IHM avec JAVA

Isabelle Debled-Rennesson
debled@loria.fr

Semaine 4

Programme de la séance

- Principe de la gestion des événements
- Exemple d'utilisation de Listeners
- Exercices

Gestion des événements
Le principe – P25-26

- Chaque fois que l'utilisateur agit, un événement est déclenché par le composant situé par exemple sous la souris :
  - clic de souris, gauche ou droit
  - déplacement de la souris (mouse moved)
  - glissement de la souris avec bouton appuyé (mouse dragged)
  - utilisation du clavier
- Le composant est appelé source de l'événement.
- A chaque déclenchement, un objet événement est créé ; il contient entre autres une référence au composant source (classes **XXXEvent** de **java.awt.event**).
  - *ActionEvent*
  - *ChangeEvent*
  - *FocusEvent*
  - *MouseEvent, KeyEvent, etc.*

Gestion des événements
Le principe de la délégation

Le composant graphique concerné par l'événement ne gère pas l'événement lui-même, il délègue à des objets dits auditeurs ou écouteurs (listeners) ou récepteurs :

- Le composant est associé à des objets écouteurs (listeners), enregistrés au préalable avec l'instruction suivante :
  - **objetSource.addXXXListener(objetEcouteur)**
- Le composant prévient les écouteurs de l'arrivée d'un événement qui les concerne (notified).
- Lorsqu'il est prévenu, l'écouteur exécute une méthode de traitement spécifique à l'événement.

Quelques listeners

- Listeners gérés par tous les composants Swing
  - **ComponentListener : changement de taille, de position, de visibilité**
  - **FocusListener : réception/absence de focus**
  - **KeyListener : frappe d'une touche (pour un composant qui a le focus)**
  - **MouseListener : click et mouvement de souris**
  - **MouseMotionListener : changement de la position du curseur dans un composant**
  - **MouseWheelListener : roulette de souris**
- Listeners gérés par certains composants Swing
  - **ActionListener : sélection/désélection d'un bouton, etc.**
  - **ItemListener : sélection/désélection des items de menu, de listes déroulantes**
  - **WindowListener : ouverture, fermeture, iconification, ... d'une fenêtre**

Les Listeners sont des interfaces

Les Listeners sont des interfaces

- Une interface est une classe « vide », elle ne possède pas de champs. Seuls les en-têtes de méthodes sont données.
- Une classe peut implanter une ou plusieurs interface et en même temps elle peut étendre une classe
- Quand une classe implante (implémente) une interface, elle s'engage à définir toutes les méthodes de l'interface qu'elle implante.
- Pour signifier qu'une classe se conforme à une interface, le nom de la classe est suivi du mot-clé ***implements***, puis du nom de l'interface (ou de la liste des interfaces séparées par des virgules)

ActionListener

- L'écouteur ActionListener est une interface qui décrit le comportement d'un ActionListener sans en détailler la réalisation.

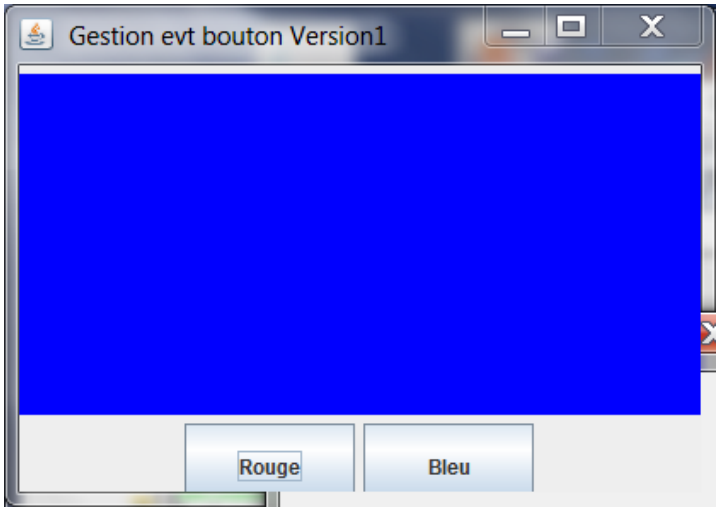
```
Interface ActionListener {
public void actionPerformed(ActionEvent e) ;
}
```

http://java.sun.com/docs/books/tutorial/uiswing/events/index.html

- Des informations peuvent être obtenues à partir de l'événement e de la classe ActionEvent :
  - Object source = e.getSource(); //permet d'obtenir le composant source de l'événement
  - String input=e.getActionCommand(); // permet d'obtenir dans input la chaîne de caractère située sur le bouton source de l'événement

Un exemple

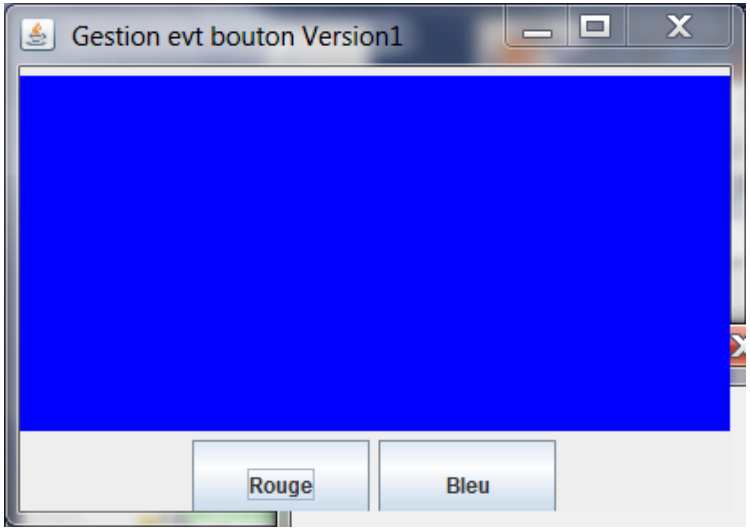
- Gestion des événements sur des boutons



-Trois versions-
P 26 à 29

Un exemple

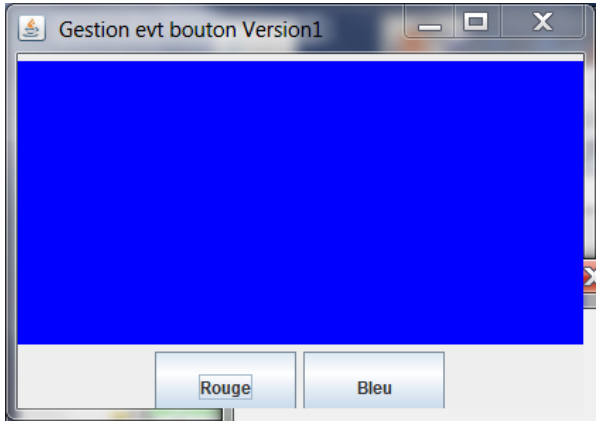
- Gestion des événements sur des boutons



Troisième version **Evt_JButton_V3**

-la classe héritant du JPanel associé à la JFrame implémente ActionListener-

```
public class Evt_JButton_V3 extends JPanel   implements ActionListener{

    private JButton br,bb; //declarations
    private JPanel dess ;

    public Evt_JButton_V3 ( ){

        dess= new JPanel();
        dess.setPreferredSize(new Dimension(400,200));
        dess.setBackground(Color.blue);

        br=new JButton("Rouge"); //créations des composants
        br.addActionListener(this);   //Le JPanel est écouteur des evts sur le bouton br
        br.setPreferredSize(new Dimension(100,50));

        bb= new JButton("Bleu");
        bb.addActionListener(this);
        bb.setPreferredSize(new Dimension(100,50));

        //Ajout des composants dans le JPanel
        this.add(dess);
        this.add(br);
        this.add(bb);
    }

    public void actionPerformed(ActionEvent e){
        Object source = e.getSource(); // Récupération de la source de l'evt
        if(source.equals(br))  dess.setBackground(Color.red);
        if(source.equals(bb)) dess.setBackground(Color.blue);
    }
}
```



Les classes internes (inner classes)-P21

- Une classe est dite interne lorsque sa définition est située à l'intérieur d'une autre classe

- Un objet d'une classe interne a toujours accès aux champs et méthodes de l'objet externe lui ayant donné naissance

→ Version 1 **Evt_JButton_V1**

```
public class Evt_JButton_V1 extends JPanel  {
    private JButton br,bb; //declarations
    private JPanel dess ;

    public Evt_JButton_V1 ( ){
        dess= new JPanel();
        dess.setPreferredSize(new Dimension(400,200));
        dess.setBackground(Color.blue);
        EcouteurBouton ecouteur = new EcouteurBouton();
        br=new JButton("Rouge"); //créations des composants
        br.addActionListener(ecouteur);  //Une instance de la classe EcouteurBouton est écouteur
    des evts sur le bouton br
        br.setPreferredSize(new Dimension(100,50));

        bb= new JButton("Bleu");
        bb.addActionListener(ecouteur);
        bb.setPreferredSize(new Dimension(100,50));

        //Ajout des composants dans le JPanel
        this.add(dess);
        this.add(br);
        this.add(bb);
    }

//Classe interne
class EcouteurBouton implements ActionListener{
    public void actionPerformed(ActionEvent e){
        Object source = e.getSource(); // Récupération de la source de l'evt
        if(source.equals(br))  dess.setBackground(Color.red);
        if(source.equals(bb)) dess.setBackground(Color.blue);
```

Les classes internes et anonymes -P23

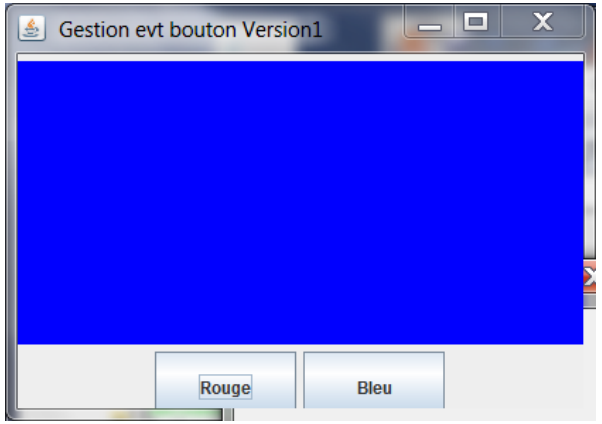
- Une classe anonyme est une classe sans nom.
  - Elle peut hériter d'une autre classe
  - Elle peut implémenter une interface

```
br.addActionListener( new ActionListener(){
      public void actionPerformed(ActionEvent e){
          dess.setBackground(Color.red);
      }
   });
```

- Intérêt : allège le code
  - Pas de nom de classe superflu, pas de constructeur

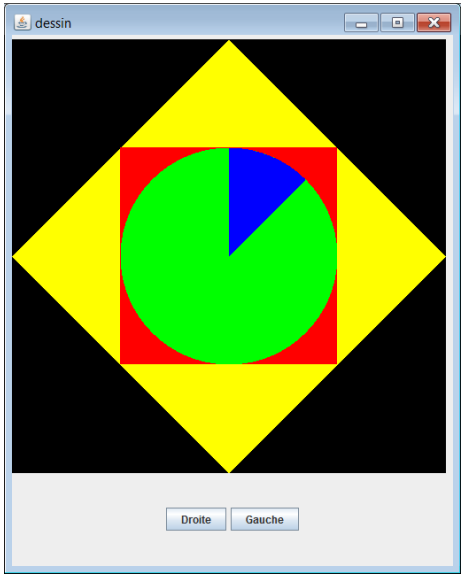
➔ Seconde version **Evt_JButton_V2**

```
public class Evt_JButton_V2 extends JPanel{
    private JButton br,bb; //declarations
    private JPanel dess ;

    public Evt_JButton_V2 ( ){
        dess= new JPanel();
        dess.setPreferredSize(new Dimension(400,200));
        dess.setBackground(Color.blue);

        br=new JButton("Rouge"); //créations des composants
        br.addActionListener( new ActionListener(){
          public void actionPerformed(ActionEvent e){
                dess.setBackground(Color.red);
           }
        } );
        br.setPreferredSize(new Dimension(100,50));

        bb= new JButton("Bleu");
        bb.addActionListener( new ActionListener(){
          public void actionPerformed(ActionEvent e){
                dess.setBackground(Color.blue);
           }
        } );
        bb.setPreferredSize(new Dimension(100,50));

        //Ajout des composants dans le JPanel
        this.add(dess);
        this.add(br);
        this.add(bb);
    }
```



Exercices

1. Faire l'exercice de la page P29 en reprenant la classe de dessin créée au second cours.



2. Finir l'exercice sur l'histogramme.

3. Reprendre l'exemple de la P27 et changer les actions liées aux boutons : le premier permet le dessin d'un cercle vert et le second d'un carré jaune plein.