Gestion des événements

- Historique des modèles d'événements
- Gestion des événements
- Classification des événements
- Conseils méthodologiques
- Événements de focus et d'entrée de données
- Événements légers
- spécification JavaBean
- Beans : utilisation des *PCL* et *VCL*

Historique (1/2)

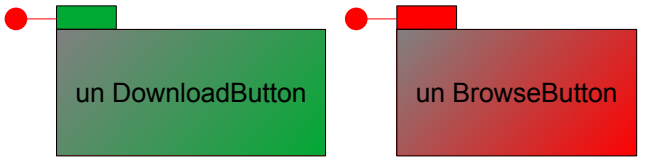
- Modèle 1.0 (*Inheritence Event Model*)
  - une seule classe d'événements : `Event` (obsolète)
  - tout se fait dans `Component` (et sous-classes)

```
public abstract class Component ... {
    ...
    @Deprecated
    public boolean handleEvent(Event evt) {
        switch (evt.id) {
          // 6 cas MOUSE
          case Event.MOUSE_ENTER:
              return mouseEnter(evt, evt.x, evt.y);
          ...
          // 4 cas KEY
          case Event.KEY_PRESS:
              return keyDown(evt, evt.key);
          ...
          // 2 cas FOCUS
          case Event.GOT_FOCUS:
              return gotFocus(evt, evt.arg);
          ...
          // 1 cas ACTION
          case Event.ACTION_EVENT:
              return action(evt, evt.arg);
        }
        return false;
    }
}
```

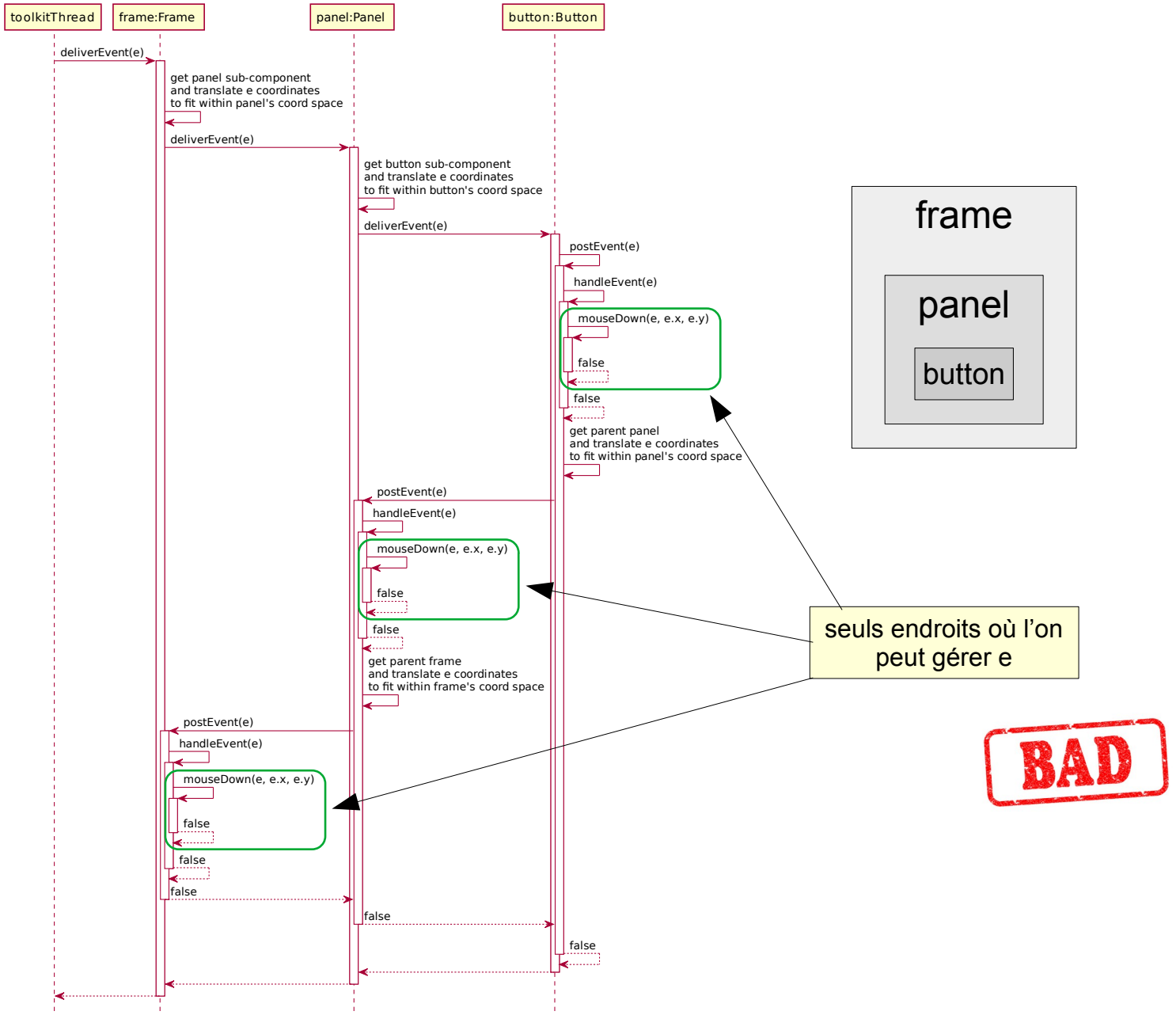
```
public class DownloadButton extends java.awt.Button {
    public boolean action(Event evt, Object what) {
        // télécharger...
    }
}
```

```
public class BrowseButton extends java.awt.Button {
    public boolean action(Event evt, Object what) {
        // naviguer...
    }
}
```

...

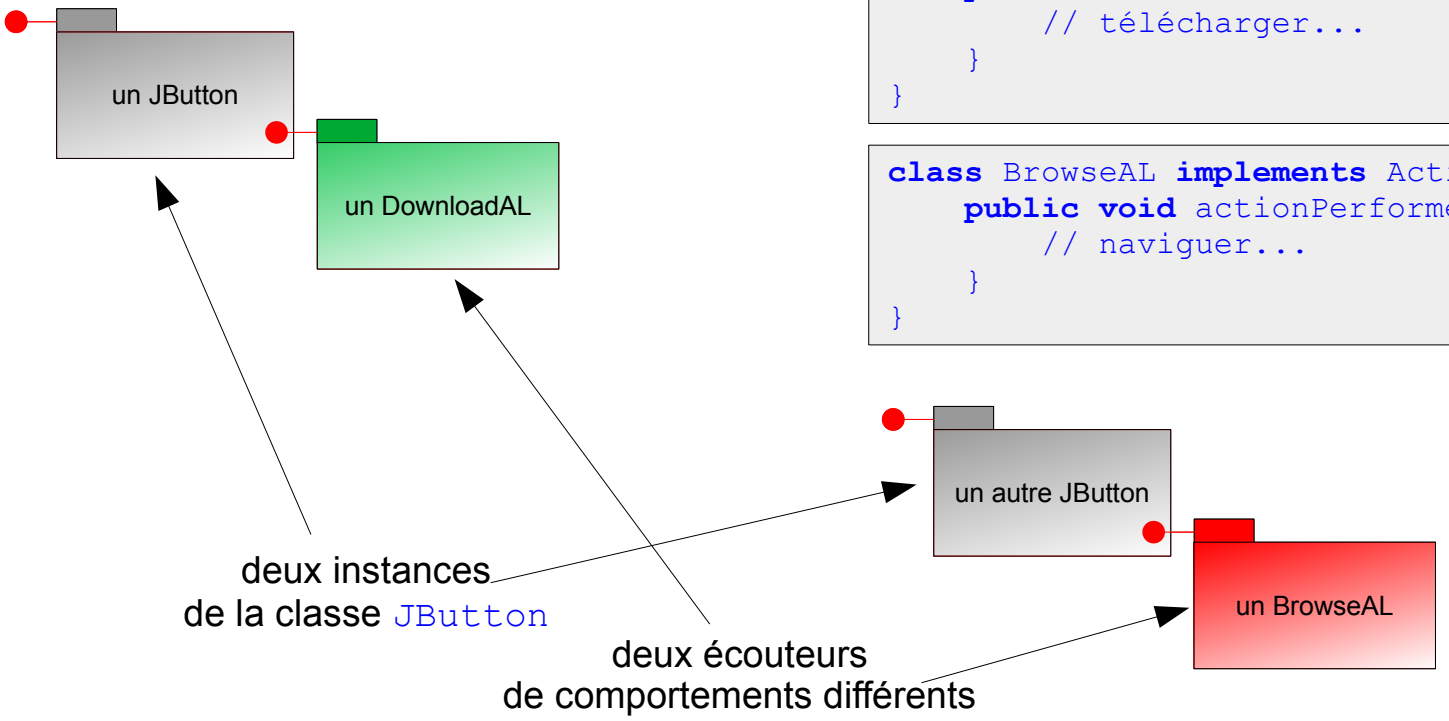


Exemple typique (modèle 1.0)



Historique (2/2)

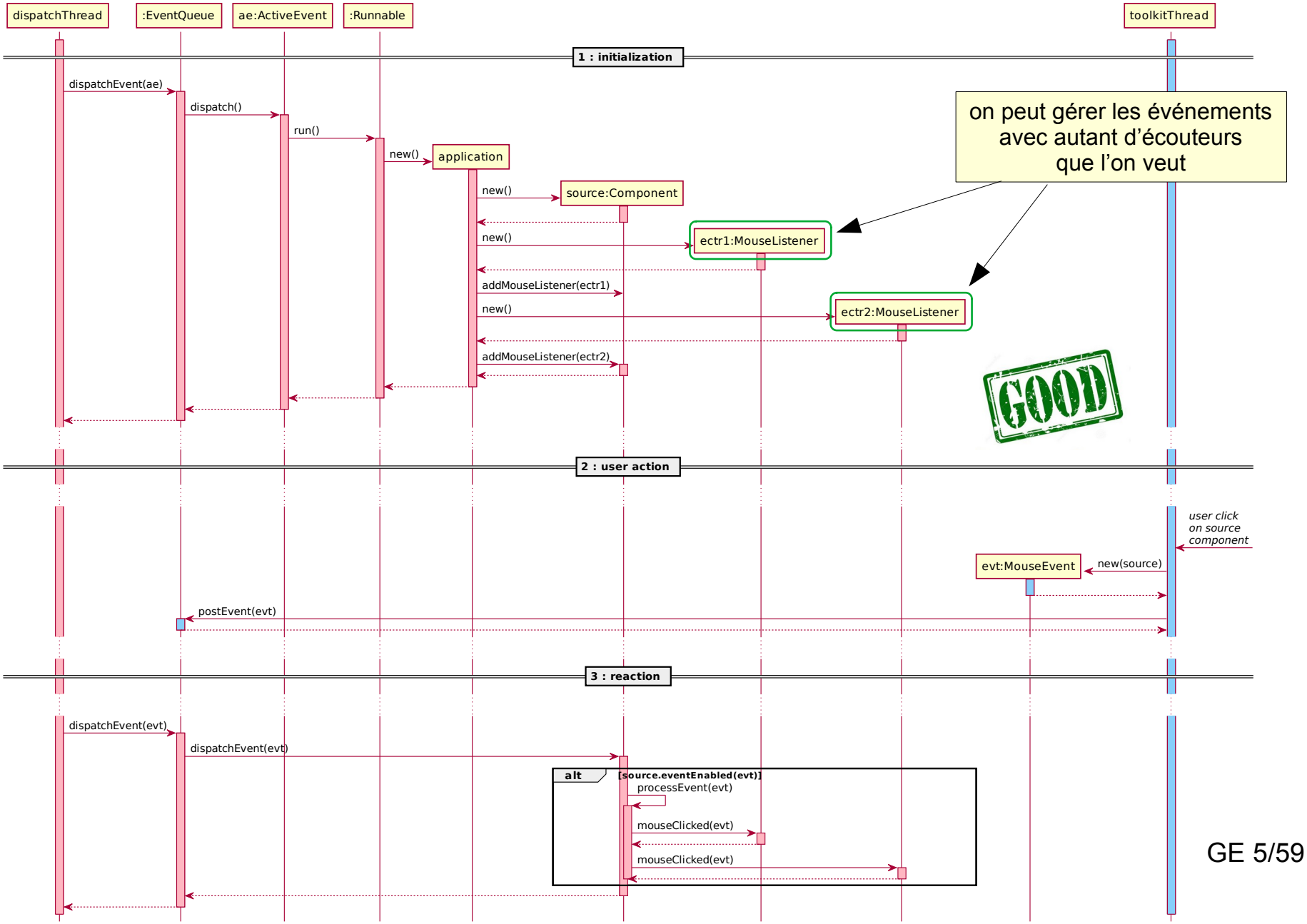
- Modèle 1.1 (*Delegation Event Model*)
  - événements traités par des écouteurs
  - écouteurs indépendants des composants-sources




```
class DownloadAL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // télécharger...
    }
}
```

```
class BrowseAL implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // naviguer...
    }
}
```

Exemple typique (modèle 1.1)



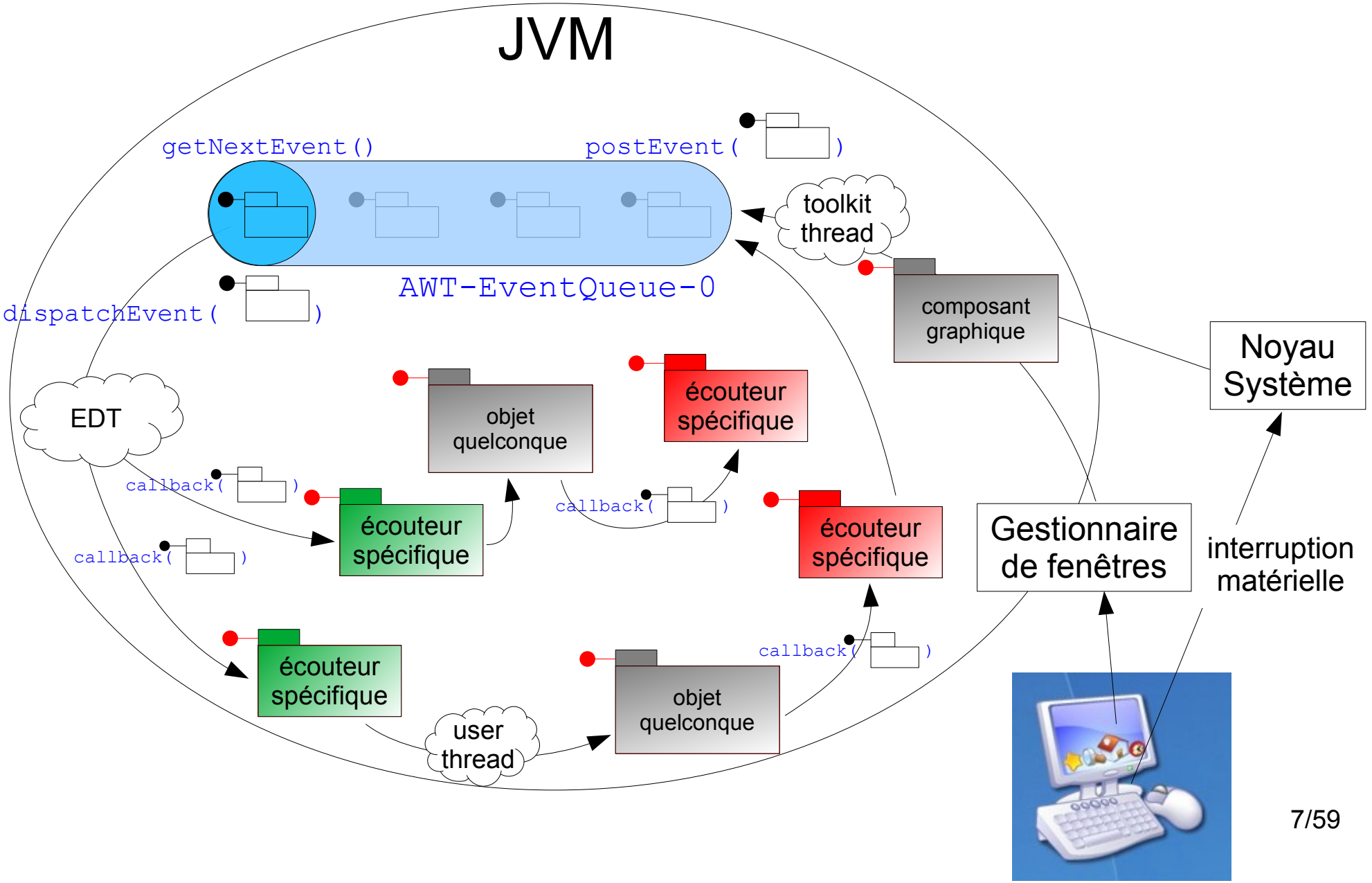
Codage des applications graphiques en Java

- Principe fondamental :
  - toute manipulation (directe ou indirecte) d'un composant graphique doit se faire sur le thread graphique (EDT)

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        new Appli().display();
    }
});
```

- *Event Dispatch Thread* (EDT) :
  - thread dédié à la gestion des composants graphiques
  - accède aux événements graphiques par une `EventQueue`

Schéma de principe



Catégories d'événements

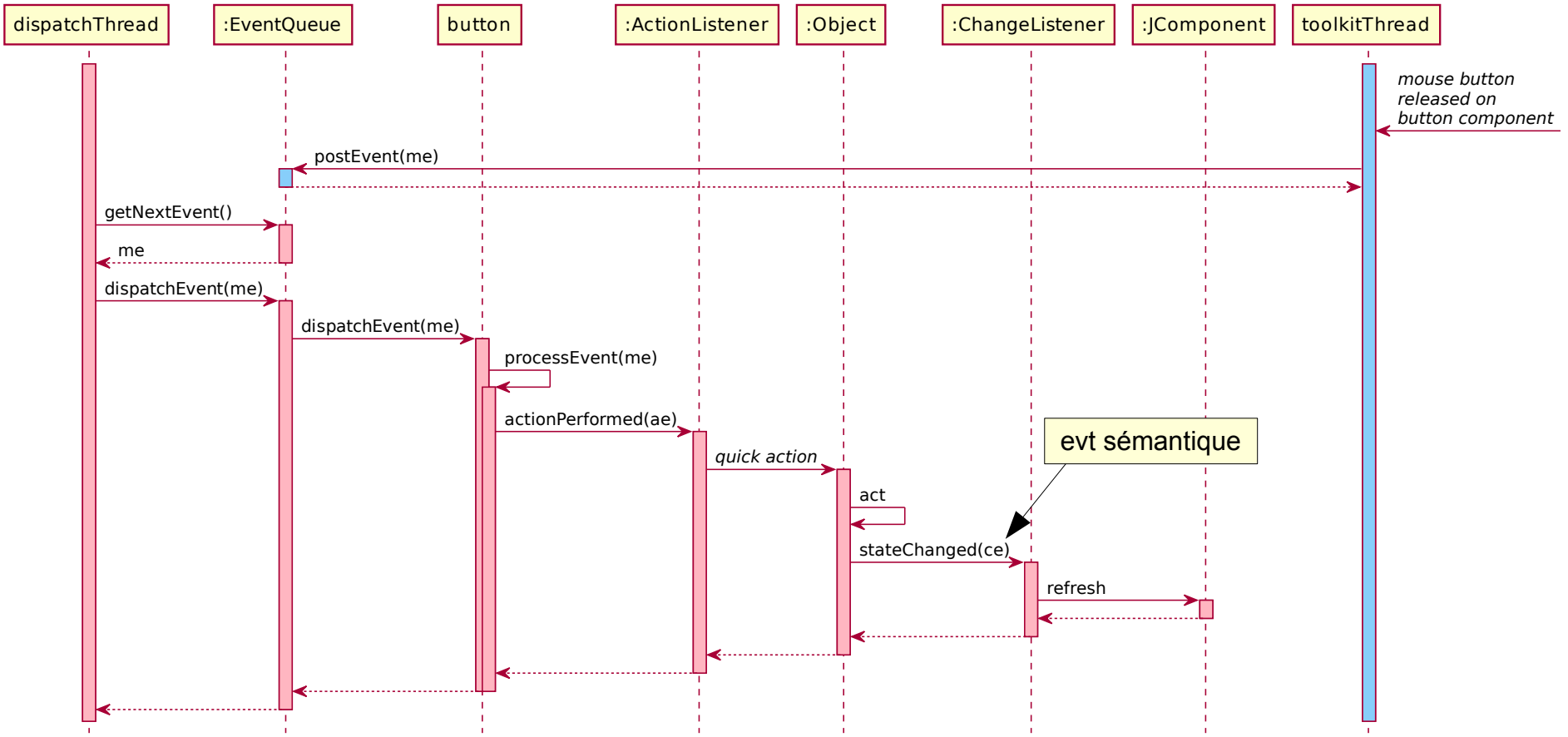
- **evt bas-niveau** ← écouteurs notifiés sur EDT
  - evt véhiculant un signal du SE
  - `AWTEvent` dont la source est un composant graphique
- **evt sémantiques** ← écouteurs notifiés sur... ça dépend !
  - evt véhiculant une intention élaborée
  - la source est un objet qcq

transmis à des écouteurs

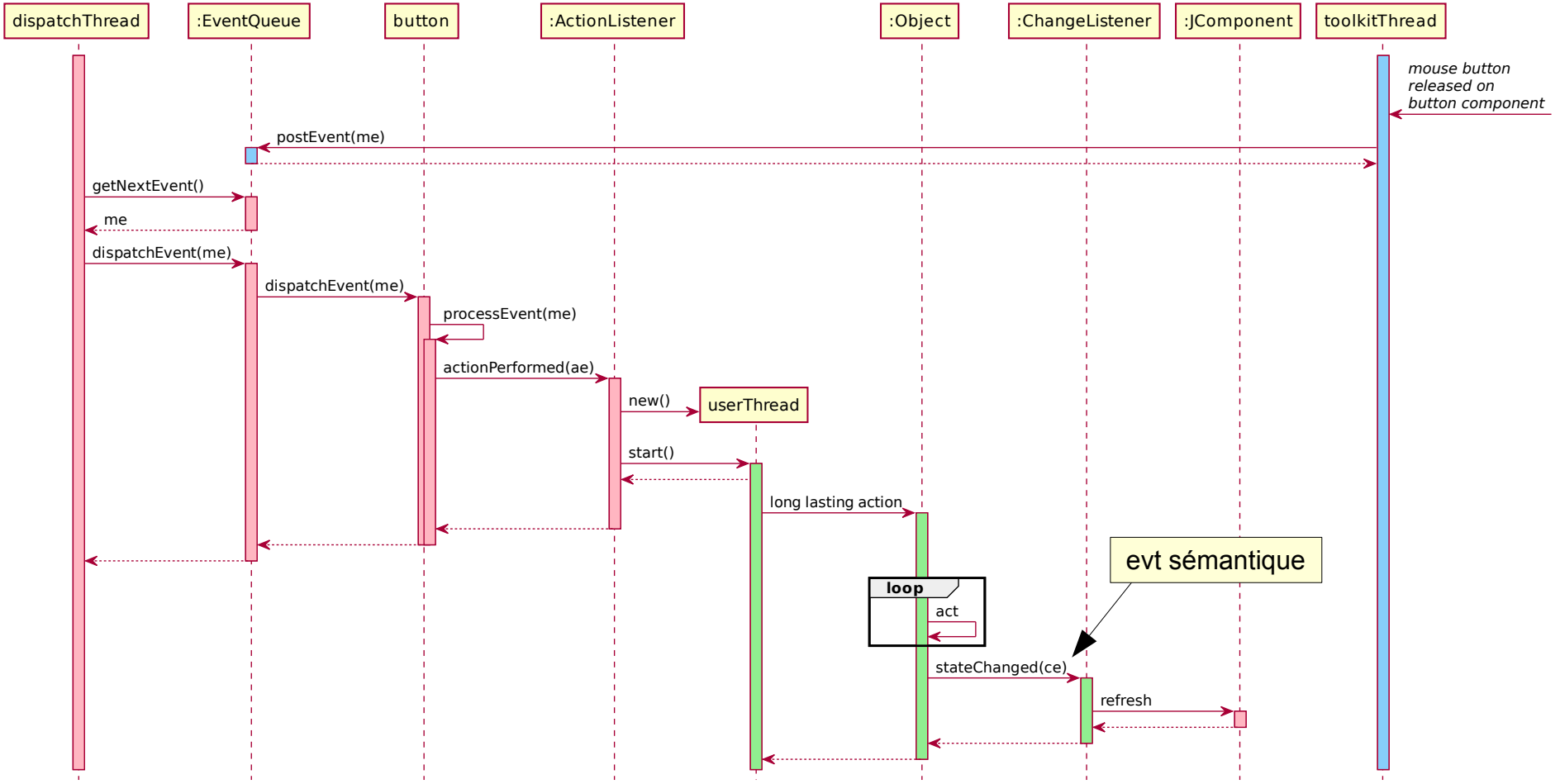
- **evt spéciaux** ← pas d'écouteurs, evt directement traités sur EDT
  - exécution de code sur EDT
    - `ActiveEvent` ← p. ex. : `invokeLater`, `invokeAndWait`
    - `PaintEvent`

pas d'écouteur

Événement sémantique gestion avec EDT uniquement



Événement sémantique gestion avec thread utilisateur



Attention : erreur classique !

- Contexte :
  - l'utilisateur clique (souris) sur un bouton
- Erreur méthodologique :
  - écouter les `MouseEvent`s du bouton pour détecter les actions souris de l'utilisateur sur ce bouton
- Pourquoi ?

- 3 manières distinctes pour activer un bouton :

3 x `MouseEvent`
+
1 x `ActionEvent`

3 x `KeyEvent`
+
1 x `ActionEvent`

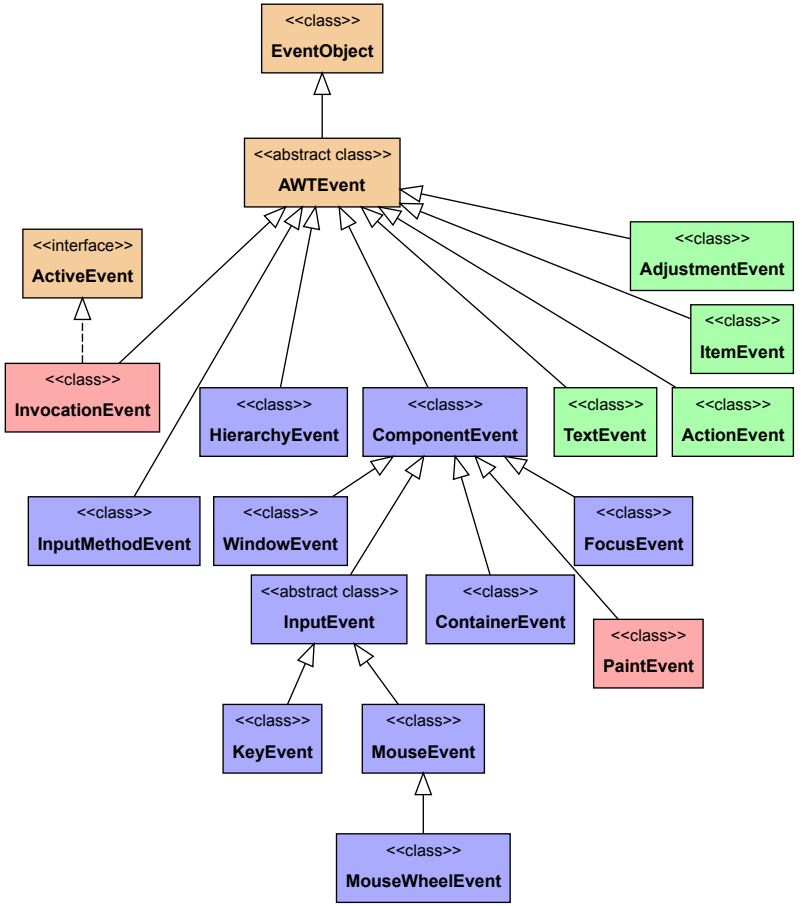
```
cmds = new EnumMap<Order, JB
for (Order o : Order.values(
    cmds.put(o, new JButton(
    cmds.get(o).setName(o.na
}
restart = new JButton("Nouve
restart.doClick();
output = new JTextArea(OUT_M
output.setEditable(false);
output.setLineWrap(true);
soAllower = new JCheckBox("A
```

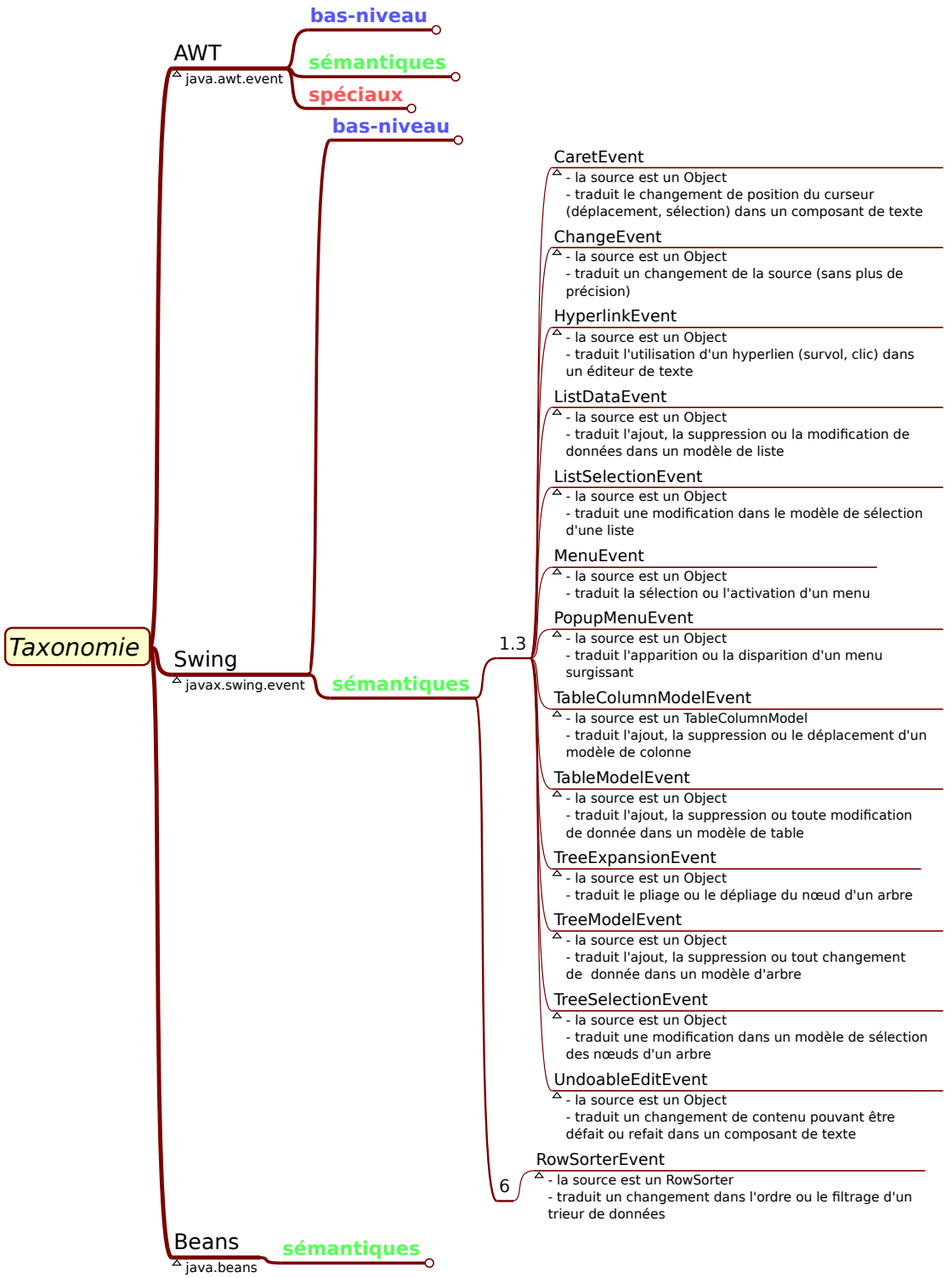
1 x `ActionEvent`

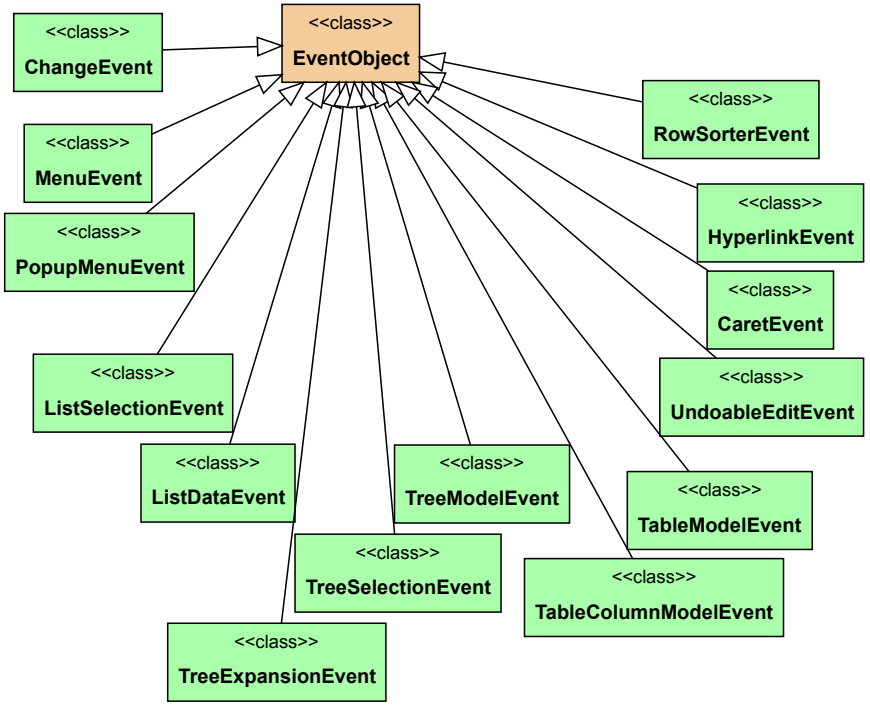
- Pour toujours réagir à l'activation d'un bouton :
  - écouter `ActionEvent` (événement sémantique)
  - ne pas écouter `MouseEvent` ou `KeyEvent` (événements bas niveau)

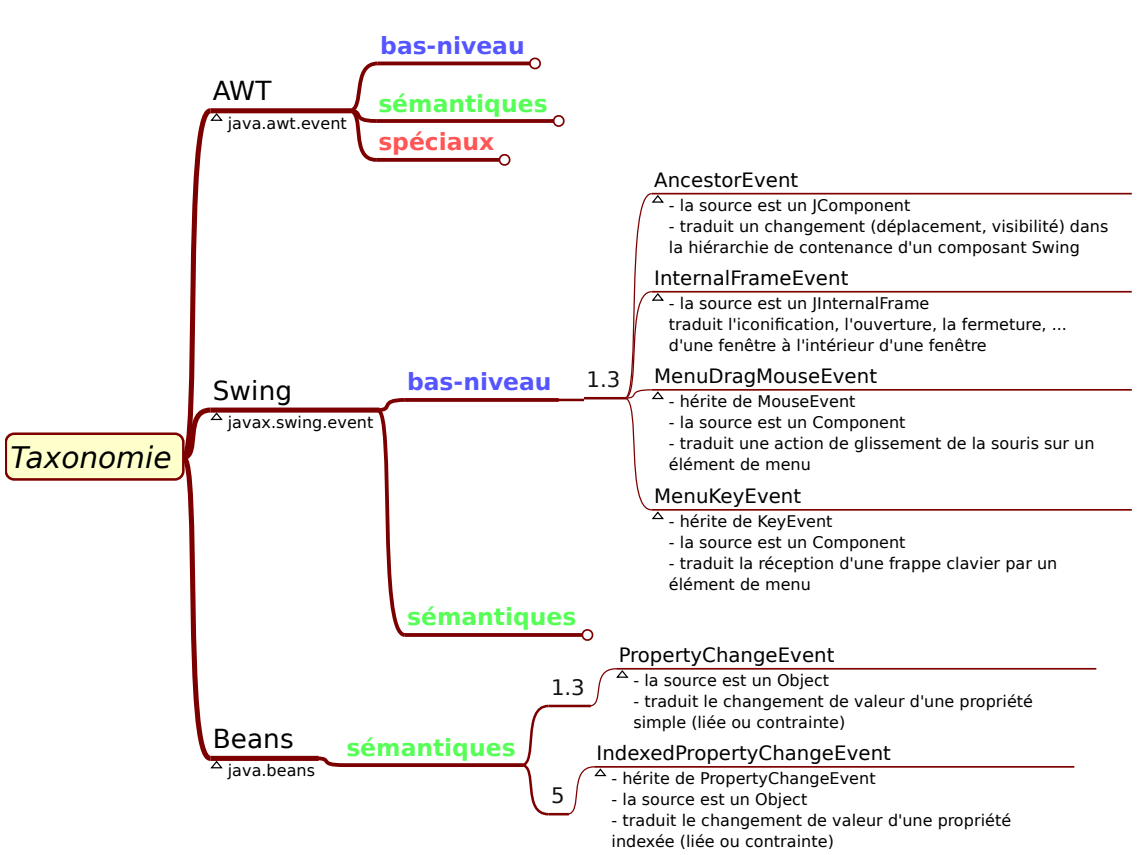
- **Taxonomie**
  - **AWT**
    - java.awt.event
      - **bas-niveau**
        - 1.1 **ComponentEvent**
          - la source est un Component
          - traduit un changement de taille, de position ou de visibilité d'un composant
        - 1.2 **InputMethodEvent**
          - la source est un Component
          - traduit la composition d'une partie d'un texte à l'aide d'alphabets complexes (1 lettre = combinaison de touches)
        - 1.3 **HierarchyEvent**
          - la source est un Component
          - traduit un changement (ajout, retrait, déplacement ou retaillage) dans la hiérarchie de contenance d'un composant
        - 1.4 **MouseWheelEvent**
          - hérite de MouseEvent
          - la source est un Component
          - traduit l'activation de la molette sur un composant
      - **sémantiques**
        - 1.1
          - **ActionEvent**
            - la source est un Object
            - traduit l'activation d'un élément de contrôle de l'interface graphique en vue d'exécuter une action spécifique
          - **AdjustmentEvent**
            - la source est un Adjustable
            - traduit un changement dans la valeur numérique bornée d'un objet ajustable
          - **ItemEvent**
            - la source est un ItemSelectable
            - traduit un changement dans la sélection d'un objet sélectionnable
          - **TextEvent**
            - la source est un Object
            - traduit un changement dans le texte d'un composant de texte
      - **spéciaux**
        - 1.1 **PaintEvent**
          - hérite de ComponentEvent
          - la source est un Component
          - traduit la nécessité de redessiner un composant
        - 1.2 **ActiveEvent**
          - interface
          - **InvocationEvent**
            - hérite de AWTEvent & implémente ActiveEvent
            - la source est un Object
            - traduit la nécessité d'exécuter la méthode run() d'un Runnable sur EDT
  - **Swing**
    - javax.swing.event
      - **bas-niveau**
      - **sémantiques**
  - **Beans**
    - java.beans
      - **sémantiques**

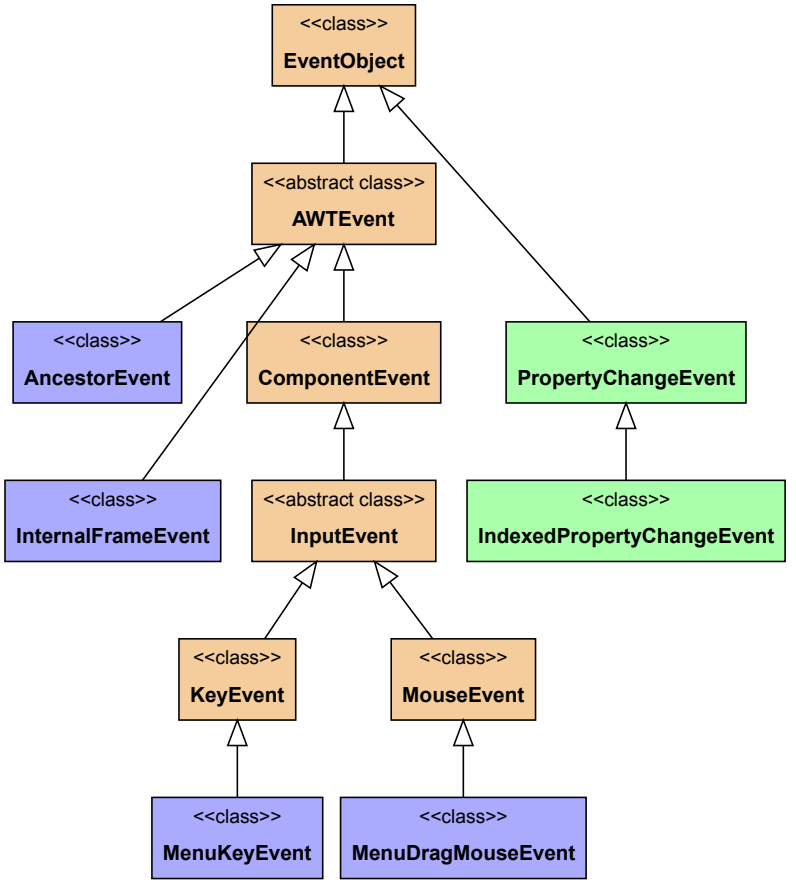
- **ContainerEvent**
  - la source est un Container
  - traduit un ajout ou une suppression de composant dans un conteneur
- **FocusEvent**
  - la source est un Component
  - traduit le gain ou la perte du focus pour un composant
- **InputEvent**
  - classe abstraite
  - **KeyEvent**
    - la source est un Component
    - traduit l'activation du clavier pour un composant
  - **MouseEvent**
    - la source est un Component
    - traduit l'activation de la souris pour un composant
- **WindowEvent**
  - la source est un Window
  - traduit l'iconification, l'ouverture, la fermeture, ... d'une fenêtre



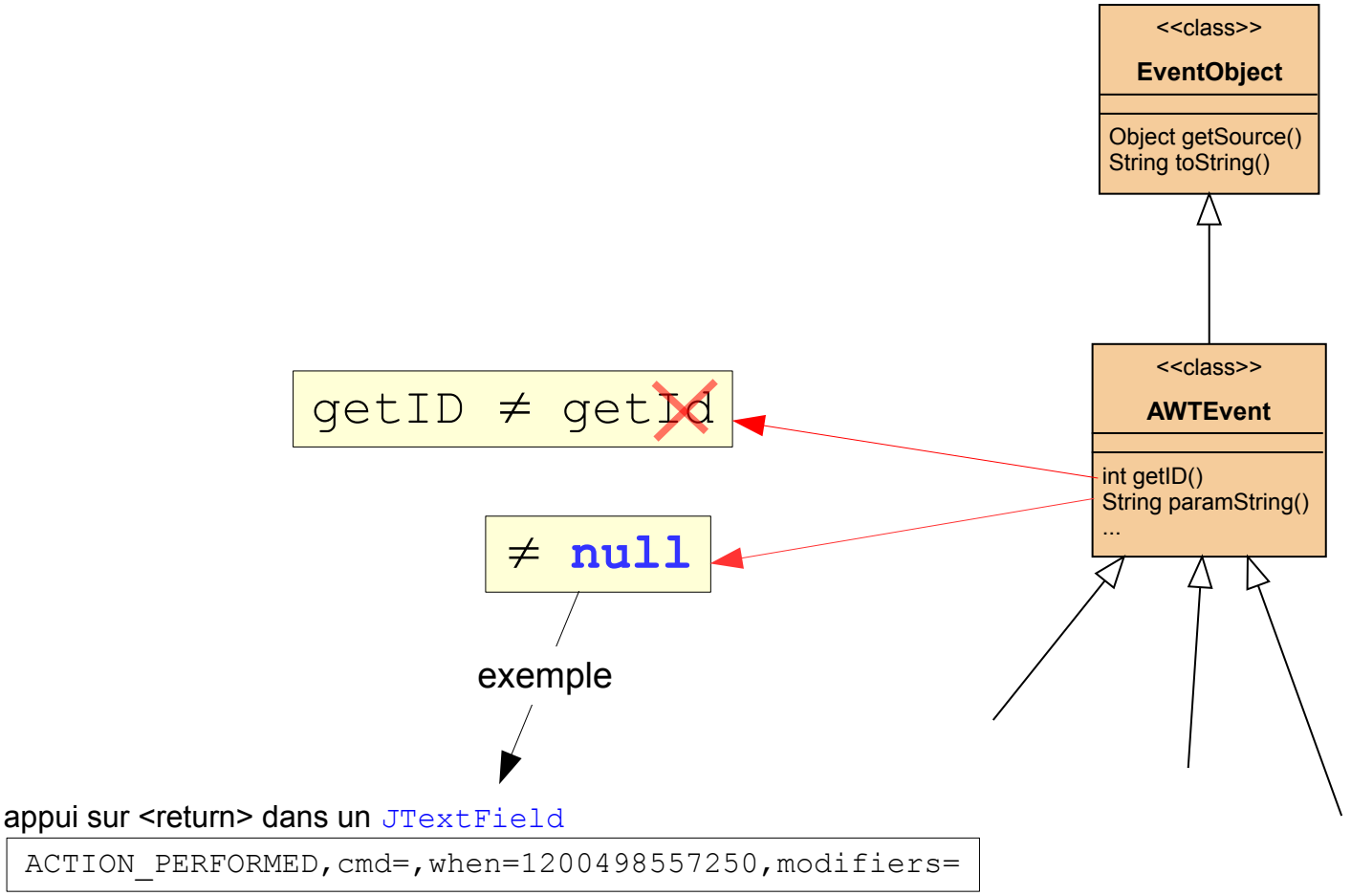

Taxonomie
AWT
java.awt.event
bas-niveau
sémantiques
spéciaux
Swing
javax.swing.event
bas-niveau
sémantiques
1.3
CaretEvent
- la source est un Object
- traduit le changement de position du curseur (déplacement, sélection) dans un composant de texte
ChangeEvent
- la source est un Object
- traduit un changement de la source (sans plus de précision)
HyperlinkEvent
- la source est un Object
- traduit l'utilisation d'un hyperlien (survol, clic) dans un éditeur de texte
ListDataEvent
- la source est un Object
- traduit l'ajout, la suppression ou la modification de données dans un modèle de liste
ListSelectionEvent
- la source est un Object
- traduit une modification dans le modèle de sélection d'une liste
MenuEvent
- la source est un Object
- traduit la sélection ou l'activation d'un menu
PopupMenuEvent
- la source est un Object
- traduit l'apparition ou la disparition d'un menu surgissant
TableColumnModelEvent
- la source est un TableColumnModel
- traduit l'ajout, la suppression ou le déplacement d'un modèle de colonne
TableModelEvent
- la source est un Object
- traduit l'ajout, la suppression ou toute modification de donnée dans un modèle de table
TreeExpansionEvent
- la source est un Object
- traduit le pliage ou le dépliage du nœud d'un arbre
TreeModelEvent
- la source est un Object
- traduit l'ajout, la suppression ou tout changement de donnée dans un modèle d'arbre
TreeSelectionEvent
- la source est un Object
- traduit une modification dans un modèle de sélection des nœuds d'un arbre
UndoableEditEvent
- la source est un Object
- traduit un changement de contenu pouvant être défait ou refait dans un composant de texte
6
RowSorterEvent
- la source est un RowSorter
- traduit un changement dans l'ordre ou le filtrage d'un trieur de données
Beans
java.beans
sémantiques


<<class>>
EventObject
<<class>>
ChangeEvent
<<class>>
MenuEvent
<<class>>
PopupMenuEvent
<<class>>
ListSelectionEvent
<<class>>
ListDataEvent
<<class>>
TreeExpansionEvent
<<class>>
TreeSelectionEvent
<<class>>
TreeModelEvent
<<class>>
TableColumnModelEvent
<<class>>
TableModelEvent
<<class>>
UndoableEditEvent
<<class>>
CaretEvent
<<class>>
HyperlinkEvent
<<class>>
RowSorterEvent

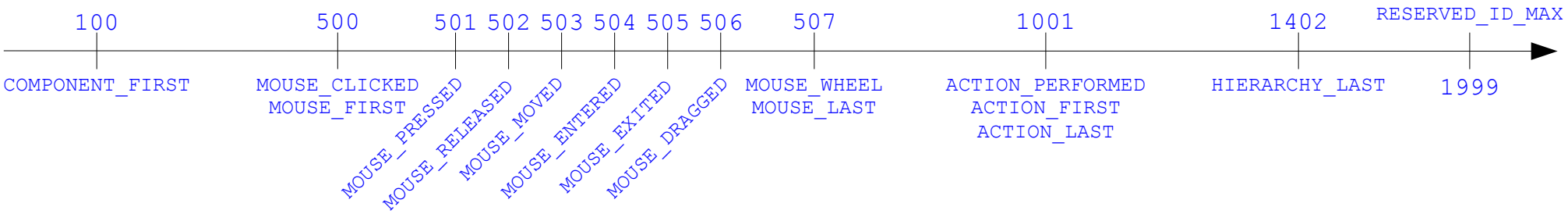

Taxonomie
AWT
java.awt.event
bas-niveau
sémantiques
spéciaux
Swing
javax.swing.event
bas-niveau
1.3
AncestorEvent
- la source est un JComponent
- traduit un changement (déplacement, visibilité) dans la hiérarchie de contenance d'un composant Swing
InternalFrameEvent
- la source est un JInternalFrame
traduit l'iconification, l'ouverture, la fermeture, ... d'une fenêtre à l'intérieur d'une fenêtre
MenuDragMouseEvent
- hérite de MouseEvent
- la source est un Component
- traduit une action de glissement de la souris sur un élément de menu
MenuKeyEvent
- hérite de KeyEvent
- la source est un Component
- traduit la réception d'une frappe clavier par un élément de menu
sémantiques
Beans
java.beans
sémantiques
1.3
PropertyChangeEvent
- la source est un Object
- traduit le changement de valeur d'une propriété simple (liée ou contrainte)
5
IndexedPropertyChangeEvent
- hérite de PropertyChangeEvent
- la source est un Object
- traduit le changement de valeur d'une propriété indexée (liée ou contrainte)


<<class>>
EventObject
<<abstract class>>
AWTEvent
<<class>>
AncestorEvent
<<class>>
ComponentEvent
<<class>>
PropertyChangeEvent
<<class>>
InternalFrameEvent
<<abstract class>>
InputEvent
<<class>>
IndexedPropertyChangeEvent
<<class>>
KeyEvent
<<class>>
MouseEvent
<<class>>
MenuKeyEvent
<<class>>
MenuDragMouseEvent

Les deux classes principales d'événements



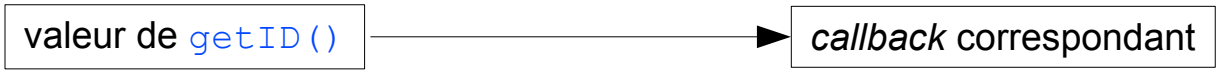
Champ `id` de `AWTEvent`



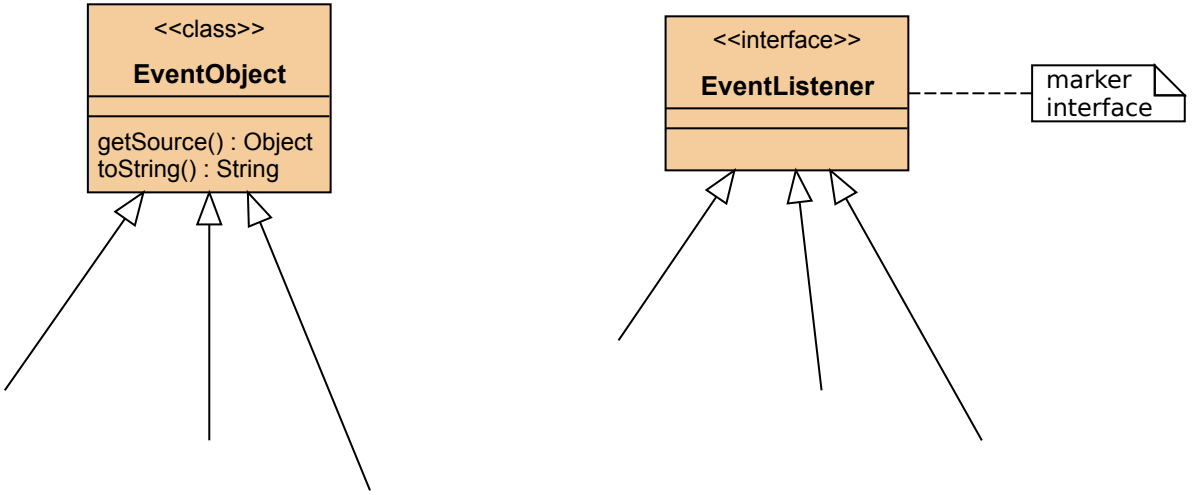
- Ex. `MouseEvent` :
  - `MOUSE_FIRST` ≤ `getID()` ≤ `MOUSE_LAST`
- `getID()` ≤ `RESERVED_ID_MAX`
  - événements prédéfinis dans l'API

Utilité du champ `id`

- Correspondances `getID()` → *callback*
- Ex. : `MouseEvent` / `MouseListener`
  - `MOUSE_CLICKED` → `mouseClicked(MouseEvent)`
  - `MOUSE_PRESSED` → `mousePressed(MouseEvent)`
  - `MOUSE_RELEASED` → `mouseReleased(MouseEvent)`
  - `MOUSE_ENTERED` → `mouseEntered(MouseEvent)`
  - `MOUSE_EXITED` → `mouseExited(MouseEvent)`



java.util
EventObject/EventListener



Correspondance event/listener

- `XEvent` → `XListener`
  - `ActionEvent` → `ActionListener`
  - `DocumentEvent` → `DocumentListener`
  - `MouseEvent` → `MouseListener`, `MouseMotionListener`
- `XListener` avec ++ méthodes → adaptateur
  - `FocusListener` (2 méthodes) → `FocusAdapter`
  - `WindowListener` (7 méthodes) → `WindowAdapter`
  - `ActionListener` (1 méthode) → pas d'adaptateur

cf. https://docs.oracle.com/javase/tutorial/uiswing/events/api.html

Écouteurs associés à `Component`

- `ComponentListener` → visibilité, taille, position
- `FocusListener` → cible du clavier
- `KeyListener` → clavier
- `MouseListener` → souris
- `MouseMotionListener` → souris
- `MouseWheelListener` → molette
- `HierarchyListener` → super hiérarchie graphique
- `HierarchyBoundsListener` → super hiérarchie graphique
- `InputMethodListener` → composition complexe de texte
- `PropertyChangeListener` → état d'un composant

Écouteurs associés à `Container`

- Tous ceux de `Component`
- `ContainerListener` → ajout/retrait de composants

Écouteurs associés à `JComponent`

- Tous ceux de `Container`
- `AncestorListener` → visibilité, mouvement
- `VetoableChangeListener` → état d'un composant

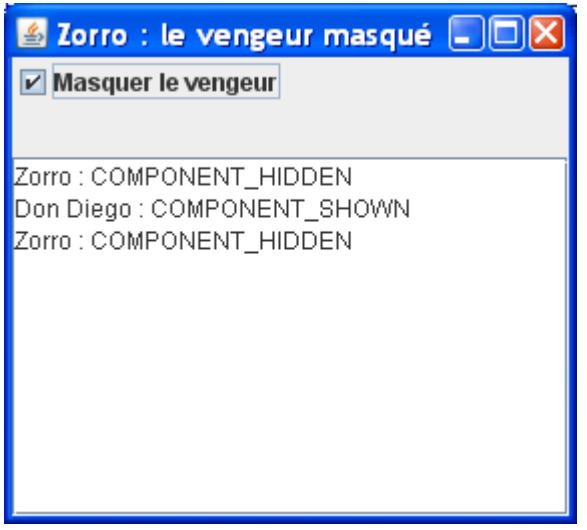
Rappel : affichabilité et visibilité

- `isVisible()`
  - visible (1)
- `isDisplayable()`
  - affichable (2)
- `isShowing()`
  - visible↑ & affichable

```
JFrame f = new JFrame("...");
```
`f` (-)
```
JButton b = new JButton("...");
```
`b` (1) / `f` (-)
```
f.add(b);
```
`b` (1) / `f` (-)
```
f.pack();
```
`b` (1 & 2) / `f` (2)
```
f.setVisible(true);
```
`b` (1↑ & 2) / `f` (1↑ & 2)

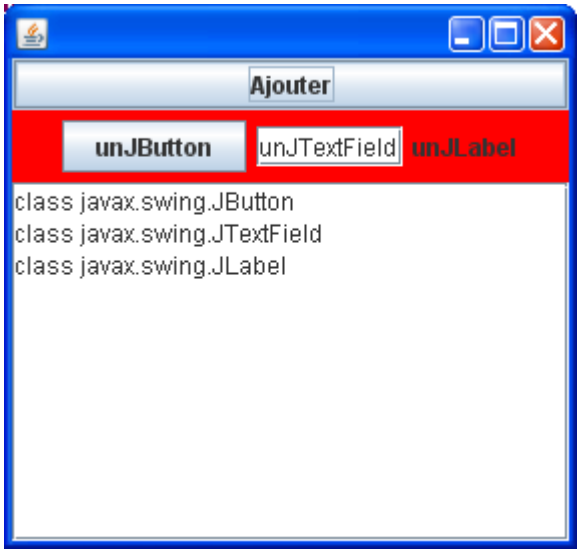
Exemple : détection de masquage

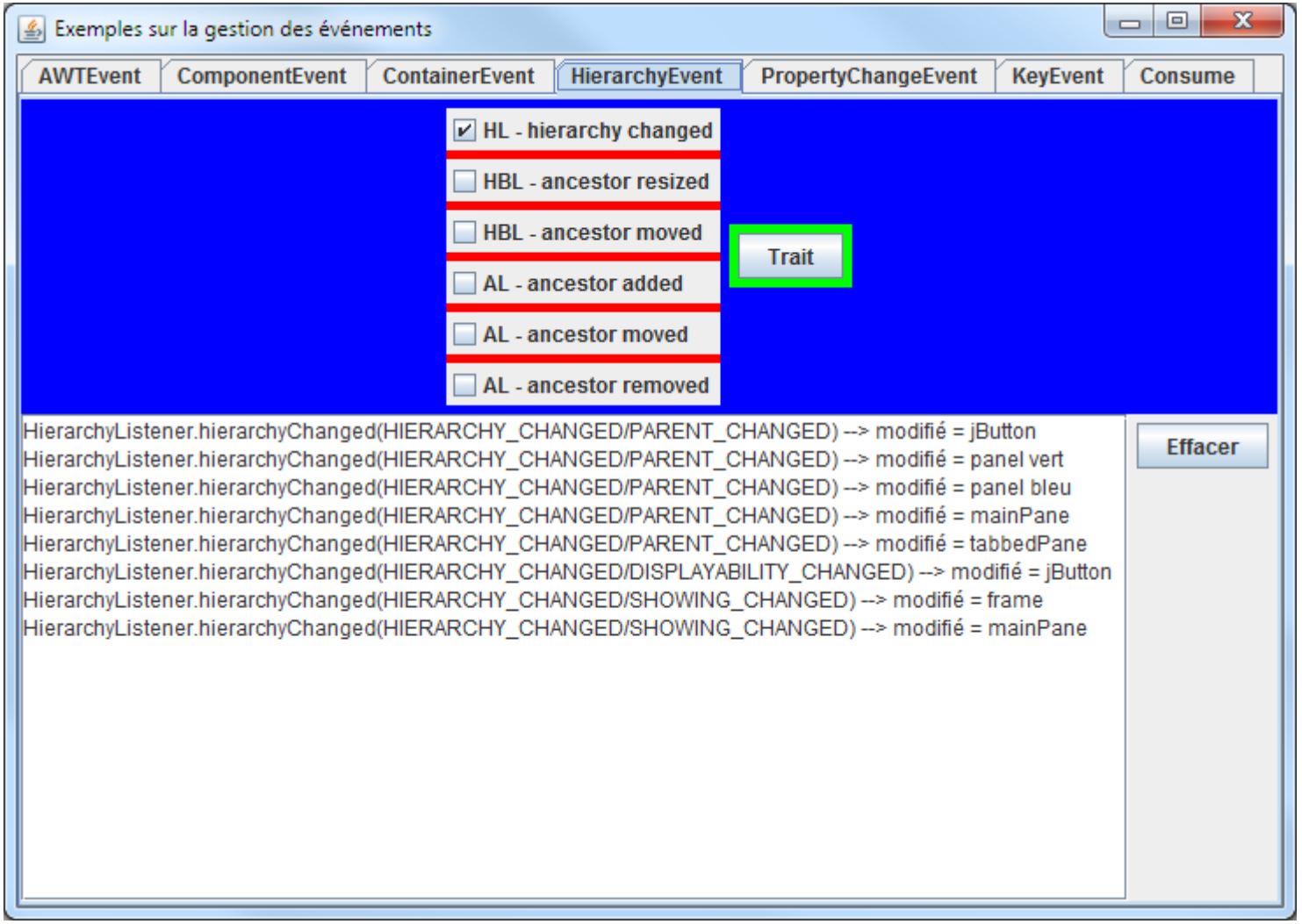
```
final JTextArea output = new JTextArea(11, 25);
final JLabel jl = new JLabel("Le vengeur");
jl.setBackground(Color.RED);
jl.setOpaque(true);
jl.addComponentListener(new ComponentAdapter() {
    public void componentHidden(ComponentEvent e) {
        output.append("Zorro : COMPONENT_HIDDEN\n");
    }
    public void componentShown(ComponentEvent e) {
        output.append("Don Diego : COMPONENT_SHOWN\n");
    }
});
final JCheckBox jcb = new JCheckBox("Masquer le vengeur");
jcb.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jl.setVisible(!jcb.isSelected());
    }
});
```



Exemple : détection d'ajout de composant

```
final JTextArea output = new JTextArea(11, 25);
final JPanel p = new JPanel();
p.setBackground(Color.RED);
p.addContainerListener(new ContainerAdapter() {
    public void componentAdded(ContainerEvent e) {
        output.append(e.getChild().getClass() + "\n");
    }
});
JButton b = new JButton("Ajouter");
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        p.add(getNewChild());
        p.revalidate();
    }
});
```





Le fichier `events.jar` est téléchargeable sur UniversiTICE, section « Ressources »
c'est un fichier *exécutable* **et** qui contient les *sources*

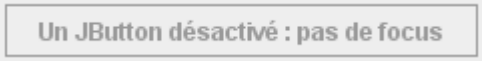
Remarque

- « *[These] events are provided for notification purposes ONLY. The AWT will automatically handle [according changes] internally so that GUI layout and displayability works properly regardless of whether a program is receiving these events or not.* »
- `ComponentListener`, `ContainerListener` et `Hierarchy[Bounds]Listener` ne doivent pas être utilisés pour gérer les modifications des composants ou de leur hiérarchie : ces mécanismes sont déjà implantés dans l'API

Notion de focus

- Focus = capacité pour un composant d'être la cible des actions clavier à venir
- Certains composants peuvent avoir le focus...
  - Boutons
  - Champs de texte

- ... d'autres non
  - Étiquettes
  - Boutons désactivés



Gestion du focus dans `Component`

- Focus acceptable ou non
  - **`boolean`** `isFocusable()`
- Modification du comportement
  - `setFocusable(`**`boolean`**`)`
- Réquisition du focus
  - **`boolean`** `requestFocusInWindow()`
  - retour **`false`** → garantie d'échec
  - retour **`true`** → succès possible mais non garanti
  - si important → vérifier avec un `FocusListener`

java.awt.event
FocusEvent

- `Component getOppositeComponent()`
  - l'autre composant (celui qui n'est pas la source)
- **`boolean`** `isTemporary()`
  - type de changement de focus (permanent ou temporaire)
- Correspondance avec les `FocusListener` :
  - focus obtenu (`id == FOCUS_GAINED`) → `focusGained()`
  - focus perdu (`id == FOCUS_LOST`) → `focusLost()`