



# Programmation IHM - Cours 1

## Développement interfaces utilisateurs en Java

V. DESLANDRES

[veronique.deslandres@univ-lyon1.fr](mailto:veronique.deslandres@univ-lyon1.fr)

Déc 2021

# Sommaire de ce cours

- Présentation du module [3](#)
- Introduction : interfaces utilisateurs [5](#)
- Swing : Composants, Conteneurs [13](#)
- Créer une fenêtre [21](#)

# Organisation du cours

- **Volume Horaire:** 24h
- 11 séances de 2h (cours + TP) + 2h d'examen
- **Modalités d'évaluation :**
  - Note de TP (moyenne des 4 TP, 40%)
  - DM final sur les concepts vus en TPs (60%)
  - Bonus/malus sur votre implication (fréquence / pertinence des interactions, 2 pts)
- **Contenu:**
  - Bases d'IHM, Interface utilisateur, Gestion événementielle
  - (Lien avec les BD JDBC)

# Programme du Module

- Introduction sur les interfaces utilisateurs
- Bases de l'API Swing : **Conteneurs, Composants**
- **Création** des interfaces
- Gestionnaire de **dispositions**, mise en page :
  - **layout**
- Gestion des **événements**
- Focus spécifiques :
  - Présentation **table (JTable)**, fenêtres de **dialogue**
  - **JDBC** : API liens avec les bases de données

# Introduction : les interfaces utilisateur (IHM, GUI)



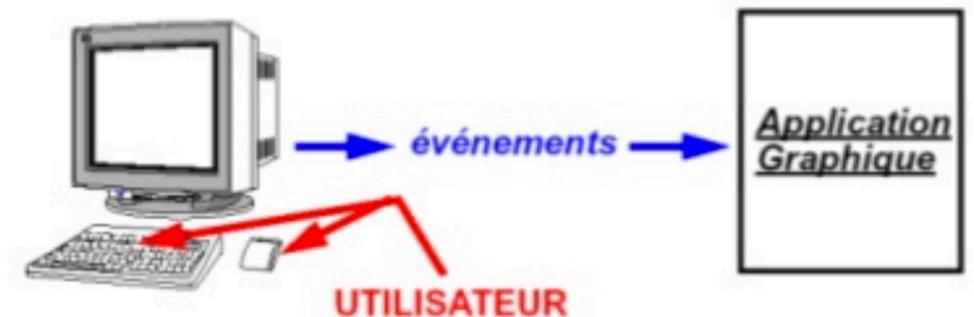
# Principes de base

- **Interface Console**

- C'est le programme qui pilote l'utilisateur, en le sollicitant quand nécessaire pour qu'il fournisse des données
  - ➔ Dialogue **en mode texte et séquentiel** dans une fenêtre appelée « Console »

- **Interface graphique (GUI – Graphical User Interface)**

- L'utilisateur pilote le programme, qui réagit à ses demandes (sélection d'articles, d'item de menu, clic bouton,...)
- Chaque action de l'user = événement
- Programmation « **évenementielle** »

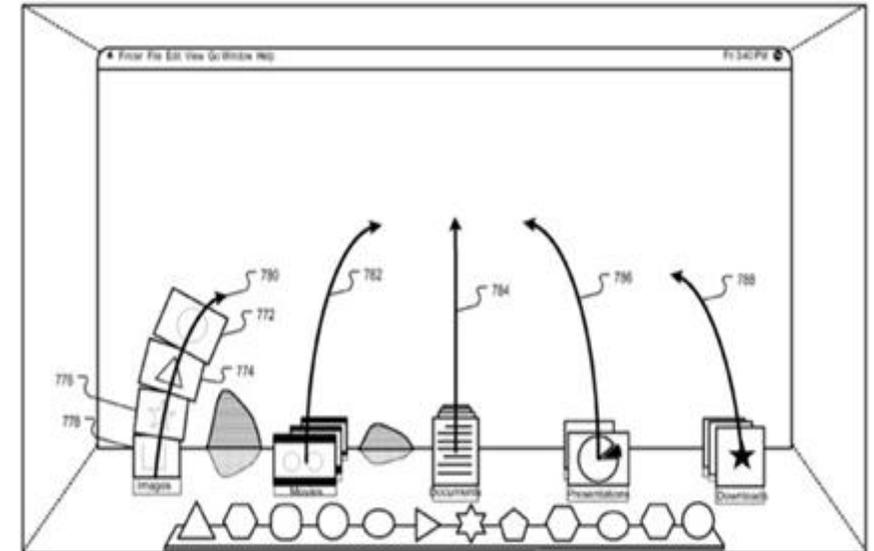


# Fondamentaux de l'IHM

- L'interface visible d'une application est constituée de 2 éléments :

## (1) Les **conteneurs** :

Contiennent des objets graphiques qui peuvent être des composants ou d'autres conteneurs



- (2) Les **composants atomiques** : boutons, cases à cocher, zone de texte, slider, etc.

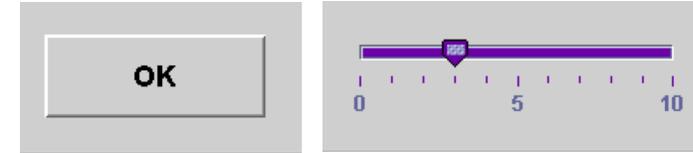
# Une interface utilisateur se compose de :

- Une **fenêtre** de travail
- Une **zone** où afficher les composants graphiques dans cette fenêtre de travail :
  - Un panneau (**Panel**)
- Des **composants** à insérer dans cette fenêtre
  - Boutons, cases à cocher, menus, barre de tâches,...
- Une **mise en page** des composants
  - Le **Layout**: mis bout à bout, centrés, en tableau,...
- La **représentation graphique** des composants
  - Couleur, forme, image,...
- Des gestionnaires **d'évènements**
  - Répondre aux actions de l'utilisateur

# Java propose

- Des **composants graphiques**

- Widgets



- Des classes de **gestion de la position** des composants sur la fenêtre

- `LayoutManager`

- Des éléments de **représentation graphique**

- `Color`, `Font`, `Graphics`, `Point`, `Rectangle`, `Image`, `Icon`...

- Des mécanismes de **gestion d'événements**

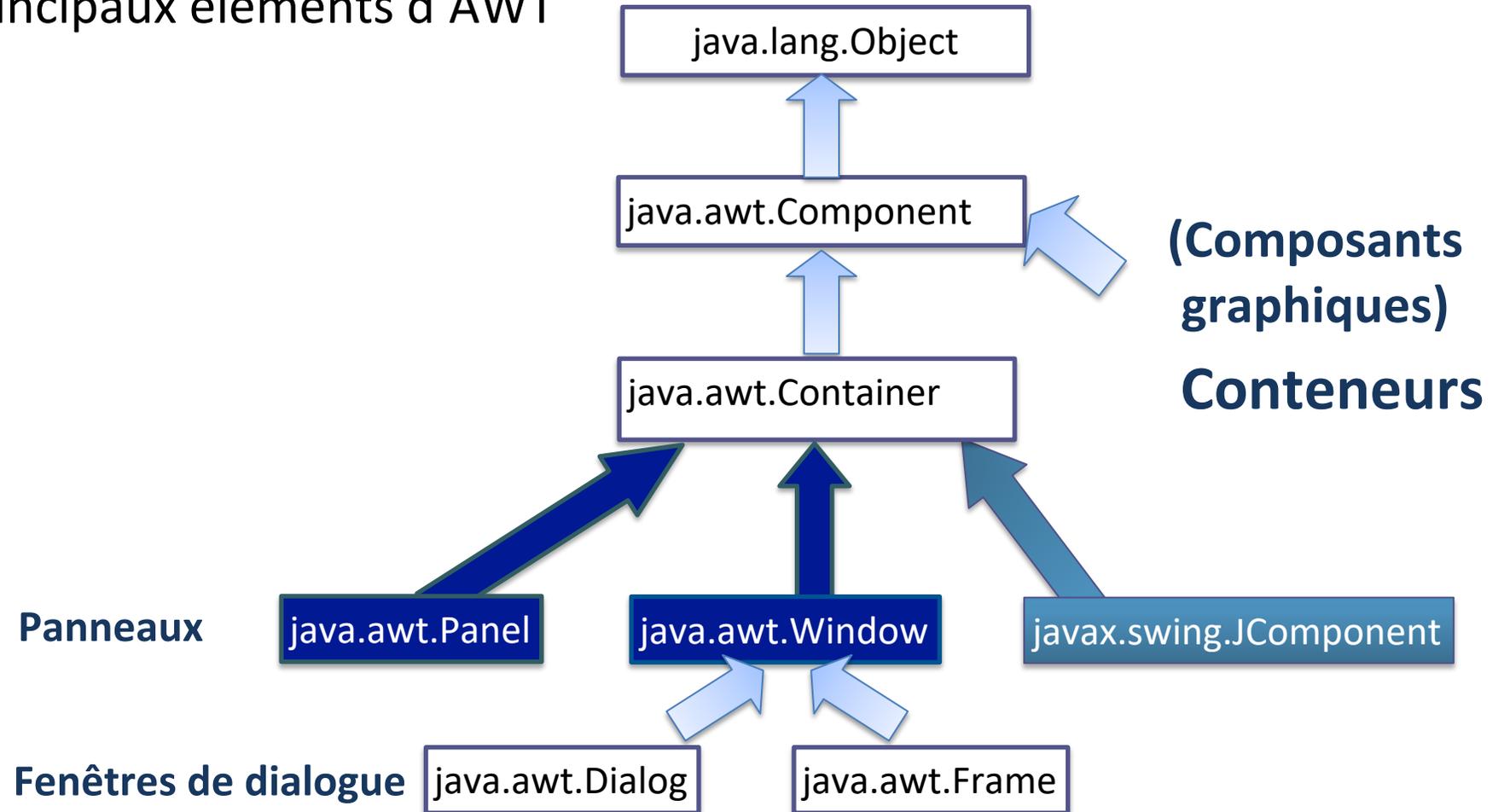
- `java.awt.events`

# AWT, SWING

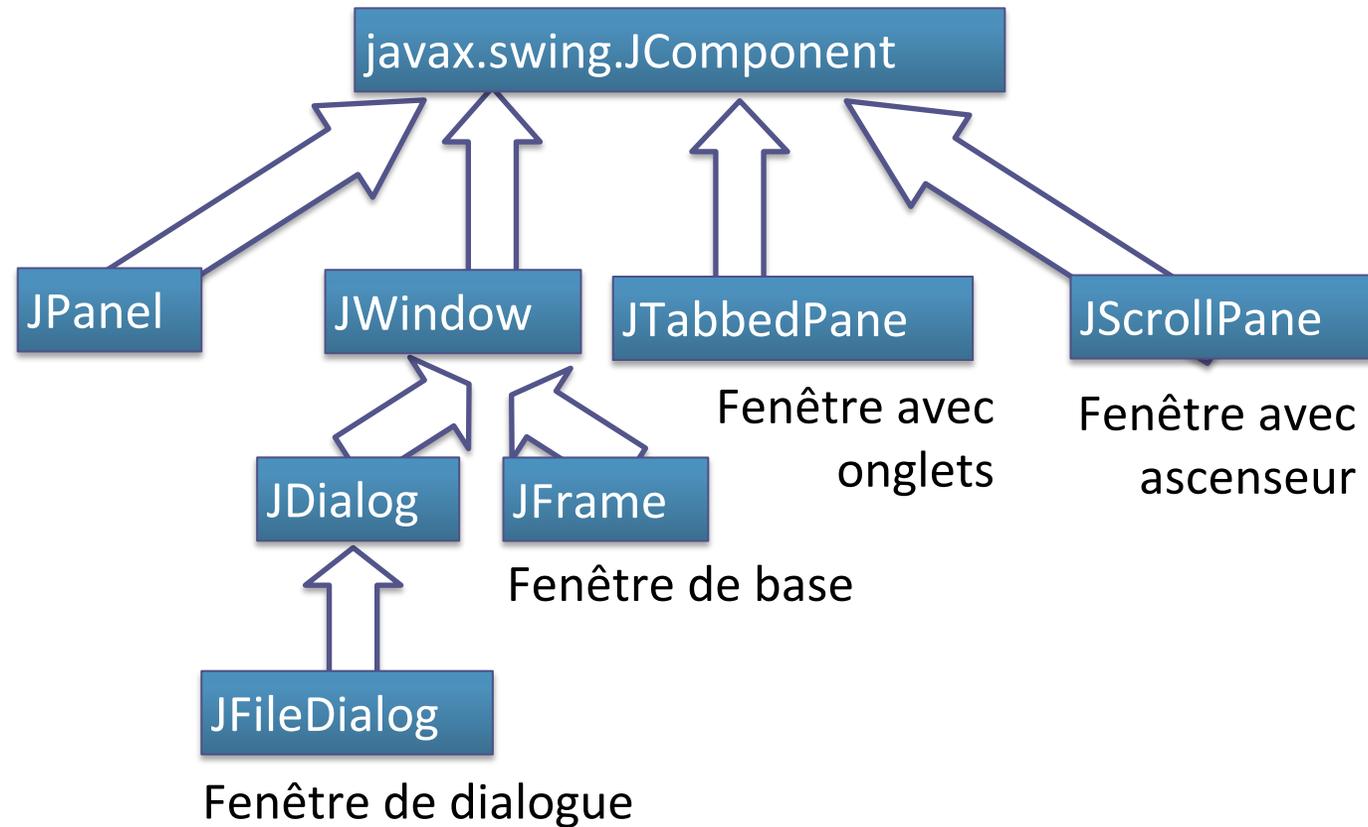
- Première bibliothèque graphique JAVA: **AWT**
  - Package: `java.awt`
  - Utilisation du code qui dépend du système d'exploitation
  - Une gestion des événements
  - Composants limités
- Nouvelle bibliothèque graphique JAVA: **SWING**
  - Package: `javax.swing`
  - Plus riche et plus personnalisable
  - Construite sur AWT, fournit des composants plus performants

# Arborescence des packages AWT/SWING

- Hiérarchie des principaux éléments d'AWT



# Arborescence des packages SWING

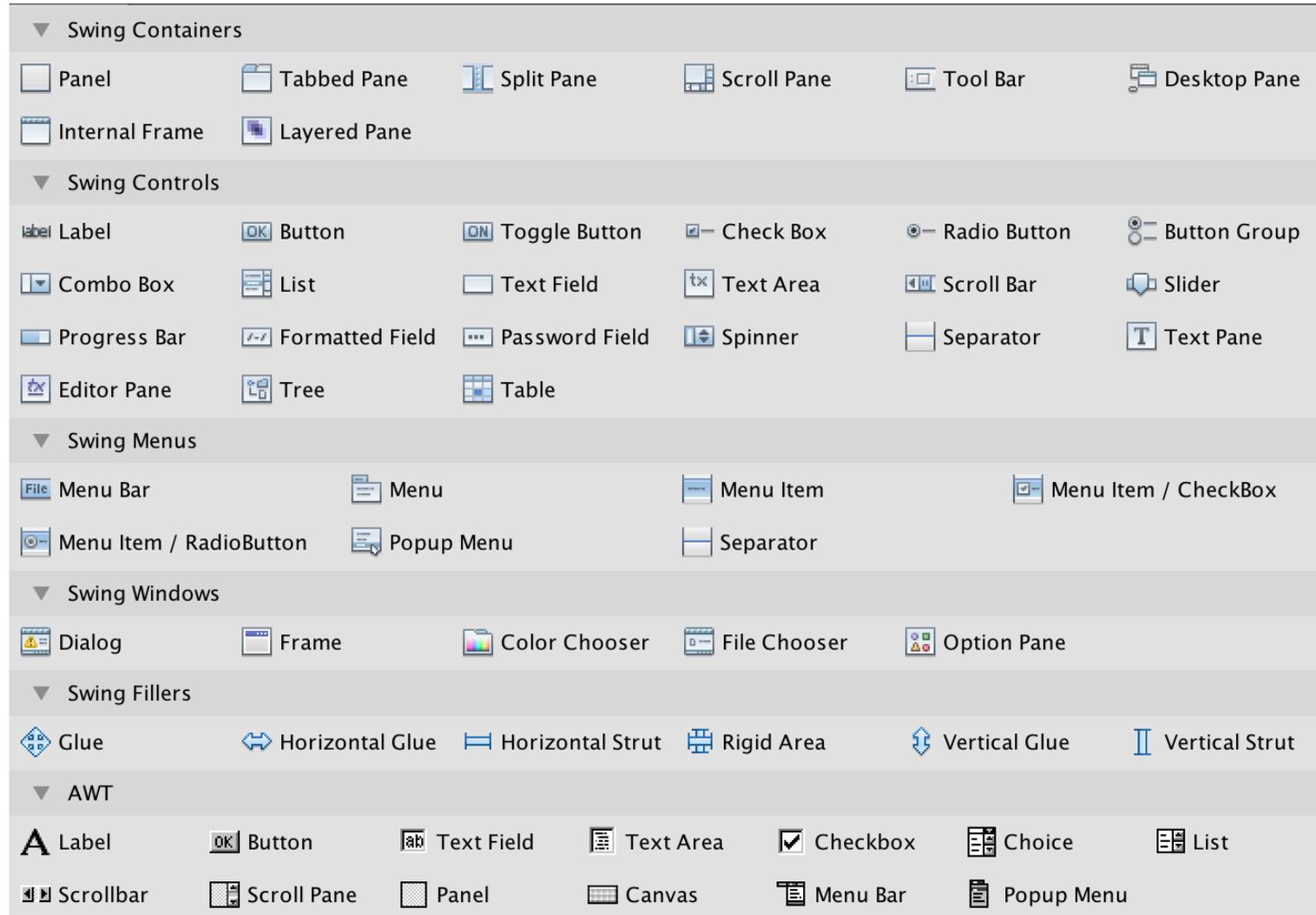


# Bases de Swing Composants, Conteneurs

# Composants graphiques (SWING)

## Les widgets

- JLabel
- JButton
- JToggleButton
- JCheckBox
- JRadioButton
- ButtonGroup
- JComboBox
- JList
- JTextField
- JTextArea
- JScrollBar
- JMenuBar
- JPopupMenu



## Les containers

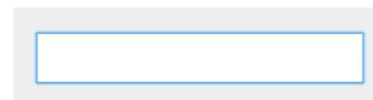
- JWindow
  - JFrame
  - JDialog
    - JFileDialog
- JPanel
  - Applet
- JTabbedPane
- JScrollPane

# Composants Texte de la Swing



## JTextField

```
JTextField numTel = new JTextField("",10);
```



## JTextArea

```
JTextArea commentaire=new JTextArea("Enter more text to see scrollbars", 10, 10);
```

A visual representation of a **JTextArea** component, which is a multi-line text input area with scrollbars. The text "Enter more text to see scrollbars" is displayed inside the area.

# JLabel et JButton

**JLabel**

```
JLabel texte=new JLabel("Texte");
```

Texte

**JButton**

```
JButton bouton=new JButton("OK");
```

OK

**JCheckBox**

```
JCheckBox box1=new JCheckBox("BOX1", true);  
JCheckBox box2=new JCheckBox("BOX2");
```

BOX1  BOX2

*pour cocher un élément par défaut (facultatif)*

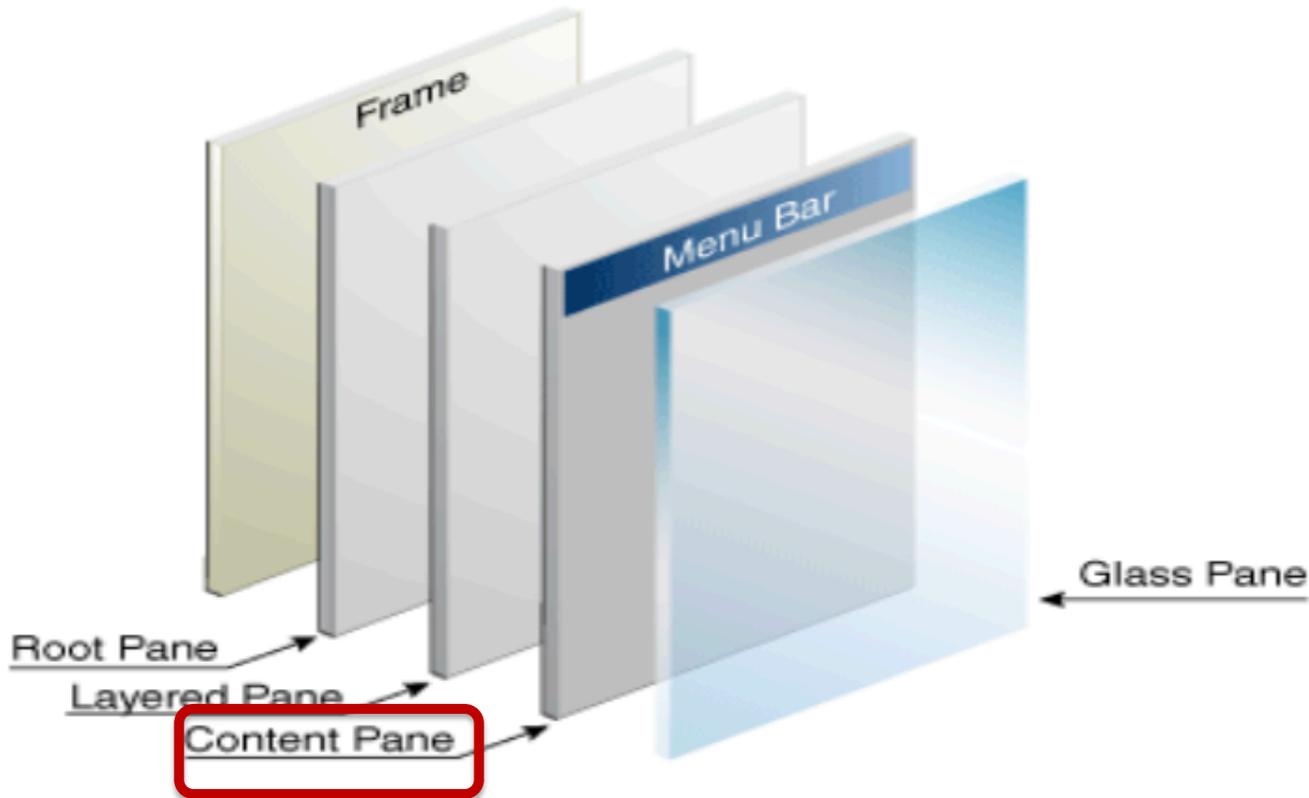
**JRadioButton**

```
JRadioButton radio1=new JRadioButton("Radio1", true);  
JRadioButton radio2=new JRadioButton("Radio2");  
ButtonGroup groupRadio=new ButtonGroup();  
groupRadio.add(radio1);  
groupRadio.add(radio2);
```

Radio1  Radio2

*Groupe : pour gérer choix exclusif (facultatif)*

# Une fenêtre : plusieurs couches



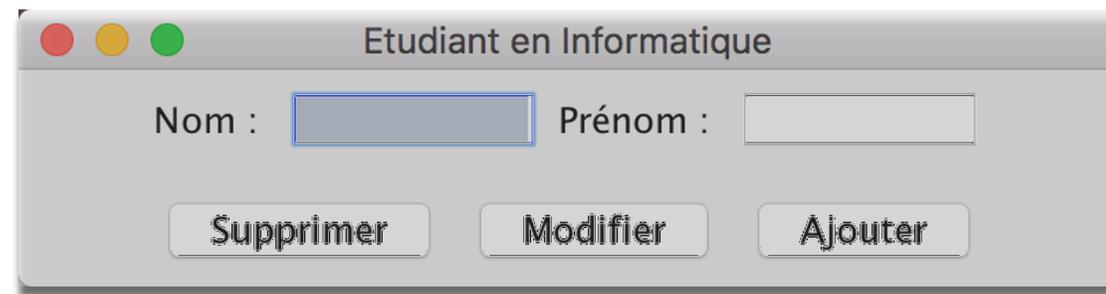
C'est le **contentPane** qui contient les composants (boutons et autres widgets)

C'est une instance de la classe **Container**

Par exemple pour ajouter un bouton sur une fenêtre Frame :

```
maFrame.getContentPane().add(new JButton("OK")) ;
```

# Création d'une fenêtre

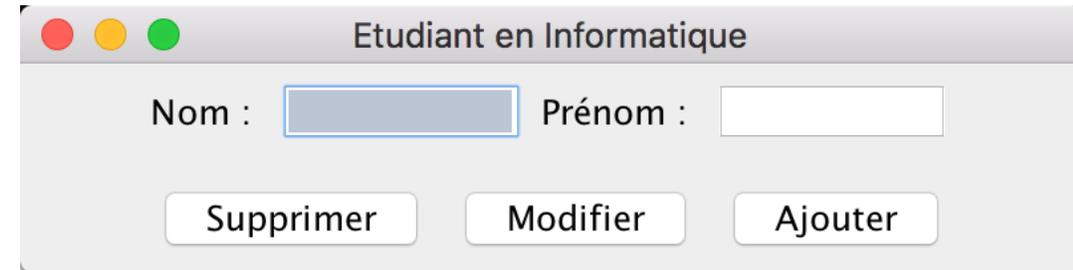


Etudiant en Informatique

Nom :  Prénom :

# Création d'une JFrame + appel dans le *main*

```
public class FenetreEtudiant extends JFrame {  
  
    // les composants Swing de la fenêtre  
    JLabel lblnom, lblprénom;  
    JTextField txtnom, txtprénom;  
    JButton btSupp, btModif, btAjout;  
  
    public FenetreEtudiant() {  
        initComponents();  
        setTitle("Etudiant en Informatique");  
        setSize(400, 100);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    } // fin du constructeur
```



→ Quitter qd on ferme la fenêtre

## setDefaultCloseOperation() de JFrame

**JFrame.EXIT\_ON\_CLOSE**

— **quitte** l'application

**JFrame.HIDE\_ON\_CLOSE (choix par défaut)**

— Cache la fenêtre **sans quitter l'application**

*(quand on a plusieurs fenêtres, on peut souhaiter fermer une fenêtre et continuer l'application)*

**JFrame.DISPOSE\_ON\_CLOSE**

— Quand on a plusieurs fenêtres, **rend la main à la fenêtre parent** tout en fermant la fenêtre courante

**JFrame.DO\_NOTHING\_ON\_CLOSE**

— **Ignore** la demande de fermeture

*(quand par ex on gère « Quitter » dans un menu de fenêtre)*

## Création d'une JFrame (suite)

```
private void initComponents() {  
    JPanel cp = (JPanel) getContentPane();  
    cp.setLayout(new BorderLayout());
```

```
// CRÉATION DE LA PARTIE CONTENANT LE NOM ET LE PRÉNOM
```

```
    JPanel pann_nomPrenom = new JPanel(); // par déf., en FlowLayout centré
```

```
// Création des textes nom et prénom
```

```
    JLabel lblnom = new JLabel("Nom : ");
```

```
    pann_nomPrenom.add(lblnom);
```

```
    JTextField txtnom = new JTextField("          ");
```

... ..

```
// CRÉATION DE LA PARTIE CONTENANT LES BOUTONS
```

```
    JPanel pann_boutons = new JPanel();
```

```
    pann_boutons.add(btSupp);
```

```
    pann_boutons.add(btModif);
```

```
    pann_boutons.add(btAjout);
```

```
// on ajoute ces panneaux au contentPane de la fenetre :
```

```
    JPanel panneau = (JPanel) getContentPane(); // par default en BorderLayout
```

```
    panneau.add(pann_nomPrenom, BorderLayout.NORTH);
```

```
    panneau.add(pann_boutons, BorderLayout.SOUTH);
```

```
}
```

# Appel dans le *main*

```
// Appel  
public static void main(String args[]) {
```

```
    MaFenetre premFen = new MaFenetre();  
    premFen.pack();  
    premFen.setVisible(true);
```

*Adapter la taille de la fenêtre à ses composants*

```
} // fin du main
```

```
public static void main(String args[]) {
```

Écriture du  
main par l'IDE

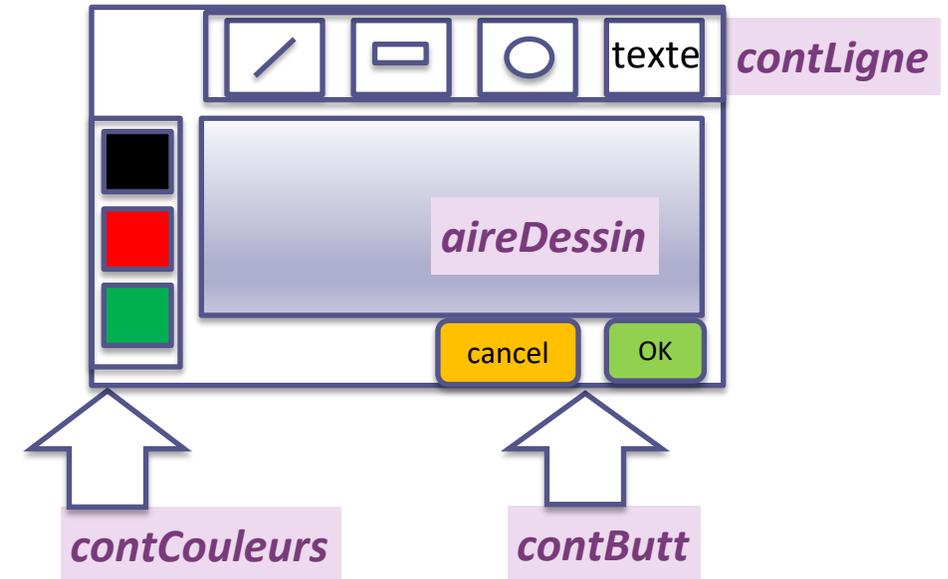
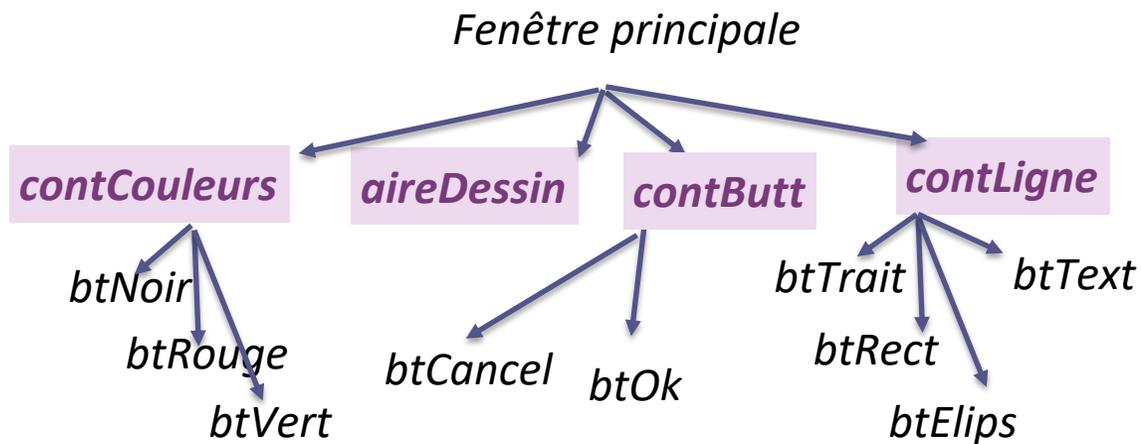
```
    /* Create and display the form */
```

```
    java.awt.EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            new MaFenetre().setVisible(true);  
        }  
    });
```

*Lancer le thread de l'interface en dernier, une fois tous les traitements effectués (sinon, IHM figée)*

# La fenêtre `java.swing.JFrame`

- Construire une IHM, c'est mettre des composants les uns à l'intérieur des autres, **dans le bon ordre**
- Dans cet **arbre d'instanciation**, la flèche signifie « contient » :



# Qu'est-ce qu'un Layout ?

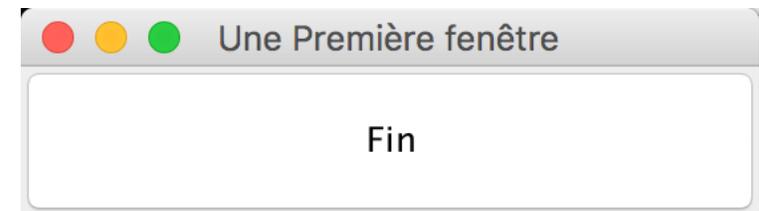
3 étapes :

1- Instanciation des **composants**

2- Récupération du **conteneur de la fenêtre**

3- Insertion des **composants** dans le **conteneur**

**Affichage :**



```
public class MaFenetre extends JFrame {
    private JButton btVoir, btDebut, btPrecedent, btSuivant, btFin;

    // Constructeur
    public MaFenetre() {
        initComponents();
        this.setTitle("Une Première fenêtre");
        this.setResizable(false);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

```
private void initComponents() {
    btVoir = new javax.swing.JButton("Voir");
    btDebut = new javax.swing.JButton("Début");
    btPrecedent = new javax.swing.JButton("Précédent");
    btSuivant = new javax.swing.JButton("Suivant");
    btFin = new javax.swing.JButton("Fin");
```

```
    Container cp = this.getContentPane();
    cp.add(btVoir);
    cp.add(btDebut);
    cp.add(btPrecedent);
    cp.add(btSuivant);
    cp.add(btFin);
```

1

2

3

# Répartition des composants

- PROBLÈME : la méthode `add` de `JPanel` ajoute toujours le composant **au même endroit** dans le conteneur
  - Ainsi lorsqu'on veut ajouter plusieurs composants dans le panel, seul le dernier composant apparaît
  - Il faut donc **répartir** les composants avec la méthode `setLayout()`
- Pour gérer la disposition des composants  
Il existe des modes de répartition *prédéfinis*  
On choisit le **gestionnaire de répartition** avec `setLayout(monLayout)`
- Les gestionnaires de répartition appartiennent au package `java.awt`

