2ème Ingéniorat en Informatique

Programmation Orientée Objet II

Chapitre III
Programmation événementielle

2023/2024

Dr. HANNOUSSE Abdelhakim – hannousse.abdelhakim@univ-guelma.dz

Contenu du chapitre

- Introduction
- Les objets graphiques en Java
  - Hiérarchie des composants graphiques
  - Composants graphiques principaux
  - Gestionnaires de disposition
  - Quelques objets graphiques outils
- Écouteurs et gestionnaires des événements
  - Écouteurs d'événements
    - Écouteurs simples
    - Écouteurs de plusieurs composants
    - Auto-écouteurs
  - Gestionnaires des événements

Introduction (1/2)

- La programmation événementielle se concentre sur la manipulation d'événements:
  - *Les interactions utilisateur*: clics de souris, frappes de clavier
  - *Les signaux système*: connecter, ou déconnecter un périphérique
  - *Les changements d'état des objets*: modification du solde d'un compte bancaire
- En programmation orientée objet, un **événement** est généralement associé à un objet et peut être déclenché par différentes interactions ou conditions dans le système.
  - *Changement d'état d'objet*: Le patrons observateur est très utile, lorsque l'événement se produit, le sujet notifie tous ses observateurs, qui peuvent alors réagir en conséquence.
  - *Action sur un objet:* La programmation événementielle constitue la caractéristique essentielle des interfaces graphiques.

Introduction (2/2)

- Un système de gestion des événements est constitué de:
  - Un **déclencheur d'évènement**, qui peut être, un utilisateur, système ou un objet spécifique.
  - Un **gestionnaire d'événements** est une entité (classe/objet) encapsulant une ou plusieurs méthodes qui sont appelées en réponse à des événements spécifiques.
  - Les **écouteurs d'événements** sont des objets permettant de détecter et éventuellement de répondre à des événements spécifiques.
- Certains systèmes permettent la propagation d'événements, où un événement est transmis à travers une hiérarchie d'objets.

Les composants graphiques en Java

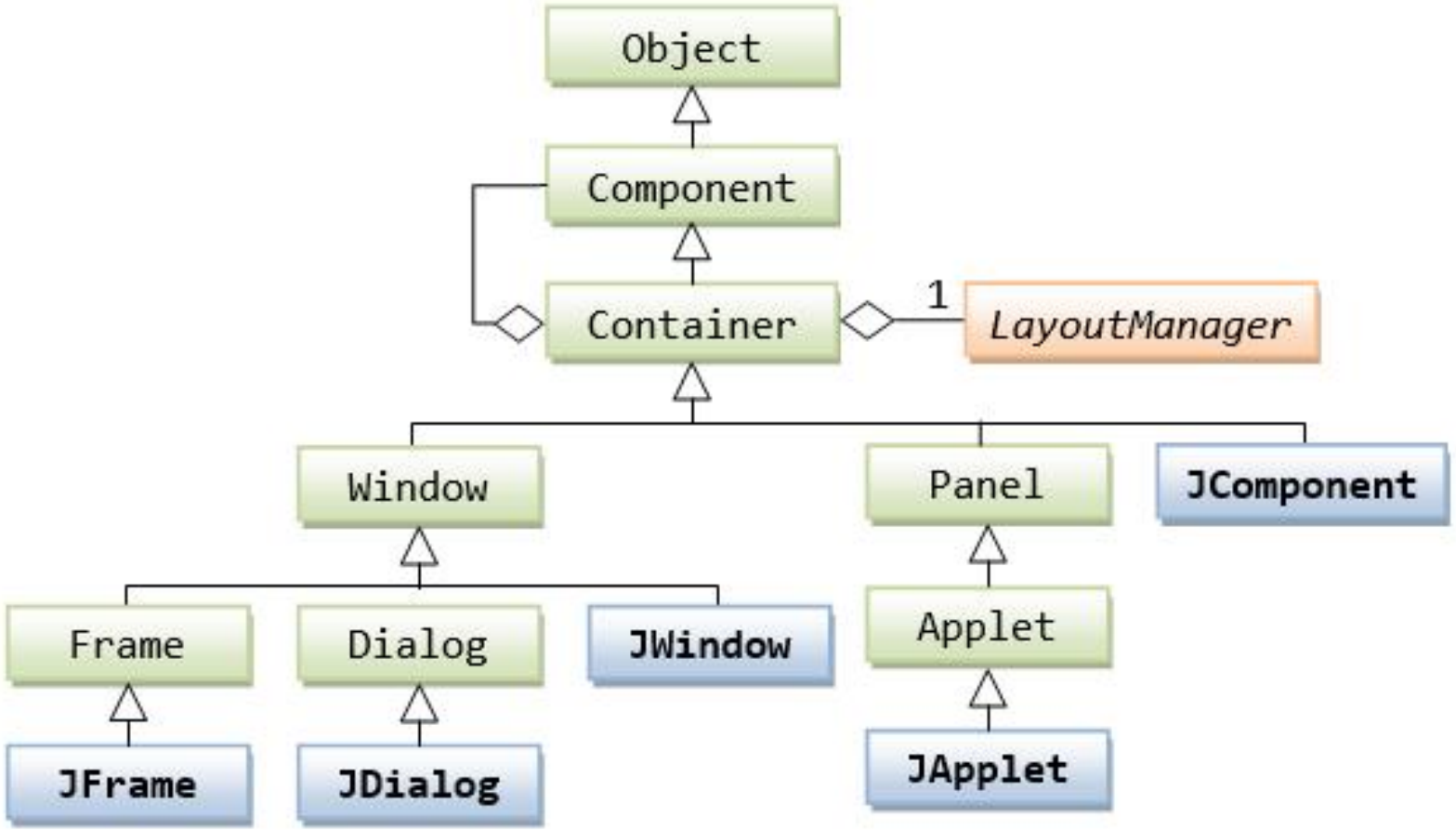
Programmation événementielle en Java (1/2)

- En Java, la programmation événementielle est couramment utilisée dans les bibliothèques **AWT** et **Swing** pour créer des interfaces graphiques utilisateur (**GUI**).
- **AWT** est la première bibliothèque graphique introduite dans Java. Elle utilise les composants ***natifs*** du système d'exploitation pour créer des interfaces graphiques.
- Les composants **AWT** dépendent du système d'exploitation sous-jacent, ce qui peut entraîner des différences d'apparence d'une plate-forme à l'autre.
- Les composants graphiques du **AWT** peuvent être divisés en deux catégories :
  - ***Lightweight components (composants légers):*** composants d'interface utilisateur qui sont entièrement gérés par Java, sans dépendance directe des composants natifs du système d'exploitation sous-jacent.
  - ***Heavyweight components (composants lourds):*** des composants qui délèguent leur rendu et leur gestion des événements aux composants natifs du système d'exploitation sur lequel le programme Java est exécuté.

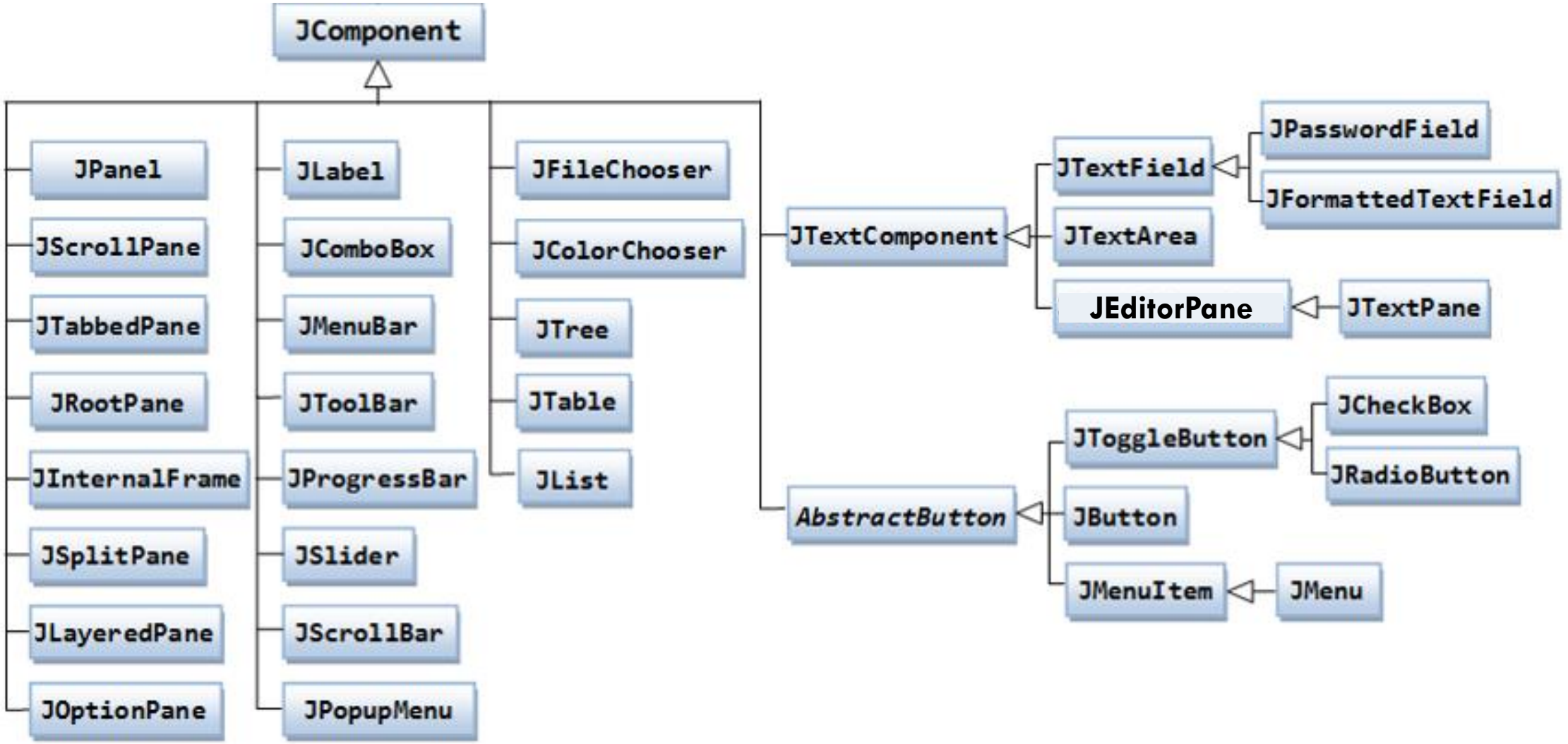
Programmation événementielle en Java (2/2)

- **Swing** est une extension d'**AWT** qui a été introduite plus tard pour fournir une bibliothèque graphique plus riche et plus flexible.
- **Swing** utilise ses propres composants, indépendants du système d'exploitation, assurant une apparence et un comportement cohérents sur toutes les plateformes.
- Avec **Swing**, il est aussi possible de spécifier l'aspect visuel d'un système d'exploitation particulier en utilisant le concept de look and feel (L&F) - apparence et comportement.
- Par défaut, Swing utilise un L&F propre à Java, appelé ***Metal***. Cependant, il est possible de spécifier le L&F d'un système d'exploitation spécifique en utilisant la méthode ***UIManager.setLookAndFeel()***.
  - Look and feel natif de Windows: **com.sun.java.swing.plaf.windows.WindowsLookAndFeel**
  - Look and feel natif de MacOs: **com.apple.laf.AquaLookAndFeel**

Hiérarchie des composants graphiques (1/2)

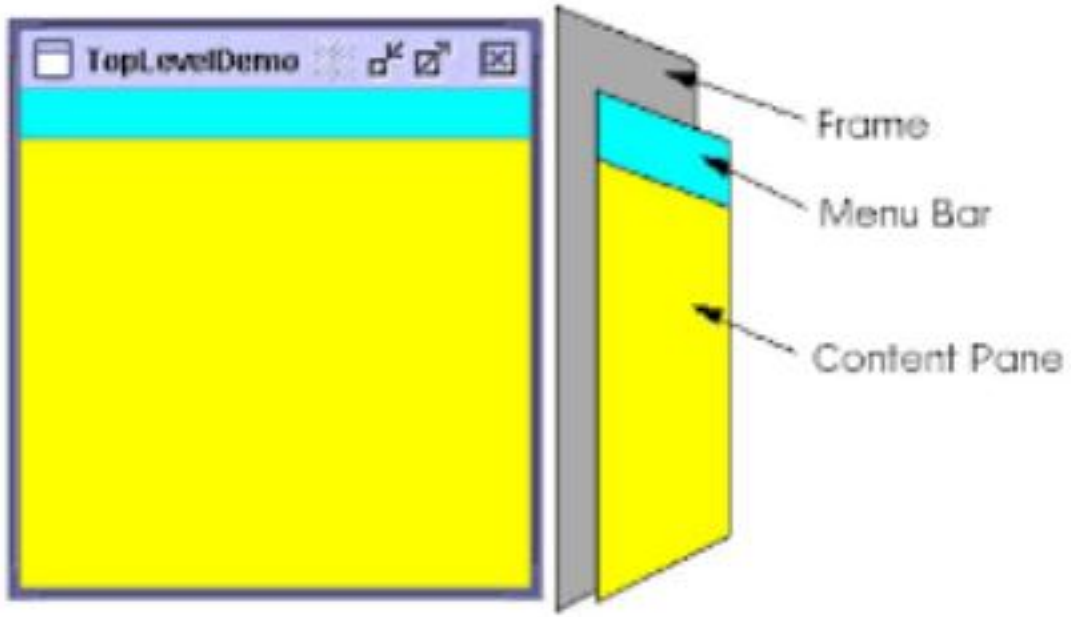


Hiérarchie des composants graphiques (2/2)



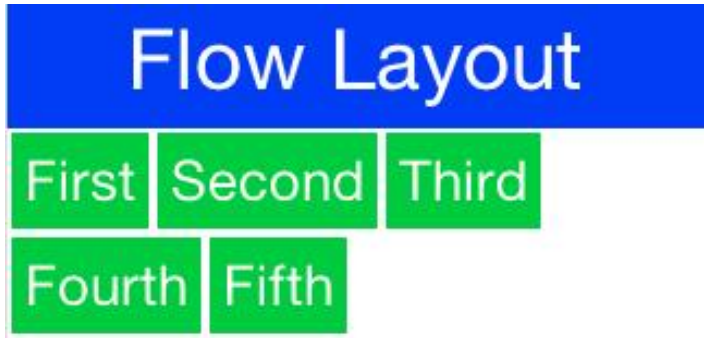
Composants graphiques principaux en Java

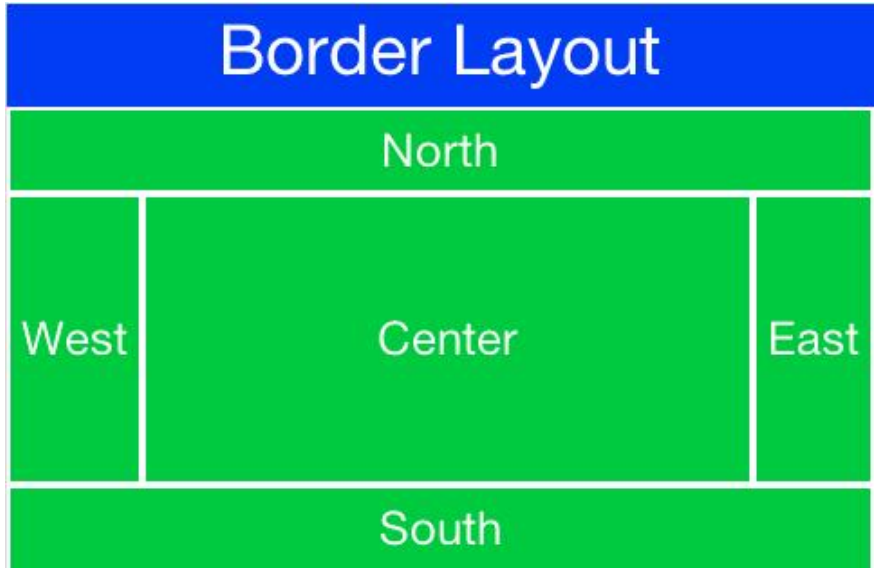
- Dans une application Java, les composants graphiques sont organisés dans une hiérarchie, selon l'ordre suivant:
  - **Fenêtre ([J]Frame)**: Une fenêtre avec une bordure et une barre de titre. Elle comporte des sous-éléments appelés **Panes**. La plus part du temps, on utilise le **ContentPane**, c'est là où on met les composants sur la fenêtre.
  - **Conteneur (Container)**: est un composant qui peut contenir d'autres composants tels que des boutons, des champs de texte, etc. Un exemple des conteneur est ***JPanel*** qui peut être ajouter au ***contentPane***.



Gestionnaires de disposition (Layout)

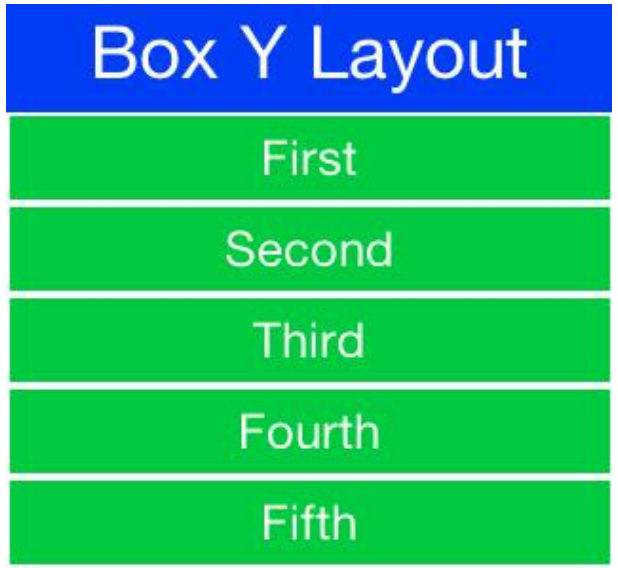
- **Gestionnaire de disposition (Layout)**: est responsable de la disposition spatiale des composants au sein d'un conteneur. Il décide de la manière dont les composants seront positionnés et redimensionnés.
  - **FlowLayout :** organise les composants dans une seule ligne, ajoutant une nouvelle ligne si l'espace est insuffisant.
  - **BorderLayout :** divise le conteneur en cinq régions (Nord, Sud, Est, Ouest, Centre) et place les composants en conséquence.

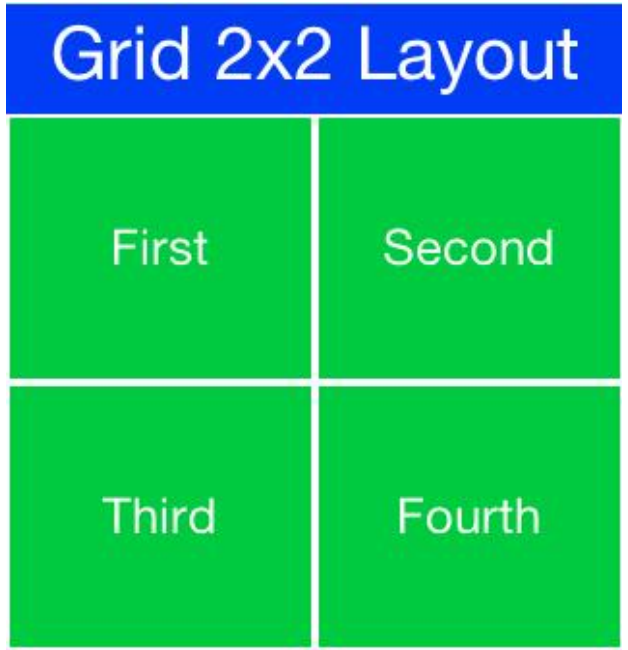




Gestionnaires de disposition (Layout)

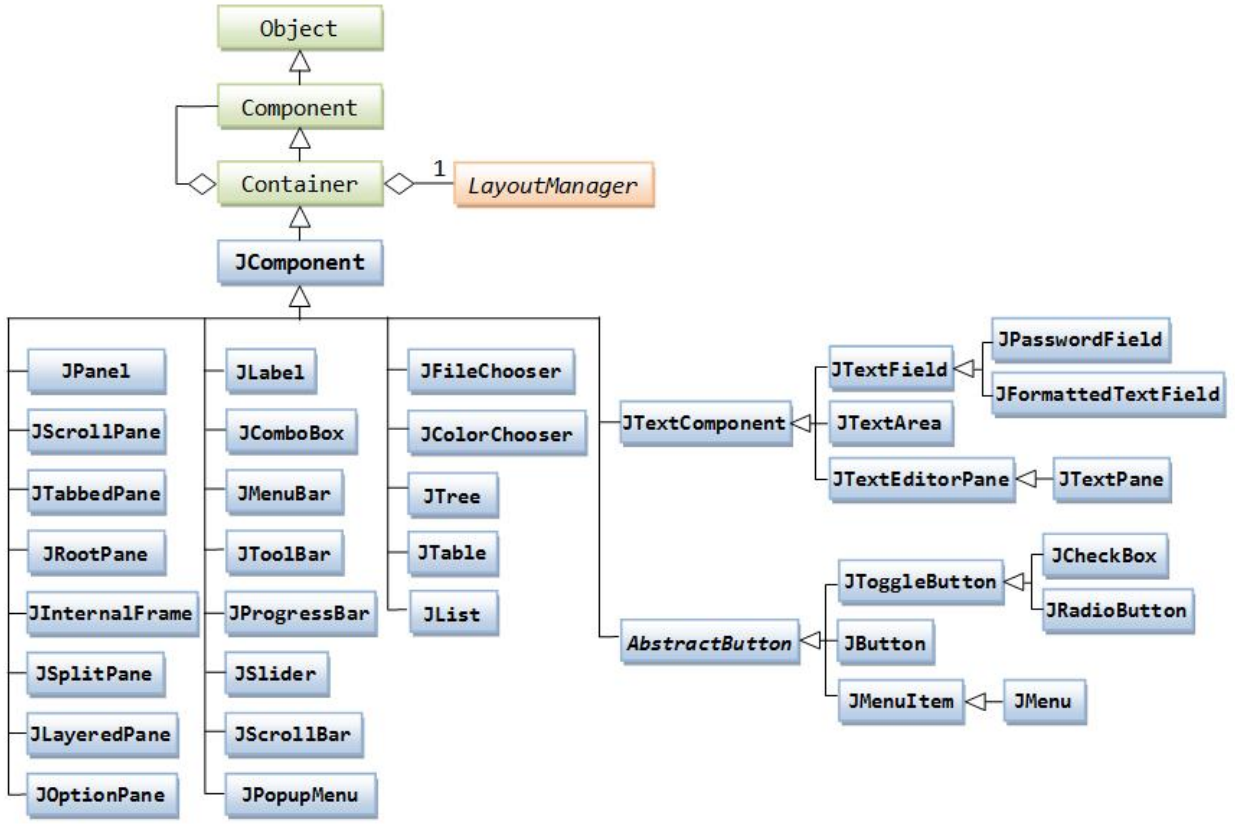
- **GridLayout :** organise les composants dans une grille de cellules.
- **BoxLayout :** organise les composants dans une seule linge horizontal ou vertical, de manière séquentielle.





Composants graphiques en Java

- **Composant ([J]Component):** est l'éléments de base qui peut être ajouté à un conteneur. Ils peut être un composant léger (lightweight) de Swing ou un composant lourd (heavyweight) de AWT.
  - ***JOptionPane***: affichage d'une boite de dialogue avec un message.
  - **JFrame:** utilisé pour créer une fenêtre graphique où les composants graphiques tels que des boutons, des champs de texte, etc. peuvent être affichés.



Composants graphiques en Java – Fenêtre

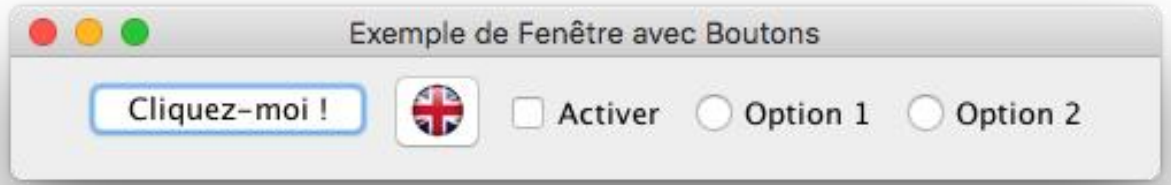
- **JFrame:** crée une fenêtre graphique avec un barre de titre e
  de 3 boutons (fermeture, réduction et d'agrandissement).
  - **setBackground(Color):** définir sa couleur de fond.
  - **setTitle(String):** définir le titre d'une fenêtre.
  - **setBounds(x, y, largeur, hauteur):** définir sa position et sc
    taille.
  - **setDefaultCloseOperation(int):** définir le comportement
    par défaut de la fenêtre lorsque l'utilisateur clique sur le
    bouton de fermeture:
    - EXIT_ON_CLOSE
    - HIDE_ON_CLOSE
    - DISPOSE_ON_CLOSE
    - DO_NOTHING_ON_CLOSE
  - **setVisible(boolean):** rendre la fenêtre visible ou pas.

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

public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setBackground(Color.BLUE);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("Ma Fenêtre Swing");
        frame.setBounds(100, 100, 300, 200);
        frame.setVisible(true);
    }
}
```

Composants graphiques en Java – Boutons

- Les boutons peuvent être créés en utilisant la classe **JButton.** Il existe plusieurs types de boutons:
  - **Bouton Standard (JButton) :** bouton de base.
  - **Bouton avec Icône (JButton avec ImageIcon) :** un bouton avec une icône à la place du texte en utilisant ImageIcon.
  - **Bouton de Case à Cocher (JCheckBox) :** un bouton qui peut être coché ou décoché.
  - **Bouton Radio (JRadioButton) :** utilisé dans des groupes où un seul bouton peut être sélectionné à la fois.

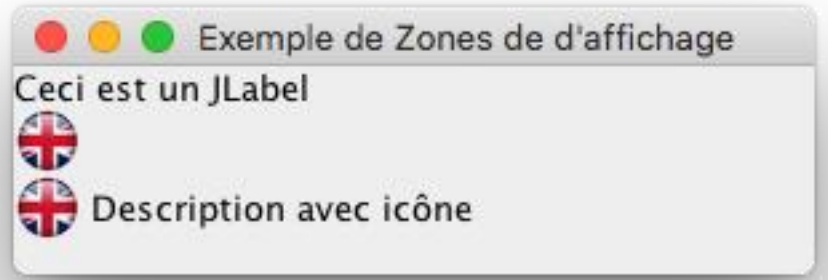
```java
import javax.swing.*;
import java.awt.*;

public class Main {
    public static void main(String[] args) {
        JFrame maFenetre = new JFrame("Exemple de Fenêtre avec Boutons");
        maFenetre.setBounds(100, 100, 500, 75);
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton boutonStandard = new JButton("Cliquez-moi !");
        ImageIcon icone = new ImageIcon("en.gif");
        JButton boutonAvecIcone = new JButton(icone);
        JCheckBox caseACocher = new JCheckBox("Activer");
        JRadioButton boutonRadio1 = new JRadioButton("Option 1");
        JRadioButton boutonRadio2 = new JRadioButton("Option 2");
        ButtonGroup groupeRadio = new ButtonGroup();
        groupeRadio.add(boutonRadio1); groupeRadio.add(boutonRadio2);
        maFenetre.setLayout(new FlowLayout());
        maFenetre.add(boutonStandard); maFenetre.add(boutonAvecIcone);
        maFenetre.add(caseACocher); maFenetre.add(boutonRadio1);
        maFenetre.add(boutonRadio2);
        maFenetre.setVisible(true);
    }
}
```



Composants graphiques en Java – Zones d'affichage

- Les zones d'affichage (**JLabel**) permettent l'affichage du textes, icônes ou images dans une interface.

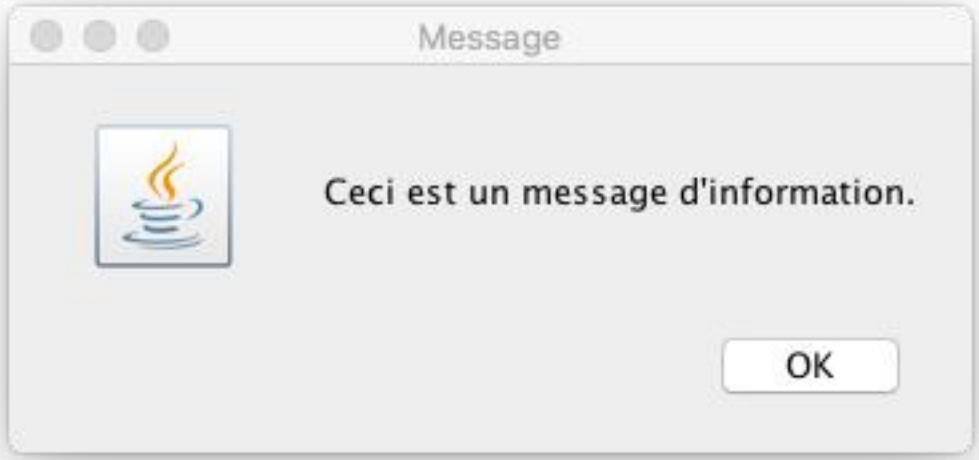
```java
import javax.swing.*;

public class Main {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame();
        fenetre.setBounds(100,100, 300, 100);
        fenetre.setTitle("Exemple de Zones de d'affichage");
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel labelTexte = new JLabel("Ceci est un JLabel");
        ImageIcon icone = new ImageIcon("en.gif");
        JLabel labelIcone = new JLabel(icone);
        JLabel labelTexteIcone = new JLabel("Description avec icône", icone, JLabel.CENTER);
        fenetre.setLayout(new BoxLayout(fenetre.getContentPane(), BoxLayout.Y_AXIS));
        fenetre.add(labelTexte);
        fenetre.add(labelIcone);
        fenetre.add(labelTexteIcone);
        fenetre.setVisible(true);
    }
}
```



Composants graphiques en Java – Boites de dialogue

- **Boite de dialogue (JOptionPane):** fournit un ensemble de méthodes statiques pour créer et gérer différents types de boîtes de dialogue.
  - **showMessageDialog(parent, message):** affiche un message d'information.

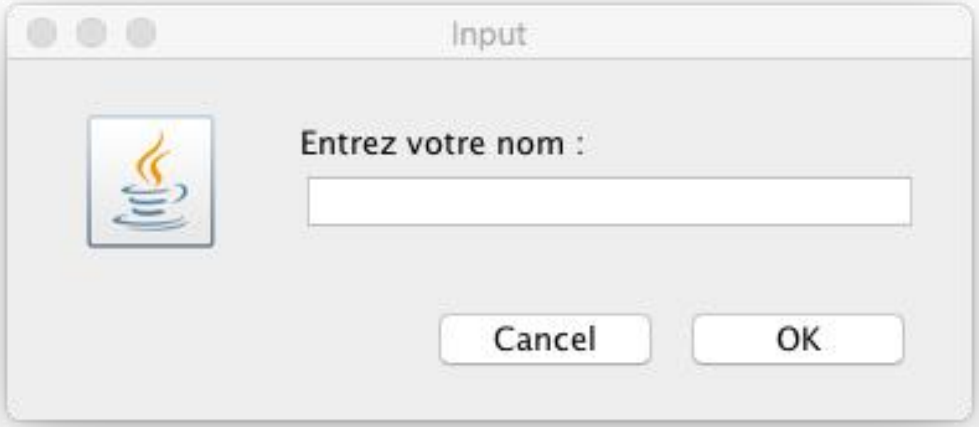
```
JOptionPane.showMessageDialog(null, "Ceci est un message d'information.");
```



Composants graphiques en Java – Boites de dialogue

- ***JOptionPane*: fournit un ensemble de méthodes statiques pour créer et gérer différents types de boîtes de dialogue.**
  - **showInputDialog(message) :** affiche une boîte de dialogue demandant une entrée de l'utilisateur.

```
String userInput = JOptionPane.showInputDialog("Entrez votre nom :");
```



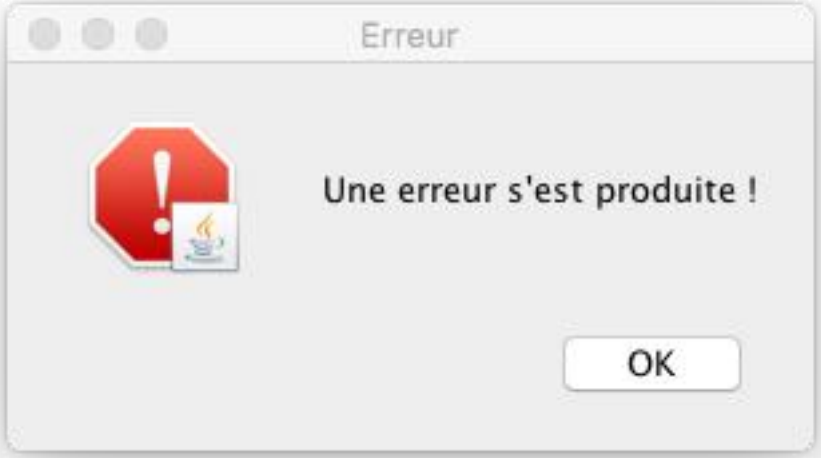
Composants graphiques en Java – Boites de dialogue

- ***JOptionPane*: fournit un ensemble de méthodes statiques pour créer et gérer différents types de boîtes de dialogue.**
  - **showConfirmDialog(parent, message, titre, type)** : Affiche une boîte de dialogue avec des options (type):
    - DEFAULT_OPTION
    - YES_NO_OPTION
    - YES_NO_CANCEL_OPTION
    - OK_CANCEL_OPTION


```
int resultat = JOptionPane.showConfirmDialog(null,"Voulez-vous continuer ?","Confirmation",JOptionPane
    .YES_NO_CANCEL_OPTION);
```

Composants graphiques en Java – Boites de dialogue

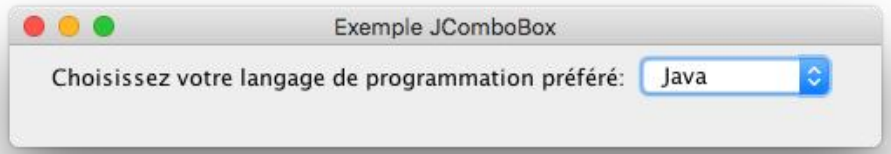
- ***JOptionPane*: fournit un ensemble de méthodes statiques pour créer et gérer différents types de boîtes de dialogue.**
  - **showMessageDialog(parent, message, titre, type) :** affiche un message d'alerte, avec un type spécifique:
    - ERROR_MESSAGE
    - INFORMATION_MESSAGE
    - WARNING_MESSAGE
    - QUESTION_MESSAGE
    - PLAIN_MESSAGE




```
JOptionPane.showMessageDialog(null,"Une erreur s'est produite !", "Erreur",JOptionPane.ERROR_MESSAGE);
```

Composants graphiques en Java – JComboBox

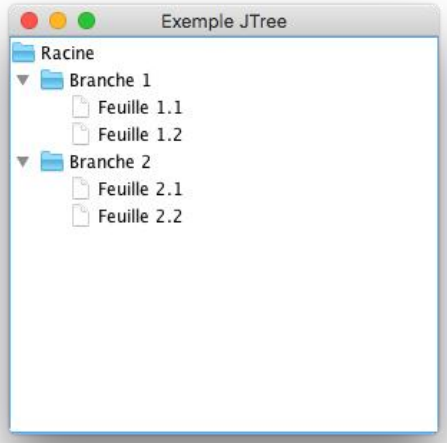
- **JComboBox:** composant qui qui permet à l'utilisateur de choisir un élément à partir d'une liste déroulante. Cela offre une interface utilisateur compacte pour la sélection d'éléments parmi plusieurs options.




```java
import javax.swing.*;

public class ExempleJComboBox {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame("Exemple JComboBox");

        // Données pour la JComboBox
        String[] langages = {"Java", "Python", "C++", "JavaScript"};

        // Création d'un JLabel pour indiquer le but de la sélection
        JLabel label = new JLabel("Choisissez votre langage de programmation préféré:");

        // Création de la JComboBox avec les options
        JComboBox<String> comboBox = new JComboBox<>(langages);

        // Création d'un panneau pour organiser les composants
        JPanel panel = new JPanel();
        panel.add(label);
        panel.add(comboBox);

        // Ajout du panneau à la fenêtre
        fenetre.add(panel);

        fenetre.setBounds(100, 100, 500, 80);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setVisible(true);
    }
}
```

Composants graphiques en Java – JTree

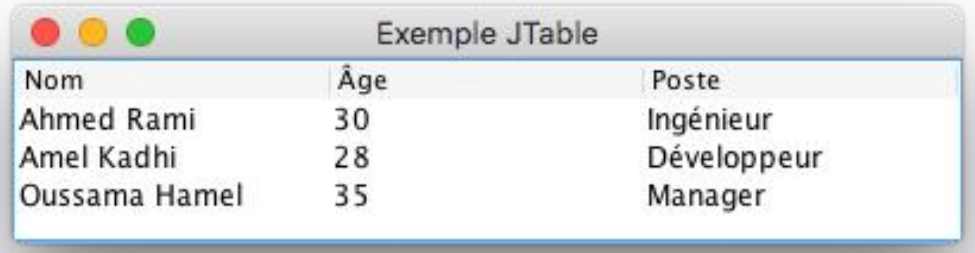
- **JTree:** composant qui permet de créer des arbres hiérarchiques. Il est couramment utilisé pour afficher des données de manière organisée et structurée sous forme d'arborescence. Chaque nœud de l'arbre peut avoir un nombre arbitraire de nœuds enfants, créant ainsi une structure hiérarchique.




```java
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

public class ExempleJTree {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame();
        DefaultMutableTreeNode racine = new DefaultMutableTreeNode("Racine");

        DefaultMutableTreeNode branche1 = new DefaultMutableTreeNode("Branche 1");
        branche1.add(new DefaultMutableTreeNode("Feuille 1.1"));
        branche1.add(new DefaultMutableTreeNode("Feuille 1.2"));

        DefaultMutableTreeNode branche2 = new DefaultMutableTreeNode("Branche 2");
        branche2.add(new DefaultMutableTreeNode("Feuille 2.1"));
        branche2.add(new DefaultMutableTreeNode("Feuille 2.2"));

        racine.add(branche1);
        racine.add(branche2);
        DefaultTreeModel modeleArbre = new DefaultTreeModel(racine);
        JTree arbre = new JTree(modeleArbre);

        fenetre.add(new JScrollPane(arbre));

        fenetre.setTitle("Exemple JTree");
        fenetre.setBounds(100, 100, 300, 300);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setVisible(true);
    }
}
```

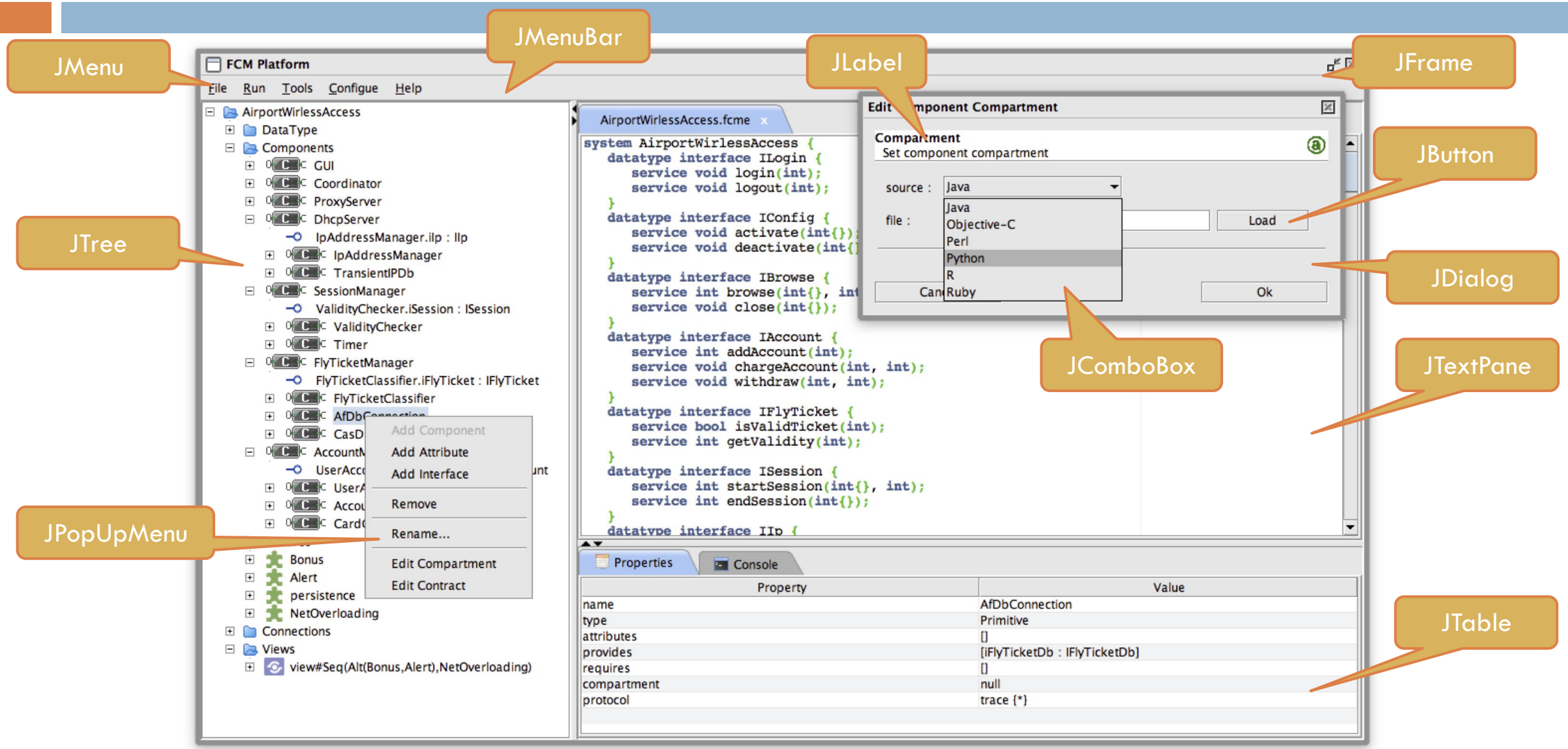
Composants graphiques en Java – JTree

- **JTable:** composant permettant d'afficher des données sous forme de tableau. Il offre une grande flexibilité pour présenter des données tabulaires, que ce soit pour des applications de gestion, des outils de visualisation de données, ou d'autres applications nécessitant un affichage organisé en lignes et colonnes.




```java
import javax.swing.*;
import javax.swing.table.DefaultTableModel;

public class ExempleJTable {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame("Exemple JTable");

        // Données pour la table
        Object[][] donnees = {
                {"Ahmed Rami", 30, "Ingénieur"},
                {"Amel Kadhi", 28, "Développeur"},
                {"Oussama Hamel", 35, "Manager"}
        };

        // Noms des colonnes
        String[] colonnes = {"Nom", "Âge", "Poste"};

        // Création d'un modèle de table par défaut
        DefaultTableModel modeleTable = new DefaultTableModel(donnees, colonnes);

        // Création de la JTable avec le modèle
        JTable table = new JTable(modeleTable);

        JScrollPane scrollPane = new JScrollPane(table);

        fenetre.add(scrollPane);
        fenetre.setBounds(100, 100, 400, 100);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setVisible(true);
    }
}
```

Composants graphiques en Java – Exemple complet (Hannousse'19)



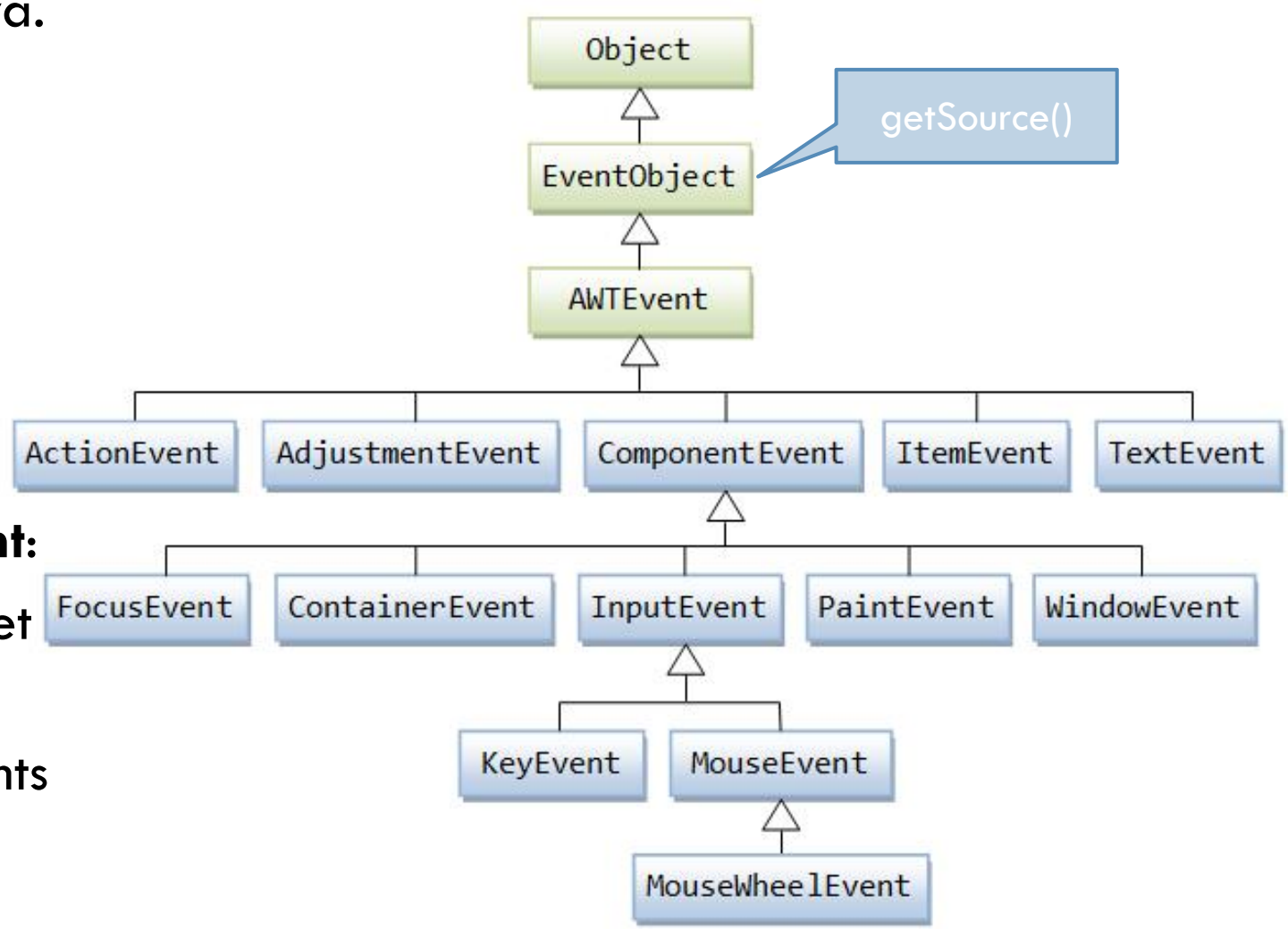
Écouteurs et gestionnaires des évènements

Les événements en Java (1/3)

- Les événements sont des actions qui se produisent pendant l'exécution d'une application, et qui peuvent être gérés par le programme pour créer des composants graphiques interactifs.
  - **Événements de souris :** déclenchés par les actions de l'utilisateur avec la souris, comme cliquer, faire glisser ou déplacer la souris.
  - **Événements de clavier :** déclenchés par les actions de l'utilisateur avec le clavier, telles que l'appui sur une touche ou la libération d'une touche.
  - **Événements de composant :** déclenchés par des actions sur les composants graphiques simples, tels que le changement de texte dans un champ de texte ou la sélection d'un élément dans une liste déroulante.
  - **Événements de fenêtre :** liés aux actions de l'utilisateur sur les fenêtres de l'interface utilisateur, telles que l'ouverture, la fermeture, la minimisation ou la maximisation d'une fenêtre.
  - **Événements de focus :** déclenchés lorsque le focus du clavier est déplacé d'un composant à un autre.

Les événements en Java (2/3)

- Un événement est un objet en Java.
- AWT offre la grande partie des événements sur le package **java.awt.AWTEvent**
- **Swing** utilise généralement les événements **AWT.**
- Il propose d'autres évènements spécifiques sur **javax.swing.event:**
  - **MenuEvent:** liés à l'ouverture et fermeture d'un Menu.
  - **CaretEvent:** liés aux mouvements du curseur.



Les événements en Java (3/3)
Exemple du KeyEvent

- Les événements de **KeyEvent** sont utilisés pour détecter les interactions de l'utilisateur avec le clavier. Ces événements comprennent les touches enfoncées, les touches relâchées et les caractères saisis.
- Des codes sont associés à chaque touche du clavier, des constantes entières définies dans la classe **KeyEvent.** Ces constantes sont utilisées pour identifier la touche qui a déclenché l'événement en appelant la méthode ***getKeyCode()***, et ***getKeyText()***:
  - **KeyEvent.VK_A** : Code pour la touche "A".
  - **KeyEvent.VK_ENTER** : Code pour la touche "Entrée".
  - **KeyEvent.VK_ESCAPE** : Code pour la touche "Échap".
  - **KeyEvent.VK_UP** : Code pour la touche de direction "Haut".
  - **KeyEvent.VK_SPACE** : Code pour la barre d'espace.
  - **KeyEvent.VK_SHIFT, KeyEvent.VK_CONTROL, KeyEvent.VK_ALT** : Ces constantes représentent les touches de modification telles que **Maj, Ctrl** et **Alt.**

Écouteurs et gestionnaires des événements en Java (1/5)

- **Écouteurs d'événements :** objets notifiés lorsqu'un événement se produit, déclenchant l'exécution d'un gestionnaire d'événements associé.
- **Gestionnaires d'événements:** objets responsables de la gestion des événements déclenchés par des écouteurs. Lorsqu'un événement survient, le gestionnaire d'événements associé prend en charge l'exécution du code spécifique lié à cet événement.
- En Java, les écouteurs sont des interfaces déclarant des méthodes liées à des événements spécifiques. Ces interfaces font partie du package **java.awt.event** pour les événements **AWT**, et de **javax.swing.event** pour les événements **Swing**.
- Chaque écouteur correspond a un évènement particulier.
- Les gestionnaires d'événements sont généralement des méthodes implémentées par des classes implémentant des écouteurs des évènements.

Écouteurs et gestionnaires des événements en Java (2/5)

Action	Évènement	Écouteur
Clic sur un bouton	ActionEvent	ActionListener
Ouvrir, fermer, réduire une fenêtre	WindowEvent	WindowListener
Clic sur un composant	MouseEvent	MouseListener
Changement du texte dans un JTextField	TextEvent	TextListener
Appui sur une touche clavier	KeyEvent	KeyListener
Clic/Sélection d'un élément en JCheckbox, JRadioButton, JComboBox	ItemEvent, ActionEvent	ItemListener, ActionListener

Écouteurs et gestionnaires des événements en Java (3/5)

- Tous les composants graphiques du Java sont des sources d'événements possibles.
- Il existe différents types d'écouteurs d'événements en Java:
  - **KeyListener :** détecteur des événements liés au clavier, tels que les pressions de touches.
    - *KeyTyped(KeyEvent key), KeyPressed(KeyEvent key), KeyReleased(KeyEvent key)*
  - **MouseListener :** détecteur des événements de la souris sur les composants, tels que les clics, déplacements, etc.
    - *mouseClicked(MouseEvent e), mousePressed(MouseEvent e), mouseReleased(MouseEvent e), mouseEntered(MouseEvent e), et mouseExited(MouseEvent e)*
  - **ActionListener :** détecteur des événements d'action générés par les composants graphiques, tels que les bouton, menu, etc.
    - *actionPerformed(ActionEvent e)*

Les écouteurs et gestionnaires des événements en Java (4/5)

- Pour éviter de fournir une implémentation pour chaque méthode de l'interface. Java fournit les classes ***KeyAdapter*** et ***MouseAdapter***, qui sont des implémentations par défaut de ces interfaces avec toutes les méthodes vides. L'utilisateur peut donc substituer uniquement les méthodes dont il a besoin.
- Chaque type de composant propose des méthodes permettant à un objet écouteur de s'enregistrer (ou de se dés-enregistrer) auprès de lui. Pour ajouter un gestionnaire d'événements à un composant c, la méthode c.**add???Listener(…)** est utilisée.
- Un objet peut écouter plusieurs composants.
- Un composant peut s'écouter lui-même.
- On utilise souvent des classes internes/anonymes pour les écouteurs.

Les écouteurs et gestionnaires des événements en Java (5/5)

- Pour gérer un événement dans Java:
  - Identifier l'objet source de l'événement.
  - Identifier le type de l'événement.
  - Écrire une classe qui implémente une l'interface modélisant l'écouteur qui correspond.
  - Implémenter le(s) gestionnaire(s) de l'événement associé.
  - Attacher un objet de la classe écouteur à l'objet source en utilisant sa méthode **add???Listener(…)**

Les écouteurs et gestionnaires des événements en Java – Exemple (1/2)

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

class MonGestionnaire implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Bouton cliqué !");
    }
}

public class Main {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame("Exemple avec Gestionnaire");
        JButton bouton = new JButton("Cliquez-moi !");

        MonGestionnaire gestionnaire = new MonGestionnaire();
        bouton.addActionListener(gestionnaire);

        fenetre.setLayout(new FlowLayout());

        fenetre.add(bouton);
        fenetre.setBounds(100, 100, 200, 150);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setVisible(true);
    }
}
```

Solution avec une classe normale

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

public class Main {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame("Exemple avec Gestionnaire");
        JButton bouton = new JButton("Cliquez-moi !");

        // Utilisation d'une classe anonyme pour définir l'ActionListener
        bouton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "Bouton cliqué !");
            }
        });

        // Utilisation de FlowLayout pour organiser les composants
        fenetre.setLayout(new FlowLayout());

        fenetre.add(bouton);
        fenetre.setBounds(100, 100, 200, 150);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setVisible(true);
    }
}
```

Solution avec une classe anonyme

Les écouteurs et gestionnaires des événements en Java – Exemple (2/2)

```java
import javax.swing.*;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class EcouterPlusieursComposants {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame("Exemple Multiple Composants");

        JButton bouton1 = new JButton("Bouton 1");
        JButton bouton2 = new JButton("Bouton 2");

        // Un seul objet écoute plusieurs composants (boutons)
        ActionListener monEcouteur = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "Bouton cliqué !");
            }
        };

        // Enregistrement du même écouteur pour les deux boutons
        bouton1.addActionListener(monEcouteur);
        bouton2.addActionListener(monEcouteur);

        // Utilisation de FlowLayout pour organiser les composants
        fenetre.setLayout(new FlowLayout());

        fenetre.add(bouton1);
        fenetre.add(bouton2);

        fenetre.setBounds(100, 100, 300, 150);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setVisible(true);
    }
}
```

Écouteur de plusieurs composants

```java
import javax.swing.*;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class BoutonAutoEcouteur extends JButton implements ActionListener {
    public BoutonAutoEcouteur(String texte) {
        super(texte);
        // Le bouton est son propre écouteur
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Bouton cliqué !");
    }
}

public class ExempleAutoEcouteur {
    public static void main(String[] args) {
        JFrame fenetre = new JFrame("Exemple Auto-Écouteur");

        // Utilisation de notre classe personnalisée
        BoutonAutoEcouteur bouton = new BoutonAutoEcouteur("Cliquez-moi !");

        // Utilisation de FlowLayout pour organiser les composants
        fenetre.setLayout(new FlowLayout());

        fenetre.add(bouton);

        fenetre.setBounds(100, 100, 200, 150);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setVisible(true);
    }
}
```

Auto-écouteur