/index.html

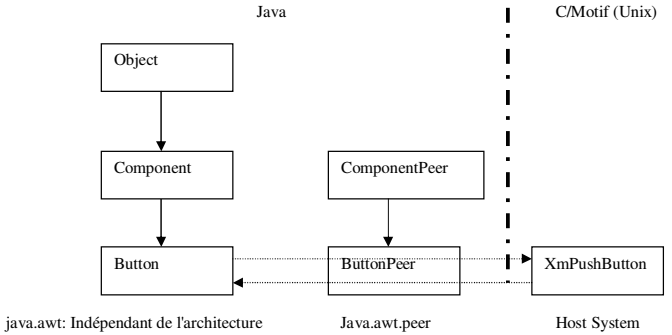
Les livres suivants:

Claude Dellanoy "Programmer en Java" (niveau IFT170)

Core Java -1- Pour la partie basic des GUI.
Core Java -2- Pour la partie avancée des GUI.

AWT (Abstract Windowing Toolkit)

- est le paquetage de base pour construire et manipuler des interfaces utilisateurs graphiques,
- parmi les paquetages originaux de Java.
- réalisé en quelques semaines …mais mis à jour plus d'une fois … d'où toutes les questions relatives à la compatibilité.
- les composants de AWT sont appelés composants lourds, car assez gourmands en ressources systèmes.
- dépendent du système de fenêtrage de la plate-forme locale et sont donc étroitement liés à cette plate-forme.
- En effet, chaque composant a un pendant ou un pair (peer) responsable de l'interaction entre le composant et la plate-forme locale, ainsi que l'affichage et de la manipulation du composant.

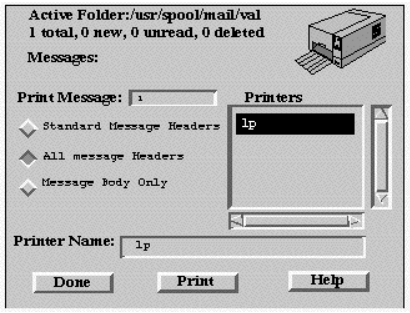


- Sun déconseille l'utilisation de AWT et encourage les programmeurs à utiliser plutôt Swing.
- Swing un nouveau paquetage dont les composantes sont écrites, manipulées et affichées complètement en Java ("pur" java).
- Ce nouveau paquetage repose sur AWT mais certaines composantes ont été complètement réécrites. En réalité dans AWT pour chaque composante (exemple un bouton) était associée une fenêtre. Une simple application graphique se retrouvait avec une multitude de fenêtres d'où la gourmandise en ressources systèmes.

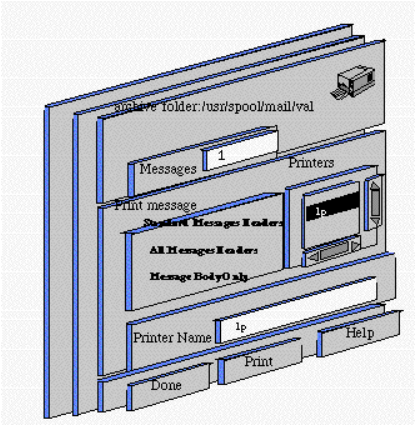
- Cependant, vu que les anciennes GUI ont été développées à base de AWT et que certains composants de Swing sont eux aussi à base de AWT, il est utile de connaître les composants de AWT.

La face cachée d'une interface graphique

Quand vous voyez ceci:



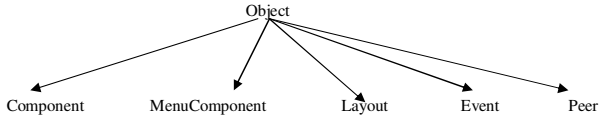
En réalité vous avez codé ceci:



Il existe donc des composants graphiques qui en contiennent d'autres (et gèrent leur apparition, leur positionnement, ...) et d'autres qui n'en contiennent pas comme les boutons poussoirs, les fenêtres textes de saisie, ... Ces derniers sont parfois appelés les contrôles.

La hiérarchies des classes AWT

Les composants de AWT font donc partie des plates-formes JDK 1.0 & JDK 1.1. La plate-forme Java 2 continue de supporter les AWT pour des raisons de comptabilités.

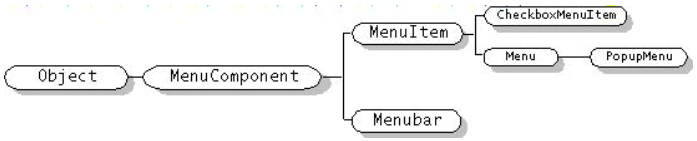


Layout : Dans une toolkit graphique, les placements des objets graphiques sont calculés et non précisés par des coordonnées absolues. Les layout précisent les contraintes de placement des objets graphiques contenu dans un container.

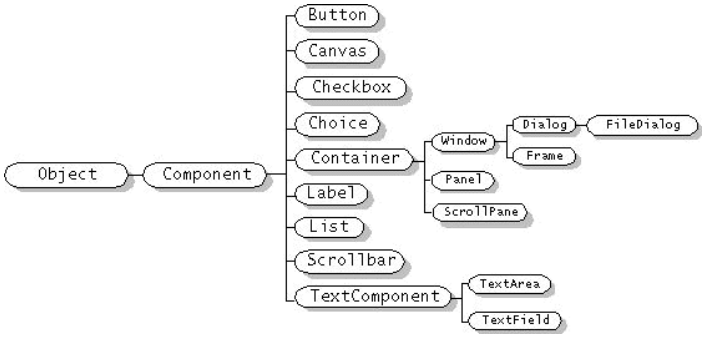
Event : Cette classe représente les événements. Le modèle d'événement de l'AWT a été modifié à chaque version de JDK.

Peer: Les classes de connexions aux toolkits graphiques des systèmes hôtes. Pour chaque classe de component ou menucomponent, il existe une classe componentpeer ou menucomponentpeer.

MenuComponent : Toute l'artillerie pour faire des menus (menubar, menuitem, menu, Checkboxmenu). Cette différenciation est due à l'hétérogénéité des systèmes hôtes où les menus sont traités de manière tellement disparates qu'il n'était pas possible d'avoir une seule hiérarchie de composants.



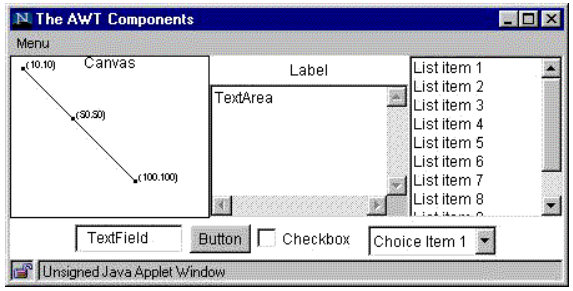
Component : Un objet abstrait ayant une position, une taille, pouvant être dessiné à l'écran et pouvant recevoir des événements. On peut différencier les objets graphiques primitifs (bouton, label), des containers qui permettent de composer des objets primitifs ou d'autres containers. On peut faire l'analogie avec des fichiers qui sont des éléments de stockage primitifs et les répertoires qui peuvent contenir soit des fichiers soit d'autres répertoires. Les containers sont donc bien des répertoires d'objets graphiques.



Container: classe abstraite de composants qui en contiennent d'autres

Button bouton
Checkbox boîte à cocher
Canevas: pour dessiner
TextComponent
TextField texte d'une ligne
TextArea texte général

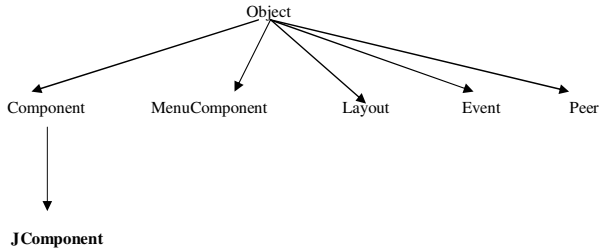
Window: fenêtre de 1er niveau, sans bordure
Frame: fenêtre principale (avec titre et bordure)
Dialog: fenêtre de dialogue (boite de dialogue)
FileDialog: boite de dialogue pour sélection de fichiers
Panel: zone dans un container, regroupement de composants ou zone de dessin ou un "fourre-tout".




```
import java.awt.*;

public class HelloWorldAWT {
    public static void main(String[] args) {
      Frame frame = new Frame("HelloWorldSwing");
      final Label label = new Label("Hello World");
      frame.add(label);
            frame.pack();
      frame.setVisible(true);
    }
}
```

La hiérarchie des classes SWING

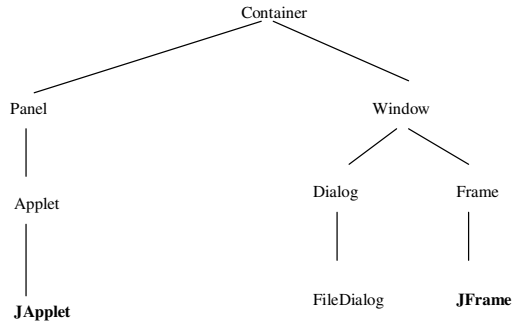
Les swings font partie de la JFC (Java Foundation Class JDK 1.1, 1.2), qui contient en plus des swings, les outils pour la gestion des impressions, API pour des utilisateurs ayant un handicap, l'API java 2D etc.



Vous pouvez identifier les composants de swing à partir de leur nom qui débute par la lettre J (JLabel, Jbutton etc.). Ces composants se trouvent dans le paquetage:

javax.swing

x vient du mot "eXtension"




```
import javax.swing.*;

public class HelloWorldSwing {
    public static void main(String[] args) {
      JFrame frame = new JFrame("HelloWorldSwing");
      final JLabel label = new JLabel("Hello World");
      frame.getContentPane().add(label);

      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.pack();
      frame.setVisible(true);
    }
}
```


Réalisation d'un constructeur de phrases (ou phrazibus)

-1- tout dans le main

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

public class Phrazibus1 extends JFrame {

    Phrazibus1(int largeur, int hauteur) {
      super("Premiere version: Fenetre 1");
      setSize(largeur,hauteur);
    }

  //Main method
  public static void main(String[] args) {

     Phrazibus1 top = new Phrazibus1(400,200);
     top.setTitle("Phrazibus");

     // un ensemble de sujet
     String  Sujet[] = {"Marie", "Julie", "Linda", "Pierre","Jacques"};
     String  Verbes[] = {"aime", "deteste", "adore"};
     String  Cod[] = {"les pommes","Fabienne","Jean","le sucre"};

     // le Layout par defaut des Frames est un BorderLayout qui
     // ici ne convient pas (nous le verrons plus tard)

     Container c = top.getContentPane();

     c.setLayout(new FlowLayout());
     // ajoutons les boutons dans la fenetre top
     for (int i=0; i<Sujet.length; i++) c.add(new JButton(Sujet[i]));
     for (int i=0; i<Verbes.length; i++) c.add(new JButton(Verbes[i]));
     for (int i=0; i<Cod.length; i++) c.add(new JButton(Cod[i]));

     // afichage de la phrase
     c.add(new JLabel("Phrase:"));
     c.add(new JTextField(30));

     // un dernier bouton
     // ou bien c.add(new JButton("Nouvelle phrase"));
     // puisque c fait référence à top.getContentPane();
     top.getContentPane().add(new JButton("Nouvelle phrase"));
     top.show();
     top.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }
}
```

Les boutons sont rangés l'un après l'autre de gauche à droite (c'est le FlowLayout qui responsable de cette organisation). S'il n'y a pas de place sur une ligne, il saute à la ligne qui suit. Remarquer la présence du mot Phrase juste après "le sucre" ….
Il ne faut pas oublier que le frame a une taille FIXE (ici de 400x200). Il faut que cette taille soit suffisante pour contenir tous les composants. Sinon ces composants ne seront pas visibles (Essayer de réduire la taille de votre browser Netscape, et vous allez remarquer que certains composants ne seront plus visibles).

-2- création d'une classe distincte

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


public class Phrazibus2 extends JFrame {

    Phrazibus2 (int l,int h, String titre) {

      setSize(l,h);
      setTitle(titre);


      // un ensemble de sujet
      String  Sujet[] = {"Marie", "Julie", "Linda", "Pierre","Jacques"};
      String  Verbes[] = {"aime", "deteste", "adore"};
      String  Cod[] = {"les pommes","Fabienne","Jean","le sucre"};

      // le Layout par defaut des Frames est un BorderLayout qui
      // ici ne convient pas (nous le verrons plus tard)

      Container c = getContentPane();

      c.setLayout(new FlowLayout());
      // ajoutons les boutons dans la fenetre top
      for (int i=0; i<Sujet.length; i++) c.add(new JButton(Sujet[i]));
      for (int i=0; i<Verbes.length; i++) c.add(new JButton(Verbes[i]));
```



```
      for (int i=0; i<Cod.length; i++) c.add(new JButton(Cod[i]));

      // afichage de la phrase
      c.add(new JLabel("Phrase:"));
      c.add(new JTextField(30));

      // un dernier bouton
      // ou bien c.add(new JButton("Nouvelle phrase"));
      // puisque c fait référence à top.getContentPane();
      c.add(new JButton("Nouvelle phrase"));
      show();
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }

  public static void main(String[] args) {
    new Phrazibus2(400,200,"Phrazibus 2");
  }

}
```

Même sortie que dans le premier cas.

-3- ajout de Panels

Une réorganisation des composants dans la fenêtre principale à l'aide de Panels, qui sont des composants légers pouvant contenir d'autres composants. Il est utilisé pour regrouper des composants dans une zone d'écran.

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

// ----------------------------------------------
// la même avec des panels pour organiser un peu les composants dans la fenêtre
// ----------------------------------------------

public class Phrazibus3 extends JFrame {

    Phrazibus3 (int l,int h, String titre) {

      setSize(l,h);
      setTitle(titre);

      // un ensemble de sujet
      String  Sujet[] = {"Marie", "Julie", "Linda", "Pierre","Jacques"};
      String  Verbes[] = {"aime", "deteste", "adore"};
      String  Cod[] = {"les pommes","Fabienne","Jean","le sucre"};

      Container c = getContentPane();

      // le Layout par defaut des Frames est un BorderLayout qui
      // ici ne convient pas (nous le verrons plus tard)

      c.setLayout(new FlowLayout());

      // creation  de Panels qui vont contenir certains composants

      JPanel panelSujet = new JPanel();
```

```
      JPanel panelVerbes = new JPanel();
      JPanel panelCod = new JPanel();
      JPanel panelPhrase = new JPanel();
      JPanel panelNouvelle = new JPanel();

      // remplissage des panels
      // note le Layout par defaut des panels est FlowLayout
      for (int i=0; i<Sujet.length; i++) panelSujet.add(new JButton(Sujet[i]));
      for (int i=0; i<Verbes.length; i++)
                                    panelVerbes.add(new JButton(Verbes[i]));
      for (int i=0; i<Cod.length; i++) panelCod.add(new JButton(Cod[i]));

      panelPhrase.add(new JLabel("Phrase:"));
      panelPhrase.add(new JTextField(30));
      panelNouvelle.add(new JButton("Nouvelle phrase"));

      // il reste encore a ajouter les panels dans la fenetre
      c.add(panelSujet);
      c.add(panelVerbes);
      c.add(panelCod);
      c.add(panelPhrase);
      c.add(panelNouvelle);

      show();
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }


  public static void main(String[] args) {
    new Phrazibus3(450,230,"Phrazibus 3");
  }
}
```

Remarquez que les composants Noms, verbes et Cod sont chacun à part sur une ligne. Pour réaliser cela, nous nous sommes servis du composant Panel.

Traitement des événements

L'utilisateur va intervenir sur le programme via le clavier ou la souris. Le programme devra associer des traitements aux actions possibles de l'utilisateur.

On distingue deux types d'événements:

événements de bas niveau: appuyer ou relâcher un bouton de souris.
événements logiques: clic sur une souris.

Si vous appuyez sur la lettre A, vous produisez les événements suivants:

4 événements de bas niveau:
- appuie sur la touche shift
- appuie sur la touche A
- relâchement de la touche A
- relâchement de la touche shift

1 événement logique:
- frappe du caractère A

Les événements sont représentés par des instances de sous-classes de java.util.EventObject

Événements de bas niveau : KeyEvent (action sur une touche), MouseEvent (action sur la souris)

Événements de haut niveau : FocusEvent (une fenêtre qui prend le focus ou la main), WindowEvent (fenêtre fermée, ouverte, icônifiée), ActionEvent (une action est déclenchée), ItemEvent (choisir un Item dans une liste), ComponentEvent (un composant caché, montré, déplacé, retaillé)

Dans le pragraphe qui suit, nous allons nous intéresser plus particulièrement à ActionEvent. Puis par la suite, nous introduirons au fur et à mesure les autres événements.

ActionEvent

Chacun des composants graphiques a ses écouteurs (listeners) qui s'enregistrent (ou se désenregistrent) auprès de lui comme écouteur d'un certain type d'événement (par exemple, clic de souris)

Un objet écouteur intéressé par les événements de type "action" (classe ActionEvent) doit appartenir à une classe qui implémente l'interface java.awt.event.ActionListener

Définition de ActionListener :

```
public interface ActionListener extends EventListener {
      void actionPerformed(ActionEvent e);
}
```

actionPerformed représente le message qui sera envoyé à l'écouteur.

On inscrit un tel écouteur auprès d'un composant nommé bouton par la méthode

```
bouton.addActionListener(ecouteur);
```

Quand un événement, un ActionEvent, est engendré par une action de l'utilisateur sur le bouton qui envoie le message actionPerformed() à l'écouteur. Le bouton lui passe l'événement déclencheur : ecouteur.actionPerformed(unActionEvent);

-1- Une première interaction limitée …

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

public class Phrazibus4 extends JFrame implements ActionListener {

  // ----------------------------------------------------
  // Les données membres
  // ----------------------------------------------------

  String  Sujet[] = {"Marie", "Julie", "Linda", "Pierre","Jacques"};
  String  Verbes[] = {"aime", "deteste", "adore"};
  String  Cod[] = {"les pommes","Fabienne","Jean","le sucre"};

  JPanel panelSujet = new JPanel();
  JPanel panelVerbes = new JPanel();
  JPanel panelCod = new JPanel();
  JPanel panelPhrase = new JPanel();
  JPanel panelNouvelle = new JPanel();

  JButton boutons[] = new JButton [50];
  int nbBoutons = 0;

  // ----------------------------------------------------
  // Le constructeur
  // ----------------------------------------------------
  Phrazibus4 (int l,int h, String titre) {
     setSize(l,h);
     setTitle(titre);

     Container c = getContentPane();

     // le Layout par defaut des Frames est un BorderLayout qui
     // ici ne convient pas (nous le verrons plus tard)

     c.setLayout(new FlowLayout());
```



```
     // remplissage des panels
     // note: le Layout par defaut des panels est FlowLayout


     // dans cet exemple, tous les boutons de Sujet ont été mis sur écoute.
     // chaque clic dessus va faire appel à la méthode actionPerformed

     for (int i=0; i<Sujet.length; i++) {
       boutons[nbBoutons] = new JButton(Sujet[i]);
       panelSujet.add(boutons[nbBoutons]);
       boutons[nbBoutons].addActionListener(this);
       nbBoutons++;
     }

     for (int i=0; i<Verbes.length; i++)
            panelVerbes.add(new JButton(Verbes[i]));
     for (int i=0; i<Cod.length; i++) panelCod.add(new JButton(Cod[i]));

     panelPhrase.add(new JLabel("Phrase:"));
     panelPhrase.add(new JTextField(30));

     panelNouvelle.add(new JButton("Nouvelle phrase"));

     // il reste encore a ajouter les panels dans la fenetre
     c.add(panelSujet);
     c.add(panelVerbes);
     c.add(panelCod);
     c.add(panelPhrase);
     c.add(panelNouvelle);
     show();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }

  // ----------------------------------------------------------------
  // la gestion des evenements
  // ----------------------------------------------------------------

  public void actionPerformed (ActionEvent evt) {
    System.out.println("Action");
  }

  // ----------------------------------------------------------------
  public static void main(String[] args) {
    new Phrazibus3(450,230,"Phrazibus 4");
  }

}
```

L'interface graphique ne change pas, par rapport à Phrazibus3, par contre à chaque action sur un des boutons de Sujet, le mot "Action" sera affiché sur la sortie standard.

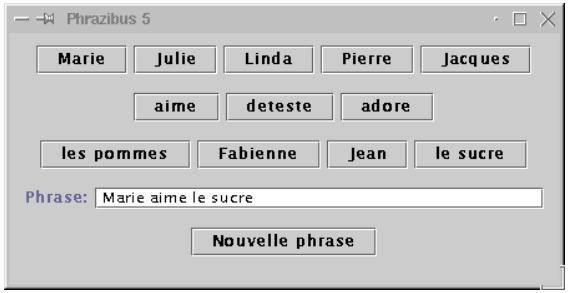
Chaque clic sur un des composants du panel verbe, va provoquer l'afficher en sortie de la chaîne "Action".

-2- une action plus poussée suite à l'interaction …

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

public class Phrazibus5 extends JFrame implements ActionListener {

  // ----------------------------------------------------
  // Les donnees membres
  // ----------------------------------------------------

  String  Sujet[] = {"Marie", "Julie", "Linda", "Pierre","Jacques"};
  String  Verbes[] = {"aime", "deteste", "adore"};
  String  Cod[] = {"les pommes","Fabienne","Jean","le sucre"};

  JPanel panelSujet = new JPanel();
  JPanel panelVerbes = new JPanel();
  JPanel panelCod = new JPanel();
  JPanel panelPhrase = new JPanel();
  JPanel panelNouvelle = new JPanel();

  JButton boutons[] = new JButton [50];
  int nbBoutons = 0;

  private JTextField texte; // une référence sur la phrase

  // ----------------------------------------------------
  // Le constructeur
  // ----------------------------------------------------
  Phrazibus5 (int l,int h, String titre) {
     setSize(l,h);
     setTitle(titre);

     Container c = getContentPane();

     // le Layout par defaut des Frames est un BorderLayout qui
     // ici ne convient pas (nous le verrons plus tard)

     c.setLayout(new FlowLayout());

     // remplissage des panels
     // note: le Layout par defaut des panels est FlowLayout

     for (int i=0; i<Sujet.length; i++) {
       boutons[nbBoutons] = new JButton(Sujet[i]);
       panelSujet.add(boutons[nbBoutons]);
       boutons[nbBoutons].addActionListener(this);
       nbBoutons++;
     }


     for (int i=0; i<Verbes.length; i++) {
        panelVerbes.add(boutons[nbBoutons]=new JButton(Verbes[i]));
        boutons[nbBoutons++].addActionListener(this);
     }

     for (int i=0; i<Cod.length; i++) {
        panelCod.add(boutons[nbBoutons]=new JButton(Cod[i]));
```



```
        boutons[nbBoutons++].addActionListener(this);
     }


     panelPhrase.add(new JLabel("Phrase:"));
     panelPhrase.add(texte = new JTextField(30));


     panelNouvelle.add(new JButton("Nouvelle phrase"));

     // il reste encore a ajouter les panels dans la fenetre
     c.add(panelSujet);
     c.add(panelVerbes);
     c.add(panelCod);
     c.add(panelPhrase);
     c.add(panelNouvelle);
     show();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }

  // ----------------------------------------------------------------
  // la gestion des evenements
  // ----------------------------------------------------------------

  public void actionPerformed (ActionEvent evt) {
    String arg = evt.getActionCommand();
    texte.setText(texte.getText() + " " + arg);
  }

  // ----------------------------------------------------------------
  public static void main(String[] args) {
    new Phrazibus5(450,230,"Phrazibus 4");
  }

}
```



Les composants graphiques sont véritablement des objets i.e. des données et du code associé. Cliquer sur le bouton de label Marie, est d'une certaine taille en pixels (données) et le fait apparaître enfoncé (code lancé). Ce code va afficher, dans le TextField, le mot associé à chaque bouton (ici le label du bouton).
La phrase "Marie aime le sucre" a été obtenue en cliquant successivement sur les boutons "Marie", "aime" et "le sucre".

La variable texte (JTextField) a été ajoutée pour récupérer la référence sur l'objet crée du type JTextField.

getActionCommand permet d'obtenir la chaîne de commande associé à la source d'un événement. Par défaut, dans le cas d'un bouton, la chaîne de commande est l'étiquette du bouton.

-3- Le champ TextField

Dans les précédents programmes, nous n'avons réalisé aucun traitement spécifique sur la zone de texte, i.e. le champ TextField se situant à droite du label "Phrase:". Ainsi, on peut se retrouver dans la situation suivante:

On constate que la zone texte n'a pas été effacée, suite à une seconde saisie. Les phrases "Marie aime les pommes" et "Linda deteste Jean" apparaissent l'une à la suite de l'autre.

Pour corriger cela, nous allons activer le bouton "Nouvelle phrase" permettant d'effacer le contenu de la zone texte "Phrase".

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

public class Phrazibus6 extends JFrame implements ActionListener {
```



```
// ------------------------------------------------
// Les donnees membres
// ------------------------------------------------

String  Sujet[] = {"Marie", "Julie", "Linda", "Pierre","Jacques"};
String  Verbes[] = {"aime", "deteste", "adore"};
String  Cod[] = {"les pommes","Fabienne","Jean","le sucre"};

JPanel panelSujet = new JPanel();
JPanel panelVerbes = new JPanel();
JPanel panelCod = new JPanel();
JPanel panelPhrase = new JPanel();
JPanel panelNouvelle = new JPanel();

JButton boutons[] = new JButton [50];
int nbBoutons = 0;

private JTextField texte; // une ref sur la phrase

// ------------------------------------------------
// Le constructeur
// ------------------------------------------------
Phrazibus6 (int l,int h, String titre) {
   setSize(l,h);
   setTitle(titre);

   Container c = getContentPane();

   // le Layout par defaut des Frames est un FlowLayout


   c.setLayout(new FlowLayout());

   // remplissage des panels
   // note: le Layout par defaut des panels est FlowLayout

   for (int i=0; i<Sujet.length; i++) {
     boutons[nbBoutons] = new JButton(Sujet[i]);
     panelSujet.add(boutons[nbBoutons]);
     boutons[nbBoutons].addActionListener(this);
     nbBoutons++;
   }
   for (int i=0; i<Verbes.length; i++) {
      panelVerbes.add(boutons[nbBoutons]=new JButton(Verbes[i]));
      boutons[nbBoutons++].addActionListener(this);
   }

   for (int i=0; i<Cod.length; i++) {
      panelCod.add(boutons[nbBoutons]=new JButton(Cod[i]));
      boutons[nbBoutons++].addActionListener(this);
   }

   panelPhrase.add(new JLabel("Phrase:"));
   panelPhrase.add(texte = new JTextField(30));

   // on ne peut plus écrire directement dans la zone de texte
   texte.setEnabled(false);
```

```
    texte.setDisabledTextColor(Color.black); // sinon gris!

    panelNouvelle.add(boutons[nbBoutons] =
                          new JButton("Nouvelle phrase"));
    boutons[nbBoutons++].addActionListener(this);


    // il reste encore a ajouter les panels dans la fenetre
    c.add(panelSujet);
    c.add(panelVerbes);
    c.add(panelCod);
    c.add(panelPhrase);
    c.add(panelNouvelle);
    show();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }


  // ---------------------------------------------------------
  // la gestion des evenements
  // ---------------------------------------------------------

  public void actionPerformed (ActionEvent evt) {
    String arg = evt.getActionCommand();

    /*
    // Voici une premiere methode

    if (arg.equals("Nouvelle phrase")) {
      texte.setText("");
    } else
      texte.setText(texte.getText() + " " + arg);
    */

    // Celle-ci est plus propre

    Object obj = evt.getSource();
    if (obj == boutons[nbBoutons-1]) texte.setText("");
    else{
      texte.setText(texte.getText() + " " + evt.getActionCommand());
    }
  }

  // ---------------------------------------------------------
  public static void main(String[] args) {
    new Phrazibus6(450,230,"Phrazibus 6");
  }

}
```

Maintenant le bouton "Nouvelle phrase est actif"!

texte.setEnabled(false): avant nous pouvions écrire dans la zone de texte ; avec cette commande, il n'est plus permis de le faire.

texte.setDisabledTextColor(Color.black): nous avons remarqué que l'affichage par défaut était gris. Nous l'avons remis à noir.

evt.getSource(): cette commande permet d'identifier la source d'un événement.
texte.setText: permet de modifier donc la zone de texte.

texte.getText(): permet de récupérer le texte présent dans la zone de texte.

4- Contrôle de l'interaction

Pour l'instant toutes les séquences de clics étaient possibles avec les boutons. Le champ texte pouvait contenir par exemple la séquence suivante de mots: "Marie Linda deteste aime le sucre" etc. Il faut donc une certaine gestion des interactions. Cette gestion est réalisée plutôt de manière algorithmique que graphique.

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

public class Phrazibus7 extends JFrame implements ActionListener {

    // ------------------------------------------------
    // Les donnees membres
    // ------------------------------------------------

    String  Sujet[] = {"Marie", "Julie", "Linda", "Pierre","Jacques"};
    String  Verbes[] = {"aime", "deteste", "adore"};
    String  Cod[] = {"les pommes","Fabienne","Jean","le sucre"};

    JPanel panelSujet = new JPanel();
    JPanel panelVerbes = new JPanel();
    JPanel panelCod = new JPanel();
    JPanel panelPhrase = new JPanel();
    JPanel panelNouvelle = new JPanel();

    JButton boutons[] = new JButton [50];
    int nbBoutons = 0;

    private JTextField texte; // une ref sur la phrase


    // et de quelques variables d'etat
    private boolean aUnSujet,aUnVerbe;
    private int nbMots;
    private final int indiceDebutSujet,indiceDebutVerbe;
    private final int indiceDebutCod,indiceFinCod;



    // ------------------------------------------------
    // Le constructeur
    // ------------------------------------------------
```

```
Phrazibus7 (int l,int h, String titre) {
  setSize(l,h);
  setTitle(titre);

  initEtat();

  Container c = getContentPane();

  // le Layout par defaut des Frames est un FlowLayout

  c.setLayout(new FlowLayout());

  // remplissage des panels
  // note: le Layout par defaut des panels est FlowLayout

  for (int i=indiceDebutSujet=0; i<Sujet.length; i++) {
      boutons[nbBoutons] = new JButton(Sujet[i]);
      panelSujet.add(boutons[nbBoutons]);
      boutons[nbBoutons].addActionListener(this);
      nbBoutons++;
  }

  indiceDebutVerbe = nbBoutons;

  for (int i=0; i<Verbes.length; i++) {
      panelVerbes.add(boutons[nbBoutons]=new JButton(Verbes[i]));
      boutons[nbBoutons++].addActionListener(this);
  }

  indiceDebutCod = nbBoutons;

  for (int i=0; i<Cod.length; i++) {
      panelCod.add(boutons[nbBoutons]=new JButton(Cod[i]));
      boutons[nbBoutons++].addActionListener(this);
  }

  indiceFinCod = nbBoutons;

  panelPhrase.add(new JLabel("Phrase:"));
  panelPhrase.add(texte = new JTextField(30));

  // on ne peut plus écrire directement dans la zone de texte
  texte.setEnabled(false);
  texte.setDisabledTextColor(Color.black); // gris sinon.

  panelNouvelle.add(boutons[nbBoutons]=
                              new JButton("Nouvelle phrase"));
  boutons[nbBoutons++].addActionListener(this);

  // il reste encore a ajouter les panels dans la fenetre
  c.add(panelSujet);
  c.add(panelVerbes);
  c.add(panelCod);
  c.add(panelPhrase);
```



```
  c.add(panelNouvelle);
  show();
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

// ------------------------------------------------------------
// la gestion des événements
// ------------------------------------------------------------

public void actionPerformed (ActionEvent evt) {
  String arg = evt.getActionCommand();

  Object obj = evt.getSource();
  if (obj == boutons[nbBoutons-1]) {
      texte.setText("");
      initEtat();
  }
  else
      if (nbMots < 3) {// premiere contrainte
        if (aUnSujet && aUnVerbe && estUnCod(obj)) {
            texte.setText(texte.getText()
                                + " " + evt.getActionCommand());
            nbMots++;
        }
        else
            if (aUnSujet && (!aUnVerbe) && estUnVerbe(obj)) {
              texte.setText(texte.getText()
                                + " " + evt.getActionCommand());
              nbMots++;
              aUnVerbe = true;
            }

            else
              if ((!aUnSujet) && estUnSujet(obj)) {
                  texte.setText(texte.getText()
                          + " " + evt.getActionCommand());
                  nbMots++;
                  aUnSujet = true;
              }
      }
}

// ------------------------------------------------------------
// quelques prédicats et méthodes privées
// ------------------------------------------------------------

private boolean estUnSujet(Object obj) {
  for (int i = indiceDebutSujet; i<indiceDebutVerbe; i++)
      if (obj == boutons[i]) return true;
  return false;
}

private boolean estUnVerbe(Object obj) {
  for (int i = indiceDebutVerbe; i<indiceDebutCod; i++)
      if (obj == boutons[i]) return true;
  return false;
}
```

```
    private boolean estUnCod(Object obj) {
      for (int i = indiceDebutCod; i<indiceFinCod; i++)
          if (obj == boutons[i]) return true;
      return false;
    }

    private void initEtat() {
      aUnSujet = aUnVerbe = false;
      nbMots = 0;
    }

    // ------------------------------------------------------------
    public static void main(String[] args) {
      new Phrazibus7(450,230,"Phrazibus 7");
    }
}
```

Cet exemple est un exercice d'algorithmique. On peut avoir un sujet suivi d'un verbe suivi d'un complément d'objet direct. Un flag a été associé à chaque catégorie, et il est mis à l'état initial par la méthode initEtat. Puis on teste en fonction des différents cas, la présence ou non de ces 3 catégories dans la phrase.

Réalisation d'un checkbox

- Un CheckBox est une case à cocher. Elle permet à l'utilisateur d'effectuer un choix de type oui/non.
- Son constructeur accepte un libellé qui représente le nom du checkbox et qui sera affiché à cote de celui ci.
- Un CheckBox génère deux événements: un ActionEvent comme étudié précédemment et un ItemEvent. Dans cette exemple nous introduisons l'utilisation de ItemEvent:

  Définition de ItemEvent :

  ```
  public interface ItemListener extends EventListener {
        void itemStateChanged(ItemEvent e);
  }
  ```

  itemStateChanged représente le message qui sera envoyé à l'écouteur.

  Il faut savoir que getActionCommand n'est disponible que pour les ActionEvent.

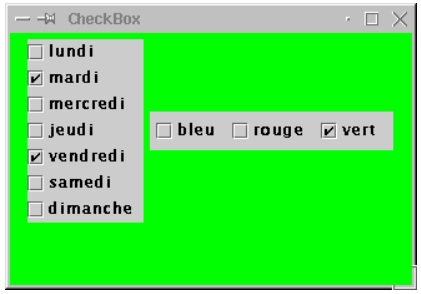
- On exploite donc un CheckBox en réagissant immédiatement à une action de la case (le cas des événements) ou bien en cherchant à connaître l'état du CheckBox à un instant donné. Pour ce dernier cas, il faut faire appel à la méthode isSelected de la classe JCheckBox:

  if (uncheckbox.isSelected()) etc.

- GridLayout permet de disposer les composants sous la forme d'une grille ligne par colonne où chaque composant occupe une cellule de cette grille.
  GridLayout(7,1) nous avons une grille de 7 lignes et une colonne. Le parcours se fait sur les lignes.

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

public class CheckBoxTest extends JFrame implements ItemListener {

    // quelques references et donnees primitives
    private JCheckBox bleue,rouge,vert;
    private JCheckBox jours[] = {
      new JCheckBox("lundi"),
      new JCheckBox("mardi"),
      new JCheckBox("mercredi"),
      new JCheckBox("jeudi"),
      new JCheckBox("vendredi"),
      new JCheckBox("samedi"),
      new JCheckBox("dimanche")
          };
    boolean bleueSelectionnee, vertSelectionnee, rougeSelectionnee;

    // le constructeur
    public CheckBoxTest(int l,int h, String titre) {
      setSize(l,h);
      setTitle(titre);

      // inutile mais plus propre
      bleueSelectionnee=vertSelectionnee=rougeSelectionnee=false;

      Container c = getContentPane();

      // le Layout par defaut des Frames est un FlowLayout qui


      c.setLayout(new FlowLayout());

      JPanel panelCouleur = new JPanel();

      bleue = new JCheckBox("bleu");
      rouge = new JCheckBox("rouge");
      vert = new JCheckBox("vert");
      panelCouleur.add(bleue);
      panelCouleur.add(rouge);
      panelCouleur.add(vert);
      bleue.addItemListener(this);
      vert.addItemListener(this);
      rouge.addItemListener(this);


      JPanel panelJours= new JPanel(new GridLayout(7,1));
```

```
    for (int i=0; i<jours.length; i++) {
        jours[i].addItemListener(this);
        panelJours.add(jours[i]);
    }
    c.add(panelJours);
    c.add(panelCouleur);
    show();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }

  // la seule méthode de l'interface ItemListener

  public void itemStateChanged(ItemEvent e) {
    Container c = getContentPane();
    if (e.getSource() == bleue) {
        bleueSelectionnee = ! bleueSelectionnee;
        c.setBackground(getCouleur());
    } else
        if (e.getSource() == rouge) {
          rougeSelectionnee = ! rougeSelectionnee;
          c.setBackground(getCouleur());
        } else
          if (e.getSource() == vert) {
              vertSelectionnee = ! vertSelectionnee;
              c.setBackground(getCouleur());
          } else { // il s'agit d'un jour
              for (int i=0; i< jours.length; i++)
                if (e.getSource() == jours[i]) {
                    System.out.println(jours[i].getText());
                    break;
                }
          }
  }

  public static void main(String [] args) {
    new CheckBoxTest(330,230,"CheckBox");
  }

  // gestion du mélange des couleurs
  private Color getCouleur() {
    if (bleueSelectionnee && rougeSelectionnee && vertSelectionnee)
                            return Color.gray;
    if (bleueSelectionnee && rougeSelectionnee) return Color.magenta;
    if (bleueSelectionnee && vertSelectionnee) return Color.orange;
    if (bleueSelectionnee) return Color.blue;

    // no blue

    if (rougeSelectionnee && vertSelectionnee) return Color.yellow;
    if (rougeSelectionnee) return Color.red;

    if (vertSelectionnee) return Color.green;
    return Color.white;
  }

}
```

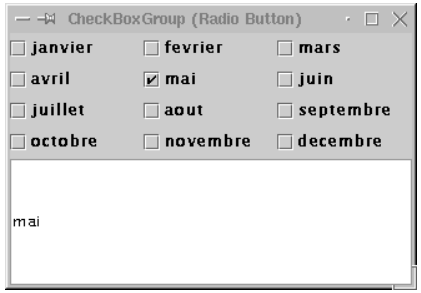



Réalisation d'un RadioButton

- Un RadioButton est une case à cocher. Elle permet à l'utilisateur d'effectuer un choix de type oui/non.
- Fait partie d'un groupe de boutons dans lequel un seul bouton est sélectionné à la fois.
- Génère les mêmes événements qu'un CheckBox.
- On peut exploiter son état à un instant donné.

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

public class RadioTest extends JFrame implements ItemListener {

    private String mois[] = {
      "janvier","fevrier","mars","avril","mai","juin",
      "juillet","aout","septembre","octobre","novembre","decembre"
    };

    private JCheckBox ref [] = new JCheckBox [mois.length];
    private JTextField texte = new JTextField ("no selection");

    // le constructeur

    public RadioTest(int l,int h, String titre) {
      setSize(l,h);
      setTitle(titre);

      ButtonGroup groupe = new ButtonGroup();
      Panel p = new Panel(new GridLayout(4,3));
```

```
      for (int i=0; i<mois.length; i++) {
          p.add(ref[i] = new JCheckBox(mois[i],false));
          groupe.add(ref[i]);
          ref[i].addItemListener(this);
      }

      ref[4].setSelected(true);


      Container c = getContentPane();

      // le Layout par defaut des Frames est un BorderLayout qui
      // ici ne convient pas (nous le verrons plus tard)
      //setLayout(new FlowLayout());   // une meilleure solution ici

      c.setLayout(new GridLayout(2,1));

      c.add(p);
      c.add(texte);
      texte.setEnabled(false);
      texte.setDisabledTextColor(Color.black);
      show();
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

    // la seule methode de l'interface ItemListener
    public void itemStateChanged(ItemEvent e) {
      for (int i=0; i<ref.length; i++)
          if (e.getSource() == ref[i]) {
            texte.setText(mois[i]);
          }
    }

    public static void main(String [] args) {
      new RadioTest(330,230,"CheckBoxGroup (Radio Button)");
    }
}
```



Réalisation d'un TextArea

Rien de nouveau par rapport aux précédents exemples. Ici on ajoute l'idée de positionner les boutons et la zone devant contenir le texte à un endroit donné dans le Frame.

Panel: contient deux boutons (ouvrir et sauver)
TextArea: la zone texte devant contenir le fichier ouvert en mode lecture et écriture.

On les organise à l'aide d'un borderlayout, sauf que nous précisons les positionnements.

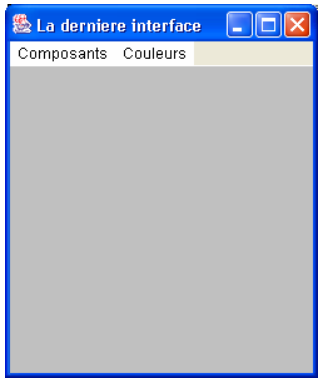
L'exemple est une représentation très basic d'un mode de lecture et écriture d'un fichier. Il y a d'autres composants à prévoir (les ascenseurs, la détection de frappe, la remise à zéro de certaines états de lecture ou de sauvegarde etc.)

```
import java.io.*; // pour les fichiers
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TextTest extends JFrame implements ActionListener {

  private JTextArea texte;
  private JButton load,save;
  private String nomFichier;

  // le constructeur
  public TextTest(int l,int h, String titre) {
     setSize(l,h);
     setTitle(titre);

     Container c = getContentPane();

     c.setLayout(new BorderLayout());

     load = new JButton("Ouvrir");
     save = new JButton("Sauver");

     load.addActionListener(this);
     save.addActionListener(this);
     save.setEnabled(false);

     JPanel p = new JPanel();
     p.add(load);
     p.add(save);

     texte = new JTextArea (10,40);

     c.add(p,BorderLayout.NORTH);
     c.add(texte,BorderLayout.SOUTH);

     show();
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }
```

```
  // illustration de la lecture et de l'écriture d'un fichier texte
  public void actionPerformed (ActionEvent evt) {

    if (evt.getSource() == load) { // LECTURE
      FileDialog boiteFichier =
            new FileDialog(this, "Ouverture de : ",FileDialog.LOAD);
      boiteFichier.show();
      nomFichier = boiteFichier.getFile();

      if (nomFichier != null) { // lecture du fichier
        try {
          BufferedReader in =
                        new BufferedReader (new FileReader(nomFichier));
          String ligne;

          texte.setText(""); // remise a zero du texte
          while ((ligne = in.readLine()) != null)
            texte.append(ligne+"\n");
          in.close();
        }
        catch (IOException e) {
          System.err.println("Erreur de lecture avec le fichier "
                                                    + nomFichier);
        }

      }

     save.setEnabled(true);
    }
  else
    if (evt.getSource() == save) {   // ECRITURE
      System.out.println("Sauvegarde de " + nomFichier);
      try {
        PrintWriter out =
                  new PrintWriter (new FileWriter(nomFichier,true));
        out.print(texte.getText());
        out.close();
        }
      catch (IOException e) {
          System.err.println("Erreur d'ecriture avec le fichier "
                                                    + nomFichier);
      }
    }
  }

  public static void main(String [] args) {
    new TextTest(330,230,"Text widget");
  }
}
```

état de départ …

Réalisation d'un Menu




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

public class MenuTest extends JFrame implements ActionListener {

  private String couleurs [] = {
     "black","blue","cyan","darkgray","gray",
     "green","lightgray","magenta","orange",
     "pink","red","white","yellow"
  };

  private Color [] colors = {
     Color.black,Color.blue,Color.cyan,Color.darkGray,Color.gray,
     Color.green,Color.lightGray,Color.magenta,Color.orange,
```

```
    Color.pink,Color.red,Color.white,Color.yellow
};

private JMenuBar mb;

private Color getColor(String s) {
  for (int i=0; i<couleurs.length; i++)
    if (couleurs[i].equals(s)) return colors[i];
  return Color.white;
}
// constructeur
public MenuTest(int l, int h, String title) {

  setTitle(title);
  setSize(l,h);

  addWindowListener(new WindowAdapter () {
      public void windowClosing (WindowEvent e) {System.exit(0);}
    }
  );


  // et une barre de menus, une ...
  mb = new JMenuBar();
  JMenuItem mi; // une reference de travail
  JMenu comp = new JMenu("Composants");
  JMenu couleur = new JMenu("Couleurs");

  // le sous menu Textes
  JMenu texteMenu = new JMenu("Textes");
  mi = new JMenuItem("TextArea");
  mi.addActionListener(this);
  texteMenu.add(mi);

  mi = new JMenuItem("TextField");
  mi.addActionListener(this);
  texteMenu.add(mi);

  // le sous menu Choix
  JMenu choixMenu = new JMenu("Choix");

  mi = new JMenuItem("Choice");
  mi.addActionListener(this);
  choixMenu.add(mi);

  mi = new JMenuItem("List");
  mi.addActionListener(this);
  choixMenu.add(mi);

  // le sous menu boutons
  JMenu boutonMenu = new JMenu("Boutons");

  mi = new JMenuItem("Button");
  mi.addActionListener(this);
  boutonMenu.add(mi);

  mi = new JMenuItem("CheckButon");
```



```
  mi.addActionListener(this);
  boutonMenu.add(mi);

  // un item qui n'est pas un sous menu
  mi = new JMenuItem("Canevas");
  mi.addActionListener(this);

  // organisation du menu composants
  comp.add(boutonMenu);
  comp.add(texteMenu);
  comp.add(choixMenu);
  comp.addSeparator();
  comp.add(mi);

  // des items
  for (int i=0 ; i<couleurs.length; i++) {
    mi = new JMenuItem(couleurs[i]);
    couleur.add(mi);
    mi.addActionListener(this);
  }


  // composition de la barre de menu
  mb.add(comp);
  mb.add(couleur);

  setJMenuBar(mb);

  show();
}

// l' action appelee lorsqu'on a clique sur un item

public void actionPerformed(ActionEvent e) {
    Container c = getContentPane();

  if (e.getSource() instanceof JMenuItem) {

    JMenuItem m = (JMenuItem)e.getSource();

    // est-ce une couleur ?
    // on pourrait comparer le texte du menuitem avec les strings
    // de la table couleurs, mais pour illustrer quelques
    // methodes supplementaires...

    boolean pasUneCouleur = true;
    // le menu 1 est le menu des couleurs (commence a 0)
    JMenu menu = mb.getMenu(1);

    for (int i=0; i<menu.getItemCount(); i++)
      if (menu.getItem(i) == m) {
        System.out.println(m.getText());

        c.setBackground(getColor(m.getText()));
        pasUneCouleur = false;
        break;
      }
```

```
      // si ca n'est pas une couleur, alors on affiche
      // le texte de l'item
      if (pasUneCouleur) {
        System.out.println(((JMenuItem)e.getSource()).getText());
      }
     }
  }

  public static void main(String [] args) {
     new MenuTest(250,300,"La derniere interface");
  }
}
```

Réalisation d'un choix de listes

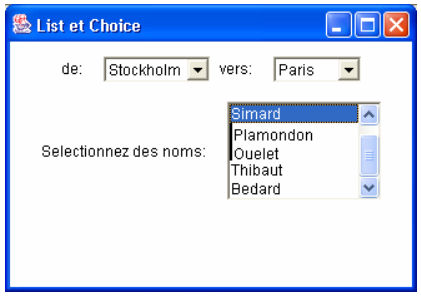
Dans swing, nous n,avons plus besoin de « choice » ni « list », mais nous utilisons à la place « ComboBox » et « JList ».
Il n'y a pas d'écoute du type événement (Action, Item) sur les listes. Nous disposons donc que de :

```
import javax.swing.event.*; // ListSelectionListener

public interface ListSelectionListener {
   void valueChanged(ListSelectionEvent e);
}
```

Il faudra faire attention aux différents clics d'entrées sorties lors de l'accès à la liste. Par ailleurs, le double clic n'est pas géré, Il faudra le prévoir lors du codage.

Il n'y a pas de scrollpane par défaut dans une liste!! Il faut le prévoir pour toute liste. Penser donc à inclure le scrollpane dans la liste mais pas l'inverse!!! Par la suite, il faudra inclure le scrollpane dans le getContentPane.

Pour le ComboBox, il faut autoriser l'édition dessus sinon vous ne verrez pas les changements!

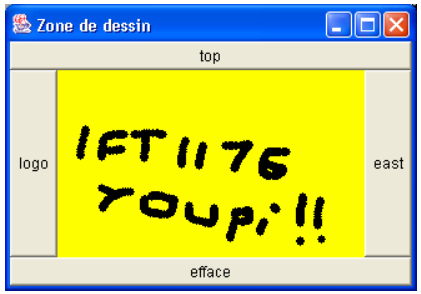



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*; // ListSelectionListener

class FenetreFermante extends JFrame {

  public FenetreFermante (int largeur,int hauteur) {
    setTitle("Quatrieme version:" + getClass().getName());
    setSize(largeur,hauteur);
    addWindowListener(new WindowAdapter () {
        public void windowClosing (WindowEvent e) {System.exit(0);}
      }
    );
  }
}

public class GuiList extends FenetreFermante implements ItemListener,
ListSelectionListener {

  private String depart[] = { "Marseille","Stockholm", "Montréal"};
  private String arrivee[] = { "Paris","Umea", "Québec"};
  private String noms[] = {
      "Tremblay","Simard","Plamondon","Ouelet","Thibaut","Bedard",
      "Martin","Marie","Elisabeth","Sophie"
  };
  private JList liste;
  private JComboBox fromChoice,toChoice;
  private JLabel fromLabel, toLabel, nomLabel;

  // le constructeur
  public GuiList(int l,int h, String titre) {
     super(l,h);
     setTitle(titre);

     Container c = getContentPane();

     c.setLayout(new FlowLayout());

     JPanel p1 = new JPanel();
     JPanel p2 = new JPanel();

     fromLabel = new JLabel("de:");
     toLabel = new JLabel("vers:");
     fromChoice = new JComboBox(depart);
     toChoice = new JComboBox(arrivee);
     fromChoice.setEditable(true);
     toChoice.setEditable(true);


     p1.add(fromLabel);
     p1.add(fromChoice);
     p1.add(toLabel);
     p1.add(toChoice);


     liste = new JList (noms);
```

```
    JScrollPane defil = new JScrollPane(liste);

    nomLabel = new JLabel("Selectionnez des noms:");

    p2.add(nomLabel);
    p2.add(defil);

    // ajout des panels
    c.add(p1);
    c.add(p2);


    // enregistrement des gestionnaires d'evenements
    fromChoice.addItemListener(this);
    toChoice.addItemListener(this);
    liste.addListSelectionListener(this);
    show();
  }

  // la seule methode de l'interface ItemListener
  public void itemStateChanged(ItemEvent e) {
    if (e.getSource() == fromChoice)
      System.out.println("Depart de " + e.getItem());
    else
      System.out.println("Arrivee de " + e.getItem());
  }

  //la seule méthode de l'interface ListSelectionListener
  public void valueChanged(ListSelectionEvent e){
      if (!e.getValueIsAdjusting()){
          System.out.print("ListactionPerformed ");
          Object[]  itemsSelectionnes = liste.getSelectedValues();
          for (int i=0; i<itemsSelectionnes.length; i++)
            System.out.print(" " + itemsSelectionnes[i]);
          System.out.println("");
      }
  }


  public static void main(String [] args) {
    new GuiList(330,250,"Liste et Choix");
  }
}
```


Réalisation d'un canevas

Par cet exemple nous allons montrer aussi la gestion de la souris.



Le « canevas » n'existe plus en swing. Il a été remplacé par un « JLabel » ou un « JPanel »

La souris peut avoir 7 actions possibles :

- MouseListener

```
public void mousePressed(MouseEvent e) ...
public void mouseClicked(MouseEvent e) ...
public void mouseReleased(MouseEvent e) ...
public void mouseEntred(MouseEvent e) ...
public void mouseExited(MouseEvent e) ...
```

- MouseMotionListener

bouton de la souris est pressé souris déplacée

```
public void mouseDragged(MouseEvent e) ...
```

souris déplacée et le curseur se trouve dans un composant

```
public void mouseMoved(MouseEvent e) ...
```

Implanter « MouseListener » signifie redéfinir toutes les méthodes! Une solution consiste à utiliser « MouseAdapter » comme suit :

```
class MouseAdapter implements MouseListener {

     public void mousePressed(MouseEvent e) {}
     public void mouseClicked(MouseEvent e) {}
     public void mouseReleased(MouseEvent e) {}
     public void mouseEntred(MouseEvent e) {}
     public void mouseExited(MouseEvent e) {}
     public void mouseDragged(MouseEvent e) {}
     public void mouseMoved(MouseEvent e) {}

}

class EcouteMaSouris extends MouseAdapter {
     public void mouseClicked(MouseEvent e) { bla bla bla bla}
}

class MaFenetre extends JFrame {
     addMouseListener(new EcouteMaSouris());
}

class MaFenetre extends JFrame {
     addMouseListener( new MouseAdapter {
                          public void mouseClicked(MouseEvent e) {
                               bla bla bla bla
                          }
                     }
                       );
}
```

Pour retrouver lequel des boutons qui a été cliqué, il faudra utiliser l'une des méthodes :

```
InputEvent.BUTTON1_MASK (gauche)
InputEvent.BUTTON2_MASK (central)
InputEvent.BUTTON3_MASK (droite)
```

```
MouseEvent --> getModifiers
if (e.getModifiers() & InputEvent.BUTTON1_MASK) != 0) blablabla ...
```

Graphics: Il est utiliser pour manipuler un contexte graphique. Pour dessiner dans un composant, il nous faut redéfinir la méthode « `void paintComponent(Graphics g);` »

La commande « `super.paintComponent(g);` » permet de réaliser en effet le dessin du composant.

ATTENTION: Il faudra faire appel à cette méthode avant de dessiner votre chef d'oeuvre, sinon votre dessin sera perdu! Il est à signaler (c'est bien dommage) que dans AWT, il y a un tour de passe rapide pour éviter ces tracasseries techniques. Voir pour cela, le code de Langlais. Avec Swing, la première impression est qu'il faut swinguer pour obtenir les mises à jour des dessins sans rien perdre en échange!


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

class MonPoint {
    int x,y;
    MonPoint(int x, int y) {
      this.x = x;
      this.y = y;
    }
    public void dessine (Graphics g) {
      g.setColor(Color.black);
      g.fillOval(x,y,7,7);
    }
}

class MonCanvas extends JPanel implements ActionListener {

    private Image image;
    private Vector memoire = new Vector();

    // constructeur juste pour creer l'image
    public MonCanvas() {
      image = Toolkit.getDefaultToolkit().getImage("logo.gif");

      // gestion du deplacement de la souris
      addMouseMotionListener(
                          new MouseMotionAdapter() {
                               public void mouseDragged (MouseEvent e) {
                                 MonPoint mp =
                                    new MonPoint(e.getX(),e.getY());
                                 Graphics g = getGraphics();
                                 mp.dessine(g);
                                 g.dispose();
                                 memoire.addElement(mp);
                               }
                          }
                         );
    }


    public void paintComponent(Graphics g) {
      super.paintComponent(g); //paint background
      setBackground(Color.yellow);
      Enumeration enumere = memoire.elements();
      // On sauvegarde les anciens points dessinés
      while(enumere.hasMoreElements())
           ((MonPoint)enumere.nextElement()).dessine(g);
    }
```

```
    public void actionPerformed (ActionEvent e) {

      memoire.clear();
      // dans tous les cas, on efface la zone d'affichage
      // On prend la réféence sur le graphique
      Graphics g = getGraphics();

      g.clearRect(0,0,getWidth(),getHeight());

      if (e.getActionCommand().equals("logo")) {
          g.drawImage(image,50,50,this);
      }else{
          // Si non redéfinie => appel à paintComponent
          repaint();
      }
      // On libère la référence sur le graphique
      g.dispose();
    }
}

// ------------------------------------------------
// test des canvas
// ------------------------------------------------

public class CanevasTest extends JFrame {

    private int xValue, yValue;

    // le constructeur
    public CanevasTest(int l,int h, String titre) {
      setSize(l,h);
      setTitle(titre);

      JButton top = new JButton("top");
      JButton west = new JButton("logo");
      JButton east = new JButton("east");
      JButton south = new JButton("efface");

      final MonCanvas center = new MonCanvas();

      Container c = getContentPane();



      // que faire lorsque l'on clique sur efface ou logo
      south.addActionListener(center);
      west.addActionListener(center);

      // Un BorderLayout ... pour positionner les composants
      // dans la fenêtre

      c.add(top,BorderLayout.NORTH);
      c.add(west,BorderLayout.WEST);
      c.add(east,BorderLayout.EAST);
      c.add(south,BorderLayout.SOUTH);
      c.add(center,BorderLayout.CENTER);
```



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

    }

    public static void main(String [] args) {
      new CanevasTest(330,230,"Zone de dessin");
    }
}
```

Fenêtres fermantes

Comment fermer proprement une fenêtre en java.

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

// ------------------------------------------------------
// une premiere version qui ne marche pas vraiment lorsque l'on
// souhaite la fermer en utilisant les icones classiques
// ------------------------------------------------------

class Fenetre1 extends JFrame {

  // ne fait que definir une taille et un titre
  public Fenetre1 (int largeur,int hauteur) {
    setTitle("Premiere version: Fenetre 1");
    setSize(largeur,hauteur);
  }

}

// ------------------------------------------------------
// une deuxieme version qui gere proprement la fermeture
// ------------------------------------------------------

class Fenetre2 extends JFrame implements WindowListener{

  public Fenetre2 (int largeur,int hauteur) {
    setTitle("Deuxieme version: Fenetre 2");
    setSize(largeur,hauteur);
    addWindowListener(this);
  }

  public void windowClosing (WindowEvent e) {System.exit(0);}
  public void windowClosed (WindowEvent e) {}
  public void windowIconified (WindowEvent e) {}
  public void windowOpened (WindowEvent e) {}
  public void windowDeiconified (WindowEvent e) {}
  public void windowActivated (WindowEvent e) {}
  public void windowDeactivated (WindowEvent e) {}

}
```

```
// ------------------------------------------------------
// une troisieme version qui gere proprement la fermeture mais sans
// l'inconvenient d'avoir a redefinir toutes les methodes de
// l'interface WindowListener
// ------------------------------------------------------

// juste une classe qui redefini l'action a executer si
// la fermeture d'une fenetre est demandee

class Fenetre3Adapter extends WindowAdapter {
  public void windowClosing (WindowEvent e) {System.exit(0);}
}

class Fenetre3 extends JFrame {

  public Fenetre3 (int largeur,int hauteur) {
    Fenetre3Adapter adapter = new Fenetre3Adapter();
    setTitle("Troisieme version: Fenetre 3");
    setSize(largeur,hauteur);
    // objet qui gere les evenements fenetre
    addWindowListener(adapter);
  }

  public void windowClosing (WindowEvent e) {System.exit(0);}
}

// ------------------------------------------------------
// une derniere version qui gere proprement la fermeture mais avec un
// code encore plus compact
// ------------------------------------------------------

class FenetreFermante extends JFrame {

  public FenetreFermante (int largeur,int hauteur) {
    // juste pour montrer l'usage de getClass().getName()
    setTitle("Quatrieme version:" + getClass().getName());
    setSize(largeur,hauteur);
    addWindowListener(new WindowAdapter () {
        public void windowClosing (WindowEvent e) {System.exit(0);}
      }
    );
  }
}
```