Les points acquis

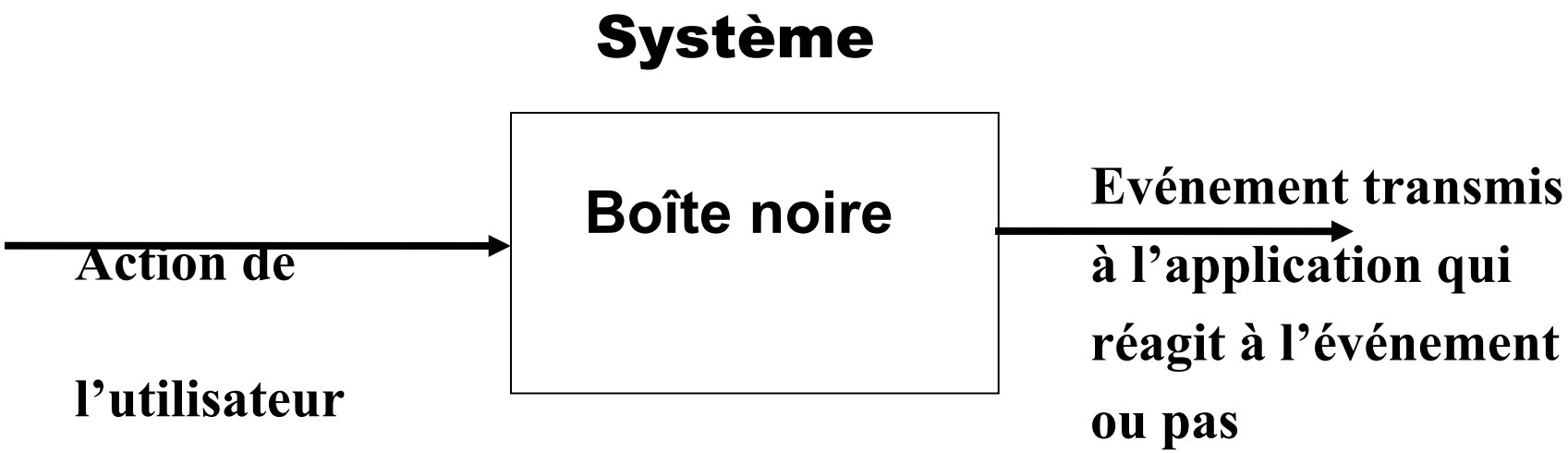
- Faire une fenêtre
- Découper une fenêtre en panneaux.
- Définir une mise en page pour la fenêtre et pour chacun des panneaux

Cours 3 : Programmation événementielle

Plan du cours

- Les événements
- Les écouteurs d'événements
- Programmer l'événementiel
- Un exemple
- Deux méthodes pour distinguer de quelle source vient un événement
- Les adapteurs

Rappels : action/événement



Evénement = Objet construit par le système en réponse à une action de l'utilisateur et qui contient toutes les informations concernant cette action

Gestion événementielle en java

Action de l'utilisateur dans un composant graphique → Création d'un événement

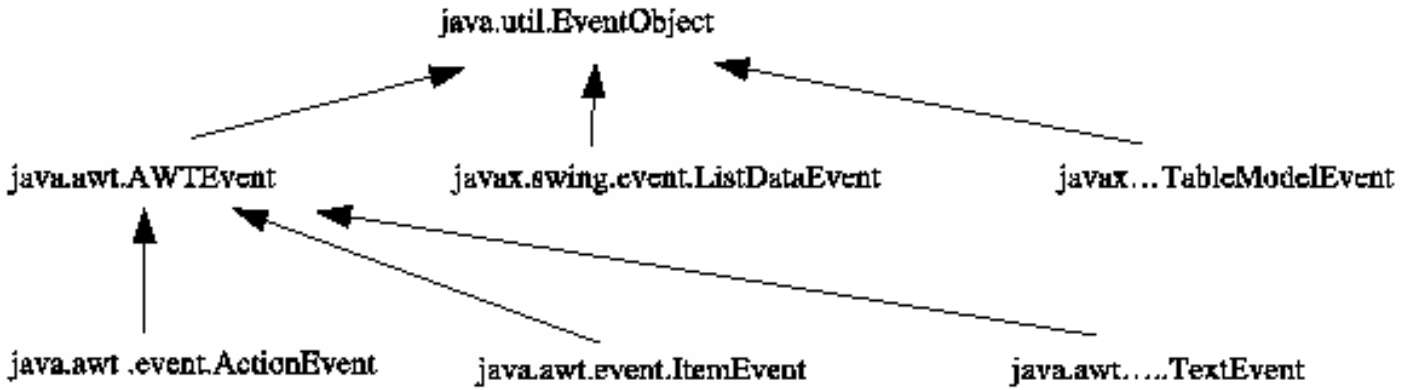
Fermer une fenêtre → Événement « fenêtre »

Cliquer sur un bouton,
Faire un choix dans un menu → Événement « action »

Bouger la souris → Événement « souris »

Les événements

Un événement est une instance d'une classe donnée qui hérite de la classe EventObject



Exemple : la classe ActionEvent

Quelques méthodes utiles

String getActionCommand()

Returns the command string associated with this action.

Object getSource()

Retourne l'objet source de l'événement

Rappel: Définition d'une interface

- C'est une notion propre au langage Java.

Définition

Une interface est une classe abstraite contenant uniquement des constantes (final) et/ou des signatures de méthodes

Une interface définit un ensemble d'opérations et/ou de constantes.
Les opérations sont uniquement décrites par leur signature.

Notion d'écouteur d'événement

- Un écouteur d'évènement est une instance d'une classe qui implémente une interface java dépendante de l'événement
- Exemple écouteur de l'événement action sur un bouton

==> interface ActionListener

```
public interface ActionListener extends EventListener{

 public void actionPerformed(ActionEvent e);

}
```

Lien composant, évènement, écouteur

Exemple : un clic sur un bouton

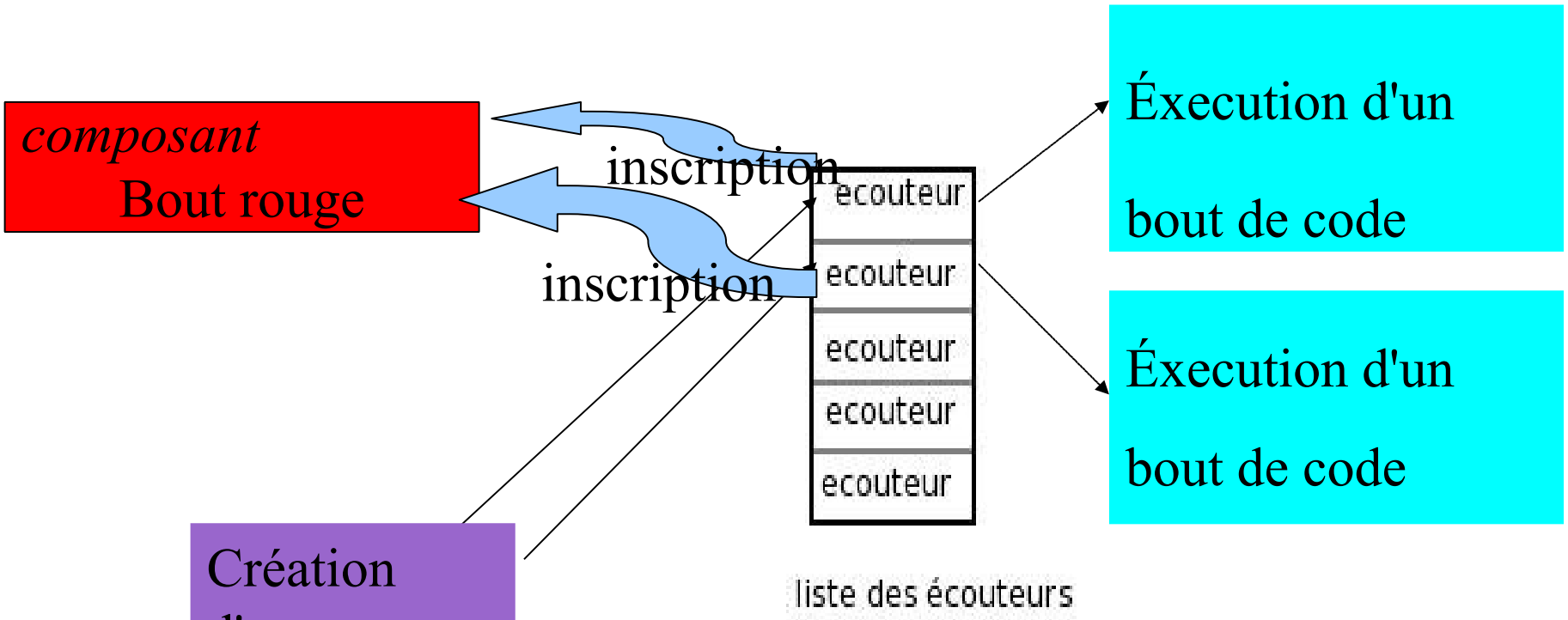
- Composant : un bouton
- Evénement : une instance de la classe ActionEvent

Ecouteur : une instance d'une classe qui implémente ActionListener qui **est inscrit aupres du composant comme écouteur**

Programmation événementielle

- Associer un morceau de code à un événement
- L'événement se produit, les écouteurs inscrits exécutent le code correspondant

Exemple: clic sur un bouton



Comment programmer ?

1. Première partie : créer une classe Ecouteur (*listener*) adaptée

Tout objet, toute classe peut devenir un Ecouteur (*listener*)

Il suffit que la classe implémente l'interface **Listener** choisie

2. Deuxième partie : relier la source à la cible

Inscrire le composant comme écouteur de l'événement

Première partie code pour qu'un composant réagisse à un événement donné

A) Définir un écouteur spécifique

==> en créant une classe qui implémente le listener adéquat,

-- choisir l'interface XXListener qui convient

-- donner un corps à l'ensemble des méthodes prévues dans cette interface

```
class BoutonOkListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
    ….}
}
```

Mettre ici le code qui s'exécutera quand on cliquera dans la source associée

Deuxième partie de code pour qu'un composant réagisse à un événement donné

B) Inscription = Lier la source (composant) à la cible (écouteur spécifique, instance de la classe définie en A.) en appelant la méthode :

void add*XX*Listener(*XX*Listener lis)

Exemple :

```
JButton bok = new JButton ("OK") ;            // la source
BoutonOkListener blis=  new BoutonOkListener();  // la cible
bok.addActionListener(blis);
```

…lien source-cible

Un exemple : Affichage dans une zone de texte du nombre de clics dans un bouton

```
public class TestBouton extends JFrame {
private JLabel labNbFois; //étiquette pour l'affichage
private int nb; //compteur

   public  TestBouton(String titre,int w, int h)  {
    super(titre);
    nb =0;

    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setBounds(400, 400, w,h);
    this.initialise();
    this.setVisible(true);
}
```

démo

```
private void initialise()  {
            JButton boutRouge=new JButton("Clique et tu verras... ");
            boutRouge.setBackground(Color.red);

            labNbFois= new JLabel();
        this.setLayout(new FlowLayout());
        this.add(boutRouge);
        this.add(labNbFois);
        BoutonListener blis=new BoutonListener();
        boutRouge.addActionListener(blis);
    }
```

Traitements des événements par une inner-classe

- Une inner-classe est une classe à l'intérieur d'une autre classe
- Privilèges particuliers :
  - Une inner-classe peut accéder directement aux variables d'instance (même privées) de la classe qui l'englobe sans passer par les méthodes d'accès
  - Une inner-classe peut appeler directement les méthodes d'instance de la classe qui l'englobe

```
class BoutonListener implements ActionListener  {
    public void actionPerformed(ActionEvent e) {
            nb = nb+1;
            labNbFois.setText( nb + " clics");
}// fin de l'inner-classe
}
   public static void main(String args[]){
   new TestBouton("Fenêtre avec un bouton",300,200);
   }
}//fin de la classe
```

Classe à l'intérieur d'une autre classe

Caractéristiques d'une inner-classe

- L'inner-classe n'existe qu'en relation avec sa classe englobante. Elle ne peut pas être utilisée en dehors de la classe qui l'englobe

- La compilation du fichier TestBouton.java génère deux fichiers TestBouton.class, TestBoutonEvt$BoutonListener.class

Caractéristiques d'une inner-classe

- Une inner-classe a des privilèges particuliers :
  - Elle peut accéder directement aux variables d'instance (même privées) de la classe qui l'englobe sans passer par les méthodes d'accès

    Exemple : nb, labNbFois
  - Elle peut appeler directement les méthodes de la classe qui l'englobe

```
class BoutonListener implements ActionListener  {
   public void actionPerformed(ActionEvent e) {
TestBoutonEvt.this…..
```

Référence de l'instance de la classe englobant l'inner-classe

Remarques sur les écouteurs

- On peut "enregistrer" plusieurs *écouteurs* auprès d'un même composant si l'on souhaite que le composant réagisse à différents types d'événements
- Plusieurs composants de même type, on peut définir une seule classe mais une ou plusieurs instances:
- Exemple:
  - plusieurs boutons --> une seule classe écouteur implémentant ActionListener
  - Il faut pouvoir différencier de quelle source vient l'événement

Cas où un même écouteur est enregistré auprès de plusieurs composants

- Comment un même écouteur lié à plusieurs sources reconnaît le composant source de l'événement ?
- 2 méthodes :
  - Méthode 1 : une seule instance : se servir de l 'instance de l'événement qui s'est produit et qui est accessible dans l'écouteur car présent en paramètre de la méthode pour distinguer la source.
  - Méthode 2 : plusieurs instances: identifier chaque instance par une valeur distincte passer au constructeur de l'écouteur.

Méthode 1 : une seule instance- utiliser l'événement produit, accessible dans la méthode de l'écouteur

- Dans la méthode appropriée de l'écouteur, adresser à

l'événement passé en paramètre une méthode particulière :

  - Dans le cas général : la méthode Object getSource() existe pour tout événement ; elle retourne la référence à la source de l'événement ; il reste à savoir quelle est cette source :

```
if (e.getSource()  ==  unComposant)
// l'événement provient de la source unComposant
```

  - ou une méthode plus spécifique présente dans la classe de l'événement qui s'est produit

Exemple de la méthode 1 pour des boutons

- Si plusieurs boutons sont associés à un même écouteur de type ActionEvent, On peut utiliser la la méthode

  String getActionCommand() de la classe ActionEvent

- Cette méthode retourne l'étiquette du composant qui a déclenché l'événement

Exemple détaillé : Changement de couleur de fond de la fenêtre en cliquant sur 2 boutons (1 par couleur)

Démo

```
public class FenDeuxBoutons extends JFrame   {
    private JButton bRouge = null;
    private JButton bBleu = null;
    public FenDeuxBoutons(String titre,int w, int h)  {
        super(titre);
        this.setBounds(400, 400, w,h);
        this.initialise();
        this.initconnections();
        this.setVisible(true);
    }
```

```
private void initialise()  {
            this.setLayout(new FlowLayout());
            bRouge=new JButton("Fond rouge");
            bRouge.setForeground(Color.red);
             this.add(bRouge);
            bBleu=new JButton("Fond bleu");
            bBleu.setForeground(Color.blue);
            this.add(bBleu);

        }


private void initconnections(){
      this.setDefaultCloseOperation(EXIT_ON_CLOSE);
         BoutonListener list=new BoutonListener(); //une instance
             bRouge.addActionListener(list);
             bBleu.addActionListener(list);
        }
```

La classe Ecouteur

```
class BoutonListener implements ActionListener  {
  public void actionPerformed(ActionEvent e) {
       String s=e.getActionCommand();
        if (s.equals("Fond bleu"))
FenDeuxBoutons.this.getContentPane().setBackground(Color.blue);
                        else
FenDeuxBoutons.this.getContentPane().setBackground(Color.red);
                         }
}// fin de l'inner-classe

public static void main(String[] args) {
          new FenDeuxBoutons("Fenêtre avec deux boutons",800,600);

     }
```

Méthode 2 : plusieurs instances

Chaque instance est identifiée par un mnémonique

```
public class FenDeuxBoutonsMnemo extends JFrame   {
private static final int BLEU=0, ROUGE=1;
```

Mnémoniques, constantes de classe

Écouteurs construits avec valeurs particulières

```
public void initconnections()  {
bRouge.addActionListener(new BoutonListenerMnemo(ROUGE));
bBleu.addActionListener(new BoutonListenerMnemo(BLEU));
    }
```

```
class BoutonListenerMnemo implements ActionListener  {
 private int val;//variable mémorise la valeur

public  BoutonListenerMnemo(int i)  {
  val=i;
  }
```

Le constructeur

Pour savoir quelle est la source : tester la valeur de val

val est de type int :

[switch est possible

```
public void actionPerformed(ActionEvent e) {
switch (val)  {
   case BLEU :  FenDeuxBoutons.this.getContentPane().setBackground(Color.blue);
                break;
   case ROUGE : FenDeuxBoutons.this.getContentPane().setBackground(Color.red);
                break;
            }
 }
} // fin de BoutonListenerMnemo
}// fin de la classe FenDeuxBoutonsMnemo
```

Intérêts :

* on gagne en lisibilité
* switch est possible

Création d'une classe Gestionnaire des événements

Gestionnaire d'événements= Contrôleur

```
public class ControleurBoutons implements ActionListener  {
    private FenDeuxBoutonsAvecControleur fen;

    public ControleurBoutons(FenDeuxBoutonsAvecControleur fen) {
        super();
        this.fen = fen;
    }
public void actionPerformed(ActionEvent e) {
         String s=e.getActionCommand();
          if (s.equals("Fond bleu"))
fen.getContentPane().setBackground(Color.blue);
           else  fen.getContentPane().setBackground(Color.red);
            }
}
```

```
public class FenDeuxBoutonsAvecControleur extends JFrame   {
private JButton bRouge = null;
private JButton bBleu = null;

public FenDeuxBoutonsAvecControleur(String titre,int w, int h)  {
    super(titre);
this.setBounds(400, 400, w,h);
this.initialise();
this.initconnections();
this.setVisible(true);
}
...
private void initconnections(){
this.setDefaultCloseOperation(EXIT_ON_CLOSE);
ControleurBoutons cont = new ControleurBoutons(this);
bRouge.addActionListener(cont);
bBleu.addActionListener(cont);
}
....}
```

Ecouteur construit à partir d'une Interface

- Les listeners sont des interfaces ….

- Inconvénient : il faut implémenter toutes les méthodes de l'interface, même celles dont on ne se sert pas

MouseListener : 5 méthodes

WindowListener : 7 méthodes

Interception des événements souris

```
public class FenSourisListener extends JFrame {
JTextArea jta = null;
public FenSourisListener(String titre,int w, int h)  {
 .............
}
public  void initialise(){
   jta= new JTextArea(12,60 );
   this.setLayout(new FlowLayout());
   this.add(jta);
}
```

```
public void initconnections(){
this.setDefaultCloseOperation(EXIT_ON_CLOSE);
this.getContentPane().addMouseListener(new SourisAction())  ;
}
class SourisAction implements MouseListener  {
        public void mousePressed(MouseEvent e) {
            jta.append("  Bouton de la souris appuyé\n");
        }
        public void mouseReleased(MouseEvent e) {
            jta.append("  Bouton de la souris relâché\n");
        }
        public void mouseEntered(MouseEvent e) {
            jta.append("         entrée du curseur\n");
        }
        public void mouseExited(MouseEvent e) {
            jta.append("          sortie du curseur\n");
        }
        public void mouseClicked(MouseEvent e) {
            jta.append(" clic sur la souris\n");
        }
    }
```

Pour pallier cet inconvénient, on peut définir un écouteur en utilisant une classe "adapter"

- Une classe Adapter est une classe qui implémente une interface listener mais ses méthodes n'ont pas de code, donc elles ne font rien
- On n'implémente donc que les méthodes qui nous intéressent. Les autres sont déjà définies et on ne s'en soucie pas
- Il y a une classe Adapter pour les listeners qui possèdent beaucoup de méthodes

Les classes Adapter

FocusAdapter

MouseAdapter

WindowAdapter

KeyAdapter

MouseMotionAdapter

MouseInputAdapter

Chacune implémente toutes les méthodes de l'interface correspondant

Affichage des coordonnées du clic souris

Démo

```
public class FenClicSouris extends JFrame{
   private JLabel lab;
public void initconnections(){
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.getContentPane().addMouseListener(new CliqueAdapter())  ;
    }
```

On définit juste la méthode qui nous intéresse

Inner-classe

```
class CliqueAdapter extends MouseAdapter {

public void mouseClicked(MouseEvent e){
        int x = e.getX() ;
    int y = e.getY() ;
        lab.setText(" clic dans la fenêtre en ("+ x + "," + y + ")");
     }
 }  // fin de Clique
  }// fin de FenClicSouris
```

On détermine dans quelle partie de la fenêtre le clic s'est produit

On l 'affiche dans la zone de texte