Université de Bouira
Département d'Informatique
Module : Les Interactions Homme Machine
Niveau 3 info

Série TP 4

JPanel, JButton, et Event Handlers

Etapes à suivre

I. Top-Level Containers - JFrame

- Créer un nouveau projet Java sous le nom *IHMTP4*.
- Créer dans celui-ci une classe *MyJFrame* héritant de JFrame (extends JFrame).
- Pour configurer l'état initial de la fenêtre créée, ajouter une méthode *initJFrame* et appeler celle-ci depuis un constructeur. Utiliser les méthodes suivantes pour l'initialisation :
  - ✓ setTitle(String title)
  - ✓ setSize(int width, int height)
  - ✓ setLocationRelativeTo(null)
  - ✓ setResizable(booblean resizable)
  - ✓ setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
- Ajouter une classe main nommée *TestJFrame*. Créer dans cette dernière une instance de la classe *MyJFrame* qui devrait s'exécuter dans l'Event Dispatch Thread (en utilisant SwingUtilities).
- Pour afficher cette instance, utiliser la méthode *setVisible*(true).

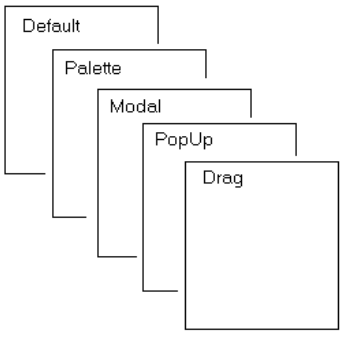
II. JLayeredPane

JLayeredPane est un container qui permet le positionnement des composants en profondeur (z-order). La profondeur de chaque composant est exprimée par un objet *Integer*. Leur affichage est par ordre croissant. Il ne possède pas de layout manager donc le positionnement de ses composants se fait avec la méthode *setBounds*( ).

- Ajouter ce code à la méthode *initJFrame* puis exécuter :

```
JLabel label = new JLabel();
label.setBounds(100, 100, 200, 100);
label.setOpaque(true);
label.setBackground(Color.cyan);
this.getLayerdPane().add(label, new Integer(2));


JLabel sublabel = new JLabel();
sublabel.setBounds(80, 80, 180, 80);
sublabel.setOpaque(true);
sublabel.setBackground(Color.yellow);
this.getLayerdPane().add(sublabel, new Integer(1));
```



III. Le container JPanel

Swing offre différents container lightweight pour contenir et ajouter des composants à une JFrame. Parmi eux : le 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 JComponents, il est secondé par un LayoutManager pour les placer au bon endroit dans une fenêtre. FlowLayout est son manager par défaut.

- Créer un **JPanel** dans la méthode *initJFrame* comme suit :

```
JPanel panel = new JPanel() ;
this.setContentPane(panel) ;
```

- Ajouter à *panel* un composant JLabel, nommé *label*, avec le texte : Social Media.

```
JLabel label = new JLabel("Social media") ;
panel.add(label);
```

- Ajouter au JPanel un composant **JTree** avec un nœud père et trois nœuds fils comme suit:

```
DefaultMutableTreeNode root = new DefaultMutableTreeNode("Exple");

DefaultMutableTreeNode twitter = new
                                  DefaultMutableTreeNode("Twitter");
DefaultMutableTreeNode facebook = new
                            DefaultMutableTreeNode("Facebook");

facebook.add(new DefaultMutableTreeNode("Messenger");

DefaultMutableTreeNode instagram = new
                            DefaultMutableTreeNode("Instagram");

root.add(twitter);
root.add(facebook);
root.add(instagram);


JTree tree = new JTree(root);
panel.add(tree);
```

Il faut noter, qu'on peut aussi placer d'autres JPanel dans un JPanel. Chaque sous panel ayant son propre LayoutManager.

- Appliquer le gestionnaire de positionnement BorderLayout au JPanel *panel*.

```
panel.setLayout(new BorderLayout());
```

- Créer 2 sous JPanel, *panel1* et *panel2,* et les ajouter à *panel* en utilisant : *panel.add( ).*
- Ajouter une bordure *MatteBorder* rouge aux deux JPanel *panel1* et *panel2*.

```
JPanel panel1 = new JPanel();
panel.add(panel1, BorderLayout.NORTH);
panel1.setBorder(BorderFactory.createMatteBorder(2, 2, 2, 2,
Color.red));

JPanel panel2 = new JPanel();
panel.add(panel2, BorderLayout.SOUTH);
panel2.setBorder(BorderFactory.createMatteBorder(2, 2, 2, 2,
Color.red));
```

- Ajouter le JLabel *label* à *panel1*, puis ajouter le JTree *tree* à *panel2*.

```
panel1.add(label) ;
panel2.add(tree) ;
```

- Changer la police du JLabel *label* comme suit :

```
label.setFont(new Font("Serif", Font.PLAIN, 14));
```

- Afficher votre nouvelle fenêtre. Qu'est-ce que vous remarquez ?

IV. Les JButton et les Event Handlers

La principale fonctionnalité d'une application graphique est la programmation événementielle, c'est-à-dire le fait que l'utilisateur puisse déclencher des événements et réagir à ce qui se passe dans la fenêtre. Pour que l'utilisateur puisse interagir avec le programme, on dispose par exemple, des boutons. Ce sont des éléments graphiques sur lesquels l'utilisateur peut cliquer pour déclencher une action. Le bouton ne fait rien tant que l'utilisateur n'a pas cliqué dessus.[1]

Le composant pour créer un bouton dans Swing est le **JButton**.

- Réinitialiser la méthode *initJFrame* à son état Etape 1.
- Créer un JPanel, *panel*, et l'appliquer comme ContentPane de la fenêtre.
- Créer deux JButton : un bouton "OK" nommé button1, et un autre "Annuler" nommé button2. Les ajouter au JPanel créé.
- Afficher la fenêtre. Cliquer sur l'un des deux boutons.

```
JPanel panel = new JPanel() ;
this.setContentPane(panel) ;

JButton button1 = new JButton("ok") ;
panel.add(button1) ;
JButton button2 = new JButton("Annuler") ;
panel.add(button2) ;
```

[1] Création interface graphique avec Swing : les bases - http://baptiste-wicht.developpez.com/

En cliquant sur n'importe quel bouton, rien ne se passe car jusque-là aucune action ne leur a été attribuée.

Attribuer une action à un bouton

Attribuer une action à un bouton revient à lui indiquer ce qui va devoir se passer en cas de clic de la part de l'utilisateur.

Le fonctionnement des actions en Swing suit toujours le même principe, celui des **événements** et des **écouteurs (listeners)**. Donc lors du clic sur un bouton (**source**), on aura un événement (ActionEvent dans ce cas) qui sera envoyé à tous les écouteurs du bouton. Par conséquent, si on veut savoir quand se passe l'action d'un bouton, il faut mettre un écouteur sur ce bouton et réagir en cas d'événement.

L'une des manières de faire avec un bouton est d'utiliser une classe qui *implémente ActionListener* et écouter le bouton. Il faut avoir une classe implémentant ActionListener et l'ajouter comme écouteur aux boutons. Ensuite, à chaque clic, la méthode *actionPerformed* va être appelée et il faudra tester le bouton qui a envoyé l'événement.

- Pour indiquer que la classe *MyJFrame* est un listener d'actions, implémenter l'interface *ActionListener* et redéfinisser la méthode *actionPerformed*.

```
public class MyJFrameTP4 extends JFrame implements ActionListener {

      . . . .
      @Override
      public void actionPerformed(ActionEvent e) {
            . . . .
      }
}
```

- Dans *initJFrame*, indiquer aux boutons que la fenêtre écoute leurs actions en utilisant la méthode *addActionListener* qui leur ajoute un écouteur comme suit :

```
button1.addActionListener(this) ;
button2.addActionListener(this) ;
```

- Maintenant à chaque fois qu'on va cliquer sur un bouton, la méthode *actionPerformed* va être appelée. Vérifier cela en affichant à la console `"Clic"` depuis *actionPerformed*.

```
@Override
public void actionPerformed(ActionEvent e) {
       System.out.println("Clic") ;
}
```

- Afficher votre nouvelle fenêtre. Cliquer sur l'un des deux boutons. Vous verrez donc apparaître "Clic" dans la console à chaque clic.

Différentes Sources

a. Pour différencier les deux boutons, on va devoir comparer la référence de la source de l'événement avec nos boutons pour savoir qui a lancé l'événement comme suit :

```
public void actionPerformed(ActionEvent e) {
            Object source = e.getSource();

            if(source == button1){
                  System.out.println("Vous avez cliqué sur OK");
            }
            else if(source == bouton2){
                  System.out.println("Vous avez cliqué sur Annuler");
            }
}
```

b. Une deuxième manière de le faire consiste à utiliser ActionCommand :

```
public void actionPerformed(ActionEvent e) {
            String action = e.getActionCommand() ;
            if(action.equals("OK")){
                  System.out.println("Vous avez cliqué sur OK");
            }
            else if(action.equals("Annuler")){
                  System.out.println("Vous avez cliqué sur Annuler");
            }
}
```

- Il est tout aussi possible d'utiliser une **classe interne anonyme** (anonymous inner class) afin de gérer un événement, comme suit - depuis la méthode *initJFrame* :

```
button1.addActionListener(new ActionListener() {
             @Override
             public void actionPerformed(ActionEvent event) {
                 System.out.println("Clic");
             }
});
```