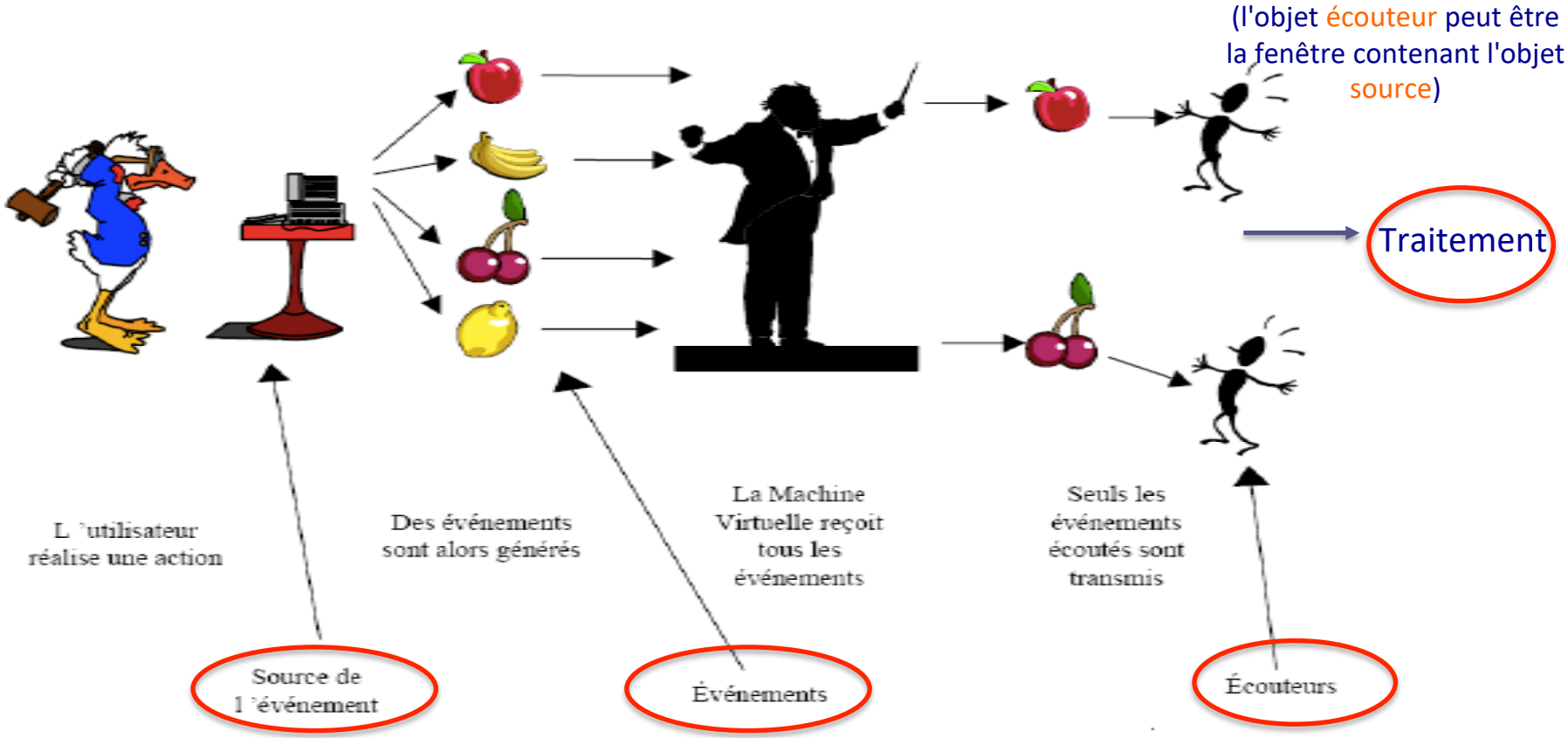
Java Avancé - Cours 2
Gestion des évènements

V. DESLANDRES, I. GUIDARA

veronique.deslandres@univ-lyon1.fr

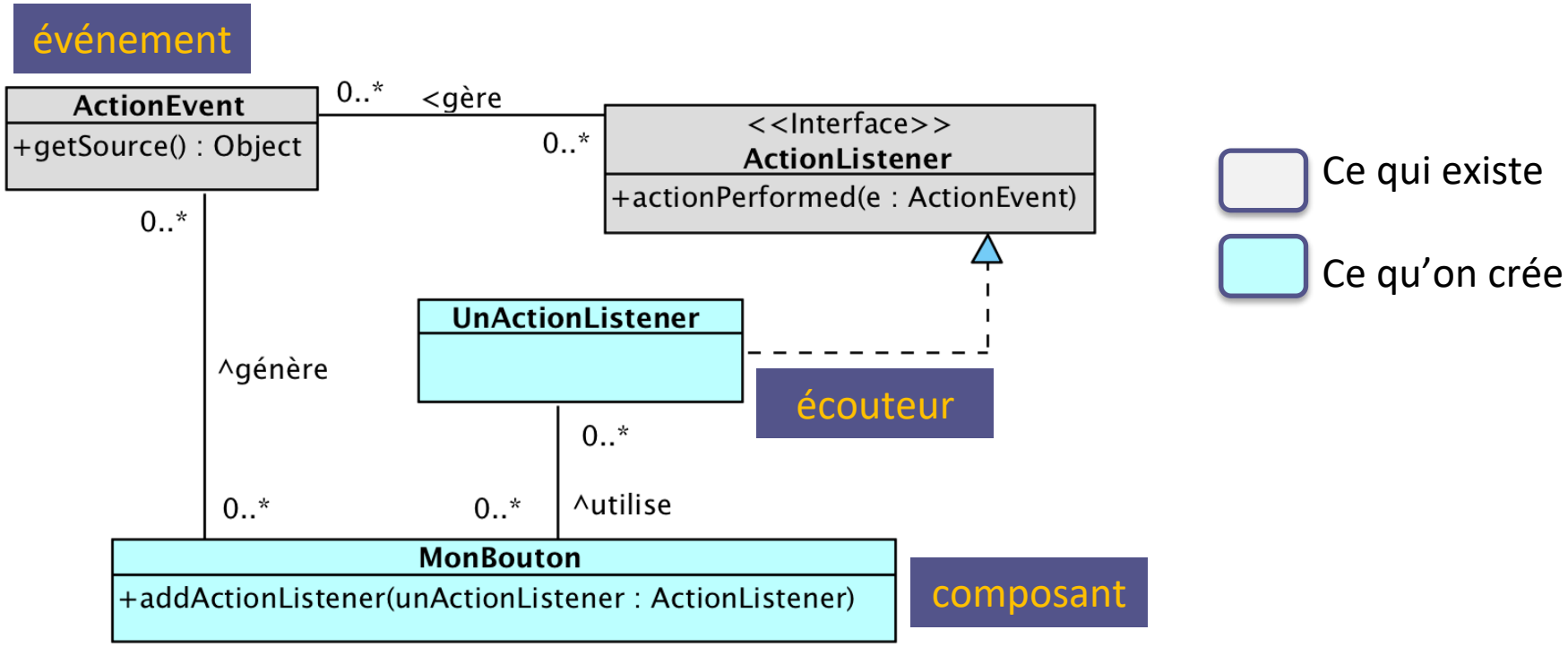
2021

Propagation des évènements



Principe (1/2)

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

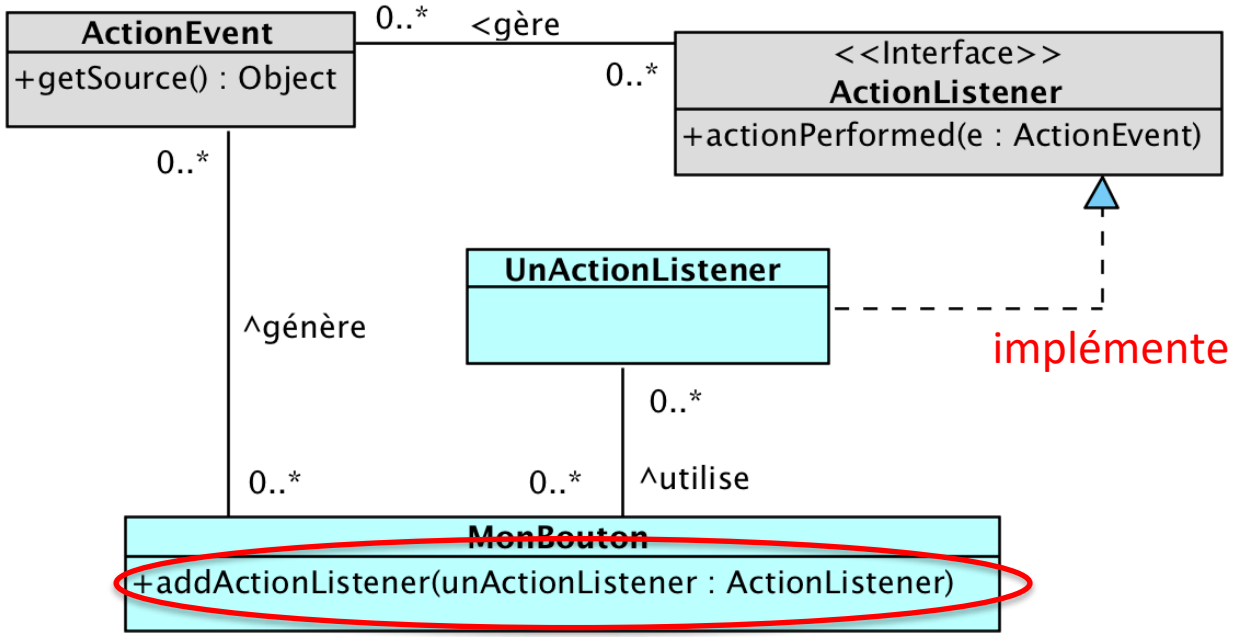


Principe (2/2)

- Les écouteurs sont des objets qui **implémentent des interfaces prédéfinies** (`MouseListener`, `ActionListener`)

- Ces écouteurs doivent être **explicitement** affectés aux composants concernés

```
myButton.addActionListener(new MyListener());
```



Package événements AWT

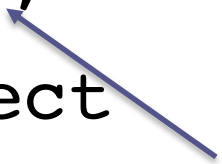
```
java.lang.Object
|
+--java.util.EventObject
        |
        +--java.awt.AWTEvent
                |
                +--java.awt.event.ActionEvent
                +--java.awt.event.AdjustmentEvent
                +--java.awt.event.AncestorEvent
                +--java.awt.event.ComponentEvent
                        |
                        +--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
```

```
import java.AWTEvent.*;
```

événements de bas niveaux

Evénements de la Swing

- Le paquet **`javax.swing.event`** définit quelques événements supplémentaires
  - (`CaretEvent, MenuEvent, ...`)
- à partir de `EventObject`



événement lié au curseur clignotant

- Mais Swing réutilise principalement les événements de 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.

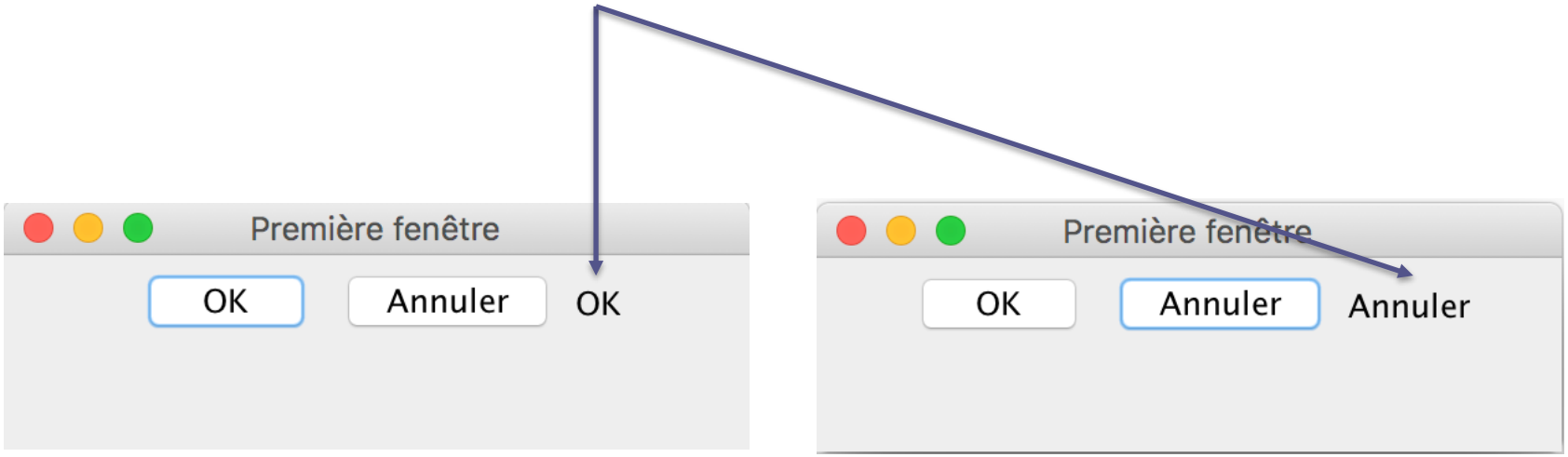
- Exemple :
  - Augmenter l'épaisseur d'un cadre quand on clique sur la touche + par ex. nécessite :
    - Composant source = `JPanel` avec un bord visible
    - Évènement : `KeyEvent` (touche +) *char c=e.getKeyChar(); if (c=='+')…*
    - Écouteur : `KeyListener` qui va augmenter l'épaisseur du bord

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



1ère implémentation d'un écouteur : classe écouteur en classe interne

Ecouteur défini en classe interne

- Créer 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 okButtonListner());
        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);
    }

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

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

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*

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

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

2ème 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 1ère 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");
    }
}
```

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* ? subjectif !).

3ème 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, juste avec* `new NomClasse () {… code de la classe }`
  - 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`

Ecouteur placé au niveau de la classe

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
- 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	Ecoute les actions de l'utilisateur : clics, barre d'espacement, entrée … • `JButton, JRadioButton et JCheckBox` : clic gauche et barre d'espacement • `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	`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 `KeyEvents` : 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
LsitSelectionListener	Ecouteur qui notifie quand la sélection d'éléments dans une liste change (L'utilisateur sélectionne (désélectionne) un item ou plusieurs items dans une `JList`, une `JTable`,…)	`JList`,`JTable`,…
MouseListener	Action sur la souris (Clic sur le bouton de la souris: appuyer, relâcher, déplacer le pointeur)	`Component`
MouseMotionListener	Action de la souris sur un composant (Evénements de glisser-déplacer)	`Component`
WindowListener	Action sur la fenêtre (Fenêtre activée, désactivée, réduite, fermée, …)	`Window`
TextListener	Changement d'une zone de texte	`JTextField`

Interfaces écouteurs - Traitement et enregistrement

Interface écouteur	Méthodes de traitement	Méthode d'enregistrement
ActionListener	`actionPerformed(ActionEvent e)`	`addActionListener()`
AdjustmentListener	`adjustmentValueChanged(adjustmentEvent e)`	`addAdjustmentListener()`

ComponentListener	`componentHidden(ComponentEvent e)`
	`componentMoved(ComponentEvent e)`
	`componentResized(ComponentEvent e)`
	`componentShown(ComponentEvent e)`
		`addComponentListener()`

ContainerListener	`componentAdded(ContainerEvent e)` `componentRemoved(ContainerEvent e)`	`addContainerListener()`
FocusListener	`focusGained(FocusEvent e)` `focusLost(FocusEvent e)`	`addFocusListener()`
ItemListener	`itemStateChanged(ItemEvent e)`	`addItemListener()`

KeyListener	`keyPressed(KeyEvent e)`
	`keyReleased(KeyEvent e)`
	`keyTyped(KeyEvent e)`
		`addKeyListener()`

DocumentListener	`changedUpdate(DocumentEvent e)`
	`insertUpdate(DocumentEvent e)`
	`removeUpdate(DocumentEvent e)`
		`addDocumentListener()`

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

Interfaces écouteurs - Traitement et enregistrement (Suite)

Interface écouteur	Méthodes de traitement	Méthode d'enregistrement
LsitSelectionListener	`valueChanged (ListSelectionEvent e)`	`addListSelectionListener()`

MouseListener	`mouseClicked(MouseEvent e)`
	`mouseEntered(MouseEvent e)`
	`mouseExited(MouseEvent e)`
	`mousePressed(MouseEvent e)`
	`mouseReleased(MouseEvent e)`
		`addMouseListener()`

MouseMotionListener	`mouseDragged(MouseEvent e)` `mouseMoved(MouseEvent e)`	`addMouseMotionListener()`

WindowListener	`windowActivated(WindowEvent e)`
	`windowClosed(WindowEvent e)`
	`windowClosing(WindowEvent e)`
	`windowDeactivated(WindowEvent e)`
	`windowDeiconified(WindowEvent e)`
	`windowIconified(WindowEvent e)`
	`windowOpened(WindowEvent e)`
		`addWindowListener()`

TextListener	`textValueChanged(TextEvent e)`	`addTextListener()`

Exemple: MouseListener

```
public class TestEvent7 extends JFrame implements ActionListener, MouseListener{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvent7(){
        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);
        ButOK.addMouseListener(this);
    }
    public static void main(String args[]){
        JFrame frame=new TestEvent7();
        frame.setTitle("Première fenêtre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }

    public void mouseClicked(MouseEvent e){
        System.out.println("Mouse Clicked");
    }

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

Imaginons qu'on veuille **voir dans la fenêtre de sortie** du pgm (pas sur l'IHM), un message quand on clique sur le bouton OK avec la souris : `System.out.println("Souris clic bouton OK");`

Si on clique sur le bouton Ok, le message sera affiché dans la console d'exécution

Limites des interfaces écouteurs

- Rappel: une classe qui implémente une interface doit définir **toutes** les méthodes de l'interface
- Or certaines interfaces `listener` ont beaucoup de méthodes
  - ce qui oblige parfois à définir des méthodes dont on n'a aucune utilité
- Exemple:
  - définir toutes les méthodes de l'interface `MouseListener`

    (elles n'étaient pas toutes implémentées dans le code précédent)

Exemple: MouseListener (suite)

```
public void mouseClicked(MouseEvent e){
    System.out.println("Mouse Clicked");
}
public void mouseEntered(MouseEvent e){
    System.out.println("Mouse Entered");
}
public void mouseExited(MouseEvent e){
    System.out.println("Mouse Exited");
}
public void mousePressed(MouseEvent e){
    System.out.println("Mouse Pressed");
}
public void mouseReleased(MouseEvent e){
    System.out.println("Mouse Released");
}
```

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 redéfinire 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

- L'interface `MouseListener` possède 5 méthodes (cf diapo 30) qui doivent toutes être implémentées si l'on veut gérer les événements liés à 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`)

```
interface MouseListener {
 // Déclare cinq méthodes
}
class MouseAdapter implements MouseListener {
 // implémentation vide des 5 méthodes
}
public class MaClasseEcouteur extends MouseAdapter {
 // surcharger les méthodes utiles (<5)
}
```

Illustration Adapter 1: classe écouteur en classe interne

```
public class TestEvents9 extends JFrame implements ActionListener{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents9(){
        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);
        ButOK.addMouseListener(new MyMouseListener());
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents9();
        frame.setTitle("Première fenêtre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);
    }
    class MyMouseListener extends MouseAdapter{
    public void mouseClicked(MouseEvent e){
        System.out.println("Mouse Clicked");
        }}
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==ButOK){
            label.setText("OK");
        }
        else
            label.setText("Annuler");
    }
}
```

Remarque: Ici on a utilisé **2 méthodes différentes** pour implémenter les écouteurs :

- Classe principale est son propre écouteur pour `ActionListener`
- Classe écouteur en classe interne pour `MouseListener`

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

Illustration Adapter 2: la classe principale est son propre écouteur

```
public class TestEvents10 extends MouseAdapter implements ActionListener{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    JFrame frame;
    TestEvents10(){
        ButOK=new JButton("OK");
        ButAnnuler=new JButton("Annuler");
        frame=new JFrame();
        label=new JLabel();
        pane=(JPanel) frame.getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(ButOK);
        pane.add(ButAnnuler);
        pane.add(label);

        ButOK.addActionListener(this);
        ButAnnuler.addActionListener(this);
        ButOK.addMouseListener(this) ;

        frame.setTitle("Première fenêtre");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,100);
        frame.setVisible(true);

    }
    public static void main(String args[]){
        TestEvents10 frame=new TestEvents10();
    }
    public void mouseClicked(MouseEvent e){
        System.out.println("Mouse Clicked");
    }
    public void actionPerformed(ActionEvent e){
        if(e.getSource()==ButOK){
            label.setText("OK");
        }
        else
            label.setText("Annuler");
    }
}
```

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

On déclare la fenêtre dans le constructeur

La fenêtre s'ouvre dans le constructeur

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

Solution 3: classe anonyme

```
public class TestEvents8 extends JFrame implements ActionListener{
    JButton ButOK, ButAnnuler;
    JPanel pane;
    JLabel label;
    TestEvents8(){
        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);
        ButOK.addMouseListener(new MouseAdapter(){
        public void mouseClicked(MouseEvent e){
        System.out.println("Mouse Clicked");
        }}) ;
    }
    public static void main(String args[]){
        JFrame frame=new TestEvents8();
        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");
    }
}
```

Ici aussi on a utilisé **2 méthodes différentes** pour implémenter les écouteurs :

- La classe principale est son propre écouteur pour `ActionListener`
- Une classe anonyme pour `MouseListener`

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