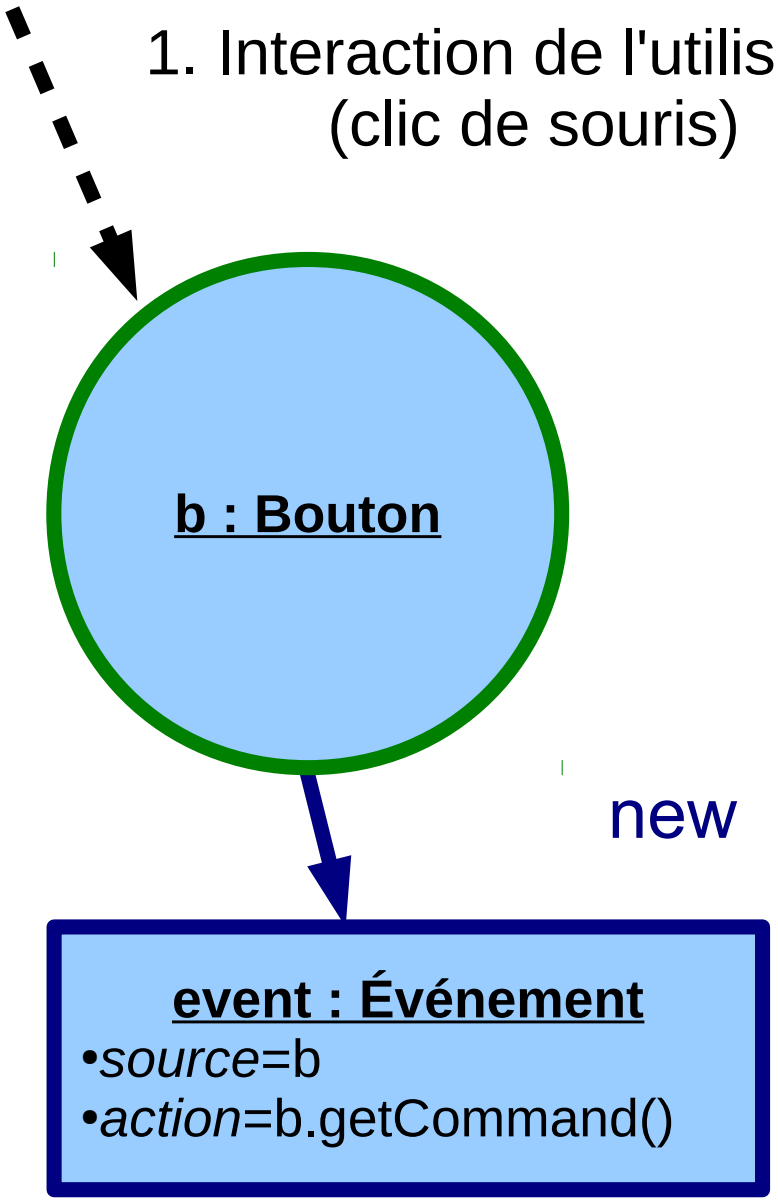
Gestion des événements en **Java**

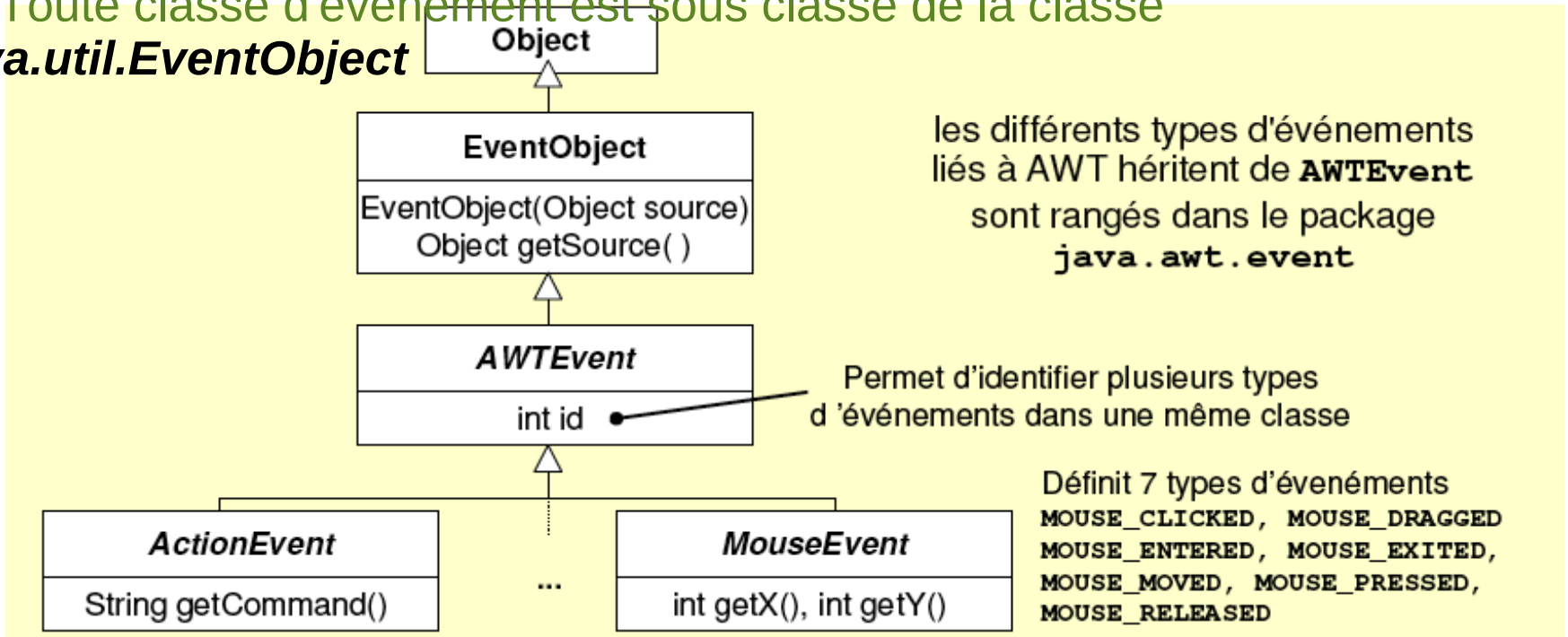
Les *Graphic Events* (événements graphiques)



Un **événement** est un **objet** créé lorsqu'une action particulière est effectué sur un **objet source.**

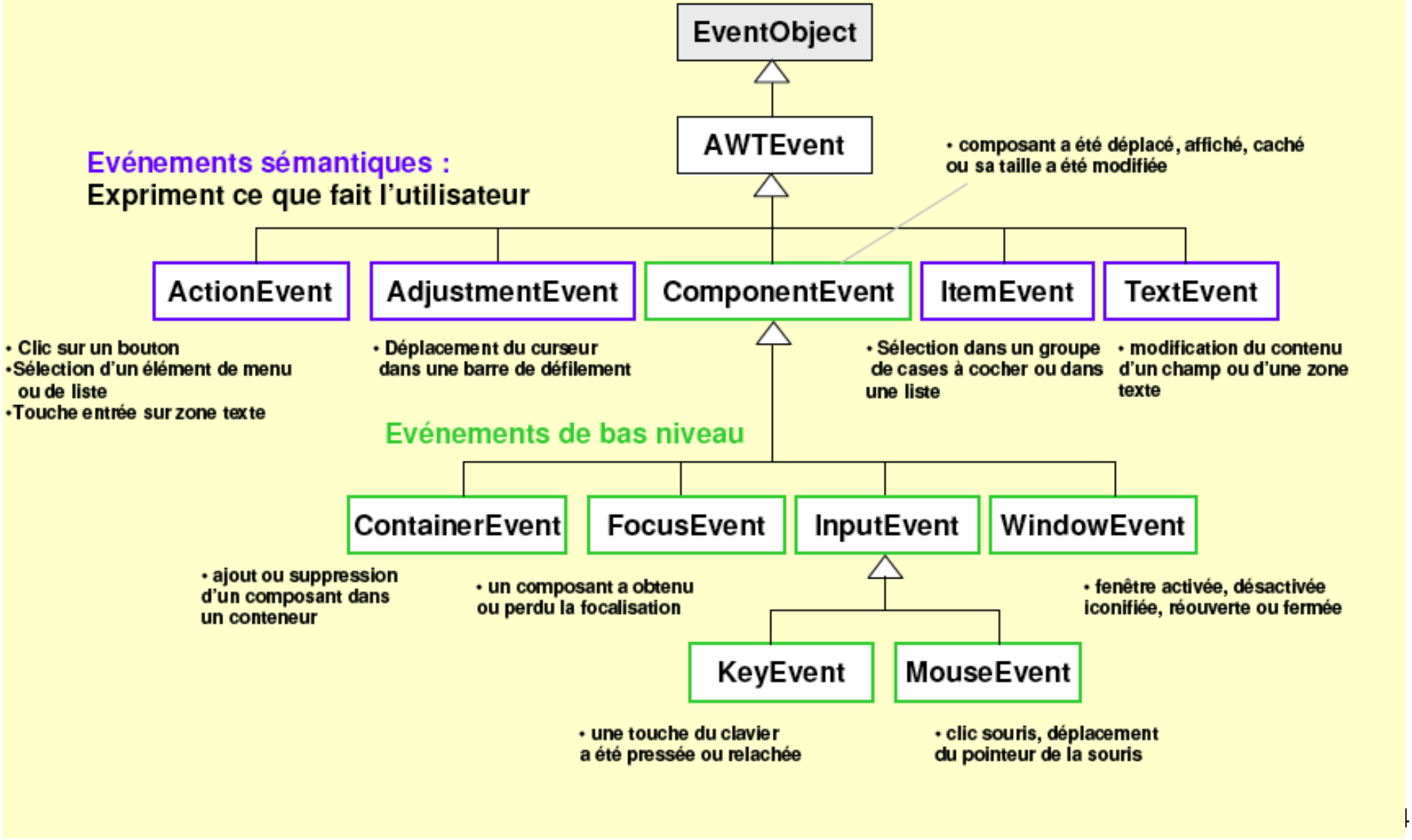
Les Objets *Event*

- Un objet événement encapsule une information spécifique à une instance d'événement
    - exemple : un événement représentant un clic souris contient la position du pointeur souris
- Les différents types d'événements sont représentés par des classes différentes :
    - ActionEvent, MouseEvent …
- → Toute classe d'événement est sous classe de la classe ***java.util.EventObject***

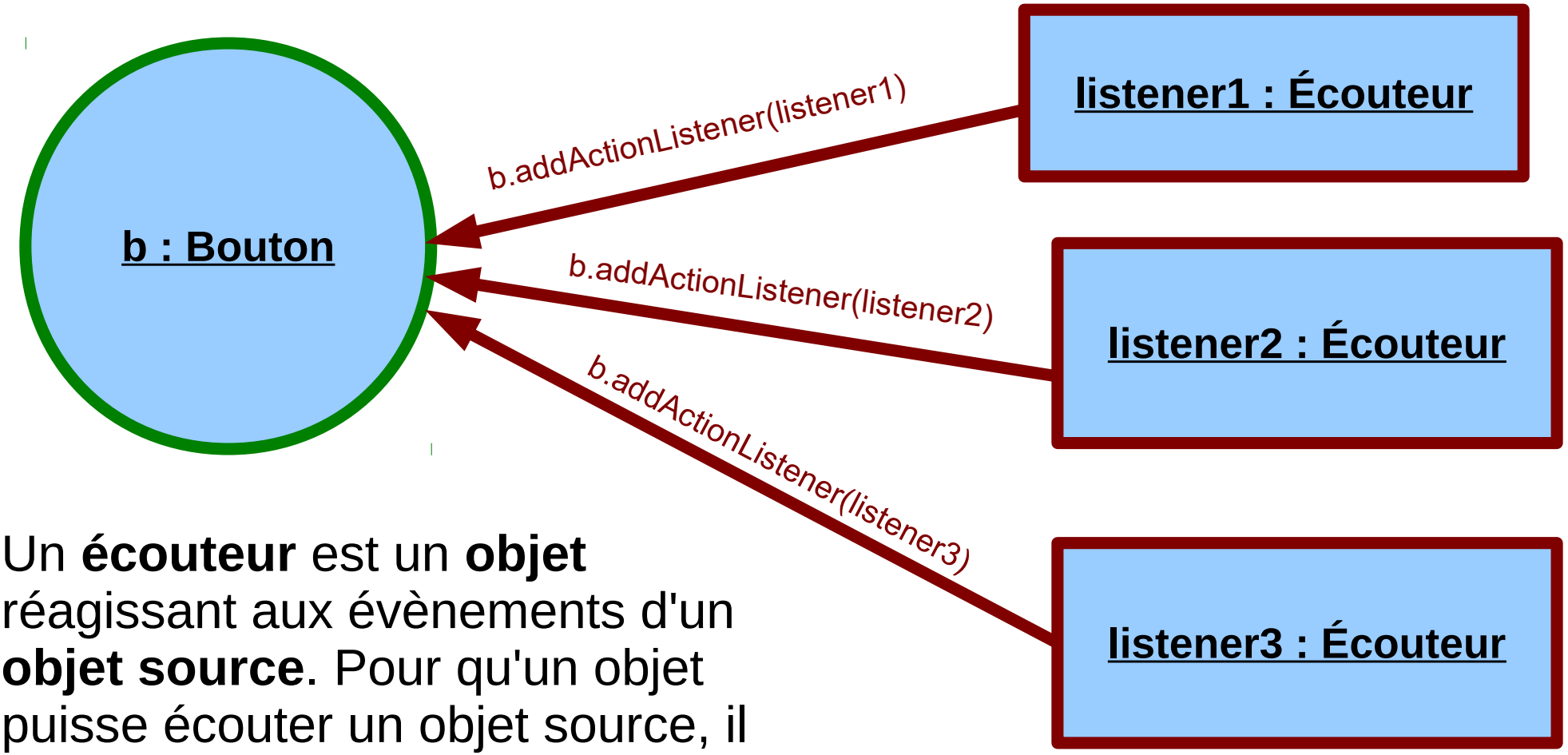


Hiérarchie des Events

→ Une partie de la hiérarchie des événements AWT



Les *Listeners* (écouteurs)



Un **écouteur** est un **objet** réagissant aux évènements d'un **objet source**. Pour qu'un objet puisse écouter un objet source, il doit s'enregistrer auprès de celui-ci

Récepteur d'événement ou *Listener*

- Un récepteur d'événements est un objet **qui doit être prévenu** ("notified") par la source lorsque certains événements se produisent
- les notifications d'événements se font **en invoquant des méthodes de l'objet** qui écoute, un événement étant transmis en paramètre
- → Pour chaque classe d'événements **une interface spécifique** définit les méthodes à appeler pour notifier les événements de cette classe

exemple : interface ActionListener pour les ActionEvent

```
package java.awt.event;
import java.util.EventListener;
public interface ActionListener extends EventListener {
    /** Invoked when an action occurs.*/
    public void actionPerformed(ActionEvent e);
}
```

Toute classe désirant recevoir des notifications d'un événement donné devra implémenter cette interface.

- un récepteur d'ActionEvent doit implémenter l'interface ActionListener

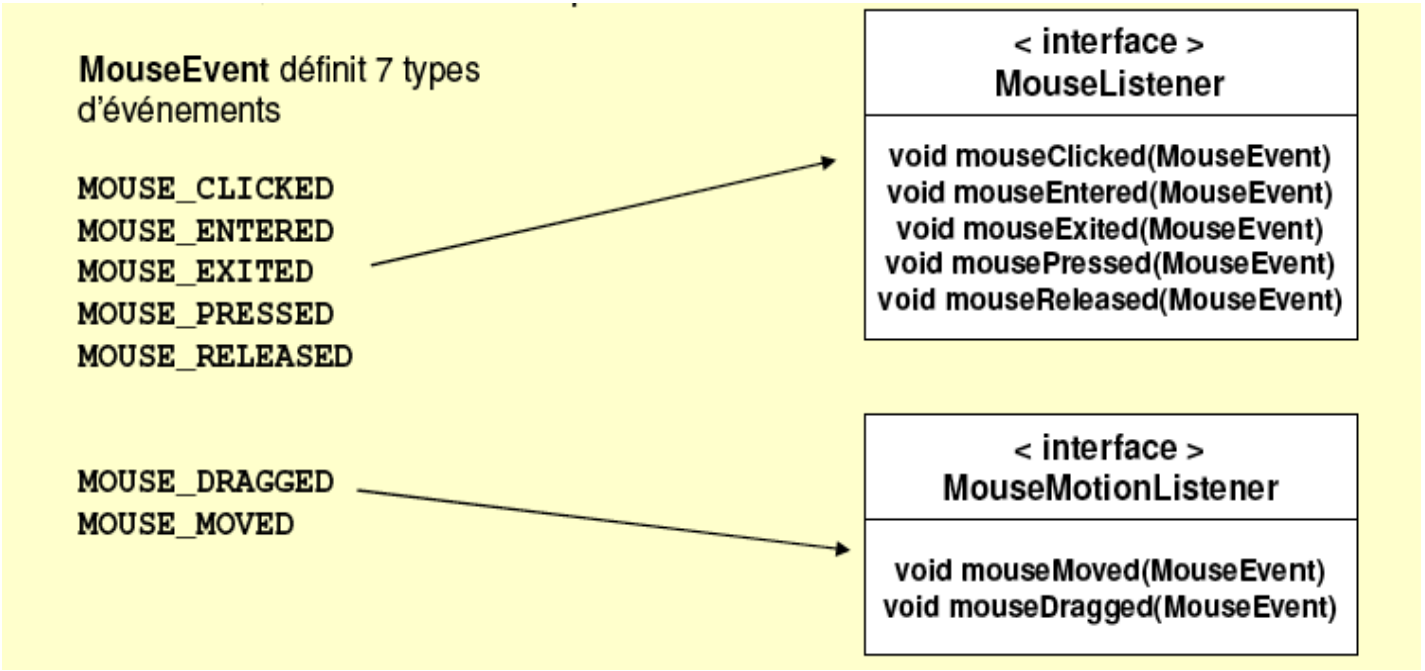
Interface d'écoute des événements

- Toutes les interfaces d'écoute d'événements héritent de **java.util.EventListener**
- Par convention toutes les interfaces des récepteurs d'événements ont des noms de la forme **<EventType>Listener**
    - exemple : les événements de AWT et les interfaces correspondantes pour les récepteurs

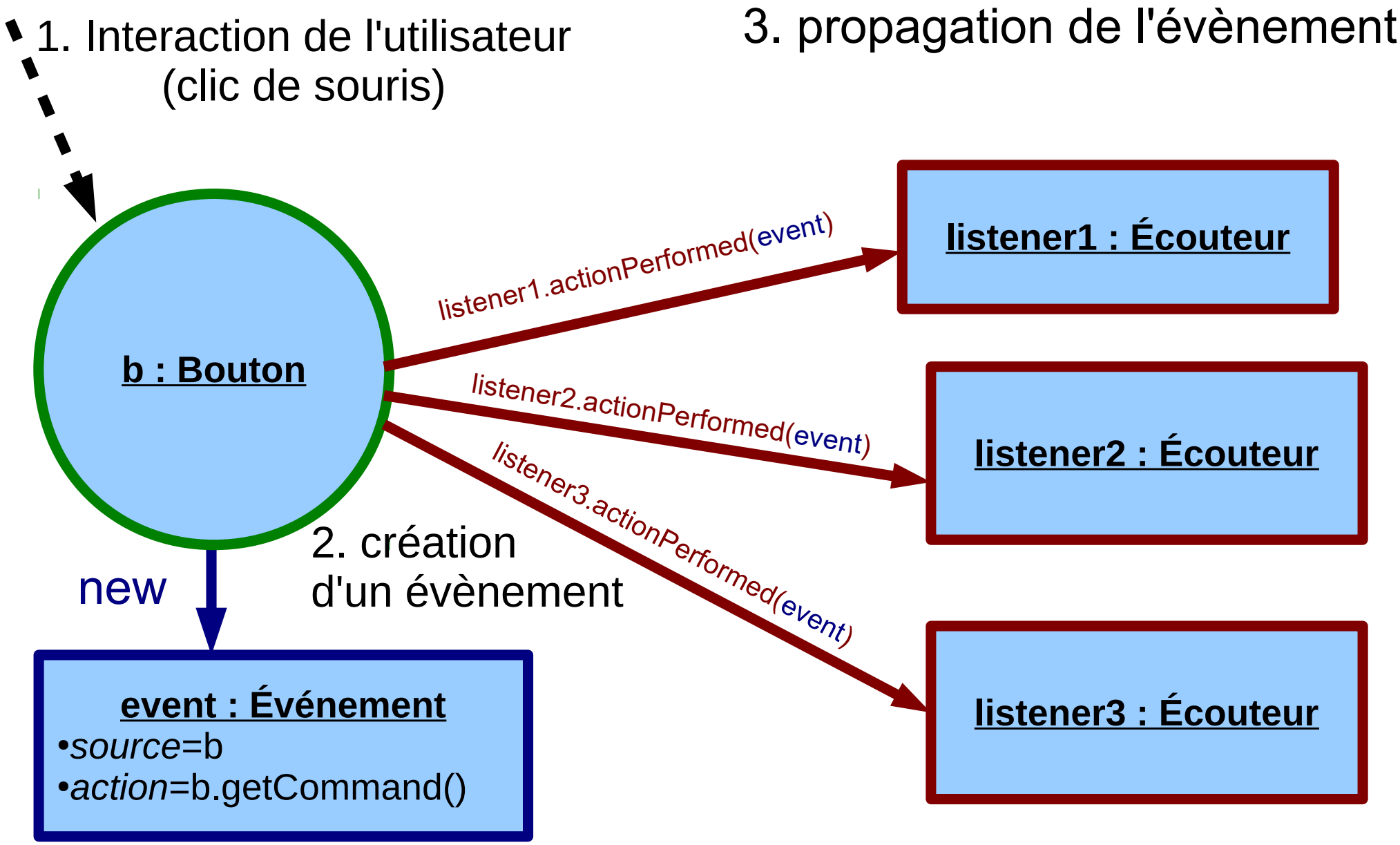
Classe d 'événement	Interface d 'écoute
ActionEvent	ActionListener
AdjustmentEvent	AdjustmentListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener
ItemEvent	ItemListener
KeyEvent	KeyListener
MouseEvent	MouseListener
	MouseMotionListener
TextEvent	TextListener
WindowEvent	WindowListener

Méthodes des Interfaces d'écoute

- Une interface d'écoute d'événements peut contenir un nombre **quelconque** de méthodes, **chacune correspondant à un événement différent**

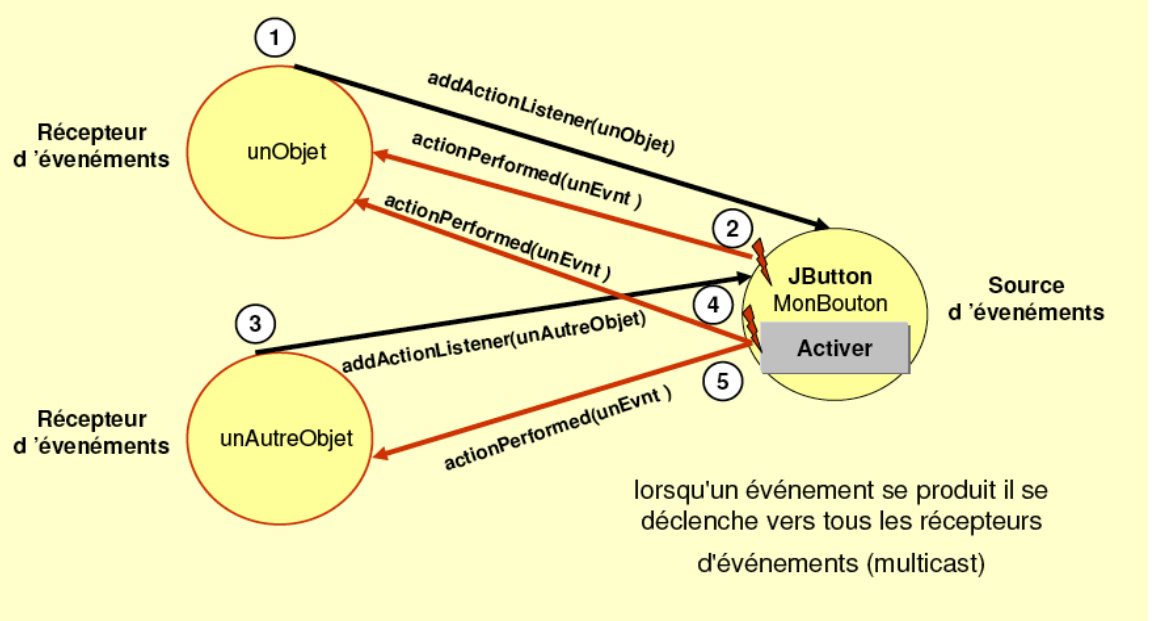


Propagation des événements



Enregistrement des *Listener*

- Une source d'événements **propose des méthodes d'enregistrement** dont la signature a la forme suivante :
  - `public void add<ListenerType>(<ListenerType> listener)`



A tout moment un objet récepteur d'événements peut annuler sa demande de Notification: une source d'événements pour une interface d'écoute d'événements propose aussi des méthodes d'annulation de notification dont la signature a la forme suivante :

- `public void remove<ListenerType>(<ListenerType> listener)`

Exemple : *MouseMotionListener*

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public  class ZoneDessin  extends  JPanel implements MouseMotionListener {
   private BarreEtat be;

   public ZoneDessin(BarreEtat be) {
      setBackground(Color.white);
      setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
      this.be = be;
      this.addMouseMotionListener(this);
   }

   public void mouseMoved(MouseEvent e) {
      be.afficheCoord(e.getX(),e.getY());
   }

   public void mouseDragged(MouseEvent e) {
      be.afficheCoord(e.getX(),e.getY());
   }


} // ZoneGraphique
```

① L 'objet zone graphique va être à l 'écoute des événements MouseEvent de type MouseMotion

③ L 'objet zone graphique s 'enregistre lui même comme récepteur des événements MouseEvent de type MouseMotion qu'il est susceptible de générer

② L 'objet zone graphique utilise les informations contenues dans l 'objet MouseEvent qui lui est transmis pour mettre à jour la barre d'état.

Gestion des événements en résumé

→ **identifier l'objet à l'origine des événements (souvent un composant)**

- **identifier le type de l 'événement** que l 'on veut intercepter pour cela : lister dans sa classe toutes les méthodes de type **addXXXListener**
- créer une classe qui implémente l'interface associée à l'événement que l'on veut gérer
    - **celle du composant** (ou du conteneur du composant) à l'origine de l 'événement ("facilité" d 'implémentation)
    - **une classe indépendante** qui détermine la frontière entre l 'interface graphique (émission des événements) et ce qui représente la logique de l 'application (traitement des événements). Une bonne séparation permet de faciliter l'évolution du logiciel.
- Implémenter dans cette classe la (les) méthode(s) associées à l'événement.
- L'événement passé en paramètre contient des informations qui peuvent être utiles (position du curseur, état du clavier, objet source de l'événement).

Des adaptateurs aux classes internes

- Le modèle événementiel de JAVA 1.1 peut rapidement devenir difficile à gérer
  - si un ***Listener*** est à l'écoute d'un grand nombre sources,
  - si un objet est à l'écoute d'événements issus de plusieurs sources du même type (le récepteur ne peut implémenter qu'une seule fois l'interface réceptrice)
- → Cela peut conduire à du code difficile à lire et/ou difficile à écrire
- Une solution consiste à introduire un objet "médiateur" entre la source d'événements et le récepteur d'événements
    - son rôle : adapter la source aux besoins spécifiques d'où le nom d'adaptateur d'événements

Exemple

```
import java.awt.event.*;
import java.awt.*;

public class AdaptateurEfface implements ActionListener
{
    ZoneDessin zone;
    public adaptateurEfface(ZoneDessin z) {
       zone = z;
    }
    public void actionPerformed(ActionEvent e)
    {
       zone.efface();
    }
}
```

Adaptateurs d'événements

```
import java.awt.event.*;
import java.awt.*;

public class AdaptateurAnnule implements ActionListener
{
    ZoneDessin zone;
    public AdaptateurEfface(ZoneDessin z) {
       zone = z;
    }

    public void actionPerformed(ActionEvent e) {
       zone.annule();
    }
}
```

Dans le code de BarreOutils

```
...
BarreOutils(ZoneDessin zd) {
  ...
  JButton b;
  this.add(b= new JButton("Défaire"));
  b.addActionListener(new AdaptateurAnnule(zd));
  this.add(b = new JButton("Tout effacer"));
  b.addActionListener(new AdaptateurEfface(zd));
  ...
}
...
```

- Simplification de `ZoneDessin`
- Meilleur découplage des différents éléments de l'interface utilisateur

Mais
- multiplication du nombre de classes
- classes pas vraiment réutilisables

→ Problème ca fait beaucoup de nouvelles petites classes et le code est dispersé, et plus difficile à lire …

Utilisation des classe anonymes comme adapteurs

Alors que faire ?

Classe anonyme adaptateur d'événements pour le ***JButton***

```
import java.awt.event.*;
import java.awt.*;
Import java.swing.*;
public class BarreOutils extends JPanel {

   public BarreOutils(final ZoneDessin zd) {
      ...
      JButton bDefaire = new JButton("Défaire");
      this.add(bDefaire);
      JButton bEffacer = new JButton("Tout effacer");
      this.add(bEffacer);
      ...
      bDefaire.addActionListener(
         new ActionListener() {
           public void actionPerformed(ActionEvent e)
           {
              zd.defaire();
           }
         }
      );
      bEffacer.addActionListener(
         new ActionListener() {
           public void actionPerformed(ActionEvent e)
           {
              zd.efface();
           }
         }
      );
      ...

}
```