



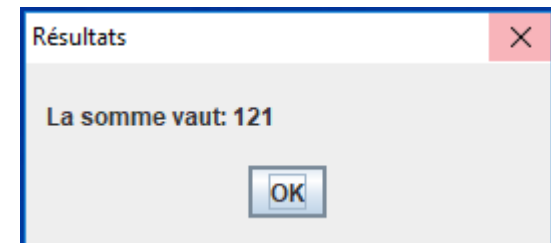
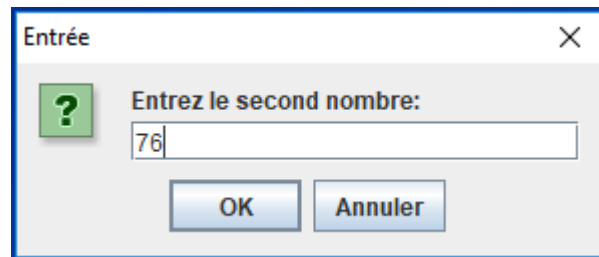
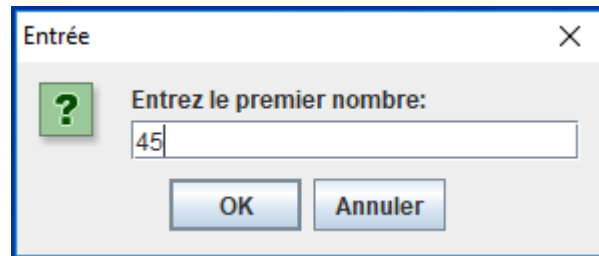
Chapitre 7

Composants Swing

Interface utilisateur graphique et gestion des événements

1. JOptionPane

JOptionPane



```
import javax.swing.*;
```

```
public class Principal {
```

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

```
        int nombre1, nombre2, total;
```

```
        String premier = JOptionPane.showInputDialog ("Entrez le premier nombre:");
```

```
        String second = JOptionPane.showInputDialog ("Entrez le second nombre: ");
```

```
        nombre1 = Integer.parseInt(premier); —————> Transforme String en entier
```

```
        nombre2 = Integer.parseInt(second);
```

```
        total = nombre1 + nombre2;
```

Contenu de la boîte de dialogue

```
JOptionPane.showMessageDialog (null, "La somme vaut: " + total,
                                "Résultats", JOptionPane.PLAIN_MESSAGE);
```

icône



Titre de la boîte de dialogue

```
        System.exit(0);
```

```
    }
```

```
}
```

JOptionPane

JOptionPane.QUESTION_MESSAGE



JOptionPane.ERROR_MESSAGE



JOptionPane.INFORMATION_MESSAGE



JOptionPane.WARNING_MESSAGE



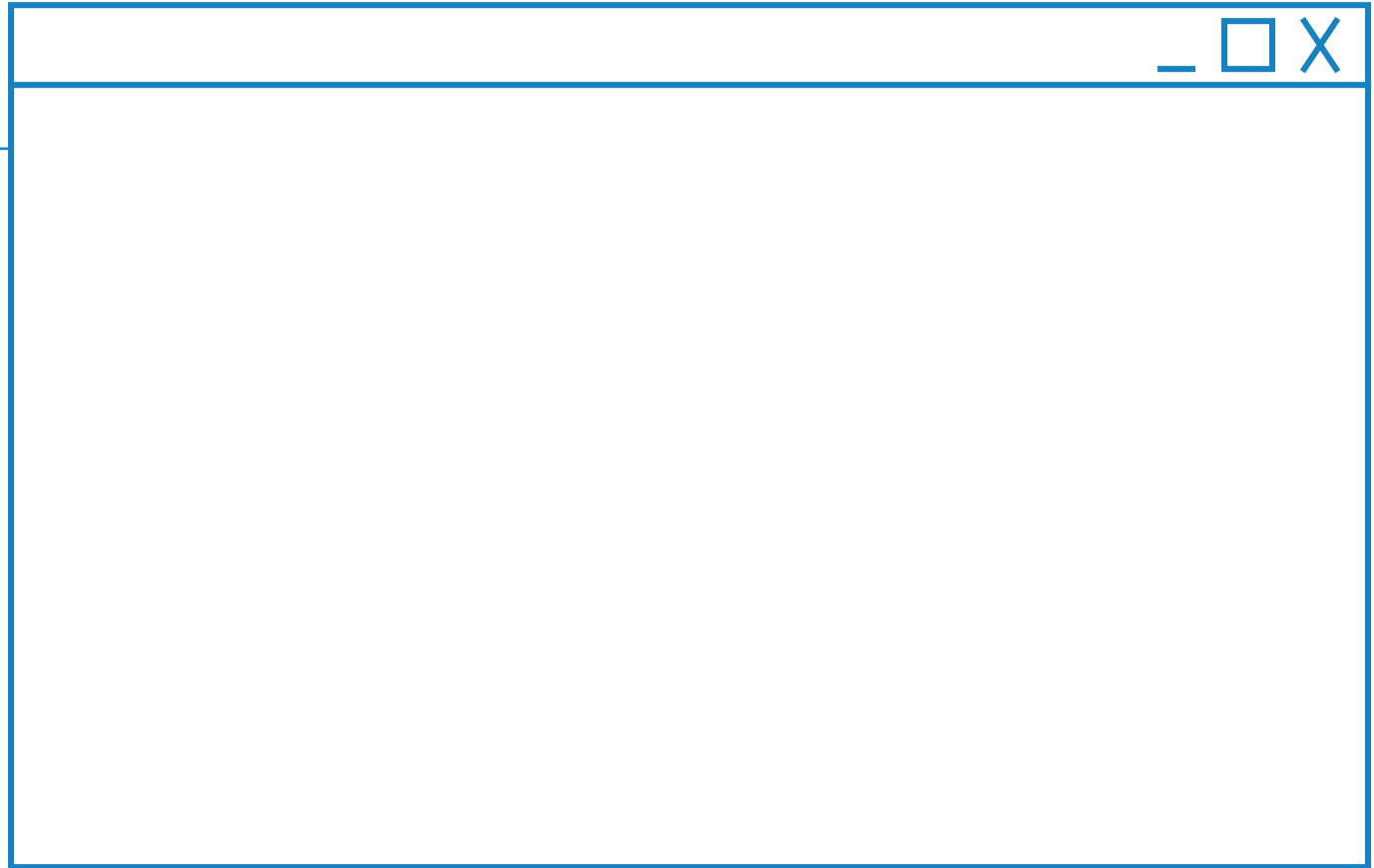
Swing

1. JOptionPane

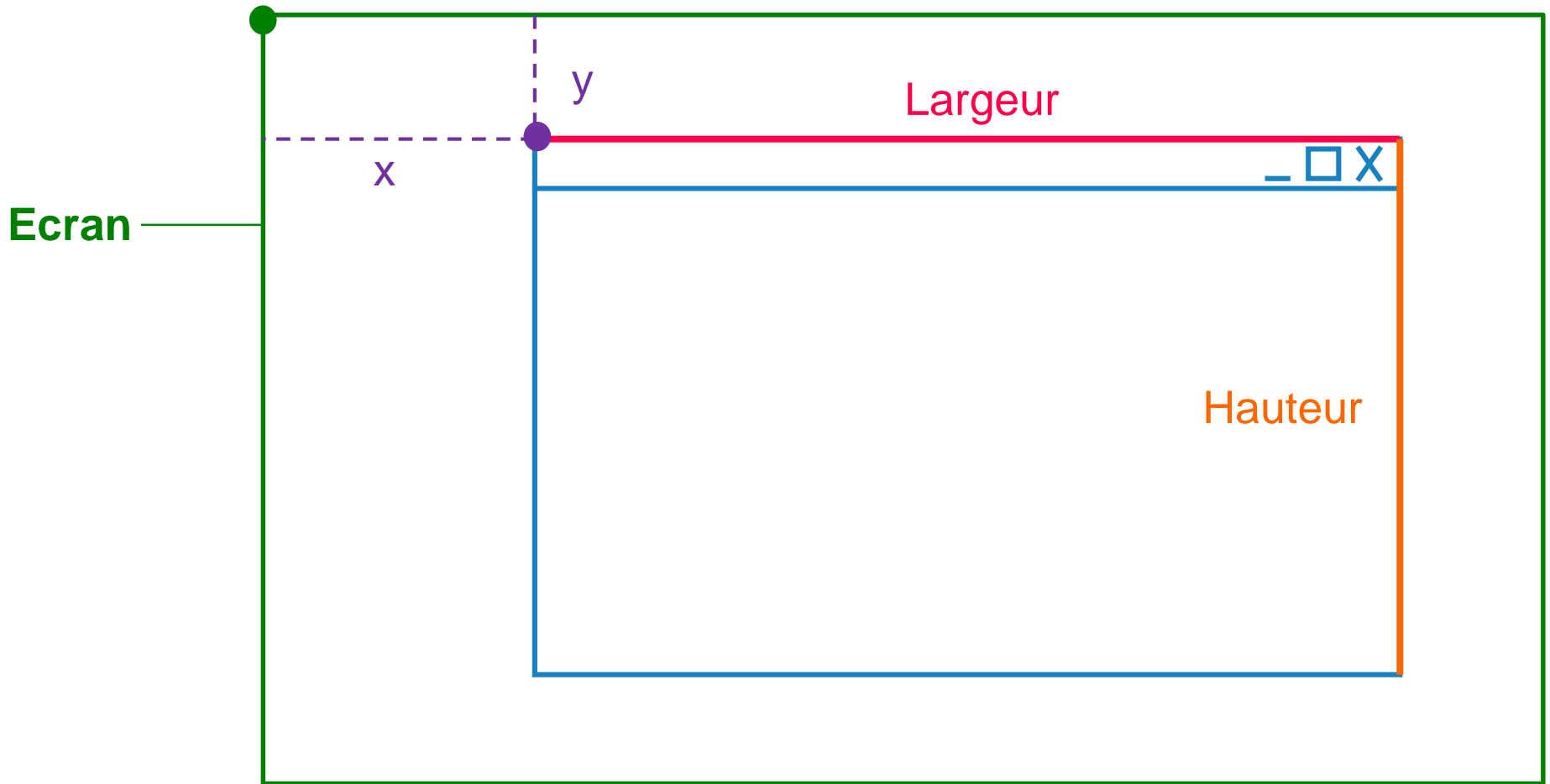
2. JFrame

JFrame

Fenêtre



JFrame



JFrame

```
import javax.swing.*;
```

```
public class FirstWindow extends JFrame
```

```
{
```

```
    public FirstWindow( )
```

```
    { super("Titre de la fenêtre");
```

```
      setBounds(100,100,500,500);
```

```
      setVisible(true);
```

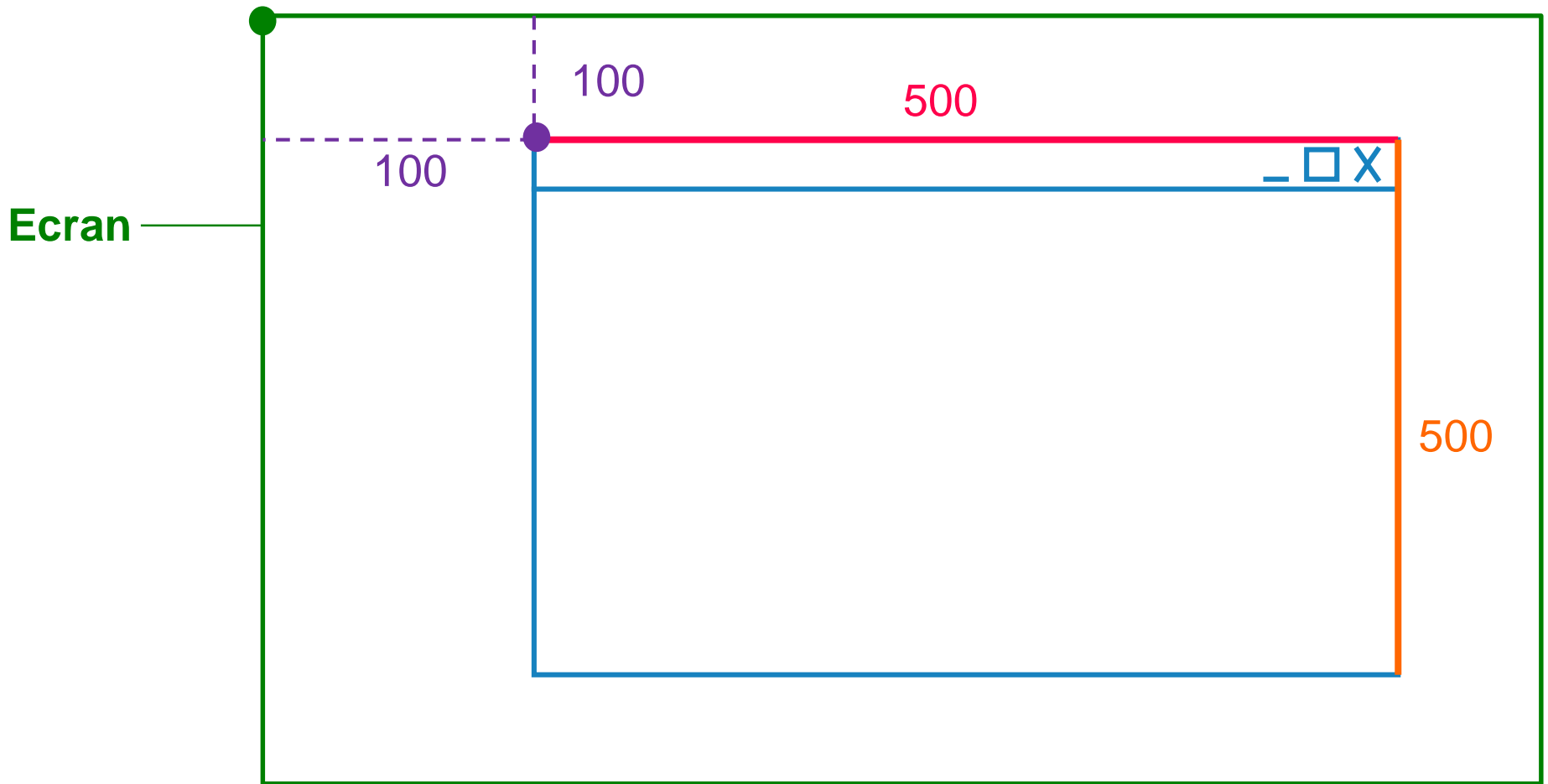
```
    }
```

```
}
```

Positionne la fenêtre par rapport au coin supérieur gauche de l'écran via coordonnées x et y du coin supérieur gauche de la fenêtre + sa largeur et sa hauteur

Affiche la fenêtre à l'écran dès sa création

JFrame



JFrame

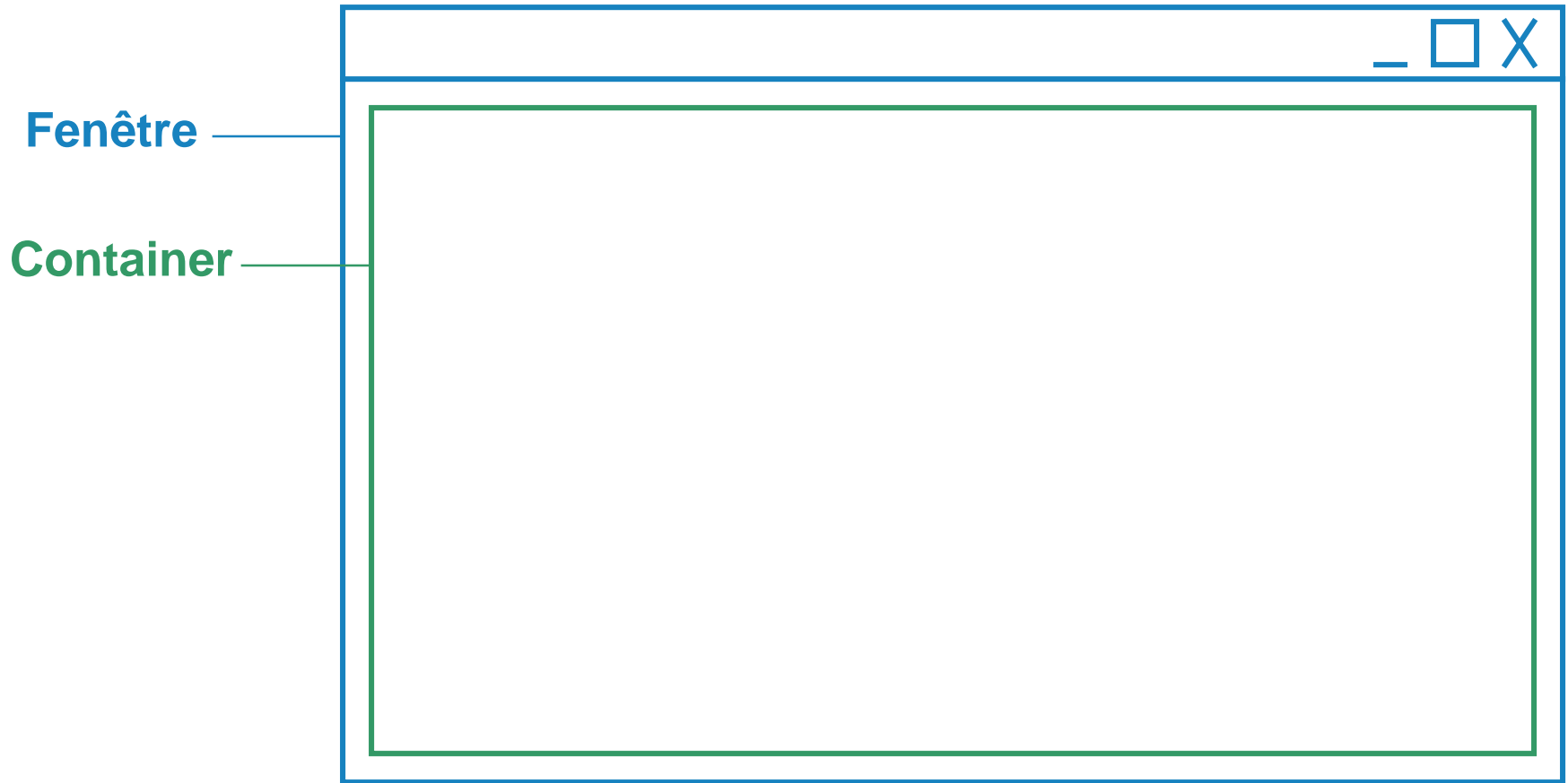
```
public class Principal
{
    public static void main (String[ ] args)
    {
        FirstWindow windowToDisplay = new FirstWindow( ) ;
    }
}
```

JFrame

Mais la fenêtre est vide

- ⇒ Les composants doivent être affichés dans le conteneur de la fenêtre
- ⇒ = un objet de la classe **Container**
- ⇒ **getContentPane()** sur l'objet JFrame pour récupérer ce container

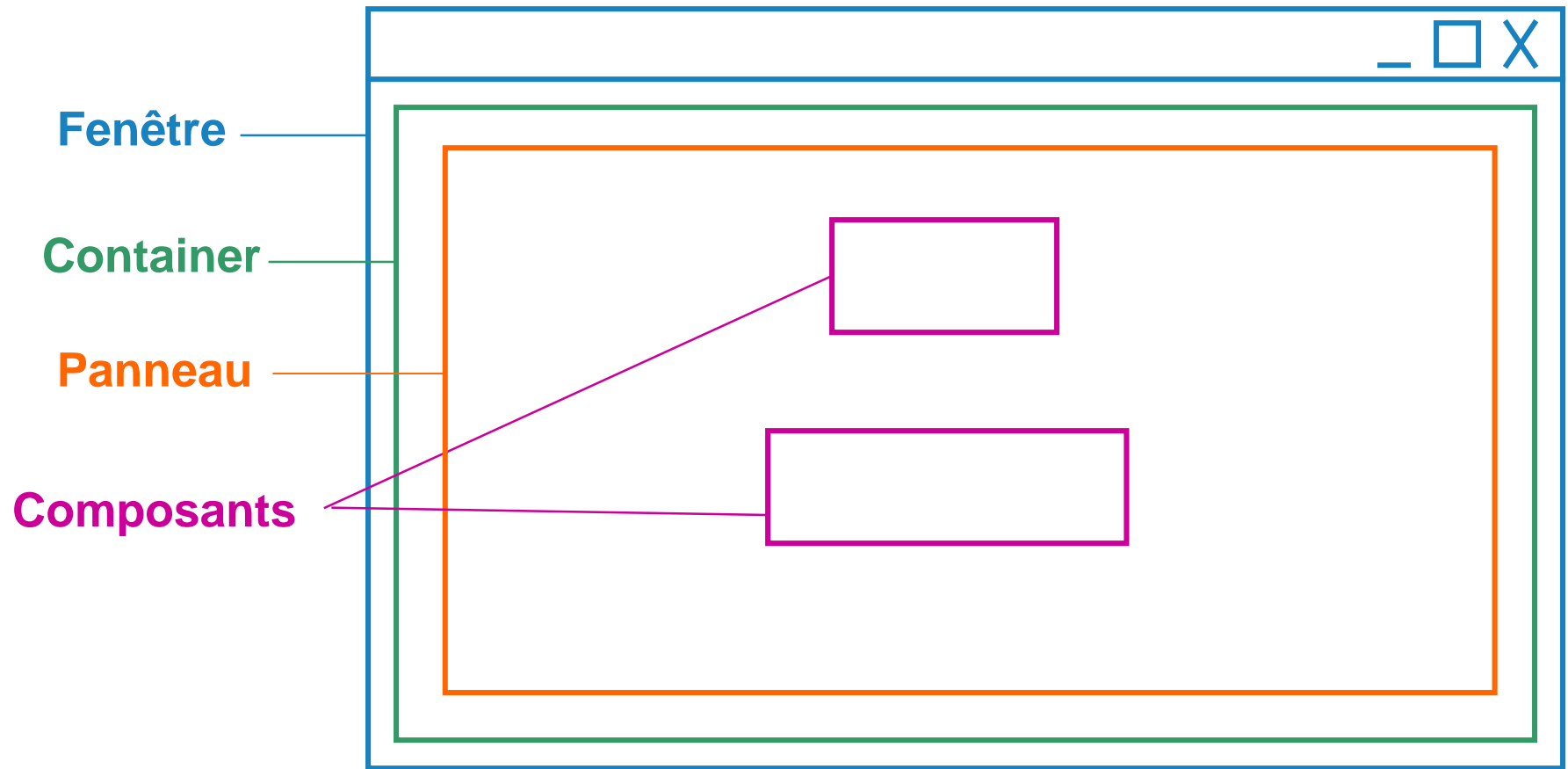
JFrame



Swing

1. JOptionPane
2. JFrame
3. JPanel

JPanel



JPanel

Un **panneau** est **conteneur**

= un objet de la classe **JPanel** (sous-classe de **Container**)

Les **composants** sont des composants Swing

Exemples : JLabel, JTextField, JButton...

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
 - Sans gestionnaire de tracé (Layout)

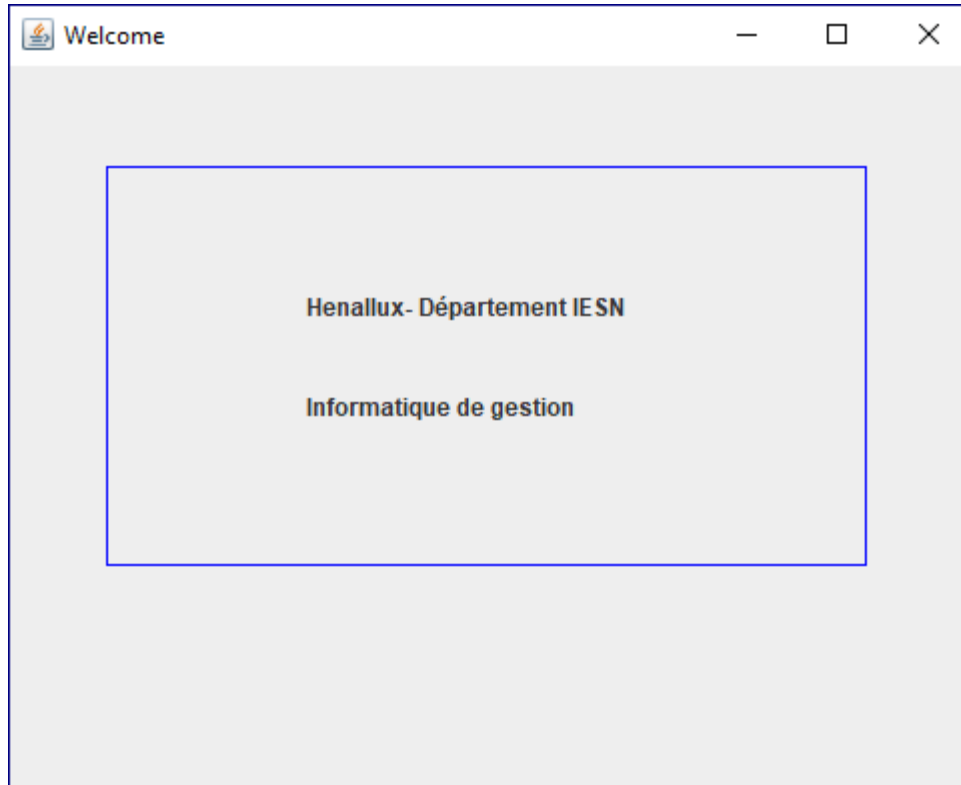
Sans gestionnaire de tracé

Il est possible de positionner les composants dans un conteneur
au pixel près.

Inconvénients :

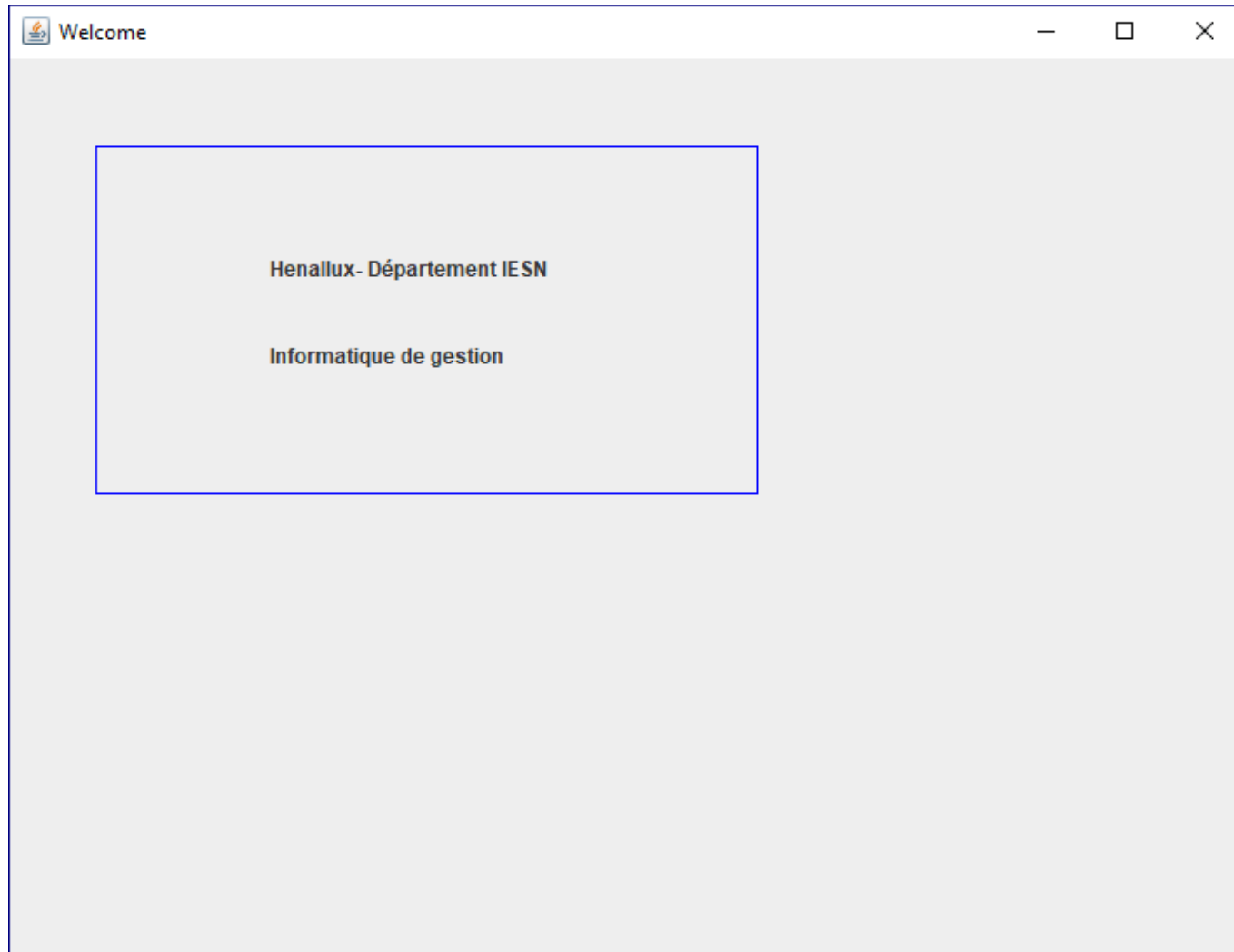
- Tâche fastidieuse
- Les composants sont fixes ;
 - ⇒ Ils ne sont pas réorganisés quand l'utilisateur agrandit la fenêtre

Sans gestionnaire de tracé



Sans gestionnaire de tracé

Si l'utilisateur agrandit la fenêtre



Sans gestionnaire de tracé

```
public class WithSetBoundsPanel extends JPanel
```

```
{ private JLabel line1Label, line2Label; ———> Composants à placer dans le panneau
```

```
public WithSetBoundsPanel()
```

```
{ this.setBounds(50,50,380,200); ———> Positionne le panneau dans le container de la fenêtre
```

```
line1Label = new JLabel("Henallux- Département IESN"); ———> Crée un label
```

```
line1Label.setBounds(100,20,300,100); ———> Spécifie la position du label dans le panneau
```

```
line2Label = new JLabel("Informatique de gestion");  
line2Label.setBounds(100,70,300,100);
```

```
this.setLayout(null); ———> Pour supprimer tout gestionnaire de tracé, car par défaut :  
FlowLayout ⇒ Le setBounds sur le label sera pris en compte
```

```
this.add(line1Label); ———> Ajoute le label au panneau sur base du layout associé  
(ici aucun) ⇒ se base sur le setBounds
```

```
this.add(line2Label);
```

```
    }  
}
```

Sans gestionnaire de tracé

```
public class WithSetBoundsPanel extends JPanel
```

```
{ private JLabel line1Label, line2Label;
```

```
    public WithSetBoundsPanel()
```

```
    { ...
```

```
        this.setBorder(BorderFactory.createLineBorder(Color.BLUE));
```

Affiche une bordure bleue au panneau

```
        line1Label = new JLabel("Henallux- Département IESN");
```

```
        line1Label.setToolTipText("Nom et département de la Haute Ecole");
```

```
        ...
```

Affiche une bulle d'aide

```
    }
```

```
}
```

Sans gestionnaire de tracé

public class **WithoutLayoutWindow** extends **JFrame**

```
{ private Container frameContainer;  
  private WithSetBoundsPanel panel;
```

```
  public WithoutLayoutWindow()  
  { super("Welcome");  
    setBounds(100,100,500,400);
```

```
    panel = new WithSetBoundsPanel();
```

—————→ Crée le panneau

```
    frameContainer = this.getContentPane();
```

—————→ Récupère la référence vers le conteneur de la fenêtre

```
    frameContainer.setLayout(null);
```

—————→ Pour supprimer tout gestionnaire de tracé ⇒ Le setBounds sur le panneau sera pris en compte

```
    frameContainer.add(panel);
```

```
    this.setVisible(true);
```

```
}
```

```
}
```

—————→ Ajoute le panneau au conteneur de la fenêtre sur base du layout associé (ici aucun) ⇒ se base sur le setBounds du panneau

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
 - Sans gestionnaire de tracé (Layout)
 - Avec gestionnaire de tracé (Layout)

Avec gestionnaire de tracé

Il est possible de déléguer à un gestionnaire de tracé la disposition des composants dans un container.

Exemples de gestionnaire de tracé : FlowLayout, BorderLayout, GridLayout...

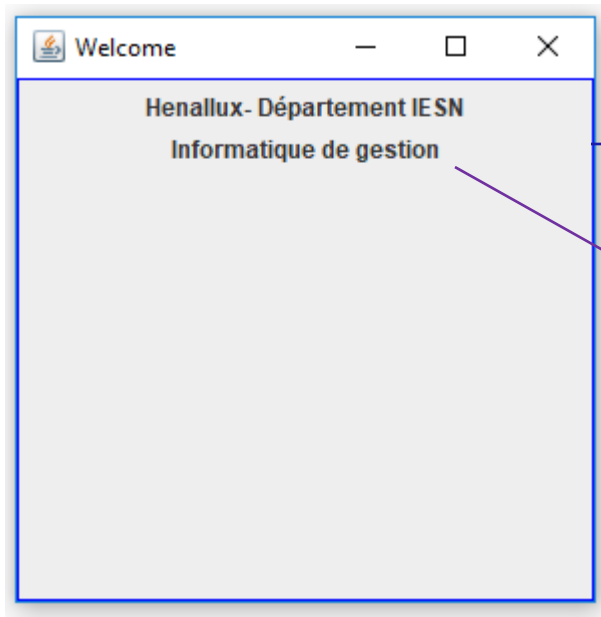
Avantage :

- Les composants ne sont pas fixes ;
⇒ Ils sont réorganisés quand l'utilisateur agrandit la fenêtre

Avec gestionnaire de tracé

- **FlowLayout**
 - Affichage des composants les uns à la suite des autres par ligne
 - Si plus de place sur la ligne \Rightarrow passe à la ligne suivante
- **BorderLayout**
 - Affiche les composants au *nord*, *sud*, *ouest*, *est* et au *centre*
 - Si un seul composant placé au centre \Rightarrow il prend toute la place
- **GridLayout**
 - Affiche les composants sous forme de lignes et de colonnes

Avec gestionnaire de tracé

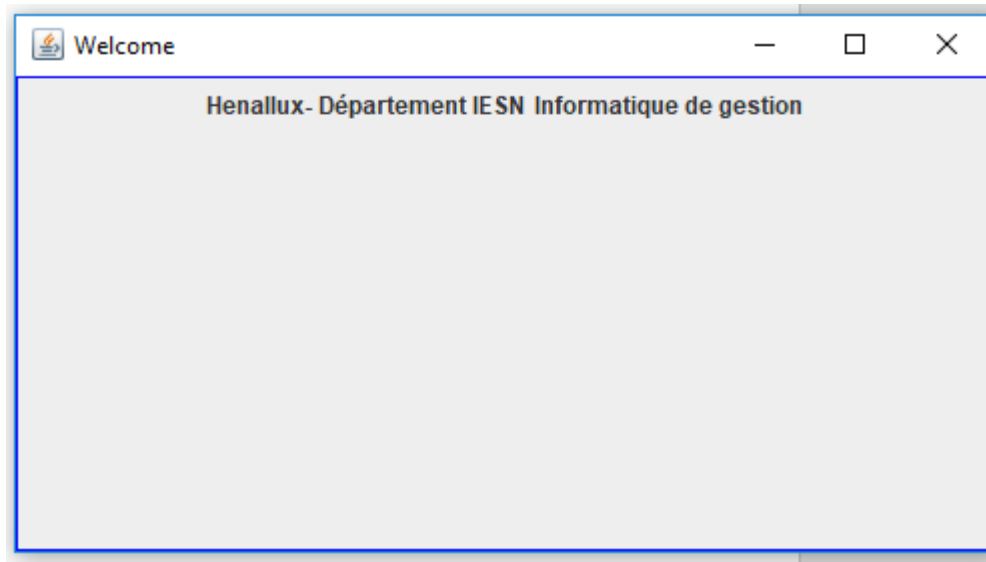


Le panneau est placé au centre du conteneur de la fenêtre via **BorderLayout.CENTER**

Les labels sont placés dans le panneau via un **FlowLayout**

Avec gestionnaire de tracé

Si l'utilisateur agrandit la fenêtre ⇒ réorganisation des labels dans le panneau grâce au FlowLayout



Avec gestionnaire de tracé

```
public class WithFlowLayoutPanel extends JPanel
```

```
{ private JLabel line1Label, line2Label;
```

```
public WithFlowLayoutPanel()  
{
```

```
    line1Label = new JLabel("Henallux- Département IESN");  
    line2Label = new JLabel("Informatique de gestion");
```

```
    this.setLayout(new FlowLayout()); —————> Associe le gestionnaire de tracé FlowLayout
```

```
    this.add(line1Label); —————> Ajoute le label au panneau sur base du FlowLayout associé  
    this.add(line2Label);
```

```
}
```

```
}
```

Avec gestionnaire de tracé

public class **WithLayoutWindow** extends **JFrame**

```
{ private Container frameContainer;  
  private WithFlowLayoutPanel panel;
```

```
  public WithLayoutWindow()  
  { super("Welcome");  
    setBounds(100,100,300,300);
```

```
    panel = new WithFlowLayoutPanel();
```

```
    frameContainer = this.getContentPane();
```

```
    frameContainer.setLayout(new BorderLayout());
```

```
    frameContainer.add(panel, BorderLayout.CENTER);
```

```
    this.setVisible(true);  
  } }
```

Associe le gestionnaire de tracé
BorderLayout au conteneur de la fenêtre

Place le panneau au centre de la fenêtre
⇒ seul composant ⇒ prend toute la place

GridLayout

Exemple : 6 boutons à disposer en 2 lignes X 3 colonnes

⇒ `new GridLayout(2,3)`

un	deux	trois
quatre	cinq	six

GridLayout

```
public class WithGridLayoutPanel extends JPanel {
```

```
    private JButton un, deux, trois, quatre, cinq, six;  —————> Déclare 6 boutons
```

```
    public WithGridLayoutPanel()
```

```
    {    un = new JButton(" un ");
        deux = new JButton(" deux ");
        trois = new JButton(" trois ");
        quatre = new JButton(" quatre ");
        cinq = new JButton(" cinq ");
        six = new JButton(" six ");
```

```
        this.setLayout(new GridLayout(2,3));
```

```
        this.add(un);
        this.add(deux);
        this.add(trois);
        this.add(quatre);
        this.add(cinq);
        this.add(six);
```

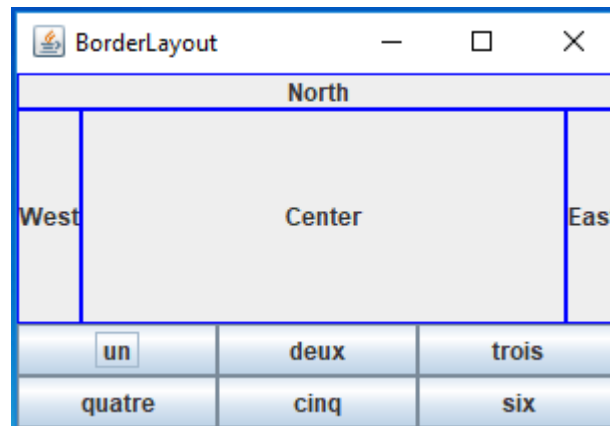
```
    } }
```


BorderLayout

5 panneaux placés aux 4 points cardinaux + 1 au centre

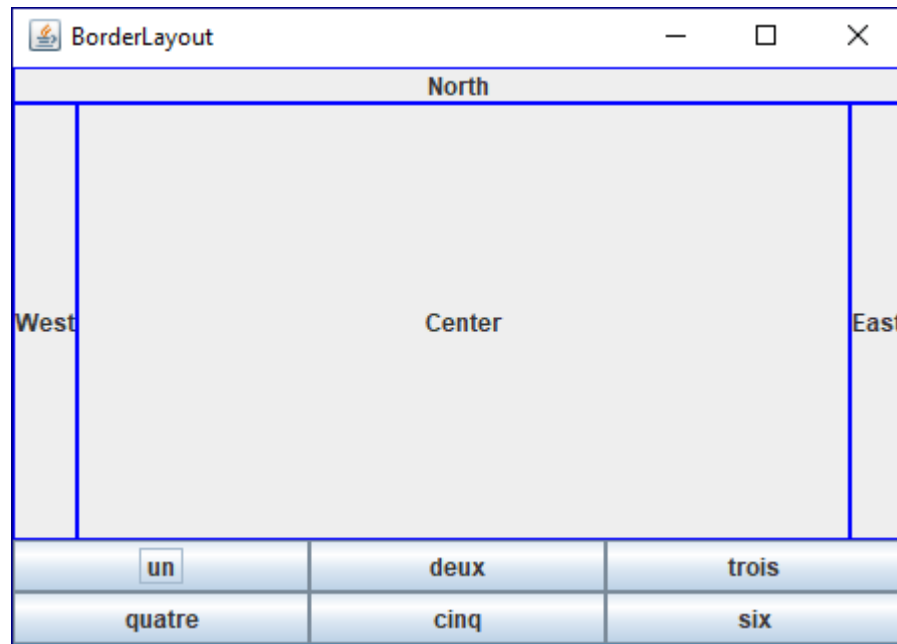
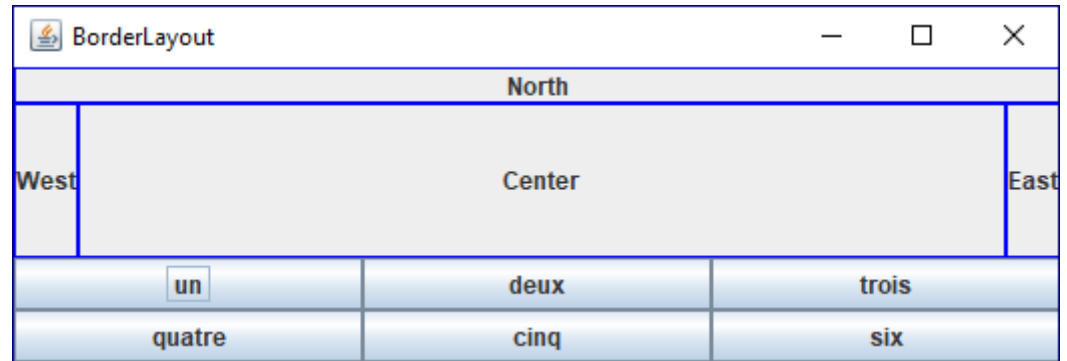
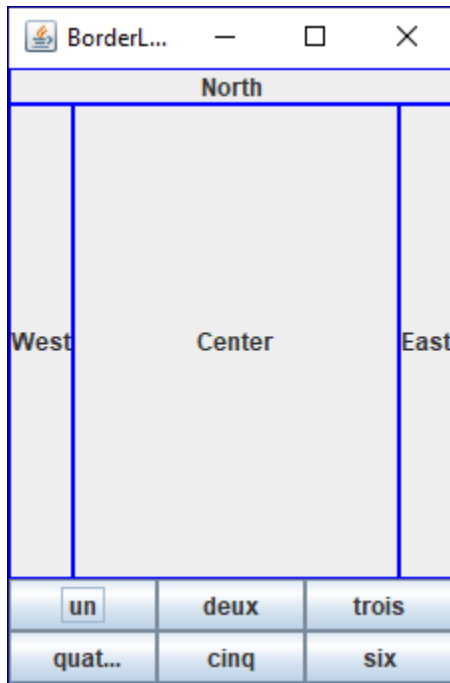
Les panneaux au **nord**, **ouest**, **centre** et **est** contiennent un label placé au centre via BorderLayout.Center

Le panneau au **sud** contient 6 boutons disposés via GridLayout (2,3)



BorderLayout

Si l'utilisateur agrandit la fenêtre \Rightarrow réorganisation des labels dans le panneau



BorderLayout

```
public class NorthPanel extends JPanel {  
  
    private JLabel label;  
  
    public NorthPanel()  
    {  
        label = new JLabel("North");  
        label.setHorizontalAlignment(SwingConstants.CENTER);  
  
        this.setBorder(BorderFactory.createLineBorder(Color.BLUE));  
  
        this.setLayout(new BorderLayout());  
  
        this.add(label, BorderLayout.CENTER);  
    }  
}
```

Centre le texte dans le label

BorderLayout

```
public class EastPanel extends JPanel { ... }
```

```
public class WestPanel extends JPanel { ... }
```

```
public class CenterPanel extends JPanel { ... }
```

BorderLayout

```
public class BorderLayoutWindow extends JFrame {  
    private NorthPanel northPanel;  
    private WithGridLayoutPanel gridLayoutPanel;  
    private EastPanel eastPanel;  
    private WestPanel westPanel;  
    private CenterPanel centerPanel;  
    private Container frameContainer;  
  
    public BorderLayoutWindow()  
    { ...  
        northPanel = new NorthPanel();  
        westPanel = new WestPanel();  
        centerPanel = new CenterPanel();  
        eastPanel = new EastPanel();  
        gridLayoutPanel = new WithGridLayoutPanel();  
        frameContainer = this.getContentPane();  
  
        frameContainer.setLayout(new BorderLayout());  
        frameContainer.add(northPanel, BorderLayout.NORTH);  
        frameContainer.add(westPanel, BorderLayout.WEST);  
        frameContainer.add(centerPanel, BorderLayout.CENTER);  
        frameContainer.add(eastPanel, BorderLayout.EAST);  
        frameContainer.add(gridLayoutPanel, BorderLayout.SOUTH);  
    }  
}
```

Gestionnaires de tracé

Autres gestionnaires de tracé

GridBagLayout

CardLayout

BoxLayout

...

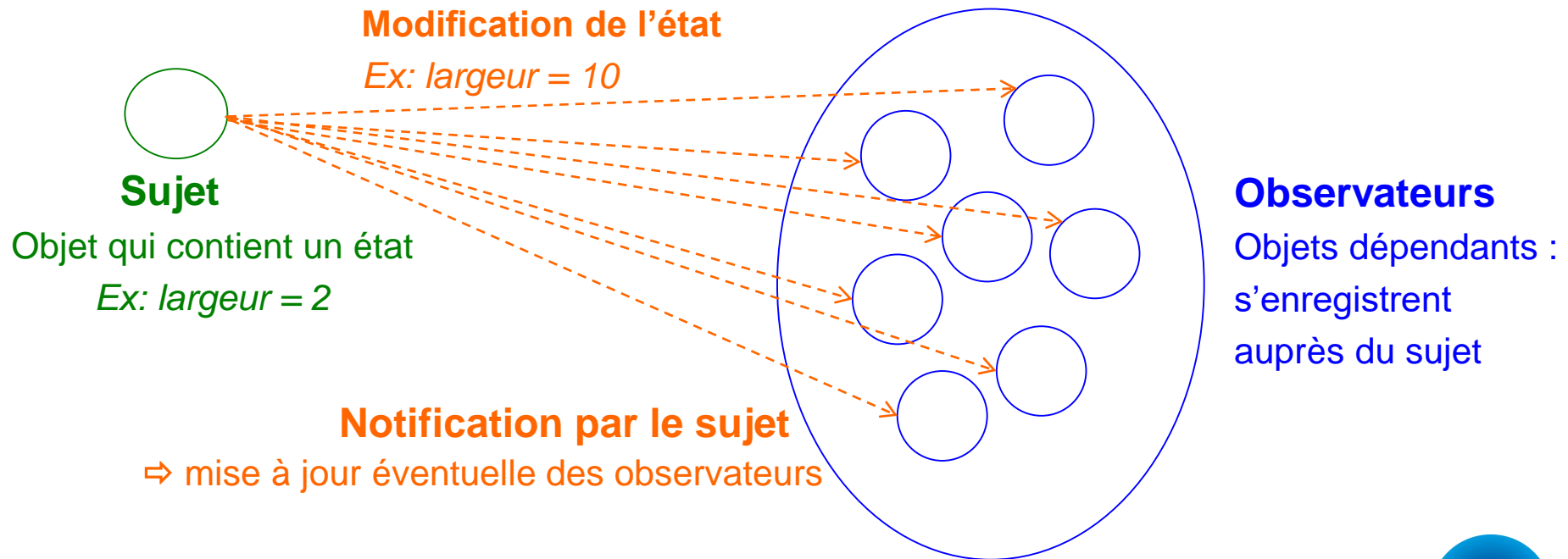
Autre librairie de composants graphiques

JavaFX

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern

Objectif du pattern **observateur**

Lorsqu'un objet change d'état, notifier tous ceux qui en dépendent afin qu'ils soient mis à jour automatiquement (+ réaction éventuelle)



Observer Pattern

Le **sujet** contient

- une **liste des observateurs**
- une méthode pour **ajouter/supprimer un observateur** de la liste
- une méthode qui **boucle sur les observateurs pour les actualiser** :

Appel d'une méthode sur chacun d'eux

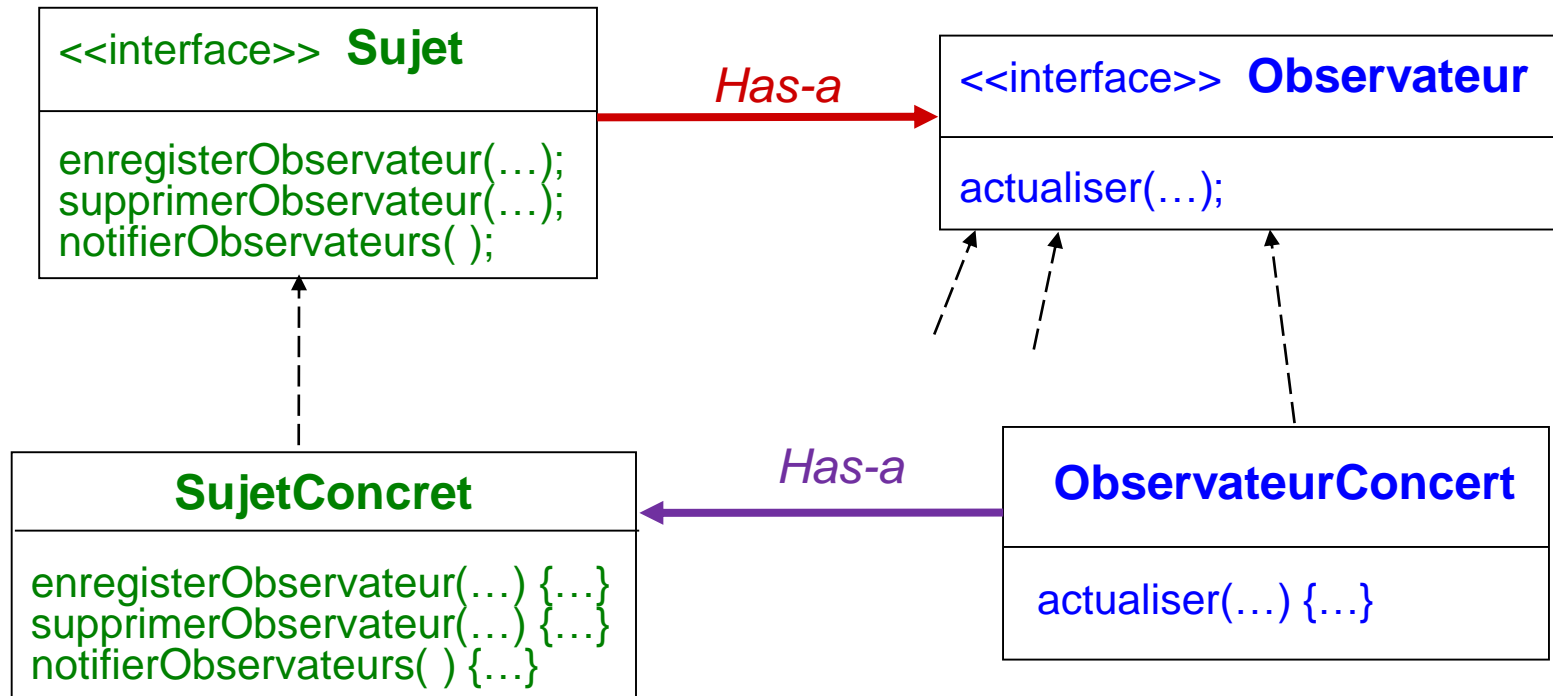


Quelle méthode?



Les **observateurs** doivent implémenter une interface

Observer Pattern



Observer Pattern

Exemple 1

Gestion des événements des composants Swing :

Sujet : *JButton* bouton

Observateur : objet (écouteur) d'une classe qui implémente *ActionListener*

① L'observateur s'enregistre auprès du sujet :

⇒ bouton.*addActionListener* (ecouteur)

② Quand clic sur le bouton :

⇒ Appel par le sujet de la méthode *actionPerformed* sur tous les observateurs enregistrés

Observer Pattern

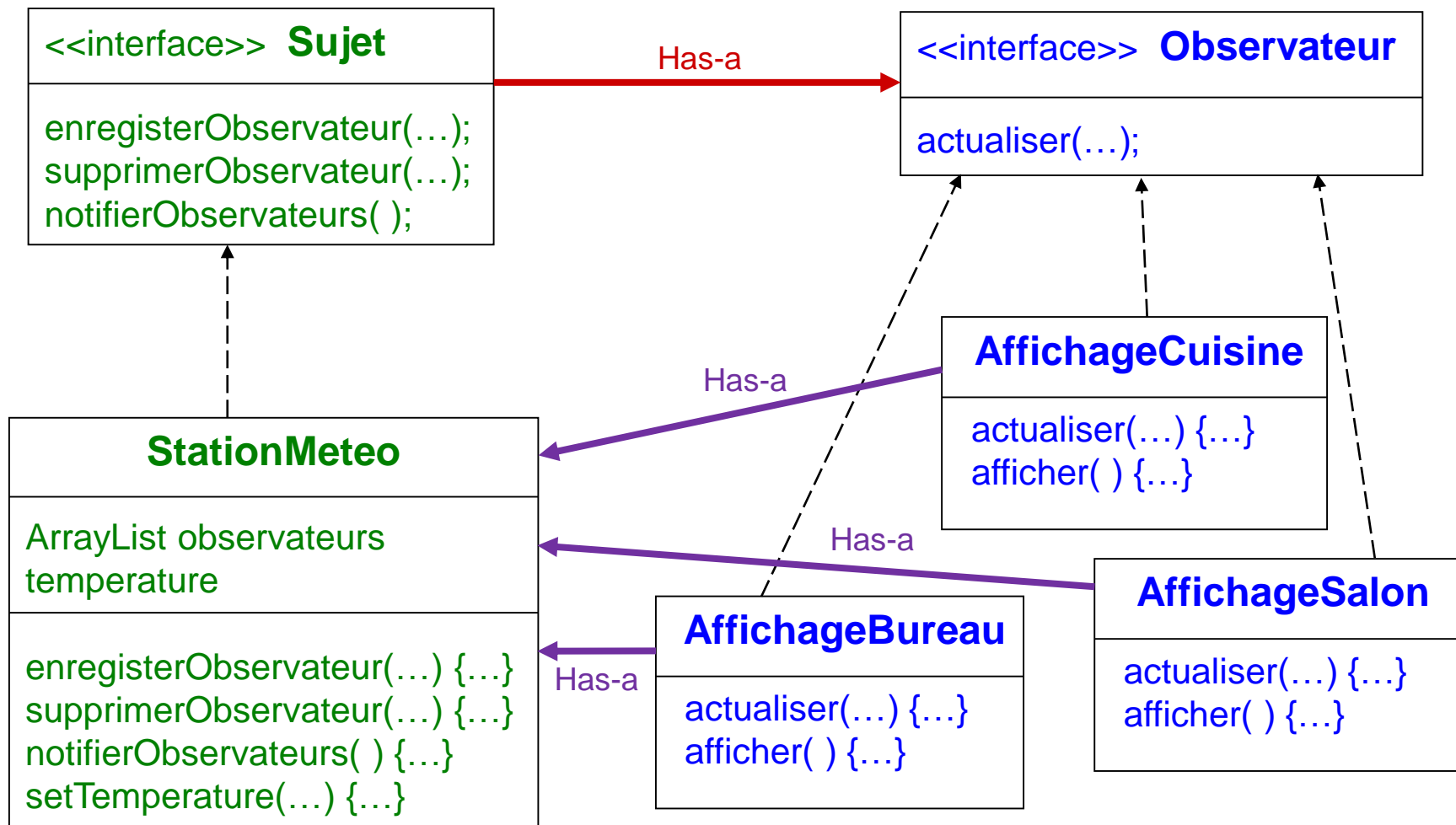
Exemple 2

Sujet

station météo qui capte la température

Observateurs

appareils qui affichent la température captée par la station



Observer Pattern

```
public interface Sujet
{
    public void enregistrerObservateur (Observateur o);
    public void supprimerObservateur (Observateur o);
    public void notifierObservateurs ( );
}
```

```
public interface Observateur
{
    public void actualiser (float temperature);
    public void afficher ( );
}
```

```
public class StationMeteo implements Sujet
```

```
{ private ArrayList<Observateur> observateurs;  
  private float temperature;
```

```
  public StationMeteo ( ) { observateurs = new ArrayList<Observateur>( ); }
```

```
  public void enregistrerObservateur (Observateur o)  
  { observateurs.add(o); }
```

```
  public void supprimerObservateur (Observateur o)  
  { observateurs.remove(o); }
```

```
  public void notifierObservateurs ( )  
  { for (Observateur o: observateurs)  
    { o.actualiser(temperature); }  
  }
```

```
  public void setTemperature (float newTemperature)  
  { temperature = newTemperature;  
    notifierObservateurs( );  
  }  
}
```

A chaque modification de
température,
les observateurs sont notifiés

Observer Pattern

```
public class AffichageSalon implements Observateur
```

```
{ private Sujet donneesMeteo;
```

```
private float temperature;
```

```
public AffichageSalon (Sujet donneesMeteo)
```

```
{ this.donneesMeteo = donneesMeteo;
```

```
donneesMeteo.enregistrerObservateur(this); }
```

L'observateur s'enregistre
auprès du sujet

```
public void actualiser (float temperature)
```

```
{ this.temperature = temperature;
```

```
afficher( );
```

```
}
```

L'observateur met à jour ses
données (+ réaction) quand il est
notifié d'un changement du sujet

```
public void afficher ( ) { // afficher température }
```

```
}
```


Observer Pattern

Initialisation du sujet et des observateurs (ex: dans main)

StationMeteo **donneesMeteo** = new StationMeteo();

AffichageSalon **affichageSalon** = new AffichageSalon (**donneesMeteo**);

AffichageSalon **affichageCuisine** = new AffichageCuisine (**donneesMeteo**);

AffichageSalon **affichageBureau** = new AffichageBureau (**donneesMeteo**);



Swing - *Gestion des événements*

3 intervenants (càd 3 objets différents) dans la gestion d'évènement Swing

1. Le **composant à écouter** (ex: un bouton)
2. L'**écouteur d'évènement** (ex: l'objet qui écoute le bouton)
3. L'**objet de type d'évènement** créé automatiquement quand l'évènement a lieu
(ex: l'objet créé quand on clique sur le bouton)

Swing - Gestion des événements

A l'exécution, lors de l'évènement :

1. Création automatique d'un **objet de type évènement** qui contient des **infos sur l'évènement**

(ex: si l'utilisateur a coché ou décoché une case à cocher)

Cet objet est un objet d'une **sous-classe de la classe Event**

2. Cet objet de type évènement est passé comme argument à la méthode appelée automatiquement en réaction à l'évènement

⇒ Infos sur l'évènement accessibles et utilisables par le programmeur

Swing - Gestion des événements

Programmeur

1. Le programmeur **crée** le **composant à écouter** (ex: un bouton)
2. Le programmeur **crée** une classe **écouteur d'événement**

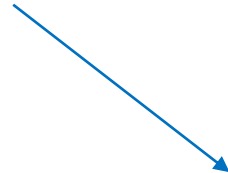
Il écrit le code de la **réaction** souhaitée dans une **méthode** de cette classe

Pour ce faire, il peut utiliser les informations stockées dans l'objet de type événement créé par Java

À condition que la méthode soit correctement déclarée!!!

Swing - Gestion des événements

classe écouteur d'événement



doit implémenter le bon interface!



Contient les **déclarations des méthodes**
appelées automatiquement en réaction aux évènements



Syntaxe à respecter!

Swing - Gestion des événements

Programmeur (suite)

3. Le programmeur **crée** un **objet** de cette classe écouteur
4. Le programmeur **associe** cet **objet écouteur** **au composant à écouter**

Swing - Gestion des événements

Machine virtuelle Java

A l'exécution, lors de l'évènement (*ex: si clic sur le bouton*):

Si le **composant** sur lequel a eu lieu l'évènement est écouté (s'il y a des observateurs associés):

⇒ boucle sur tous les objets **écouteurs d'évènement** associés

⇒ Appel de la méthode correspondant à la réaction à l'évènement (cf déclaration de la méthode dans l'interface correspondante)

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne

Classe interne

```
class ClasseEnglobante
```

```
{ private ...    Variables d'instance
```

```
...            Constructeurs et méthodes
```

```
private class ClasseInterne
```

```
{ Variables d'instance
```

```
Constructeurs et méthodes
```

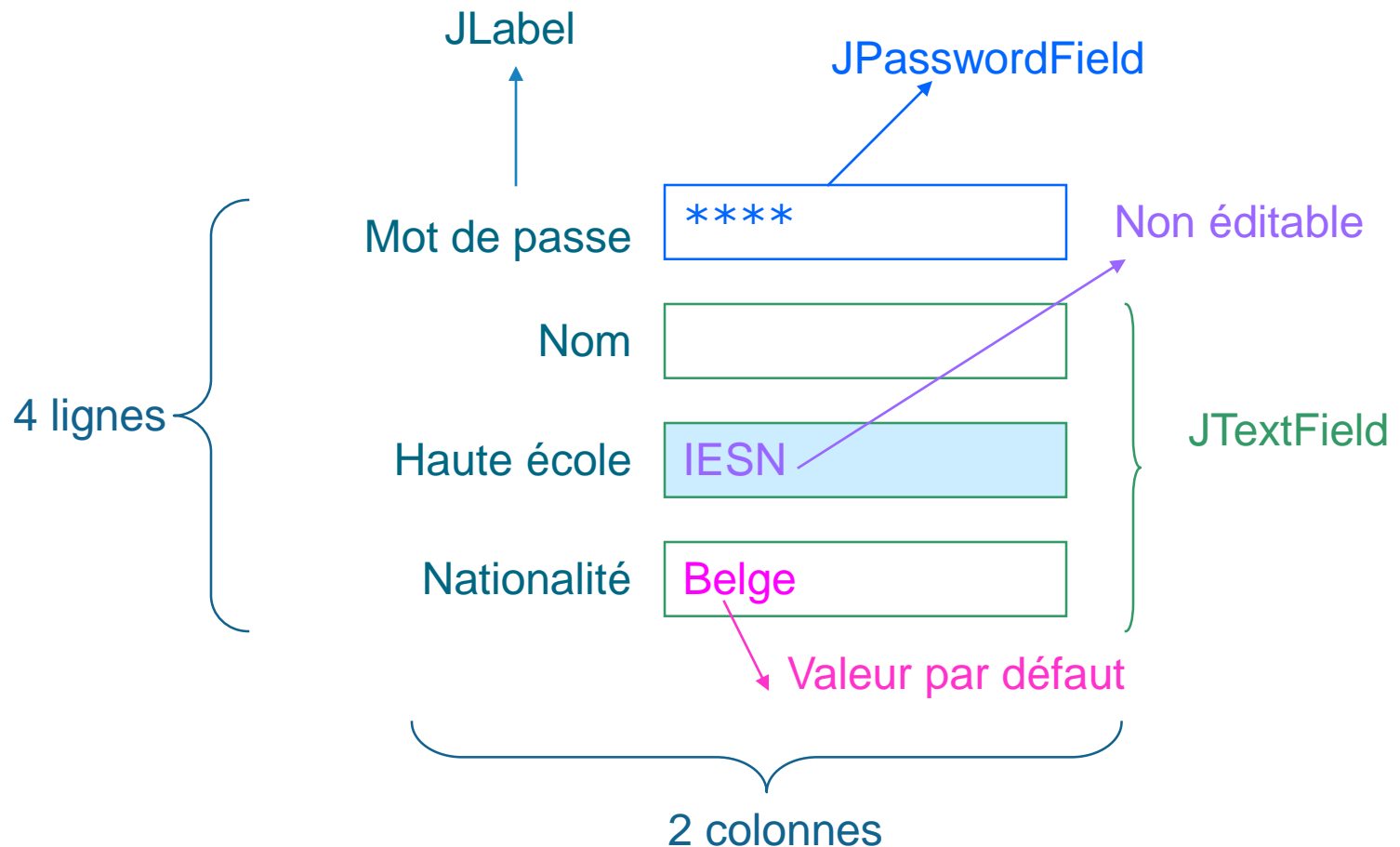
```
}
```

```
}
```

accès

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
 - JLabel
 - JTextField
 - JPasswordField

Zones de texte



⇒ Gestionnaire de tracé: **GridLayout**

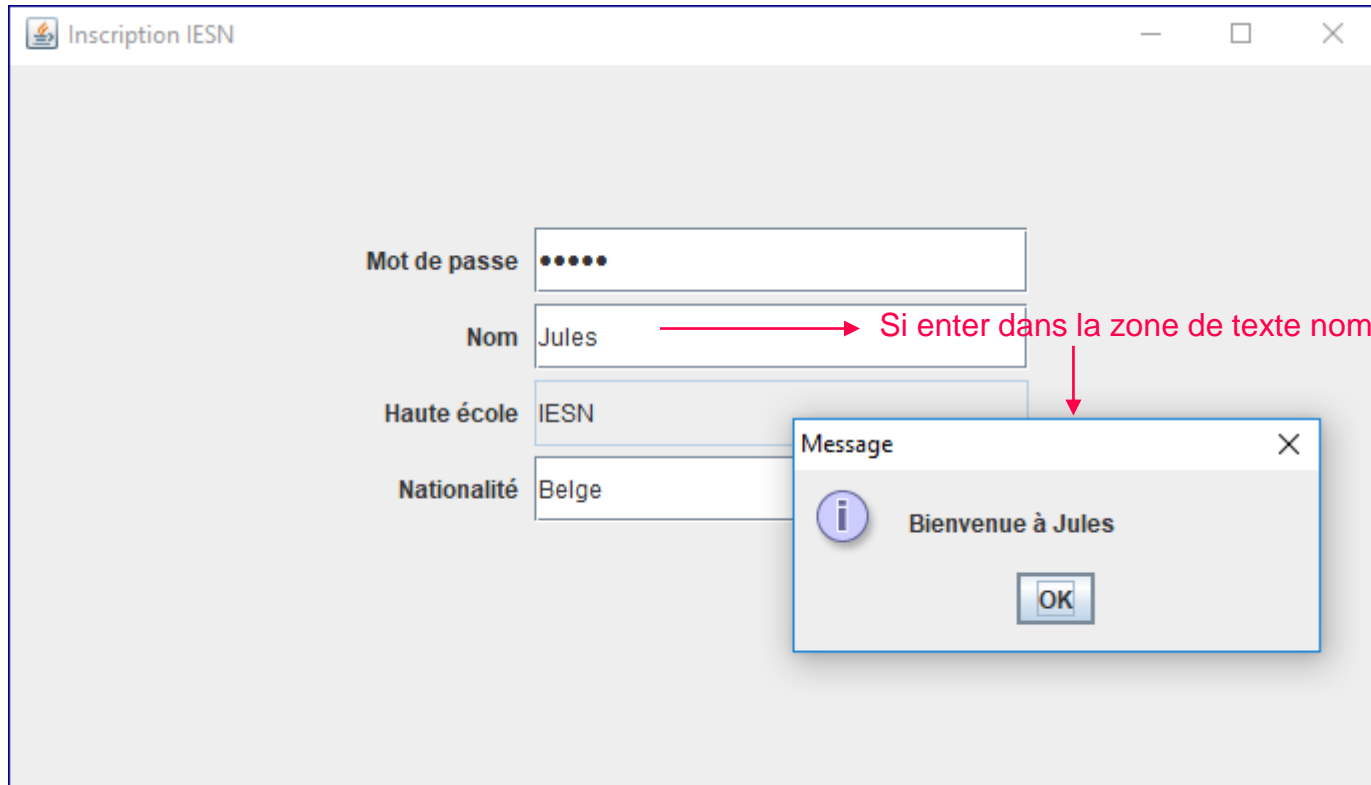
Zones de texte



The image shows a Java Swing window titled "Inscription IESN". Inside the window, there is a registration form with four text input fields. The labels and their corresponding values are:

Label	Value
Mot de passe	
Nom	
Haute école	IESN
Nationalité	Belge

Zones de texte



Zones de texte

The screenshot shows a web application window titled 'Inscription IESN'. It contains a registration form with the following fields:

- Mot de passe: masked with dots
- Nom: Jules
- Haute école: IESN
- Nationalité: Française

An 'Avertissement' (Warning) dialog box is displayed over the form. It contains a yellow warning icon and the text: 'Veuillez compléter votre dossier au secrétariat'. An 'OK' button is at the bottom of the dialog. A red arrow points from the 'Nationalité' field to the dialog box, and another red arrow points from the text 'Si enter dans la zone de texte nationalite' to the 'Nationalité' field.

Zones de texte

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

```
public class TextPanel extends JPanel  
{ private JLabel PasswordLabel, nameLabel, schoolLabel, nationalityLabel;  
  private JTextField name, school, nationality;  
  private JPasswordField password;
```

```
  public TextPanel()  
  { this.setBounds(10,80,500,150);  
    ...
```

`this.setLayout(new GridLayout(4,2,5,5));` → 5 pixels d'écart entre composants

```
PasswordLabel = new JLabel("Mot de passe ");  
this.add>PasswordLabel);  
password = new JPasswordField();  
this.add(password);
```

```
nameLabel = new JLabel("Nom ");  
this.add(nameLabel);  
name = new JTextField();  
this.add(name);
```

```
schoolLabel = new JLabel("Haute école ");  
this.add(schoolLabel);  
school = new JTextField("IESN");  
school.setEditable(false);  
this.add(school);
```

Non éditable

Valeurs d'initialisation

```
nationalityLabel = new JLabel("Nationalité ");  
this.add(nationalityLabel);  
nationality = new JTextField("Belge");  
this.add(nationality);
```


Gestion d'événement sur JTextField

```
public class TextPanel extends JPanel
{
    ...

    private JTextField name, nationality;

    public TextPanel()
    {
        ...

        TextListener listener = new TextListener();

        name.addActionListener(listener);
        nationality.addActionListener(listener);

    }

    ...
}
```

Classe écouteur d'évènement

Objet écouteur d'évènement

Associe l'écouteur d'évènement aux composants à écouter

Gestion d'événement sur JTextField

```
public class TextPanel extends JPanel
```

```
{ ... private JTextField name,nationality;
```

```
public TextPanel()
```

```
{ ...
```

```
TextListener listener = new TextListener();
```

```
name.addActionListener(listener);
```

```
nationality.addActionListener(listener);
```

```
}
```

*Un objet de type **ActionEvent** contient des informations sur l'événement.*

Rappel: généré automatiquement

Interface

```
private class TextListener implements ActionListener {
```

Méthode appelée si enter dans JTextField

```
public void actionPerformed ( ActionEvent event) {
```

```
if (event.getSource() == name)
```

Retourne la source de l'évènement

```
    JOptionPane.showMessageDialog(null,"Bienvenue à "+event.getActionCommand());
```

```
else
```

Retourne la chaîne de caractères entrée

```
    JOptionPane.showMessageDialog(null, "Veuillez compléter votre dossier au secrétariat",  
        "Avertissement", JOptionPane.WARNING_MESSAGE); } }
```

Zones de texte

```
labelMotPasse = new JLabel("Mot de passe: ");  
labelMotPasse.setHorizontalAlignment(SwingConstants.RIGHT);
```

Constante

N.B. Par défaut, alignement à gauche

Aligner le label à droite

```
zoneTexteNom = new JTextField(30);
```

```
String texte = zoneTexteNom.getText( );
```

Lire le contenu d'un JTextField

```
zoneTexteNom.setText("Jules");
```

Modifier la valeur d'un JTextField

```
zoneMotPasse = new JPasswordField(20);
```

```
String t = new String(zoneMotPasse.getPassword( ));
```

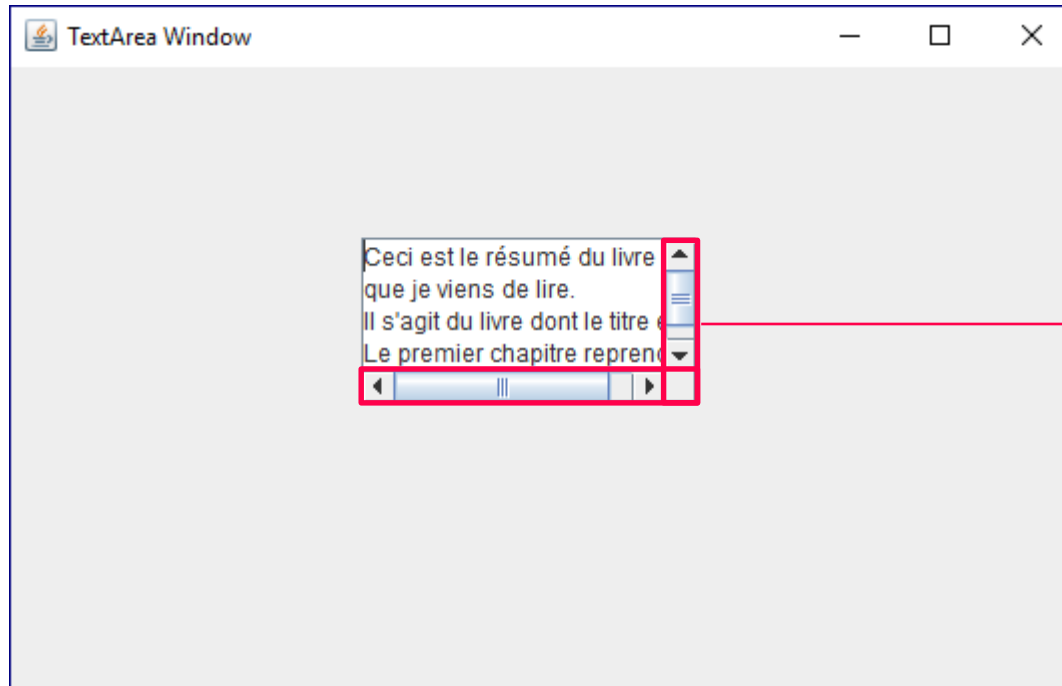
Lire un mot de passe

Attention: retourne un tableau de char

A transformer en un String

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
 - JLabel
 - JTextField
 - JPasswordField
 - JTextArea


JTextArea



Défilant s'affiche si le texte introduit
déborde de la zone initiale
⇒ JScrollPane

JTextArea

```
public class TextAreaPanel extends JPanel {  
  
    private JTextArea texte;  
  
    public TextAreaPanel( ) {  
  
        ...  
  
        this.setLayout(new FlowLayout( ));  
  
        texte = new JTextArea(5,15);    NB. Un JTextArea pas défilant par défaut!  
  
        this.add(new JScrollPane(texte));  
    }  
}
```

 *Défilant*

JTextArea

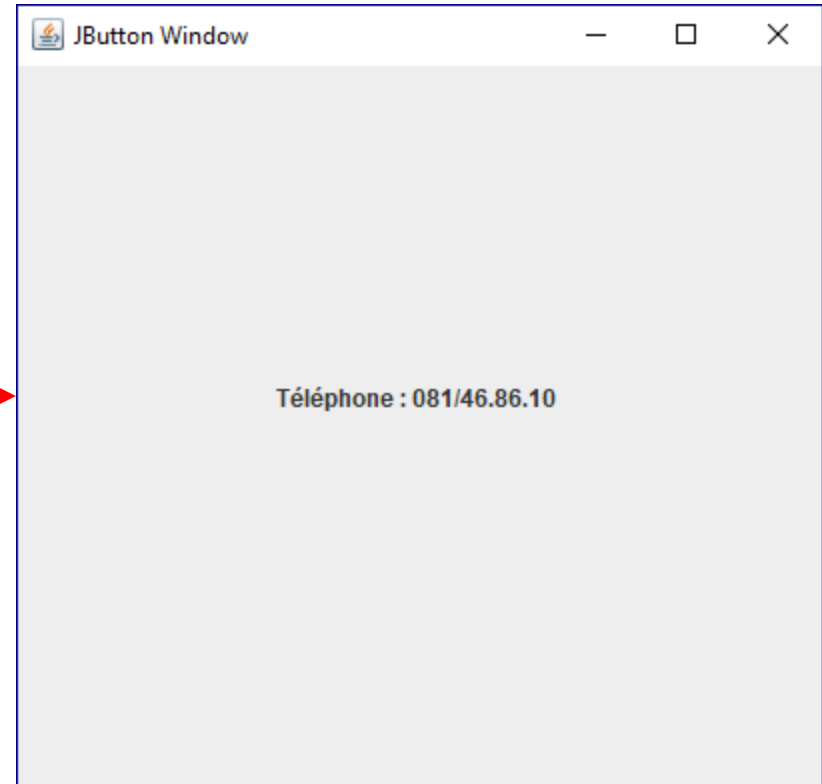
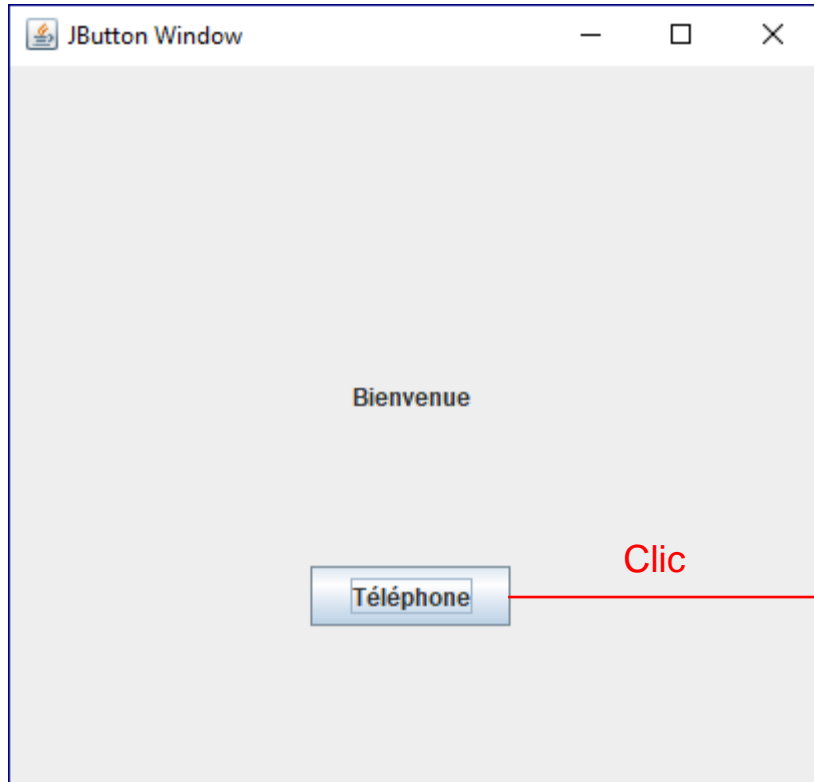
String message = **texte.getSelectedText()**



Récupère le texte sélectionné par l'utilisateur

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
8. Boutons
 - JButton

JButton



JButton

```
public class ButtonPanel extends JPanel {
```

```
    private JButton iesnButton;
```

```
    private JLabel welcome, phone;
```

```
    public ButtonPanel()
```

```
    { ... this.setLayout(null);
```

```
        welcome = new JLabel("Bienvenue");
```

```
        welcome.setBounds(...);
```

```
        phone = new JLabel("Téléphone : 081/46.86.10");
```

```
        iesnButton = new JButton("Téléphone");
```

```
        iesnButton.setBounds(...);
```

```
        this.add(welcome);
```

```
        this.add(iesnButton);
```

*Seulement deux des trois composants
ajoutés au container!*

JButton

```
public class ButtonPanel extends JPanel {  
    private JButton iesnButton;  
    private JLabel welcome, phone;
```

Créé et initialisé mais pas affiché

```
public ButtonPanel()  
{ ... phone = new JLabel("Téléphone : 081/46.86.10");
```

```
    ButtonListener listener = new ButtonListener();
```

→ Crée un écouteur d'évènement

```
    iesnButton.addActionListener(listener);
```

→ Associe l'écouteur au composant à écouter

```
}
```

Interface



```
private class ButtonListener implements ActionListener
```

```
{ public void actionPerformed( ActionEvent event) {
```

→ Appelée si clic sur bouton

```
    ButtonPanel.this.removeAll();
```

→ Vide le contenu du container

```
    phone.setBounds(30,50,200,30);
```

```
    ButtonPanel.this.add(phone);
```

→ Ajoute le label au container

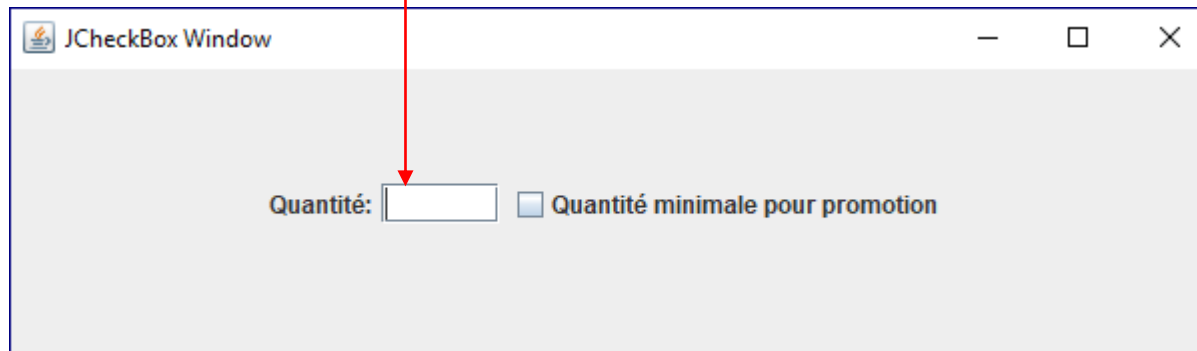
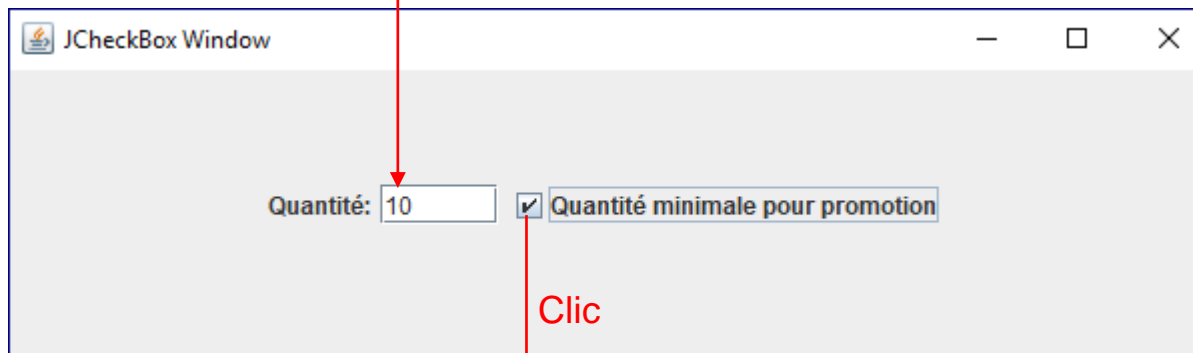
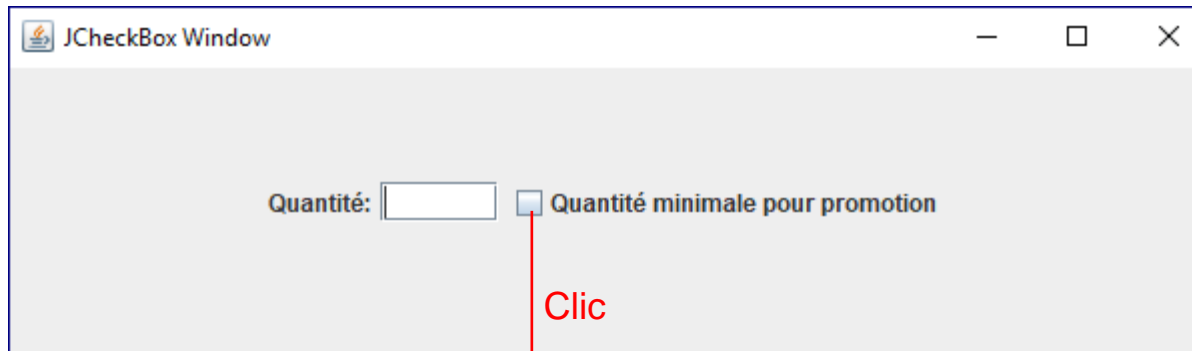
```
    ButtonPanel.this.repaint();
```

→ Redessine le panneau !!!

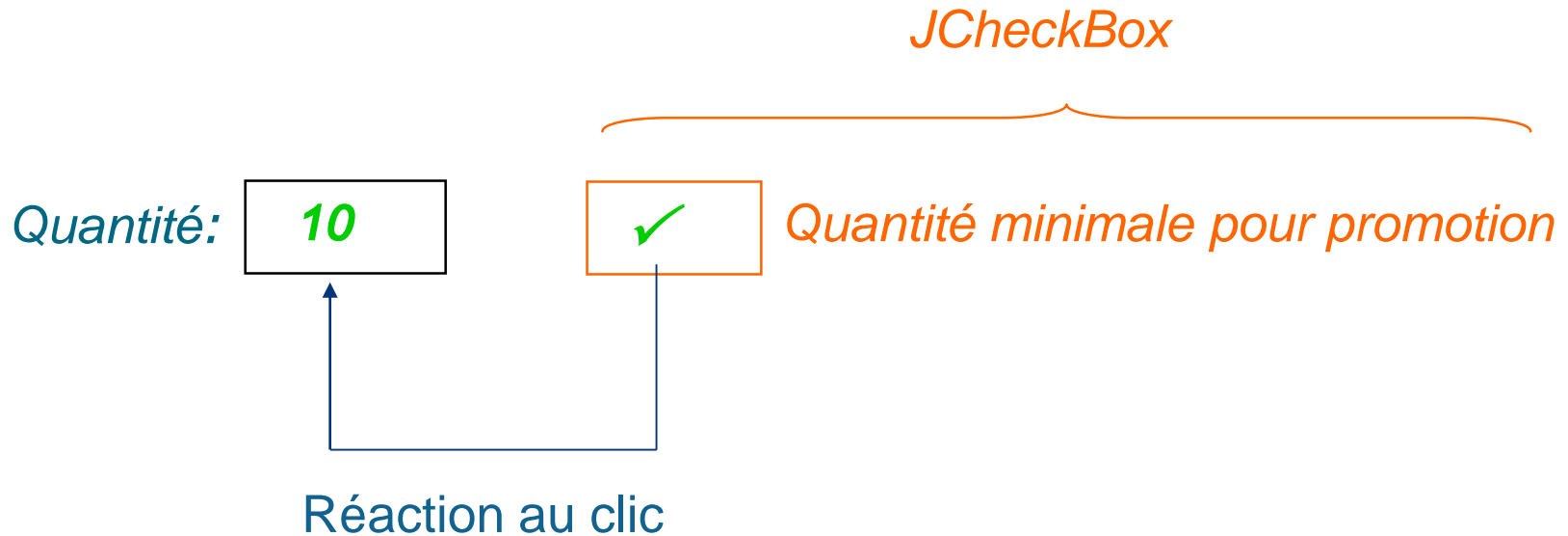
```
} } }
```

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
8. Boutons
 - JButton
 - JCheckBox

JCheckBox



JCheckBox



```
public class CheckBoxPanel extends JPanel
```

```
{ private JCheckBox defaultQuantity;
```

```
private JTextField quantityText;
```

```
private JLabel quantityLabel;
```

```
public CheckBoxPanel() {
```

```
...
```

```
this.setLayout(new FlowLayout());
```

```
quantityLabel = new JLabel("Quantité:");
```

```
this.add(quantityLabel);
```

```
quantityText = new JTextField(5);
```

```
this.add(quantityText);
```

```
defaultQuantity = new JCheckBox(" Quantité minimale pour promotion");
```

```
this.add(defaultQuantity);
```

JCheckBox

Nouveau type d'écouteur :

Interface **ItemListener**

capable de détecter deux états possibles: **coché** ou **décoché**


```
public class CheckBoxPanel extends JPanel
```

```
{ private JCheckBox defaultQuantity; ...
```

```
public CheckBoxPanel() {
```

```
...
```

```
CheckBoxListener listener = new CheckBoxListener();
```

Crée un écouteur d'évènement

```
defaultQuantity.addItemListener(listener);
```

Associe l'écouteur au composant à écouter

```
}
```

Interface

```
private class CheckBoxListener implements ItemListener
```

```
{ public void itemStateChanged(ItemEvent event) {
```

Appelé si checkBox cochée/décochée

```
if ( event.getStateChange() == ItemEvent.SELECTED )
```

```
    quantityText.setText("10");
```

Si checkBox cochée

```
else    quantityText.setText("");
```

```
}}}
```

JCheckBox

N.B.

```
if ( event.getStateChange() == ItemEvent.SELECTED )
```

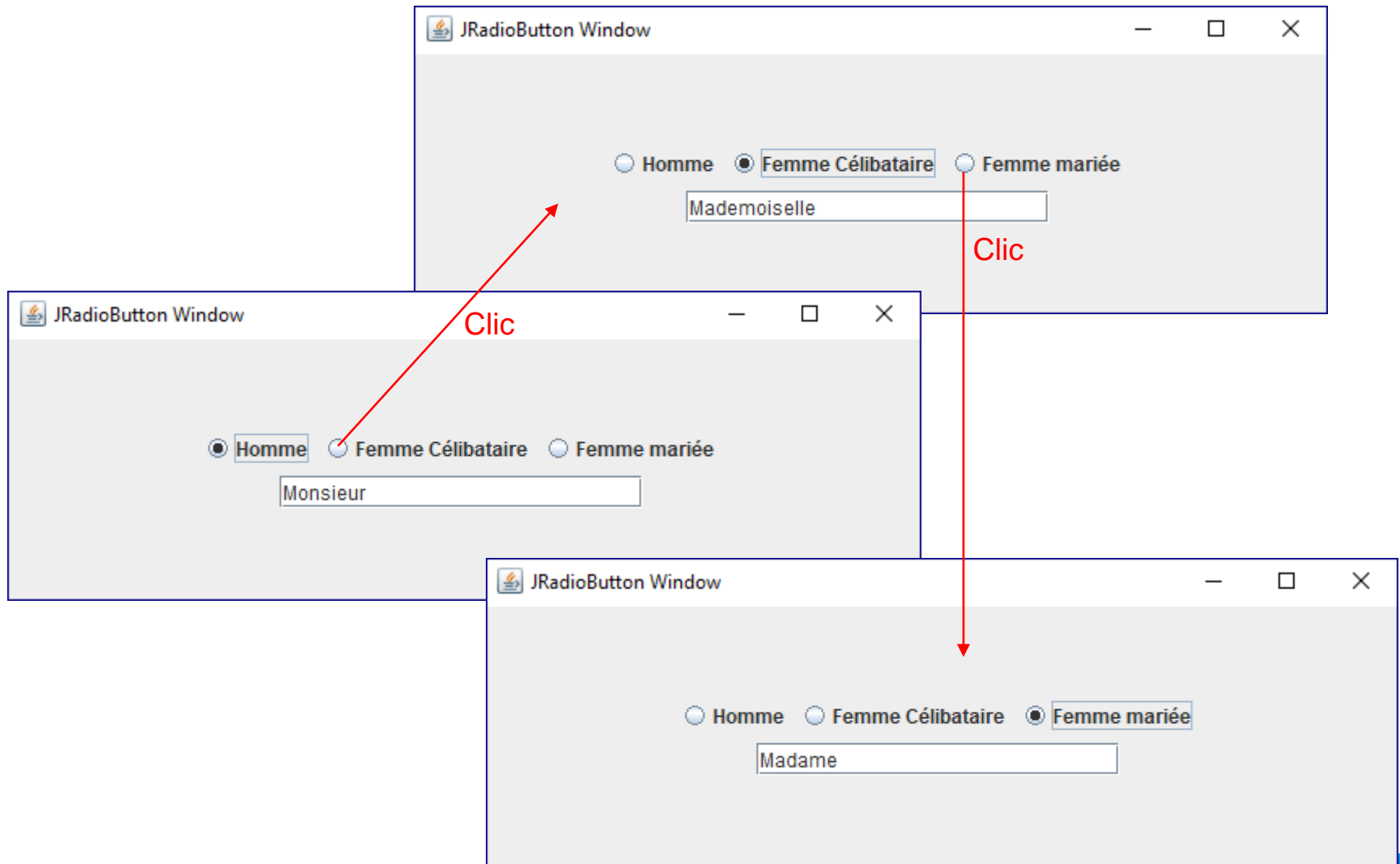
≡

defaultQuantity.isSelected() → return true if selected
return false if not selected

↓
nom du CheckBox

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
8. Boutons
 - JButton
 - JCheckBox
 - JRadioButton

JRadioButton



JRadioButton

- Au lancement du programme
 - le bouton radio Homme est coché
- Boutons radio exclusifs (un seul choix possible)
 - Créer un ButtonGroup
 - Y placer les boutons radio
- Rôle du ButtonGroup
 - Quand l'utilisateur coche un bouton radio
 - ⇒ le ButtonGroup décoche le bouton précédemment coché

```
public class RadioButtonsPanel extends JPanel {
```

```
    private JRadioButton button1, button2, button3;
```

```
    private ButtonGroup buttonGroup;    —————→  Gère le groupe : un seul bouton radio coché à la fois
```

```
    private JTextField text;
```

```
    public RadioButtonsPanel() {
```

```
        ...
```

```
        button1 = new JRadioButton("Homme",true);    —————→  coché
```

```
        this.add(button1);
```

```
        button2 = new JRadioButton("Femme Célibataire",false);    —→  décoché
```

```
        this.add(button2);
```

```
        button3 = new JRadioButton("Femme mariée",false);    —————→  décoché
```

```
        this.add(button3);
```

①
*Ajouter les boutons
au container*

```
        buttonGroup = new ButtonGroup();
```

```
        buttonGroup.add(button1);
```

```
        buttonGroup.add(button2);
```

```
        buttonGroup.add(button3);
```

②
Ajouter les boutons au ButtonGroup

```
public class RadioButtonsPanel extends JPanel {
```

```
...
```

```
public RadioButtonsPanel() {
```

```
...
```

```
RadioButtonListener listener = new RadioButtonListener();
```

```
button1.addItemListener(listener);
```

```
button2.addItemListener(listener);
```

```
button3.addItemListener(listener);
```

```
}
```

→ Crée un écouteur d'évènement

→ Associe l'écouteur aux boutons radio

```
private class RadioButtonListener implements ItemListener {
```

```
public void itemStateChanged( ItemEvent event) { → Appelé si boutons cochés/décochés
```

```
if (event.getSource() == button1 && event.getStateChange() == ItemEvent.SELECTED)  
    text.setText("Monsieur");
```

```
else if (event.getSource() == button2 && event.getStateChange() == ItemEvent.SELECTED)  
    text.setText("Mademoiselle");
```

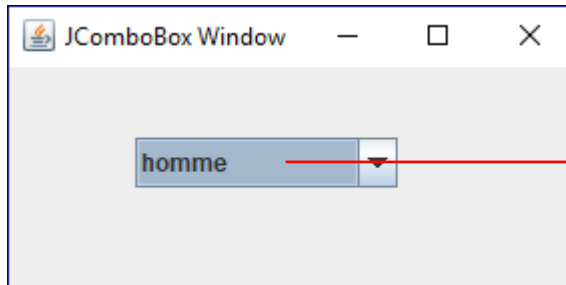
```
else if (event.getSource() == button3 && event.getStateChange() == ItemEvent.SELECTED)  
    text.setText("Madame");
```

```
}
```

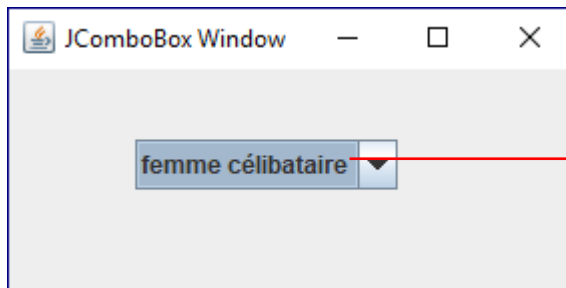
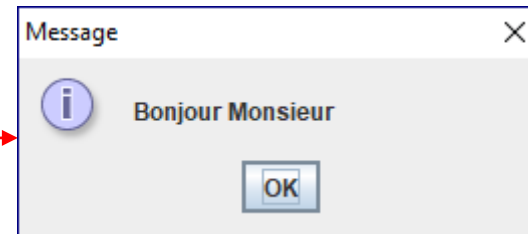
```
}}
```

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
8. Boutons
9. Listes
 - JComboBox

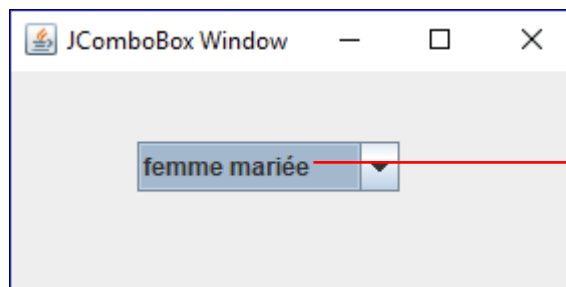
JComboBox



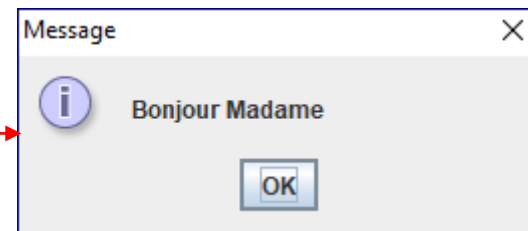
Clic



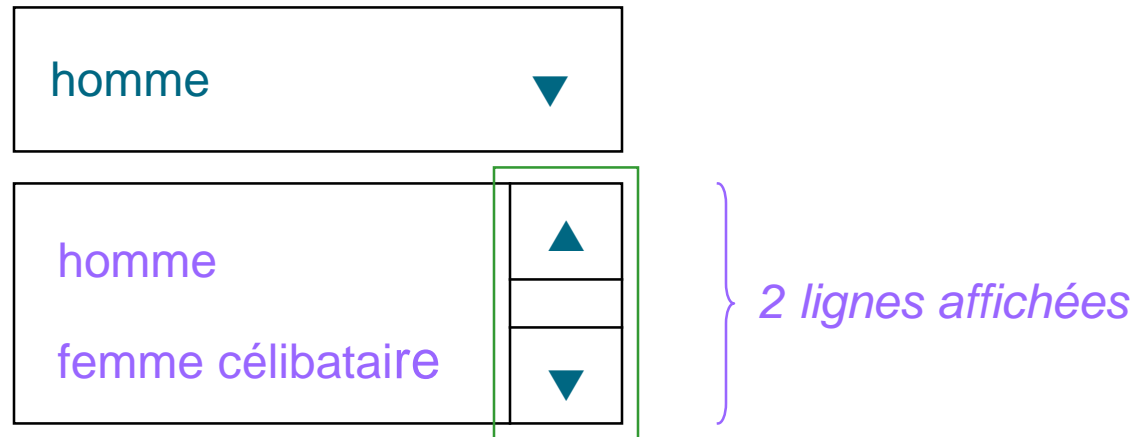
Clic



Clic



JComboBox



Déroulant automatique car plus de deux lignes de contenu

```
public class ComboBoxPanel extends JPanel {
```

```
    private JComboBox combox;
```

```
    public ComboBoxPanel()
```

```
    {    ...
```

```
        String[ ] values = { "homme", "femme célibataire", "femme mariée" };
```

```
        combox = new JComboBox(values);
```

```
        combox.setSelectedItem("homme");
```

→ Valeur sélectionnée par défaut

```
        combox.setMaximumRowCount(2);
```

→ Deux lignes du contenu affichées

```
        this.add(combox);
```

↓
Or, trois valeurs possibles

↓
Déroulant automatique

JComboBox

```
combox.setEditable(true);
```



L'utilisateur peut entrer une valeur
autre que celles proposées

```

public class ComboBoxPanel extends JPanel {
    private JComboBox combox;
    public ComboBoxPanel()
    { ...
        ComboBoxListener listener = new ComboBoxListener();
        combox.addItemListener(listener);
    }
    private class ComboBoxListener implements ItemListener
    { public void itemStateChanged( ItemEvent event)
      { switch (combox.getSelectedIndex())
        { case 0 : if ( event.getStateChange() == ItemEvent.SELECTED )
                    JOptionPane.showMessageDialog(null,"Bonjour Monsieur"); break;
          case 1 : if ( event.getStateChange() == ItemEvent.SELECTED )
                    JOptionPane.showMessageDialog(null,"Bonjour Mademoiselle"); break;
          case 2 : if ( event.getStateChange() == ItemEvent.SELECTED )
                    JOptionPane.showMessageDialog(null,"Bonjour Madame"); break;
        }
      }
    }
}

```

Crée un écouteur d'évènement

Associe l'écouteur à la combo box

Appelé quand choix dans la combo box

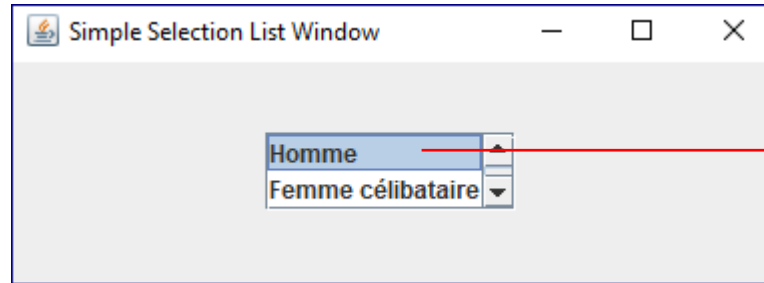
7. Zones de texte

8. Boutons

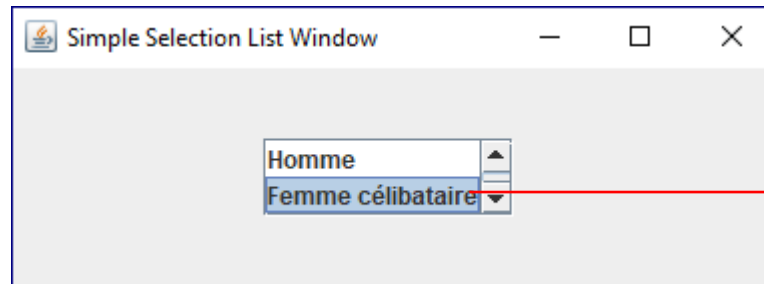
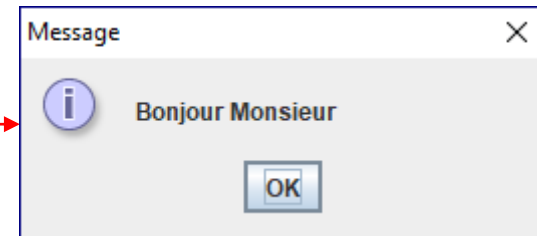
9. Listes

- JComboBox
- JList
 - A sélection simple

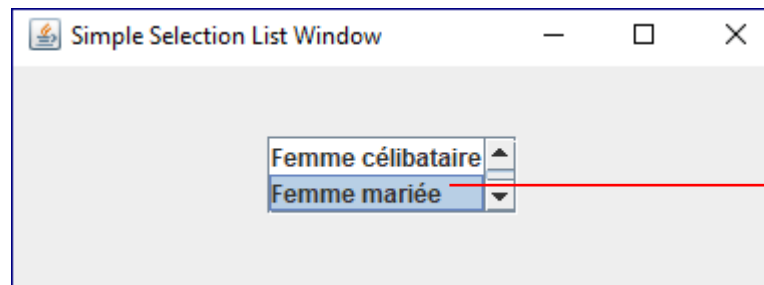
JList



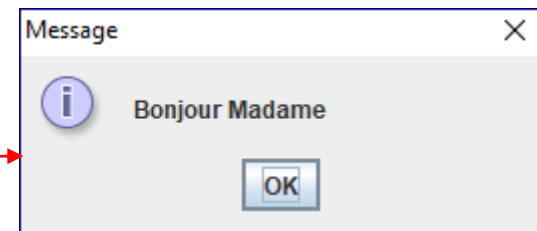
Clic



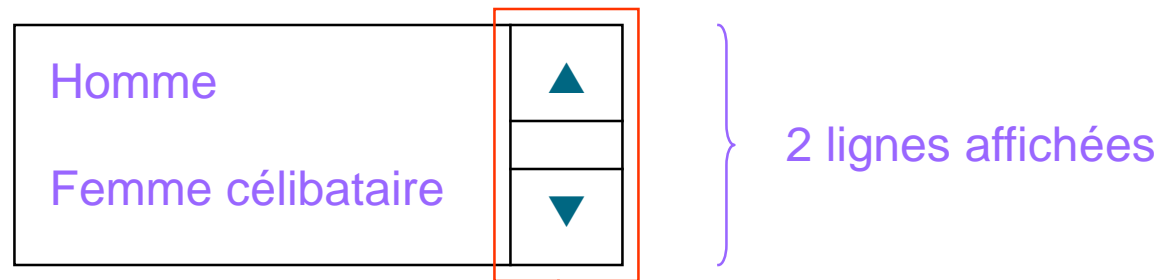
Clic



Clic



JList



Déroulant pas automatique sur JList

JList

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

```

public class SimpleSelectionListPanel extends JPanel {
    private JList simpleSelectionList;

    public SimpleSelectionListPanel()
    { ...
        String[ ] values = { "Homme","Femme célibataire","Femme mariée" };
        simpleSelectionList = new JList(values);
        simpleSelectionList.setVisibleRowCount(2); —————→ Deux lignes du contenu affichées
        simpleSelectionList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        ↓
        Détermine le type de sélection :
        this.add(new JScrollPane(simpleSelectionList));
        ↓
        Créer explicitement un déroulant
    }

```

Attention: on place le déroulant dans le container et pas la JList!

Nouveau type d'écouteur :

Interface **ListSelectionListener**

capable de détecter la sélection d'une ou plusieurs valeurs
dans une liste

```

public class SimpleSelectionListPanel extends JPanel {
    private JList simpleSelectionList;

    public SimpleSelectionListPanel()
    { ...
        ListListener listener = new ListListener(); —————> Crée un écouteur d'évènement
        simpleSelectionList.addListSelectionListener(listener);
        —————> Associe l'écouteur à la liste
    }

```

```

private class ListListener implements ListSelectionListener
{ public void valueChanged( ListSelectionEvent event )
    Appelé à chaque nouvelle sélection dans la liste
    { switch ( simpleSelectionList.getSelectedIndex() )
        Retourne l'index de la valeur sélectionnée
        { case 0 : JOptionPane.showMessageDialog(null,"Bonjour monsieur"); break;
          case 1 : JOptionPane.showMessageDialog(null,"Bonjour mademoiselle"); break;
          case 2 : JOptionPane.showMessageDialog(null,"Bonjour madame"); break;
        }
    }
}

```

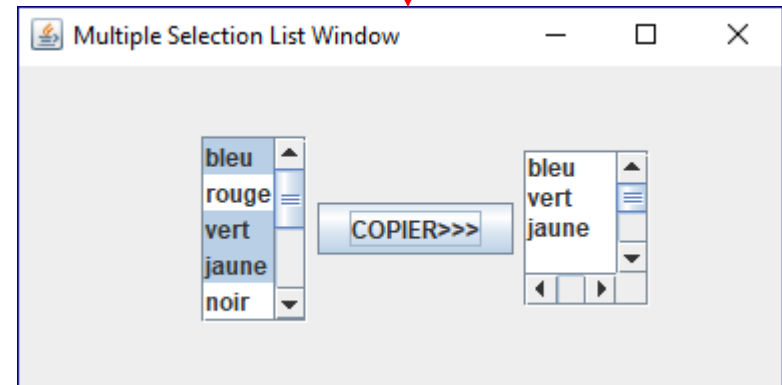
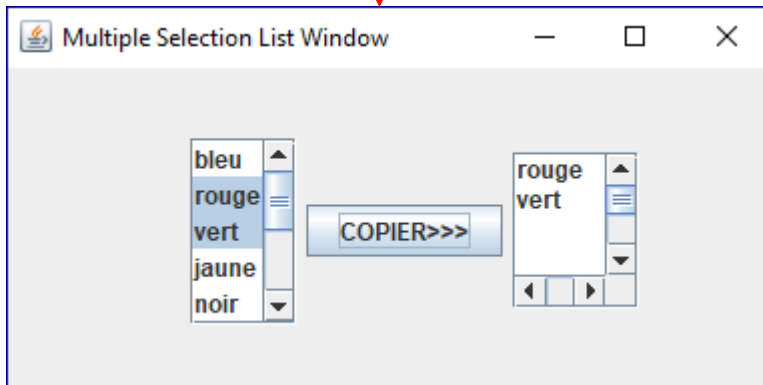
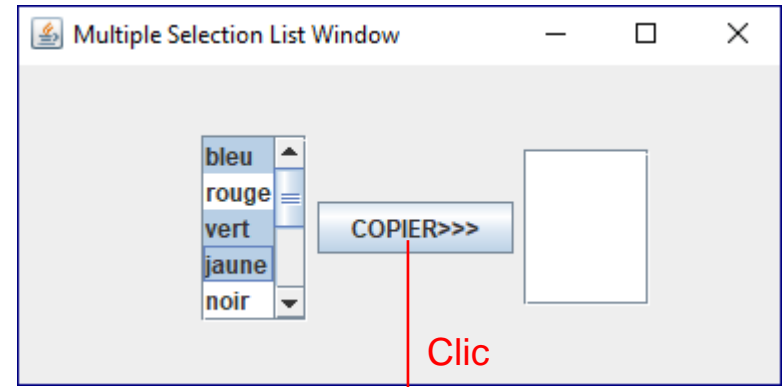
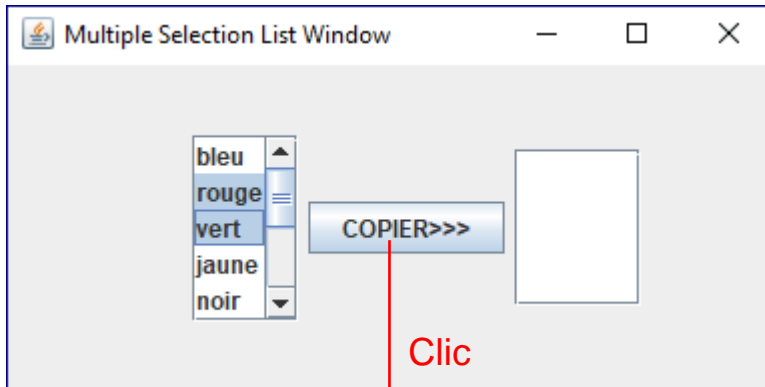
7. Zones de texte

8. Boutons

9. Listes

- JComboBox
- JList
 - A sélection simple
 - A sélection Multiple

JList – Sélection multiple



```
public class MultipleSelectionListPanel extends JPanel {  
    private JList colors, chosenColors;  
    private JButton bouton;  
    public MultipleSelectionListPanel()  
    { ...  
        String[ ] colorNames = {"bleu","rouge","vert","jaune","noir","blanc","violet","rose","gris","brun"};  
        colors = new JList(colorNames);  
        colors.setVisibleRowCount(5);  
        colors.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);  
        this.add(new JScrollPane(colors));  
    }  
}
```

Détermine le type de sélection :

Cinq lignes affichées

L'utilisateur peut sélectionner plusieurs options dans la JList

Créer explicitement un déroulant

Obligatoires car liste vide

```
chosenColors = new JList();
```

```
chosenColors.setVisibleRowCount(5); —————→ Cinq lignes affichées
```

```
chosenColors.setFixedCellWidth(60); —————→ Détermine la largeur à l'affichage
```

```
chosenColors.setFixedCellHeight(15); —————→ Détermine la hauteur à l'affichage
```

```
chosenColors.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
```

```
this.add(new JScrollPane(chosenColors));
```

L'utilisateur ne peut sélectionner qu'une seule option dans la JList


```
bouton = new JButton("COPIER>>>");
```

```
ButtonListener listener = new ButtonListener();
```

```
bouton.addActionListener(listener); —————> C'est le bouton qui est écouté !
```

```
this.add(bouton);
```

```
}
```

```
private class ButtonListener implements ActionListener
```

```
{ public void actionPerformed( ActionEvent event)    Appelé quand clic sur le bouton
```

```
{ Object[ ] selectedColors = ...
```

```
for (Object color : colors.getSelectedValuesList() )  Récupère les valeurs sélectionnées
```

```
{ // remplir selectedColors }
```

```
chosenColors.setListData(selectedColors);           Modifie les valeurs de la liste
```

```
MultipleSelectionListPanel.this.repaint();
```

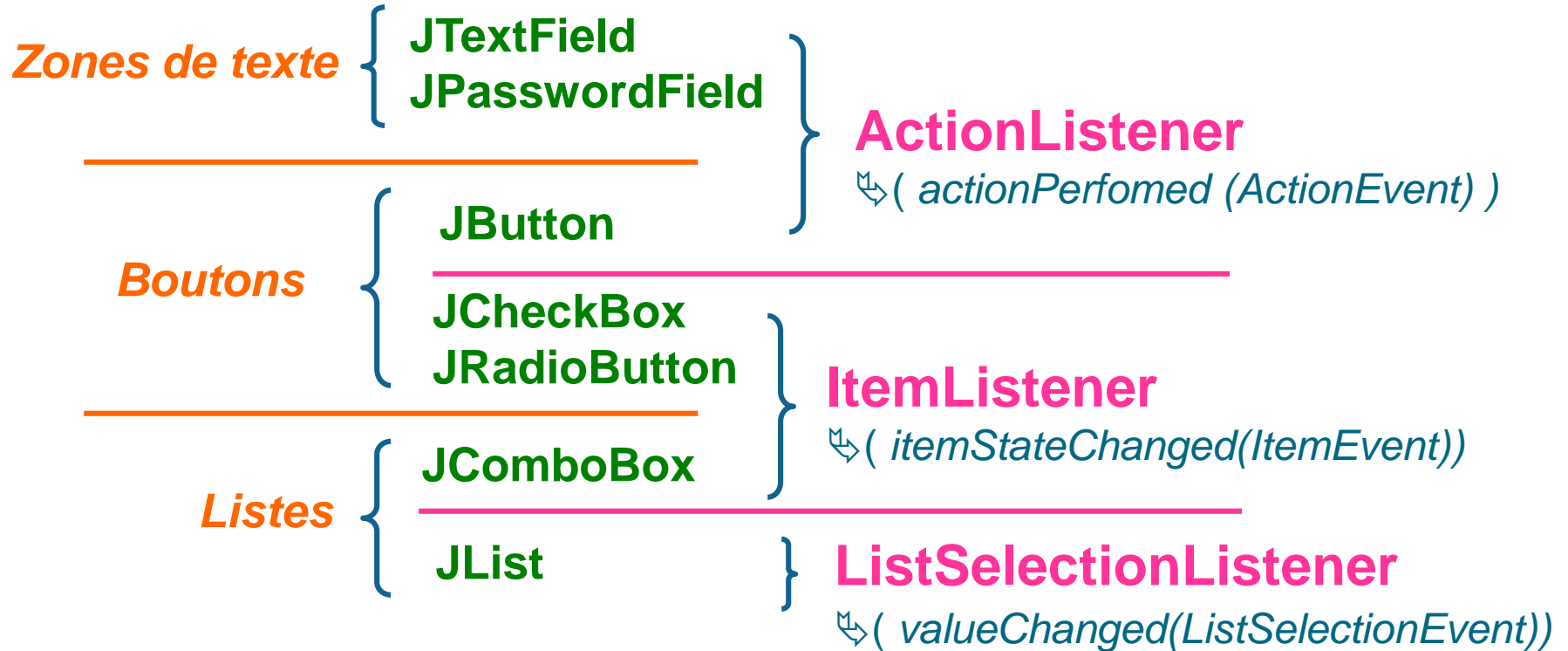
```
}           Pour redessiner le panneau
```

```
}
```

```
}
```

Gestion des événements

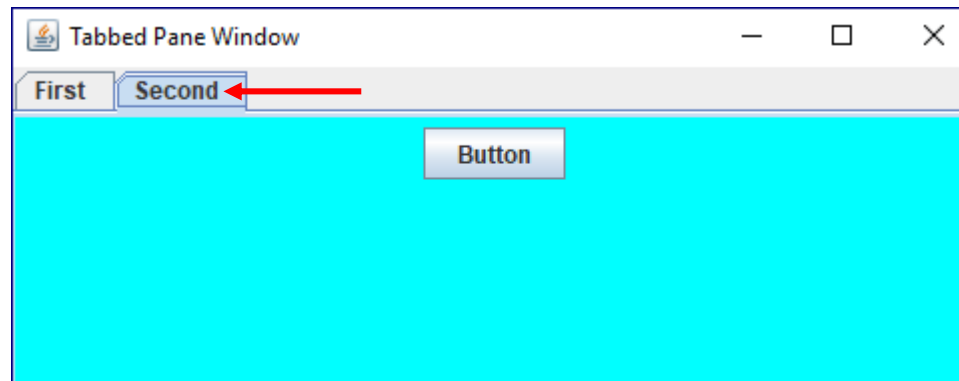
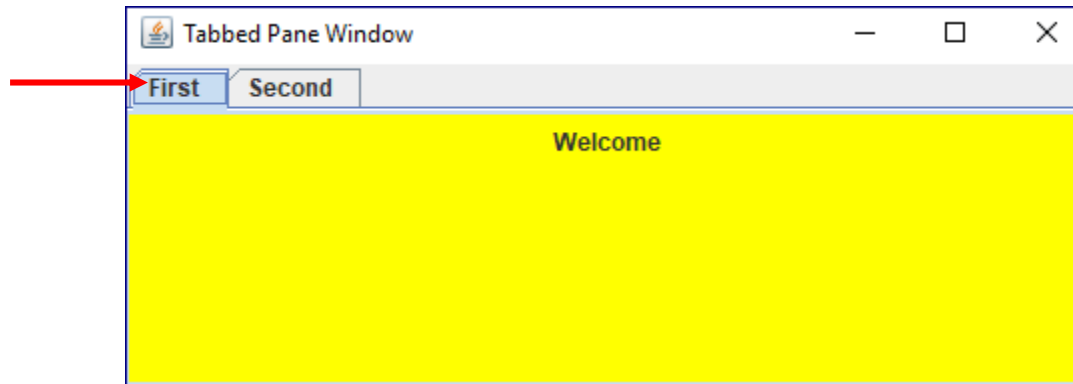
RESUME



Swing

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
8. Boutons
9. Listes
10. JTabbedPane (onglets)

JTabbedPane



```

public class TabbedPanel extends JPanel {
    private JLabel label;
    private JButton button;
    private JPanel panel1, panel2;
    private JTabbedPane tabbedPane;
    public TabbedPanel()
    { panel1 = new JPanel();           → Panneau du premier onglet
      panel1.setBackground(Color.YELLOW);
      label = new JLabel("Welcome");
      panel1.add(label);

      panel2 = new JPanel();           → Panneau du second onglet
      panel2.setBackground(Color.CYAN);
      button = new JButton("Button");
      panel2.add(button);

      tabbedPane = new JTabbedPane();
      tabbedPane.insertTab( "First ", null, panel1, "This is the first panel", 0);
      tabbedPane.insertTab( "Second ", null, panel2, "This is the second panel", 1);

      this.setLayout(new BorderLayout());
      this.add(tabbedPane, BorderLayout.CENTER); → JTabbedPane placé au centre
    }
}

```

JTabbedPane

tabbedPane.insertTab("First ", null, panneau1, "This is the first panel", 0);

Titre de l'onglet

Composant

Index

Image (Icon)

Bulle d'aide

JTabbedPane

```
System.out.println("Nombre d'onglets : " + tabbedPane.getComponentCount( ));
```

↓
Nombre d'onglets

Attention au casting

```
Component component = (JPanel) ( tabbedPane.getComponentAt(1) );
```

↓
Retourne le second composant

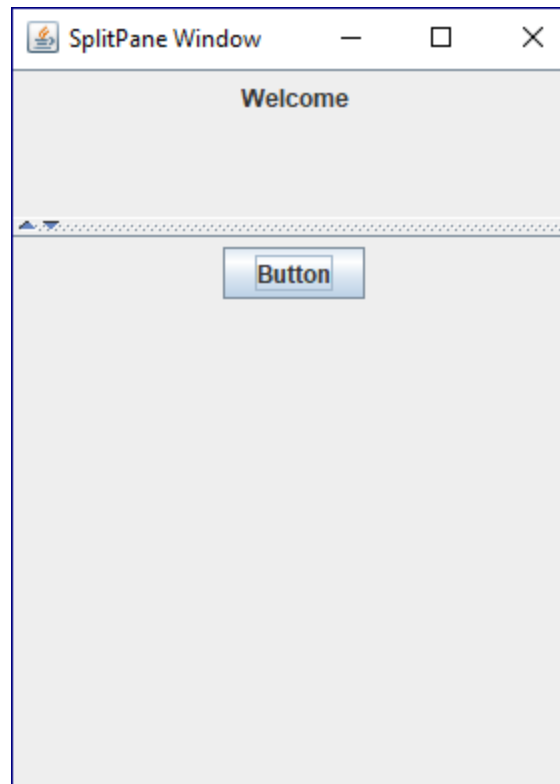
```
component.setBackground(java.awt.Color.RED);
```

↓
Couleur rouge

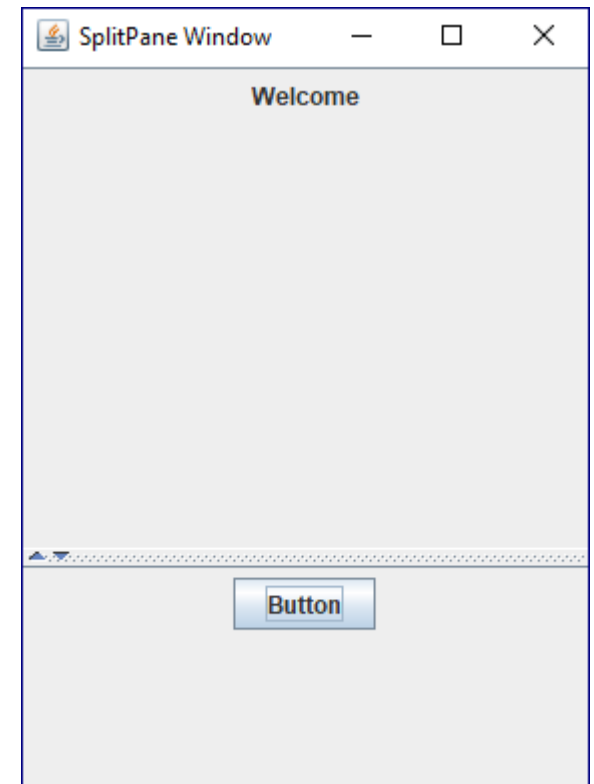
Swing

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
8. Boutons
9. Listes
10. JTabbedPane
11. JSplitPane

JSplitPane



Si tiré vers le haut



Si tiré vers le bas

```

public class SplitPanel extends JPanel {
    private JLabel label;
    private JButton button;
    private JPanel panel1, panel2;
    private JSplitPane splitPane;

    public SplitPanel()
    { panel1 = new JPanel();
      label = new JLabel("Welcome");
      panel1.add(label);

      panel2 = new JPanel();
      button = new JButton("Button");
      panel2.add(button);

      splitPane = new JSplitPane( JSplitPane.VERTICAL_SPLIT, true, panel1, panel2 );
      splitPane.setOneTouchExpandable(true);
      splitPane.setDividerLocation(150);

      this.setLayout(new BorderLayout());
      this.add(splitPane, BorderLayout.CENTER);
    }
}

```

Extension verticale



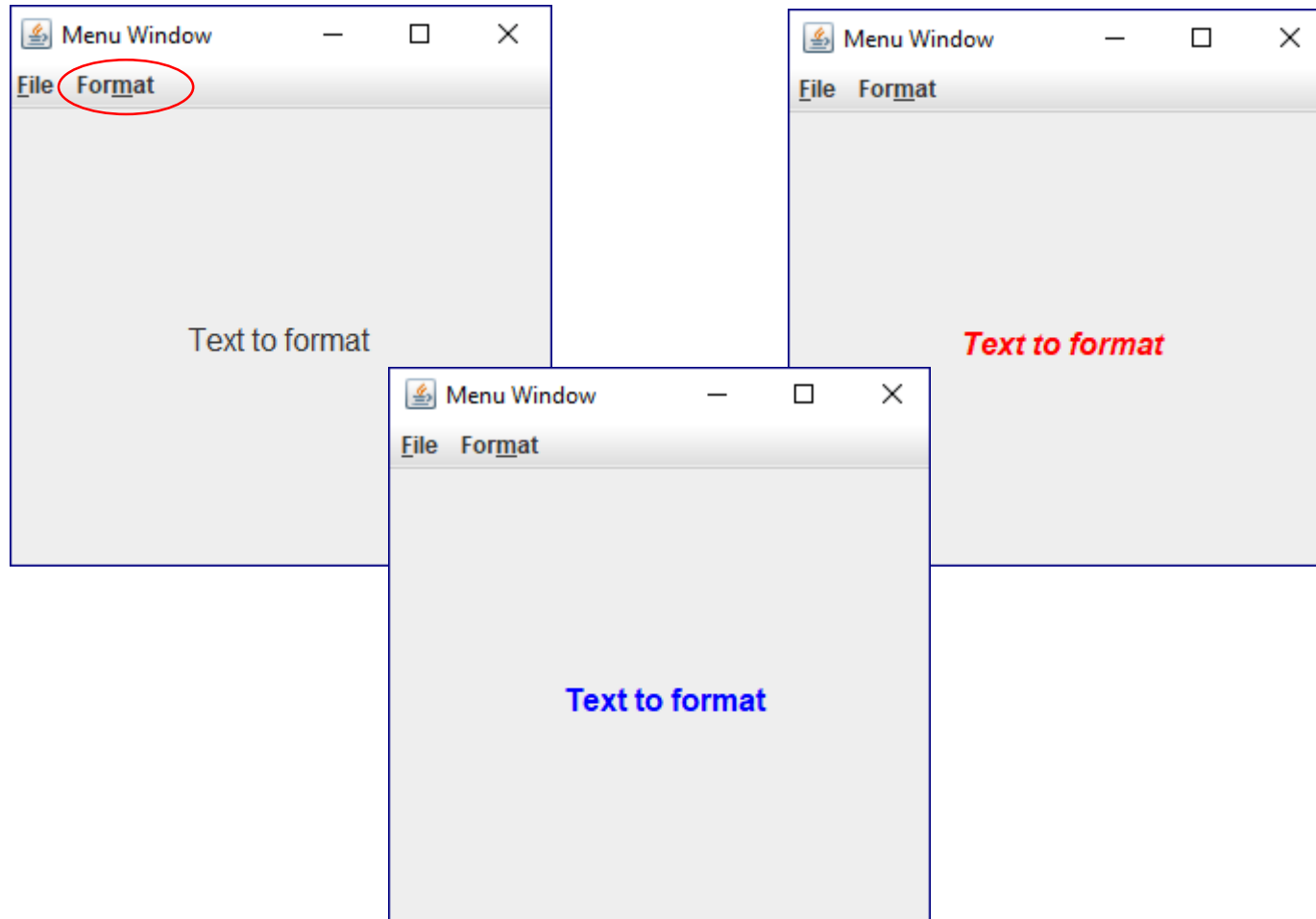
Affiche: ▲▼

Position de la séparation

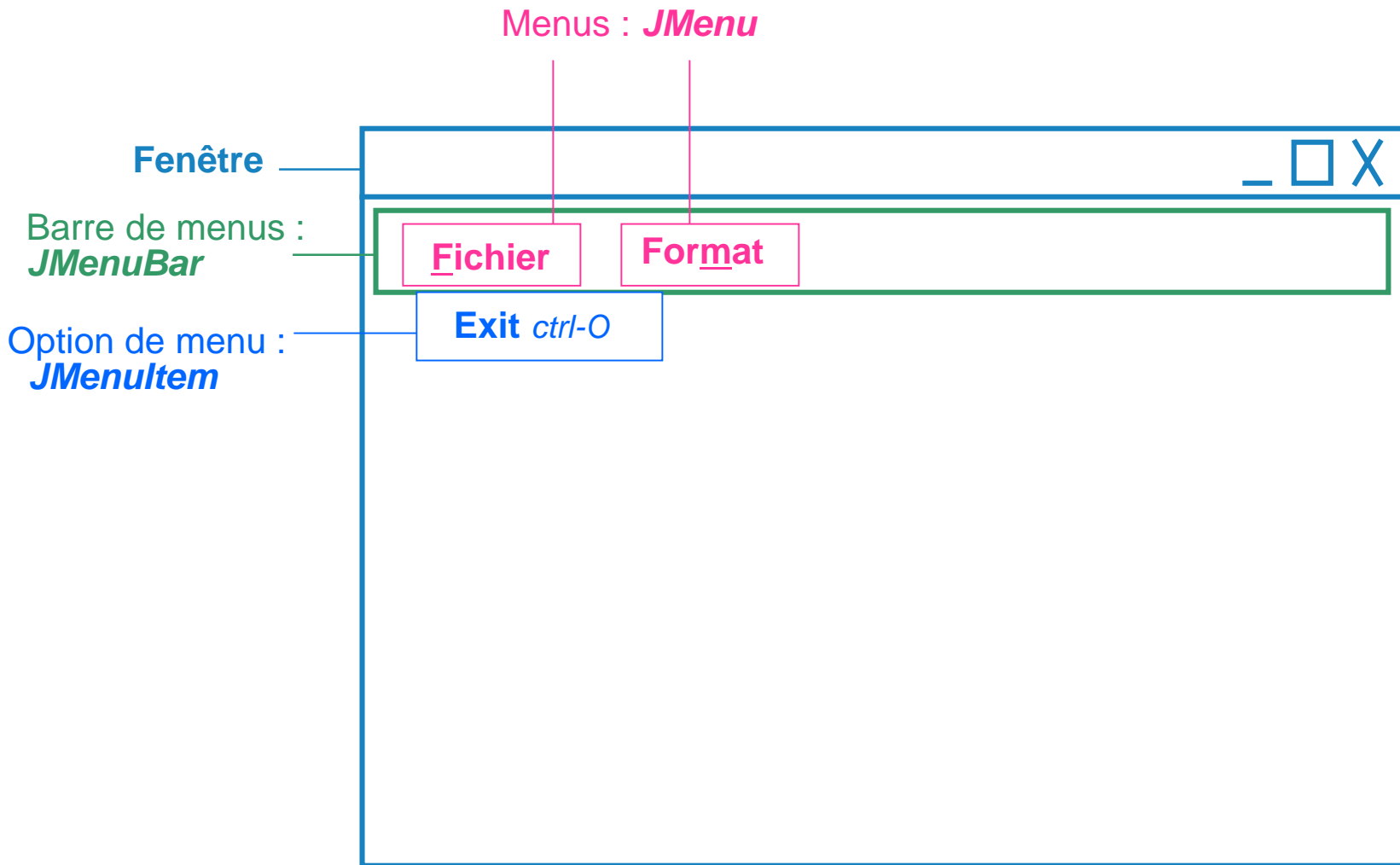
Swing

1. JOptionPane
2. JFrame
3. JPanel
4. Affichage de composants dans un conteneur
5. Observer Pattern
6. Classe interne
7. Zones de texte
8. Boutons
9. Listes
10. JTabbedPane
11. JSplitPane
12. Menus

Menus



Menus



```
public class MenuWindow extends JFrame {  
    private JMenuBar menuBar;  
    private JMenu fileMenu, formatMenu, ... ;  
    private JMenuItem exit; ...
```

```
public MenuWindow()  
{ ...
```

```
    menuBar = new JMenuBar();  
    setJMenuBar(menuBar);
```

→ Ajoute la barre de menus à la fenêtre

```
    fileMenu = new JMenu("File");
```

→ Crée un menu

```
    fileMenu.setMnemonic('F');
```

→ Raccourci mnémonique: alt + F

```
    menuBar.add(fileMenu);
```

→ Ajoute le menu *Fichier* à la barre de menus

```
    exit = new JMenuItem("Exit");
```

→ Crée une option de menu

```
    exit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, InputEvent.CTRL_MASK));
```

```
    fileMenu.add(exit);
```

→ Accélérateur clavier: ctrl + O

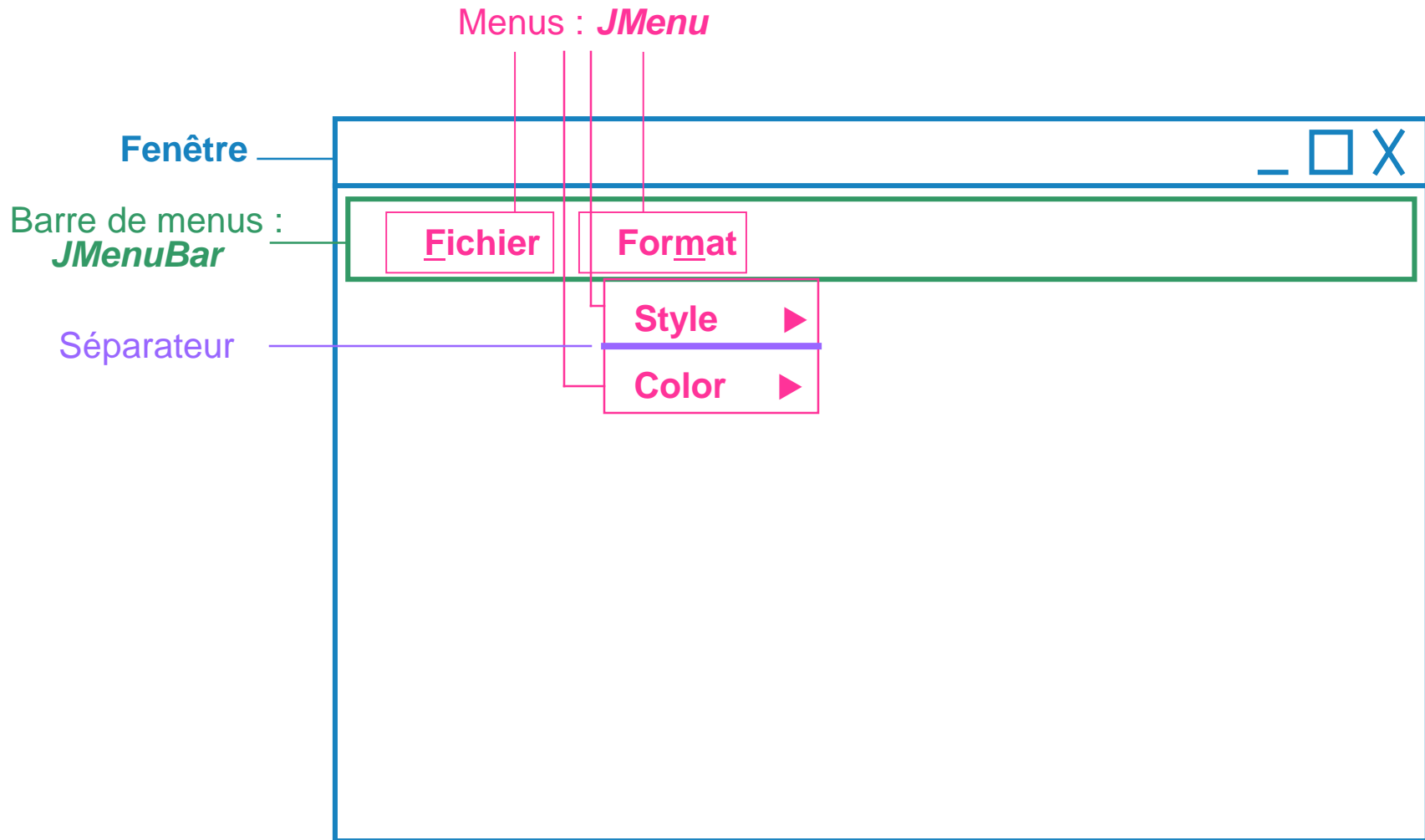
→ Ajoute l'option de menu *Sortie* au menu *Fichier*

```
    formatMenu = new JMenu("Format");
```

```
    formatMenu.setMnemonic('m');
```

```
    menuBar.add(formatMenu);
```

Menus



Menus

```
private JMenu styleMenu, colorMenu;
```

Variables d'instance

Constructeur

```
styleMenu = new JMenu("Style");
```

```
formatMenu.add(styleMenu);
```

→ Ajoute le menu *Style* au menu *Format*

```
formatMenu.addSeparator();
```

→ Ajoute un séparateur

```
colorMenu = new JMenu("Color");
```

```
formatMenu.add(colorMenu);
```

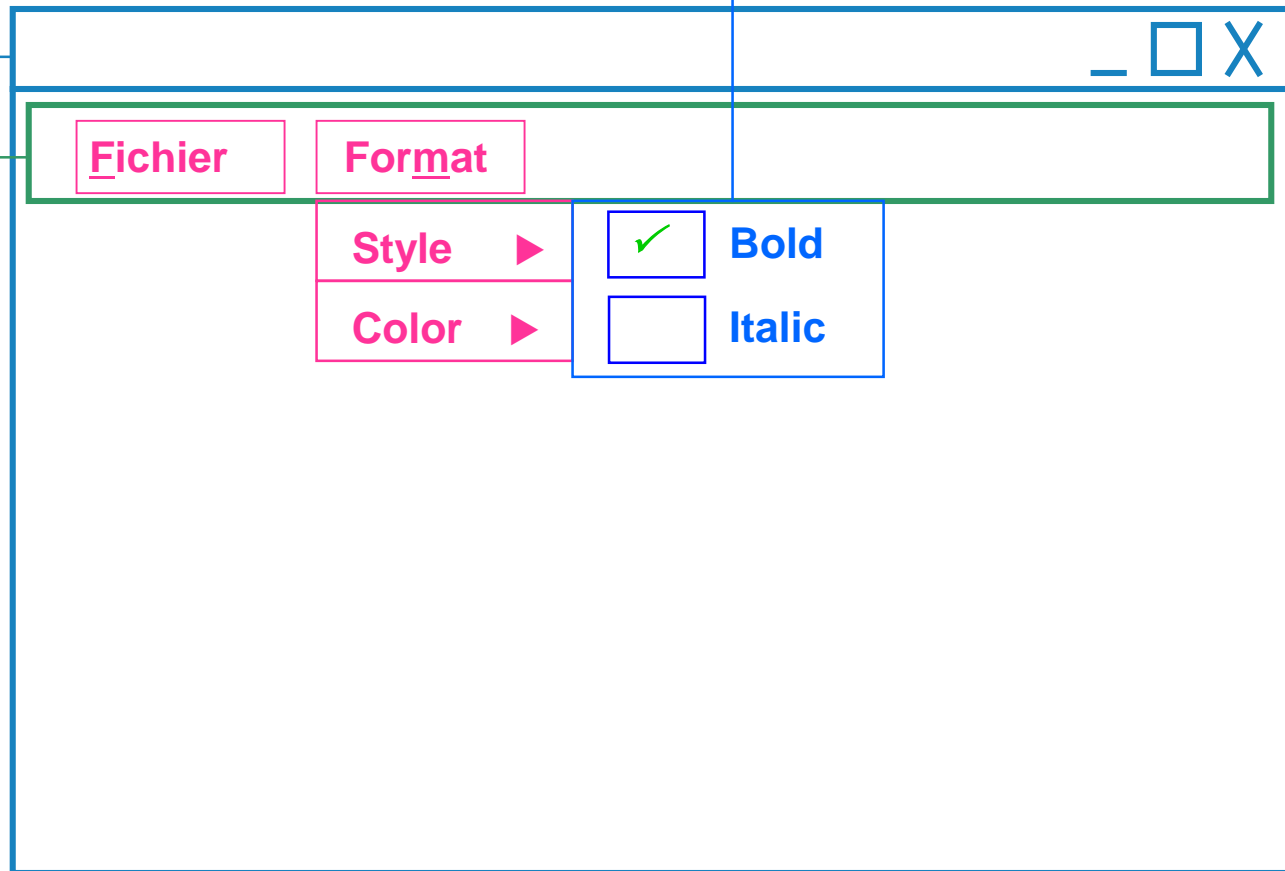

Menus

Menus : **JMenu**

Menus de type case à cocher :
JCheckBoxMenuItem

Fenêtre

Barre de menus :
JMenuBar



Menus

```
private JCheckBoxMenuItem bold, italic;
```

Variables d'instance

Constructeur

```
bold = new JCheckBoxMenuItem("Bold");
```

```
styleMenu.add(bold);
```

 Ajoute l'option de menu *Bold* au menu *Style*

```
italic = new JCheckBoxMenuItem("Italic");
```

```
styleMenu.add(italic);
```

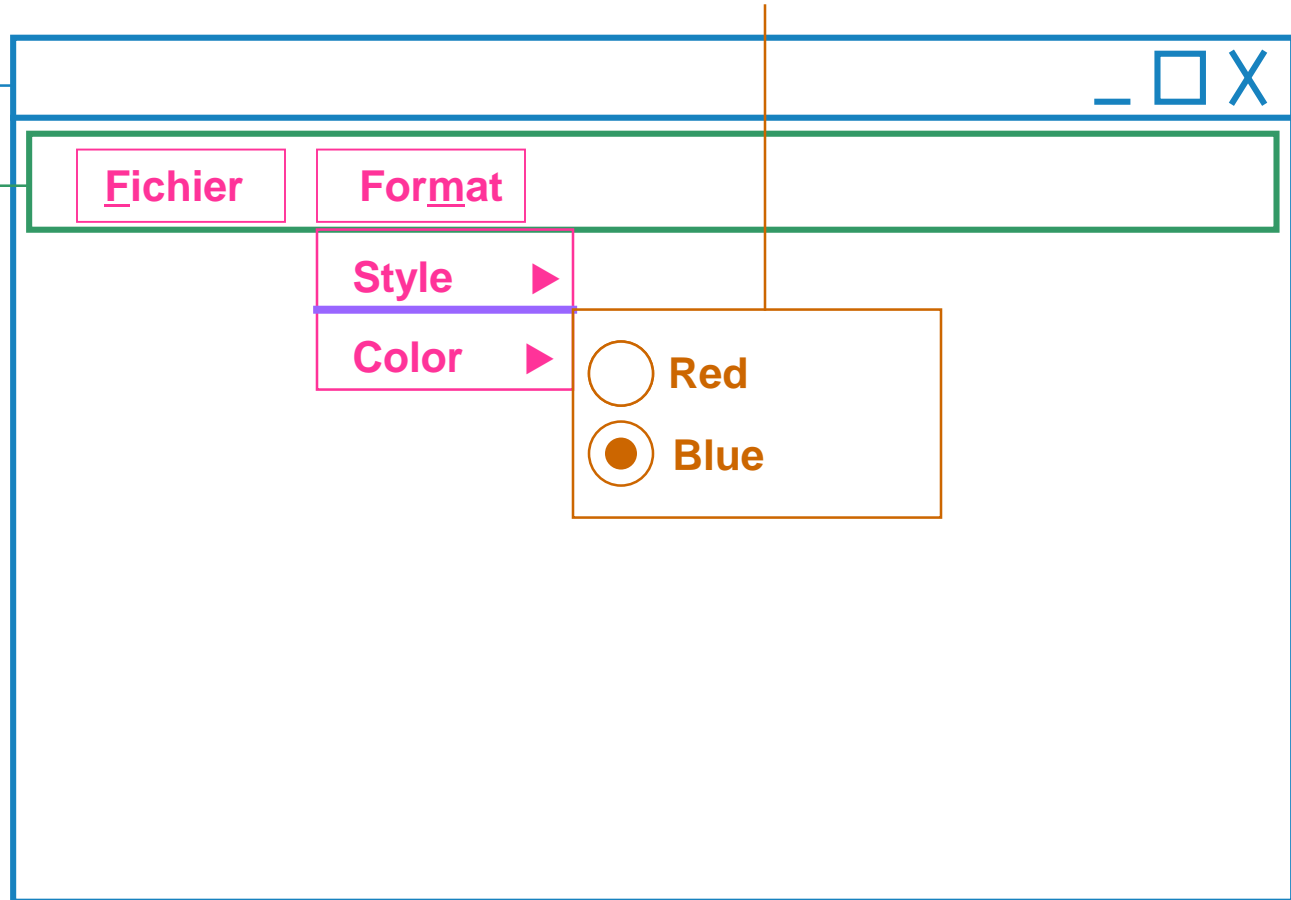
Menus

Menus : *JMenu*

Menus de type bouton radio :
JRadioButtonMenuItem

Fenêtre

Barre de menus :
JMenuBar



Menus

```
private JRadioButtonMenuItem red, blue;  
private ButtonGroup radioButtonGroup;
```

Variables d'instance

Constructeur

```
red = new JRadioButtonMenuItem("Red");  
colorMenu.add(red);
```

—————→ Ajoute l'option de menu *Red* au menu *Color*

```
blue = new JRadioButtonMenuItem("Blue");  
colorMenu.add(blue);
```

Rappel : ButtonGroup :
un seul bouton radio coché à la fois

```
radioButtonGroup = new ButtonGroup();  
radioButtonGroup.add(red);  
radioButtonGroup.add(blue);
```

Menus

```
private Container frameContainer;  
private JLabel text;  
private int currentStyle = Font.PLAIN;
```

Variables d'instance

Constructeur

```
text = new JLabel("Text to format", SwingConstants.CENTER);
```

```
text.setFont(new Font("Helvetica", Font.PLAIN, 16));
```

```
frameContainer = getContentPane();
```

```
frameContainer.add(text, BorderLayout.CENTER);
```

Menus – Gestion événements

```
exit = new JMenuItem("Exit");
```

```
ExitListener exitListener = new ExitListener();
```

```
exit.addActionListener(exitListener);
```

```
private class ExitListener implements ActionListener
```

```
{
```

```
    public void actionPerformed (ActionEvent event)
```

```
    {    System.exit(0);
```

```
    }
```

```
}
```

Menus – Gestion événements

```
CheckBoxMenuItemListener checkBoxMenuItemListener = new CheckBoxMenuItemListener();  
bold.addItemListener(checkBoxMenuItemListener);  
italic.addItemListener(checkBoxMenuItemListener);
```

```
private class CheckBoxMenuItemListener implements ItemListener
```

```
{ public void itemStateChanged(ItemEvent event)
```

```
{    currentStyle = Font.PLAIN;
```

PLAIN = 0

```
    if (bold.isSelected())
```

```
        currentStyle += Font.BOLD;
```

BOLD = 1

```
    if (italic.isSelected())
```

```
        currentStyle += Font.ITALIC;
```

ITALIC = 2

```
    text.setFont(new Font("Helvetica", currentStyle, 16));
```

```
    }
```

Menus – Gestion événements

```
RadioMenuListener radioListener = new RadioMenuListener();  
red.addItemListener(radioListener);  
blue.addItemListener(radioListener);
```

```
private class RadioMenuListener implements ItemListener  
{ public void itemStateChanged(ItemEvent event)  
    { if (event.getSource() == red  
        && event.getStateChange() == ItemEvent.SELECTED)  
        text.setForeground(Color.RED);  
        else text.setForeground(Color.BLUE);  
    }  
}
```


7. Zones de texte

8. Boutons

9. Listes

10. JTabbedPane

11. JSplitPane

12. Menus

13. Gestion des événements sur une fenêtre

Gestion d'événements sur la fenêtre

On peut écouter la fenêtre

⇒ réagir aux différents événements sur la fenêtre :

- Fermeture
- Agrandissement
- Minimisation (iconification)
- "Dé-iconification"
- ...



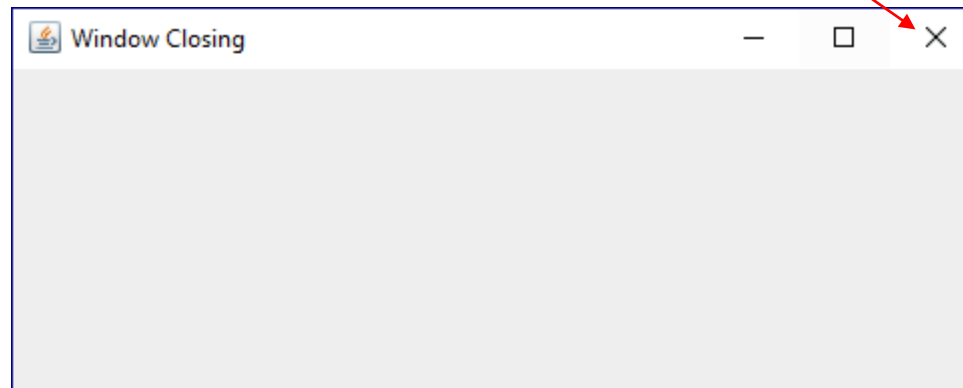
Gestion de la fermeture de la fenêtre

Exemple de gestion à faire

Si utilisation de composants Swing

⇒ `System.exit(0)` pour terminer l'application

⇒ `System.exit(0)` à exécuter si clic sur



Gestion de la fermeture de la fenêtre

Nouveau type d'écouteur :

Interface **WindowListener**

Capable de détecter et réagir aux événements suivants

- *Window closing*
- *Window opened*
- *Window closed*
- *Window iconified*
- *Window deiconified*
- *Window activated*
- *Window deactivated*

Gestion de la fermeture de la fenêtre


Version 1

```
public class WindowClosing extends JFrame
{
    public WindowClosing() {
        super("Window Closing");

        setBounds(100,100,500,200);

        this.addWindowListener (new ClosingListener());

        setVisible(true);
    }
}
```



C'est la fenêtre qu'on écoute

Gestion de la fermeture de la fenêtre

```
public class ClosingListener implements WindowListener
```

```
{
```

```
    public void windowClosing (WindowEvent event)
```

```
    { System.exit(0); }
```

Méthode appelée automatiquement lors
de la fermeture de la fenêtre

```
    public void windowOpened (WindowEvent event) { }
```

```
    public void windowClosed (WindowEvent event) { }
```

```
    public void windowIconified (WindowEvent event) { }
```

```
    public void windowDeiconified (WindowEvent event) { }
```

```
    public void windowActivated (WindowEvent event) { }
```

```
    public void windowDeactivated (WindowEvent event) { }
```

```
}
```

Gestion de la fermeture de la fenêtre

Version 2

La classe **WindowAdapter** (classe existante) permet d'écrire moins de code dans la classe écouteur

```
public abstract class WindowAdapter implements WindowListener, ...  
{  
    public void windowClosing (WindowEvent e) {} —————→ Implémentation vide  
    public void windowOpened (WindowEvent e) {}  
    public void windowClosed (WindowEvent e) {}  
    public void windowIconified (WindowEvent e) {}  
    public void windowDeiconified (WindowEvent e) {}  
    public void windowActivated (WindowEvent e) {}  
    public void windowDeactivated (WindowEvent e) {}  
}
```

Gestion de la fermeture de la fenêtre

```
public class WindowClosing extends JFrame
{
    public WindowClosing()
    {
        ...

        this.addWindowListener(new ClosingListener());

        setVisible(true);
    }

    public class ClosingListener extends WindowAdapter
    {
        public void windowClosing( WindowEvent e)
        { System.exit(0); }
    }
}
```


Gestion de la fermeture de la fenêtre

Version 3 : Classe interne anonyme

```
public class WindowClosing extends JFrame
```

```
{
```

```
    public WindowClosing()
```

```
    { super("Window Closing");
```

```
        ...
```

```
        this.addWindowListener( new WindowAdapter()
```

```
        { public void windowClosing( WindowEvent e)
```

```
            { System.exit(0); }
```

```
        } );
```

```
        setVisible(true);
```

```
    }
```

```
}
```

Crée une occurrence
d'une sous-classe anonyme
de la classe WindowAdapter

Redéfinition de la méthode
windowClosing(...) héritée

7. Zones de texte

8. Boutons

9. Listes

10. JTabbedPane

11. JSplitPane

12. Menus

13. Gestion des événements sur une fenêtre

14. Différentes versions de gestion d'événements

Gestion des événements

Version 1 : Via une classe interne

```
public class ButtonPanel extends JPanel {  
    private JButton button;  
    public ButtonPanel()  
    {  
        ...  
        button = new JButton("...");  
  
        ButtonListener listener = new ButtonListener();  
        button.addActionListener(listener);  
    }  
  
    private class ButtonListener implements ActionListener  
    {  
        public void actionPerformed( ActionEvent event)  
        {  
            // Code de la réaction à l'événement  
        }  
    }  
}
```

Gestion des événements

Version 2 : Via une classe interne anonyme

```
public class ButtonPanel extends JPanel {
```

```
    private JButton button;
```

```
    public ButtonPanel()
```

```
    { ...  
      button = new JButton("...");
```

Crée une occurrence d'une classe anonyme
qui implémente l'interface ActionListener

```
      button.addActionListener( new ActionListener()  
                                { public void actionPerformed( ActionEvent event)  
                                  { // Code de la réaction à l'événement }  
                                }  
                                );
```

Implémentation de la
méthode actionPerformed

```
    }  
}
```

Gestion des événements

Version 3 : Via une classe qui est son propre écouteur

```
public class ButtonPanel extends JPanel implements ActionListener {  
  
    private JButton button;  
  
    public ButtonPanel()  
    {  
        ...  
        button = new JButton("...");  
  
        button.addActionListener(this);  
    }  
  
    public void actionPerformed( ActionEvent event)  
    {  
        // Code de la réaction à l'événement  
    }  
}
```

The diagram consists of a red rectangular box enclosing the `addActionListener(this);` line in the constructor and the `actionPerformed` method. A red arrow points from the `(this)` argument in `addActionListener` to the `implements ActionListener` text. Another red arrow points from the `actionPerformed` method signature to the `implements ActionListener` text.

- 7. Zones de texte
- 8. Boutons
- 9. Listes
- 10. JTabbedPane
- 11. JSplitPane
- 12. Menus
- 13. Gestion des événements sur une fenêtre
- 14. Différentes versions de gestion d'événements
- 15. JPanel avec référence vers la JFrame parent

Référence vers la JFrame parent

```
public class PanelWithReferenceToParent extends JPanel {  
  
    private ParentWindow parentWindow;  
  
    public PanelWithReferenceToParent (ParentWindow parentWindow)  
    {  
        ...  
  
        this.parentWindow = parentWindow;  
  
        ...  
    }  
}
```

Référence vers la JFrame parent

```
public class ParentWindow extends JFrame {  
  
    public ParentWindow()  
  
    {  
        ...  
  
        PanelWithReferenceToParent panel = new PanelWithReferenceToParent(this);  
  
    }  
}
```