Principe de la programmation événementielle en Java

I. Nature des événements

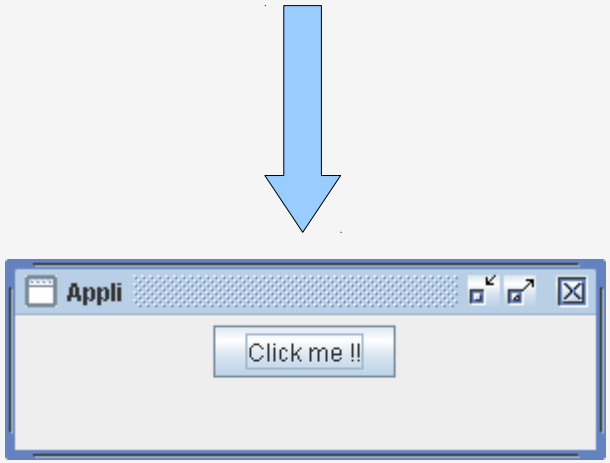
Que se passe-t-il lors d'un clic ?

JFrame

```
public Appli(){
    p = new MonJPanel();
    add(p);
}
```

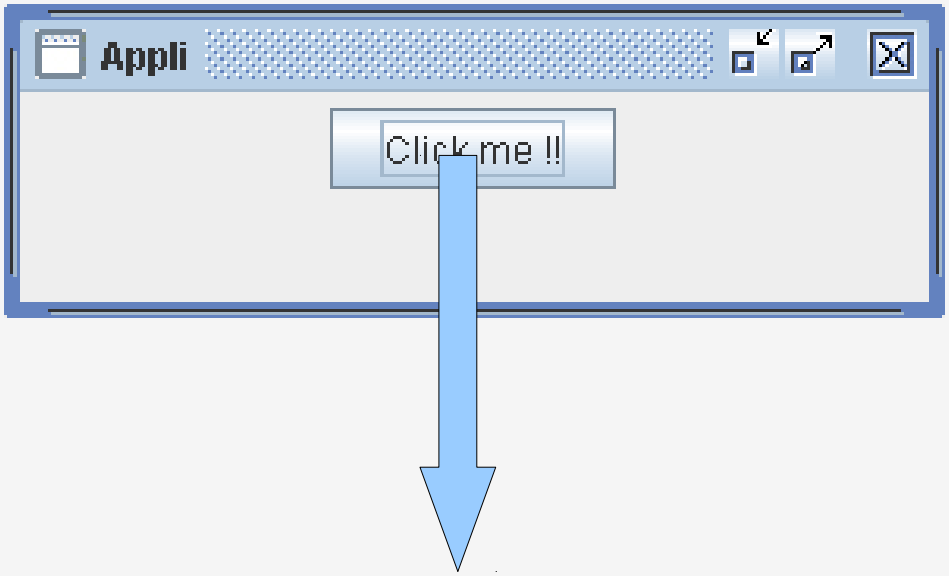
MonJPanel

```
public MonJPanel() {
    JButton b = new JButton("Click me !!");
    add(b);
}
```



Que se passe-t-il lors d'un clic ?

- Un clic génère ce qu'on appelle un événement



Événement

- Il s'agit en fait d'un **objet**

L'objet **java.util.EventObject**

java.util

Class EventObject

java.lang.Object
  └ **java.util.EventObject**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AWTEvent, BeanContextEvent, CaretEvent, ChangeEvent, ConnectionEvent, DragGestureEvent, DragSourceEvent, DropTargetEvent, FlavorEvent, HandshakeCompletedEvent, HyperlinkEvent, LineEvent, ListDataEvent, ListSelectionEvent, MenuEvent, NamingEvent, NamingExceptionEvent, NodeChangeEvent, Notification, PopupMenuEvent, PreferenceChangeEvent, PrintEvent, PropertyChangeEvent, RowSetEvent, SSLSessionBindingEvent, TableColumnModelEvent, TableModelEvent, TreeExpansionEvent, TreeModelEvent, TreeSelectionEvent, UndoableEditEvent, UnsolicitedNotificationEvent

---

```
public class EventObject
extends Object
implements Serializable
```

The root class from which all event state objects shall be derived.

All Events are constructed with a reference to the object, the "source", that is logically deemed to be the object upon which the Event in question initially occurred upon.

L'objet événement de Java

Field Summary

protected Object	**source** The object on which the Event initially occurred.

Constructor Summary

EventObject(Object source)
Constructs a prototypical Event.

Method Summary

Object	**getSource**() The object on which the Event initially occurred.
String	**toString**() Returns a String representation of this EventObject.

Qui crée ce type d'objet ?

- Il existe un thread spécifique, chargé de récupérer les interactions utilisateurs (souris, clavier, etc.)
- Lors d'une interaction, ce thread instancie un événement du type correspondant, avec le composant concerné comme **source**.
- Conséquence : **on ne crée pas** les événements, **on ne fait que les gérer**.

GUI → java.awt.AWTEvent

java.awt

Class AWTEvent

java.lang.Object
  └ java.util.EventObject
      └ **java.awt.AWTEvent**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

ActionEvent, AdjustmentEvent, AncestorEvent, ComponentEvent, HierarchyEvent, InputMethodEvent, InternalFrameEvent, InvocationEvent, ItemEvent, TextEvent

```
public abstract class AWTEvent
extends EventObject
```

The root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class. Subclasses of this root AWTEvent class defined outside of the java.awt.event package should define event ID values greater than the value defined by RESERVED_ID_MAX.

II. Le package **java.awt.event**

Provides interfaces and classes for dealing with different types of events fired by AWT components.

Events **bas niveau** vs. **sémantique**

- 2 types d'événements sont distingués :
  - **bas niveau** : e.g. clique de souris, clavier, etc.
  - **sémantiques** : e.g. « *le bouton X a été pressé* »

- En priorité, on traite des *events* sémantiques :
  - ils contiennent une information de plus haut niveau .
  - Ils facilitent la gestion : presser un bouton avec le clavier ou la souris revient au même.

java.awt.event.ComponentEvent

java.awt.event

Class ComponentEvent

java.lang.Object
  └ java.util.EventObject
    └ java.awt.AWTEvent
      └ **java.awt.event.ComponentEvent**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

ContainerEvent, FocusEvent, InputEvent, PaintEvent, WindowEvent

---

```
public class ComponentEvent
extends AWTEvent
```

A low-level event which indicates that a component moved, changed size, or changed visibility (also, the root class for the other component-level events).

Component events are provided for notification purposes ONLY; The AWT will automatically handle component moves and resizes internally so that GUI layout works properly regardless of whether a program is receiving these events or not.

java.awt.event.InputEvent

java.awt.event

Class InputEvent

java.lang.Object
  └ java.util.EventObject
    └ java.awt.AWTEvent
      └ java.awt.event.ComponentEvent
        └ **java.awt.event.InputEvent**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

KeyEvent, MouseEvent

```
public abstract class InputEvent
extends ComponentEvent
```

The root event class for all component-level input events.

java.awt.event.MouseEvent

java.awt.event

Class MouseEvent

java.lang.Object
  └ java.util.EventObject
      └ java.awt.AWTEvent
          └ java.awt.event.ComponentEvent
              └ java.awt.event.InputEvent
                  └ **java.awt.event.MouseEvent**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

MenuDragMouseEvent, MouseWheelEvent

```
public class MouseEvent
extends InputEvent
```

An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the unobscured part of the component's bounds when the action happens. Component bounds can be obscurred by the visible component's children or by a menu or by a top-level window. This event is used both for mouse events (click, enter, exit) and mouse motion events (moves and drags).

java.awt.event.KeyEvent

java.awt.event

Class KeyEvent

java.lang.Object
  └ java.util.EventObject
    └ java.awt.AWTEvent
      └ java.awt.event.ComponentEvent
        └ java.awt.event.InputEvent
          └ **java.awt.event.KeyEvent**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

MenuKeyEvent

```
public class KeyEvent
extends InputEvent
```

An event which indicates that a keystroke occurred in a component.

This low-level event is generated by a component object (such as a text field) when a key is pressed, released, or typed. The

java.awt.event.ActionEvent

java.awt.event

Class ActionEvent

java.lang.Object
  └ java.util.EventObject
    └ java.awt.AWTEvent
      └ **java.awt.event.ActionEvent**

All Implemented Interfaces:

Serializable

```
public class ActionEvent
extends AWTEvent
```

A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed).

III. Gestion des événements

Comment gérer des événements ?

- Un très grand nombre d'événements sont générés
- Il est hors de question de tous les gérer
- Le principe consiste à **écouter** uniquement les **événements désirés** sur les **composants ciblés**.
- → on crée des **écouteurs**, pour chaque *event* désiré, et on les associent aux composants cibles

Qu'est-ce qu'un écouteur ?

- Un écouteur est un objet destiné à recevoir et à gérer les événements générés
- Les écouteurs principaux se trouvent aussi dans le **package java.awt.event**
- La plupart sont définis par une interface java : n'importe quel objet peut devenir un écouteur.
- Exemple :

java.awt.event.ActionListener

java.awt.event.ActionListener

```
public interface ActionListener
extends EventListener
```

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.

Since:

1.1

See Also:

`ActionEvent`, How to Write an Action Listener

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
void	**`actionPerformed(ActionEvent e)`** Invoked when an action occurs.

Création d'un écouteur

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MonEcouteur implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        System.out.println("click !!");
    }
}
```

Utilisation de l'écouteur

```
public Appli(){
        p = new MonJPanel();
        add(p);
        MonEcouteur ecouteur = new MonEcouteur();
                }
```

- Au clique, rien ne se passe...

Principe de l'écoute

- Par défaut, un écouteur ne récupère pas tous les événements produits par le système, heureusement !
- Il n'écoute **que les objets qu'on lui dit d'écouter** !
- Pour fonctionner, **il doit être associé à un objet** qui génère des événements.
- Ainsi, **les objets** qui génèrent des événements **possèdent des méthodes qui permettent de leur associer les écouteurs adéquats**

java.awt.event.ActionListener

```
public interface ActionListener
extends EventListener
```

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.

Since:

1.1

See Also:

`ActionEvent`, How to Write an Action Listener

Method Summary

All Methods	Instance Methods	Abstract Methods

Modifier and Type	Method and Description
void	**`actionPerformed(ActionEvent e)`** Invoked when an action occurs.

javax.swing.JButton

Methods inherited from class javax.swing.AbstractButton

addActionListener, addChangeListener, addImpl, addItemListener, checkHorizontalKey, checkVerticalKey, createActionListener, createActionPropertyChangeListener, createChangeListener, createItemListener, doClick, doClick, fireActionPerformed, fireItemStateChanged, fireStateChanged, getAction, getActionCommand, getActionListeners, getChangeListeners, getDisabledIcon, getDisabledSelectedIcon, getDisplayedMnemonicIndex, getHorizontalAlignment, getHorizontalTextPosition, getIcon, getIconTextGap, getItemListeners, getLabel, getMargin, getMnemonic, getModel, getMultiClickThreshhold, getPressedIcon, getRolloverIcon, getRolloverSelectedIcon, getSelectedIcon, getSelectedObjects, getText, getUI, getVerticalAlignment, getVerticalTextPosition, imageUpdate, init, isBorderPainted, isContentAreaFilled, isFocusPainted, isRolloverEnabled, isSelected, paintBorder, removeActionListener, removeChangeListener, removeItemListener, setAction, setActionCommand, setBorderPainted, setContentAreaFilled, setDisabledIcon, setDisabledSelectedIcon, setDisplayedMnemonicIndex, setEnabled, setFocusPainted, setHorizontalAlignment, setHorizontalTextPosition, setIcon, setIconTextGap, setLabel, setLayout, setMargin, setMnemonic, setMnemonic, setModel, setMultiClickThreshhold, setPressedIcon, setRolloverEnabled, setRolloverIcon, setRolloverSelectedIcon, setSelected, setSelectedIcon, setText, setUI, setVerticalAlignment, setVerticalTextPosition

Rappel sur l'application: 3 classes

JFrame

```
public Appli(){
    p = new MonJPanel();
    add(p);
}
```

MonJPanel

```
public MonJPanel() {
    JButton b = new JButton("Click me !!");
    add(b);
}
```

MonEcouteur

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MonEcouteur implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        System.out.println("click !!");
    }
}
```

Utilisation d'un écouteur

```
public MonJPanel() {
      JButton b = new JButton("Click me !!");
      add(b);
      b.addActionListener(new MonEcouteur());
}
```


```
click !!
click !!
```

2 boutons : 1 écouteur par objet

```
public class MonEcouteurBis implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        System.out.println("right choice !!");
    }
}
```

```
public MonJPanel() {
    JButton b = new JButton("Click me !!");
    JButton b2 = new JButton("No Click me first !!");
    add(b); add(b2);
    b.addActionListener(new MonEcouteur());
    b2.addActionListener(new MonEcouteurBis());
}
```


```
right choice !!
click !!
right choice !!
```

Remarques sur les écouteurs

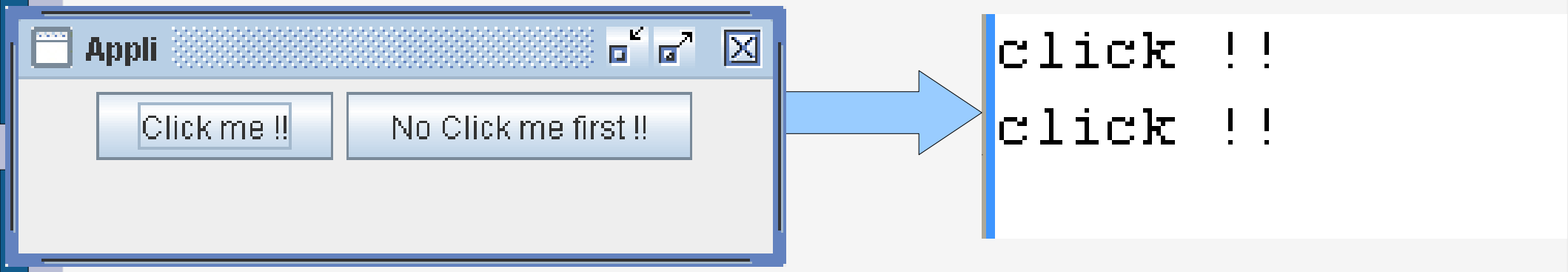
- Il est possible d'ajouter plusieurs écouteurs à un seul objet
- Il est possible d'écouter plusieurs objets avec un seul écouteur

1 écouteur pour 2 objets

```
public MonJPanel() {
    JButton b = new JButton("Click me !!");
    JButton b2 = new JButton("No Click me first !!");
    add(b); add(b2);
    MonEcouteur m = new MonEcouteur();
    b.addActionListener(m);
    b2.addActionListener(m);
}
```


```
click !!
click !!
```

Problème : la source ?



- Comment faire pour distinguer les deux avec un seul écouteur ?

Rappel : java.util.EventObject

Field Summary

protected Object	**source** The object on which the Event initially occurred.

Constructor Summary

EventObject(Object source) Constructs a prototypical Event.

Method Summary

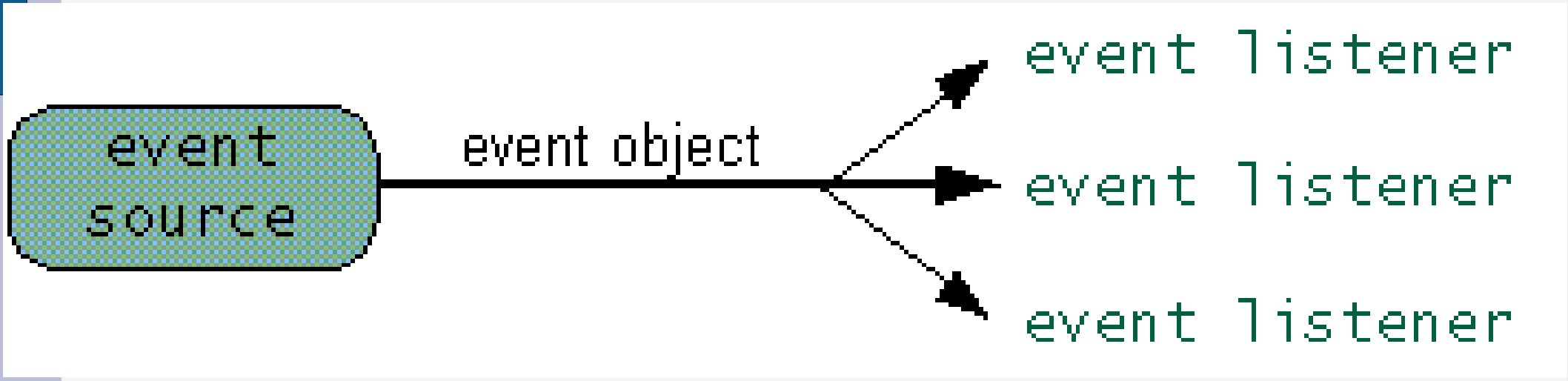
Object	**getSource**() The object on which the Event initially occurred.
String	**toString**() Returns a String representation of this EventObject.

On connaît la source !

```
public class MonEcouteur implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        JButton b = (JButton)e.getSource();

        if(b.getActionCommand().equals("Click me !!"))
            System.out.println("click !!");
        else
            System.out.println("right choice !!");
    }
}
```


```
right choice !!
click !!
right choice !!
```

Pour résumer



IV. Écoute des événements souris

java.awt.event.MouseListener

```
public interface MouseListener
extends EventListener
```

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component. (To track mouse moves and mouse drags, use the MouseMotionListener.)

The class that is interested in processing a mouse event either implements this interface (and all the methods it contains) or extends the abstract MouseAdapter class (overriding only the methods of interest).

The listener object created from that class is then registered with a component using the component's addMouseListener method. A mouse event is generated when the mouse is pressed, released clicked (pressed and released). A mouse event is also generated when the mouse cursor enters or leaves a component. When a mouse event occurs, the relevant method in the listener object is invoked, and the MouseEvent is passed to it.

java.awt.event.MouseListener

Method Summary	
void	**mouseClicked**(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	**mouseEntered**(MouseEvent e) Invoked when the mouse enters a component.
void	**mouseExited**(MouseEvent e) Invoked when the mouse exits a component.
void	**mousePressed**(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(MouseEvent e) Invoked when a mouse button has been released on a component.

java.awt.event.MouseEvent

```
public class HouseEvent
extends InputEvent
```

An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the unobscured part of the component's bounds when the action happens. Component bounds can be obscurred by the visible component's children or by a menu or by a top-level window. This event is used both for mouse events (click, enter, exit) and mouse motion events (moves and drags).

This low-level event is generated by a component object for:

- Mouse Events
  - a mouse button is pressed
  - a mouse button is released
  - a mouse button is clicked (pressed and released)
  - the mouse cursor enters the unobscured part of component's geometry
  - the mouse cursor exits the unobscured part of component's geometry
- Mouse Motion Events
  - the mouse is moved
  - the mouse is dragged

java.awt.event.MouseEvent

Method Summary

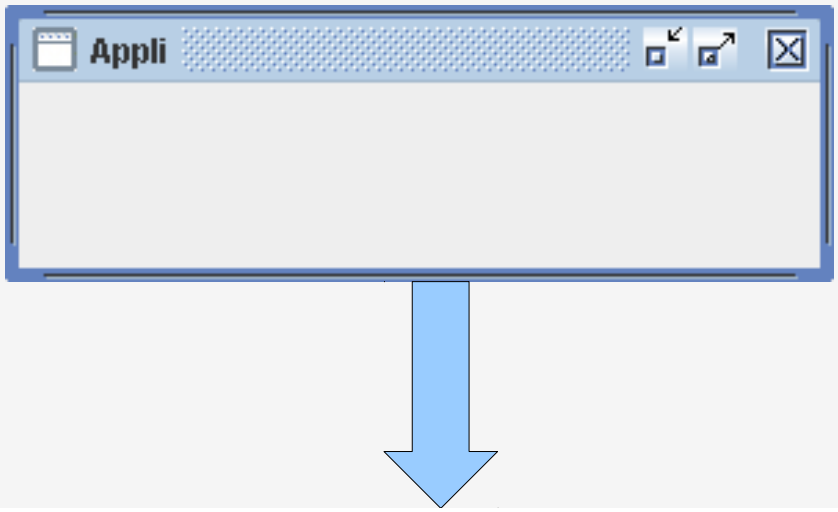
int	**getButton**() Returns which, if any, of the mouse buttons has changed state.
int	**getClickCount**() Returns the number of mouse clicks associated with this event.
static String	**getMouseModifiersText**(int modifiers) Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".
Point	**getPoint**() Returns the x,y position of the event relative to the source component.
int	**getX**() Returns the horizontal x position of the event relative to the source component.
int	**getY**() Returns the vertical y position of the event relative to the source component.
boolean	**isPopupTrigger**() Returns whether or not this mouse event is the popup menu trigger event for the platform.
String	**paramString**() Returns a parameter string identifying this event.
void	**translatePoint**(int x, int y) Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.

java.awt.event.MouseListener

```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class MonEcouteurSouris implements MouseListener{

    public void mouseClicked(MouseEvent e) {
        System.out.println("position du clic : x="+e.getX()+";y="+e.getY());
    }

    public void mousePressed(MouseEvent e) {
    }

    public void mouseReleased(MouseEvent e) {
    }

    public void mouseEntered(MouseEvent e) {
    }

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

java.awt.event.MouseListener

```
public Appli(){
    p = new JPanel();
    add(p);
    MonEcouteurSouris    ecouteur = new MonEcouteurSouris();
    p.addMouseListener(ecouteur);
        }
```




```
position du clic : x=192;y=31
position du clic : x=66;y=46
position du clic : x=52;y=11
```

Autre exemple

```
public void mouseClicked(MouseEvent e) {
    System.out.println("position du clic : x="+e.getX()+";y="+e.getY());
}

public void mousePressed(MouseEvent e) {
}

public void mouseReleased(MouseEvent e) {
}

public void mouseEntered(MouseEvent e) {
    System.out.println("Entrée dans la zone de clic");
}

public void mouseExited(MouseEvent e) {
    System.out.println("Sortie dans la zone de clic");
}
```

Remarque : un objet peut être son propre écouteur !

```
public Appli(){
    this.addMouseListener(this);
        }

public void mouseClicked(MouseEvent e) {
    System.out.println("position du clic : x="+e.getX()+";y="+e.getY());
}

public void mousePressed(MouseEvent e) {
}

public void mouseReleased(MouseEvent e) {
}

public void mouseEntered(MouseEvent e) {
    System.out.println("Entrée dans la zone de clic");
}

public void mouseExited(MouseEvent e) {
    System.out.println("Sortie dans la zone de clic");
}
```