Les bases de la programmation graphique

1) Généralités

Le langage Java propose deux bibliothèques dédiées à la conception d'interfaces graphiques. La bibliothèque AWT et la bibliothèque SWING. Les principes d'utilisation sont quasiment identiques pour ces deux bibliothèques.

Une interface utilisateur Swing est un assemblage de composants (components) affichés dans une fenêtre ou un autre type de conteneur (container).

La bibliothèque Swing dispose de trois classes permettant de jouer ce rôle :

JApplet : représente une fenêtre graphique embarquée à l'intérieur d'une page html pour être prise en charge par un navigateur. Cet élément est étudié en détail dans le chapitre qui lui est consacré.

JWindow : représente une fenêtre graphique la plus rudimentaire qui soit. Celle-ci ne dispose pas de barre de titre, de menu système, pas de bordure, c'est en fait un simple rectangle sur l'écran. Cette classe est rarement utilisée sauf pour l'affichage d'un écran d'accueil lors du démarrage d'une application (splash screen).

JFrame : représente une fenêtre graphique complète et pleinement fonctionnelle. Elle dispose d'une barre de titre, d'un menu système, d'une bordure, elle peut facilement accueillir un menu, c'est bien sûr cet élément que nous allons utiliser dans la très grande majorité des cas.

2) La classe JFrame

Pour créer une fenêtre graphique, il faut appeler une classe standard nommée JFrame.

```
JFrame fen = new JFrame() ;
```

Par défaut une telle fenêtre est créée avec une taille nulle, il est nécessaire d'en définir les dimensions auparavant :

```
fen.setSize(300,150) ;
```

On peut lui définir un titre :

```
fen.setTitle(« ma première fenêtre ») ;
```

Il ne faut pas se limiter à cela, car rien n'apparaîtra à l'écran.

```
fen.setVisible(true) :
```

Création d'une fenêtre personnalisée

```
public class MaFenetre extends JFrame{
          public MaFenetre(){// constructeur
              setSize(300,150) ;
              setTitle("ma première fenêtre") ;
}


}
// Classe de test
    public static void main(String[] args) {
        // TODO code application logic here
        MaFenetre fen = new MaFenetre();
        fen.setVisible(true) ;

    }
```

3) Gestion d'un clic dans la fenêtre

La programmation événementielle constitue la caractéristique essentielle d'une interface graphique. La plupart des événements sont créés par des composants qu'on aura introduits (menus, boutons, boîte de dialogue…)

Avant d'introduire de nouveaux composants, nous allons nous intéresser à un événement : le clic sur la fenêtre.

Les différentes démarches à suivre pour gérer cet événement sont généralisables aux autres événements.

L'implémentation de l'interface MouseListener

Pour traiter un événement, on associe à la source un objet de son choix dont la classe implémente une interface particulière correspondante à la catégorie d'événements.

On dit que cet objet est un écouteur de cette catégorie d'événements.

Chaque méthode proposée par l'interface correspond à un événement de la catégorie.

Par exemple, il existe une catégorie d'événements souris qu'on peut traiter avec un écouteur de souris, c'est-à-dire une classe implémentant une interface avec 5 méthodes.

```
public class EcouteurSouris implements MouseListener {
   public void mouseClicked(MouseEvent e) {    }
   public void mousePressed(MouseEvent e) {    }
   public void mouseReleased(MouseEvent e) {    }
   public void mouseEntered(MouseEvent e) {    }
   public void mouseExited(MouseEvent e) {    }
}
```

Ajout d'un écouteur au sein de la classe MaFenetre

```
public class MaFenetre extends JFrame implements MouseListener{
    public MaFenetre(){// constructeur
        setSize(300,150) ;
        setTitle("ma première fenêtre") ;

   // la fenêtre est son propre écouteur
        this.addMouseListener(this);
}

   public void mouseClicked(MouseEvent e) {
        System.out.println("clic");
            int x,y;
            x=e.getX();
            y=e.getY();
            System.out.println("x:"+x + "\ny:"+y);
        }
   public void mousePressed(MouseEvent e) {    }
   public void mouseReleased(MouseEvent e) {    }
   public void mouseEntered(MouseEvent e) {    }
   public void mouseExited(MouseEvent e) {    }


}
```

La notion d'adaptateur

Dans les exemples précédents, nous avons du fournir des définitions vides de méthodes car nous souhaitions pas toutes les implémenter.

Une solution consiste à passer par une classe intermédiaire : MouseAdapter
Cette classe particulière implémente MouseListener.

C'est comme si elle était définie ainsi :

```
public class MouseAdapter implements MouseListener {
    public void mouseClicked(MouseEvent e) {    }
    public void mousePressed(MouseEvent e) {    }
    public void mouseReleased(MouseEvent e) {    }
    public void mouseEntered(MouseEvent e) {    }
    public void mouseExited(MouseEvent e) {    }
}

public class EcouteurSouris extends MouseListener {
    public void mouseClicked(MouseEvent e) {
        System.out.println("clic");
        int x,y;
        x=e.getX();
        y=e.getY();
        System.out.println("x:"+x + "\ny:"+y);
    }
}
```

```
public class MaFenetre extends JFrame {
    public MaFenetre(){// constructeur
        setSize(300,150) ;
        setTitle("ma première fenêtre") ;

   // la fenêtre est son propre écouteur
         addMouseListener(new EcouteurSouris());
}
```

Intérêt :
De plus, les traitements sont externalisés dans une classe à part.
Le code apparaît plus clair.

4) La gestion des événements en général

Un événement, déclenché par un objet nommé source, peut être traité par un objet nommé écouteur et préalablement associé à la source.

A une catégorie donnée `Xxx`, on associe un objet écouteur des événements (de type `XxxxEven`t) par une méthode nommée `addXxxxlistener`.

Chaque fois qu'une catégorie donnée disposera de plusieurs méthodes, on pourra :

Soit redéfinir toutes les méthodes de l'interface correspondante `XxxListener` (avec la clause `implements XxxxListener` dans l'en-tête de l'écouteur). Les méthodes non utilisées auront un corps vide.

Soit faire appel à une classe dérivée d'une classe adaptateur `Xxxadaptateur` et ne fournir que les méthodes qui nous intéressent.

Premier composant : un bouton

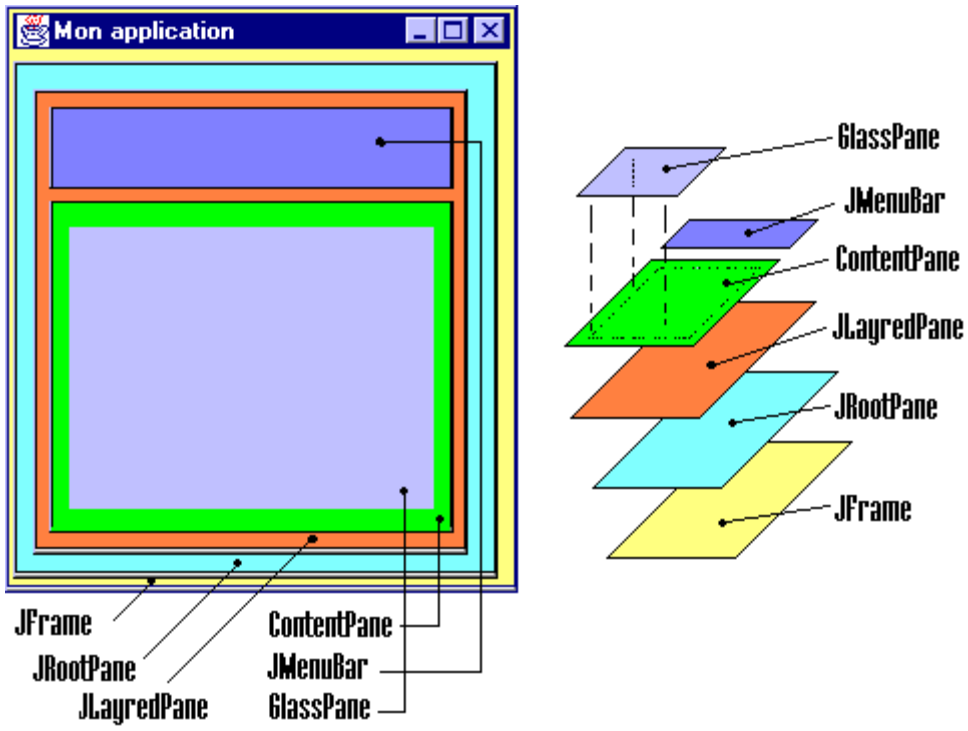
Création d'un bouton et ajout dans la fenêtre

Pour créer un bouton, on utilise le constructeur de la classe `JButton`.

```
JButton monBouton ;
MonBouton = new JButton("Etiquette") ;
```

La structure d'une Frame

Il faut ensuite introduire ce composant dans la fenêtre. Pour cela, il faut comprendre la structure d'une JFrame



Une `JFrame` est une superposition de plusieurs éléments, en particulier une racine, un contenu et une vitre

C'est la partie « contenu » qui nous intéresse puisque nous y incorporerons les différents composants.
La méthode `getContentPane()` de la classe `JFrame` fournit la référence à ce conteneur, de type Container.

Voilà comment obtenir la référence :

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

La méthode `add` de la classe `Container` permet d'ajouter un composant quelconque à un objet de ce type.

```
c.add(monBouton) ;
```

Affichage du bouton : la notion de gestionnaire de mise en forme

La disposition des composants dans une fenêtre est gérée par ce qu'on appelle un gestionnaire de mise en forme ou de disposition (`Layout Manager`).
Par défaut, Java utilise un gestionnaire de classe `BorderLayout` (en l'absence d'information un composant occupe toute la fenêtre)..

Un gestionnaire plus intéressant est celui de la classe `Flowlayout` qui dispose les composants en flot c'est-à-dire sur une même ligne puis ligne par ligne.

`setLayout` est la méthode permettant de choisir le gestionnaire.

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

Gestion d'un bouton avec un écouteur

Un bouton ne peut déclencher qu'un seul événement correspondant à l'action de l'utilisateur sur ce bouton. Généralement, cette action est déclenchée par un clic sur le bouton.

Il faut ici

- Créer un écouteur qui sera objet d'une classe qui implémente `ActionListener`. Cette dernière ne comporte qu'une méthode nommée `actionPerformed()` ;

- Associe cet écouteur au bouton par la méthode `addActionListener`

Avec deux boutons

Quand un même écouteur est associé à différents boutons, ont peut différencier les traitements grâce à :
getSource
Méthode getSource (utile quand le listener est défini dans la même classe que la fenêtre
e.getSource() == monBouton1

e est un ActionEvent

getActionCommand
e.getActionCommand() = "Bouton 1"