

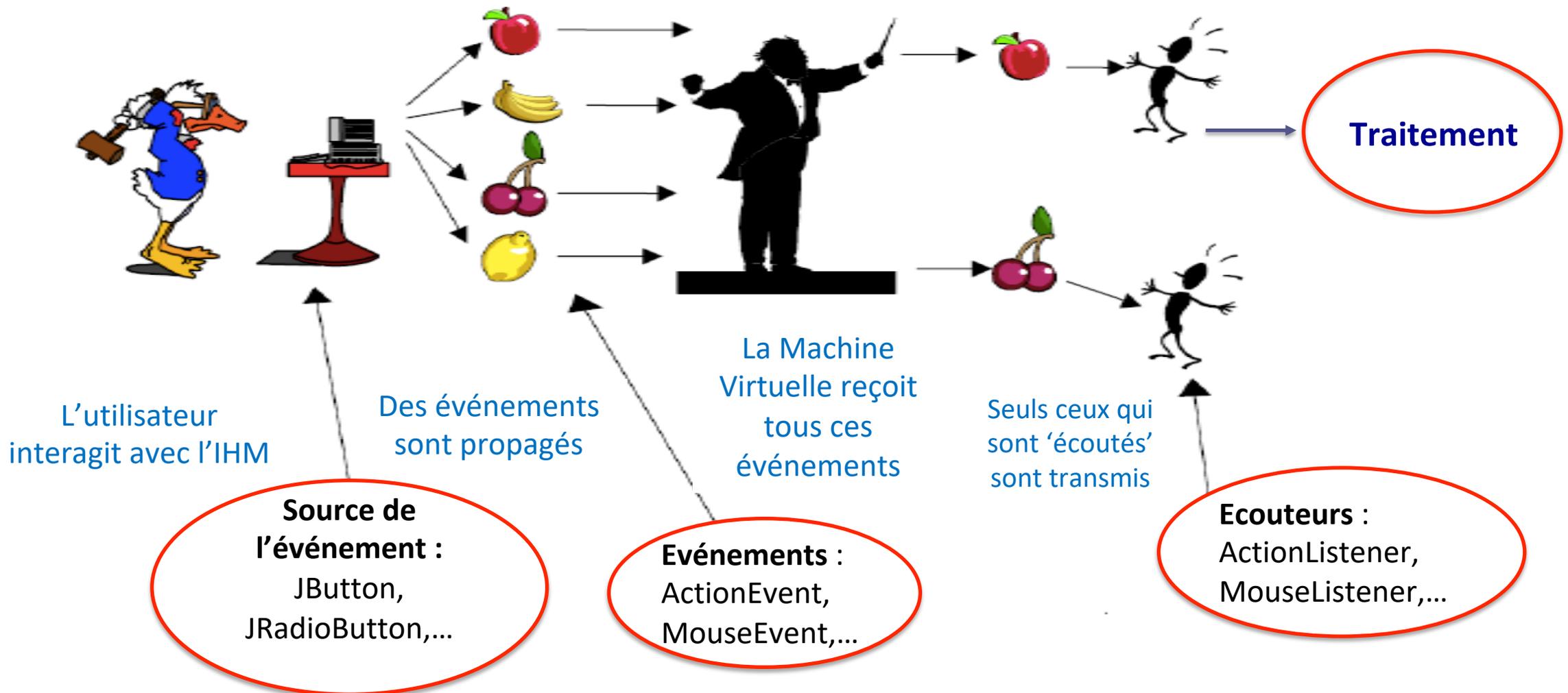
Développement d'applications avec IHM - Cours 2

Gestion des évènements

V. DESLANDRES

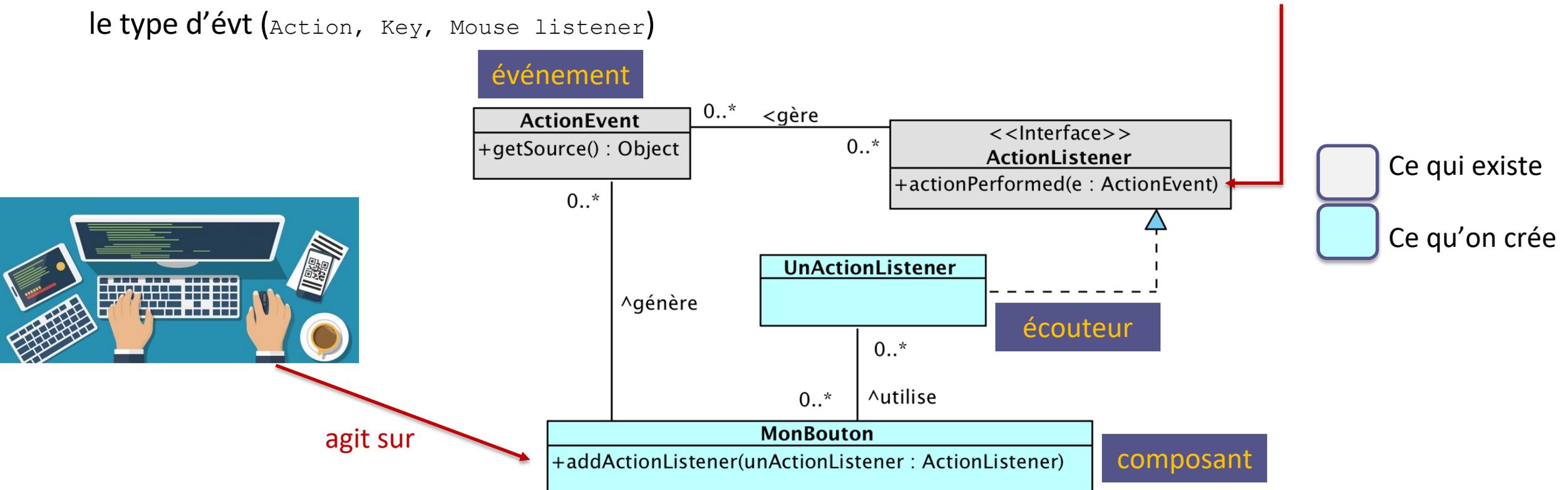
veronique.deslandres@univ-lyon1.fr
2021/2022

Propagation des évènements



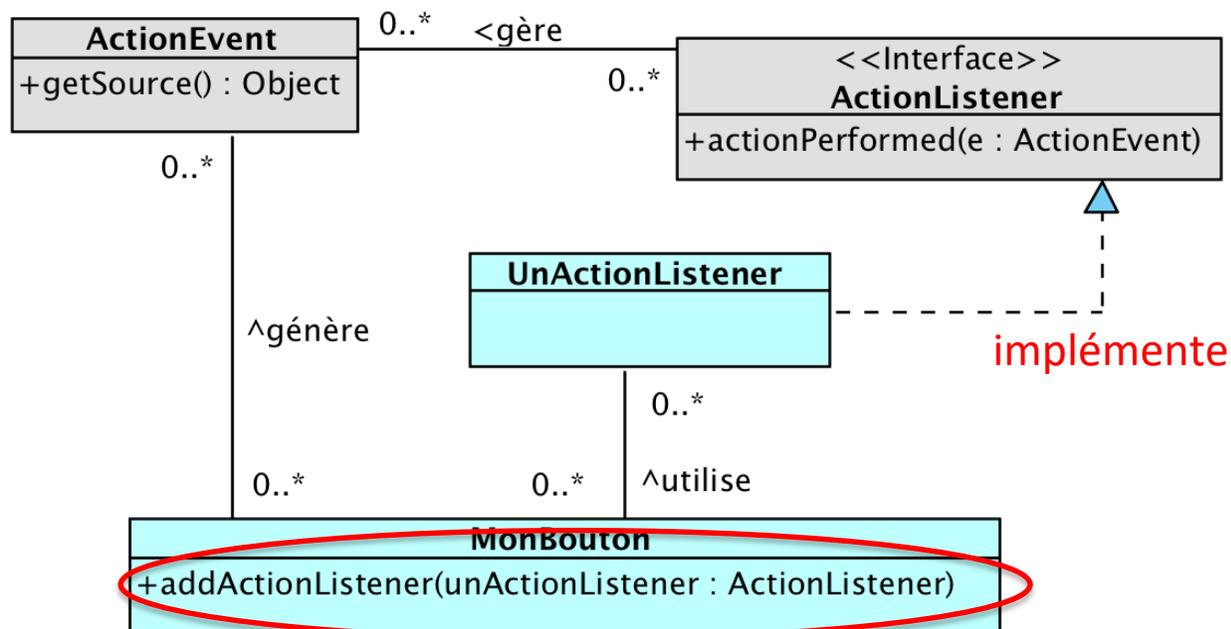
Principe (1/2)

- Un composant peut générer certains **événements**
- Un objet **événement** reflète une **action** de l'utilisateur (clic, passage souris, redimensionnement de la fenêtre, etc.)
- La gestion de l'évènement est déléguée à un **écouteur** d'évènements qui active le **traitement** associé selon le type d'évt (`Action`, `Key`, `MouseListener`)



Principe (2/2)

- Les écouteurs sont des objets qui **implémentent des interfaces prédéfinies** (MouseListener, ActionListener)
- Ces écouteurs doivent être **explicitement** affectés aux composants concernés
`monBouton.addActionListener(new UnActionListener());`



Package événements AWT

```
java.lang.Object
|
+--java.util.EventObject
|
+--java.awt.AWTEvent import java.awt.AWTEvent.*;
|
+--java.awt.event.ActionEvent
+--java.awt.event.AdjustmentEvent
+--java.awt.event.AncestorEvent
+--java.awt.event.ComponentEvent
|
+--java.awt.event.ContainerEvent
+--java.awt.event.FocusEvent
+--java.awt.event.InputEvent
|
+--java.awt.event.KeyEvent
+--java.awt.event.MouseEvent
+--java.awt.event.PaintEvent
+--java.awt.event.WindowEvent
+--java.awt.event.HierarchyEvent
+--java.awt.event.InputMethodEvent
+--java.awt.event.InternalFrameEvent
+--java.awt.event.InvocationEvent
+--java.awt.event.ItemEvent
+--java.awt.event.TextEvent
```

événements de bas niveaux

Événements de la Swing

- Le paquet `javax.swing.event` définit quelques événements supplémentaires
 - (`MouseEvent`, `CaretEvent`, ...)
 - à partir de `EventObject`
- événements liés au curseur clignotant*
- Mais Swing réutilise principalement les **événements d'AWT**

Démarche

En résumé, il faut :

- Identifier les **objets source** et le **type d'événement**, donc l'**écouteur** nécessaire,
- Personnaliser le traitement des événements en **implémentant les méthodes** des écouteurs,
- **Relier** les objets écouteurs aux objets sources pour permettre le traitement.

- Exemple :

- Augmenter l'épaisseur d'un cadre **quand on clique sur la touche +** nécessite :
 - Composant source = `JPanel` avec un bord visible
 - Évènement : `KeyEvent` (touche +)
 - Écouteur : `KeyListener` qui, quand '+' est tapé, va augmenter l'épaisseur du bord

`char c=e.getKeyChar(); if (c=='+')...`

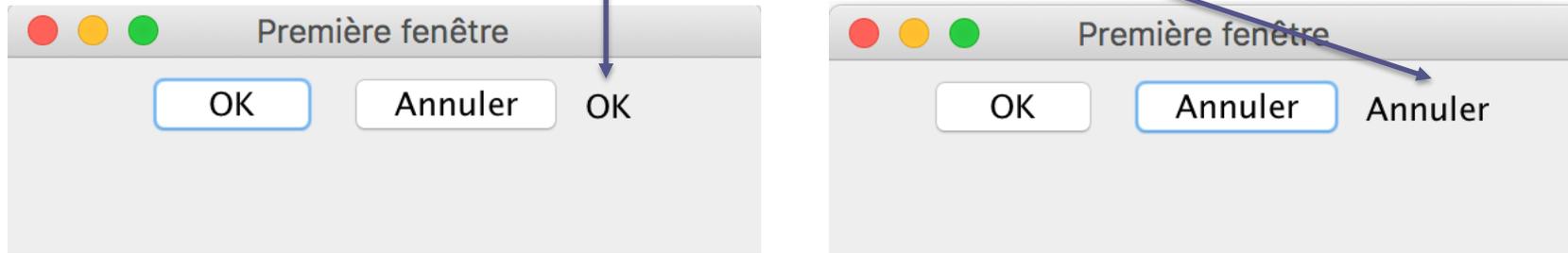
3 façons pour implémenter les écouteurs

- Définir les classes écouteurs en **classes internes**
 - Définir une classe `xxxListener` au sein de la classe principale
- Faire que la classe principale soit son **propre écouteur**
 - Très fréquent
 - Utilisé pour les **IHM simples**
- Coder un écouteur en **classe anonyme**
 - (classe écouteur sans nom)
 - Là où on ajoute un écouteur
 - Lorsque le code associé au traitement est court



Exemple fil rouge

Le label affiché contiendra un texte différent selon le bouton cliqué



1^{ère} implémentation d'un écouteur :
classe écouteur en **classe interne**

Écouteur défini en classe interne

- Créer la ou les **classes internes** (au sein de la classe principale)
 - Il faut indiquer que la classe interne **est un écouteur** : elle **implémente** l'interface `XXXListener`
- **Dans la classe interne, redéfinir** la ou les méthodes de l'interface d'écoute pour les événements à gérer :
 - Ex.: la méthode `actionPerformed()` si l'événement est un `ActionEvent`
 - L'événement est passé en paramètre
- Le composant doit ajouter un **écouteur** pour **chaque classe** d'événements à traiter, par exemple:
 - `boutonA.addActionListener(new MonEcouteurAction());`
 - `boutonA.addMouseListener(new MonEcouteurSouris());`

```

public class TestEvents1 extends JFrame{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents1(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        pane=(JPanel) getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);
        ButOK.addActionListener(new OkButtonListener());
        ButAnnuler.addActionListener(new AnnulerButtonListner());
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents1();
        frame.setTitle("Première fenêtre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }
}

```

Ce label contiendra un texte différent selon le bouton cliqué

Ajouter à chaque bouton l'écouteur associé

Ici on choisit de définir un écouteur interne pour chaque bouton

```

class OkButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e){
        label.setText("OK");
    }
};

```

```

class AnnulerButtonListner implements ActionListener {
    public void actionPerformed(ActionEvent e){
        label.setText("Annuler");
    }
};

```

On sait que les événements liés à l'action des boutons sont des **ActionEvent** : on implémente donc **ActionListener**, qui n'a qu'une seule méthode: **actionPerformed()**

```

public class TestEvents2 extends JFrame{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents2(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        pane=(JPanel) getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);
        ButOK.addActionListener(new MyButtonListner());
        ButAnnuler.addActionListener(new MyButtonListner());
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents2();
        frame.setTitle("Première fenêtre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }
}

```

```

class MyButtonListner implements ActionListener {
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==ButOK){
            label.setText("OK");
        }
        else
            label.setText("Annuler");
    }
};

```

Ici on définit une seule **classe écouteur interne** pour tous les composants

On doit donc tester dans la méthode, **de quel bouton** provient l'évt capté : la méthode `getSource()` d'un `ActionEvent` retourne le composant concerné

Discussion : implémentation en classe interne

- Gestion des événements bien isolée dans une classe
- Rajoute du code

On choisira la classe interne si cette classe n'a **aucun intérêt à être exposée** aux autres classes ; elle peut être utilisée par différents composants de la classe principale (**factorisation**).

C'est donc un choix intéressant si on a **beaucoup d'actions différentes** à coder en fonction des composants sollicités.

2^{ème} implémentation d'un écouteur :
la classe principale est son **propre** écouteur

La classe principale est son propre écouteur

- Indiquer que la classe principale **implémente** la ou les interfaces `XXXListener`
- *Comme dans la 1^{ère} implémentation :*

Redéfinir la ou les méthodes de l'écouteur pour les événements à gérer (*mais cette fois dans la classe même*) :

- Ex.: la méthode `actionPerformed()` si l'écouteur est un `ActionListener` pour un `ActionEvent`
- Cette fois, chaque composant qui va réagir doit ajouter l'**écouteur** (qui est la classe) pour toutes les classes d'événements à traiter :
 - Exemple:
 - `boutonA.addActionListener(this);`
 - `boutonA.addMouseListener(this);`

```

public class TestEvents3 extends JFrame implements ActionListener{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents3(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        pane=(JPanel) getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);

        ButOK.addActionListener(this);
        ButAnnuler.addActionListener(this);
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents3();
        frame.setTitle("Première fenêtre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent e){
        if(e.getSource()==ButOK){
            label.setText("OK");
        }
        else
            label.setText("Annuler");
    }
}

```

implements ActionListener{

Indiquer que la classe principale implémente l'interface
XXXListener

Ajouter à chaque composant l'objet
écouteur associé, qui est la classe elle-
même (this)

Redéfinir les méthodes de l'interface
XXXListener (ici actionPerformed())
pour définir le traitement

Discussion : la classe est son propre écouteur

- Pas de code en plus
- Gestion des événements noyée dans le code de la classe

*Rmq : ce choix d'implémentation n'est pas limitant pour la définition de la classe, puisque l'héritage multiple **d'interfaces** est possible en Java.*

Pour la **lisibilité** du code, on utilisera ce mode quand on a *peu* de traitements à coder (*peu ?* à définir en fonction du contexte).

3^{ème} implémentation d'un écouteur :

Coder un écouteur **en classe anonyme**

Coder un écouteur en classe anonyme

- Lorsque l'action relative à un évènement est **simple** on passe par une **classe d'écouteur anonyme** pour définir les observateurs associés au composant
 - Rappel classe anonyme : classe instanciée sans nom, avec `new NomClasse()` {... code de la classe }
 - Nota : dans le cas d'interface, le code `new InterfaceEcouteur()` crée en fait **une classe anonyme** qui implémente l'interface...
- Plus besoin d'indiquer que la classe *implémente* l'interface écouteur
- Cela permet de définir **un écouteur par composant source**

```

public class TestEvents4 extends JFrame {
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents4(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        pane=(JPanel) getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);

        ButOK.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e){
                label.setText("OK");
            }
        });

        ButAnnuler.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e){
                label.setText("Annuler");
            }
        });
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents4();
        frame.setTitle("Première fenêtre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }
}

```

Pas d'implémentation d'interface XXXListener

L'écouteur est placé pour **chaque composant**

Ecouteur instancié en **classe anonyme**

On ne code que les méthodes qui seront utilisées

Discussion : écouteur codé en classe anonyme

- Pas de code en plus
- Pas de déclaration en entête de classe des écouteurs utilisés par la classe
- Ecouteurs moins visibles, noyé dans le code
- Pas de déclaration en entête de classe : lisibilité des dépendances moins facile
- Code parfois difficile à lire

Mode utilisé par les IDE générant automatiquement le code des IHMs. Assez utilisé en réalité car peu contraignant au niveau de la déclaration de la classe, mais code difficile à maintenir.

Récapitulatif (Interfaces écouteurs, adaptateurs, évènements, méthodes)

Interfaces écouteurs - Description - Composants

Interface écouteur	Description	Composant qui génère l'événement
ActionListener	<p>Ecoute les actions de l'utilisateur : clics, barre d'espace, entrée</p> <ul style="list-style-type: none">• JButton, JRadioButton et JCheckBox : clic gauche et barre d'espace• JComboBox : déplacement dans la zone déroulante avec les flèches de déplacement• JTextField : touche Entrée	JButton, JRadioButton, JCheckBox JComboBox, JTextField
AdjustmentListener	Ajustement d'une barre de défilement	JScrollBar
ComponentListener	Déplacement, affichage, masquage ou modification de la taille de composants (ex. : une fenêtre)	Component
ContainerListener	Ajout ou suppression d'un composant dans un conteneur	Container
FocusListener	Obtention ou perte du focus par un composant	Component
ItemListener	Sélection d'un élément dans un combobox ou dans une liste (mais pas avec JList) ou dans un groupe de cases à cocher	Checkbox, JComboBox, List, JRadioButton...
KeyListener	Ecoute les KeyEvent : actions sur les touches du clavier (pressée ou relâchée)	Component
DocumentListener	Changement ou insertion dans un document	Document

Interfaces écouteurs - Description - Composants (Suite)

Interface écouteur	Description	Composant qui génère l'événement
ListSelectionListener	Écouteur qui notifie quand la sélection d'éléments dans une liste change (L'utilisateur sélectionne (désélectionne) un item ou plusieurs items dans une <code>JList</code> , une <code>JTable</code> , ...)	<code>JList</code> , <code>JTable</code> , ...
MouseListener	Action sur la souris (clic sur un bouton de la souris: appuyer, relâcher, etc.)	Component
MouseMotionListener	Action de la souris sur un composant (événements de glisser-déplacer)	Component
WindowListener	Action sur la fenêtre (fenêtre activée, désactivée, réduite, fermée, ...)	Window
TextListener	Changement sur une zone de texte	<code>JTextField</code>

Interfaces écouteurs - Traitement et enregistrement

Interface écouteur	Méthodes de traitement	Méthode d'enregistrement
ActionListener	<code>actionPerformed(ActionEvent e)</code>	<code>addActionListener()</code>
AdjustmentListener	<code>adjustmentValueChanged(AdjustmentEvent e)</code>	<code>addAdjustmentListener()</code>
ComponentListener	<code>componentHidden(ComponentEvent e)</code> <code>componentMoved(ComponentEvent e)</code> <code>componentResized(ComponentEvent e)</code> <code>componentShown(ComponentEvent e)</code>	<code>addComponentListener()</code>
ContainerListener	<code>componentAdded(ContainerEvent e)</code> <code>componentRemoved(ContainerEvent e)</code>	<code>addContainerListener()</code>
FocusListener	<code>focusGained(FocusEvent e)</code> <code>focusLost(FocusEvent e)</code>	<code>addFocusListener()</code>
ItemListener	<code>itemStateChanged(ItemEvent e)</code>	<code>addItemListener()</code>
KeyListener	<code>keyPressed(KeyEvent e)</code> <code>keyReleased(KeyEvent e)</code> <code>keyTyped(KeyEvent e)</code>	<code>addKeyListener()</code>
DocumentListener	<code>changedUpdate(DocumentEvent e)</code> <code>insertUpdate(DocumentEvent e)</code> <code>removeUpdate(DocumentEvent e)</code>	<code>addDocumentListener()</code>

*Enregistrement de l'écouteur sur un composant :
`monBouton.addActionListener(xxx);`*

Interfaces écouteurs - Traitement et enregistrement (Suite)

Interface écouteur	Méthodes de traitement	Méthode d'enregistrement
ListSelectionListener	<code>valueChanged (ListSelectionEvent e)</code>	<code>addListSelectionListener ()</code>
MouseListener	<code>mouseClicked (MouseEvent e)</code> <code>mouseEntered (MouseEvent e)</code> <code>mouseExited (MouseEvent e)</code> <code>mousePressed (MouseEvent e)</code> <code>mouseReleased (MouseEvent e)</code>	<code>addMouseListener ()</code>
MouseMotionListener	<code>mouseDragged (MouseEvent e)</code> <code>mouseMoved (MouseEvent e)</code>	<code>addMouseMotionListener ()</code>
WindowListener	<code>windowActivated (WindowEvent e)</code> <code>windowClosed (WindowEvent e)</code> <code>windowClosing (WindowEvent e)</code> <code>windowDeactivated (WindowEvent e)</code> <code>windowDeiconified (WindowEvent e)</code> <code>windowIconified (WindowEvent e)</code> <code>windowOpened (WindowEvent e)</code>	<code>addWindowListener ()</code>
TextListener	<code>textValueChanged (TextEvent e)</code>	<code>addTextListener ()</code>

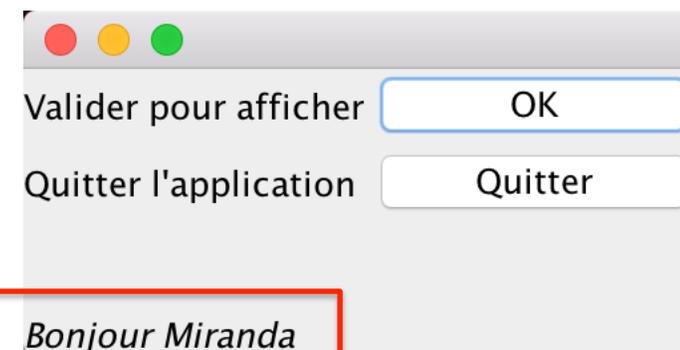
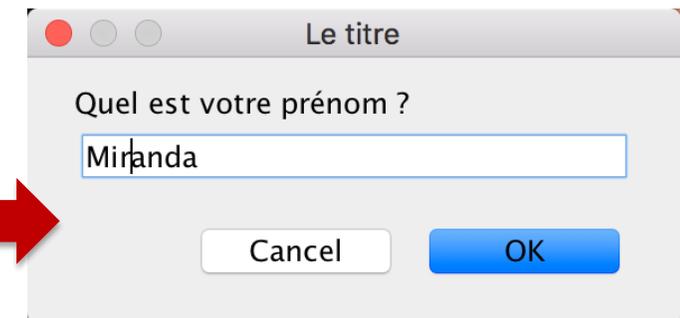
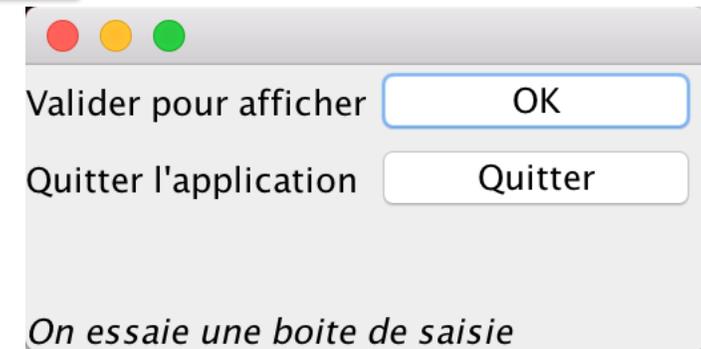
Exemple: MouseListener

```
public class BoiteInfo3 extends JFrame implements MouseListener {
```

```
    JLabel label;  
    JButton btnOK, btnQUIT;
```

```
@Override
```

```
public void mouseClicked(MouseEvent e) {  
    System.out.println("Un bouton a été cliqué !");  
    if (e.getSource() == btnOK) {  
        String message = "Quel est votre prénom ?";  
  
        String reponse = JOptionPane.showInputDialog(this, message,  
            "Le titre", JOptionPane.PLAIN_MESSAGE);  
  
        if (reponse != null && reponse.length() != 0) {  
            label.setText("Bonjour " + reponse);  
        }  
    } else if (e.getSource() == btnQUIT) {  
        System.exit(0);  
    }  
}
```



Limites des interfaces écouteurs

- Une classe qui implémente une interface doit définir **toutes** les méthodes de l'interface
- Or certaines interfaces `listener` ont beaucoup de méthodes
 - Certaines n'ayant aucune utilité
- Dans l'exemple précédent : on a dû définir toutes les méthodes de l'interface `MouseListener`

```
@Override
public void mousePressed(MouseEvent e) {
}

@Override
public void mouseReleased(MouseEvent e) {
}

@Override
public void mouseEntered(MouseEvent e) {
}

@Override
public void mouseExited(MouseEvent e) {
}
```

Les classes Adaptateurs

Obj.: simplifier les interfaces écouteurs

Classes Adaptateurs

- Pour remédier aux limites des interfaces écouteurs, Java propose des classes `Adapter`
- Ces classes sont valables pour **certaines** interfaces `Listener`
- Au lieu **d'implémenter** l'interface il faut **hériter** de la classe `Adapter`
 - Dans ce cas, le développeur ne peut redéfinir que les méthodes dont il a besoin
 - Limite : la classe ne doit déjà hériter d'aucune autre classe (pas d'héritage multiple en java)
- Principe de nommage très simple:
 - une classe nommée `XXXAdapter` correspond à l'interface écouteur `XXXListener`

Interfaces écouteurs - Adaptateurs - Évènements

Interface écouteur	Adaptateur	Evènements
ActionListener	-	ActionEvent
AdjustmentListener	-	AdjustmentEvent
ItemListener	-	ItemEvent
FocusListener	FocusAdapter	FocusEvent
KeyListener	KeyAdapter	KeyEvent
MouseListener	MouseAdapter	MouseEvent
MouseMotionListener	MouseMotionAdapter	MouseMotionEvent
WindowListener	WindowAdapter	WindowEvent
TextListener	-	TextEvent

Exemple: gérer les évènements de la souris

- Si **moins** des 5 méthodes sont nécessaires pour l'interface graphique :
 - Utiliser la classe **Adapter** : `MouseAdapter`
 - Et ne redéfinir que les méthodes souhaitées (e.g., `mouseClicked`)

```
public class MaClasseEcouleur extends MouseAdapter {  
    // surcharger les méthodes utiles (<5)  
}
```

(Ce code est écrit par défaut dans l'API Swing, inutile de le réécrire),
`MouseAdapter` existe

```
interface MouseListener {  
    // Déclare cinq méthodes  
}  
class MouseAdapter implements MouseListener {  
    // implémentation vide des 5 méthodes  
}
```

```
public class BoiteInfo31 extends JFrame {
```

```
...  
lePanneau.add(new JLabel("Valider pour afficher"));  
btnOK = new JButton("OK");  
btnOK.addMouseListener(new MonEcouteur());  
lePanneau.add(btnOK);  
...
```

```
private class MonEcouteur extends MouseAdapter {
```

```
@Override  
public void mouseClicked(MouseEvent e) {  
    System.out.println("Un bouton a été cliqué !");  
    if (e.getSource() == btnOK) {  
        String message = "Quel est votre prénom ?";
```

Illustration Adapter 1: classe écouteur en classe interne

On n'utilise plus
implements `MouseListener`
mais
extends `MouseAdapter`

Illustration Adapter 2: la classe principale est son propre écouteur

On ne met plus `extends JFrame` (car on ne peut pas faire d'héritage multiple)

On déclare donc la fenêtre dans les attributs et on l'initialise dans le constructeur

```
public class BoiteInfo4 extends MouseAdapter {  
  
    private JLabel label;  
    private JButton btnOK, btnQUIT;  
    private JFrame laFrame;  
  
    ...  
    // constructeur  
    BoiteInfo4() {  
        laFrame = new JFrame();  
        label = new JLabel("On essaie une boîte de saisie");  
        Font maPolice = new Font(label.getFont().getName(), Font.ITALIC, label.get  
        label.setFont(maPolice);  
  
        ...  
  
        laFrame.pack();  
        laFrame.setVisible(true);  
    } // fin du constructeur  
  
    ...  
    btnOK.addMouseListener(this);  
  
    @Override  
    public void mouseClicked(MouseEvent e) {  
        System.out.println("Un bouton a été cliqué !");  
        if (e.getSource() == btnOK) {  
            String message = "Quel est votre prénom ?";  

```

On dit que la classe est son propre écouteur

@override : on ne définit que la méthode utile de l'interface
MouseListener

Solution 3: classe anonyme

```
btnOK.addMouseListener(new MouseAdapter() {  
    public void mouseClicked(MouseEvent e) {  
        System.out.println("Un bouton a été cliqué !");  
        if (e.getSource() == btnOK) {  
            String message = "Quel est votre prénom ?";  
  
            String reponse = JOptionPane.showInputDialog(null, message,  
                "Le titre", JOptionPane.PLAIN_MESSAGE);  
  
            if (reponse != null && reponse.length() != 0) {  
                label.setText("Bonjour " + reponse);  
            }  
        } else if (e.getSource() == btnQUIT) {  
            System.exit(0);  
        }  
    }  
});
```

Autres exemples: WindowAdapter

Classe écouteur en interne

```
public class TestEvents13 extends JFrame {
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents13(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        pane=(JPanel) getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);

        addWindowListener(new MyWindowListener());
    }
    class MyWindowListener extends WindowAdapter{
        public void windowClosing(WindowEvent e){
            System.exit(0);
        }
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents13();
        frame.setTitle("Première fenêtre");
        //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }
}
```

Classe principale est son propre écouteur

```
public class TestEvents14 extends WindowAdapter{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    JFrame frame;
    TestEvents14(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        frame=new JFrame();
        frame.setTitle("Première fenêtre");
        frame.setSize(300,100);
        frame.setVisible(true);
        pane=(JPanel) frame.getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);

        frame.addWindowListener(this);
    }
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
    public static void main(String args[]){
        TestEvents14 frame=new TestEvents14();
    }
}
```

Autres exemples: WindowAdapter

Classe anonyme, 2 implémentations

```
public class TestEvents12 extends JFrame {
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents12(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        pane=(JPanel) getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);

        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents12();
        frame.setTitle("Première fenêtre");
        //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }
}
```

```
public class TestEvents11{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    JFrame frame;
    TestEvents11(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        label=new JLabel();
        frame=new JFrame();
        frame.setTitle("Première fenêtre");
        frame.setSize(300,100);
        frame.setVisible(true);
        pane=(JPanel) frame.getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);

        frame.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }
    public static void main(String args[]){
        TestEvents11 frame=new TestEvents11();
    }
}
```

Quelques méthodes de composants (pour le TP)

Quelques exemples de méthodes utiles pour le TP

Méthode	Rôle
<code>myText.getText()</code>	Récupérer le contenu d'un champs text
<code>myButton.getText()</code>	Récupérer le nom d'un bouton
<code>myRadioButton.getText()</code>	Récupérer le nom d'un bouton radio
<code>myCheckBox.getText()</code>	Récupérer le nom d'un checkBox
<code>myComboBox.getSelectedItem()</code>	Récupérer l'élément sélectionné dans une combobox
<code>myRadioButton.isSelected()</code>	Vérifier si un bouton radio est sélectionné
<code>myRadioButton.setSelected(true)</code>	Sélectionner un bouton radio
<code>myCheckBox.isSelected()</code>	Vérifier si un checkBox est sélectionné
<code>myCheckBox.setSelected(true)</code>	Sélectionner un checkBox
<code>myList.getSelectedIndex()>=1</code>	Vérifier s'il y a au moins un élément sélectionné dans une JList