INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

FORMATION

Java

Introduction aux IHM Java

Composants graphiques

et notion d'événements

GUI AWT/Swing

membre de UNIVERSITÉ DE LYON

INSA LYON

Plan du cours

- Concepts
  - Les interfaces graphiques, choix d'un GUI
  - La programmation événementielle

- Réalisations
  - Création d'une fenêtre par étapes
  - La gestion des événements par étapes

Partie 1

INTERFACES GRAPHIQUES GÉNÉRALITÉS

Interface graphique: généralités

- Une interface graphique(GUI) est un outil permettant d'interagir avec un programme informatique.

- Pour une interface vivante et interactive:
  - **Il faut des composants avec lesquels on interagit:**
    - Les classes d'interactions : les *composants (boutons, menus…)*
  - **Il faut des boîtes de rangement:**
    - Les conteneurs et gestionnaires de dispositions *qui contiennent des composants*
  - **Il faut animer l'ensemble:**
    - Les classes d'événements

Le programmeur :
- Met les éléments graphiques en place (cadre, menu, boutons)
- Son travail consiste à élaborer la logique de son application, les algorithmes mis en œuvre.

Interface graphique: GUI Java

- L'API Java propose plusieurs bibliothèques graphiques pour définir ces interfaces

  - **L'AWT** (Abstract Windowing Toolkit): package *java.awt*
  - **Swing** fait parti de Java Foundation Classes (JFC) et s'appuie sur AWT: package *javax.swing*
    - *import javax.swing.*;*
    - *import javax.swing.event.*;*

ou préciser le nom du composant

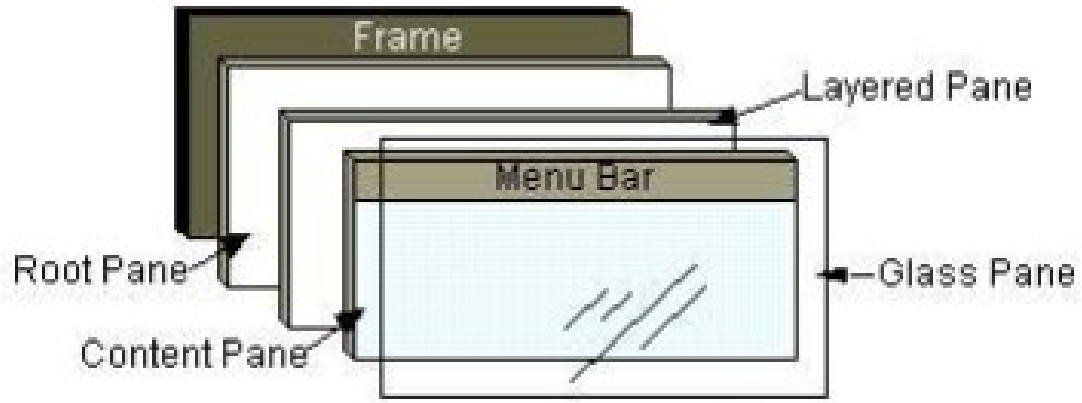
```
import
javax.swing.NomDuComposant
```

  - SWT (Standard Widget Toolkit) bibliothèque graphique libre pour Java
  - D'autres propres à certains environnements (ex: Androïd)

Interface graphique : SWING

- La partie interface d'une application graphique repose sur une hiérarchie de composants graphiques

  - Visuels de cadrage: **J**Frame, **J**Dialog
  - Issus de conteneurs: **Component** (racine des classes)
  - Intégrant des composants appelés Widgets:
    - Boutons
    - Champ de texte
    - Menu
    - Cases à cocher
    - Liste
    - …

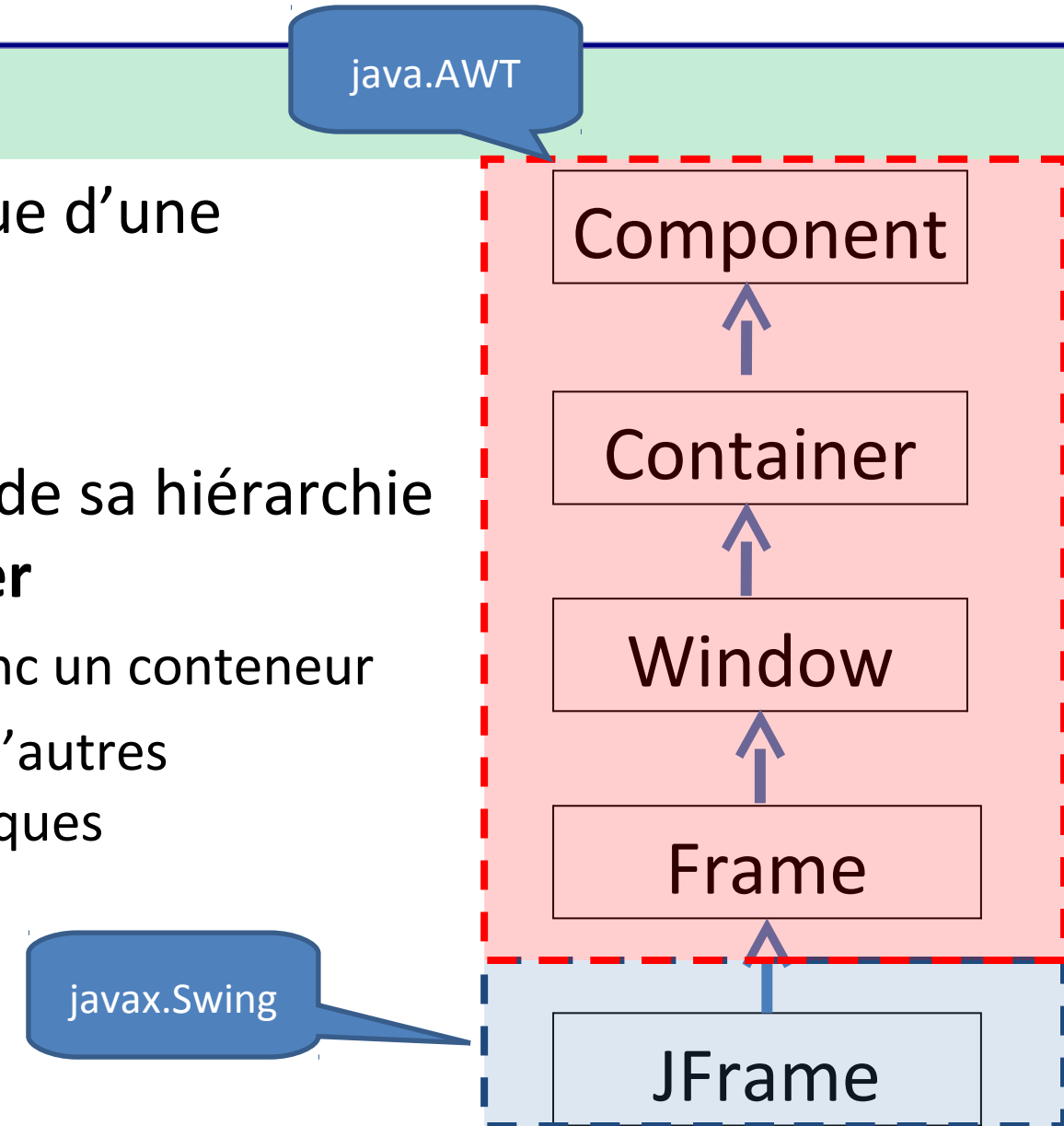


Tout objet de la classe Swing qui a une représentation graphique peut interagir avec l'utilisateur

Le **JFrame** (fenêtre de l'application) est structurée en plusieurs couches superposées

Hiérarchie des classes Swing

- La classe **JFrame**
  - Une fenêtre est issue d'une hiérarchie
  - Au quasi - sommet de sa hiérarchie : la classe **Container**
    - Une fenêtre est donc un conteneur
    - Elle peut contenir d'autres composants graphiques



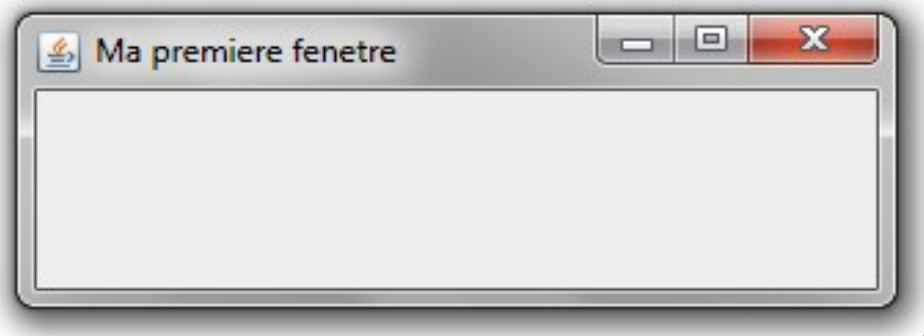
Exemple 1.a: Fenêtre basique

Par la création d'une fenêtre dans le main

Import de la classe JFrame

Main contenant l'instruction de création

```
1  import javax.swing.*;
2
3  class TestFenetreSimple {
4
5      public static void main(String[] args){
6          JFrame fenetre= new JFrame("Ma première fenêtre");
7          fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8          fenetre.setVisible(true);
9          }
10 }
```

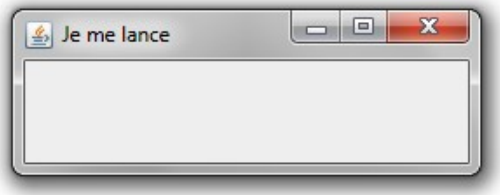


Fichier TestFenetreSimple.java

Exemple 1.b: Fenêtre un peu plus évoluée

Par la création d'une classe fenêtre héritière

Création d'une classe ClasseFenetre héritière de JFrame => réutilisable




```
2   import javax.swing.*;
3
4   public class ClasseFenetre extends JFrame {
5
6       private String nom;
7       public ClasseFenetre(String nom)
8        {
9           super(nom);
10          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11          this.setVisible(true);
12       }
13       public static void main(String[] args)
14        {
15         ClasseFenetre fenetre= new ClasseFenetre("Je me lance");
16        }
17  }
```

Constructeur de la classe -appel du constructeur ancêtre par super()

main simplifié ne contenant que l'appel du constructeur

La fenêtre ne contient ici aucun composant visuel (widget). L'interaction avec elle est très limitée !
La classe contient un **main**, elle peut donc s'exécuter

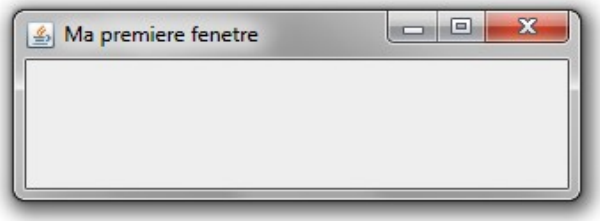
Fichier ClasseFenetre.java

Exemple 1.c: Fenêtre par EDI Jdev

Séparation entre classe graphique et classe principale

```
package essaifenetre;

import ...;

public class ClasseFenetre exte
    public ClasseFenetre() {
        try {
            jbInit();
        } catch (Exception e) {
            e.printStackTrace()
        }
    }

    private void jbInit() throw
        this.getContentPane().s
        this.setSize(new Dimens
        this.setTitle( "Ma prem
    }
}
```

```
package essaifenetre;

import ...;

public class ClassePrincipale {
    public ClassePrincipale() {
        JFrame frame = new ClasseFenetre();
        Dimension screenSize = Toolkit.getDefaultToolkit().ge
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation( ( screenSize.width - frameSize.width ) / 2, ( screenSize.height - frameSize.height ) / 2 );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookA
        } catch (Exception e) {
            e.printStackTrace();
        }
        new ClassePrincipale();
    }
}
```



Démo jdev

La classe **ClasseFenetre** ne contient pas de main.
L'exécution passe par la classe **ClassePrincipale**

Deux fichiers
ClasseFenetre.java
et ClassePrincipale.main

Composants visuels usuels Swing

JButton

JCheckBox

JComboBox

JList

JMenu

JRadioButton

JSlider

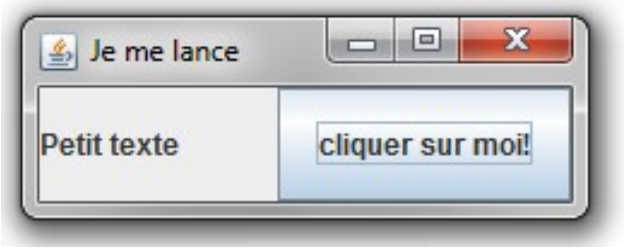
JSpinner

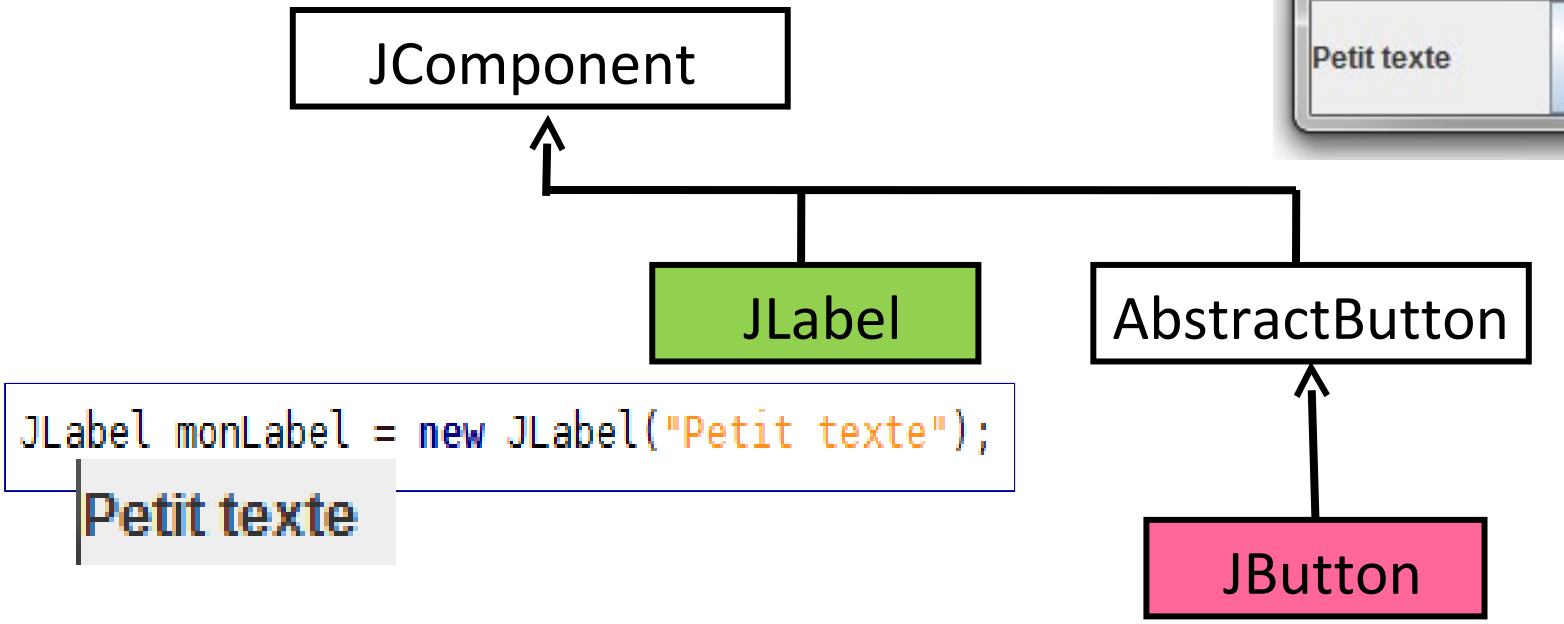
JTextField

JPasswordField

Composants usuels

- Ex de composants atomiques: **JLabel**, **JButton**
  - Non destinés a priori à en contenir d'autres
  - Permet d'interagir avec l'utilisateur






```
JLabel monLabel = new JLabel("Petit texte");
```

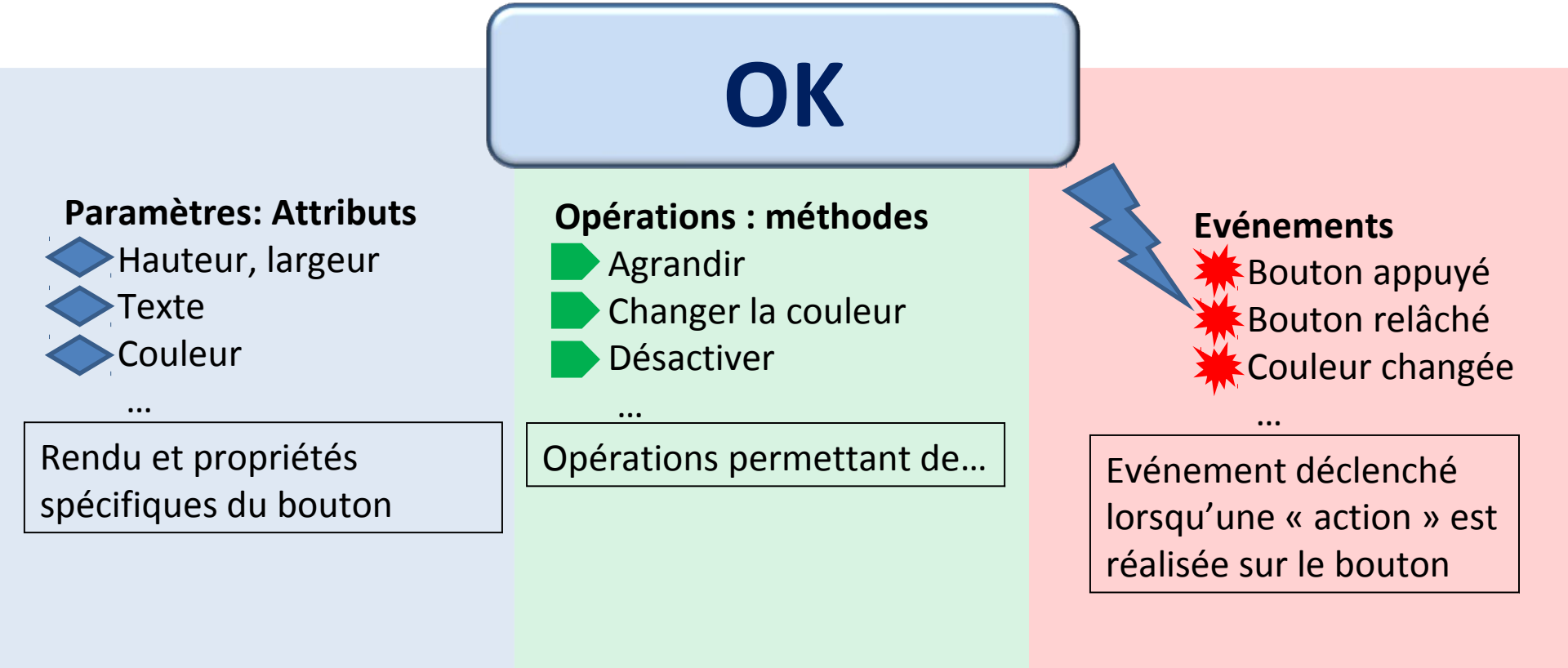
Petit texte

```
JButton bouton=new JButton("cliquer sur moi!");
```

cliquer sur moi!

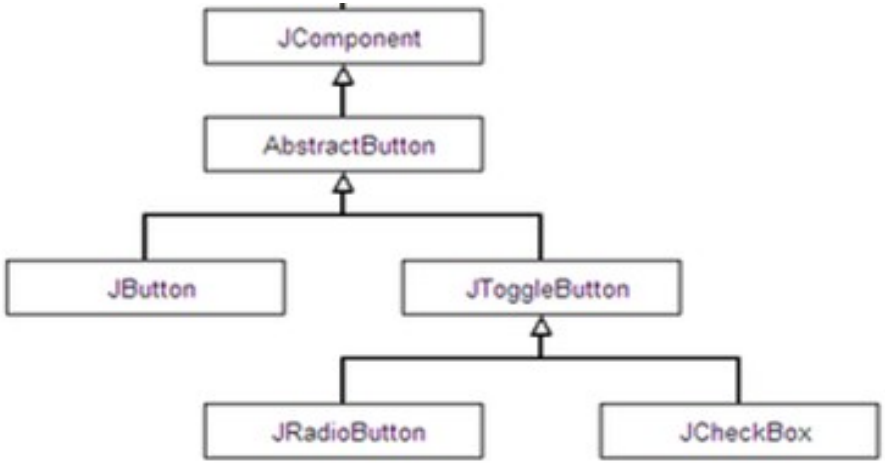
Composants usuels

- Ex de composants atomiques: **JButton**
  - Attributs
  - Méthodes
  - Réactions (notion d'événements)



Composants usuels

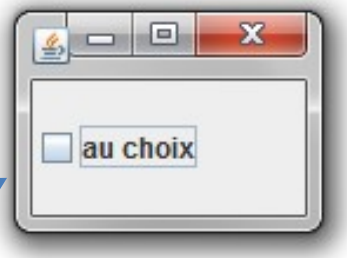
- Les checkBox, les radioButton




```
JButton bouton = new JButton("Cliquez-moi");
```

```
JCheckBox ckBox = new JCheckBox ("au choix");
```

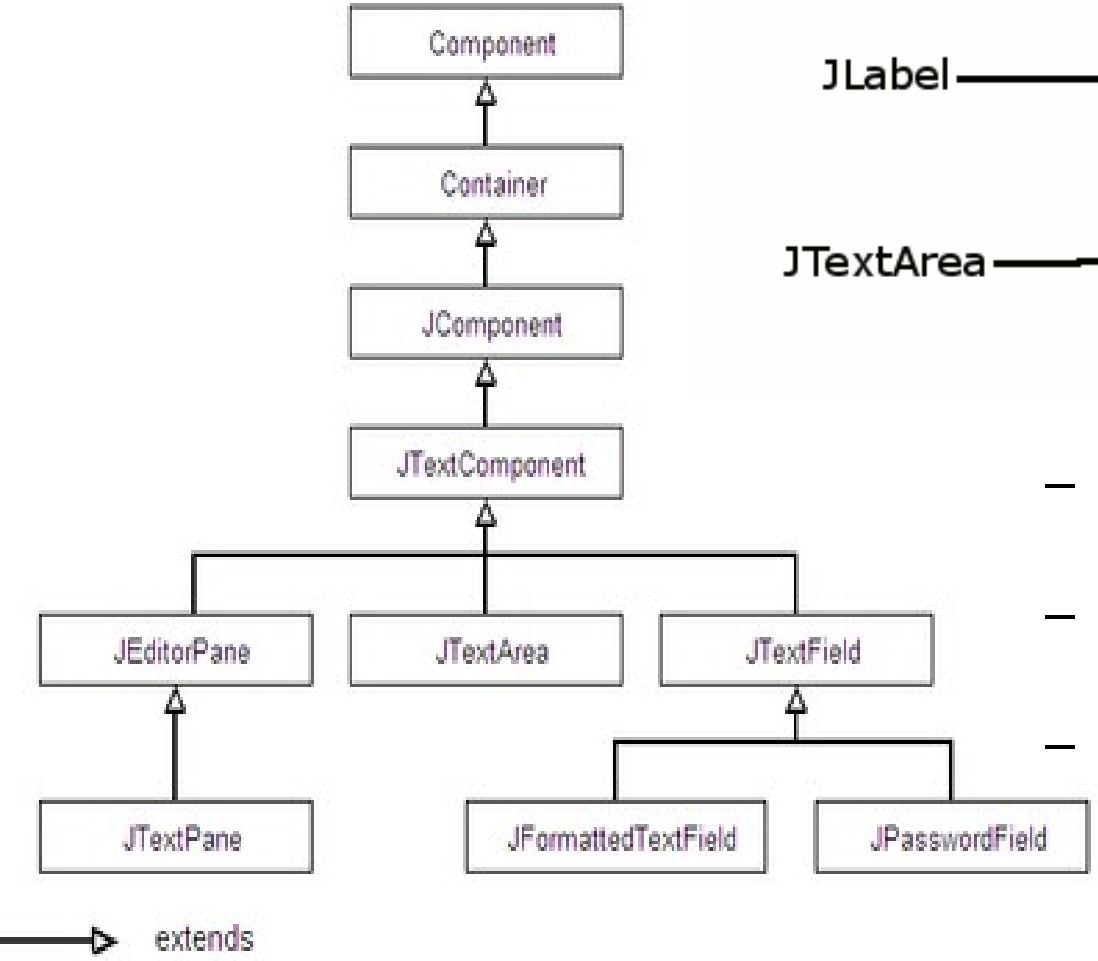
```
JRadioButton rdBouton = new JRadioButton ("a cocher");
```



Il est possible de spécifier l'état de départ : décider si les cases à cocher sont cochées ou pas, les boutons radio sont sélectionnés ou pas en utilisant à chaque fois le constructeur adéquat.

Composants usuels

- Les zones de textes



- **JTextArea** représente une grande zone de texte (plusieurs ligne).
- **JTextField** représente une simple zone de texte pour une saisie.
- **JEditorPane** et **JTextPane** pour des zones de texte plus complexes, incluant différents styles ou des images

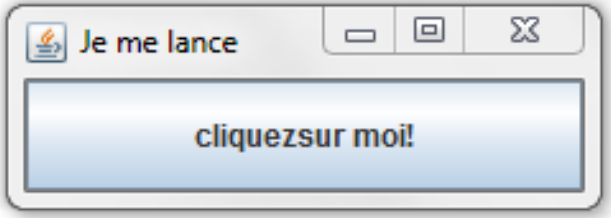
Exemple 2.a: Fenêtre avec bouton basique

Frame et bouton dans le main

```
1  import javax.swing.*;
2
3  class TestBouton1 {
4
5      public static void main(String[] args){
6          JFrame fenetre= new JFrame("Je me lance");
7          JButton bouton=new JButton("cliquez sur moi!");
8          fenetre.add(bouton);
9          fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         fenetre.setVisible(true);
11         }
12 }
```

String en paramètre qui représente le texte visible

Le bouton ainsi crée sans dimension se positionne intégralement dans le cadre de la fenêtre



Il est possible de spécifier une taille du bouton particulière (par exemple 150*120), en ajoutant:

```
import java.awt.Dimension;
bouton.setPreferredSize(new Dimension(150, 120));
```

Fichier TestBouton1.java

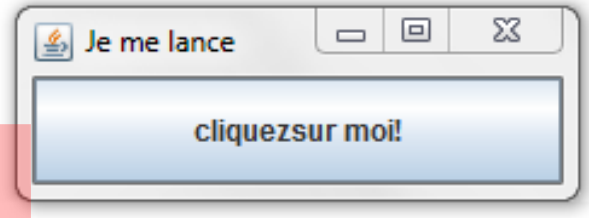
Exemple 2.b: Fenêtre un peu plus évoluée

Classe fenêtre héritière et attribut bouton

```
1  import javax.swing.*;
2
3  public class BoutonFrame extends JFrame {
4
5      private JButton bouton;
6
7      public BoutonFrame(){
           setTitle("Je me lance");
           this.bouton=new JButton();
           bouton.setLabel("cliquez sur moi ! ");
           this.add(bouton);
           this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           this.setVisible(true);
       }

       public static void main(String[] args){
           BoutonFrame fenetre= new BoutonFrame();
       }
   }
```

Constructeur du BoutonFrame

Les objets graphiques (ici le bouton) sont crées dans le constructeur de la classe BoutonFrame



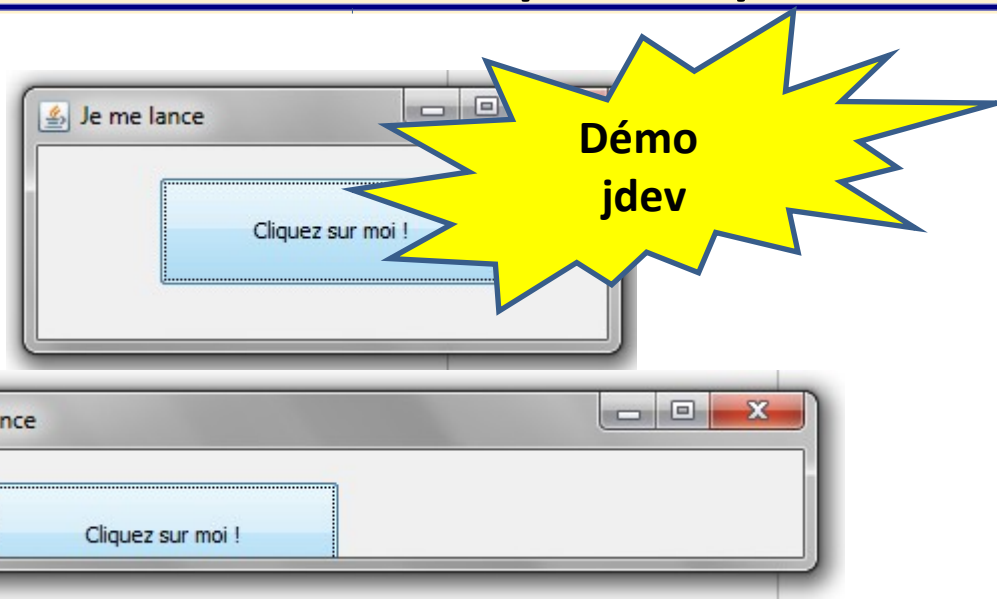
Question: Comment afficher simultanément plusieurs widgets sur une même fenêtre ?

Fichier BoutonFrame.java

Exemple 2.c: Fenêtre par EDI Jdev

Séparation classe fenêtre et classe principale

```
package essaifenetre;

import ...;

public class ClasseFenetre extends JFrame {
    private JButton jButton1 = new JButton();

    public ClasseFenetre() {
        try {
            jbInit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.getContentPane().setLayout( null );
        this.setSize(new Dimension(300, 125));
        this.setTitle( "Je me lance" );
        jButton1.setText("Cliquez sur moi !");
        jButton1.setBounds(new Rectangle(60, 15, 175, 55));
        this.getContentPane().add(jButton1, null);
    }
}
```



Le bouton crée ne se redimensionne pas automatiquement

Question: Comment afficher simultanément plusieurs widgets sur une même fenêtre ?
Et permettre un redimensionnement adapté ?

Les objets graphiques conteneurs
« les boîtes de rangement »

Constat:

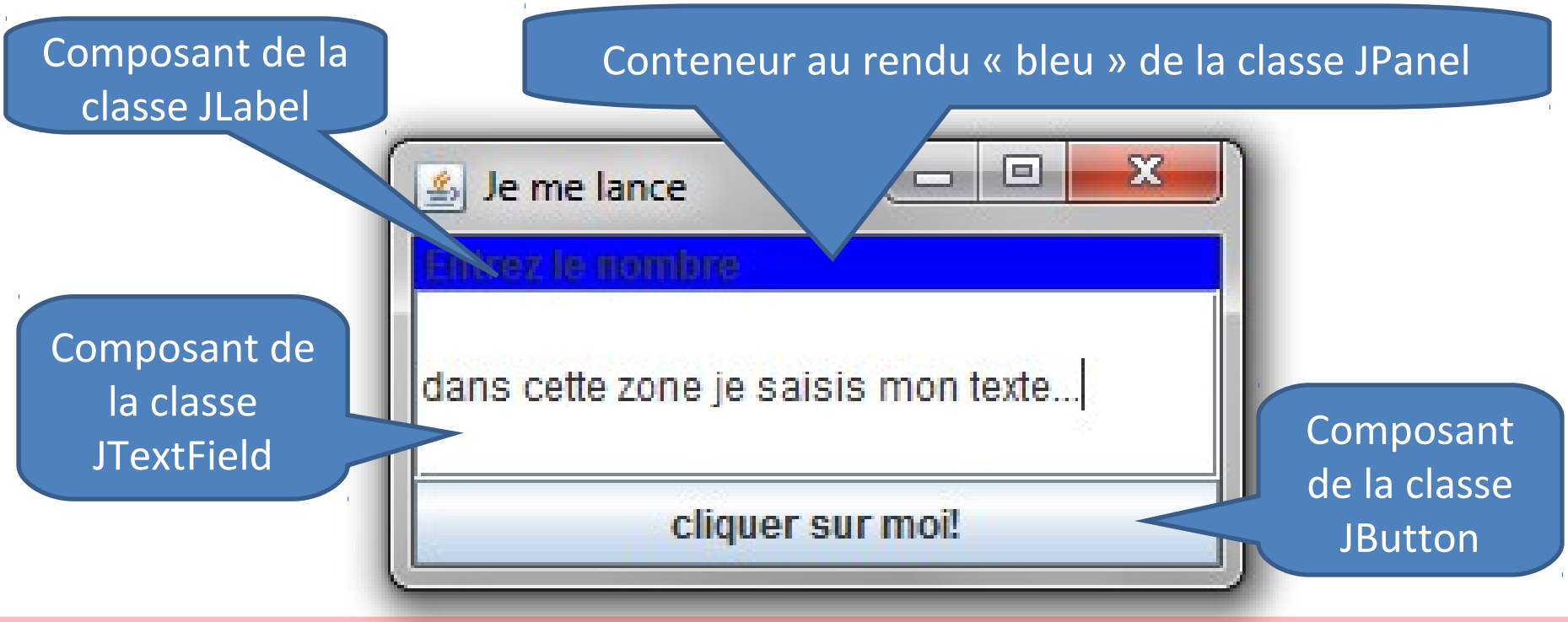
Tout ne peut pas être placé dans la fenêtre directement

- Une fois les composants créés: il faut indiquer la façon dont ils sont disposés à l'écran. Pour cela, il faut:
  - Récupérer le "conteneur" de la fenêtre, c'est-à-dire la zone graphique associée à la fenêtre (*tout JFrame possède un container par défaut*)
  - Ajouter les composants un à un à l'aide de la méthode *add* de la classe *Container* qui prend en argument l'objet graphique à insérer dans le "conteneur".

Exemple…

Problème: *On voudrait avoir une fenêtre avec une grande zone de texte en haut, et une petite zone de texte en bas avec un bouton juste à côté. Comment faire ?*

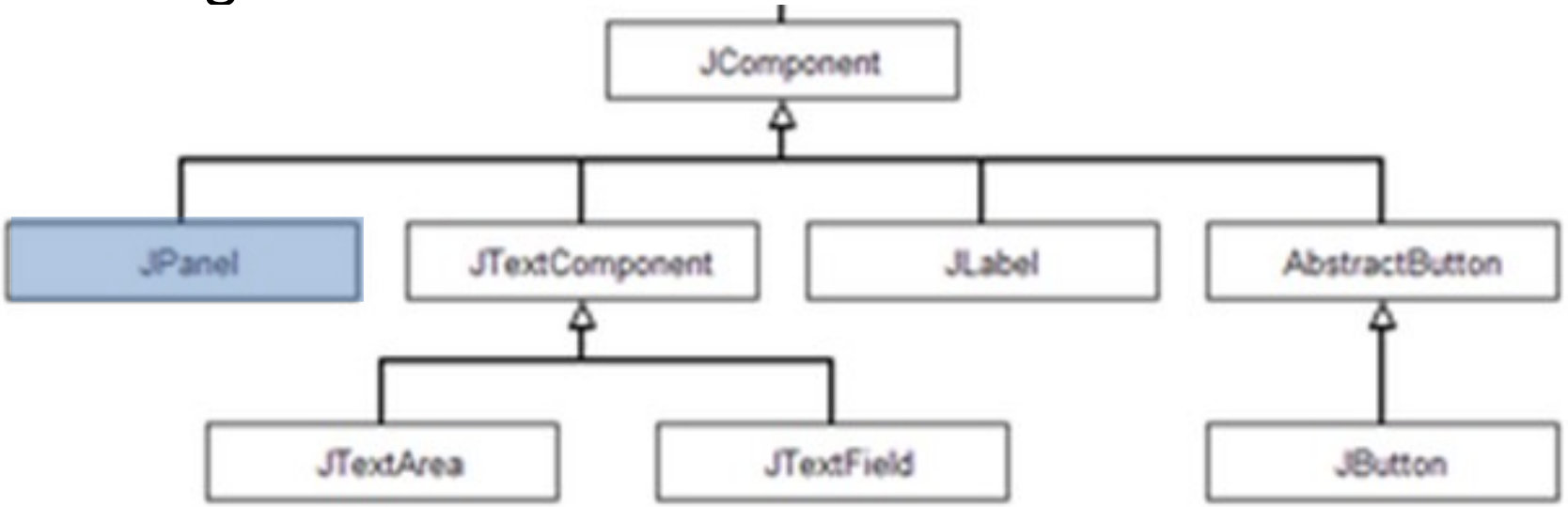
Création d'une fenêtre intégrant plusieurs composants visuels dans un **conteneur** JPanel



1. Pour construire cette fenêtre avec des composants placés sur la surface de la fenêtre, il ne suffit pas de dire quels sont les objets à afficher dans la fenêtre, **mais il faut préciser comment ces objets seront disposés dans cette fenêtre.**
2. La disposition de composants graphiques se fait dans un objet de type **Container** et est effectuée par des objets particuliers : les gestionnaires de disposition (« **Layout managers** »)

Le conteur JPanel

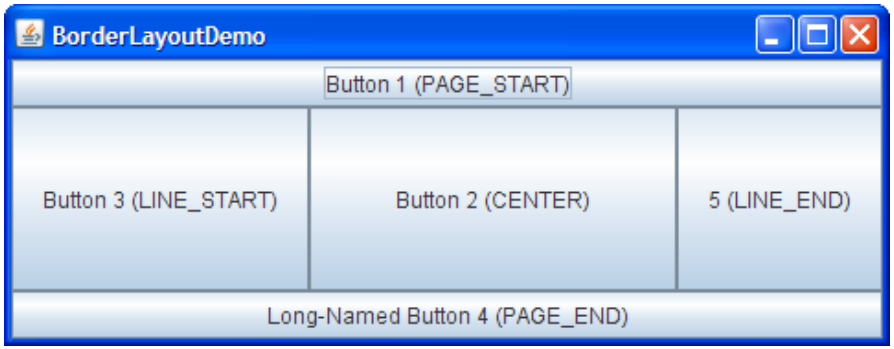
- Un composant - conteneur de placement: le **JPanel**
  - Facilite le positionnement des boutons, des textes, des zones graphiques
  - s'intègre dans une fenêtre



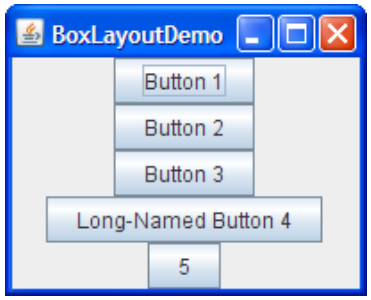

```
JPanel cadre=new JPanel();
```

Exemples de « Layout managers »

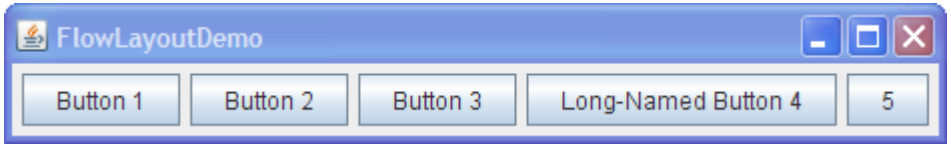
BorderLayout : place les composants en indiquant simplement leur emplacement de la façon NORTH, SOUTH, EAST, WEST, CENTER (layout par défaut des composants SWING)



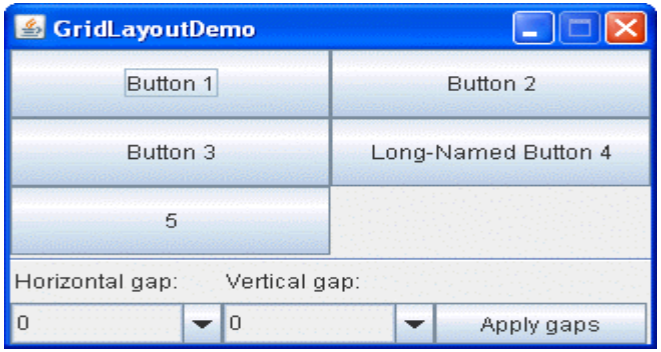
BoxLayout : aligne les composants sur un axe vertical ou horizontal.



FlowLayout : place les composants à la suite sur une ligne, puis sur une autre ligne si le conteneur n'est pas assez large (*par défaut des fenêtres JPanel*).



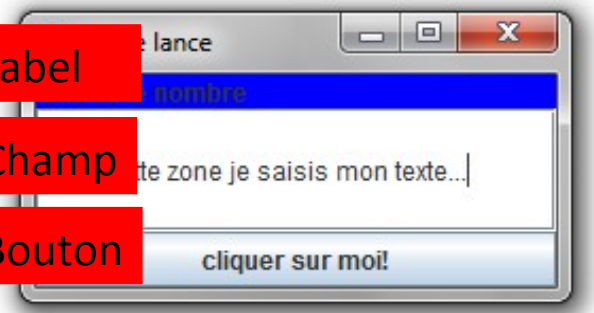
GridLayout : place les composants dans un tableau de cellules de tailles identiques.



Exemple 3.1 « manuel »

Le **conteneur JPanel** associé au gestionnaire de placement **BorderLayout**

```
1  import javax.swing.*;
2  import java.awt.*;
3
4      public class MaFenetrePanel extends JFrame {
5         private JPanel monCadre;
6         private JButton monBouton;
7         private JLabel monLabel;
8         private JTextField monChamp;
9
10        public MaFenetrePanel(){
11           setTitle("Je me lance");
12           monCadre=new JPanel();
13           monCadre.setLayout(new BorderLayout());
14           monCadre.setBackground(Color.blue);
15
16           monBouton=new JButton("cliquer sur moi!");
             monChamp=new JTextField(10);
             monLabel = new JLabel(" Entrez le nombre");

             monCadre.add("Center",monChamp);
             monCadre.add("North",monLabel);
             monCadre.add("South",monBouton);

             this.setContentPane(monCadre);
             this.pack();
             this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27           this.setVisible(true);
28       }
29       public static void main(String[] args){
30          MaFenetrePanel fenetre= new MaFenetrePanel();
31       }
32    }
```



La classe MaFenetrePanel contient autant de champs que d'objets graphiques

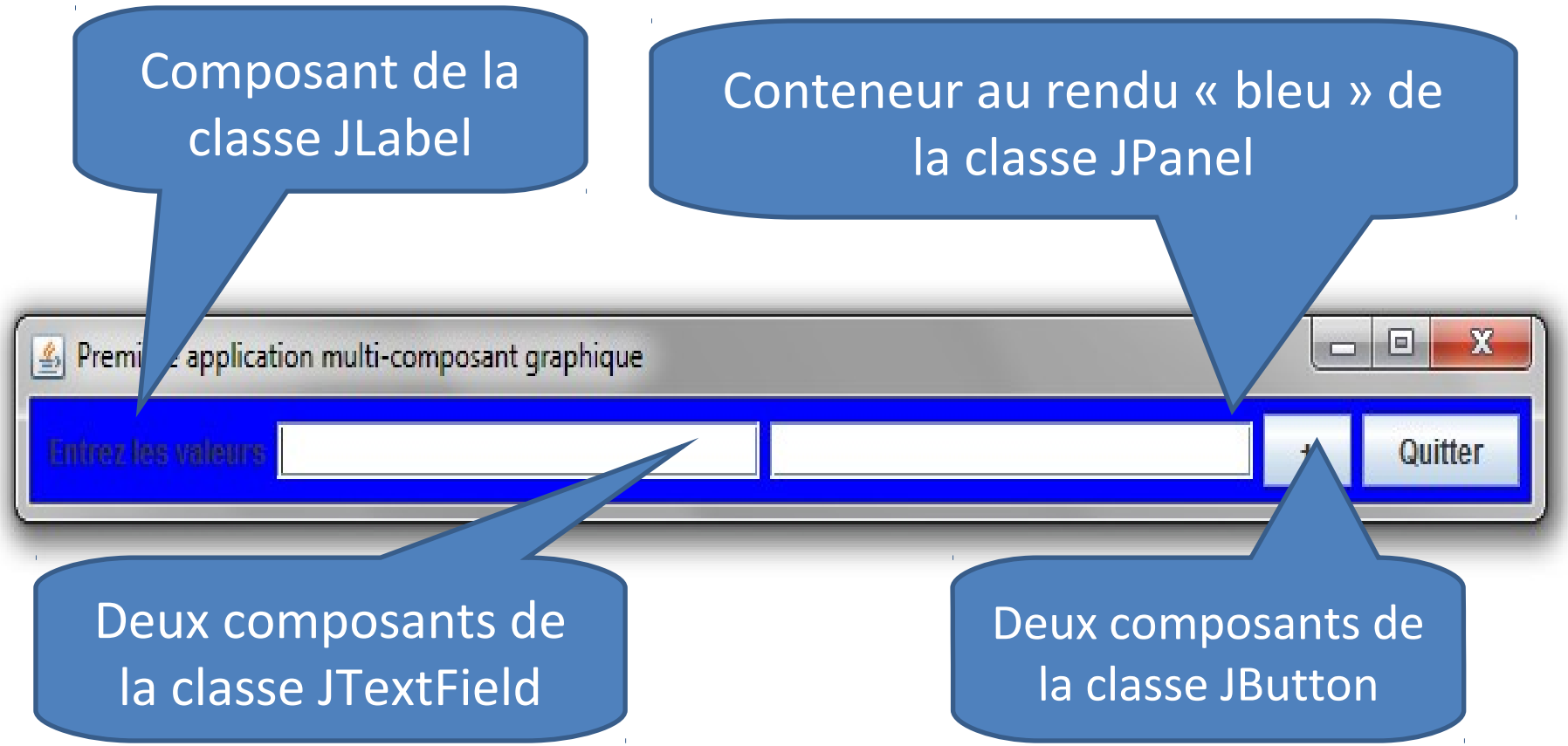
On précise la localisation au moment du placement

Pour la mise en forme: le gestionnaire « BorderLayout », un standard de positionnement

Exemple 3.1 « Jdev »

On peut utiliser directement le conteneur associé à la JFrame

Autre gestionnaire de placement: le **FlowLayout**

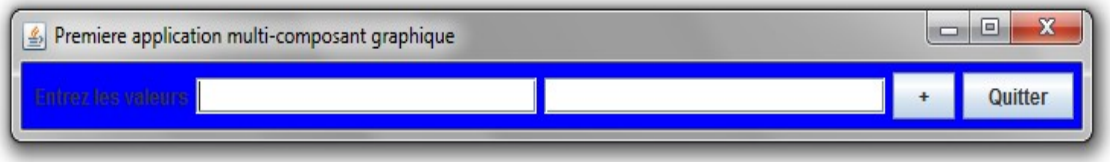


Quel gestionnaire de position? Le **FlowLayout**…

Exemple 3.2 « manuel »

Le **conteneur JPanel** associé au gestionnaire de placement **FlowLayout**

```
1  import javax.swing.*;
2  import java.awt.*;
3
4      public class MaFenetrePanel2 extends JFrame {
5          private JPanel cadre;
6          private JButton boutonPlus;
7          private JButton boutonQuitter;
8          private JLabel affichage;
9          private JTextField champA;
10         private JTextField champB;
11
12         public MaFenetrePanel2(){
13             setTitle("Premiere application multi-composant graphique");
14             JPanel conteneur=new JPanel();
15             conteneur.setLayout(new FlowLayout());
16             conteneur.setBackground(Color.blue);
17
18             this.affichage = new JLabel(" Entrez les valeurs");
19             this.champA = new JTextField(20);
20             this.champB = new JTextField(20);
21             this.boutonPlus = new JButton("+");
22             this.boutonQuitter = new JButton("Quitter");
23
24             conteneur.add(this.affichage);
25             conteneur.add(this.champA);
26             conteneur.add(this.champB);
27             conteneur.add(this.boutonPlus);
28             conteneur.add(this.boutonQuitter);
29
30             this.setContentPane(conteneur);
31             this.pack();
32             this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33             this.setVisible(true);
34         }
35         public static void main(String[] args){
36             MaFenetrePanel2 f= new MaFenetrePanel2();
37         }
38     }
```



Gestionnaire de placement de type FlowLayout

Exemple 3.2 « Jdev »

Le **conteneur JPanel** associé au gestionnaire de placement **FlowLayout**

BILAN

Pour la création d'une interface évoluée avec Swing

Les étapes de création:

1. Création d'une **fenêtre** par **héritage** et spécialisation par ajout sous la forme **d'attributs** privés des composants visuels , les **widgets (bouton**, zones de saisies…)
2. Configuration des composants (couleurs, dimensions…)
3. Récupération / insertion du conteneur (*Container*) de la fenêtre principale et placement des composants selon le Layout manager choisi

=> Manuellement ou à l'aide de l'EDI Jdev

Partie 2

PROGRAMMATION ÉVÉNEMENTIELLE: PRINCIPES

Introduction: Rappel sur programmation procédurale

- **Principe**: l'application a le contrôle, le déroulement est contrôlé par une **séquence** d'instructions écrites (ou de méthodes)
- Le programmeur écrit la boucle principale

```
Programme principal
    Initialisations
    Répéter
            lire une commande
            traiter une commande
            jusqu'à finir la commande
```

Introduction: Programmation événementielle

- **Principe**:
  - l'utilisateur a le contrôle (et pas le programmeur)
  - le déroulement est contrôlé par la survenue d'événements (dont les actions de l'utilisateur)

- **Une boucle d'attente**
  - des initialisations
  - des instructions et méthodes de réaction aux événements

```
Initialisation
while (true) {
  attendre événement suivant
E
  traiter événement E
}
```

Le programme ne se termine pas implicitement mais l'application se termine par `System.exit(0)`

Notion d'événements: généralités

- **Conséquences**:
  - On parle d'application toujours prête à réagir
  - Elle s'appuie sur des **interfaces graphiques**

- **Evénement** (ou « message »)
  - Est envoyé à l'application ciblée à chaque action élémentaire de l'utilisateur
  - Est un "objet" que l'on passe à une fonction particulière du programme, chargée de l'acheminer vers la partie logicielle concernée
    - *Chaque action engendre un type d'objet différent: par exemple, la position (x, y) de la souris*
  - Il existe donc des **classes d'événements** spécifiques pour chaque type d'interaction avec l'utilisateur
  - Une **boucle de gestion des événements** les intègre au fur et à mesure de leur arrivée

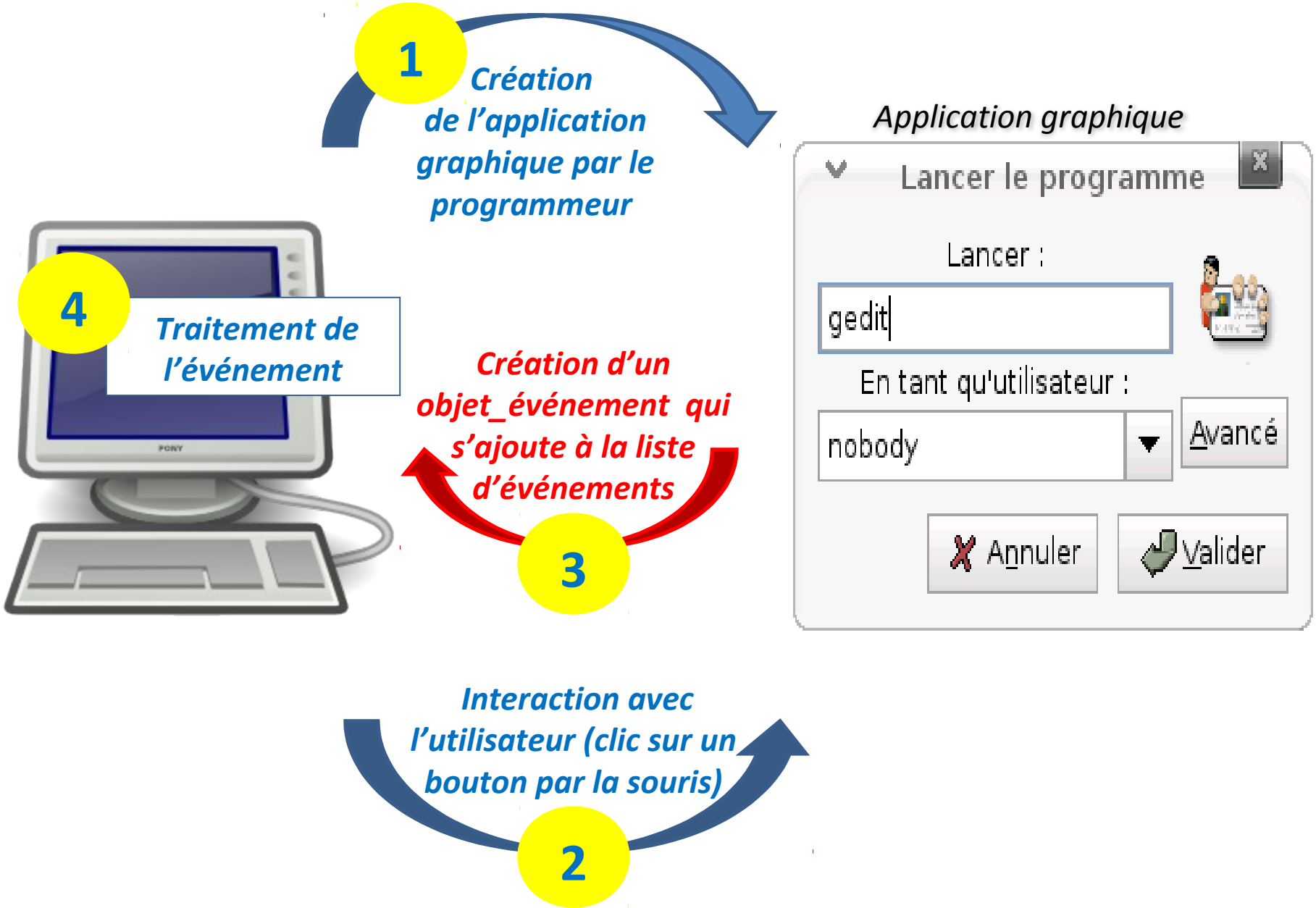
Notion d'événements: généralités

- **Événements liés aux périphériques**
  - Entrée/sortie du curseur dans une fenêtre
  - Utilisation d'un des boutons
  - Frappe au clavier
  - Sélection d'un item dans une liste déroulante
  - …
- **Événements liés aux applications**
  - Création/destruction de fenêtres
  - Réaffichage
  - …

Notion d'événements: généralités

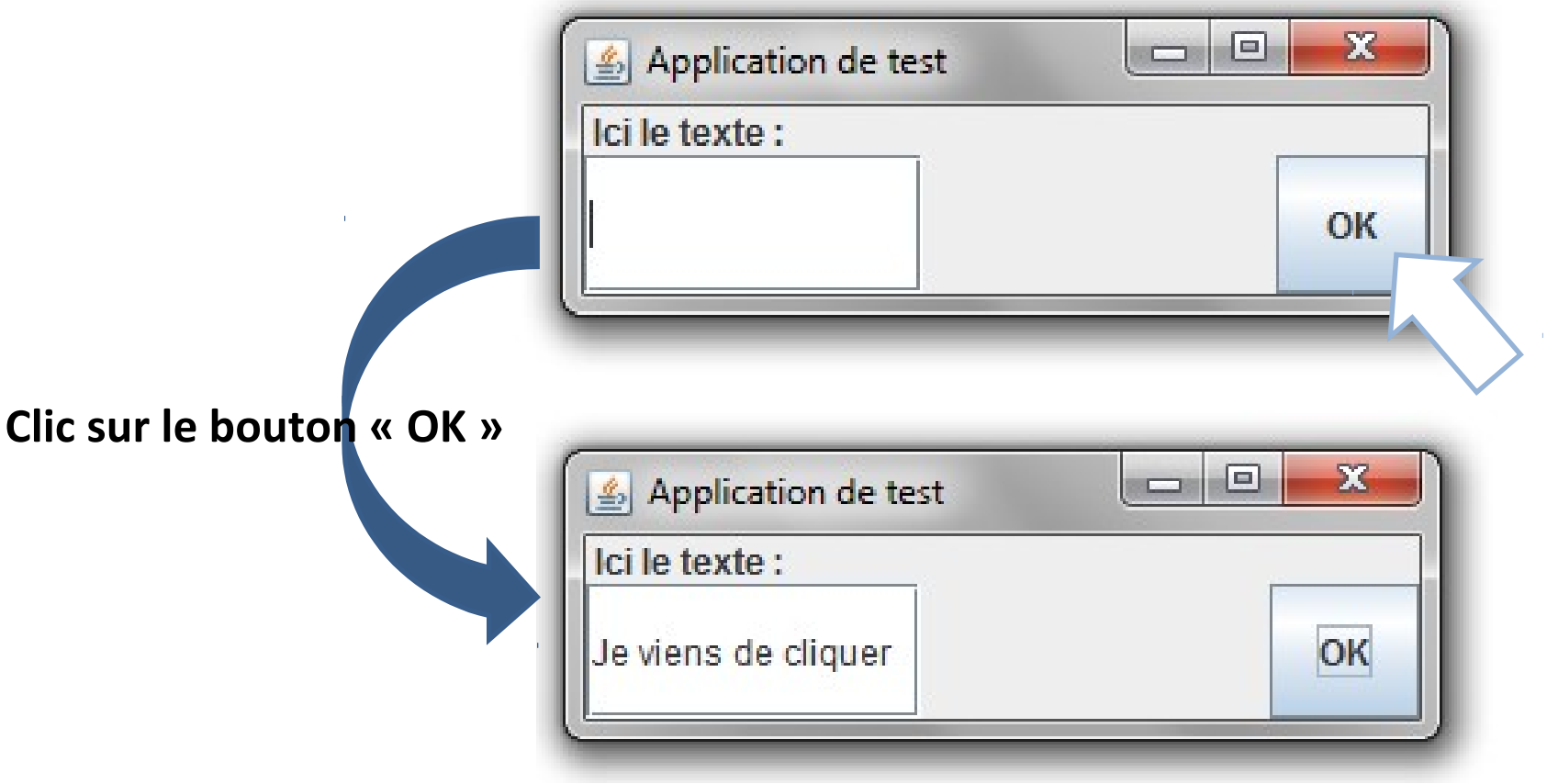
- Exemples d'événements
  - **MouseEvent**: actions discrètes de la souris
  - **MouseMotionEvent**: action continues sur la souris
  - **FocusEvent**: gain et perte du clavier
  - **KeyEvent**: actions sur le clavier
  - **ActionEvent**: pression d'un bouton, choix dans un menu
  - **WindowEvent**: événement survenant sur une fenêtre

La gestion des événements



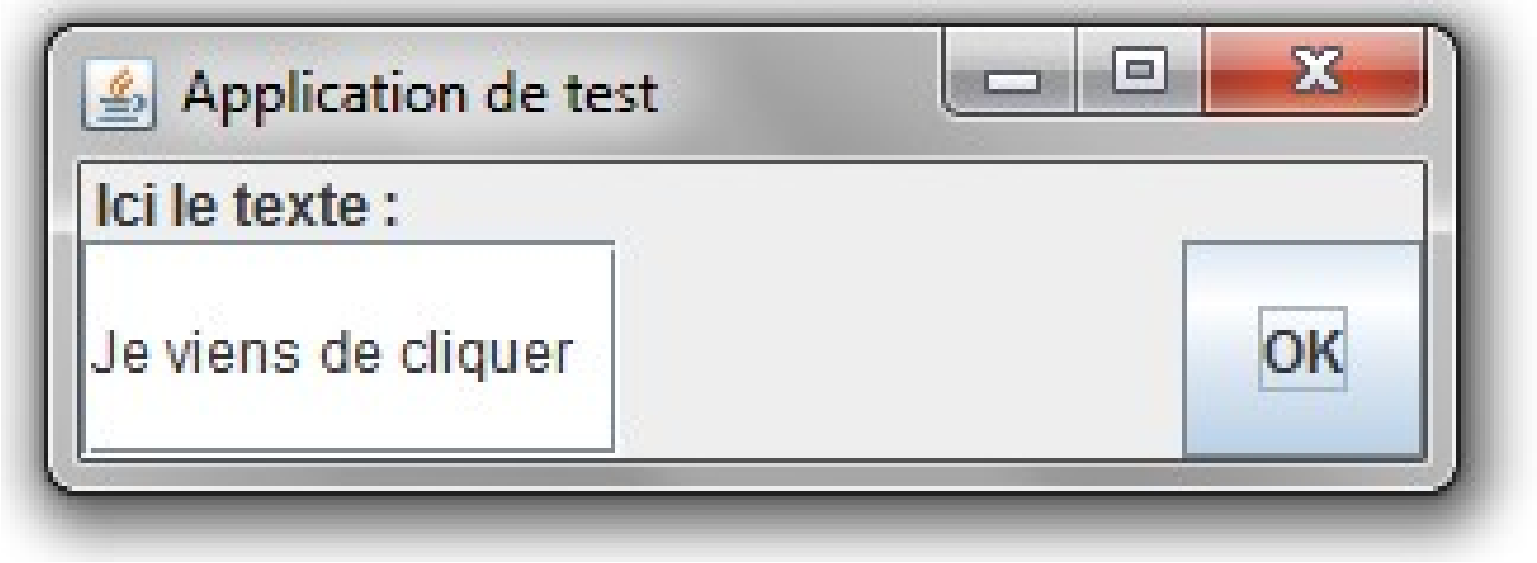
La gestion des événements

- Un exemple: L'IHM doit pouvoir réagir à une action de l'utilisateur



La gestion des événements

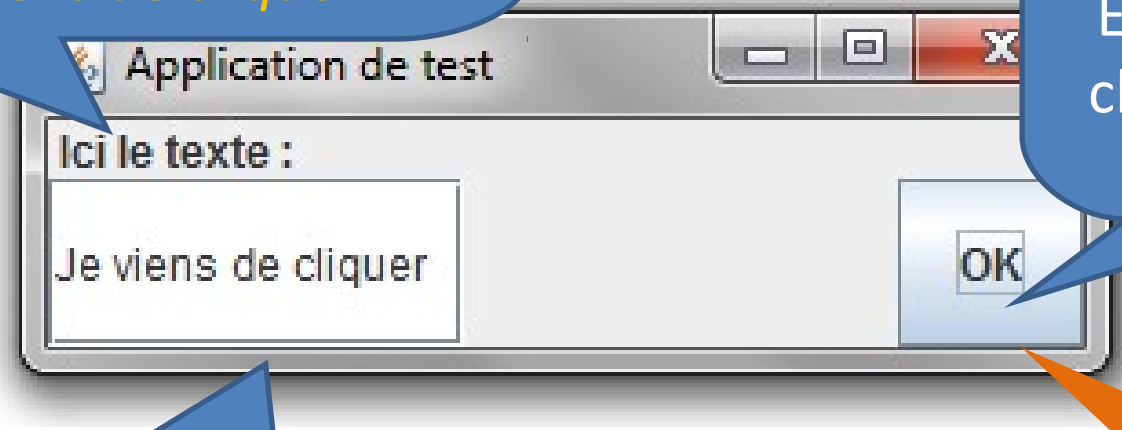
- Comment « *capturer* » l'action de clic sur le bouton et provoquer l'affichage du texte *" Je viens de cliquer "* dans la boîte de texte?



La gestion des événements

actionPerformed agit en conséquence avec ici l'affichage de *« Je viens de cliquer »*

3



1

Un clic sur le bouton va créer un objet Evénement ***e*** de la classe Action Event

ActionEvent

e

Un écouteur (interface d'écoute) de ce type d'événements nommé **ActionListener** reçoit alors l'événement et le traite dans une méthode **actionPerformed**

2

La gestion des événements

Interface d'écoutes génériques : EventListener

Les **interfaces écouteur EventListener** :

- permettent à un composant de générer des événements utilisateurs (*ex: MouseListener, KeyListener, ActionListener…*).

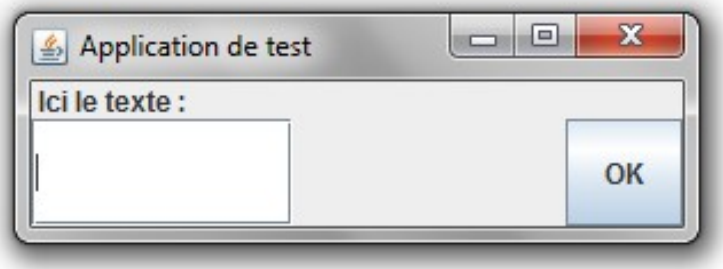
- une classe JFrame doit contenir **une interface d'écoute de ce type** pour chaque type de composant OU une interface unique chargée de rechercher le composant qui est responsable de l'événement

La gestion des événements: exemples

- Quelques événements et leurs écouteurs (listener) associés

Interface écouteur	Evénement	Méthode
MouseListener	L'utilisateur a relâché le bouton de la souris	mouseReleased
	Le pointeur de la souris est entré dans le composant	mouseEntered
	Le pointeur de la souris est sorti du composant	mouseExited
	Clic complet de la souris (pression puis relâchement)	mouseClicked
	L'utilisateur a appuyé sur un bouton de la souris	mousePressed
ActionListener	L'utilisateur clique sur un bouton ou presse « enter » dans un champ texte ou sélectionne un champ dans un menu	actionPerformed
KeyListener	L'utilisateur clique sur une touche	keyPressed
	L'utilisateur relâche une touche	keyReleased

La gestion des événements: réalisation




```
/**
 *      Classe héritière de JFrame
 */
public class FenetreEcouteur extends JFrame {
    private JPanel monCadre;
    private JButton monBouton;
    private JLabel monLabel;
    private JTextField monChamp;
```

1

```
/**
 *    Constructeur fenêtre et composants
 */
  public FenetreEcouteur(){
     setTitle("Application de test");
     monCadre=new JPanel();
     monCadre.setLayout(new BorderLayout());
     monBouton=new JButton("OK");
     monChamp = new JTextField(10);
     monLabel = new JLabel (" Ici le texte : ");
     monCadre.add("North",monLabel);
     monCadre.add("East",monBouton);
     monCadre.add("West",monChamp);

     monBouton.addActionListener(new FenetreEvent());

     this.setContentPane(monCadre);
     this.pack();
     this.setSize(300,100);
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     this.setVisible(true);
  }
```

Instanciation d'une interface d'écoute

La gestion des événements: réalisation

Création d'une classe **interne** à la Fenêtre: Interface d'écoute

2

```
/**
*    Solution proposée:
*    Gestionnaire d'événements - écouteur
*    classe Interne implémentant l'interface ActionListener
*/
class FenetreEvent implements ActionListener{
    public void actionPerformed(ActionEvent e){
        monChamp.setText("Je viens de cliquer");
    }
}
```

Type d'événement

Méthode de traitement de l'événement