



Java

Introduction à Swing

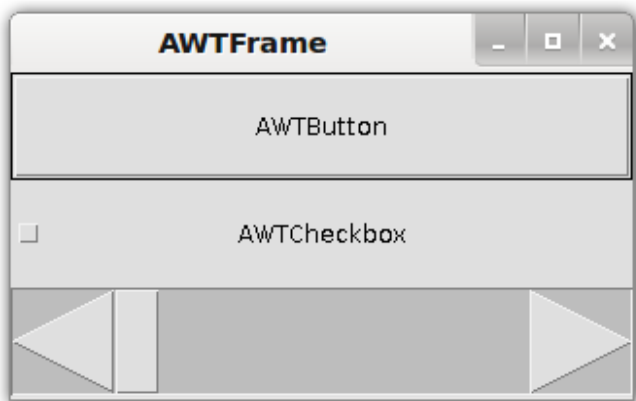
Présentation de Swing

Présentation de Swing

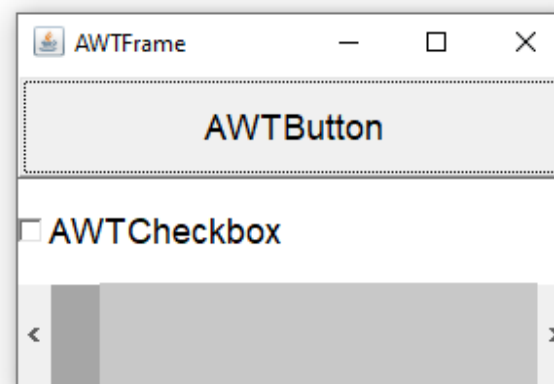
AWT (Abstract Window Toolkit)

- AWT (java.awt) est le package historique permettant de faire des interfaces graphiques en Java
- L'inconvénient majeur de AWT est qu'il utilise des composants graphiques natifs du système. Leur comportement et leur apparence peut donc changer selon la plateforme :

Ubuntu ♥

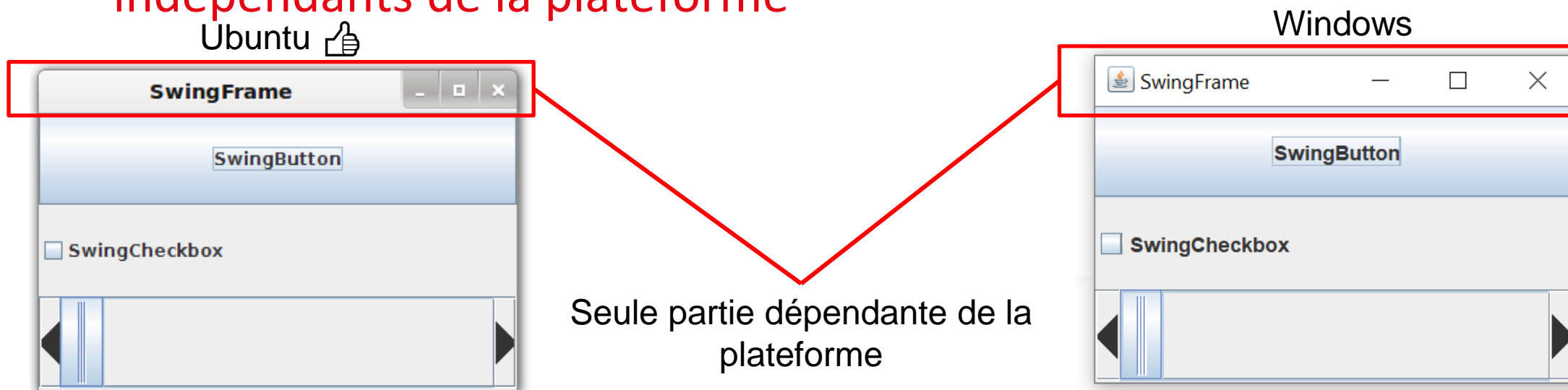


Windows



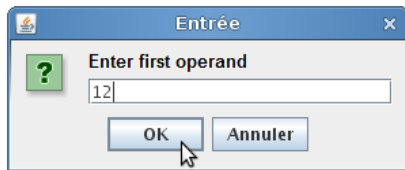
Swing

- Swing (package javax.swing) contient des composants entièrement développés en Java
- A part les fenêtres (qui sont toujours fournies par le système), l'apparence et le comportement (appelé ***look-and-feel***) sont indépendants de la plateforme

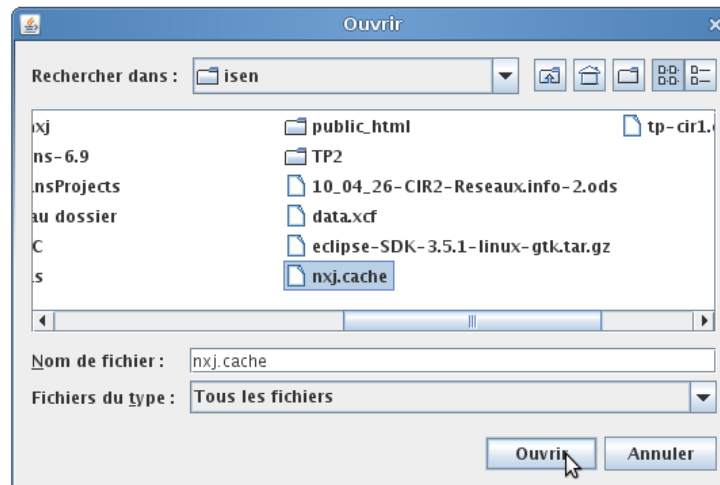


Avantages de Swing

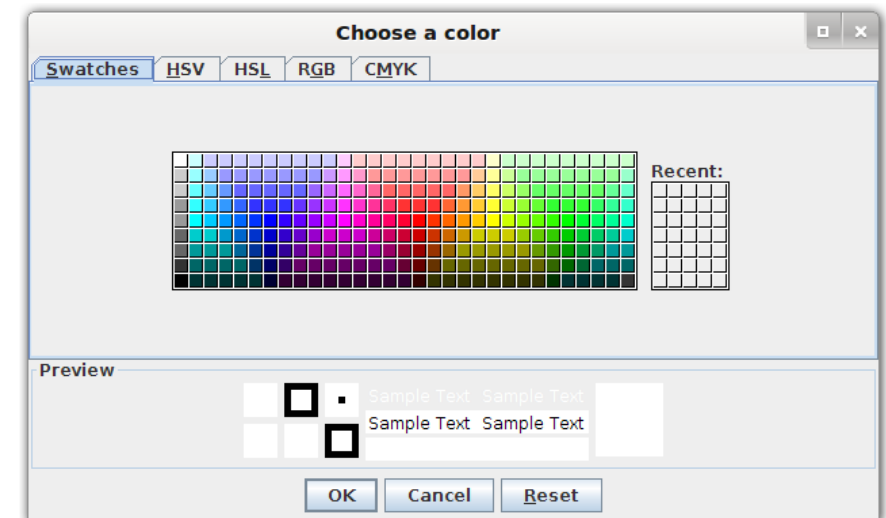
- Outre son caractère multiplateforme, Swing possède les avantages suivants :
 - Les composants Swing sont plus riches en fonctionnalités. Swing propose notamment des composants « clé en main » pour saisir du texte, choisir des couleurs, des fichiers, etc.



JOptionPane



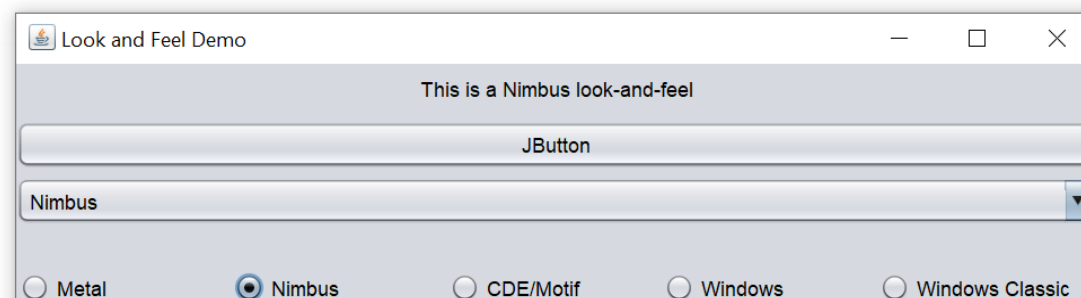
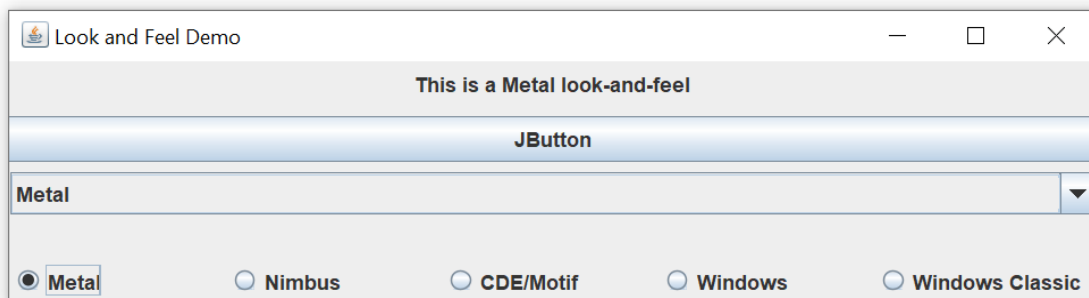
JFileChooser



JColorChooser

Avantages de Swing

- Outre son caractère multiplateforme, Swing possède les avantages suivants :
 - Les composants Swing sont plus riches en fonctionnalités. Swing propose notamment des composants « clé en main » pour saisir du texte, choisir des couleurs, des fichiers, etc.
 - Swing possède un système appelé **pluggable look and feel** (plaf) permettant d'appliquer un thème à une application, sans en modifier le code.



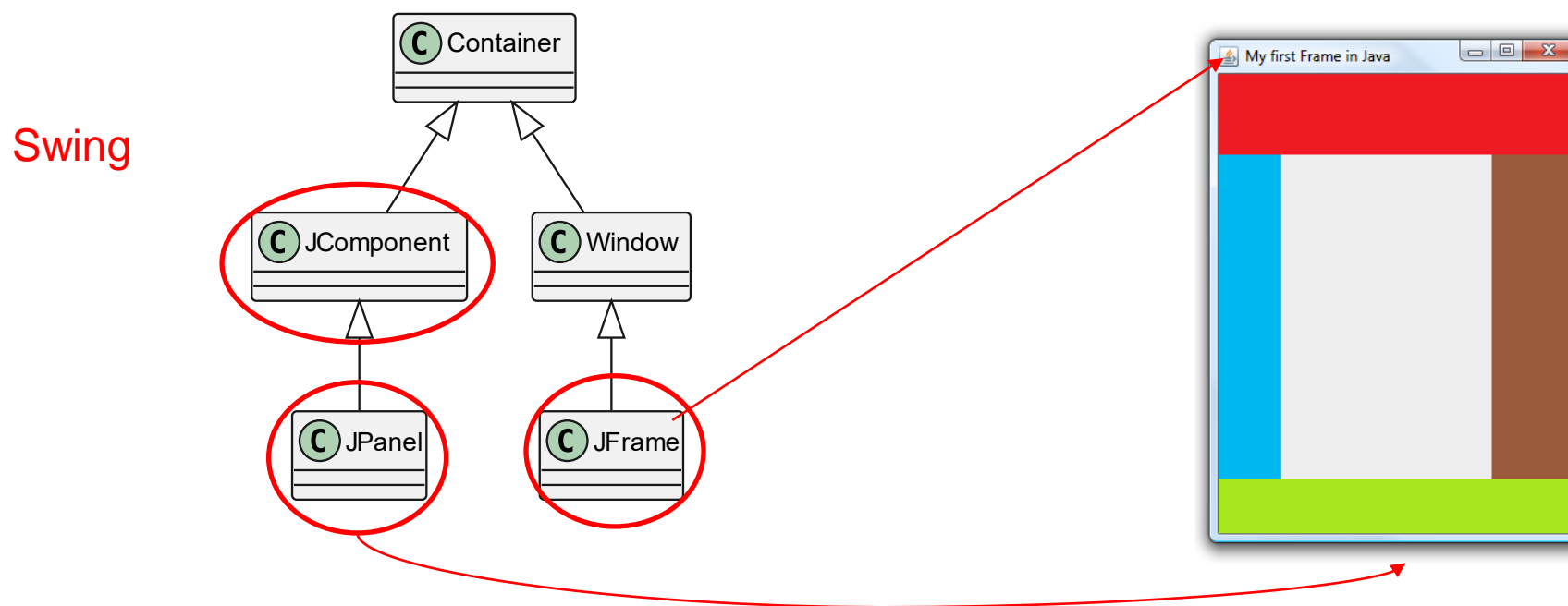
AWT reste partiellement utilisé

- Même si nous allons faire nos interfaces graphiques avec Swing, nous allons continuer à utiliser certaines interfaces de AWT (et les classes implémentant ces interfaces) pour :
 - Organiser les composants dans l'interface graphique (interface `java.awt.LayoutManager`)
 - Faire communiquer les composants entre eux par l'envoi d'événements(`package java.awt.event`)

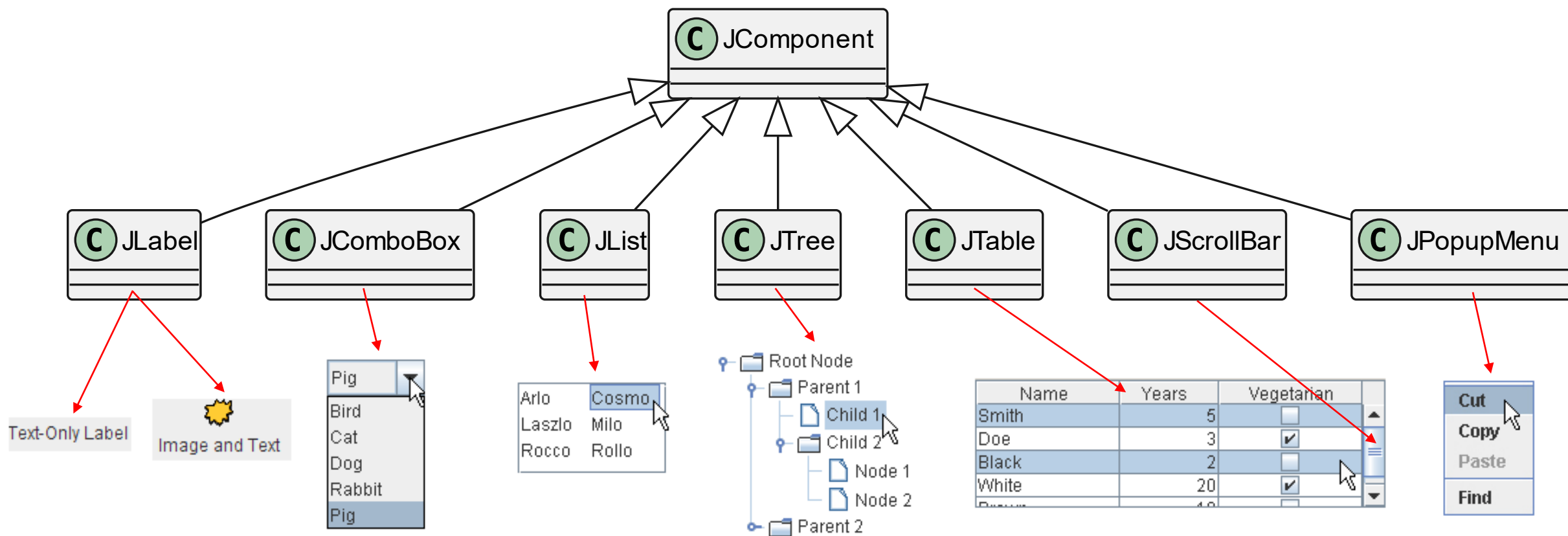
Les composants Swing

Les composants Swing

- A part **JFrame** (composant correspondant à la fenêtre), tous les composants Swing héritent de **JComponent**
- Leur nom commencent par « J »



Quelques exemples de composants Swing



Votre première fenêtre

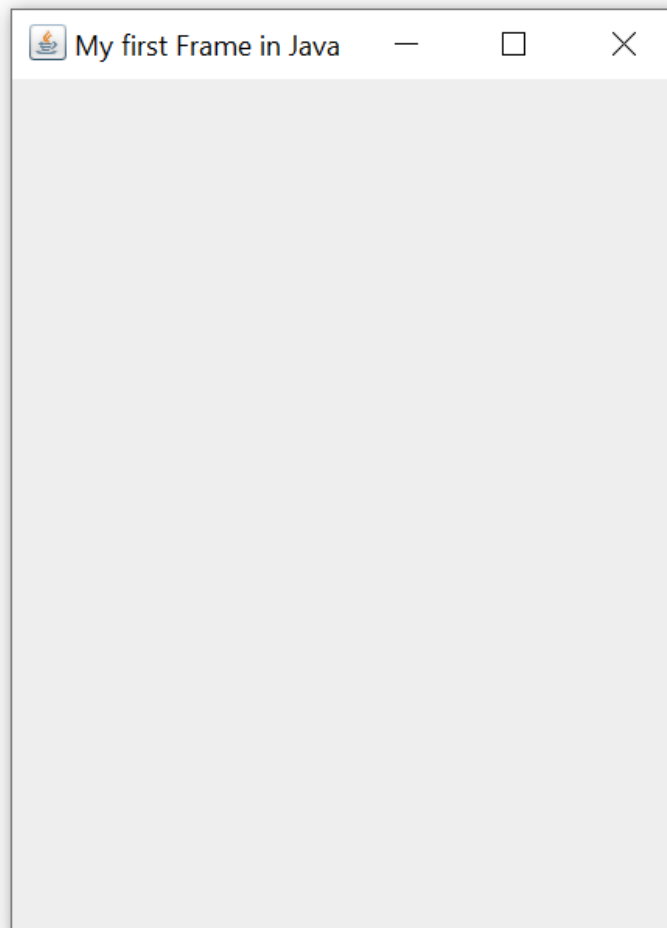
Etapes de création d'une fenêtre personnalisée

- Pour créer votre propre fenêtre :
 1. Créez une classe qui hérite de javax.swing.JFrame
- Dans le constructeur de votre classe :
 2. Changez le titre de la fenêtre
 3. Changez les dimensions de la fenêtre
 4. Changez le comportement de la fenêtre au clic sur la croix (fermeture)
 5. Enfin, rendez la fenêtre visible

Code de création d'une fenêtre personnalisée

```
import javax.swing.JFrame;
public class MyFrame extends JFrame { 1
    public MyFrame() {
        super("My first Frame in Java"); 2
        this.setSize(300,200); 3
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); 4
        this.setVisible(true); 5
    }
    public static void main(String[] args) {
        MyFrame myFrame = new MyFrame
    }
}
```

Code de création d'une fenêtre personnalisée



Les gestionnaires de placement

Les gestionnaires de placement

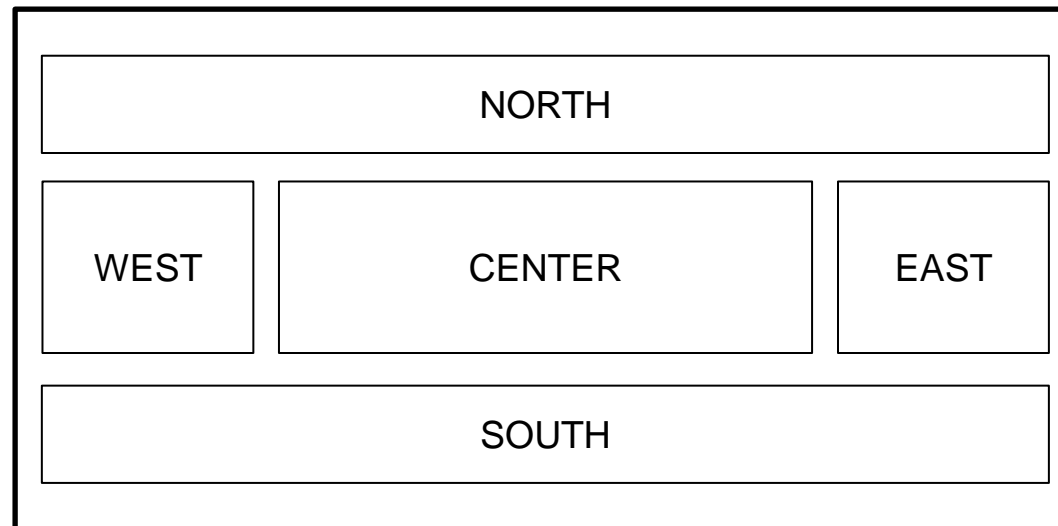
- Tous les composants Swing sont des conteneurs : un composant Swing peut en contenir d'autres.
- Ceci permet de structurer une interface en groupes (par exemple un formulaire, une barre d'outils...).
- Les composants à l'intérieur d'un conteneur sont organisés par un **gestionnaire de placement** (*Layout Manager* en anglais).
- Le gestionnaire recalcule la position et la taille des composants à chaque redimensionnement du conteneur.
- Techniquement, un gestionnaire de placement est une classe qui implémente l'interface `java.awt.LayoutManager`

Gestionnaires de placement vus dans ce cours

- Dans ce cours, nous allons focaliser sur les gestionnaires de placement suivants, qui sont très simple d'utilisation :
 - BorderLayout : organisation en 5 zones (1 centrale + 4 points cardinaux)
 - FlowLayout : organisation en ligne
 - GridLayout : organisation en grille
- Une combinaison de ces 3 gestionnaires répondra à la plupart de vos besoins
- Si vous voulez utiliser des stratégies de placement plus évoluées (contraintes d'alignement, positionnement relatif, etc.), vous pouvez consulter le [tutoriel d'Oracle](#) sur les *Layout Managers* et/ou utiliser un assistant de conception graphique d'un IDE.

BorderLayout

- Classe : `java.awt.BorderLayout`
- C'est le gestionnaire de placement par défaut pour une `JFrame`
- Il positionne les composants dans 5 zones :



- Le maximum de place est attribué à la zone centrale.

BorderLayout

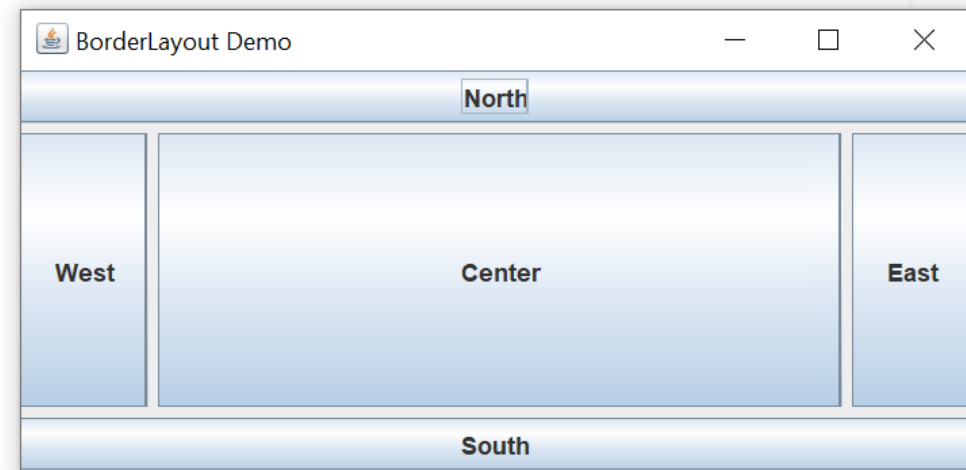
```
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.BorderLayout;
public class MyFrame extends JFrame {
    public MyFrame() {
        super("BorderLayout Demo");
        this.setSize(300, 400);
        this.setLayout new BorderLayout (5, 5);
        this.add(new JButton("North"), BorderLayout.NORTH);
        this.add(new JButton("South"), BorderLayout.SOUTH);
        this.add(new JButton("East"), BorderLayout.EAST);
        this.add(new JButton("West"), BorderLayout.WEST);
        this.add(new JButton("Center"), BorderLayout.CENTER);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    ...
}
```

Espace (gap) entre les composants sur x et y



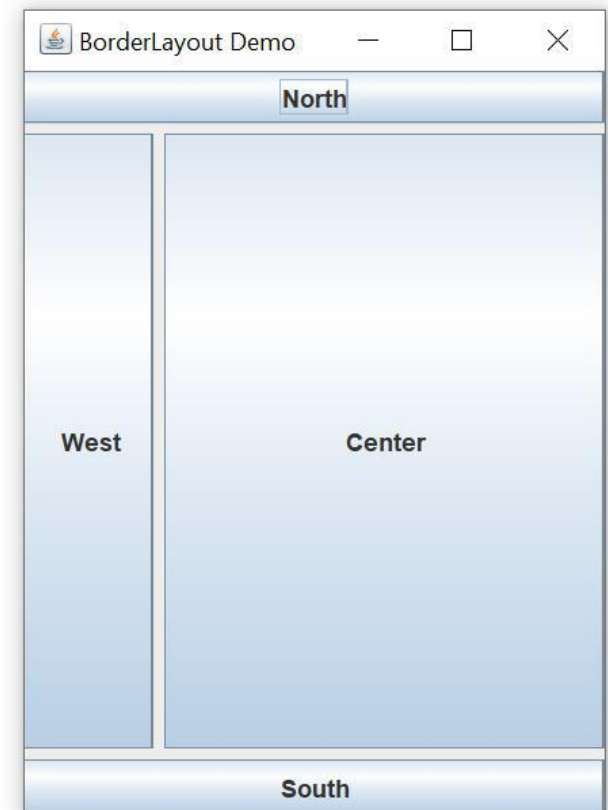
BorderLayout



BorderLayout

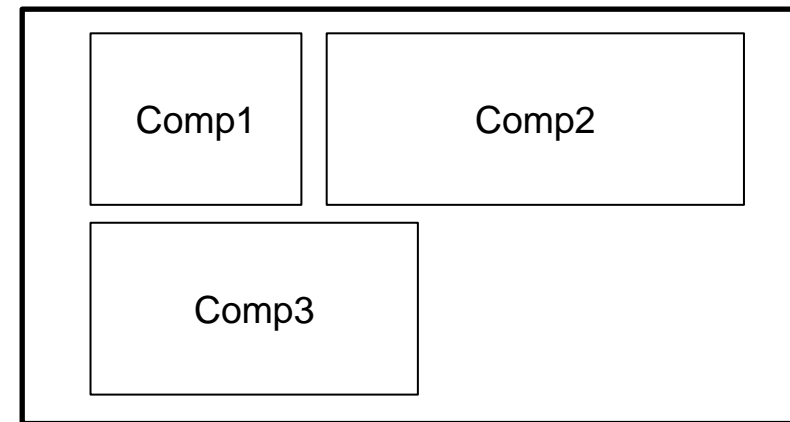
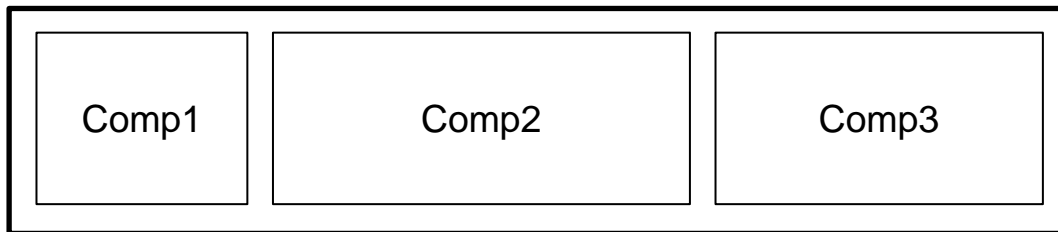
```
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.BorderLayout;
public class MyFrame extends JFrame {
    public MyFrame() {
        super("BorderLayout Demo");
        this.setSize(300, 400);
        this.setLayout new BorderLayout (5, 5);
        this.add(new JButton("North"), BorderLayout.NORTH);
        this.add(new JButton("South"), BorderLayout.SOUTH);
        this.add(new JButton("East"), BorderLayout.EAST);
        this.add(new JButton("West"), BorderLayout.WEST);
        this.add(new JButton("Center"), BorderLayout.CENTER);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    ...
}
```



FlowLayout

- Classe : `java.awt.FlowLayout`
- C'est le gestionnaire de placement par défaut pour un `JPanel`
- Il positionne les composants sur une ligne, et démarre une nouvelle ligne s'il n'y a pas assez de place.



- Les composants sont dimensionnés à leur taille préférée.
- Par défaut, les composants sont centrés.

FlowLayout

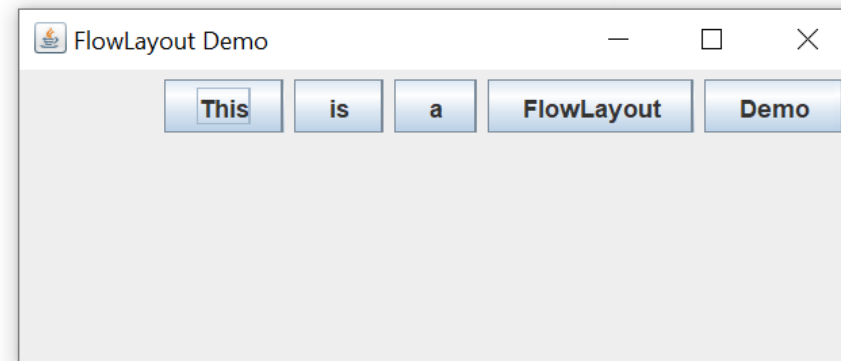
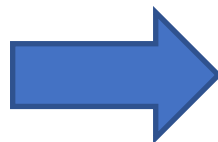
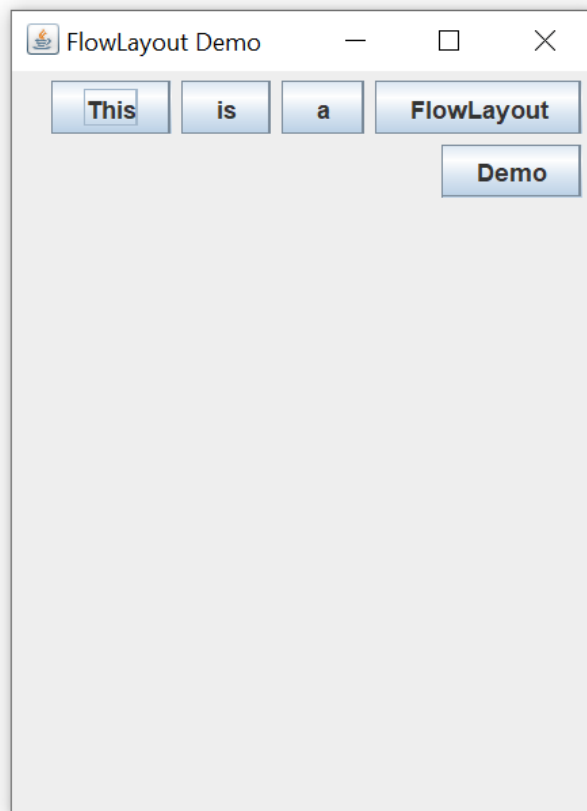
```
...
import java.awt.FlowLayout;
public class MyFrame extends JFrame {
    public MyFrame() {
        super("FlowLayout Demo");
        this.setSize(300, 400);

        this.setLayout(new FlowLayout(FlowLayout.RIGHT, 5, 5));
        this.add(new JButton ("This"));
        this.add(new JButton ("is"));
        this.add(new JButton ("a"));
        this.add(new JButton ("FlowLayout"));
        this.add(new JButton ("Demo"));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    ...
}
```

Espace (*gap*) entre les composants sur x et y

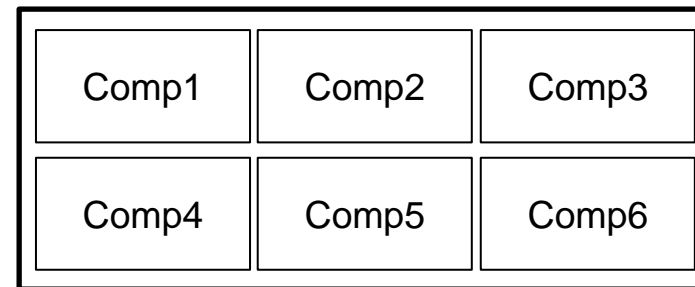
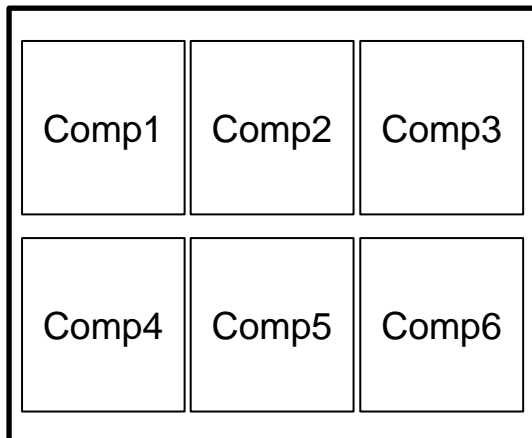
Alignement

FlowLayout



GridLayout

- Classe : `java.awt.GridLayout`
- Ce gestionnaire organise les composants en grille
- La structure de la grille (nombre de lignes et de colonnes) est figée à la construction.
- Toutes les cases de la grille ont les mêmes dimensions, calculées en divisant uniformément l'espace du conteneur.
- Chaque composant prend tout l'espace disponible dans sa case.




GridLayout

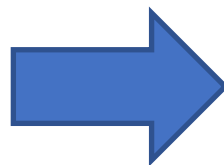
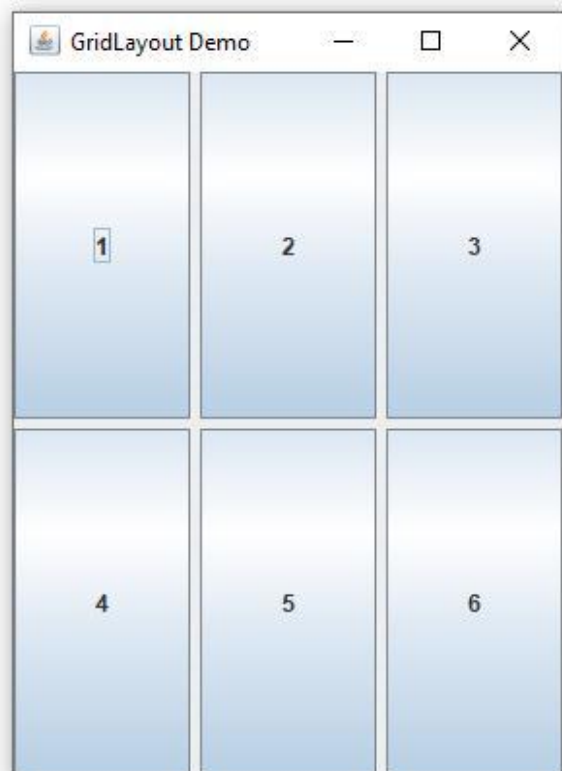
```
import java.awt GridLayout
public class MyFrame extends JFrame {
    public MyFrame() {
        super("GridLayout Demo");
        this.setSize(300, 400);
        this.setLayout (new GridLayout (2, 3, 5, 5);
        this.add(new JButton ("1"));
        this.add(new JButton ("2"));
        this.add(new JButton ("3"));
        this.add(new JButton ("4"));
        this.add(new JButton ("5"));
        this.add(new JButton ("6"));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    ...
}
```

Dimensions de la grille

Espace entre les composants

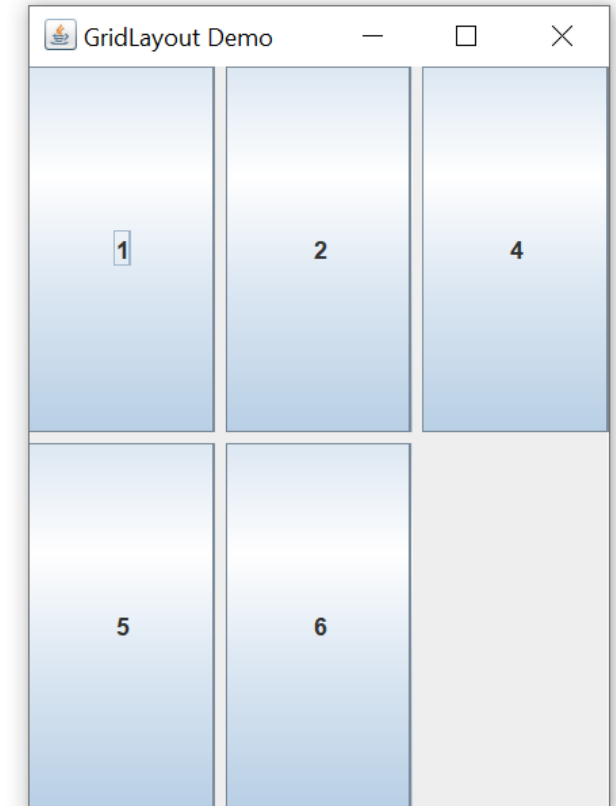


GridLayout



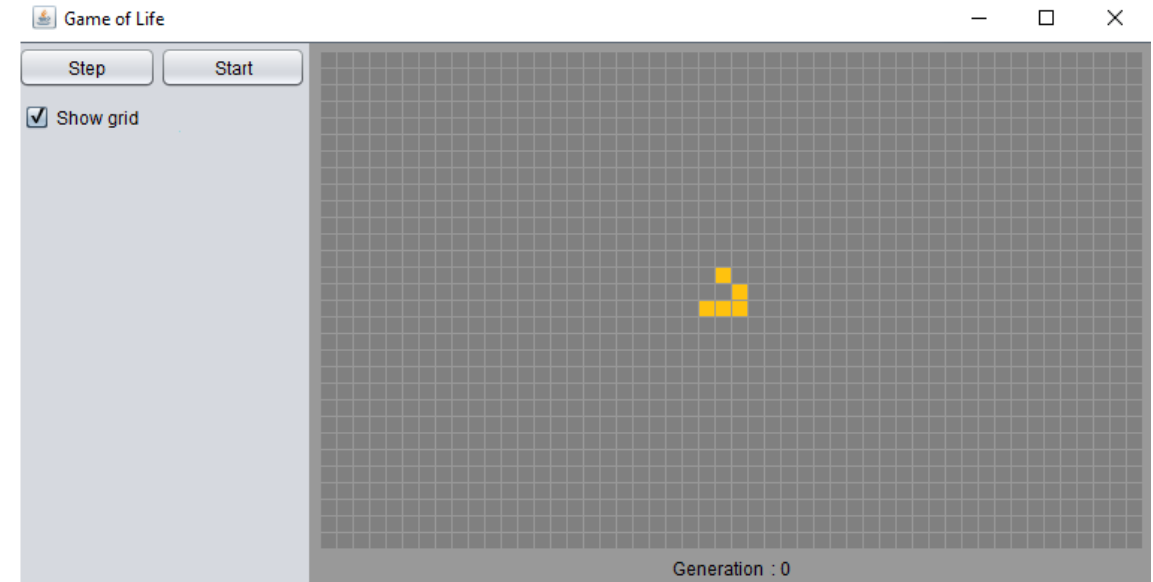
GridLayout

```
import java.awt GridLayout
public class MyFrame extends JFrame {
    public MyFrame() {
        super("GridLayout Demo");
        this.setSize(300, 400);
        this.setLayout (new GridLayout (2, 3, 5, 5));
        this.add(new JButton ("1"));
        this.add(new JButton ("2"));
        // this.add(new JButton ("3"));
        this.add(new JButton ("4"));
        this.add(new JButton ("5"));
        this.add(new JButton ("6"));
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    ...
}
```



Un cas pratique

- Sans écrire une ligne de code, pouvez vous expliquer comment vous pourriez combiner les 3 gestionnaires vus précédemment pour obtenir l'interface graphique ci contre ?



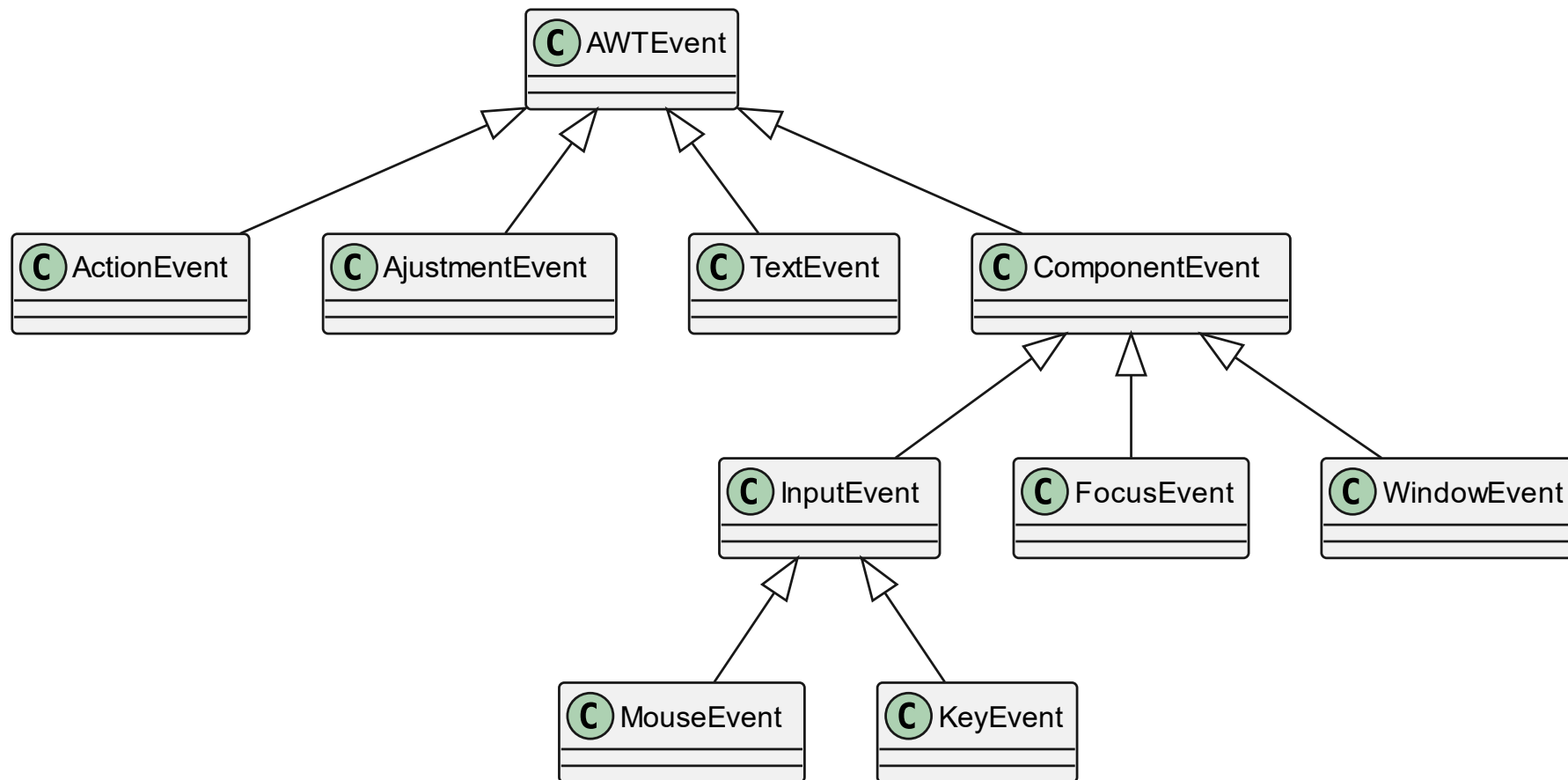
- Méthodologie : vous pouvez inclure des **JPanel** les un dans les autres, avec un gestionnaire différent.
- Dessinez les contours des différents **JPanel** et indiquez le gestionnaire de placement de chacun.

Les événements

Évènement

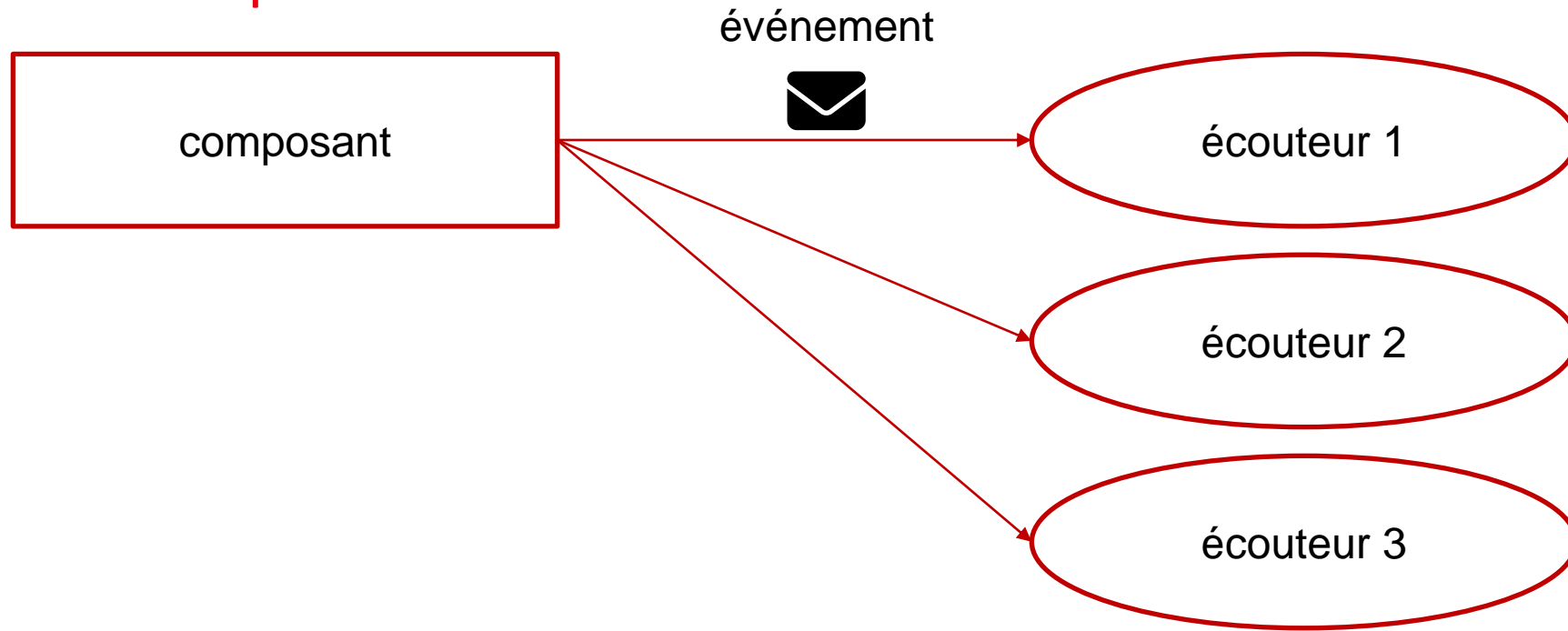
- Un événement est un message décrivant un changement d'état d'un objet.
- Dans le cas spécifique des interfaces graphiques, un événement modélise une action de l'utilisateur sur un composant, par exemple :
 - Le redimensionnement d'une fenêtre
 - La sélection d'un élément dans un menu déroulant
 - Le clic sur un bouton ...
- Les événements les plus courants font partie du package `java.awt.event`. Quelques événements spécifiques à Swing sont également présents dans le package `javax.swing.event`

Quelques événements du package `java.awt`



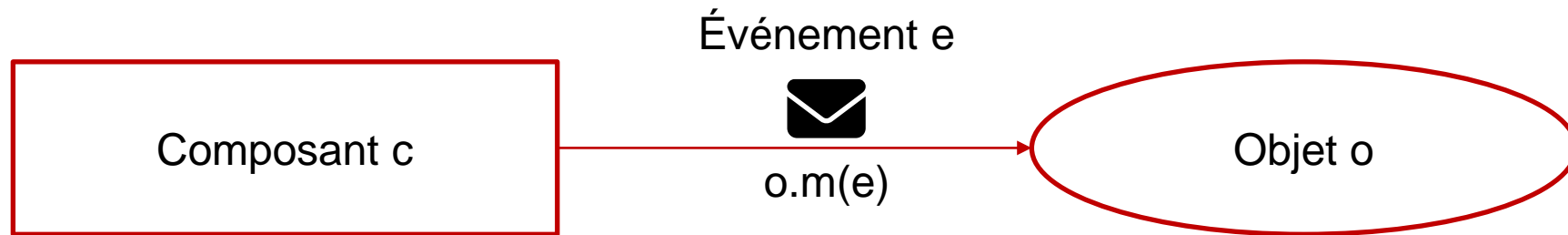
Écouteur (listener)

- Un ou plusieurs objet(s) peu(ven)t écouter les événements émis par un composant.



Écouteur (listener)

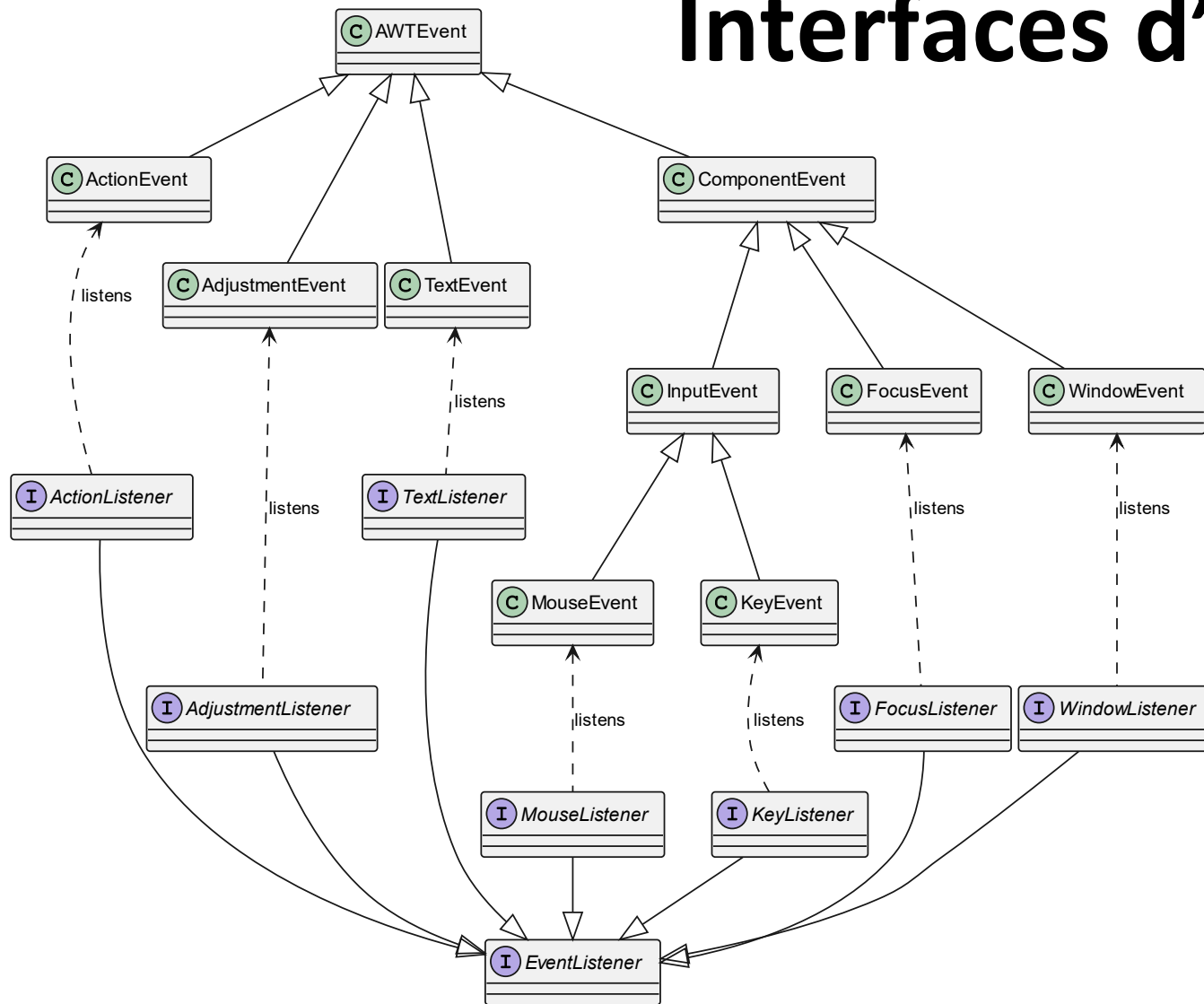
- Lorsqu'un composant émet un événement à destination d'un objet, il appelle une méthode `m()` avec cet événement en paramètre



- Exemple clic bouton



Interfaces d'écoute



- Techniquement, un objet indique qu'il écoute un événement en implémentant une interface.
- À chaque type d'événement XEvent est associé une interface XListener

Installer un écouteur

- Pour faire en sorte qu'un objet écoute les `XEvent` émis par un composant, il faut :
 - Définir une classe qui implémente `XListener`
 - Ajouter une instance de cette classe à liste des écouteurs du composant, avec la méthode `addXListener()`
- Exemple : Lorsqu'on clique sur un `JButton` , il émet un `ActionEvent` . Pour réagir à ce clic, il faut :
 - Définir une classe qui implémente `ActionListener`
 - Ajouter une instance de cette classe à liste des écouteurs du composant, avec la méthode `addActionListener ()`

Application : écoute des clics sur un bouton

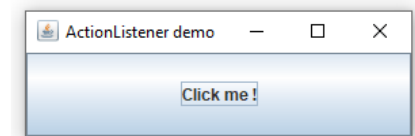
- On définit une classe `MyButtonListener` qui implémente l'interface `ActionListener`
- `ActionListener` ne contient qu'une méthode, `actionPerformed()`, qui possède un événement de type `ActionEvent` en paramètre

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class MyButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Click!");
    }
}
```

Application : écoute des clics sur un bouton

- On ajoute une instance de `MyButtonListener` à la liste des `ActionListener` du bouton

```
public class MyFrame extends JFrame{
    public MyFrame() {
        super("ActionListener demo");
        this.setSize(300, 100);
        JButton button = new JButton ("Click me!");
        this.add(button);
        MyButtonListener buttonListener = new MyButtonListener();
        button.addActionListener(buttonListener);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    ...
}
```

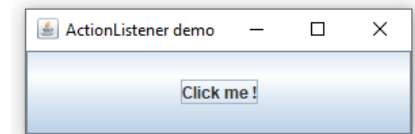


Sortie standard

Click !

Auto-écoute

- Un composant peut tout à fait écouter ses propres événements.
- Pour cela, il faut qu'il implémente les interfaces associées et s'ajouter lui même en tant qu'écouteur.



Sortie standard

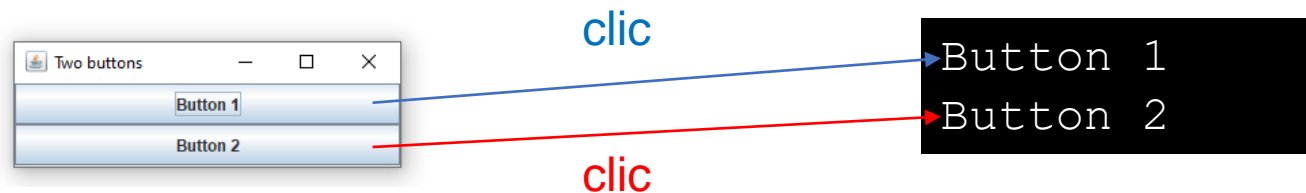
Click !

```
public class MyFrame extends JFrame implements ActionListener{
    public MyFrame() {
        ...
        MyButtonListener buttonListener = new MyButtonListener();
        button.addActionListener(this);
        ...
    }
    public void actionPerformed(ActionEvent e) {
        System.out.println("Click me!");
    }
    ...
}
```


Comment distinguer plusieurs composants qui émettent le même type d'événement ?

■ Illustration du problème :

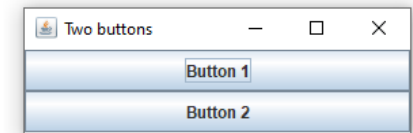
- Supposons qu'une interface possède deux `JButton`. On souhaite afficher « Button 1 » lorsqu'on clique sur le 1^{er} bouton, et « Button 2 » lorsqu'on clique sur le 2^{ème}



- Comment faire ceci sachant que les boutons émettent tous deux des `ActionEvent`

Base de code à compléter

```
public class MyFrame extends JFrame{
    public MyFrame(){
        super("Two buttons");
        this.setSize(300, 100);
        JButton b1 = new JButton ("Button 1");
        JButton b2 = new JButton ("Button 2");
        this.setLayout(new GridLayout(2, 1);
        this.add(b1);
        this.add(b2);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    ...
}
```



Solution 1 : Un écouteur par bouton

```
public class MyFrame extends JFrame{
    public MyFrame(){
        ...
        b1.addActionListener(new Button1Listener);
        b2.addActionListener(new Button2Listener);
    }
    ...
}
```

```
public class Button1Listener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button 1");
    }
}
```

```
public class Button2Listener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button 2");
    }
}
```

Solution 2 : Ecouteur unique + test de l'expéditeur

```
public class MyFrame extends JFrame implements ActionListener{
    private JButton b1;
    private JButton b2;
    public MyFrame(){
        ...
        this.b1 = new JButton("Button 1");
        this.b2 = new JButton ("Button 2");
        ...
        this.b1.addActionListener(this);
        this.b2.addActionListener(this);
    }
    public void
    actionPerformed(ActionEvent e) {
        if (e.getSource() == this.b1){
            System.out.println("Button 1");
        }
        else if (e.getSource() == this.b2){
            System.out.println("Button 2");
        }
    }
}
```

- La méthode `getSource()` retourne l'expéditeur d'un événement
- Il faut garder une trace des expéditeurs potentiels pour pouvoir les tester plus tard