Créer des interfaces utilisateurs avec Java et Swing

Premier exemple.

Nous utilisons ici les composants les plus communs d'une interface.

Leur traduction en Swing :

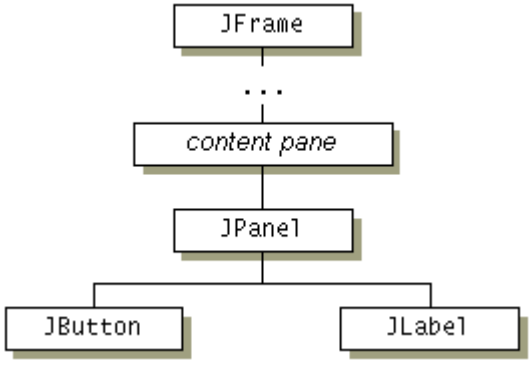
- Une fenêtre principale (avec boutons d'agrandissement, réduction, fermeture…) : *JFrame*
- Un plan rectangulaire : *JPanel*
- Un bouton : *Jbutton*
- Une zone texte : *Jlabel*

Le modèle par *Container*

La JFrame est un composant qui *contient* tous les autres.
Le JPanel est un *container* intermédiaire qui va se charger de contenir le label et le bouton.

Le JButton et le JLabel sont des composants atomiques.
Autres exemples : zone de texte éditable (JTextField), table (JTable), etc…

Diagramme de hiérarchie (contenance) – arborescence :



Remarque : tous les container de tête dans la hiérarchie (JFrame, JWindow…) contiennent un container intermédiaire qui se charge de contenir les fils (*contentPane*)

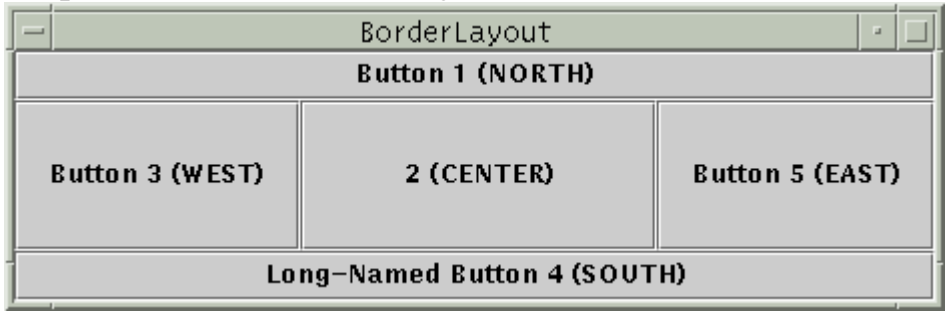
Le code de l'exemple :

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

public class Exemple1 {

  public static void main(String[] args) {

    JFrame frame = new JFrame(”exemple”);
    JButton button = new JButton(”clic clic”);
    JLabel label = new JLabel(”un petit texte”);
    JPanel pane = new JPanel();
    pane.add(button);
    pane.add(label);
    frame.getContentPane().add(pane, BorderLayout.CENTER);

  }
}
```

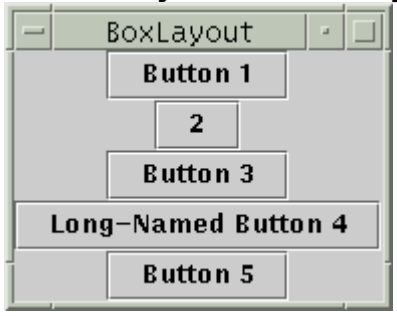
Agencer les composants avec les Layouts

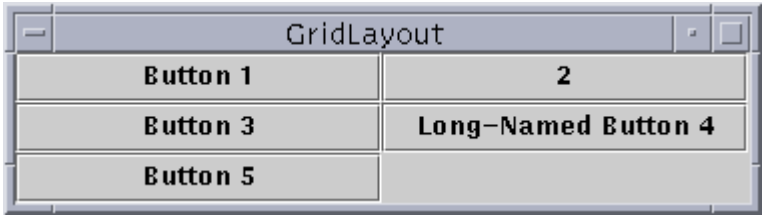
Le layout par défaut : *FlowLayout*

Le plus utilisé : *BorderLayout*



Autres layouts : *BoxLayout, GridLayout*



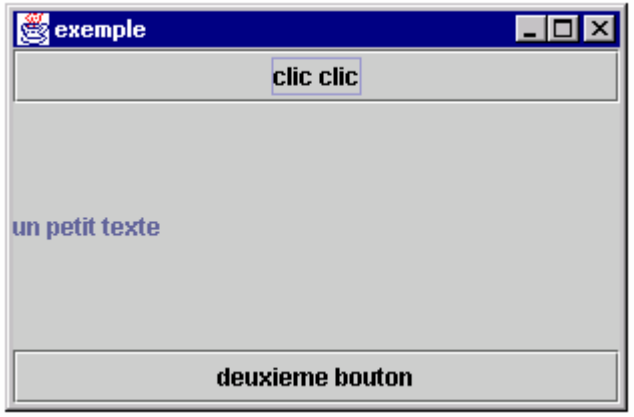


Utiliser le BorderLayout

⇒ Ajouter au panel *avec une contrainte*

frame.getContentPane().add(pane, BorderLayout.CENTER);

Exercice : modifier l'exemple pour avoir :



Remarque : définir le layout du JPanel avec *pane.setLayout(new BorderLayout())*

Solution :

```
JFrame frame = new JFrame("exemple");
JButton button = new JButton("clic clic");
JLabel label = new JLabel("un petit texte");
JButton button2 = new JButton("deuxieme bouton");
JPanel pane = new JPanel();
pane.setLayout(new BorderLayout());
pane.add(button, BorderLayout.NORTH);
pane.add(label, BorderLayout.CENTER);
pane.add(button2, BorderLayout.SOUTH);
frame.getContentPane().add(pane, BorderLayout.CENTER);
frame.show();
```

Utiliser le GridLayout

⇒ Créer le layout en le définissant :

```
pane.setLayout(new GridLayout(3,2));
```

Exercice : modifier l'exemple pour avoir :

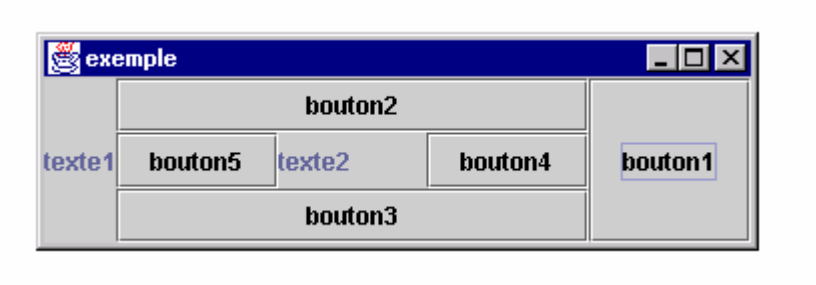
Remarque : l'ajout ne nécessite plus de contrainte : *pane.add(button);*

Solution :

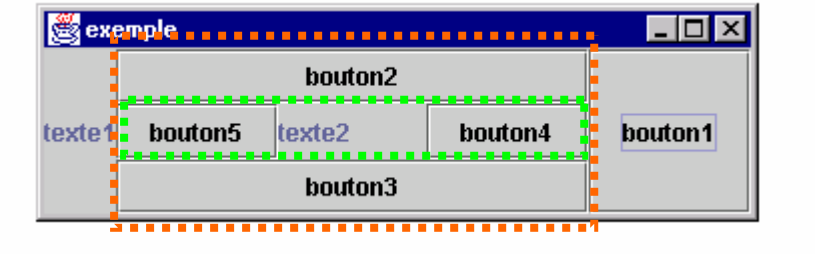
```
JFrame frame = new JFrame("exemple");
JButton button = new JButton("clic clic");
JLabel label = new JLabel("un petit texte");
JButton button2 = new JButton("deuxieme bouton");
JButton button3 = new JButton("troisieme bouton");
JButton button4 = new JButton("quatrieme bouton");
JPanel pane = new JPanel();
pane.setLayout(new GridLayout(3,2));
pane.add(button);
pane.add(label);
pane.add(button2);
pane.add(button3);
pane.add(button4);
frame.getContentPane().add(pane, BorderLayout.CENTER);
frame.show();
```

Comment agencer de façon plus riche ?

Exercice : Comment faire :



Remarque : utilisez plus d'un panel… exemple :

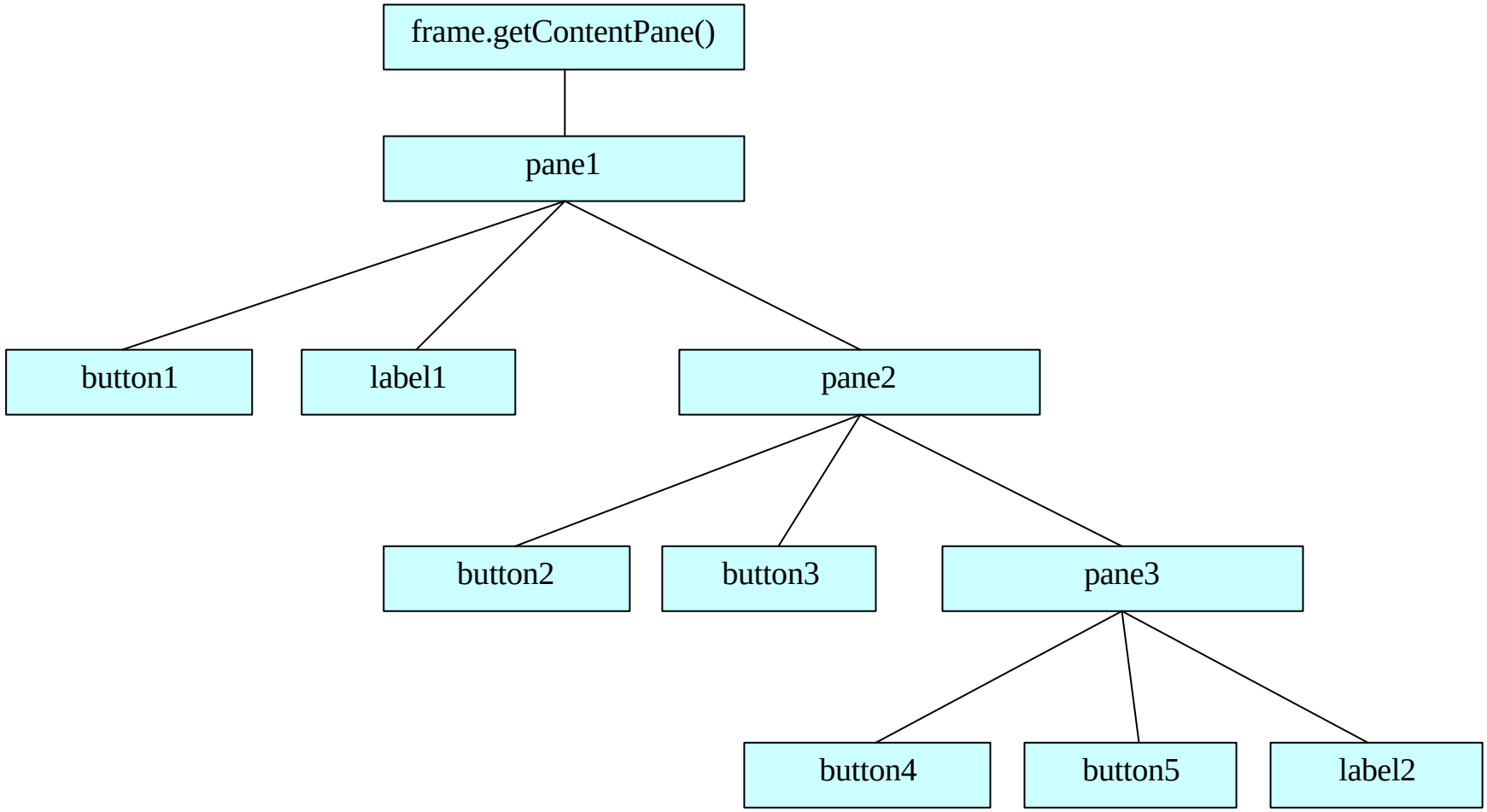


- panel2
- panel3

Solution possible :

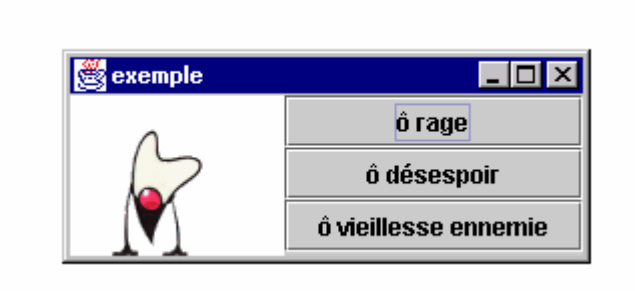
```
JFrame frame = new JFrame("exemple");

JButton button1 = new JButton("bouton1");
JLabel label1 = new JLabel("texte1");
JLabel label2 = new JLabel("texte2");
JButton button2 = new JButton("bouton2");
JButton button3 = new JButton("bouton3");
JButton button4 = new JButton("bouton4");
JButton button5 = new JButton("bouton5");

JPanel pane1 = new JPanel(new BorderLayout()); /* on peut spécifier le layout à la construction */
JPanel pane2 = new JPanel(new BorderLayout());
JPanel pane3 = new JPanel(new BorderLayout());

pane1.add(button1, BorderLayout.EAST);
pane1.add(label1, BorderLayout.WEST);
pane1.add(pane2, BorderLayout.CENTER);

pane2.add(button2, BorderLayout.NORTH);
pane2.add(button3, BorderLayout.SOUTH);
pane2.add(pane3, BorderLayout.CENTER);

pane3.add(button4, BorderLayout.EAST);
pane3.add(button5, BorderLayout.WEST);
pane3.add(label2, BorderLayout.CENTER);

frame.getContentPane().add(pane1, BorderLayout.CENTER);
frame.show();
```

Arborescence :



Exercices : de l'idée au code

Ecrire le code pour obtenir :



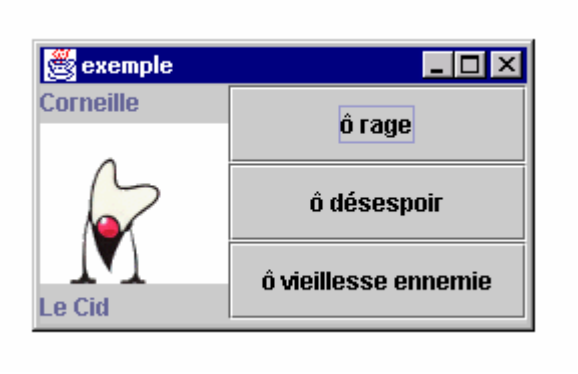
Remarque : pour créer un composant ”image” : *new JLabel(new ImageIcon(”duke.gif”))*

Solution possible :

```
JFrame frame = new JFrame("exemple");
JButton button1 = new JButton("ô rage");
JButton button2 = new JButton("ô désespoir");
JButton button3 = new JButton("ô vieillesse ennemie");

JLabel label1 = new JLabel(new ImageIcon("duke.gif"));

JPanel pane1 = new JPanel(new GridLayout(0, 1));
pane1.add(button1);
pane1.add(button2);
pane1.add(button3);
frame.getContentPane().add(pane1, BorderLayout.EAST);
frame.getContentPane().add(label1, BorderLayout.CENTER);
frame.show();
```

Remarque : une valeur 0 pour l'initialisation du GridLayout signifie "un nombre indéterminé" de composants.

Même exercice :



De même : (remarquez que Corneille et Le Cid sont centrés)

Remarque : Avez-vous remarqué le comportement du FlowLayout ?… il permet de centrer les composants…

Solutions possibles :

```
JFrame frame = new JFrame("exemple");
JButton button1 = new JButton("ô rage");
JButton button2 = new JButton("ô désespoir");
JButton button3 = new JButton("ô vieillesse ennemie");

JLabel label1 = new JLabel(new ImageIcon("duke.gif"));
JLabel label2 = new JLabel("Corneille");
JLabel label3 = new JLabel("Le Cid");

JPanel pane1 = new JPanel(new GridLayout(0, 1));
pane1.add(button1);
pane1.add(button2);
pane1.add(button3);

JPanel pane2 = new JPanel(new BorderLayout());
pane2.add(label1, BorderLayout.CENTER);
pane2.add(label2, BorderLayout.NORTH);
pane2.add(label3, BorderLayout.SOUTH);



frame.getContentPane().add(pane1, BorderLayout.EAST);
frame.getContentPane().add(pane2, BorderLayout.CENTER);
frame.show();
```

```
JFrame frame = new JFrame("exemple");
JButton button1 = new JButton("ô rage");
JButton button2 = new JButton("ô désespoir");
JButton button3 = new JButton("ô vieillesse ennemie");

JLabel label1 = new JLabel(new ImageIcon("duke.gif"));
JLabel label2 = new JLabel("Corneille");
JLabel label3 = new JLabel("Le Cid");

JPanel pane1 = new JPanel(new GridLayout(0, 1));
pane1.add(button1);
pane1.add(button2);
pane1.add(button3);

JPanel pane2 = new JPanel(new BorderLayout());
JPanel paneNorth = new JPanel(new FlowLayout());
JPanel paneSouth = new JPanel(new FlowLayout());
paneNorth.add(label2);
paneSouth.add(label3);
pane2.add(label1, BorderLayout.CENTER);
pane2.add(paneNorth, BorderLayout.NORTH);
pane2.add(paneSouth, BorderLayout.SOUTH);

frame.getContentPane().add(pane1, BorderLayout.EAST);
frame.getContentPane().add(pane2, BorderLayout.CENTER);
frame.show();
```

Réagir aux actions avec des *listeners*

Les composants Swing peuvent avertir d'une action effectuée sur eux-mêmes.

Pour être averti, il faut donc écrire un objet capable d'être averti.
Il faut pour cela **implémenter l'interface** "d'écoute" correspondant à l'événement, à l'action.
Par exemple, un JButton avertit les ActionListeners : il possède la méthode *addActionListener*.

```
class MyActionListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    System.out.println("clicked!!"); /* ecrit sur la sortie console */
  }
}

…

JButton button = new JButton("clic clic");
button.addActionListener(new MyActionListener());
```

Exemple complet :

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

public class Exemple8 {

public static void main(String[] args) {
  JFrame frame = new JFrame("exemple");
  JButton button = new JButton("clic clic");
  button.addActionListener(new MyActionListener());
  JLabel label = new JLabel("un petit texte");
  JPanel pane = new JPanel();
  pane.add(button);
  pane.add(label);
  frame.getContentPane().add(pane, BorderLayout.CENTER);
  frame.show();
}

  /* on peut écrire une class à l'intérieur d'une autre, */
  /* utilisée uniquement dans la classe englobante.      */
  static class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      System.out.println("clicked!!");
    }
  }
}
```

Exercice :

Modifier le programme suivant pour que le label affiche le nombre d'appuis sur le bouton.

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

public class Exemple9 {

  private JLabel label;

  public Exemple9() {
    JFrame frame = new JFrame("exemple");
    JButton button = new JButton("clic clic");
    button.addActionListener(new MyActionListener());
    label = new JLabel("0");
    JPanel pane = new JPanel();
    pane.add(button);
    pane.add(label);
    frame.getContentPane().add(pane, BorderLayout.CENTER);
    frame.show();
  }

  private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      //…
    }
  }

  public static void main(String[] args) {
    new Exemple9();
  }
}
```

Remarque : on peut modifier le texte d'un label avec *label.setText("...")*

Solution :

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

public class Exemple9 {

  private int count = 0;
  private JLabel label;

  public Exemple9() {
    JFrame frame = new JFrame("exemple");
    JButton button = new JButton("clic clic");
    button.addActionListener(new MyActionListener());
    label = new JLabel("0");
    JPanel pane = new JPanel();
    pane.add(button);
    pane.add(label);
    frame.getContentPane().add(pane, BorderLayout.CENTER);
    frame.show();
  }

  /* classe qui accède à count; count appartient à la classe englobante */
  private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      count++;
      label.setText(Integer.toString(count));
    }
  }

  public static void main(String[] args) {
    new Exemple9();
  }
}
```

Comment connaître les listeners associés aux composants ?

- avec une liste exhaustive (voir tutorial Sun)
- en repérant dans la documentation les méthodes "**addXXXListener**"

Exercice :

1) Trouvez le listener associé aux événements de fenêtre (fermeture, réduction…) pour JFrame
2) Modifier le programme précédent pour qu'il s'arrête lorsque l'on ferme la fenêtre

Remarque :
Pour quitter : *System.exit(0);*

Solution :

Il faut implémenter WindowListener

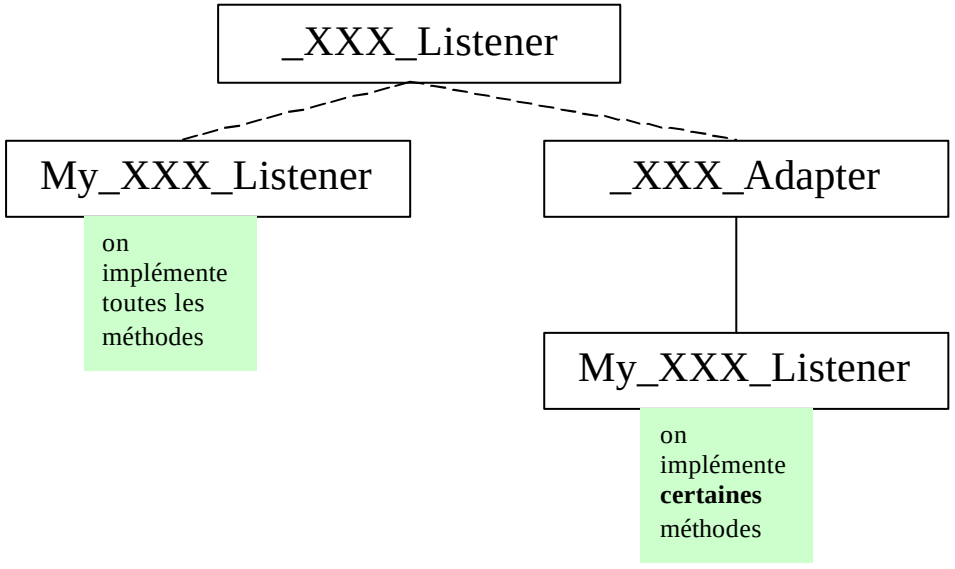
```
public Exemple10() {
    JFrame frame = new JFrame("exemple");
    frame.addWindowListener(new MyWindowListener());
    JButton button = new JButton("clic clic");
    ...
    frame.show();
  }

  private class MyActionListener implements ActionListener {
    ...
  }

  private class MyWindowListener implements WindowListener {
    public void windowActivated(WindowEvent event) {
    }
    public void windowDeactivated(WindowEvent event) {
    }
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
    public void windowClosed(WindowEvent event) {
    }
    public void windowIconified(WindowEvent event) {
    }
    public void windowDeiconified(WindowEvent event) {
    }
    public void windowOpened(WindowEvent event) {
    }
  }

...
}
```

Lorsque l'on utilise qu'une partie de l'interface d'écoute : les Adapters

Dans l'exemple précédent, seule la méthode "windowClosing" nous a été utile.
Mais il a fallu pourtant implémenter (avec rien!) les autres méthodes de l'interface.
Par facilité, on peut utiliser la classe WindowAdapter qui est le WindowListener le plus simple : il ne fait rien.
Il suffit alors de redéfinir par héritage la méthode voulue.

```
private class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
}
```

Schéma général :



Question :
Pourquoi n'existe-t-il pas de ActionAdapter ?

Utiliser les événements

Lors d'un click, d'une modification, etc., l'objet avertit ses listeners en envoyant un *event*.

Par exemple :
actionPerformed(ActionEvent event)

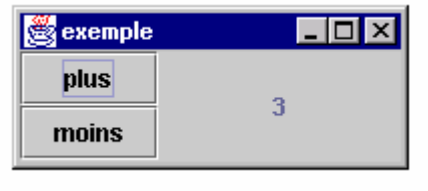
Cet event englobe des informations que l'on peut récupérer pour réagir en fonction de ces informations.

Par exemple, on peut "brancher" deux boutons sur le même listener et vouloir réagir différemment aux deux boutons.

```
button1 = new JButton();
button2 = new JButton();
ActionListener myActionListener = new ActionListener();
button1.addActionListener(myActionListener);
button2.addActionListener(myActionListener);
…
class MyActionListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    if(event.getSource() == button1) {
       // button1 clicked
    } else {
      // button2 clicked
    }
  }
}
```

Exercice :

Modifier le programme précédent afin qu'un bouton incrémente le compteur et qu'un autre bouton le décrémente.



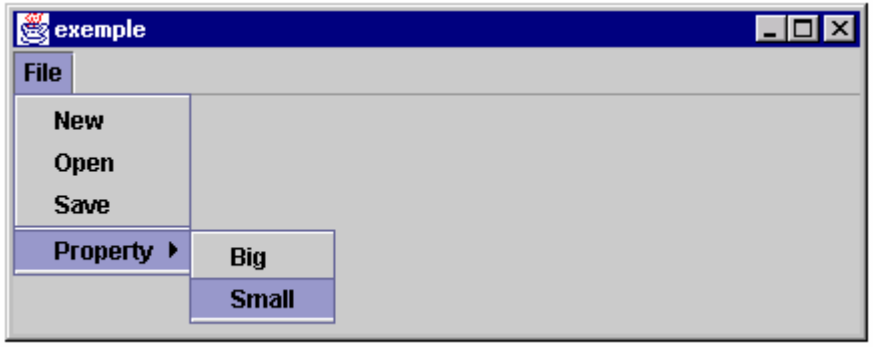
Solution :

```
public class Exemple11 {
  private int count = 0;
  private JLabel label;
  private JButton buttonPlus;
  private JButton buttonMoins;

  public Exemple11() {
    JFrame frame = new JFrame("exemple");
    frame.addWindowListener(new MyWindowListener());
    buttonPlus = new JButton("plus");
    buttonMoins = new JButton("moins");
    buttonPlus.addActionListener(new MyActionListener());
    buttonMoins.addActionListener(new MyActionListener());
    label = new JLabel("0", SwingConstants.CENTER);
    JPanel pane = new JPanel(new BorderLayout());
    JPanel buttons = new JPanel(new GridLayout(0, 1));
    buttons.add(buttonPlus);
    buttons.add(buttonMoins);
    pane.add(buttons, BorderLayout.WEST);
    pane.add(label, BorderLayout.CENTER);
    frame.getContentPane().add(pane, BorderLayout.CENTER);
    frame.show();
  }

  private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      if(event.getSource() == buttonPlus) {
        count++;
      } else {
        count--;
      }
      label.setText(Integer.toString(count));
    }
  }
...
}
```

Plus de fonctionnalités !

Dans une JFrame, on peut définir sa barre de menu avec JMenuBar, JMenu et JMenuItem.

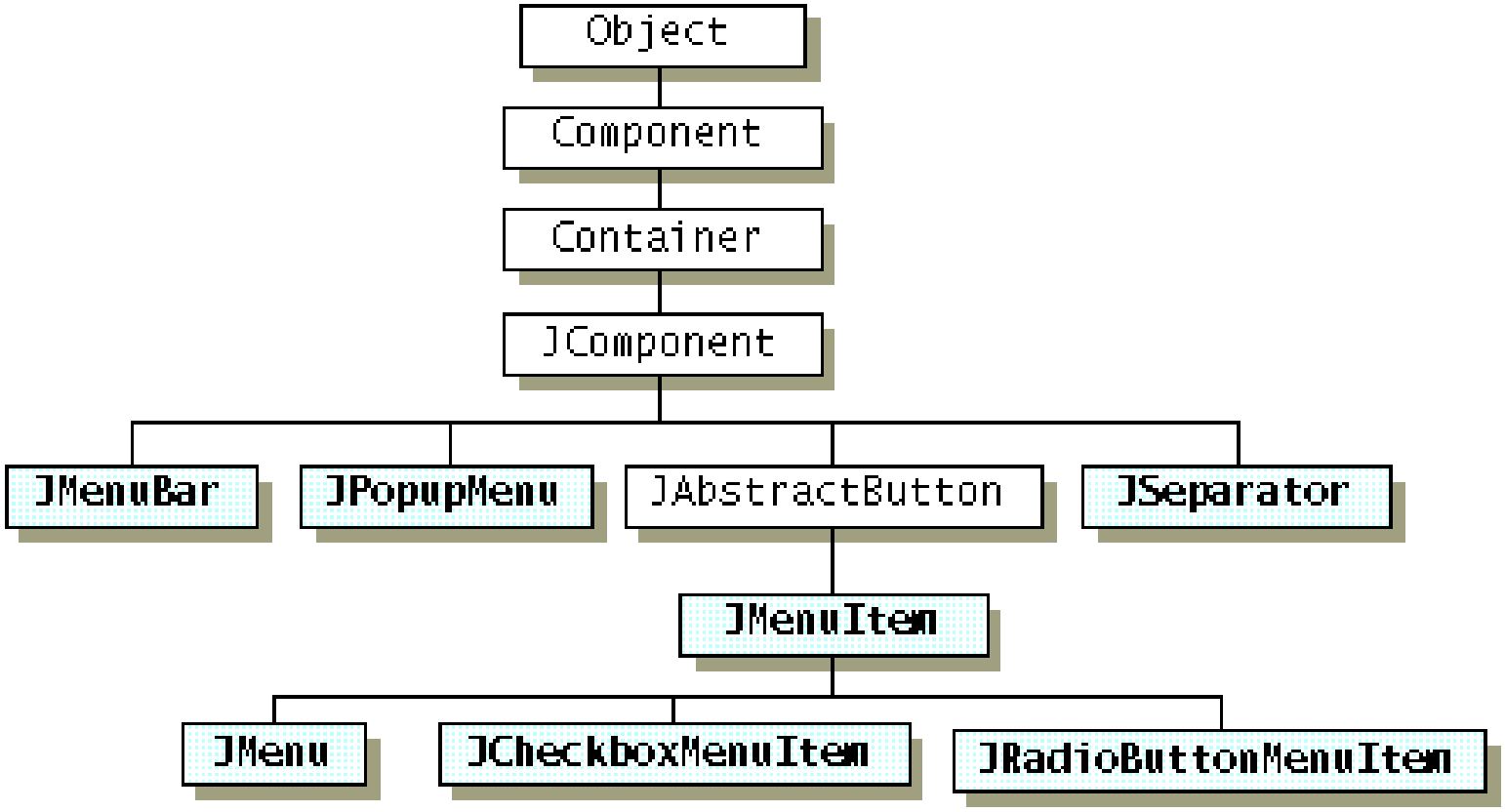
Exemple :



Programme :

```
JFrame frame = new JFrame("exemple");

JMenuBar menuBar = new JMenuBar();
JMenu menu = new JMenu("File");
JMenu submenu = new JMenu("Property");

menu.add(new JMenuItem("New"));
menu.add(new JMenuItem("Open"));
menu.add(new JMenuItem("Save"));
menu.add(new JSeparator());
menu.add(submenu);

submenu.add(new JMenuItem("Big"));
submenu.add(new JMenuItem("Small"));

menuBar.add(menu);

frame.setJMenuBar(menuBar);
frame.show();
```

Hiérarchie des composants :



Remarquez que ce schéma exprime qu'un JMenu *est un* JMenuItem : lorsqu'un menu est employé en tant qu'item, il a le comportement d'un sous-menu.

Exercice :

1) Trouvez comment réagir à la sélection d'un item du menu.
2) Modifier l'exemple précédent pour qu'un label de la fenêtre affiche le dernier item selectionné.

3) Modifier à nouveau votre programme en remplaçant le label par une zone de texte qui affiche l'historique des sélections des items.

Remarque :
- il vous faudra regarder la documentation de JMenuItem notamment.
- pour le point 3, il vous faudra regarder JTextArea.

Solution 2) :

```
public class Exemple13 {
  private JMenuItem item1, item2, item3, item4, item5;
  private JLabel label;

  public Exemple13() {
    JFrame frame = new JFrame("exemple");
    frame.addWindowListener(new MyWindowListener());

    JMenuBar menuBar = new JMenuBar();
    JMenu menu = new JMenu("File");
    JMenu submenu = new JMenu("Property");

    item1 = new JMenuItem("New");
    item2 = new JMenuItem("Open");
    item3 = new JMenuItem("Save");
    item4 = new JMenuItem("Big");
    item5 = new JMenuItem("Small");

    MyActionListener listener = new MyActionListener();
    item1.addActionListener(listener);
    item2.addActionListener(listener);
    item3.addActionListener(listener);
    item4.addActionListener(listener);
    item5.addActionListener(listener);

    menu.add(item1);
    menu.add(item2);
    menu.add(item3);
    menu.add(new JSeparator());
    menu.add(submenu);

    submenu.add(item4);
    submenu.add(item5);

    menuBar.add(menu);
```

```
    label = new JLabel("-");

    frame.setJMenuBar(menuBar);
    frame.getContentPane().add(label);
    frame.show();
  }

  private class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
  }

  private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      label.setText(((JMenuItem)event.getSource()).getText());
    }
  }

  public static void main(String[] args) {
    new Exemple13();
  }
}
```

Solution 3) :

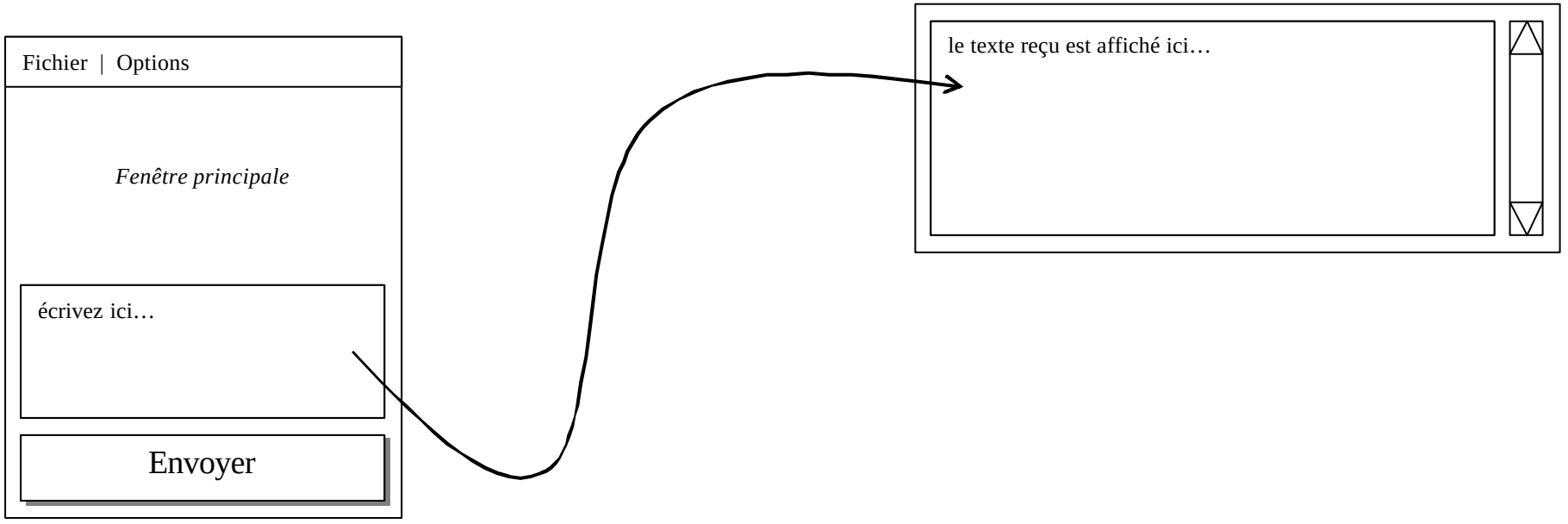
```
private JMenuItem item1, item2, item3, item4, item5;
private JTextArea text;


...

text = new JTextArea();

frame.setJMenuBar(menuBar);
frame.getContentPane().add(text);
frame.show();

...

private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      text.append(((JMenuItem)event.getSource()).getText() + "\n");
    }
}
```

Exercice

Il s'agit de créer un programme qui possède une fenêtre principale où l'on tape du texte, et une fenêtre secondaire où le texte est envoyé, ligne par ligne.

La fenêtre principale contiendra une zone de texte éditable où l'utilisateur tapera la ligne à envoyer, ainsi qu'un bouton d'envoi.
Elle permettra aussi de fermer le programme.

La fenêtre secondaire ne possèdera pas de barre de menu (regarder JWindow). Elle contiendra une zone de texte non éditable où le texte reçu sera écrit.



Remarques :

- pour régler la taille de la fenêtre : *setSize(200, 200)*
- pour positionner la fenêtre : *setLocation(400, 0)*
- pour créer un panel scrollable : *scroll = new JScrollPane(panel); panel2.add(scroll);*

Solution :

```
public class Exemple15 {

  private JTextArea text, area;

  public Exemple15() {
    JFrame frame = new JFrame("exemple");
    frame.addWindowListener(new MyWindowListener());

    text = new JTextArea();
    text.setEditable(false);

    JWindow window = new JWindow(frame);
    JPanel panel = new JPanel(new BorderLayout());
    panel.add(text, BorderLayout.CENTER);
    JScrollPane scroll = new JScrollPane(panel);
    window.getContentPane().add(scroll);

    JPanel panel2 = new JPanel(new BorderLayout());
    area = new JTextArea();
    panel2.add(area, BorderLayout.CENTER);

    JPanel panel3 = new JPanel(new BorderLayout());
    JButton button = new JButton("Envoyer");
    panel3.add(new JScrollPane(panel2), BorderLayout.CENTER);
    panel3.add(button, BorderLayout.SOUTH);

    button.addActionListener(new MyActionListener());

    frame.getContentPane().add(panel3);
    frame.setSize(200, 200);
    window.setSize(200, 200);
    frame.setLocation(0, 0);
    window.setLocation(400, 0);
    frame.show();
    window.show();
  }
```

```
  private class MyWindowListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      System.exit(0);
    }
  }

  private class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      text.append(area.getText() + "\n");
      area.setText("");
    }
  }

  public static void main(String[] args) {
    new Exemple15();
  }
}
```