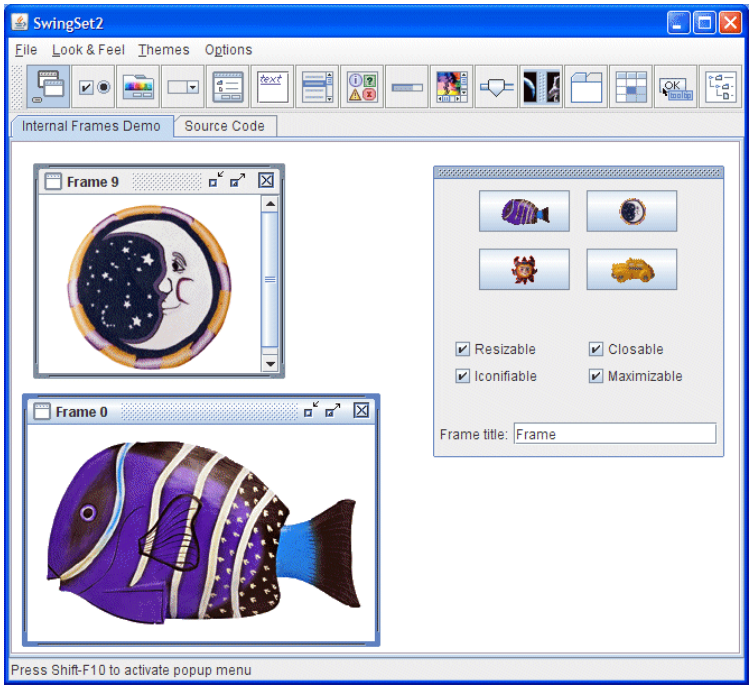
AWT et les écouteurs d'événement

Programmation graphique et événementielle en Java

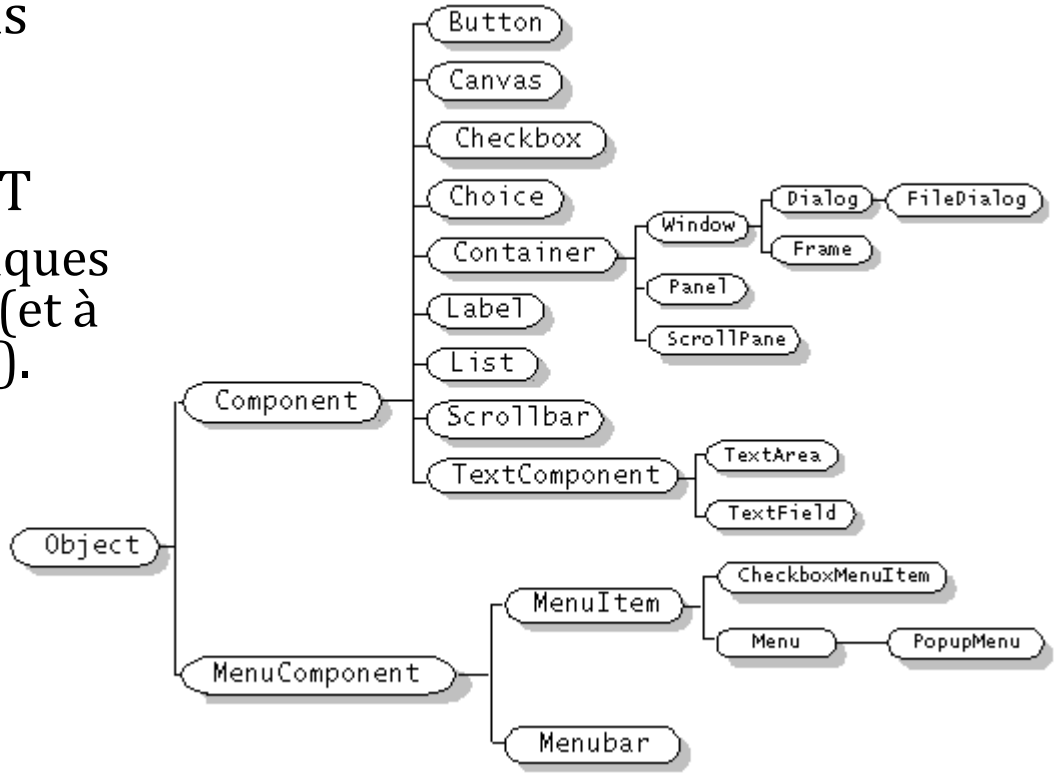
Introduction

- Plusieurs environnements disponibles
  - Standard : AWT, Swing, JavaFX
  - Extension : SWT (Eclipse)
- Affichage graphique fenêtré
  - Interface graphique utilisateur ou GUI (menus, onglets, widgets, etc.)
  - Dessin*, images, graphiques,
  - 3D



Abstract Window Toolkit

- Historiquement, AWT est le premier environnement graphique de Java et le plus simple.
- Hiérarchie des classes AWT
  - Tous les éléments graphiques dérivent de composants (et à fortiori de la classe objet).
  - Les conteneurs permettent d'agencer le contenu
  - Le Canvas permet de dessiner.



Le Canvas

- Zone rectangulaire dans laquelle on peut dessiner

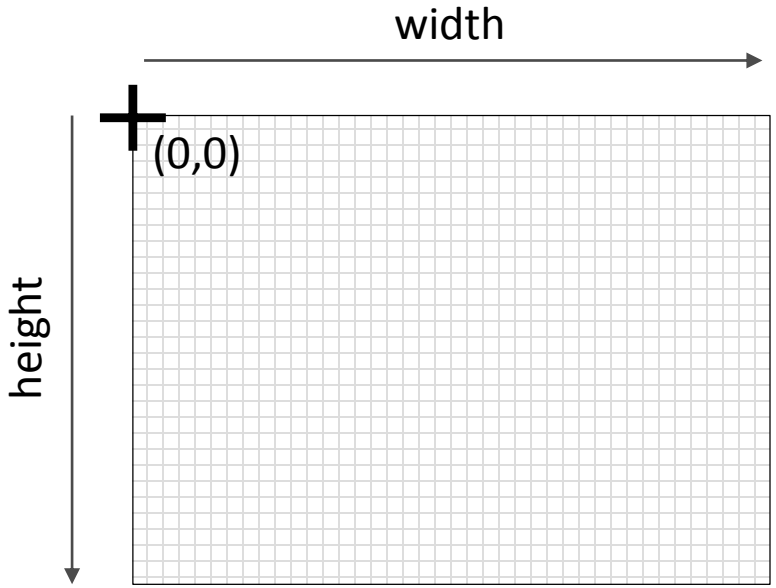
  L'unité du dessin est le pixel. Les dimensions du Canvas définissent une grille rectangulaire de pixels, chacun muni d'une paire de coordonnées.

- On peut intercepter des événements dans un Canvas

  Exemple: le clic sur un pixel

- La classe Canvas est destinée à être dérivée pour hériter de ses fonctionnalités.

  En particulier, la méthode 'paint' doit être redéfinie (surchargée) pour obtenir un affichage personnalisé.



```java
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Frame;
import java.awt.Graphics;

public class Dessin extends Canvas {

    public static void main(String args[]) {
        Dessin d = new Dessin();
        Frame f = new Frame();
        f.add(d);
        f.setSize(500,500);
        f.setTitle("Ma première app graphique");
        f.setVisible(true);

    }

    public void paint(Graphics g) {
        g.setColor(Color.blue);
        g.drawOval(10, 10, 300, 200);
    }

}
```

Structure élémentaire

L'application consiste à créer un Dessin qu'on place dans une fenêtre AWT (Frame).

Pour cela, notre classe Dessin dérive de la classe Canvas (mot clé 'extends'). Elle hérite donc de toutes les propriétés et des fonctionnalités du Canvas.

On doit surcharger la méthode paint() pour définir un affichage personnalisé. Les méthodes du contexte graphique ('Graphics', passé en paramètre) permettent de dessiner.

La taille du Canvas dépend (par défaut) de la taille de son conteneur (ici la fenêtre).

Exercices

Exercice c1

- Dessiner une cible comme illustré ci-contre

  Diamètres des cercles concentriques : 40, 150, 300, 400 pixels

  Dimensions du "carton" : (0,0)x(400,400)

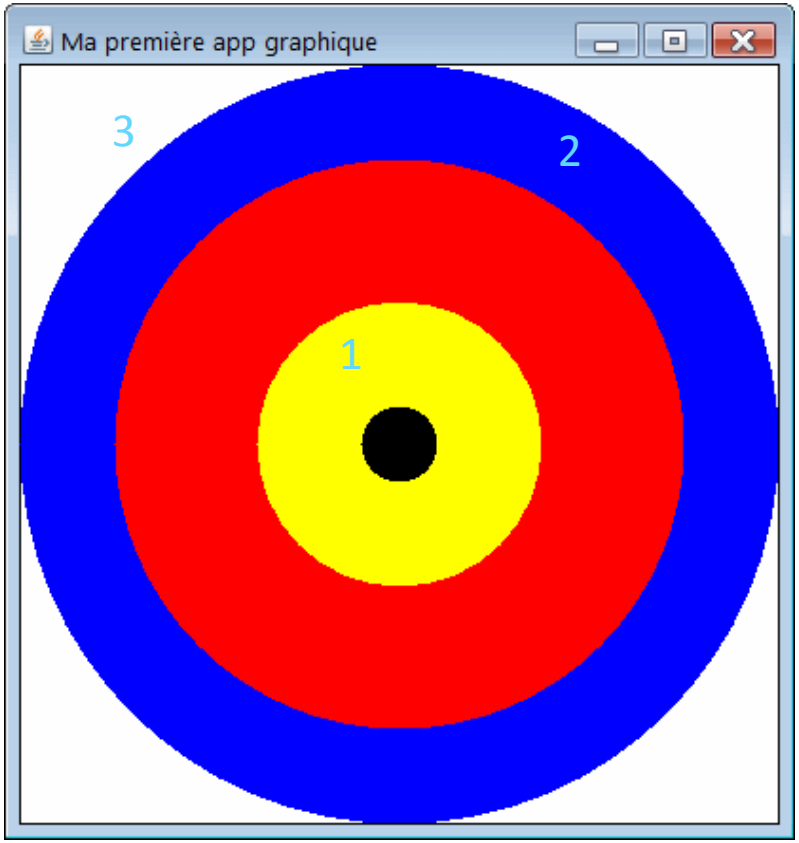
- Simuler le lancer de 3 fléchettes dans le carton.

  Une fléchette est représentée son numéro de lancer (1, 2 ou 3) centrée sur les coordonnées du lancer.

- Comptabiliser le score des 3 lancers

  Noir : 50 points, jaune : 30 points, rouge : 20 points, bleu : 10 points. Hors de la cible : 0 points.

  Par exemple, le score du lancer dans l'illustration est de 40 points.



Licence informatique

Gestion des événements

“Écouteurs” d'interactions utilisateur

La programmation événementielle

- Application textuelle
  1. Affichage d'une question et attente.
  2. Saisie clavier par l'utilisateur et validation.
  3. Traitement et continuation du programme.

≠

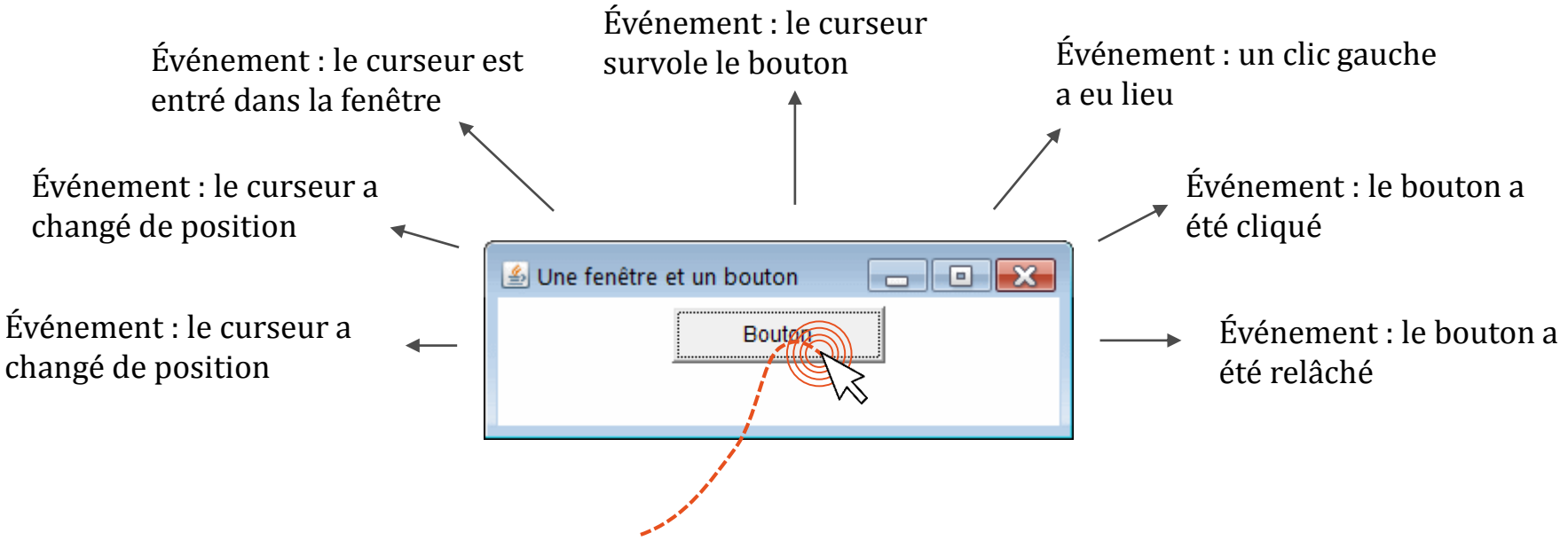
- Application graphique
  1. Affichage de multiples composants (*widgets*) et attente d'un événement.
  2. Clic sur un composant par l'utilisateur.
  3. Analyse de l'interaction, traitement et retour à l'état d'attente.

- 1 seule interaction à la fois ! L'algorithme de l'application est très linéaire.

- Interactions multiples. L'algorithme de l'application n'est pas linéaire.

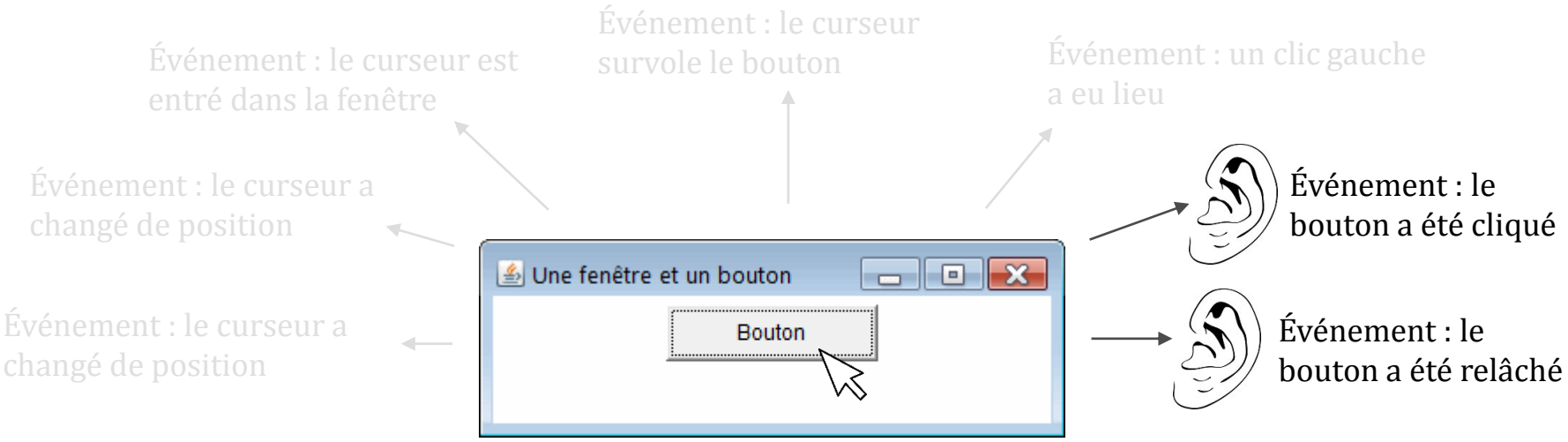
Les événements Java : principe (1)

- Tous les composants génèrent des événements en permanence.
  - Ces événements peuvent donner lieu a des traitements.



Les événements Java : principe (2)

- Mais pour traiter un événement, il faut d'abord le détecter.
  - On utilise des écouteurs d'événements.



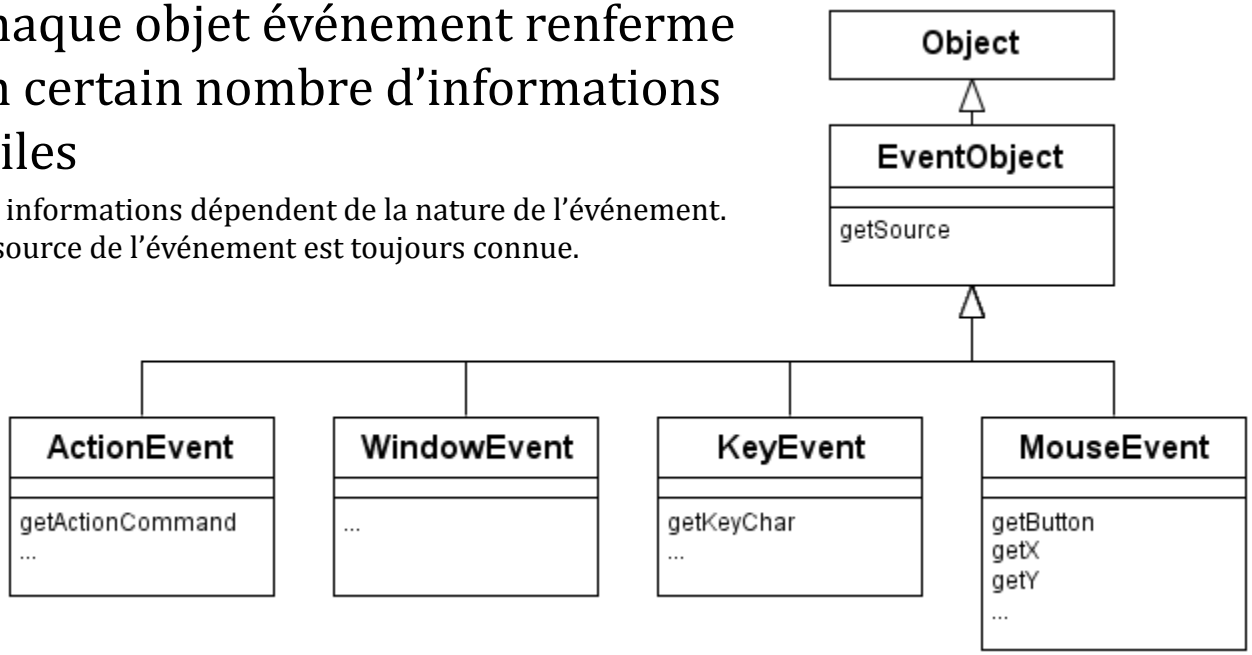
Les événements Java

Les événements sont représentés par des objets qui circulent entre les composants et les écouteurs.

— Ces objets sont instanciés (de manière automatique) à partir de classes d'événements.

— Chaque objet événement renferme un certain nombre d'informations utiles

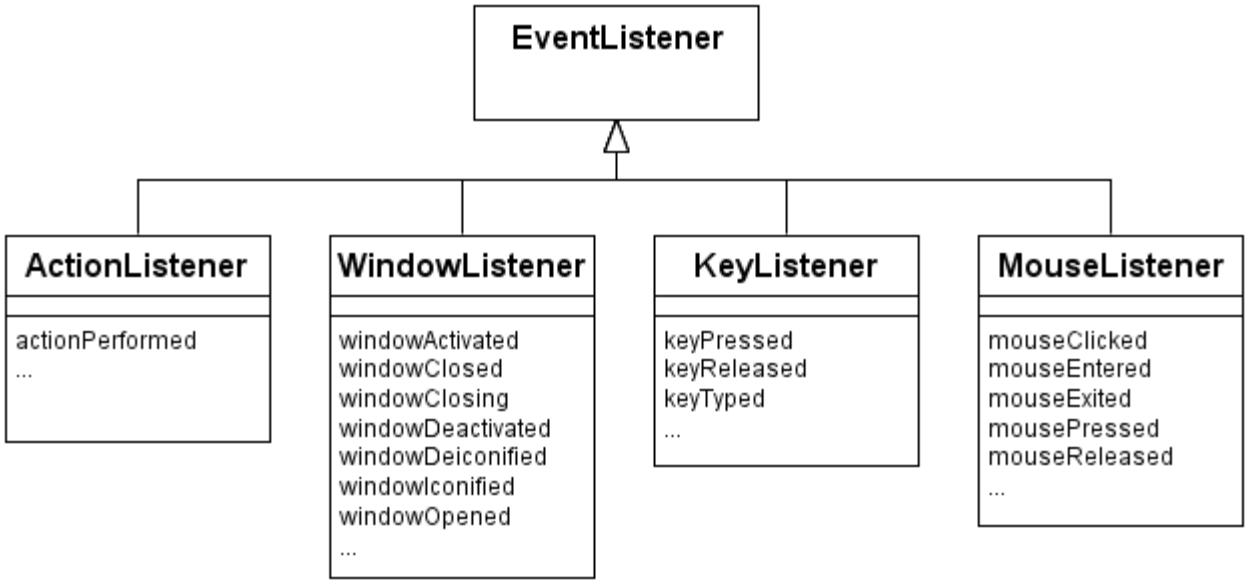
Ces informations dépendent de la nature de l'événement.
La source de l'événement est toujours connue.



Les écouteurs d'événement

- Les écouteurs sont représentés par des *interfaces* Java.
  - Ils dérivent tous de l'interface 'EventListener'
  - Chaque écouteur propose un ensemble de méthodes, chacune reflétant un événement spécifique.

    Exemple: la méthode 'mouseClicked' est appelée lorsqu'un clic a été détecté par l'écouteur. Un objet 'MouseEvent' sera passé en paramètre à cette méthode afin d'en savoir plus sur cet événement (source, coordonnées, etc.)



Les interfaces dans la POO

- Les interfaces sont des classes spécifiques qui définissent des comportements ou des capacités spécifiques dans le but d'enrichir d'autres classes.
  - Elles ne peuvent pas être *instanciées*.
  - Elles définissent un ensemble de méthodes qui correspondent aux nouveaux comportements et/ou capacités.
  - Elles doivent être *implémentées* par une classe.
    Une classe qui implémente une interface propose une implémentation (personnelle) de toutes les méthodes de l'interface.
  - Ne pas confondre héritage et implémentation !

```java
public interface WindowListener extends EventListener {

    /**
     * Invoked when the Window is set to be the
     * active Window.
     * @param e
     */
    public void windowActivated(WindowEvent e);

    /**
     * Invoked when a window has been closed as the
     * result of calling dispose on the window.
     * @param e
     */
    public void windowClosed(WindowEvent e);


    /**
     * Invoked when the user attempts to close the
     *  window from the window's system menu.
     */
    public void More windowClosing(WindowEvent e);

    ...

}
```

Exemple de définition d'une interface.

L'interface est une classe qui ne peut pas être implémentée. On utilise le mot clé 'interface'.

Par convention, on lui donne un nom de comportement ou de capacité.

Elle définit des méthodes qui devront être implémentées par d'autres classes. Aucun code de méthode n'est donc fourni dans l'interface.

Dans cet exemple, les méthodes correspondent à des événements liés à la fenêtre. Chacune reçoit en paramètre un objet événement qui contiendra (à l'appel) des informations relatives à l'événement déclencheur.

```java
public class Evénements implements WindowListener {

    public static void main(String[] args) {

        Frame f = new Frame();
        Evénements e = new Evénements();

        f.addWindowListener(e);

        f.setTitle("Ma première application graphique");
        f.setSize(400,300);
        f.setVisible(true);

    }

    public void windowOpened(WindowEvent e) {
        System.out.println("Fenêtre ouverte");

    }

    public void windowClosing(WindowEvent e) {
        // TODO Auto-generated method stub

    }

    public void windowClosed(WindowEvent e) {
        // TODO Auto-generated method stub

    }
```

Exemple d'utilisation d'une interface.

Le mot clé 'implements' permet d'indiquer qu'une classe implémente une interface.

Une implémentation (même vide) doit être proposée pour chaque méthode de l'interface.

L'association d'un écouter à un composant s'effectue en appelant une méthode spécifique (ex. 'addWindowListener' pour un 'WindowListener').

- ❑ Vérifier que la chaîne des événements est fonctionnelle.

Mise en place

- La mise en place d'une chaîne de traitement des événements s'effectue toujours de la même manière :
  1. Création d'un écouteur (*listener*) en implémentant l'interface adéquate
  2. Appariement de l'émetteur (composant) et de l'écouteur grâce à la méthode de souscription adéquate.

Checklist

La mise en place d'un écouteur s'effectue toujours de cette manière :

X.addYListener(Z)

- ☑ X est-il un composant qui émet des événements de type YEvent ?

  Exemples : MouseEvent, WindowEvent, etc.

- ☑ Z implémente-t-il l'interface YListener ?

  Exemples: MouseListener, WindowListener, etc.

```java
public class Evénements implements WindowListener {

    public static void main(String[] args) {

        Frame f = new Frame();
        Evénements e = new Evénements();

        f.addWindowListener(e);

        f.setTitle("Ma première application graphique");
        f.setSize(400,300);
        f.setVisible(true);

    }

    public void windowOpened(WindowEvent e) {
        System.out.println("Fenêtre ouverte");

    }

    public void windowClosing(WindowEvent e) {
        // TODO Auto-generated method stub

    }

    public void windowClosed(WindowEvent e) {
        // TODO Auto-generated method stub

    }

...
```

Checklist

Dans notre exemple, X=Z

- ☑ L'objet f est une Frame et ce composant produit des événements *WindowEvent*.
- ☑ L'objet e est un écouteur d'événements *WindowEvent* car la classe Evénements implémente bien l'interface *WindowListener*.

- ❑ Que se passe-t-il si l'une des deux conditions n'est pas remplie ?

Exercices

Exercice c2

— À partir de l'exercice précédent, ajouter la gestion événementielle nécessaire pour fermer l'application.

On s'intéressera à la classe 'WindowListener' et à la méthode de classe System.exit()

Exercice c3

— Toujours à partir de l'exercice c1, l'utilisateur doit maintenant cliquer pour lancer une fléchette. Celle-ci n'est pas affichée mais un score est calculé et affiché dans la console après chaque lancer.

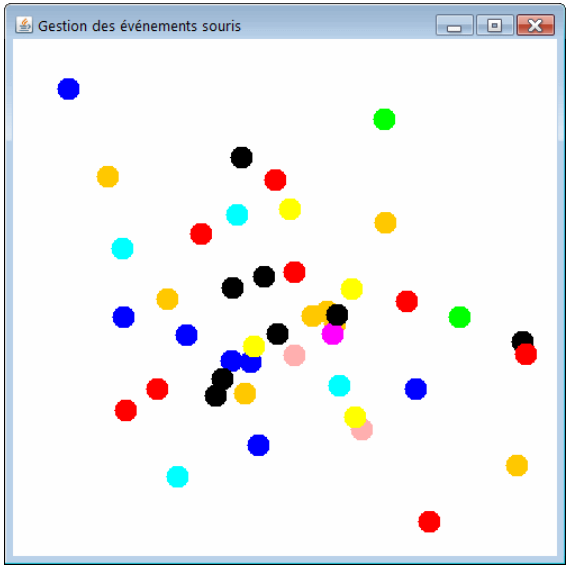
On s'intéressera à la classe 'MouseListener' pour gérer les interactions utilisateur avec la souris.

Exercices

- Exercice c4
  - Dans un cadre carré, l'utilisateur clique pour faire apparaître des disques de couleur.

    On pourra utiliser la classe Point pour stocker la liste des coordonnées des disques dans le cadre.

    La méthode 'repaint' d'un Canvas permet de le redessiner lorsqu'il a été modifié.
  - (optionnel) Désormais, si l'utilisateur clique sur un disque existant, celui-ci s'efface.



On devra associer chaque disque à une couleur.

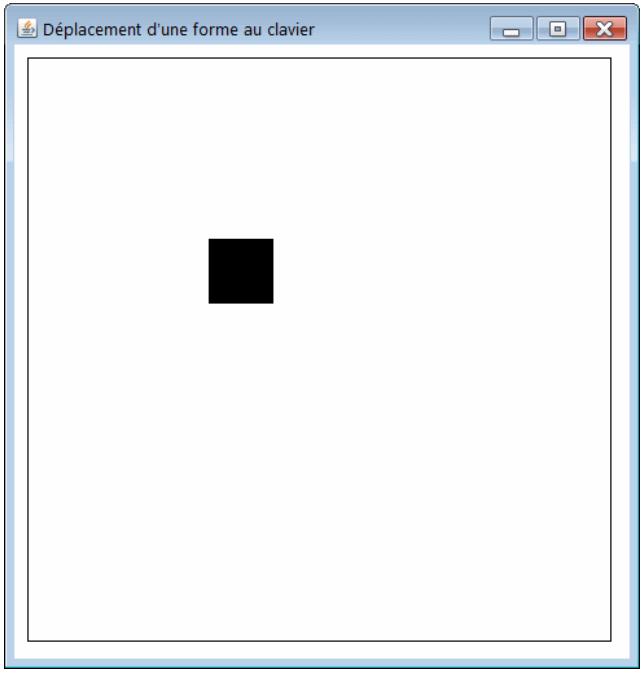
Exercices

Exercice c5

— Dans un Canvas, l'utilisateur déplace un carré noir plein de 50x50 en utilisant les flèches du clavier.

On remarquera la différence entre les méthodes 'KeyPressed' et 'KeyTyped' de la classe 'KeyListener'.

— (optionnel) Le carré ne doit pas pouvoir sortir d'une enceinte délimitée par un carré noir.



Exercices

Exercice c6 (optionnel)

— Dans cet exercice, on présente dans la même fenêtre 3 boutons et un Canvas dans lequel est affiché un disque.

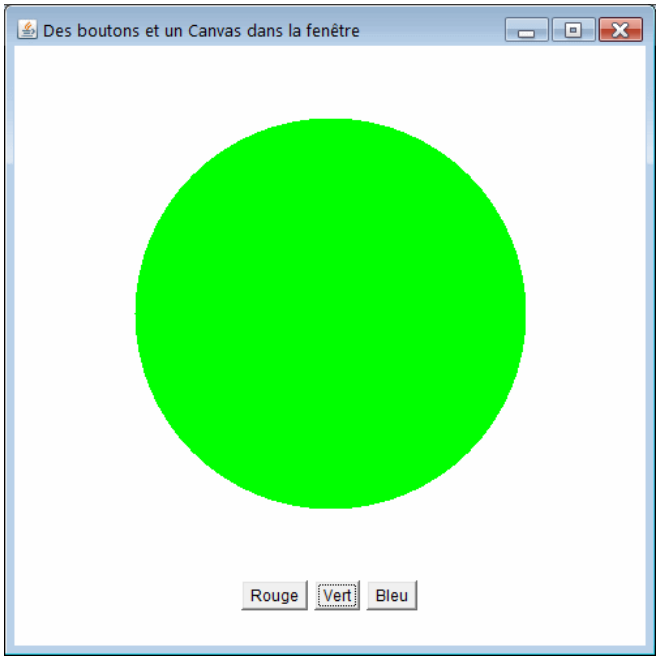
On utilisera la classe 'FlowLayout' pour organiser les composants dans la fenêtre

On prendra soin de surcharger la méthode 'getPreferredSize' du Canvas pour en définir la taille.

— En cliquant sur un bouton, la couleur du disque change en conséquence.

On s'intéressera à la classe 'ActionListener' pour écouter les actions sur les composants graphiques.

Les boutons doivent être identifiés par leur nom (attribut 'name').



Références

- La classe Canvas
  http://docs.oracle.com/javase/7/docs/api/java/awt/Canvas.html
- La documentation de la classe Graphics
  http://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html
- Les événements Java
  http://docs.oracle.com/javase/tutorial/uiswing/events/
- Les dispositions graphiques (layout) avec AWT
  http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html