IUT d’Orsay - DUT Informatique - Jean-Claude MARTIN – JEAN-CLAUDE.MARTIN@U-PSUD.FR

# Programmation Orientée Objet (POO) COURS 9 : Interfaces graphiques : événements et écouteurs

## Objectifs du cours

* Comprendre les mécanismes de base de la gestion d’événements en Java.
* Savoir les appliquer dans plusieurs configurations possibles

## Les événements

Plusieurs objets participent au modèle de gestion des événements

* La *source* de l’événement (par exemple un bouton) :   
  ………………………………………………………………………………………………………………………...
* L’objet *événement* (par exemple un événement de type action clic sur un bouton) :

Les événements sont représentés par des classes XXXEvent qui héritent de la classe EventObject :



Evenements de bas niveau vs. événements sémantiques :

* Evénements sémantiques : l’utilisateur clique sur un bouton, l’utilisateur tape entrée dans un champ de texte, …
* Evenements de bas niveau : surtout des événements qui résultent directement de l’utilisateur : souris, clavier …

Il y a d’autres événements que ceux générés par des composants graphiques (par exemple connexion).

Méthode de la classe EventObject

public Object getSource()

Returns: The object on which the Event initially occurred.

Les écouteurs

Java utilise un modèle de délégation d’événements : le programmeur doit spécifier les objets qui doivent être prévenus quand un évenement apparaît.

Un 3ème objet participe donc au modèle de gestion des événements :

* ………………………………………………………………………………………………………………………

|  |  |
| --- | --- |
| **Listener** / **écouteur** : un objet qui écoute le comportement (d’un composant graphique).    Les objets écouteurs n’ont pas forcément de représentation graphique. Ils ne sont pas forcément visibles.  N’importe quel objet peut être un écouteur (un compte bancaire, une voiture, un bouton, un champ de texte, …). |  |

Comment créer un objet listener pouvant écouter des événements de type ActionEvent ?

Pour ajouter un écouteur sur un bouton, il faut appeler la méthode

public void addActionListener(ActionListener l)

* ………………………………………………………………………………………………………………………

définie dans la classe AbstractButton.

**EXTRAIT DE LA JAVADOC de l’interface ActionListener**

java.awt.event

Interface ActionListener

**All Superinterfaces:**[EventListener](http://java.util)

public interface **ActionListener** extends [EventListener](http://java.util)

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | [**actionPerformed**](http://docs.oracle.com/javase/7/docs/api/java/awt/event/ActionListener.html#actionPerformed(java.awt.event.ActionEvent))([**ActionEvent**](http://java.awt.event) e)  Invoked when an action occurs. |

Implémenter l’interface ActionListener => définir ………………………………………………………………………………………………………………………

Il y a plusieurs manière d’implémenter un écouteur :

* Classe écouteur indépendante de la classe graphique
* Classe graphique qui est aussi écouteur
* Classe anonyme
* Classe imbriquée

## 

## Classe écouteur indépendante

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Date;

**import** javax.swing.JButton;

**public** **class** EcouteurAfficheActionEvent **…………………………………………………………………………………..** {

**public** **void** actionPerformed(ActionEvent e) {

// Afficher la date et l'heure de l'événement

System.*out*.println ("-------------------------------");

System.*out*.println (**new** Date(e.getWhen()));

// Afficher l'identificateur de l'evenement

**if** (e.getID() == ActionEvent.*ACTION\_PERFORMED*)

System.*out*.println ("e.getID() == Event Id: ACTION\_PERFORMED");

// Afficher la classe de la source de l’événement

String source = e.getSource().getClass().getName();

System.*out*.println("Classe source de l'événement : " + source);

// Si c'est un bouton, afficher son nom

**if (e.getSource() instanceof JButton) {**

**}**

// Afficher les modifieurs (touches du claviers appuyées en même temps que le clic)

**int** mod = e.getModifiers();

String m = "";

**if** ((mod & ActionEvent.*ALT\_MASK*) > 0)

m += "Alt ";

**if** ((mod & ActionEvent.*SHIFT\_MASK*) > 0)

m += "Shift ";

**if** ((mod & ActionEvent.*CTRL\_MASK*) > 0)

m += "Ctrl ";

**if** ((mod & ActionEvent.*META\_MASK*) > 0)

m += "Meta ";

System.*out*.println("Modifieurs (touches du claviers appuyées) : " + m);

// Afficher la commande d'action du bouton (par défaut c'est le nom du bouton)

System.*out*.println ("e.getActionCommand() : " + e.getActionCommand());

}

}

Comment enregistrer un écouteur auprès d’un bouton ?

**import javax.swing.\*;**

**public class FrameButton {**

**public static void main (String args []) {**

**// Creer un bouton**

**JButton b = new JButton ("Bonjour");**

**// Creer un objet ecouteur**

**EcouteurAfficheActionEvent ecouteur = new EcouteurAfficheActionEvent();**

**// Dire au bouton d'ajouter a sa liste de listener le listener**

**// que l'on vient de creer**

**…………………………………………………………………………………………………………………………………….**

**// Creer une fenetre et y ajouter le bouton**

**JFrame f = new JFrame ("Fenetre");**

**f.add (b);**

**f.setSize (300, 200);**

**f.setVisible (true);**

**}**

**}**

Exécution

**-------------------------------**

**Sun Nov 04 17:00:40 CET 2012**

**e.getID() == Event Id: ACTION\_PERFORMED**

**Classe source de l'événement : javax.swing.JButton**

**Modifieurs (touches du claviers appuyées) : Alt Ctrl**

Schéma montrant la gestion des événements

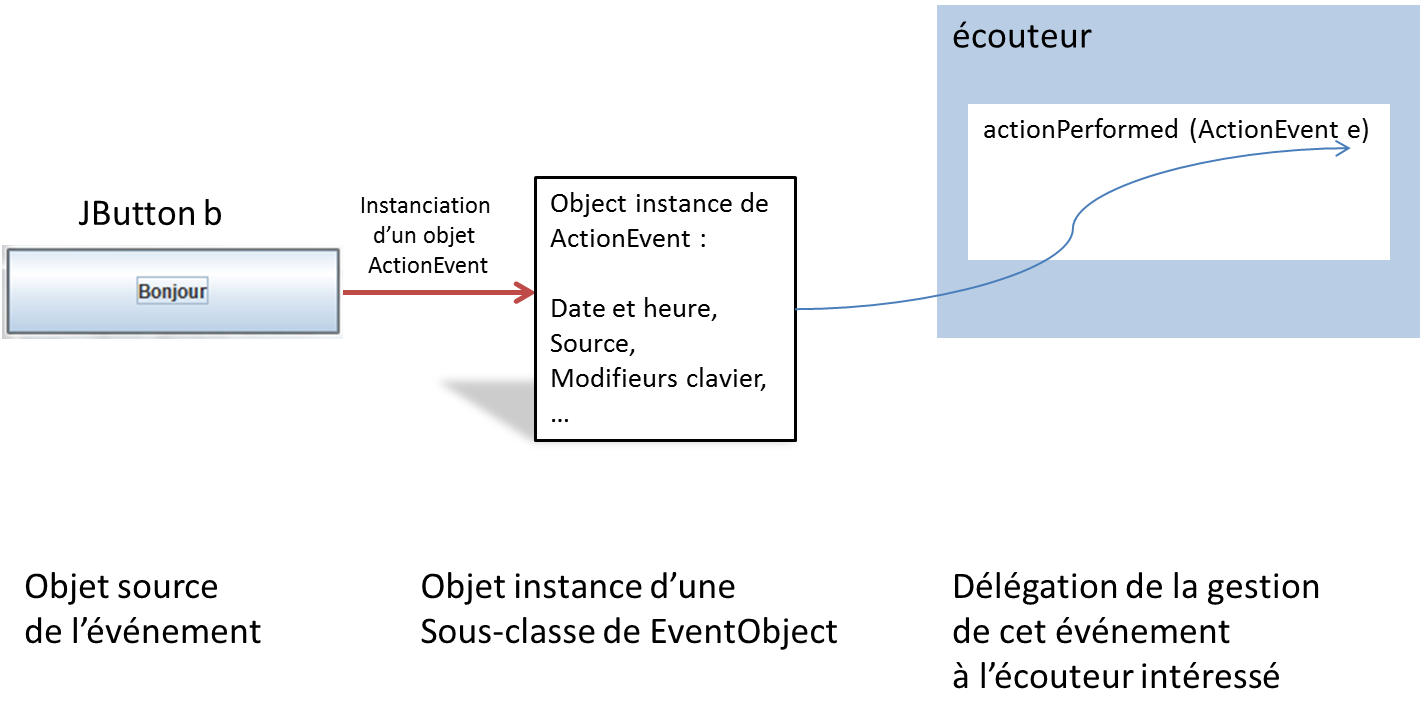
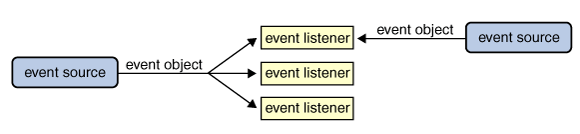


Schéma de la mémoire

Lorsque l’on clique sur le bouton :

1. un objet de type ………………………………………………………………………………… est créé
2. la méthode ………………………………………...   
     
   est appelée sur ………………………………………………………...   
     
   avec en paramètre :……………………………………………………………………………………………………………………….

Relations multiples entre composants et écouteurs



Un composant peut avoir plusieurs écouteurs (de la même classe d’écouteurs ou non)

Instructions Java :

// Créer un 2ème écouteur

EcouteurAfficheActionEvent ecouteur2 = **new** EcouteurAfficheActionEvent();

b.addActionListener (ecouteur2);

Affichage :

* Les informations sur l’événement sont affichées 2 fois

**-------------------------------**

**Fri May 02 10:01:48 CEST 2014**

**e.getID() == Event Id: ACTION\_PERFORMED**

**Classe source de l'événement : javax.swing.JButton**

**Nom du bouton : Bonjour**

**Modifieurs (touches du claviers appuyées) :**

**e.getActionCommand() : Bonjour**

**-------------------------------**

**Fri May 02 10:01:48 CEST 2014**

**e.getID() == Event Id: ACTION\_PERFORMED**

**Classe source de l'événement : javax.swing.JButton**

**Nom du bouton : Bonjour**

**Modifieurs (touches du claviers appuyées) :**

**e.getActionCommand() : Bonjour**

Mémoire :

Cf schéma

Un même écouteur peut écouter plusieurs composants

Jbutton bonjour = new Jbutton (« bonjour ») ;

Jbutton aurevoir = new Jbutton (« au revoir ») ;

Ecouteur e = new Ecouteur() ;

bonjour.addActionListener(e) ;

aurevoir.addActionListener(e) ;

dans la méthode Ecouteur .actionPerformed () :

pour récupérer le bouton qui a été cliqué : e.getSource()

Comment passer des informations à un écouteur ?

Exemple : Afficher le contenu d’un champ de texte lorsque l’on clique sur un bouton

Lorsque l’on clique sur le bouton, récupérer ce qui a été tapé dans le champ de texte et l’afficher

=> Le listener doit avoir acces au textfield

Est-ce possible d’ajouter un paramètre à la méthode actionPerformed ? ……NON…………………………………

Comment passer des informations à l’écouteur ? …………………………………………………………………………….

**Solution**:

1. ajouter un constructeur au listener qui prend en parametre la reference vers le textfield.
2. Dans le constructeur, mémoriser la reference vers le textfield dans un attribut JTextField de la classe Listener (ajouter la déclaration de cet attribut)
3. dans la méthode actionPerformed, vous aurez acces a cet attribut
4. Lors de la création de l’écouteur, passer le champ de texte en paramètre au constructeur

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.\*;

**public** **class** EcouteurAfficherTextField **implements** ActionListener {

JTextField t ;

EcouteurAfficherTextField (JTextField t) {

**this**.t = t;

}

**public** **void** actionPerformed(ActionEvent e) {

System.*out*.println ("Texte saisi : " + t.getText());

}

}

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.\*;

**public** **class** PanelEcouteurClasseIndependante **extends** JPanel {

JButton b ;

JTextField saisie ;

PanelEcouteurClasseIndependante () {

// Creer les composants et les ajouter au panel

b = **new** JButton ("Afficher");

saisie = **new** JTextField(20);

add (b);

add (saisie);

// Dire au bouton d'ajouter a sa liste de listener un écouteur indépendant

// Passer en paramètre au constructeur de l'écouteur la référence sur le champ de texte

**……………………………………………………………………………………………………………………………………………………………………………………………………………..**

**……………………………………………………………………………………………………………………………………………………………………………………………………………..**

**public** **static** **void** main (String args []) {

PanelEcouteurClasseIndependante p = **new** PanelEcouteurClasseIndependante();

JFrame f = **new** JFrame ("Fenetre");

f.add (p);

f.setSize (500, 100);

f.setVisible (**true**);

}

}

Plusieurs classes écouteurs => plus facile à maintenir et ré-utiliser mais peut réduire les performances

Classe graphique qui est aussi écouteur

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.\*;

**public class PanelEcouteur extends JPanel …………………………………………………………… {**

**JButton b ;**

**JTextField txt ;**

**PanelEcouteur () {**

**// Creer les composants et les ajouter au panel**

**b = new JButton ("Afficher");**

**txt = new JTextField(20);**

**add (b);**

**add (txt);**

**// Dire au bouton d'ajouter a sa liste de listener CET OBJET**

**……………………………………………………………**

**}**

// La classe graphique est aussi écouteur

// donc la méthode actionPerformed a accès à tous les attributs

**……………………………………………………………**

**public** **static** **void** main (String args []) {

// Creer un bouton

PanelEcouteur p = **new** PanelEcouteur();

// Creer une fenetre et y ajouter le bouton

JFrame f = **new** JFrame ("Fenetre");

f.add (p);

f.setSize (300, 200);

f.setVisible (**true**);

}

}

Avantages :

* La méthode actionPerformed a accès aux attributs de la classe graphique (dont les composants) donc la gestion des événements est plus simple à programmer

Inconvénients

* Pas modulaire : graphique et événements mélangés ; gestion des événements non réutilisable pour une autre classe graphique

Classe écouteur anonyme

Anonyme = on ne donne pas de nom à la classe (on ne pourra pas la ré-utiliser)

Imbriquée = à l’intérieur de la même classe

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.\*;

**public** **class** FrameButtonEcouteurImbriqueAnonyme {

**public** **static** **void** main (String args []) {

// Creer un bouton

**final** JButton b = **new** JButton ("Bonjour");

// Enregistrer un écouteur « anonyme » auprès du bouton

**……………………………………………………………**

// Creer une fenetre et y ajouter le bouton

JFrame f = **new** JFrame ("Fenetre");

f.add (b);

f.setSize (300, 200);

f.setVisible (**true**);

}

}

Classe imbriquée

* L’écouteur est défini par une classe qui est définie dans la même classe et qui a un nom.
* Les classes imbriquées sont utilisées en Java (pas seulement pour les interfaces graphiques) quand on veut 1) diminuer le nombre de classes « visibles », ou 2) protéger l’accès à la classe imbriquée
* La classe imbriquée permet aux méthodes de l’écouteur d’accéder aux attributs privés
* Contrairement à l’écouteur anonyme, on définit ici une classe donc on peut créer plusieurs instances si besoin !

Classes imbriquées :

visibilité pour la classe imbriquée : private, public, protected ou package

pour y accéder : ClasseEnglobante.ClasseImbriquee

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.\*;

**public** **class** PanelEcouteurImbrique **extends** JPanel {

JButton b ;

JTextField txt ;

PanelEcouteurImbrique () {

// Creer les composants et les ajouter au panel

b = **new** JButton ("Afficher");

txt = **new** JTextField(20);

add (b);

add (txt);

// Creer un objet ecouteur

EcouteurImbrique ecouteur = **new** EcouteurImbrique();

// Dire au bouton d'ajouter a sa liste de listener le listener

// que l'on vient de creer

b.addActionListener (ecouteur);

}

// La classe écouteur est interne à la classe principale

// donc les méthodes ont accès à tous les attributs

**……………………………………………………………**

**public** **static** **void** main (String args []) {

// Creer un bouton

PanelEcouteurImbrique p = **new** PanelEcouteurImbrique();

// Creer une fenetre et y ajouter le bouton

JFrame f = **new** JFrame ("Fenetre");

f.add (p);

f.setSize (300, 200);

f.setVisible (**true**);

}

}

Avantages et inconvénients des 4 méthodes

getActionCommand() et setActionCommand()

Autre possibilité pour le bouton de passer des informations à la méthode getActionCommand()   
lors d’un événement de type Action :

* getActionCommand() : retourne la « chaîne de commande de l’action » (par défaut le nom du bouton).
* monBouton1.setActionCommand (SUPPRIMER);

Utile par exemple si le nom affiché sur le bouton change durant l’exécution, ou est dans une autre langue …

Event Adapters

Certaines interfaces écouteurs imposent d’implémenter plusieurs méthodes, par exemple   
CINQ METHODES dans MouseListener !

public interface **MouseListener**

extends [EventListener](http://java.util)

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| void | [**mouseClicked**](http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html#mouseClicked(java.awt.event.MouseEvent))([**MouseEvent**](http://java.awt.event) e)  Invoked when the mouse button has been clicked (pressed and released) on a component. |
| void | [**mouseEntered**](http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html#mouseEntered(java.awt.event.MouseEvent))([**MouseEvent**](http://java.awt.event) e)  Invoked when the mouse enters a component. |
| void | [**mouseExited**](http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html#mouseExited(java.awt.event.MouseEvent))([**MouseEvent**](http://java.awt.event) e)  Invoked when the mouse exits a component. |
| void | [**mousePressed**](http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html#mousePressed(java.awt.event.MouseEvent))([**MouseEvent**](http://java.awt.event) e)  Invoked when a mouse button has been pressed on a component. |
| void | [**mouseReleased**](http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseListener.html#mouseReleased(java.awt.event.MouseEvent))([**MouseEvent**](http://java.awt.event) e)  Invoked when a mouse button has been released on a component. |

Si une seule méthode vous intéresse, il faut tout de même implémenter les autres méthodes (corps de méthode vide).

//An example that implements a listener interface directly.

public class MyClass implements MouseListener {

...

someObject.addMouseListener(this);

...

/\* Empty method definition. \*/

public void mousePressed(MouseEvent e) {

}

/\* Empty method definition. \*/

public void mouseReleased(MouseEvent e) {

}

/\* Empty method definition. \*/

public void mouseEntered(MouseEvent e) {

}

/\* Empty method definition. \*/

public void mouseExited(MouseEvent e) {

}

public void mouseClicked(MouseEvent e) {

...//Event listener implementation goes here...

}

}

Pour éviter d’avoir à implémenter ces méthodes vides, il existe des classes XXX Adapter qui incluent déjà ces méthodes vides. Il suffit d’étendre la classe Adapter et de redéfinir seulement la méthode qui vous intéresse.

public class MyClass extends MouseAdapter {

...

someObject.addMouseListener(this);

...

public void mouseClicked(MouseEvent e) {

...//Event listener implementation goes here...

}

ATTENTION : Pour détecter les mouvements de la souris il faut implémenter l’interface: MouseMotionListener ou étendre l’adapteur MouseMotionAdapter

Ecrire des écouteurs performants

Un écouteur peut prendre du temps (redessiner les composants) et peut ralentir ou bloquer le programme principal => utilisation de Thread pour exécuter l’écouteur en parallèle au programme principal.