TUTO : SWING -- INTRODUCTION

Dès sa naissance, Java a disposé d'une bibliothèque de composants graphiques : AWT (Abstract Windowing Toolkit). Ces composants étaient basés sur des contrôles natifs du système d'exploitation local (Windows, Mac, Unix) rendant les applications Java AWT assez performantes. Malheureusement, cette approche posait quelques difficultés en terme de portabilité et d'extensibilité.

Depuis **Java 2** (JDK 1.2.2), Sun incite les développeurs Java à utiliser **Swing** en remplacement de AWT. Cette nouvelle bibliothèque est beaucoup plus riche que la précédente, et offre une bien meilleure portabilité . **Swing** est une bibliothèque graphique pour le langage de programmation Java, faisant partie du package Java Foundation Classes (JFC), inclus dans J2SE. Swing constitue l'une des principales évolutions apportées par Java 2 par rapport aux versions antérieures. Swing offre la possibilité de créer des interfaces graphiques identiques quelque soit le système d'exploitation.
Pour créer des interfaces graphiques, il est nécessaire de connaître et de maîtriser certains objets.

JFrame - Les fenêtres

Les JFrame sont l'équivalent des fenêtres. Elles ont un titre, une dimension, un aspect et des éléments graphique affichés à l'intérieur. Les JFrame font partie du package **javax.swing.**

```
package org.ldv.slam;

import javax.swing.*;
public class TestSwing
{
   public static void main(String args[]) {
         JFrame  f=new JFrame("Hello World!!");
        f.setVisible(true);

   }
}
```

Tapez ce code dans"Test.java" puis compilez et exécuter ce petit programme
Cette fenêtre est assez minuscule, si vous la re-dimensionnez a la main, vous verrez apparaître son titre et son contenu vide. Comme on n'a pas spécifié de dimension pour notre JFrame, Swing ouvre une fenêtre avec une dimension par défaut (ici 0x0).

Par défaut, une JFrame est toujours invisible, c'est-à-dire que la fenêtre est créée mais jamais affichée. C'est pour cela que l'on rajoute la ligne *f.setVisible(true);* qui permet de rendre visible la fenêtre. Lorsque vous fermez la fenêtre,vous constaterez que Java ne rend pas la main au système, le programme java tourne toujours. Lorsque vous créez une JFrame, un Thread c'est-à-dire un programme d'arrière plan est créé. Par défaut, Swing ne tue pas le processus lorsque l'on ferme la JFrame avec la souris.
Notre deuxième petit programme va nous ouvrir une fenêtre avec une dimension par défaut et va quitter le logiciel lorsque l'on ferme la fenêtre.

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

public class TestSwing
{
   public static void main(String args[]) {
         JFrame  f=new JFrame("Hello World!!");
         f.setSize(new Dimension(500,300));
         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         f.setVisible(true);
   }
}
```

On a rajouté que deux lignes :

- **setSize** permet de spécifier la dimension de la fenêtre
- **setDefaultCloseOperation** permet de définir l'opération par défaut lorsqu'on ferme la fenêtre. Le paramètre **JFrame.EXIT_ON_CLOSE** spécifie que l'on quitte le programme.
-

JPanel et les layouts

Pour ajouter des objets à une JFrame, on a besoin de JPanel. Un JPanel est en quelque sorte une boite dans laquelle on peut placer des composants de l'interface graphique. Un JPanel sert uniquement à stocker les objets

Les composants de l'interface graphique sont des classes qui héritent de la classe JComponent. Il en existe déjà dans Swing : les labels, les zones de texte éditables, les scrollbars, les boutons, les tables etc… et il est bien sur possible de créer ses propres composants. Voici une petite liste des noms des classes des composants les plus utilisés :

JLabel label = new JLabel("un texte") --> pour le texte
JTextField text= new JTextField() --> un champ texte éditable
JButton bouton=new JButton("Quitter") --> un bouton
JComboBox etc....

Pour commencer, nous allons ajouter à notre exemple précédent, un texte centré :

On va créer un JPanel pour stocker les éléments.

```
JPanel panel=new JPanel();
```

Le texte sera un JLabel. On créé simplement cet objet en faisant :

```
JLabel label=new  JLabel("Bonjour tout le monde");
```

Ensuite, il faut ajouter le JLabel au panel :

```
panel.add(label);
```

Vous devez également définir un LayoutManager, c'est à dire un gestionnaire de positionnement. Nous allons utiliser un FlowLayout qui a la particularité de pouvoir placer notre JLabel centré sur la largeur de la fenêtre et attacher le LayoutManager auPanel :

```
panel.setLayout(new FlowLayout(FlowLayout.CENTER));
```

Le paramètre FlowLayout.CENTER spécifie l'alignement que l'on désire, ici on veut centrer.

Pour finir, une JFrame lors de sa création, créé déjà un JPanel, ici nous avons créé notre propre JPanel panel, il ne faut pas oublier de définir le gestionnaire de positionnement de la JFrame et ajouter le JPanel a la fenêtre :

```
f.setLayout(new FlowLayout());
f.add(panel);
```

Voici le programme complet :

```
import javax.swing.*;
import java.awt.*;
public class TestSwing {
	public static void main(String[] args) {
		JFrame frame= new JFrame("test");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		JPanel panel=new JPanel();
		JLabel label=new  JLabel("Bonjour tout le monde");
		panel.add(label);
		panel.setLayout(new FlowLayout(FlowLayout.CENTER));

		frame.setLayout(new FlowLayout());
		frame.add(panel);

		frame.setSize(500,400);
		frame.setVisible(true);
	}
}
```

Cette technique n'est pas l'idéale car la méthode **public static void main(String[] args)** ne devrait pas avoir à gérer la représentation graphique de l'application. L'idée est de spécialiser la classe de base JFrame en une classe qui contient les composants dont vous avez besoin. Vous allons donc créer une classe qui se spécialisera dans la représentation graphique et qui héritera de la classe de base JFrame.

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

public class TestSwing extends JFrame {

	public Swing3() {

		super("test");
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setSize(500,400);

		JPanel panel=new JPanel();
		JLabel label=new  JLabel("Bonjour tout le monde");
		panel.add(label);
		panel.setLayout(new FlowLayout(FlowLayout.CENTER));

		this.setLayout(new FlowLayout());
		this.add(panel);

	}

       public static void main(String[] args) {
		JFrame frame=new TestSwing();
		frame.setVisible(true);

	}
}
```

Remarque :

- le mot clé **extends** signifie que la classe hérite de la classe JFrame.
- le mot clé **super** indique que l'on fait appel au constructeur parent, c'est à dire JFrame. Dans ce cas, vous n'avez plus à indiquer pour créer un nouvel objet JFrame frame= new JFrame("test"); mais simplement à appeler le constructeur de la classe mère : **super(''test''). (voir cours sur l'héritage)**
- le mot clé **this** fait référence à l'objet en cours d'exécution, dans cet exemple il remplace l'objet frame. Son usage est facultatif, mais fortement conseillé.

SWING - LES GESTIONNAIRES DE POSITIONNEMENT

Nous allons, dans cette partie étudier un ensemble de classes chargées de résoudre les problèmes de positionnement de vos composants graphiques : les layout manager.
Notez une chose importante : même si vous êtes en train de travailler avec Swing, vous vous devez d'importer le package *java.awt.** pour pouvoir manipuler les gestionnaires de positionnement.
Il existe plusieurs gestionnaires de positionnement, les principaux sont les suivants :

Le FlowLayout

Ce premier gestionnaire est certainement le plus utilisé et le plus simple à manipuler. Si vous l'affectez à un conteneur (fenêtre ou panel), celui-ci va tenter de mettre le plus de composants possible sur une première ligne. Dès que la taille du conteneur ne permet plus d'insérer un nouveau composant, le layout utilise alors un seconde ligne, et ainsi de suite jusqu'à arriver au dernier composant.

Vous pouvez de plus configurer votre layout en spécifiant si vous souhaitez que chaque ligne de composant soit alignée à gauche, à droite ou, si vous préférez, centrée. Cette configuration peut être mise en place soit à la construction de l'objet de positionnement, soit par la suite en utilisant la méthode setAlignment. Des attributs de la classe FlowLayout sont préfédinis : FlowLayout.LEFT, FlowLayout.CENTER, FlowLayout.RIGHT .

Le BorderLayout

Ce second gestionnaire permet d'opérer une division de l'espace utilisable au sein d'un conteneur. Par opposition au FlowLayout, le nombre de zones utilisable par cette stratégie est figé : un BorderLayout divise l'espace utilisable en cinq zones :

- `BorderLayout.NORTH`
- `BorderLayout.SOUTH`
- `BorderLayout.WEST`
- `BorderLayout.EAST`
- `BorderLayout.CENTER`

Dans chacune de ces zones, vous ne pouvez placer qu'un unique composant (mais rien n'empêche que certains de ces composants soient des containers comme des JPanel).

Le GridLayout

Nous allons maintenant parler d'une autre stratégie de placement. Celle-ci permet de disposer vos composants dans une grille constituée de lignes et de colonnes. Le but de cette stratégie est de découper l'espace d'un conteneur en une grille. Pour ce faire, le constructeur de la classe GridLayout peut accepter deux paramètres : le nombre de lignes et le nombre de colonnes.

- panelGauche.setLayout(new GridLayout(4,1)); // 4 lignes et 1 colonnes
- panelGauche.setPreferredSize(new Dimension(180,120)); // fixe la nouvelle dimension du panelGauche à 180pixels de large et 25 pixels de haut.

GESTION DES ÉVÈNEMENTS

Pour attribuer une action à un bouton, il va falloir lui ajouter un listener, un écouteur d'action. En java, un écouteur d'action est un ActionListener. Il faudra donc définir qui va être l'écouteur de notre bouton, dans notre cas, on va dire que c'est la fenêtre elle-même, qui va écouter notre bouton pour ne rien compliquer. Car on peut aussi créer une autre classe qui ferait office d'écouteur. Pour qu'une classe puisse être « écouteuse » d'action, il faut qu'elle implémente, l'interface ActionListener. Pour cela, il vous suffira d'ajouter dans le signature de la classe implements ActionListener. Ainsi, la signature de notre classe deviendra :

```
public class TestBouton extends JFrame implements ActionListener{
```

Sans oublier d'importer ActionListener et ActionEvent :

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

Si vous travaillez avec un EDI tel qu'eclise , vous verrez que cette ligne, vous mettra des erreurs. Car, quand on implémente une interface, il faut qu'on définisse ces différentes méthodes, dans notre cas, ActionListener, possède seulement la méthode actionPerformed(ActionEvent e) qui va être appelée à chaque fois que l'écouteur reçoit un événement. On va donc ajouter cette méthode dans notre classe : **(voir votre cours sur les interfaces)**

```
public void actionPerformed(ActionEvent e) {

        }
```

Ensuite, on va ajouter l'écouteur à notre bouton :

```
bouton1.addActionListener(this);//On ajoute la fenêtre en tant qu'écouteur du bouton
```

Avec ce code, il n'y a toujours rien qui se passe, mais c'est normal, puisque l'on ne fait rien dans la méthode actionPerformed.
Maintenant, il va falloir définir une action dans le listener pour notre bouton. On va donc mettre cette action dans la méthode actionPerformed. Dans notre cas, comme on n'a qu'un seul bouton qui est écouté par la méthode, on pourrait se permettre de mettre directement l'action dans la méthode, mais pour prendre de bonnes habitudes, on va d'abord contrôler que l'action émane bien de notre bouton. Pour savoir d'où vient l'action, on dispose de l'événement passé en paramètre de la méthode actionPerformed, cet événement possède une méthode getSource(), qui va nous dire d'où vient l'action.

Si par exemple, on veut qu'à chaque clic sur le bouton, ça affiche un nombre dans notre label et que ce nombre augmente à chaque clic, il va falloir faire quelque chose du genre :

```
public void actionPerformed(ActionEvent e) {
      if(e.getSource() == bouton1){                          //Si l'action émane bien du bouton
           nombre++;                                         //On incrémente nombre de 1
           texte.setText("Vous avez cliqué " + nombre + " fois sur le bouton");
       }
}
```

Ainsi, avec cette méthode, dès que vous « cliquerez », le texte de bienvenue s'effacera, pour laisser place à un 1. Et ensuite, à chaque clic, le nombre augmentera.

DEMANDER DU TEXTE À L'UTILISATEUR

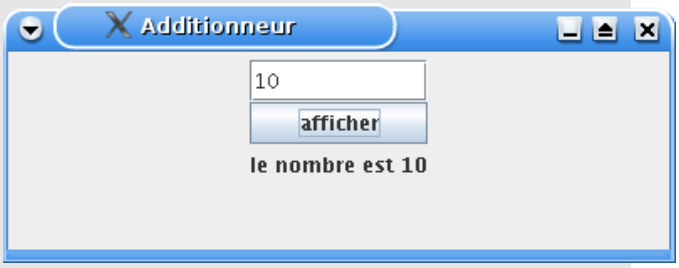
Maintenant que vous savez afficher du texte à l'écran, et employer des boutons pour créer un événement. On va apprendre à demander à l'utilisateur du texte. Pour cela, il existe plusieurs composants :

- Le JTextField : très basique, il permet seulement d'entrer un texte sur une seule ligne. C'est sur celui-là que nous allons nous étendre maintenant.
- Le JTextArea : Il permet d'entrer un texte complet sur plusieurs lignes.
- Le JEditorPane : Très complet, vous pouvez modifier la police, la taille, la couleur de votre texte. Il permet même d'afficher des pages HTML.

Pour récupérer le texte entrée, il faut employer la méthode getText(). Sinon, pour employer un JTextField, c'est à peu près de la même manière que pour un autre composant. On va employer les méthodes setText() pour définir un nouveau texte et setPreferredSize comme sur un JLabel.

```
public class Addition extends JFrame implements ActionListener{
        private JTextField text=null;
        private JLabel label=null;
        private JButton bouton=null;

        public Addition(){
                this.setTitle("Additionneur"); //On donne un titre à l'application
                this.setSize(400,150); //On donne une taille à notre fenêtre
                this.setLocationRelativeTo(null); //On centre la fenêtre sur l'écran
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setLayout(new FlowLayout());//on implémente un gestionnaire de positionnement

                JPanel pan = new JPanel();
                label=new JLabel("");
                text=new JTextField();
                bouton=new Jbutton("afficher");
                bouton.addActionListener(this); //On ajoute la fenêtre en tant qu'écouteur du bouton
                pan.setLayout(new GridLayout(3,1));//on implémente un gestionnaire de positionnement
                pan.add(text); // on ajoute le JTextField text au JPanel pan
                pan.add(bouton);
                pan.add(label);
```

```
                this.add(pan); // on ajoute le JPanel pan a la fenêtre
        }


        public void actionPerformed(ActionEvent e){
                if(e.getSource()==bouton){
                        String a=text.getText();
                        label.setText("le nombre est "+a);
                }


        }

        public static void main(String[] args){
                JFrame gui = new Addition();
                gui.setVisible(true);
        }
}
```



APPLICATION

Avec Eclipse créer un nouveau projet TutoSwing (File --> new --> Java Projetc --> Project name)

Télécharger les programmes suivants :

- TestSwing.java
- TestFlow.java
- TestBorder.java
- TestGrid.java
- Addition.java

Ces programmes sont disponibles à l'adresse suivante : http://guyonst.free.fr/tp/java/tp6 .

Ne copier pas ces programmes dans votre projet, pour les utiliser avec eclipse vous devez les importer en suivant la démarche suivantes :

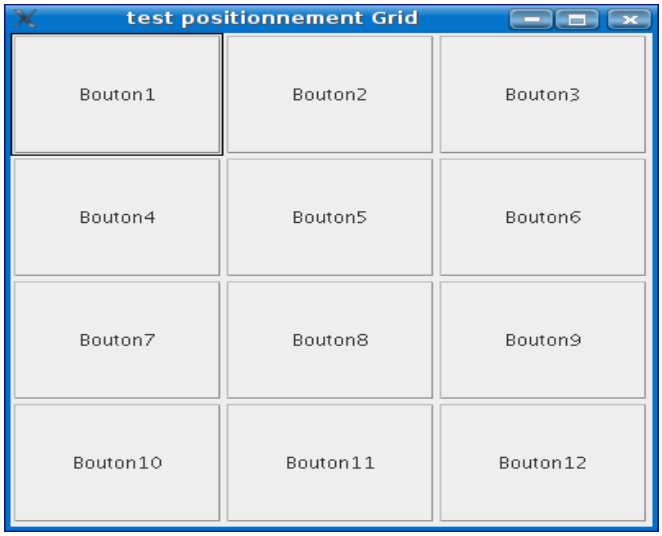
- créer un package org.ldv.slam
- dans votre projet, placez vous dans le répertoire **src/org/ldv/slam**, puis clic droit menu **import**
- type filter text --- dans l'onglet **general** choisir **file system** puis next
- aller dans le répertoire dans lequel vous avez enregistré les différents programmes puis valider
- ensuite sélectionner les programmes java que vous souhaitez mettre dans ce projet

1 - Analyser et exécuter ces différents programmes.

2 – Dans le programme TestSwing, ajouter un bouton permettant de quitter le programme : System.exit(0);

3 – Analyser le programme TestBorder.java. Commenter les lignes NORTH et SOUTH.

4 – Analyser le programme TestGrid.java. Modifier le programme pour obtenir la fenêtre suivante.



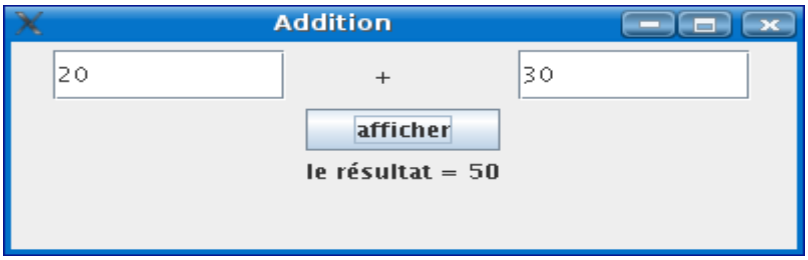
5 – Analyser le programme TestFlow.java . Essayer d'aggrandir la fenêtre. Modifier le programme pour obtenir la fenêtre suivante.

Vous pouvez déclarer et initialiser une constante :

```
static final String lMin = "abcdefghijklmnopqrstuvwxyz";
```

Vous utiliserez les méthodes length() et charAt(int i) de la class String pour initialiser les boutons.

6 – Analyser le programme Addition.java. Modifier ce programme pour obtenir le résultat suivant :



Remarque : la méthode getText() de la classe JTextField retourne une variable de type String. Il faut alors utiliser des méthodes qui permettent de convertir les données. Exemple :

```
String sNombre = text.getText() ;
int nombre = Integer.parseInt(sNombre) ;
```

7 – Plus dur : Modifier votre programme TestFlow.java, pour que le programme indique sur quel bouton on a cliqué

8 - Réaliser l'interface graphique suivante :