

Introduction à JAVA

Les interfaces SWING



Benoit Combemale

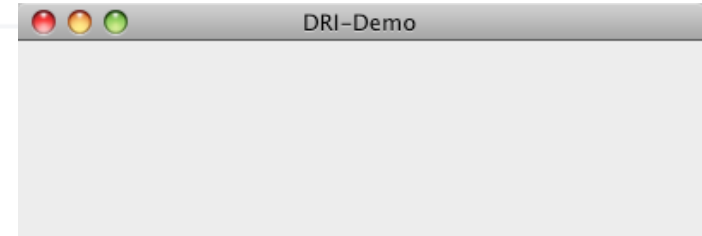
Université de Rennes 1

Équipe-Projet Triskell (INRIA & IRISA)

benoit.combemale@irisa.fr

Sur la base du cours de Grégory Nain

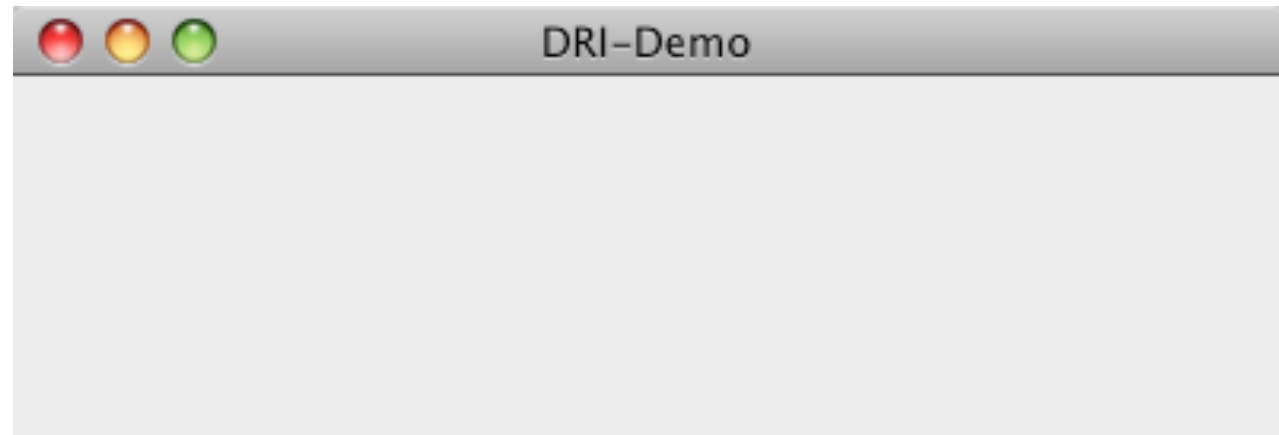
// JFrame



- Permet de créer une fenêtre
- Contient des « composants »
- Peut déterminer l'action à réaliser à la fermeture
- Possède un titre
- Méthode `pack()` pour « ajuster » la fenêtre.



// JFrame



```
JFrame frame = new JFrame("DRI-Demo");  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
frame.setPreferredSize(new Dimension(400,200));  
frame.pack();  
frame.setVisible(true);
```



// JLabel

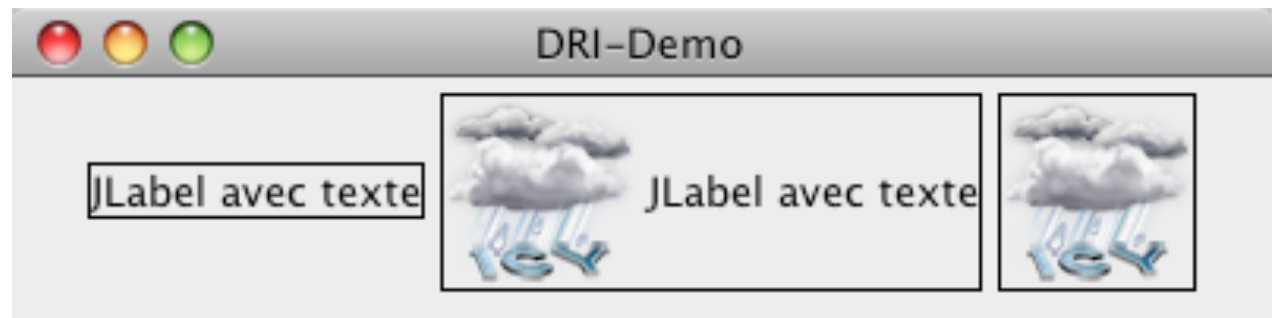


- Texte purement statique, non modifiable
- Peut être accompagné d'une image
- Peut être une simple image
- Peut être associé à un autre composant (JTextField par exemple)
- Peut avoir (ou non) une bordure



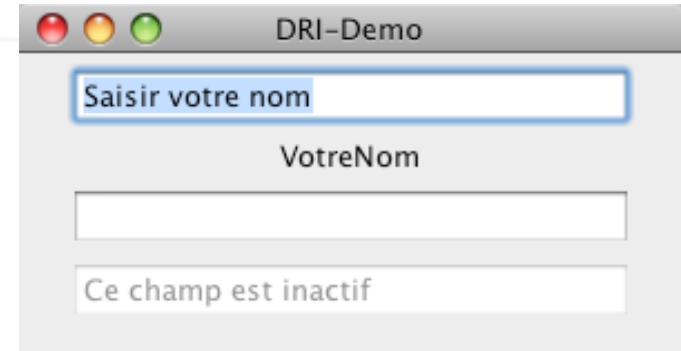
// JLabel

```
JLabel texte, textImage, image;  
texte = new JLabel("JLabel avec texte");  
textImage = new JLabel("JLabel avec texte",  
    new ImageIcon("08.png"),SwingConstants.CENTER);  
image = new JLabel(new ImageIcon("08.png"));  
texte.setBorder(new LineBorder(Color.black));  
textImage.setBorder(new LineBorder(Color.black));  
image.setBorder(new LineBorder(Color.black));  
frame.add(texte);  
frame.add(textImage);  
frame.add(image);
```



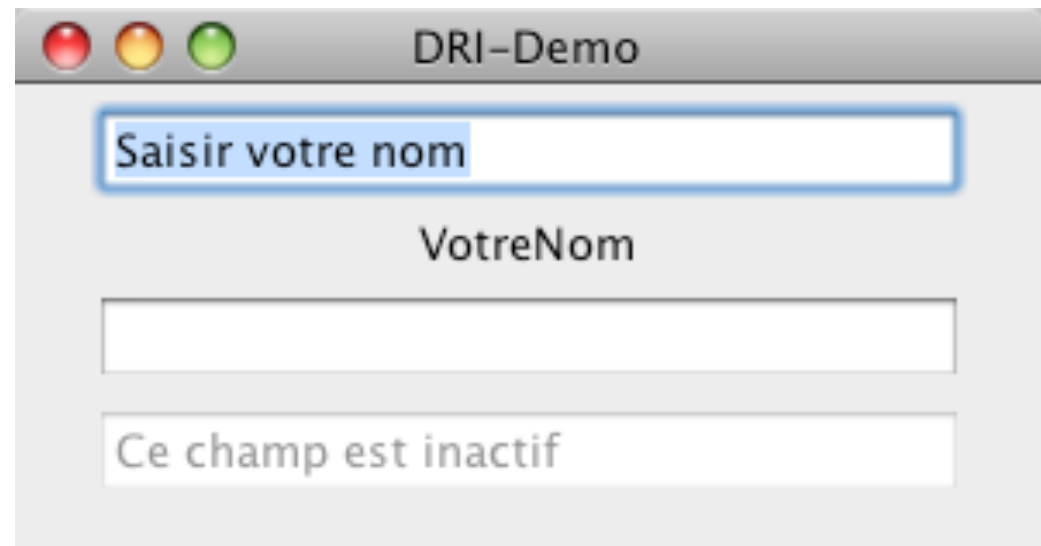
// JTextField

- Champ de saisie
- Taille fixée
- Peut être activé/désactivé
- Peut être accompagné d'un JLabel



// JTextField

```
JTextField nature, avecLabel, inactif;  
JLabel texte = new JLabel("VotreNom")  
nature = new JTextField("Saisir votre nom", 20);  
avecLabel = new JTextField(20);  
inactif = new JTextField("Ce champ est inactif",20);  
texte.setLabelFor(avecLabel);  
inactif.setEnabled(false);  
frame.add(nature);  
frame.add(texte);  
frame.add(avecLabel);  
frame.add(inactif);
```



// JButton

- Crée un bouton
- Permet de lancer des actions quand le bouton est cliqué
- Peut être activé/désactivé



// JButton

```
JButton actif = new JButton("Actif");  
JButton inactif = new JButton("Inactif");
```

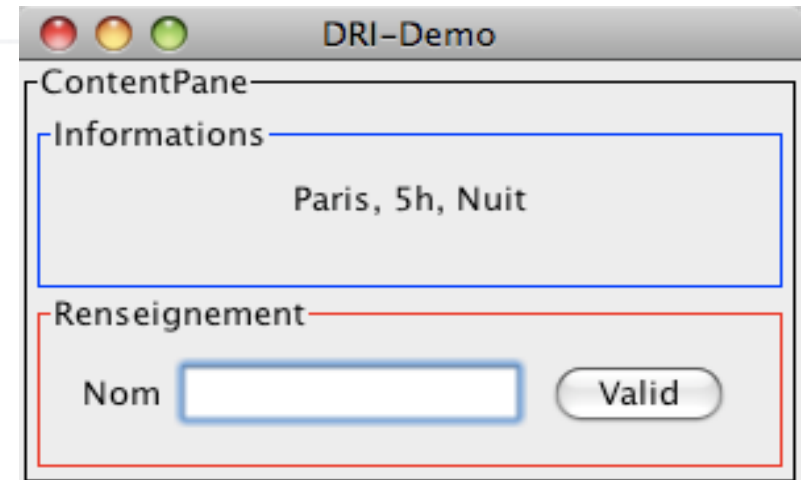
```
inactif.setEnabled(false);
```

```
frame.add(actif);  
frame.add(inactif);
```



// JPanel

- Conteneur de composants
- Est un composant
- Peut avoir une bordure
- Peut avoir un titre
- Disposition organisée des composants :
 - LayoutManagers



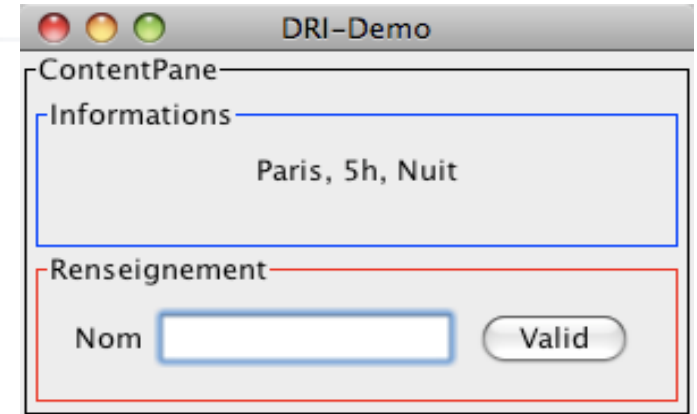
☞ Une JFrame a un Container par défaut :

```
JFrame mainFrame = new JFrame( »Ma Fenêtre" );  
Container myPanel = mainFrame.getContentPane();
```



// JPanel

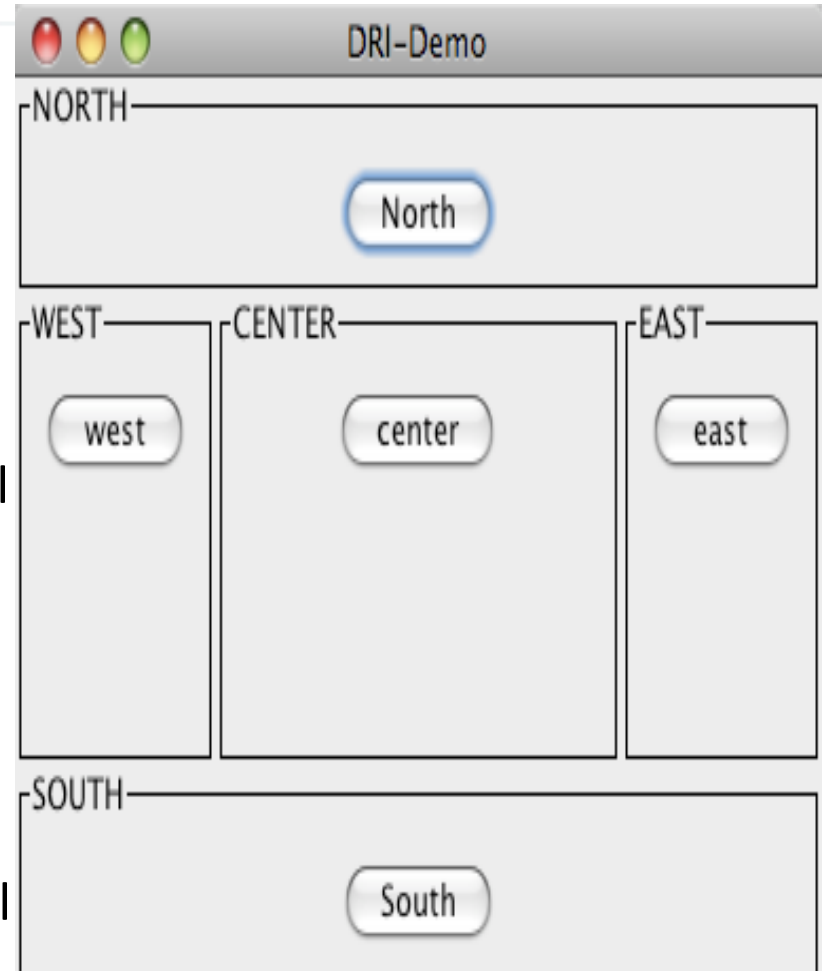
```
JPanel rens = new JPanel();
rens.setLayout(new FlowLayout());
rens.setBorder(
    BorderLayout.createTitledBorder(
        new LineBorder(Color.RED), "Renseignement"));
info.add(new JLabel("Paris, 5h, Nuit"));
rens.add(new JLabel("Nom"));
rens.add(new JTextField(10));
rens.add(new JButton("Valid"));
frame.setLayout(new GridLayout(2,1));
frame.add(rens);
```



// LayoutManagers

➔ BorderLayout

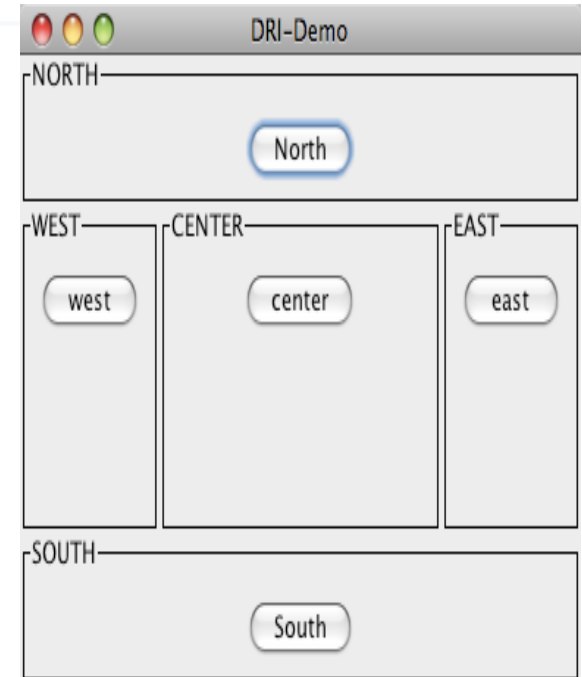
- Basé sur des zones définies
- NORTH
 - Haut de panneau, étirement horizontal
- SOUTH
 - Bas de panneau, étirement horizontal
- EAST
 - Droite du panneau, étirement vertical
- WEST
 - Gauche du panneau, étirement vertical
- CENTER
 - Centre du panneau, étirement vertical & horizontal



// LayoutManagers

➔ BorderLayout

```
frame.setLayout(new BorderLayout());
JPanel north;
north= new JPanel();
JButton northB;
northB= new JButton("North");
north.setBorder
    (BorderFactory.createTitledBorder(
        new LineBorder(Color.black),
        "NORTH"));
north.add(northB);
frame.add(north, BorderLayout.NORTH);
```



// LayoutManagers

➔ GridLayout

- Forme de grille
- Peut spécifier ou bloquer
 - NB lignes
 - NB colonnes
- Etirement des cases H et V
- Remplissage automatique



// LayoutManagers

➔ GridLayout

```
frame.setLayout(new GridLayout(3,3));
```

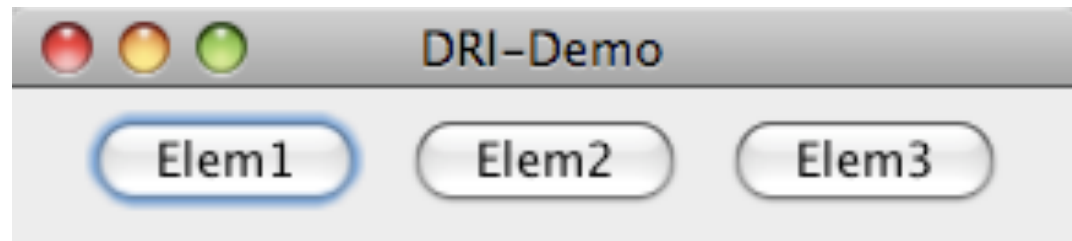
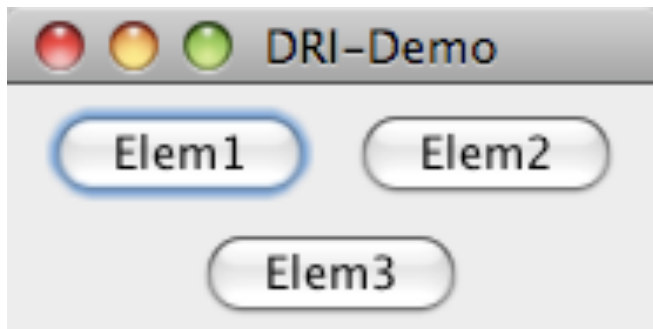
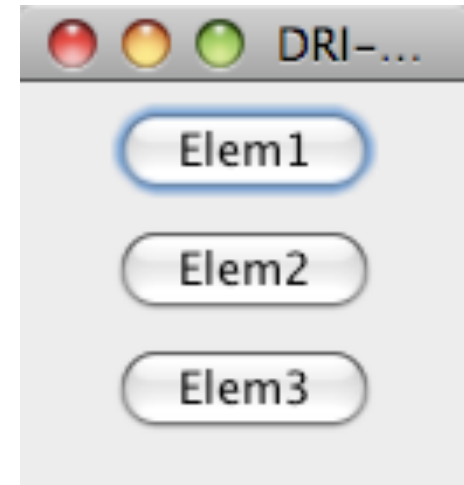
```
for(int i = 0 ; i < 3 ; i++) {  
    for( int j = 0 ; j < 3; j++) {  
        frame.add(new JButton("" + (i*3 + j + 1)));  
    }  
}
```



// LayoutManagers

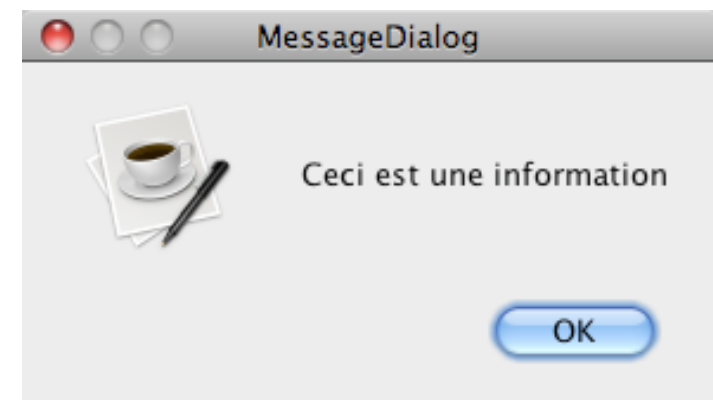
➔ FlowLayout

- Éléments placés les uns après les autres



// OptionPane

- Permet d'afficher facilement des messages (erreurs, warning, info,...)



// OptionPane

```
JOptionPane.showConfirmDialog(  
    frame, "Etes-vous sur ?",  
    "ConfirmDialog »,    JOptionPane.YES_NO_CANCEL_OPTION);  
JOptionPane.showMessageDialog(  
    frame, "Ceci est une grosse erreur",  
    "MessageDialog", JOptionPane.ERROR_MESSAGE);  
JOptionPane.showMessageDialog(  
    frame, "Ceci est un petit warning",  
    "MessageDialog", JOptionPane.WARNING_MESSAGE);  
JOptionPane.showMessageDialog(  
    frame, "Ceci est une information",  
    "MessageDialog", JOptionPane.INFORMATION_MESSAGE);
```



// Le modèle événementiel en Java

➔ Le modèle MVC

- Pour développer une interface graphique, il faut :
 - Définir le **MODELE** de l'application (classes métiers)
 - Définir l' « ergonomie » de l'interface graphique (la **VUE**)
 - Programmer la réaction aux actions de l'utilisateur sur les éléments de l'interface graphique (le **CONTROLEUR**)



// Le modèle événementiel en Java

➔ Les listeners (« observateur »)

- **Principe** : toute partie de l'application peut réagir aux événements produits pas une autre application.
- **Vocabulaire** : un événement est une information produite par un composant appelé émetteur et qui pourra déclencher des réactions sur d'autres éléments appelés récepteurs (ex: appui sur un bouton, etc.)
- **En java** :
 - La programmation événementielle n'existe pas !!
 - Elle est simulée par des « listeners » (patron de conception Observateur) :
 - Le récepteur doit définir un gestionnaire d'événements (Xlistener)
 - Le récepteur doit l'inscrire (addXListener) auprès d'un émetteur
 - Le composant avertit le gestionnaire d'événements quand un événement se produit (fireXListener)
 - Le récepteur peut désinscrire son gestionnaire d'événements
 - Un gestionnaire d'événements est spécifique à un type d'événement X.



// Le modèle événementiel en Java

➔ Exemple

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 class CoucouGUI implements ActionListener {
5     public CoucouGUI() {
6         JFrame fenetre = new JFrame(" Je débute !");
7         JButton coucou = new JButton(" Coucou");
8         fenetre.getContentPane().add(coucou);
9         coucou.addActionListener( this );
10        fenetre.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
11        fenetre.pack();           // Calculer de la taille de la fenêtre
12        fenetre.setVisible( true ); // Rendre le fenêtre visible
13    }
14    public void actionPerformed(ActionEvent coucou) {
15        System.out.println( " Bonjour !" );
16    }
17    public static void main(String [] args) {
18        new CoucouGUI();
19    }
20 }
```



// Le modèle événementiel en Java

➔ Exemple

```
1 class CoucouGUI implements ActionListener {
2     private JButton coucou = new JButton("Coucou");
3     private JButton quitter = new JButton("Quitter");
4     public CoucouGUI() {
5         JFrame fenetre = new JFrame("Réalise_ListenerAction");
6         fenetre.getContentPane().setLayout(new FlowLayout());
7         fenetre.getContentPane().add(coucou);
8         coucou.addActionListener(this);
9         fenetre.getContentPane().add(quitter);
10        quitter.addActionListener(this);
11        ...
12    }
13    public void actionPerformed(ActionEvent ev) {
14        if (ev.getSource() == coucou) {
15            System.out.println("Bonjour!");
16        } else if (ev.getSource() == quitter) {
17            System.exit(1);
18        }
19    }
20 }
```



// Le modèle événementiel en Java

➔ Exemple

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4 class CoucouGUI {
5     public CoucouGUI() {
6         JFrame fenetre = new JFrame(" Je débute!");
7         JButton coucou = new JButton(" Coucou");
8         fenetre.getContentPane().add(coucou);
9         coucou.addActionListener(new ActionCoucou());
10        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        fenetre.pack();           // Calculer de la taille de la fenêtre
12        fenetre.setVisible(true); // Rendre le fenêtre visible
13    }
14    public static void main(String[] args) {
15        new CoucouGUI();
16    }
17 }
18 class ActionCoucou implements ActionListener {
19     public void actionPerformed(ActionEvent coucou) {
20         System.out.println(" Bonjour!");
21     }
22 }
```

