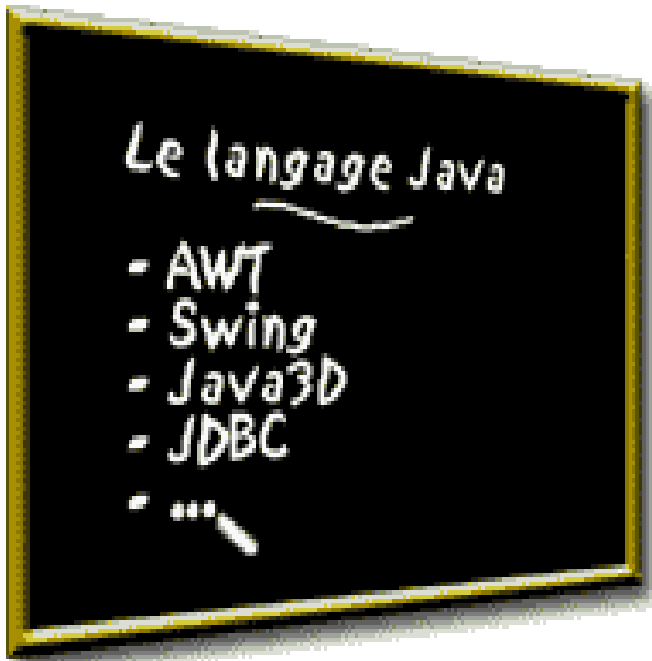


# Cours 3 : Programmation événementielle



Les événements

les écouteurs d'événements

Programmer l'événementiel

Un exemple

Deux méthodes pour distinguer de quelle source vient un événement

Les adaptateurs

# Rappels : action/événement

Action de l'utilisateur dans la fenêtre de l'application

==> Interruption matérielle

==> Récupération de l'interruption par le système

==> Construction de l'événement correspondant

<== Envoi d'un message à l'application concernée avec des informations sur l'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  
Cliquer sur son bouton gauche



Événement « souris »

# Principes de la version 1.1 et 1.2

- Pour être prévenu de l'arrivée d'un événement et pour y réagir, tout composant graphique (bouton, zone de texte, container...) doit se déclarer "à l'écoute" de l'événement

Le composant s'appelle **la source**

- Pour cela, il doit se lier à un **écouteur** d'événement à qui il va déléguer le traitement de l'événement

L'écouteur s'appelle **la cible**

**➔ Principe de délégation**

# Comment ça se passe ?

■ Si un événement se produit dans un composant, 2 cas :

1) le composant ne s'est pas déclaré à l'écoute de cet événement

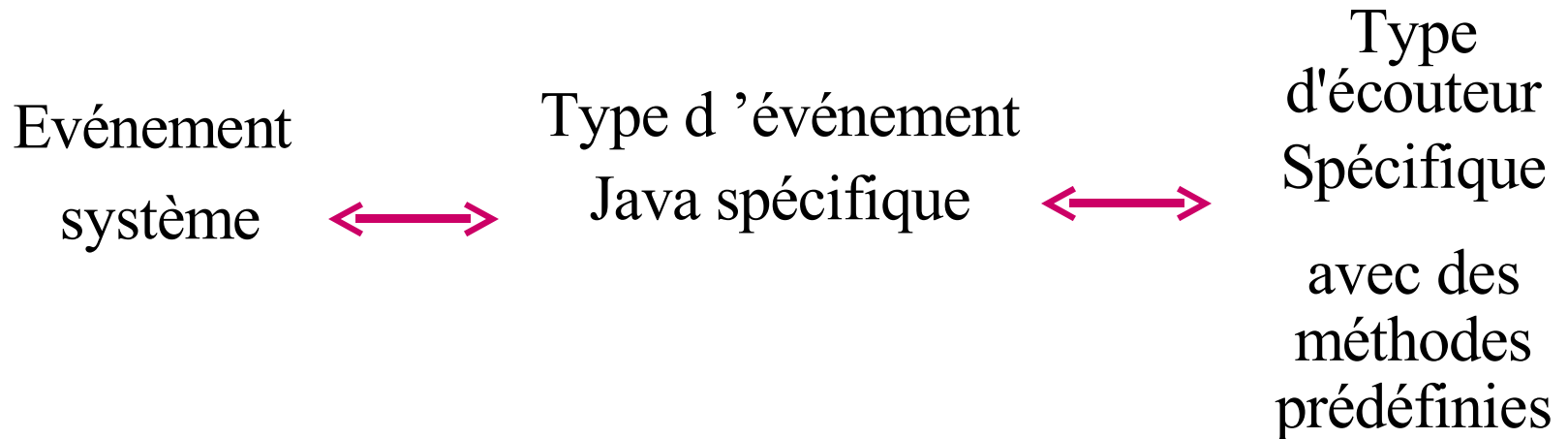
→ rien ne se passe

2) le composant s'est déclaré à l'écoute de cet événement en s'associant à un écouteur de ce type d'événement

→ cet écouteur est prévenu et agit en conséquence

# Lien Événement/Ecouteur/méthode

- Le traitement de l'événement est **délégué** à l'écouteur et l'écouteur exécute une méthode spécifique qui prend en paramètre l'événement qui s'est produit



# Classes et interfaces utilisées

■ Java utilise :

⇒ Des classes **Event** pour représenter les événements

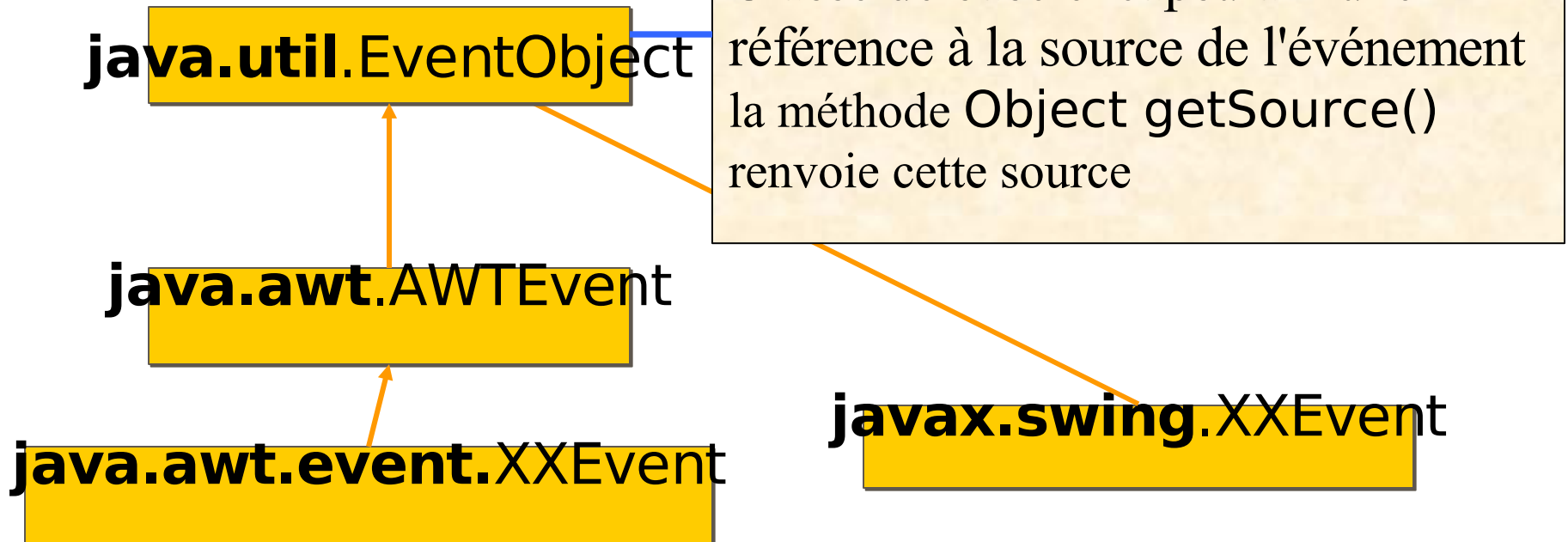
**FocusEvent, MouseEvent, KeyEvent,  
WindowEvent,  
ActionEvent, TextEvent, ListSelectionEvent,...**

⇒ Des interfaces **Listener** pour représenter les écouteurs

**FocusListener, MouseListener, KeyListener,  
WindowListener,  
ActionListener, TextListener,  
ListSelectionListener,...**

# Les événements : classes XEvent

- Un événement est représenté par un objet d'une classe donnée
- Classes organisées en hiérarchie

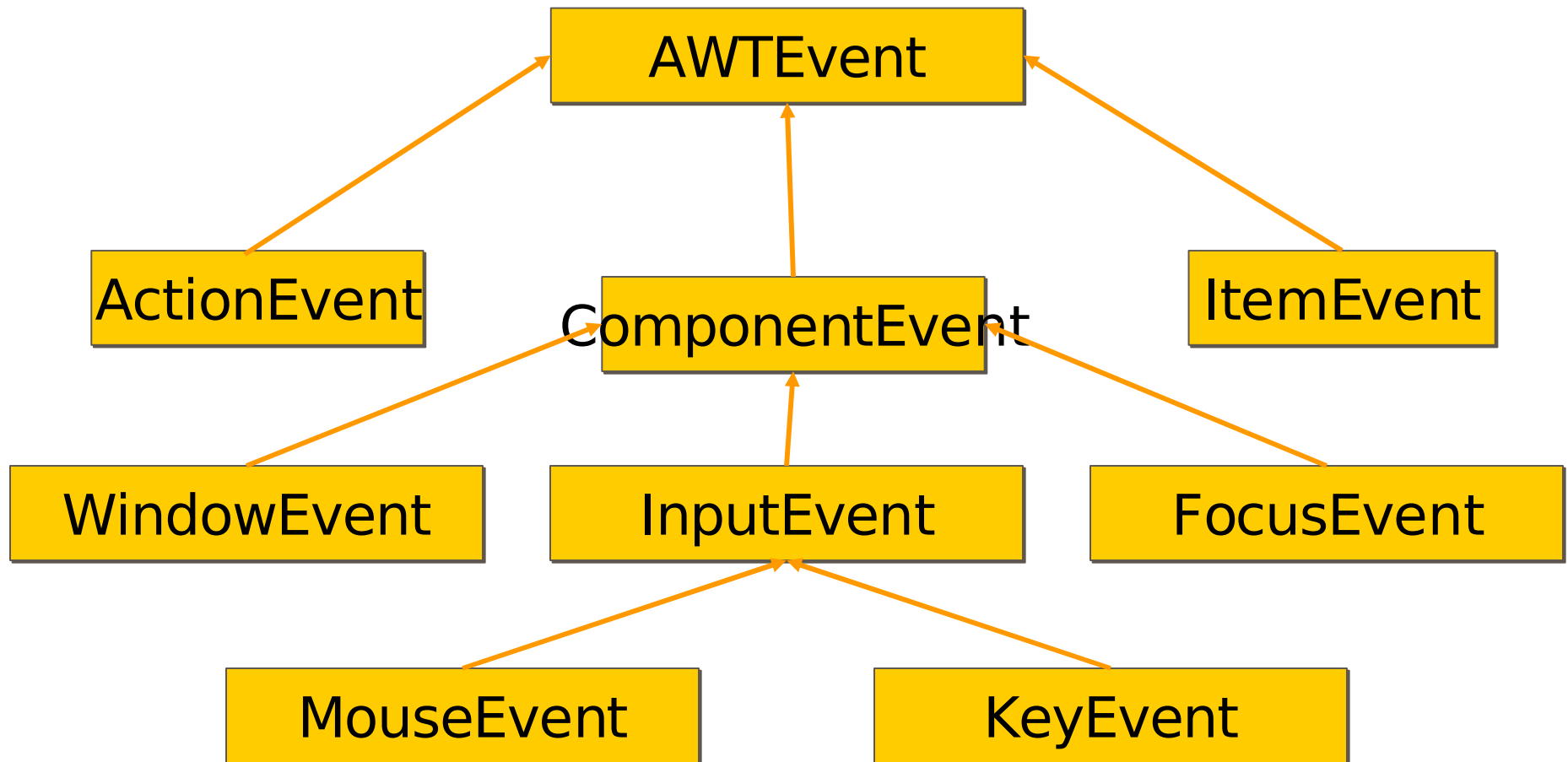




# Les événements (suite)

- Un objet événement mémorise :
  - sa source,
  - des informations caractéristiques de l'événement
- Des méthodes spécifiques (dans chaque sous-classe) permettent d'accéder à ces informations
- Exemples :
  - Pour un objet, `ke`, de la classe `KeyEvent` : caractère frappé  
`char c = ke.getKeyChar();`
  - Pour un objet, `me`, de la classe `MouseEvent` : position de la souris  
`int x = me.getX();`  
`int y = me.getY();`

# hiérarchie (partielle) des événements



# Les écouteurs d'événements : interfaces XXListener



- Chaque interface contient un ensemble de méthodes spécifiques (abstraites) correspondant au traitement de l'événement
- Toute méthode de tout écouteur a la forme :  
public void methodeEcouteur(XXEvent e)

# Lien Événement/Listener/méthode

- Ensemble de triplets à connaître :
  - Composant
  - Événement pouvant se produire dans le composant
  - Listener associé
- Exemples de triplets :
  - (JButton, ActionEvent, ActionListener)
  - (JFrame, WindowEvent, WindowListener)
- Chaque listener possède les méthodes à implémenter (**les signatures des méthodes sont données**)

# Exemples de liens

## Événement/Listener/méthode (1)

Type d'évènement

**ActionEvent**

Produit par les objets

**Boutons** : JButton, JRadioButton, JCheckBox

**Menus**: JMenuItem, JMenu, JCheckBoxMenuItem, JRadioButtonMenuItem

Interface listener associée

ActionListener

méthodes

**void actionPerformed(ActionEvent e)**

**Texte** : JTextField

# Exemples de liens

## Événement/Listener/méthode (2)

Type d'événement

**ItemEvent**

Produit par les objets

**Boutons** : JRadioButton,  
JCheckBox

interface listener associée

ItemListener

**Menus**: JCheckBoxMenuItem  
JRadioButtonMenuItem

méthodes

**void itemStateChanged(ItemEvent e)**

## Exemples de liens Événement/Listener/méthode (3)

Type d'évènement

**WindowEvent**

Produit par les fenêtres

Interface listener associée

Méthodes de l'interface

**WindowListener**

**void windowOpened(WindowEvent e)**

**void windowClosing(WindowEvent e)**

**void windowClosed(WindowEvent e)**

**void windowActivated(WindowEvent e)**

**void windowDeactivated(WindowEvent e)**

**void windowIconified(WindowEvent e)**

**void windowDeiconified(WindowEvent e)**

7 méthodes

# Exemples de liens

## Événement/Listener/méthode (4)

Type d'évènement

**MouseEvent**

Produit par un clic de souris  
dans un composant

Méthodes de l'interface

**Interface listener associée**

**MouseListener**

**void mouseClicked(MouseEvent e)**

**void mousePressed(MouseEvent e)**

**void mouseReleased(MouseEvent e)**

**void mouseEntered(MouseEvent e)**

**void mouseExited(MouseEvent e)**

**5 méthodes**



# Comment programmer ?

## 1. Première partie : créer une classe Listener (écouteur) adaptée

- Tout objet, toute classe peut devenir un listener  
Il suffit que la classe implémente l'interface **Listener** choisie

- | Mais il est préférable de créer une classe spécifique dont le seul rôle est de traiter l'événement et à qui on délègue le traitement

## 2. Deuxième partie : relier la source à la cible ~ relier le composant à une instance de ce Listener (l'écouteur).

Exemple : un clic dans un bouton

Composant :

Evénement :

Listener :

# 2 parties obligatoires de code pour qu'un composant réagisse à un événement donné

## Première partie

### A) Implémenter l'écouteur correspondant spécifique.

==> Créer une classe qui implémente l'interface `XXListener` adéquate

==> Donner un corps à l'ensemble des méthodes prévues dans l'interface

```
class BoutonOkListener implements  
ActionListener {
```

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

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

# 2 parties obligatoires de code pour qu'un composant réagisse à un événement donné

## Deuxième partie

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

`void addXXListener(XXListener 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 FenBoutonEvt extends JFrame {  
    private JLabel labNbFois;  
    private int nb;  
  
    public FenBoutonEvt(String titre,int w, int h) {  
        super(titre);  
  
        nb =0;  
  
        this.setSize(w,h);  
  
        this.initialise();  
        this.show();  
    }  
}
```

Démo

```
private void initialise() {  
    JButton b=new JButton("Clique et tu  
verras... ");
```

```
    labNbFois= new JLabel();  
    Container c=this.getContentPane();  
    c.setLayout(new FlowLayout());  
    c.add(b);  
    c.add(jtNbFois);
```

Création d'une instance d'écouteur

```
    BoutonListrener blis=new  
    BoutonListener();  
    b.addActionListener(blis);
```

Lien bouton/écouteur

```
} class BoutonListener implements  
ActionListener {
```

```
    public void  
    actionPerformed(ActionEvent e) {  
        nb = nb+1;  
        labNbFois.setText( nb + "  
clics");    }
```

Inner-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 `FenBoutonEvt.java` génère deux fichiers : `FenBoutonEvt.class`,  
`FenBoutonEvt$BoutonListener.class`

# Caractéristiques d'une inner-classe

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

Exemple : nb, tNbFois

- Elle peut appeler directement les méthodes de la classe qui l'englobe

```
class BoutonListener implements ActionListener {  
    public void actionPerformed(ActionEvent e) {
```

```
        FenBoutonEvt.this.....
```

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

# Remarques sur les écouteurs

- On peut "enregistrer" plusieurs Listeners auprès d'un même composant si l'on souhaite que le composant réagisse à différents types d'événements
- Un même écouteur peut être enregistré auprès de plusieurs composants. C'est ce qu'on fait souvent pour des composants de même classe
  - plusieurs boutons --> une seule classe écouteur implémentant ActionListener
  - Dans ce cas, il faut que l'écouteur puisse différencier de quelle source vient l'événement qui lui a été passé



# 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** : se servir de l'événement qui s'est produit et qui est accessible dans l'écouteur car présent en paramètre de la méthode
  - **Méthode 2** : donner à l'écouteur les moyens de reconnaître la source de l'événement en passant des arguments à son constructeur

# Méthode 1 : se servir de l'événement qui s'est produit et qui est accessible dans 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 méthode `String getActionCommand()` de `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 Container c;
private void initialise() {
    c=this.getContentPane();
    c.setLayout(new FlowLayout());
    JButton bRouge=new JButton("Fond rouge");
    bRouge.setForeground(Color.red);
    c.add(bRouge);
    JButton bBleu=new JButton("Fond bleu");
    bBleu.setForeground(Color.blue);
    c.add(bBleu);
    BoutonListener list=new BoutonListener();
    bRouge.addActionListener(list);
    bBleu.addActionListener(list);
}
```

```
public FenDeuxBoutons(String titre, int w, int h) {  
    super(titre);          this.setSize(w,h);  
    this.initialise();    this.show();  
}
```

```
class BoutonListener implements  
ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        String s=e.getActionCommand();  
        if (s.equals("Fond bleu"))  
            c.setBackground(Color.blue);  
        else c.setBackground(Color.red);  
    }  
}
```

→ Inner-classe

```
} // fin de l' 'inner-classe BoutonListener  
public static void main (String args[]) {  
    new FenDeuxBoutons("Fenêtre avec deux boutons",800,600);  
    } // fin du main  
} // fin de la classe FenDeuxBoutons
```

## Méthode 2 : donner à l'écouteur les moyens de reconnaître la source de l'événement en passant des arguments à son constructeur

ici grâce à des mnémoniques de type int

```
public class FenDeuxBoutonsMnemo extends JFrame
{
    private static final int BLEU=0, ROUGE=1;
    private Container c;
    public void initialise() { ....
    bRouge.addActionListener(new BoutonListenerMnemo(ROUGE));
    bBleu.addActionListener(new BoutonListenerMnemo(BLEU));
    }
```

Mnémoniques, constantes de classe

Écouteurs construits avec valeurs particulières

```
class BoutonListenerMnemo implements ActionListener {
    private int val;
    public BoutonListenerMnemo(int i) {
        val=i;
    }
```

Variable membre mémorisant la valeur

Le constructeur

```

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

```

Pour savoir quelle est la source :

tester la valeur de val

val est de type int :


⇒ switch est possible

### Intérêts :

\* on gagne en lisibilité

\* switch est possible

# Ecouteur = 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



```
public class FenSourisListener extends JFrame
{
```



```
    private JTextArea jta;
```

```
public void initialise() {
```

```
    jta = new JTextArea(12,60 );
```

Intercepter les mouvements

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

souris

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

```
    c.add(jta);
```

```
    this.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 mouseClicked(MouseEvent e)
```

```
{jta.append("clic sur la
```

```
souris\n");
```

Il y en a bien 5 !!

# 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

```
public class FenClicSouris extends JFrame {
    private JLabel lab;

```

Affichage des coordonnées du clic souris



```
public void initialise() {
    lab=new JLabel();
    Container c=this.getContentPane();
    c.add(lab);
    c.setLayout(new FlowLayout());
    this.addMouseListener(new CliqueAdap
}

```

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 + ")");
    }
}

```

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

On l'affiche dans la zone de texte

```
} // fin de Clique

```