Les bases de la programmation graphique

Plan

- Introduction
- Création d'une fenêtre graphique
- Gestion d'un clic dans une fenêtre
- Création d'un bouton et ajout dans une fenêtre
- Gestion des boutons avec un écouteur

Introduction

- Les bases de la programmation graphique avec Swing, API graphique la plus utilisée en Java
- Création d'une fenêtre graphique (événement simple sur fenêtre).
- Notion fondamentale d'écouteurs d'événement.
- Introduction d'un composant dans une fenêtre (bouton ...)
- Gestion des évènements de Java (association entre événement et écouteur)
- Notion de dessin sur un composant ou dans une fenêtre (par l'intermédiaire d'un panneau)

Première fenêtre : JFrame

- JFrame est une classe du paquetage javax.swing
- Elle possède un constructeur JFrame fen=new JFrame();
- On crée un objet de type JFrame et on place sa référence dans fen (rien n'apparaitra sur l'écran).
- L'affichage : fen.setVisible(true); (possible show())
- Les dimensions : fen.setSize(300,150);(par défaut c nulle)
- Le titre : fen.setTitle("ma première fenêtre")

Premier fenêtre : programme simple

```
import javax.swing.*;


public class Premfen0{

public static void main (String args[]){

JFrame fen=new JFrame();

fen.setSize(300,150);

fen.setTitle("ma première fenêtre");

fen.setVisible(true);

}

}
```

Création d'une fenêtre graphique simple

Malgré que vous n'avez rien prévu en particulier, l'utilisateur peut manipuler cette fenêtre comme n'importe quelle fenêtre graphique d'un logiciel et en particulier :

- La retailler
- La déplacer (ici, elle s'est affichée dans le coin haut gauche de l'écran)
- La réduire à une icône

Ces fonctionnalités sont gérées par la classe JFrame.

Création d'une classe fenêtre personnalisée

- Nous avons créé un objet de type JFrame et nous avons utilisé les fonctionnalités présentes dans cette classe.
- Pour que notre programme présente un intérêt, il faut lui associer des fonctionnalités ou des champs supplémentaires.
- Cette nouvelle fenêtre devra pouvoir réagir à certains événements.

suite

- Il nous faudra généralement définir notre propre classe dérivée de JFrame et créer un objet de ce nouveau type.

```
import javax.swing.*;

class MaFenetre extends JFrame{

public MaFenetre() {//constructeur

setTitle("Ma premiere fenetre");

setSize(300,150); } }

public class Premfen1{

public static void main (String args[]){

JFrame fen=new MaFenetre();

fen.setVisible(true);}}
```

Actions sur les caractéristiques d'une fenêtre

- Nous pouvons fixer non seulement les dimensions mais aussi la position de la fenêtre à l'écran (setBounds)

```
fen.setBounds(10,40,300,200); /*le coin supérieur gauche de la fenêtre est placé au pixel de coordonnées 10,40  et ses dimensions seront de 300*200 pixel*/
```

- L'origine des coordonnées coïncides avec le coin supérieur gauche de l'écran. L'axe des abscisses est orienté vers la droite, celui des ordonnés vers le bas.
- setBackground (modéfication de la couleur)
- getSize (obtention de la taille courante)

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 dans la fenêtre(menus, boutons, boîtes de dialogue, ...), mais d'apprendre à créer ses composants, nous allons voir comment traiter des événements qu'ils génèrent.
- On va voir l'événement de clic sur la fenêtre principale (pas besoin de créer un nouvel objet)

Implémentation de l'interface MouseListner

- Par simplicité, nous signalons l'événement en affichant un message dans la fenêtre console.
- En java, tout événement possède ce que l'on nomme une **source** (il s'agit de l'objet lui ayant donnée naissance : bouton, article, de menu, fenêtre ...) dans notre exemple sera la fenêtre principale.
- Pour traiter un événement, on associe à la source un objet dont la classe implémente une interface particulière correspondant à une catégorie d'événement.
- On dit que cet objet est un **écouteur** de cette catégorie d'événement.

suite

- Il existe une catégorie d'événement **souris** qu'on peut traiter par un écouteur de souris c'est à dire un objet d'une classe implémentant l'interface MouseListener
- Elle comporte 5 méthodes correspondant chacune à un événement particulier : mousePressed, mouseReleased, mouseEntered, mouseExited, mouseClicked
- mouseClicked correspond à un clic usuel(appui suivi de relâchement sans déplacement)

Implémentation de l'interface

```
class EcouteurSouris implements MoussListener {
public void mouseClicked(MouseEvent ev){...}
public void mousePressed(MouseEvent ev){...}
public void mouseReleased(MouseEvent ev){...}
public void mouseEntered(MouseEvent ev){...}
ublic void mouseExited(MouseEvent ev){...}
//autres méthodes et champs de la classe
}
```

suite

- Événement de mouseClicked
- Redéfinition de la méthode mouseClicked

```
public void mouseClicked (MouseEvent ev){

system.out.println("clic dans fenetre");}
```

- La classe doit redéfinir toutes les méthodes meme s'elles ne font rien en particulier.
- Pour traiter un clic souris dans notre fenêtre, il suffit d'associer à notre fenêtre un objet d'un type tel que EcouteurSouris.
- Utilisation de la méthode addMouseListener (méthodes de la classe JFrame)

suite

- Introduction dans le constructeur de notre fenêtre une instruction de la forme : addMouseListener(objetEcouteur)
- En Java : l'objet écouteur peut être n'importe quel objet dont la classe implémente l'interface voulue.
- Pour la simplicité, prenant la fenêtre elle même son propre écouteur d'événement souris (si possible car la seule chose qu'on demande à un écouteur est que sa classe implémente l'interface voulue (ici MouseListener).

Utilisation de l'information associée à un événement

- Etudier l'argument de la méthode mouseClicked
- L'objet de type MouseEvent(correspond à la catégorie d'événements gérés par l'interface **MouseListener**)
- Un objet de cette classe est automatiquement créé par Java lors du clic et transmis à l'écouteur voulu
- Il contient des informations qui concerne les cordonnées du curseur de souris au moment du clic.
- Accessibles par des méthodes getX et getY

Programme

```
    import javax.swing.* ; // pour Jframe
    import java.awt.event.* ; // pour MouseEvent et MouseListener

class MaFenetre extends JFrame implements MouseListener{

public MaFenetre (){

setTitle ("Gestion de clics") ; setBounds (10, 20, 300, 200) ;

this.addMouseListener (this) ; // la fenetre sera son propre écouteur d'événements souris
      }

public void  mouseClicked(MouseEvent ev){

int x = ev.getX() ; int y = ev.getY() ;

System.out.println ("clic au point de coordonnees " + x + ", " + y );}

       public void mousePressed (MouseEvent ev) {}
      public void  mouseReleased(MouseEvent ev) {}
      public void  mouseEntered (MouseEvent ev) {}
      public void  mouseExited (MouseEvent ev) {}

}
```

Classe d'utilisation

```
public class Clic1 {
 public static void main (String args[]) {
MaFenetre fen = new MaFenetre() ;
fen.setVisible(true) ;
}
}
```

Deuxième technique

```
import javax.swing.* ; // pour JFrame
import java.awt.event.* ; // pour MouseEvent et MouseListener

 class MaFenetre extends JFrame {

 public MaFenetre () { // constructeur

 setTitle ("Gestion de clics") ; setBounds (10, 20, 300, 200) ;

 this.addMouseListener (new  EcouteurSouris() ) ; // la fenetre sera son propre écouteur d'événements souris
}}

                    //  LA CLASSE ECOUTEUR

class EcouteurSouris implements MouseListener {

public void  mouseClicked(MouseEvent ev) { // méthode gerant un clic souris

int x = ev.getX() ; int y = ev.getY() ;

System.out.println ("clic au point de coordonnees " + x + "," + y );
    }

 public void  mousePressed (MouseEvent ev) {}
 public void  mouseReleased(MouseEvent ev) {}
 public void  mouseEntered (MouseEvent ev) {}
 public void  mouseExited (MouseEvent ev) {}
 }
```

La notion d'adaptateur

- Nous n'avions besoin que de la méthode mouseClicked mais nous avons du fournir des définitions vides pour les autres méthodes (pour l'implémentation de l'interface MouseListener)
- Java dispose d'une classe d'adaptation MouseAdapter qui implémente toutes les méthodes de l'interface MouseLinstener avec un corps vide .

Deuxième technique

```
import javax.swing.* ; // pour JFrame
import java.awt.event.* ; // pour MouseEvent et MouseListener

class MaFenetre extends JFrame {

public MaFenetre () { // constructeur

setTitle ("Gestion de clics") ; setBounds (10, 20, 300, 200) ;

this.addMouseListener (new  EcouteurSouris()) ; // la fenetre sera son propre écouteur d'événements souris
}}
```

```
class EcouteurSouris extends MouseAdapter {

public void  mouseClicked(MouseEvent ev) {  // méthode gerant un clic souris

int x = ev.getX() ; int y = ev.getY() ;

System.out.println ("clic au point de coordonnees " + x + "," + y );

}
    }
```

La gestion des événements en général

- Un événement déclenché par un objet nommé source pouvait être traité par un autre objet nommé écouteur préalablement associé à la source.
- Cette technique se généralisera aux autres événements, quels qu'ils soient et quelle que soit leur source.
- À une catégorie donnée Xxx, on associera toujours un objet écouteur des événements d'une classe implémente l'interface XxxListener par l'intermédière d'une méthode nommée addXxxListener.
- Après, soit, on redéfinit ttes les méthodes de l'interface correspondant XxxListener soit, on fait appel à une classe dérivée d'une classe adaptateur XxxAdapter.

Création et ajout d'un bouton dans une Fenêtre

Premier composant: un bouton

- Nous avons pas introduit de composant particulier dans la fenêtre graphique.

- Comment placer un bouton et intercepter les actions correspondantes

Création d'un bouton et ajout dans la fenêtre

- JButton monBouton=new JButton("ESSAI");
- Création d'un bouton pourtant l'étiquette "ESSAI"
- On va mettre les composants dans la partie contenu de JFrame.
- getContentPane() de la classe JFrame fournie la référence à ce contenu de type Container.
- Container c=getContentPane();
- c.add(monBouton);
  (getContentPane().add(monBouton))

Création d'un bouton

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

class Fen1Bouton extends JFrame{

  Private JButton monBouton;

  public Fen1Bouton (){
  setTitle ("Premier bouton") ;
  setSize (300, 200) ;

monBouton = new JButton ("ESSAI") ;

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

getContentPane().add(monBouton) ;

}

}
```

Classe d'utilisation

```
public class Bouton1 {
 public static void main (String args[]) {
 Fen1Bouton fen = new Fen1Bouton() ;
fen.setVisible(true) ;
}
}
```

Gestion du bouton avec un écouteur

- Un bouton ne peut déclencher qu'un seul événement correspondant à l'action de l'utilisateur sur ce bouton.
- La démarche présentée pour gérer clic sur une fenêtre s'adapte pour gérer une action sur un bouton.
- L'événement que nous intéresse est une événement nommée Action. Il faudra donc :

➢ *Créer un écouteur qui sera l'objet d'une classe qui implémente* ActionListener *(cette interface ne comporte qu'une méthode actionPerformed)*

➢ *Associer cet écouteur au bouton par addActionLinstener().*

Programme

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

class Fen1Bouton extends JFrame implements ActionListener{

private JButton monBouton;

// Attention : ne pas oublier implements

public Fen1Bouton (){

 setTitle ("Premier bouton"); setSize (300, 200);

monBouton=new JButton ("ESSAI");

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

getContentPane().add(monBouton);

monBouton.addActionListener(this);}

public void actionPerformed (ActionEvent ev) {
        System.out.println ("action sur bouton ESSAI");
        }
        }
```

Gestion de plusieurs composants

- En utilisant le gestionnaire de type FlowLayout, les différents boutons seront affichés séquentiellement dans l'ordre de leur ajout.
- Pour la gestion des actions : chaque événement de chaque composant peut disposer de son propre objet écouteur(bien même écouteur pour tous les boutons)
- Utilisation des méthodes : getSource() et getActionCommand()
- Différentes techniques se présentent

La fenêtre écoute les boutons

- La fenêtre l'objet écouteur de tous les boutons
- Prévoir exactement la même réponse quel que soit le bouton
- Prévoir une réponse dépendant du bouton concerné, ce qui nécessite de l'identifier, nous verrons qu'ont peut le faire en utilisant l'une des méthodes getSource ou getActionCommand

Tous les boutons déclenches la même réponses

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

class Fen2Boutons extends JFrame implements ActionListener {

 private JButton monBouton1, monBouton2 ;

public Fen2Boutons (){
    setTitle ("Avec deux boutons") ;setSize (300, 200) ;
    monBouton1 = new JButton ("Bouton A") ;
    monBouton2 = new JButton ("Bouton B") ;

    Container contenu = getContentPane() ;
    contenu.setLayout(new FlowLayout()) ;

    contenu.add(monBouton1) ;
    contenu.add(monBouton2) ;

    monBouton1.addActionListener(this); // la fenetre ecoute monBouton1
    monBouton2.addActionListener(this); // la fenetre ecoute monBouton2
    }

public void actionPerformed (ActionEvent ev) { // gestion commune a
    System.out.println ("action sur un bouton") ; // tous les boutons
    }
    }
```

Chaque bouton déclenche sa propre réponse

- La méthode getSource()

- La méthode getActionCommand()

La méthode getSource

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

class Fen2Boutons extends JFrame implements ActionListener{

        private JButton monBouton1, monBouton2 ;

        public Fen2Boutons (){
        setTitle ("Avec deux boutons") ;setSize (300, 200) ;

        monBouton1 = new JButton ("Bouton A") ;
        monBouton2 = new JButton ("Bouton B") ;

        Container contenu = getContentPane() ;
        contenu.setLayout(new FlowLayout()) ;

        contenu.add(monBouton1 ) ;
        contenu.add(monBouton2) ;

        monBouton1 .addActionListener(this);
        monBouton2.addActionListener(this);}

public void actionPerformed (ActionEvent ev) {

if (ev.getSource() == monBouton1 ) System.out.println ("action sur bouton numero 1") ;

if (ev.getSource() == monBouton2 ) System.out.println ("action sur bouton numero 2") ;}
```

La méthode getActionCommand()

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

class Fen2Boutons extends JFrame implements ActionListener{

        private JButton monBouton1, monBouton2 ;

        public Fen2Boutons (){
        setTitle ("Avec deux boutons") ;setSize (300, 200) ;

        monBouton1 = new JButton ("Bouton A") ;
        monBouton2 = new JButton ("Bouton B") ;

        Container contenu = getContentPane() ;
        contenu.setLayout(new FlowLayout()) ;

        contenu.add(monBouton1 ) ;
        contenu.add(monBouton2) ;

        monBouton1 .addActionListener(this);
        monBouton2.addActionListener(this);}

      public void actionPerformed (ActionEvent ev) {

       String nom = ev.getActionCommand() ;
        System.out.println ("Action sur bouton " + nom) ;}

}
```

Classe écouteur différente de la classe fenêtre

- Une classe écouteur par bouton

- Une seule classe écouteur pour tous les boutons

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

class Fen2Boutons extends JFrame {

        private JButton monBouton1, monBouton2
        public Fen2Boutons (){
        setTitle ("Avec deux boutons") ;setSize (300, 200) ;

        monBouton1 =new JButton ("Bouton A") ;
        monBouton2=new JButton ("Bouton B") ;

       Container contenu=getContentPane() ;
        contenu.setLayout(new FlowLayout()) ;

       c ontenu.add(monBouton1) ;
        contenu.add(monBouton2) ;

        EcouteBouton1 ecout1 =new EcouteBouton1() ;
        EcouteBouton2 ecout2=new EcouteBouton() ;

        monBouton1.addActionListener(ecout1) ;
        monBouton2.addActionListener(ecout2);}}

class EcouteBouton1 implements ActionListener {
         public void actionPerformed (ActionEvent ev) { System.out.println ("action sur bouton 1") ;}

}
```

suite

```
class EcouteBouton2 implements ActionListener{
 public void actionPerformed (ActionEvent ev){ System.out.println ("action sur bouton 2") ;}
  }


public class Boutons4 {

public static void main (String args[])

{ Fen2Boutons fen = new Fen2Boutons() ;

fen.setVisible(true) ;

}

}
```

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

class Fen2Boutons extends JFrame {

private JButton monBouton1, monBouton2 ;

        public Fen2Boutons (){ setTitle ("Avec deux boutons") ; setSize (300, 200) ;}

        monBouton1 = new JButton ("Bouton A") ;
        monBouton2 = new JButton ("Bouton B") ;

        Container contenu = getContentPane() ;
        contenu.setLayout(new FlowLayout()) ;


        contenu.add(monBouton1) ;
        contenu.add(monBouton2) ;

         EcouteBouton ecout1 = new EcouteBouton(10) ;
        EcouteBouton ecout2 = new EcouteBouton(20) ;


        monBouton1.addActionListener(ecout1);
        monBouton2.addActionListener(ecout2);}}

class EcouteBouton implements ActionListener{
        private int n ;

         public EcouteBouton (int n){ this.n = n ;}

        public void actionPerformed (ActionEvent ev){ System.out.println ("action sur bouton " + n) ;}
        }
```