

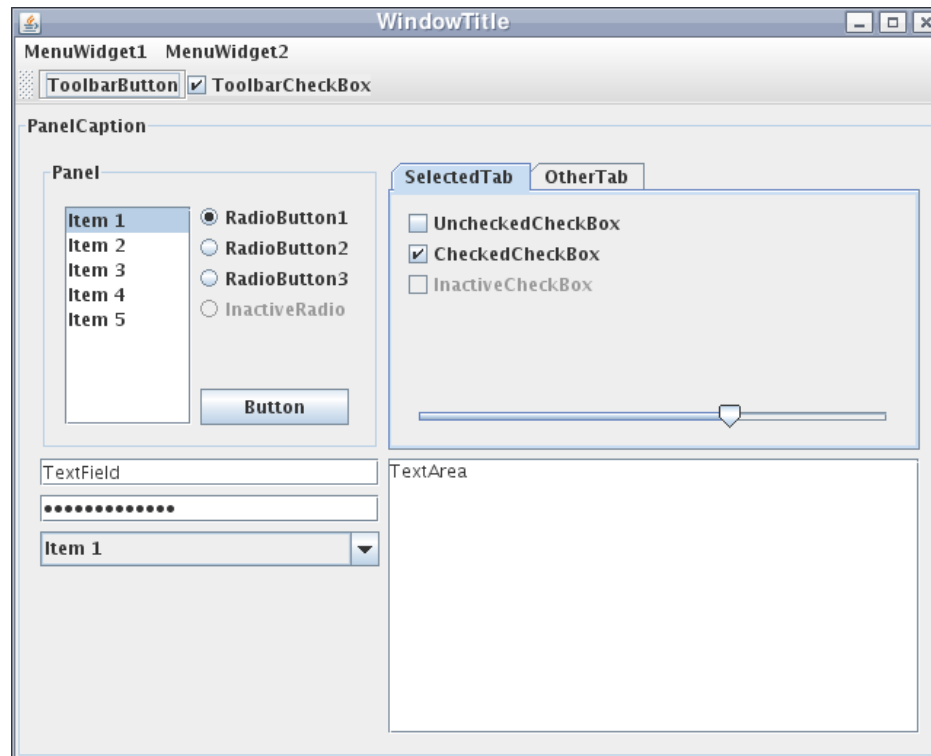
Chapitre 16

Interface graphique

Introduction

Nous avons jusque là crée des applications en mode texte, fonctionnant dans un terminal.

Nous allons maintenant créer des applications présentant une interface graphique (**GUI** : *Graphical User Interface*), dans une ou plusieurs fenêtres.



Plusieurs bibliothèques graphiques sont utilisables avec Java :

- **AWT** (*Abstract Window Toolkit*), historiquement la plus ancienne.
- **SWING**, basée sur AWT.
- **JavaFX**
- **SWT** (*Standard Widget Toolkit*)

Nous allons dans ce chapitre voir **Swing**, qui a pour avantage d'être présente dans le JDK sans avoir à rajouter de packages externes.

Comparatif : <http://koor.fr/Java/TutorialSwing/comparatif.wp>

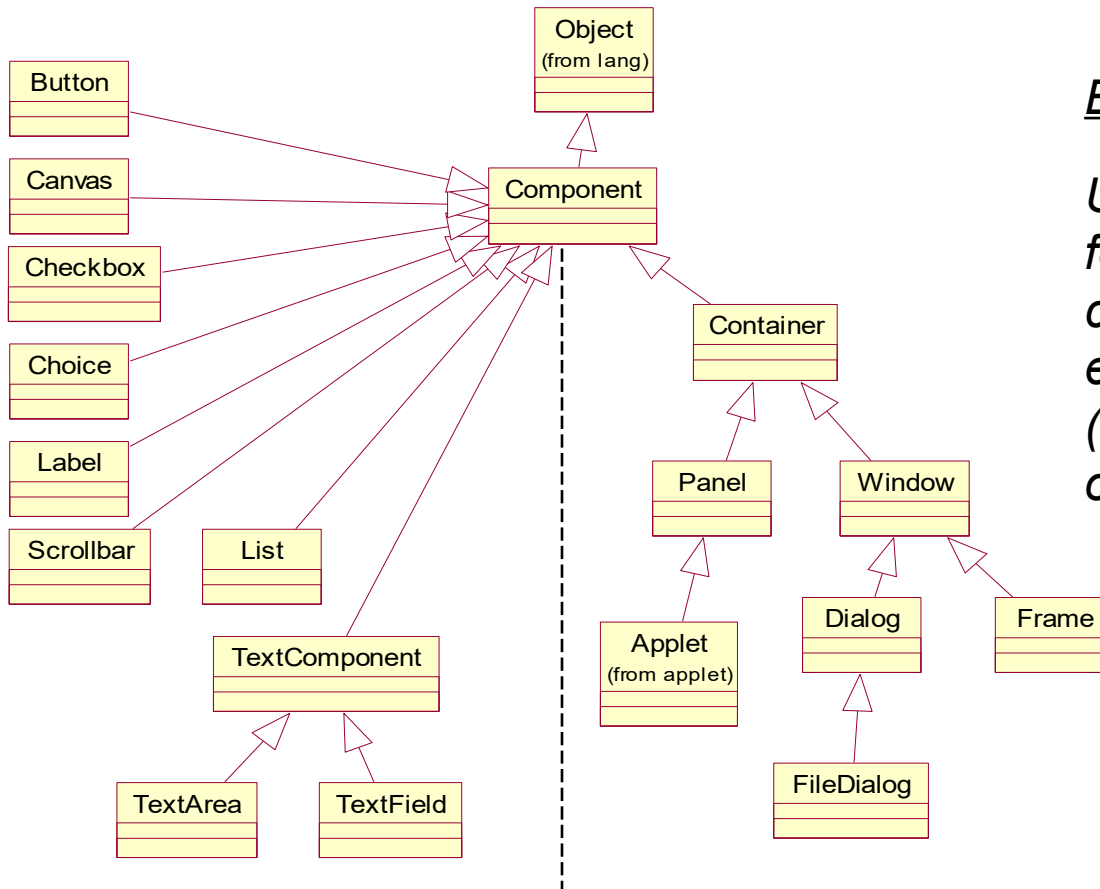
Lire aussi : <https://www.infoq.com/fr/news/2018/03/JavaFXRemovedFromJDK/>

AWT

<https://docs.oracle.com/en/java/javase/13/docs/api/java.desktop/java/awt/package-summary.html>

AWT est un ensemble de paquetages définissant des centaines de classes, avec une très forte utilisation de l'héritage :

java.awt, java.awt.event, java.awt.font, java.awt.image, ...

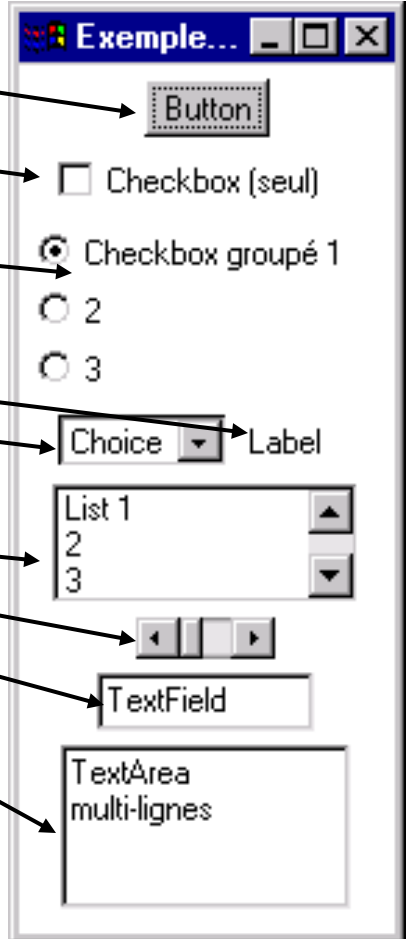


Ex:

Un cadre (Frame) est une fenêtre (Window), qui est un conteneur (Container), qui est un composant (Component), qui est un objet (Object).

Les composants graphiques

- Button
- Canvas (zone de dessin)
- Checkbox (case à cocher)
- CheckboxGroup
- Label
- Choice (Sélecteur)
- List
- Scrollbar (barre de défilement)
- TextField (zone de saisie d'1 ligne)
- TextArea (zone de saisie multilignes)
- ...



The image shows a window titled "Exemple..." containing several GUI components. Arrows from the list on the left point to the following components in the window:

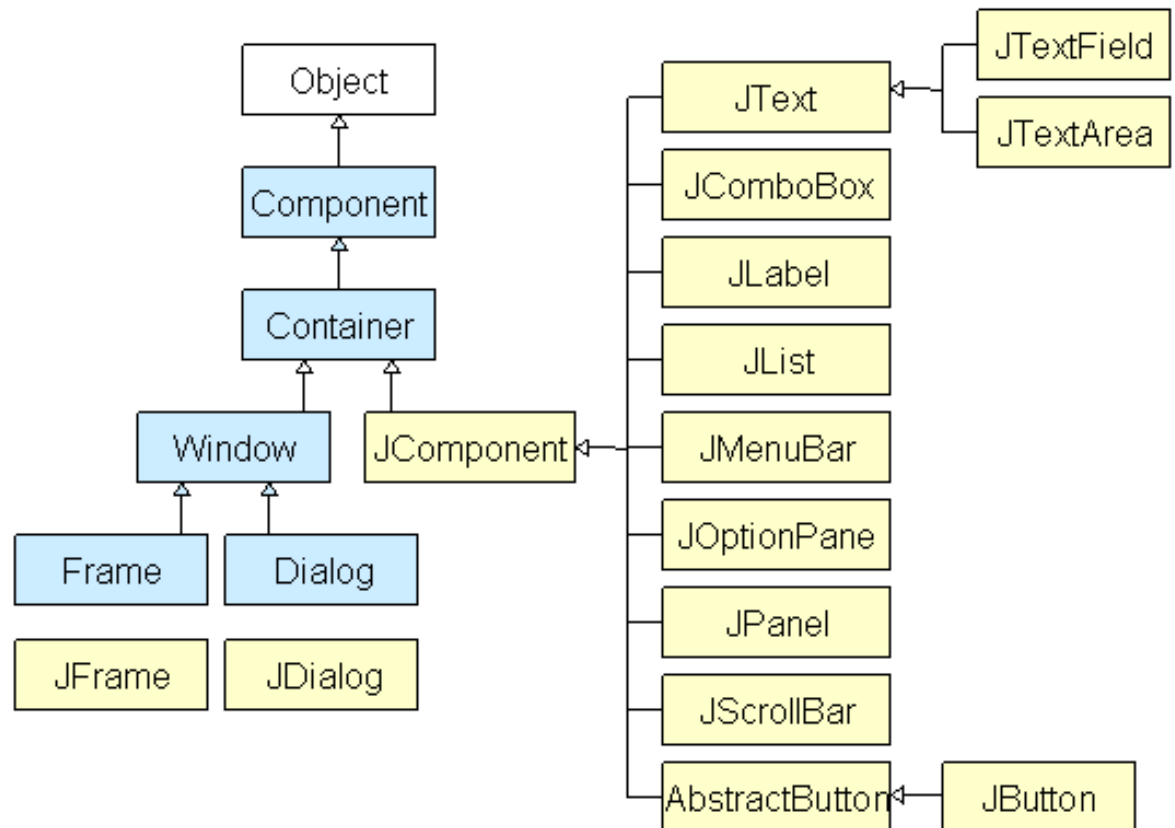
- Button**: A rectangular button with the text "Button".
- Checkbox**: A small square checkbox followed by the text "Checkbox (seul)".
- CheckboxGroup**: A group of three radio buttons, with the first one selected, followed by the text "Checkbox groupé 1".
- Choice**: A dropdown menu with a small arrow on the right, followed by the text "Label".
- List**: A list box containing the text "List 1", "2", and "3".
- Scrollbar**: A vertical scrollbar with up and down arrows.
- TextField**: A single-line text input field.
- TextArea**: A multi-line text input area labeled "multi-lignes".

Swing

<https://docs.oracle.com/en/java/javase/13/docs/api/java.desktop/javax/swing/package-summary.html>

Swing est une bibliothèque de composants graphiques pour Java situés dans le paquetage javax.swing. Elle est constituée de centaines de classes utilisant fortement l'héritage, héritant des classes de AWT.

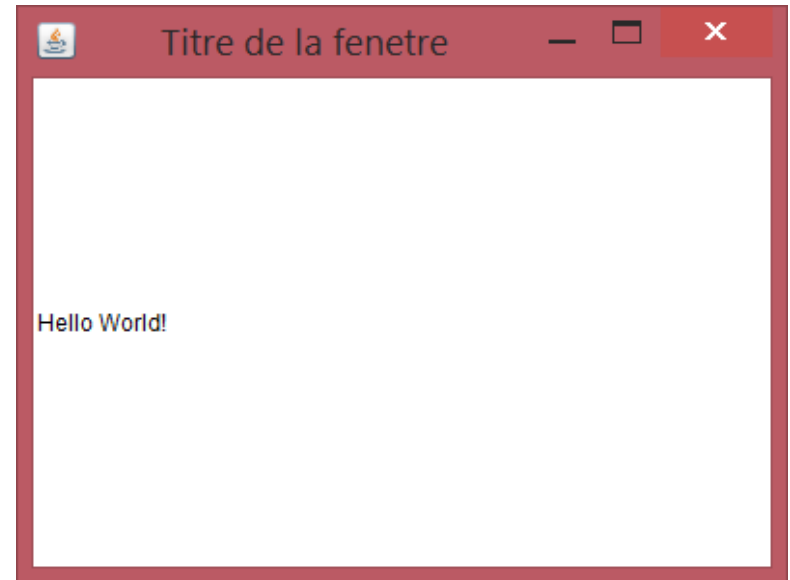
Un extrait des classes de Swing (en jaune), qui héritent des classes de AWT (en bleu)



Exemple de programme utilisant Swing

```
import javax.swing.*;

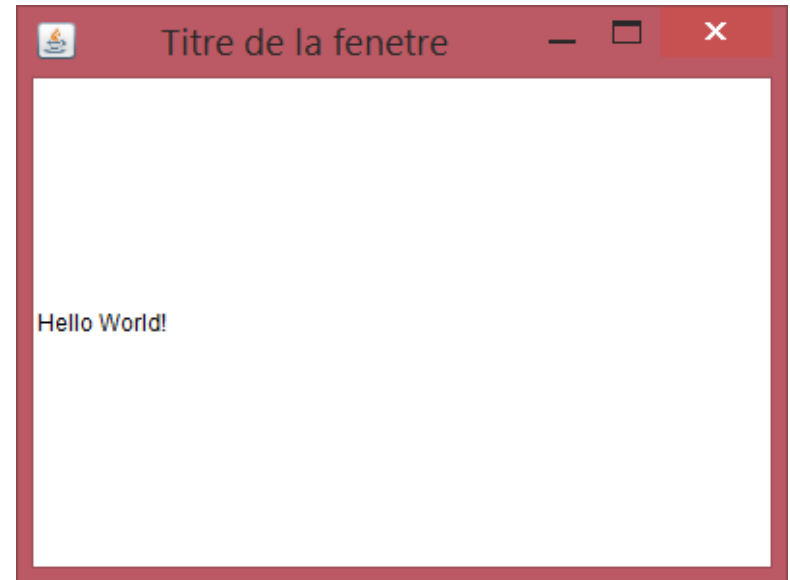
public class HelloWorld
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame("Titre de la fenetre");
        fenetre.setSize(400,300) ;
        fenetre.getContentPane().add((new JLabel("Hello World!")));
        fenetre.setVisible(true);
    }
}
```



Exemple de programme utilisant Swing

```
import javax.swing.*;

public class HelloWorld
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame("Titre de la fenetre");
        fenetre.setSize(400,300) ;
        Container c = fenetre.getContentPane() ;
        JLabel label = new JLabel("Hello World!")
        c.add(label);
        fenetre.setVisible(true);
    }
}
```



Aperçu des composants Swing

Basic Controls

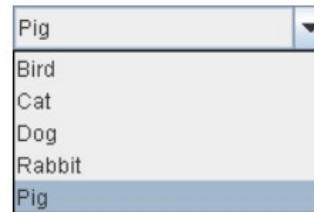
Simple components that are used primarily to get input from the user;
they may also show simple state.



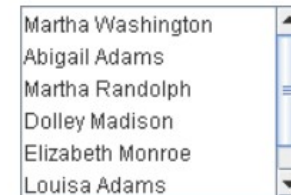
[JButton](#)



[JCheckBox](#)



[JComboBox](#)



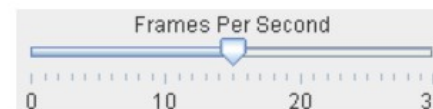
[JList](#)



[JMenu](#)



[JRadioButton](#)



[JSlider](#)



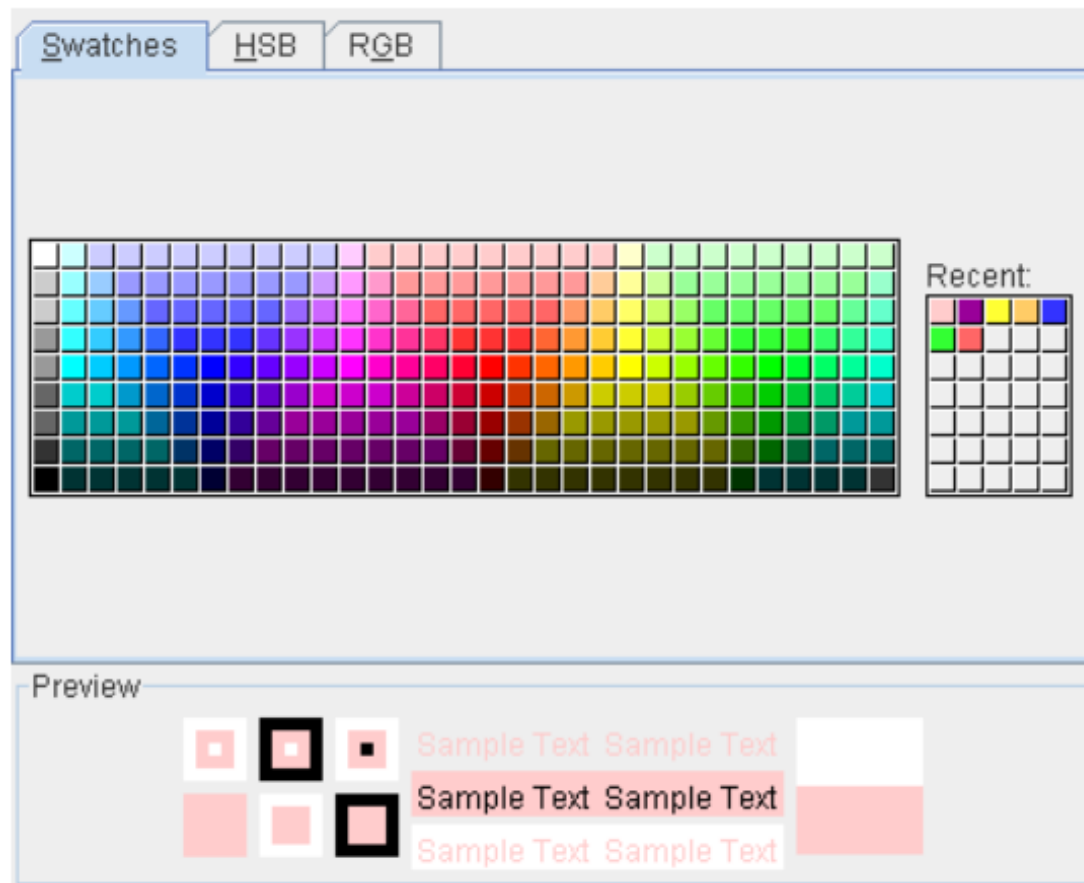
[JSpinner](#)



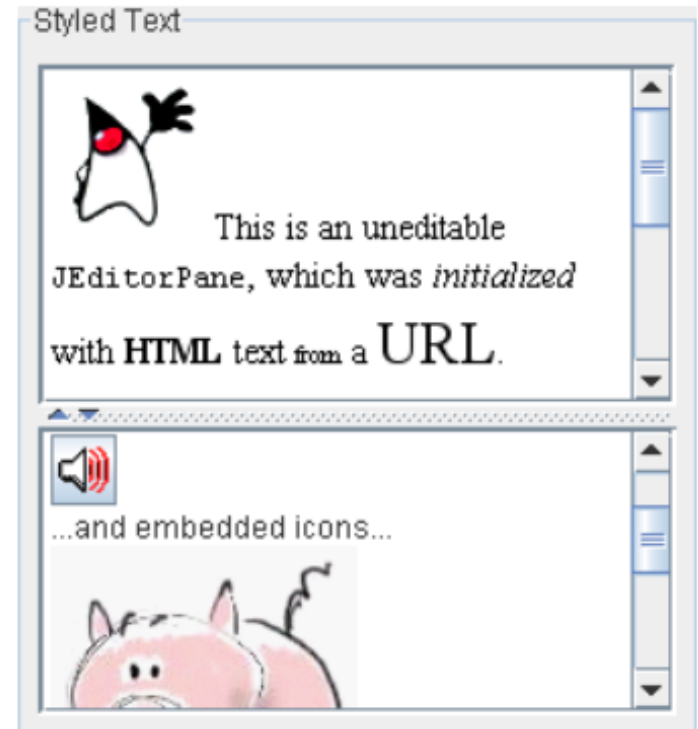
[JTextField](#)



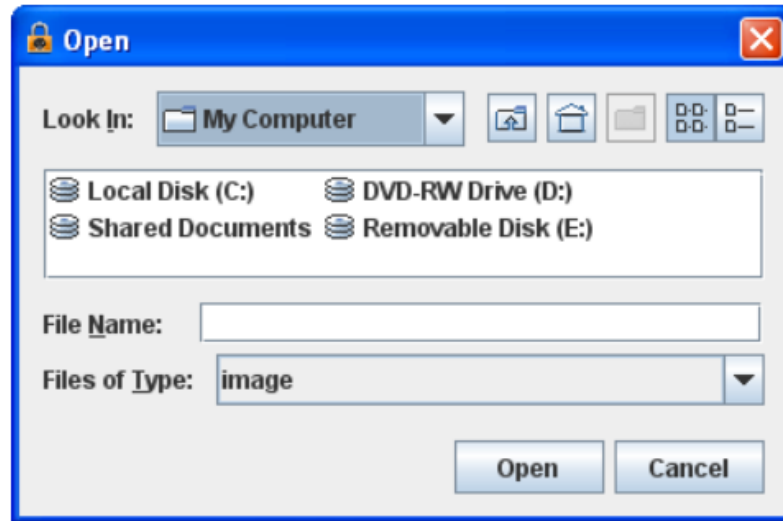
[JPasswordField](#)



[JColorChooser](#)



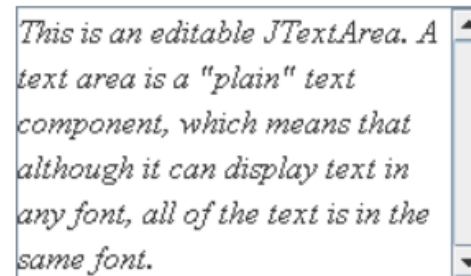
[JEditorPane](#) and [JTextPane](#)



[JFileChooser](#)

Host	User	Password	Last Modified
Biocca Games	Freddy	l#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34\$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	vbAf124%z	Feb 22, 2006

[JTable](#)



[JTextArea](#)



[JTree](#)

Uneditable Information Displays

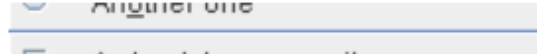
These components exist solely to give the user information.



[JLabel](#)



[JProgressBar](#)



[JSeparator](#)



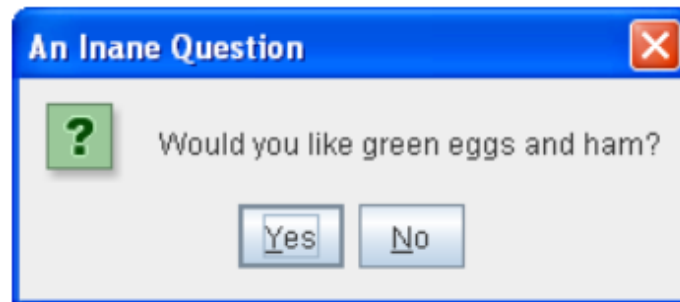
[JToolTip](#)

Top-Level Containers

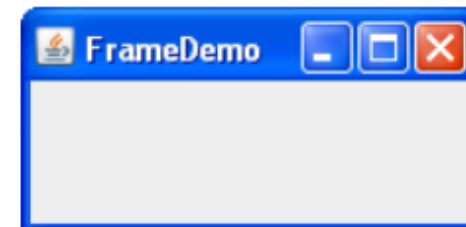
At least one of these components must be present in any Swing application.



[JApplet](#)



[JDialog](#)



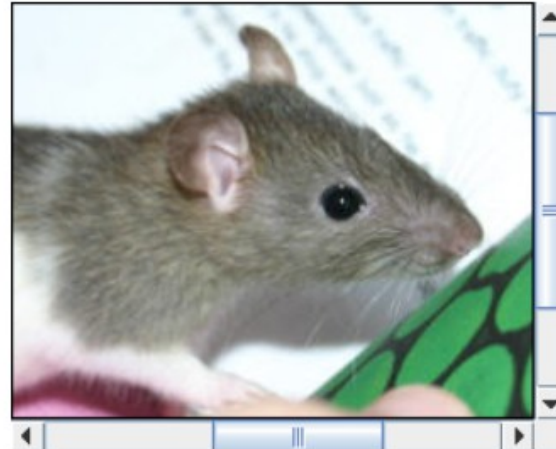
[JFrame](#)

General-Purpose Containers

These general-purpose containers are used in most Swing applications.



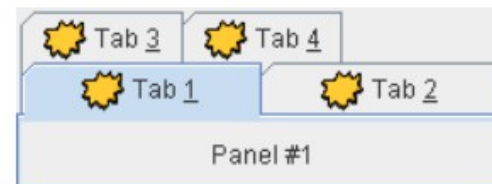
[JPanel](#)



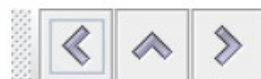
[JScrollPane](#)



[JSplitPane](#)



[JTabbedPane](#)



[JToolBar](#)

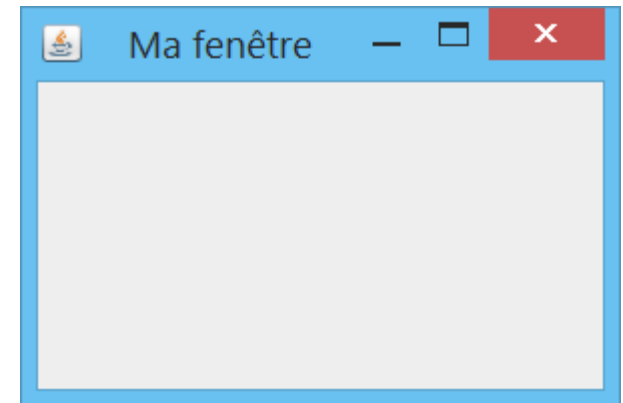
1. Créer une fenêtre avec la classe JFrame

1.1 Création de la fenêtre

Une fenêtre est un objet instance de la classe **JFrame**.

```
import javax.swing.*;

public class maFenetre
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame("Ma fenêtre");
        fenetre.setSize(300, 200);
        fenetre.setVisible(true);    // La fenêtre n'est pas affichée sinon
    }
}
```



1.2 Fermeture d'une fenêtre

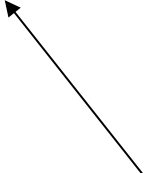
Un clic sur la case de fermeture de la fenêtre ne ferme pas le programme mais rend par défaut la fenêtre invisible.

Ce comportement par défaut peut être modifié avec la méthode **setDefaultCloseOperation()**.

4 comportements sont possibles lors de la fermeture de la fenêtre :

- **DO_NOTHING_ON_CLOSE**
- **HIDE_ON_CLOSE**
- **DISPOSE_ON_CLOSE**
- **EXIT_ON_CLOSE**

```
import javax.swing.*;
public class maFenetre
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame("Ma fenetre");
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setSize(300, 200);
        fenetre.setVisible(true);
    }
}
```



L'application se terminera lorsqu'on fermera la fenêtre

1.3 Modifier le titre d'une fenêtre

On peut donner le titre d'une fenêtre dans son constructeur :

```
JFrame fenetre = new JFrame("Ma fenetre");
```

A noter que le constructeur peut aussi ne pas recevoir de paramètre, la fenêtre n'aura alors pas de titre :

```
JFrame fenetre = new JFrame() ;
```

On peut modifier le titre à tout moment avec la méthode **setTitle()** de la classe JFrame :

```
fenetre.setTitle("Ma fenetre");
```

1.4 Positionner une fenêtre à l'écran

On peut changer la position d'une fenêtre à l'écran avec la méthode **setLocation(int x, int y)** :

```
JFrame fenetre = new JFrame("Ma fenetre");  
fenetre.setLocation(100, 200);
```

Pour centrer la fenêtre au milieu de l'écran :

```
fenetre.setLocationRelativeTo(null) ;
```

1.5 Empêcher le redimensionnement d'une fenêtre

Par défaut, on peut redimensionner une fenêtre à la souris. On peut interdire cette modification de la taille avec la méthode `setResizable()` de la classe `JFrame` qui prend comme valeur `true` ou `false` (interdiction).

```
JFrame fenetre = new JFrame("Ma fenetre");  
fenetre.setResizable(false);
```

1.6 Garder une fenêtre au premier plan

Pour faire en sorte qu'une fenêtre reste en permanence par dessus les autres, on utilise la méthode `setAlwaysOnTop()` de la classe `JFrame` qui prend comme valeur `true` (au premier plan) ou `false` (comportement normal).

```
JFrame fenetre = new JFrame("Ma fenêtre");  
fenetre.setAlwaysOnTop(true);
```

1.7 Enlever les contours d'une fenêtre

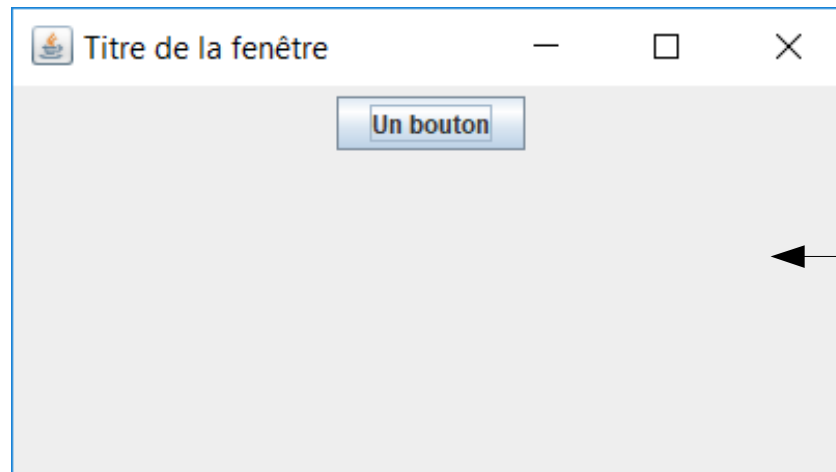
On peut enlever les bordures d'une fenêtre, sa barre de titre et ses boutons d'iconification/aggrandissement/fermeture en utilisant la méthode **setUndecorated()** de la classe JFrame qui prend comme valeur true (pas de contours) ou false (comportement normal).

```
JFrame fenetre = new JFrame("Ma fenêtre");  
fenetre.setUndecorated(true);
```

1.8 Accès au contenu d'une fenêtre

Le conteneur d'un **JFrame** possède une couche de contenu (« *Content pane* »), dans laquelle on peut ajouter les composants graphiques et dont on peut changer le gestionnaire de présentation (« *Flow layout* », voir section 4).

Cette couche est obtenue par la méthode **getContentPane()** de **JFrame**.



← *Content pane*

Ex :

```
JFrame fenetre = new JFrame ("Ma fenêtre");  
Container panneauContenu = fenetre.getContentPane();  
JButton bouton = new JButton ("Un bouton");  
panneauContenu.add(bouton);
```

2. Couleurs

La classe `Color` est utilisée pour spécifier des couleurs. Il s'agit d'une classe de AWT, qui s'importe comme suit :

```
import java.awt.Color;
```

Constantes

La classe `Color` contient des constantes prédéfinies :

```
Color.black, Color.red, Color.blue, Color.gray,  
Color.green, Color.yellow, Color.white, ...
```

Elle permet aussi de créer de nouvelles couleurs.

Définition de nouvelles couleurs

On utilise le codage RVB (appelé RGB en anglais pour Red, Green, Blue), qui consiste à créer n'importe quelle couleur par combinaison de quantités de rouge, de vert et de bleu.

couleur = (rouge,vert,bleu)

Chaque quantité est comprise entre 0 et 255, ce qui permet d'obtenir par combinaison 16 millions de couleurs différentes.

Exemple :

```
Color couleur_rouge;  
couleur_rouge = new Color(255,0,0);
```

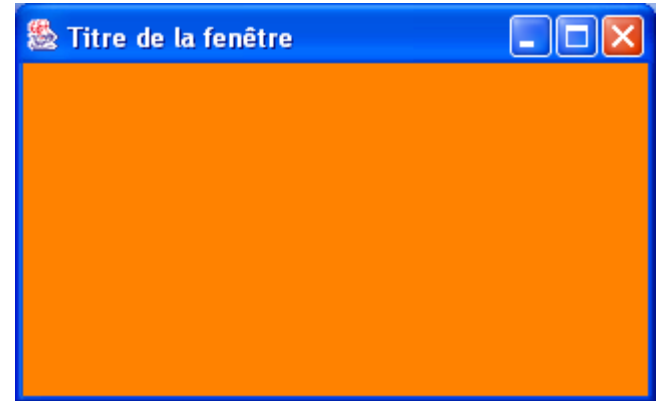

Rouge	Vert	Bleu	Couleur
0	0	0	noir
255	0	0	rouge
0	255	0	vert
0	0	255	bleu
255	255	0	jaune
255	0	255	magenta
0	255	255	cyan
160	160	160	gris
255	255	255	blanc

*Quelques exemples de couleurs obtenues par
combinaison de rouge, de vert et de bleu*

Exemple : Création d'une fenêtre avec un fond orange.

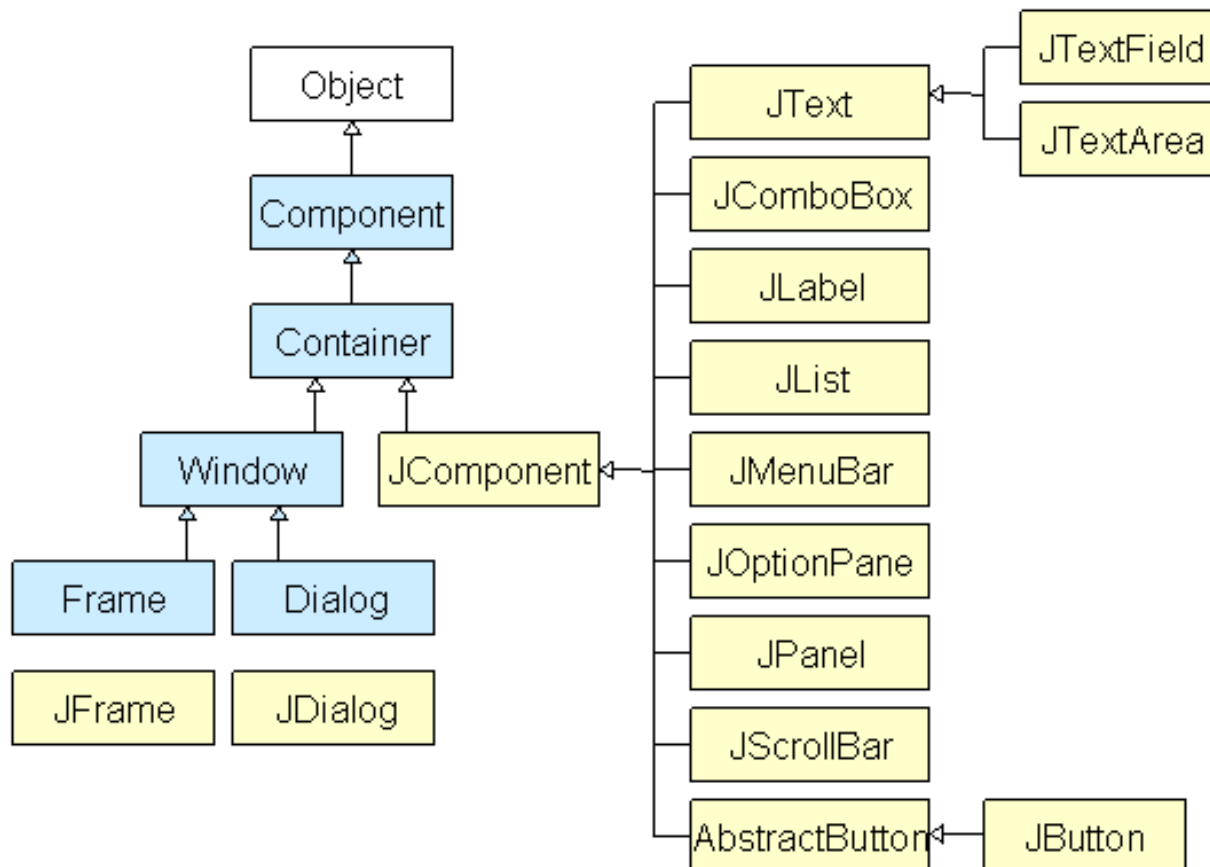
```
import java.awt.Color;
import javax.swing.*;

public class maFenetre
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame("Titre de la fenêtre");
        Color couleur = new Color(255,130,0);
        fenetre.getContentPane().setBackground(couleur);
        fenetre.setSize(300, 200);
        fenetre.setVisible(true);
    }
}
```



3. Les composants

Toutes les classes de composants d'interface graphique (JButton, JText, JLabel, ...) héritent de la classe Swing **JComponent** qui hérite elle-même de la classe AWT **Component**.



On trouve dans cette classe de base **Component** de nombreuses méthodes très utiles dont profitent toutes les classes dérivées :

<https://docs.oracle.com/en/java/javase/13/docs/api/java.desktop/java/awt/Component.html>

setVisible(boolean visible)

Affiche ou masque le composant.

setEnabled(boolean actif)

Active ou non le composant, c'est à dire le rend sensible aux événements.

Dimension getSize()

Donne la dimension actuelle du composant. Retourne le type **Dimension** qui est utilisable ainsi : **getSize().height** et **getSize().width**

setSize(int largeur, int hauteur)

Redimensionne le composant.

setLocation(int coordX, int coordY)

Déplace le composant au point indiqué (coin haut et gauche).

setForeground(Color couleur)

Définit la couleur d'avant-plan (de dessin) du composant.

setBackground(Color couleur)

Définit la couleur de fond.

3.1 Label (JLabel)

Permet d'afficher un texte non éditable :

```
JLabel textLabel = new JLabel("Texte du label");  
getContentPane().add(textLabel);
```

Permet d'afficher une image :

```
JLabel imgLabel = new JLabel(new ImageIcon("image.jpg"));  
getContentPane().add(imgLabel);
```

3.2 Bouton (JButton)

Un bouton est un composant possédant un texte, sur lequel on peut cliquer pour déclencher une action.

```
// Début du fichier TestBoutonFenetre.java
```

```
class TestBoutonFenetre
{
    public static void main(String[] args)
    {
        BoutonFenetre fenetre;
        fenetre = new BoutonFenetre ("Test bouton");
    }
}
```

```
// Fin du fichier TestBoutonFenetre.java
```

```
// Début du fichier BoutonFenetre.java
```

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

```
class BoutonFenetre extends JFrame  
{
```

```
    JButton bouton;
```

```
    BoutonFenetre(String s)
```

```
{
```

```
        super(s);
```

```
        setSize(320,200);
```

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

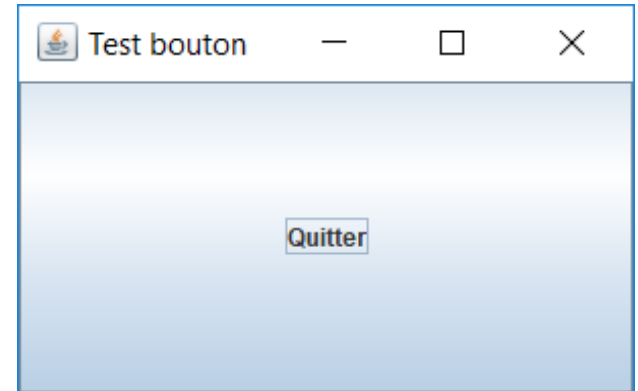
```
        getContentPane().add(bouton);
```

```
        setVisible(true);
```

```
}
```

```
}
```

```
// Fin du fichier BoutonFenetre.java
```



Notes

bouton (de classe **JButton**) est un composant de **fenetre** (de classe **BoutonFenetre**, qui hérite de **JFrame**).

Le constructeur de **BoutonFenetre** reçoit en paramètre une chaîne qu'il transmet au constructeur de **JFrame** grâce à **super**.

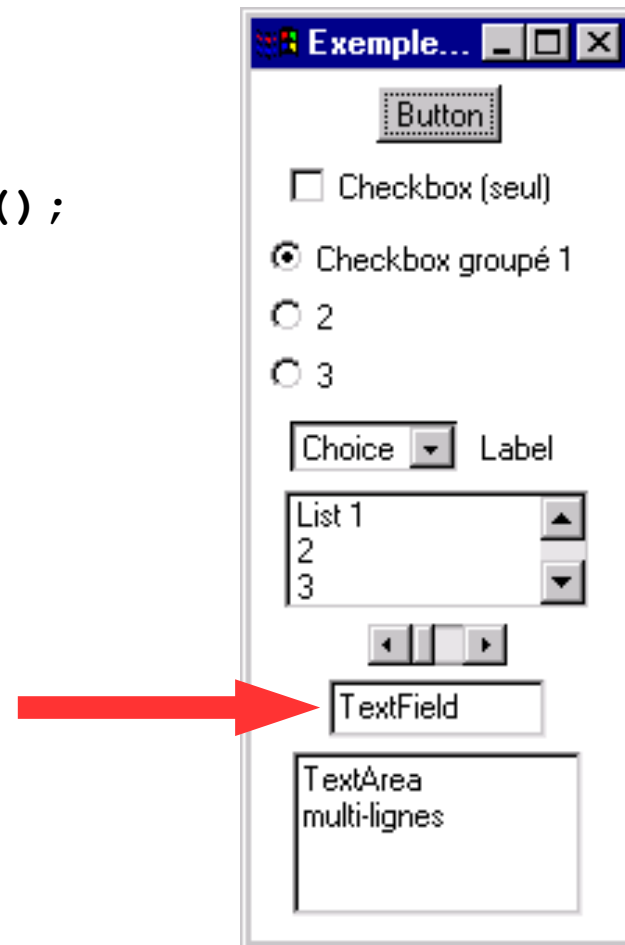
L'étiquette (le texte) d'un bouton se définit en passant une chaîne à son constructeur ou en utilisant la méthode **setLabel()** de la classe **JButton**.

La méthode **add()** de la classe **JFrame** permet de faire de l'objet **bouton** un composant de l'objet **BoutonFenetre**.

3.3 Champ texte (JTextField)

C'est un composant définissant un champ texte pour saisir et/ou afficher du texte modifiable.

```
JTextField tf = new JTextField();  
getContentPane().add(tf);
```



Constructeurs

TextField()

Crée un champ sans texte.

TextField(int nombre)

Crée un champ sans texte d'une largeur de nombre caractères.

TextField(String texte)

Crée un champ comportant un texte par défaut.

TextField(String texte, int nombre)

crée un champ comportant un texte et d'une largeur de nombre caractères.

Quelques méthodes

setText(String texte)

Spécifie le texte du champ.

String getText()

Retourne le texte contenu dans le champ.

setEditable(boolean modifiable)

Spécifie si le texte du champ est modifiable.

boolean isEditable()

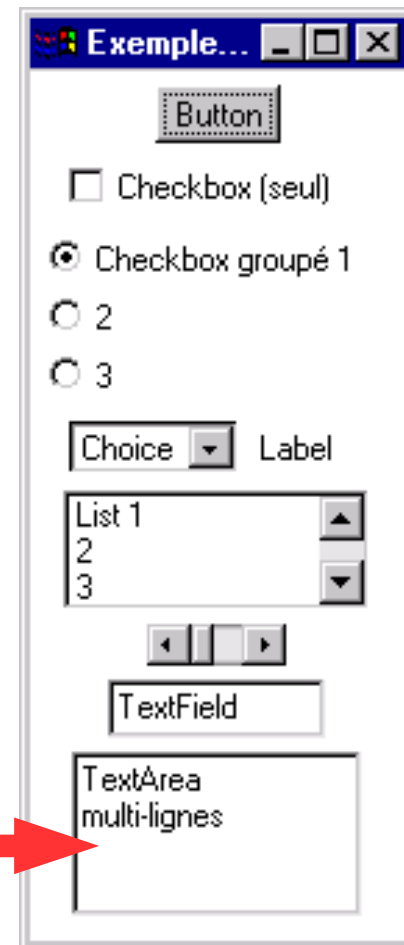
Retourne un booléen qui indique si le texte du champ est modifiable.

3.4 Zone de texte multiligne (JTextArea)

C'est un composant définissant une zone de texte multi-lignes pour saisir et/ou afficher du texte modifiable.

```
JTextArea ta = new JTextArea();  
getContentPane().add(ta);
```

```
ta.setText("Un nouveau texte");  
ta.append("ajoute du texte");  
ta.append("à la suite");
```



Par défaut, un **JTextArea** n'a pas d'ascenseurs. Pour qu'il en ait, il faut le placer dans un **JScrollPane** :

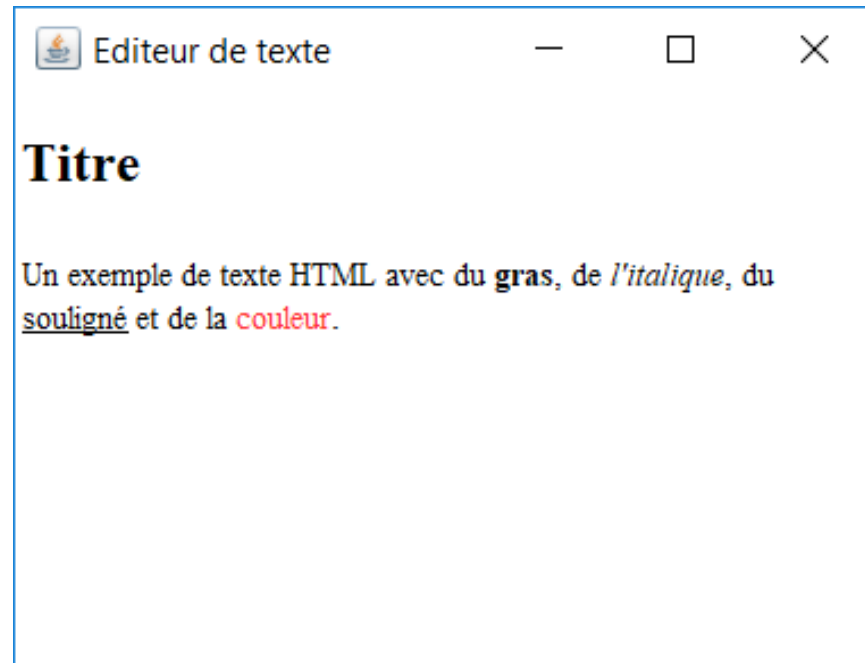
```
JTextArea textArea = new JTextArea();  
JScrollPane scrollText = new JScrollPane(textArea) ;  
getContentPane().add(scrollText);
```

JEditorPane

<https://docs.oracle.com/javase/7/docs/api/javax/swing/JEditorPane.html>

Composant permettant d'afficher et d'éditer du code aux formats :

- Texte brut
- RTF (Rich Text File), texte enrichi avec des tailles de caractère variables, couleurs, gras, italique, souligné, etc.
- HTML

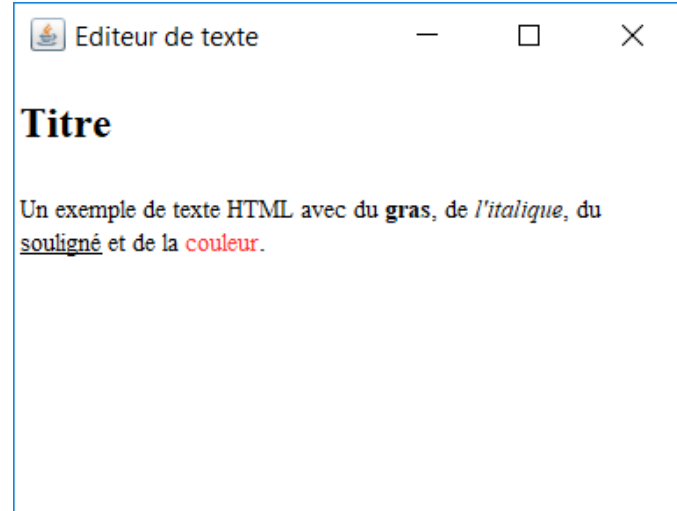


Exemple :

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

public class EditeurDeTexte
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame ("Editeur de texte");
        fenetre.setSize(400,300);
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JEditorPane editeur = new JEditorPane();
        editeur.setContentType("text/html");
        editeur.setText("<h1>Titre</h1><p>Un exemple de texte HTML
                        avec du <b>gras</b>, de <i>l'italique</i>,
                        du <u>souligné</u> et de la
                        <font color='#ff3333'>couleur</font>.</p>");
        Container panneauContenu = fenetre.getContentPane();
        panneauContenu.add(editeur);
        fenetre.setVisible(true);
    }
}
```



JTextPane

un éditeur de texte qui permet la gestion de texte formaté, le retour à la ligne automatique (word wrap), l'affichage d'images.

JPasswordField

un champ de saisie de mots de passe : la saisie est invisible et l'affichage de chaque caractère tapé est remplacé par un caractère ' * '.

setToolTipText ()

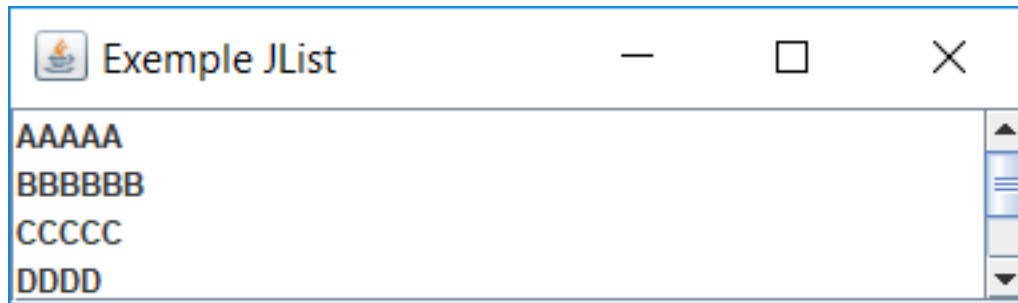
permet de créer des bulles d'aide qui apparaissent lorsque la souris passe sur un composant (bouton, textfield, ...)

```
JButton bouton = new JButton("Un bouton");  
bouton.setToolTipText("Aide de mon bouton");
```

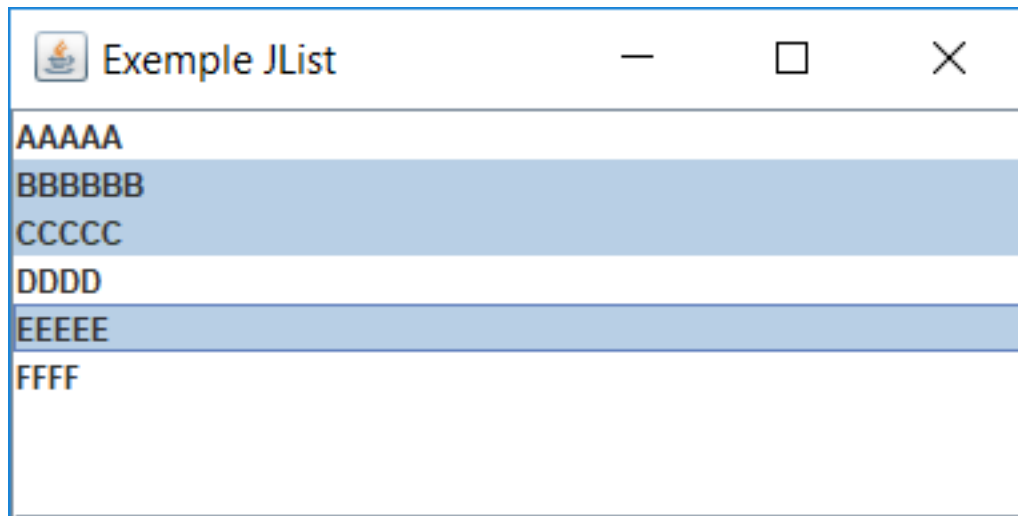
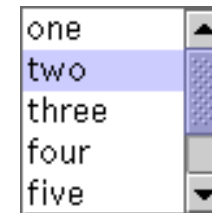
JList

Classe permettant de représenter des listes, permettant la sélection simple ou multiple.

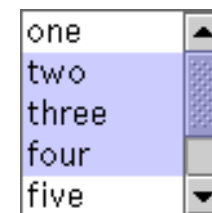
<https://docs.oracle.com/javase/tutorial/uiswing/components/list.html>



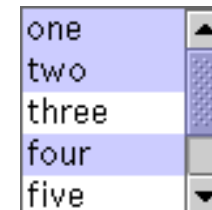
SINGLE_SELECTION

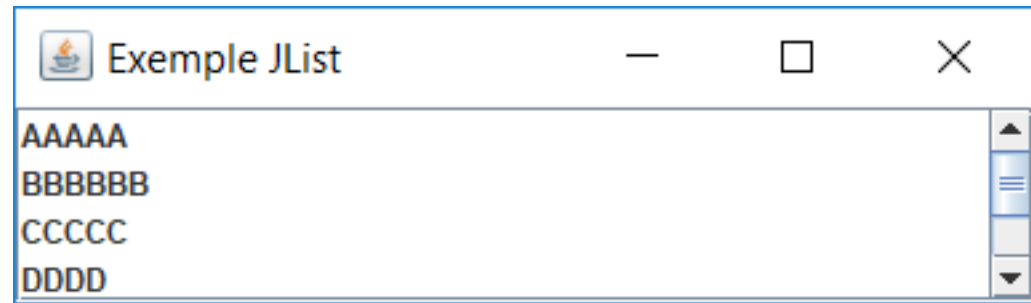


SINGLE_INTERVAL_SELECTION



MULTIPLE_INTERVAL_SELECTION





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

```
public class Fenetre extends JFrame {

    public Fenetre() {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("Exemple JList");
        this.setSize(400, 120);

        String[] donnees = {"AAAAA", "BBBBBB", "CCCCC",
                           "DDDD", "EEEE", "FFFF" };
        JList<String> list = new JList<String>(donnees);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        this.getContentPane().add(new JScrollPane(list));
    }

    public static void main(String[] args) {
        Fenetre fen = new Fenetre();
        fen.setVisible(true);
    }
}
```

Connaître les éléments sélectionnés d'une JList

Méthodes de la classe **JList** :

int[] getSelectedIndices()

Retourne un tableau des indices des éléments sélectionnés.

int getSelectedIndex()

Retourne l'indice du premier élément sélectionné (ou -1 si aucun).

List<E> getSelectedValuesList()

Retourne une liste des éléments sélectionnés.

On peut aussi utiliser un « *listener* » (écouteur) :

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

public class Fenetre extends JFrame implements ListSelectionListener
{
    String[] donnees = {"AAAAA", "BBBBBB", "CCCCC", "DDDD", "EEEEEE"};
    JList<String> list = new JList<String>(donnees);

    public Fenetre() {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("Exemple JList");
        this.setSize(400, 120);

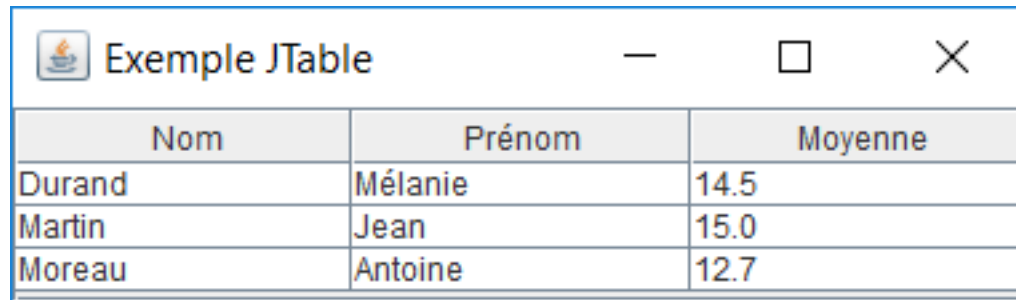
        list.addListSelectionListener(this);
        this.getContentPane().add(new JScrollPane(list));
    }

    public void valueChanged(ListSelectionEvent e)
    {
        System.out.println(list.getSelectedValuesList());
    }
}
```

JTable

Classe permettant de représenter des tableaux 2D (façon Excel).

<https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>



Nom	Prénom	Moyenne
Durand	Mélanie	14.5
Martin	Jean	15.0
Moreau	Antoine	12.7

Exemple JTable		
Nom	Prénom	Moyenne
Durand	Mélanie	14.5
Martin	Jean	15.0
Moreau	Antoine	12.7

```
import javax.swing.*;

public class Fenetre extends JFrame {

    public Fenetre(){
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("Exemple JTable");
        this.setSize(400, 120);

        //Les données du tableau
        Object[][] donnees = {
            {"Durand", "Mélanie", "14.5"},
            {"Martin", "Jean", "15.0"},
            {"Moreau", "Antoine", "12.7"}
        };

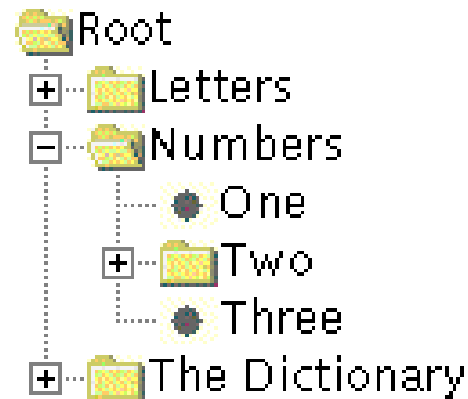
        //Les titres des colonnes
        String titres[] = {"Nom", "Prénom", "Moyenne"};
        JTable tableau = new JTable(donnees, titres);
        this.getContentPane().add(new JScrollPane(tableau));
    }

    public static void main(String[] args){
        Fenetre fen = new Fenetre();
        fen.setVisible(true);
    }
}
```

JTree

Classe permettant de représenter des données hiérarchiques.

<https://docs.oracle.com/javase/tutorial/uiswing/components/tree.html>



JLayeredPane

Conteneur qui permet de ranger ses composants en couches.

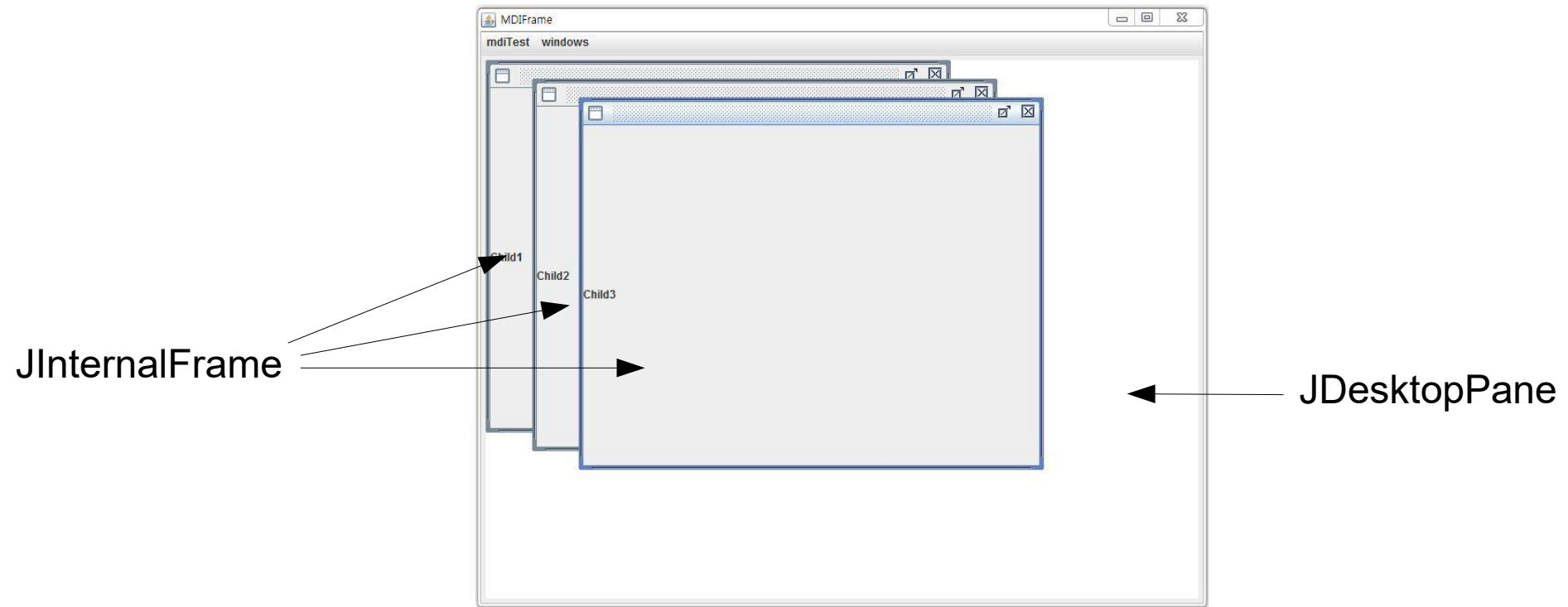
Cela permet de dessiner les composants, selon un certain ordre: premier plan, plan médian, arrière plan, etc.

Pour ajouter un composant, il faut spécifier la couche sur laquelle il doit être dessiné :

`monJlayeredPane.add (monComposant, new Integer(5));`

Fenêtres MDI (Multiple Document Interface)

Une fenêtre parent (**JDesktopPane**) peut contenir plusieurs fenêtres filles (**JInternalFrame**), permettant par exemple de travailler sur plusieurs documents simultanément.



Création de la fenêtre fille :

```
String title = "Une fenêtre fille";  
Boolean resizable = true;  
Boolean closable = true;  
Boolean maximizable = true;  
Boolean iconifiable = true;  
JInternalFrame iFrame = new JInternalFrame(title, resizable, closable,  
                                              maximizable, iconifiable) ;
```

```
// On ajoute des composants (boutons, champs éditables, ...) à la fenêtre :  
iFrame.add(...)  
iFrame.setSize(200,200);           // ou : iFrame.pack() pour que la fenêtre s'ajuste  
iFrame.setVisible(true);
```

Création de la fenêtre parent :

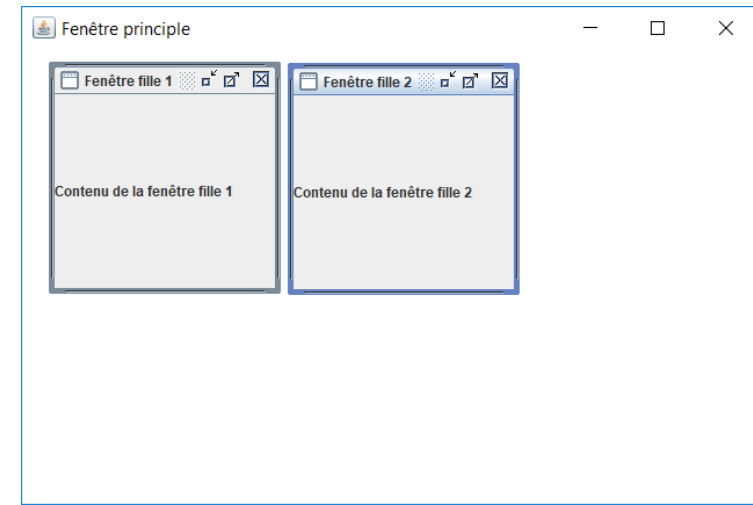
```
JDesktopPane desktopPane = new JDesktopPane();  
// On ajoute la fenêtre fille à la fenêtre parent :  
desktopPane.add(iFrame);
```

```
import javax.swing.*;

public class TestMDI
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame("Fenêtre principale");
        JDesktopPane desktopPane = new JDesktopPane();
        JInternalFrame frame1 = new JInternalFrame("Fenêtre fille 1", true, true, true, true);
        frame1.getContentPane().add(new JLabel("Contenu de la fenêtre fille 1"));
        frame1.setSize(200,200);
        frame1.setVisible(true);
        frame1.setLocation(20, 20);
        desktopPane.add(frame1);

        JInternalFrame frame2 = new JInternalFrame("Fenêtre fille 2", true, true, true, true);
        frame2.getContentPane().add(new JLabel("Contenu de la fenêtre fille 2"));
        frame2.setSize(200,200);
        frame2.setVisible(true);
        frame2.setLocation(240, 20);
        desktopPane.add(frame2);

        fenetre.getContentPane().add(desktopPane);
        fenetre.setSize(600,400);
        fenetre.setVisible(true);
    }
}
```



4. Gestion de la mise en page

Mise en page (« *layout* ») = disposition de plusieurs composants dans un conteneur au moyen de classes de mise en page : **FlowLayout**, **GridLayout**, **BorderLayout**, **CardLayout**, **GridBagLayout**.

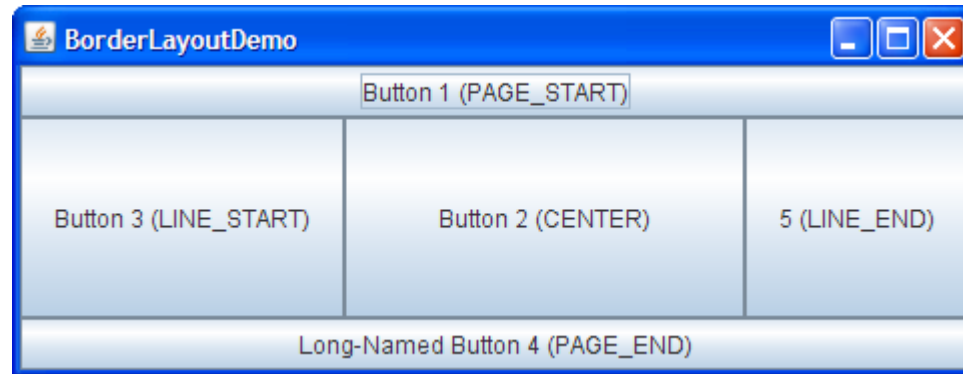
Ces classes héritent de la classe **LayoutManager**.

On définit la mise en page d'un conteneur en passant un de ces objets **LayoutManager** à la méthode **setLayout()** du conteneur.

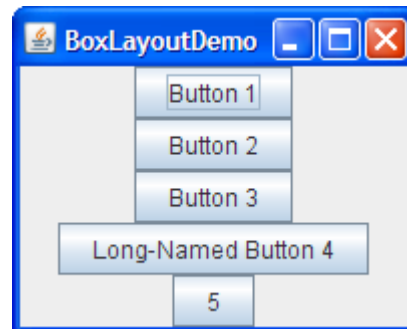
<http://docs.oracle.com/javase/tutorial/uiswing/layout/using.html>

<http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

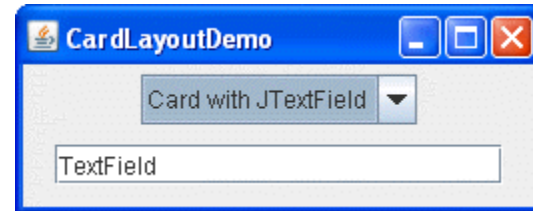
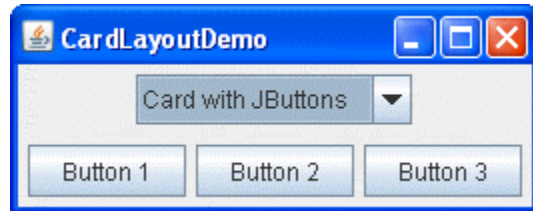
BorderLayout



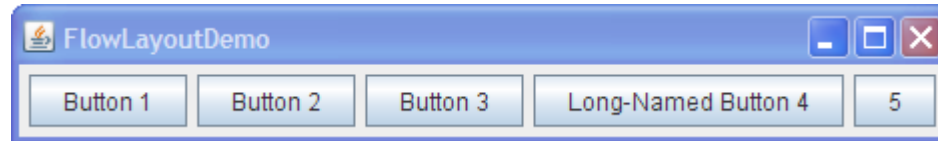
BoxLayout



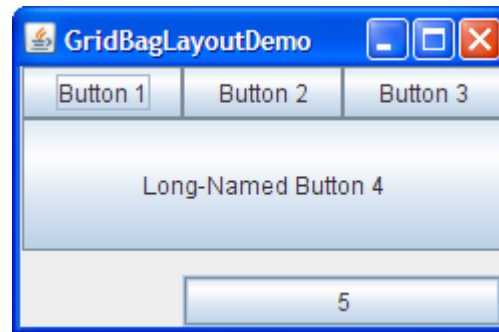
CardLayout



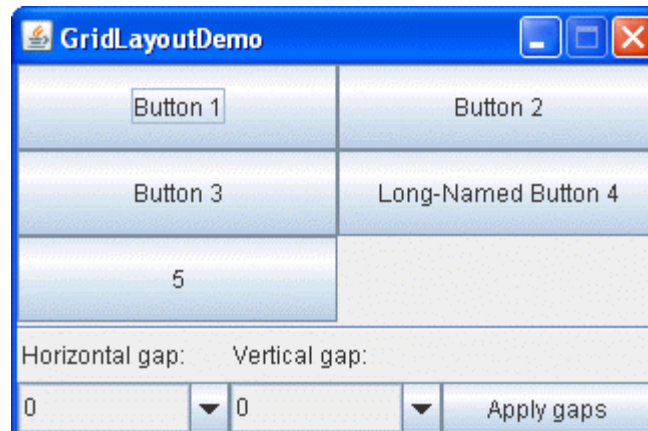
FlowLayout (par défaut)



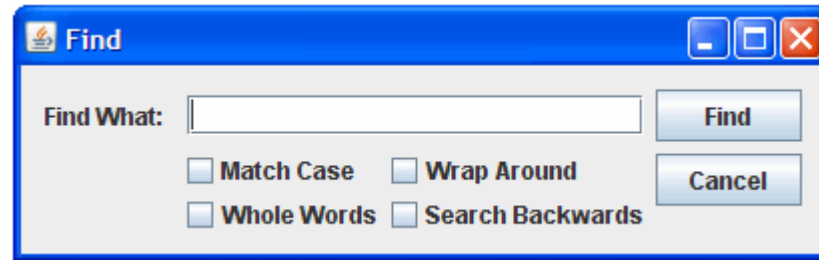
GridBagLayout



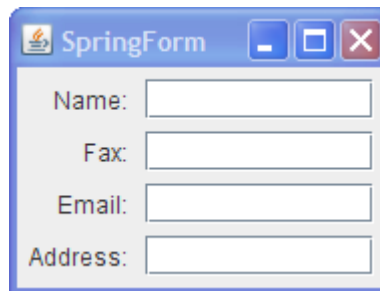
GridLayout



GroupLayout



SpringLayout

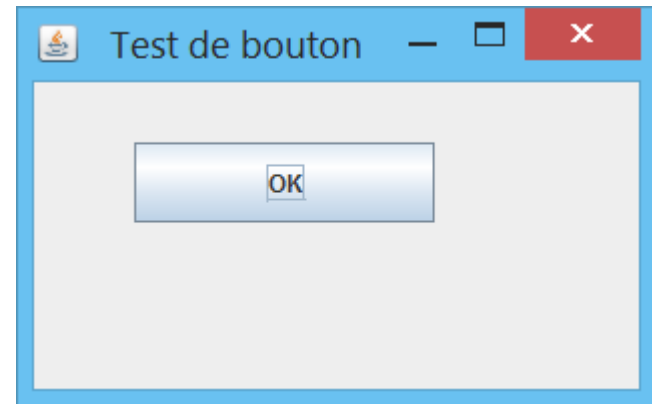


4.1 Disposer des composants sans layout

On peut aussi positionner des composants et définir leur taille librement, sans utiliser de layout, grâce à la méthode **setBounds(int x, int y, int largeur, int hauteur)** des composants et en passant **null** à la méthode **setLayout** de la fenêtre :

```
class BoutonFenetre extends JFrame
{
    JButton bouton;

    BoutonFenetre(String s)
    {
        super(s);
        setSize(320,200);
        setLayout(null);
        bouton = new JButton("OK");
        bouton.setBounds(50,30,150,40);
        add(bouton);
        setVisible(true);
    }
}
```



4.2 Gestionnaire FlowLayout

Ce gestionnaire dispose les composants de gauche à droite et de haut en bas.

La taille des boutons est définie par défaut en fonction de la taille de son étiquette.



```
// Fichier BoutonsFenetre.java
import javax.swing.*;

class BoutonsFenetre extends JFrame
{
    JButton [] boutons;

    BoutonsFenetre(String s)
    {
        super(s);
        setSize(320,200);
        FlowLayout layout = new FlowLayout();
        setLayout(layout);
        boutons = new JButton[16];

        for(int i=0; i<16; i++ )
        {
            boutons[i] = new JButton("Bouton "+i);
            getContentPane().add(boutons[i]);
        }

        setVisible(true);
    }
}
```

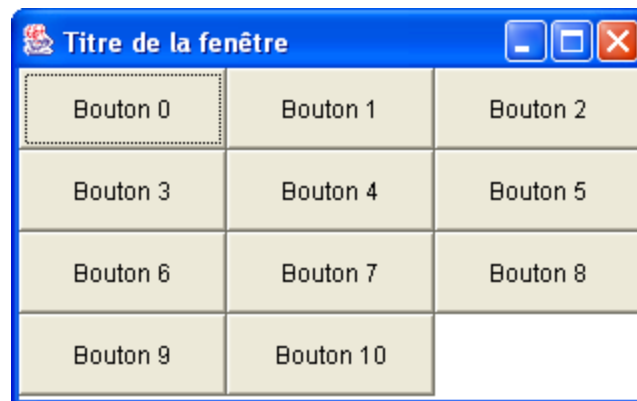
```
// Fichier TestBoutonsFenetre.java
```

```
class TestBoutonsFenetre
{
    public static void main(String[] args)
    {
        BoutonsFenetre fenetre;
        fenetre = new BoutonsFenetre ("Test de FlowLayout");
    }
}
```

4.3 Gestionnaire GridLayout

Ce gestionnaire dispose les composants selon une grille régulière de gauche à droite et de haut en bas. On indique au constructeur de **GridLayout** le nombre de lignes et de colonnes.

La taille des boutons est fonction de la taille du conteneur et du nombre de ses composants.



```
// Fichier BoutonsFenetre.java
import javax.swing.*;

class BoutonsFenetre extends JFrame
{
    JButton [] boutons;

    BoutonsFenetre(String s)
    {
        super(s);
        setSize(320,200);
        GridLayout layout = new GridLayout(4,3);
        setLayout(layout);
        boutons = new JButton[11];

        for(int i=0; i<11; i++ )
        {
            boutons[i] = new JButton("Bouton "+i);
            add(boutons[i]);
        }
        setVisible(true);
    }
}
```

```
// Fichier TestBoutonsFenetre.java
```

```
class TestBoutonsFenetre
{
    public static void main(String[] args)
    {
        BoutonsFenetre fenetre;
        fenetre = new BoutonsFenetre ("Test de GridLayout");
    }
}
```


IDE (Integrated Development Environment)

= Environnement de développement intégré.

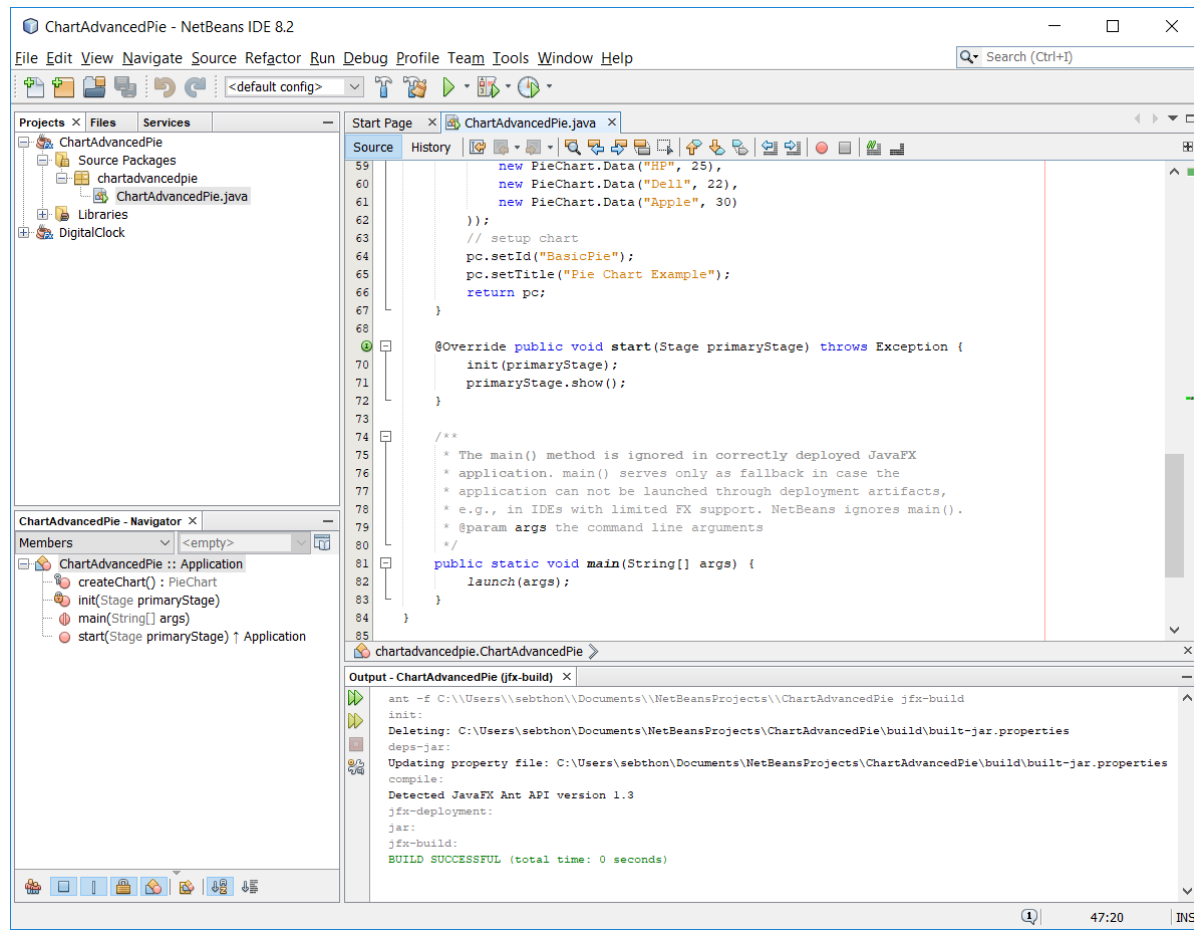
Permet de gérer des projets comportant plusieurs fichiers, d'éditer du code, de compiler, de déboguer, de construire visuellement une interface graphique, etc.

Netbeans

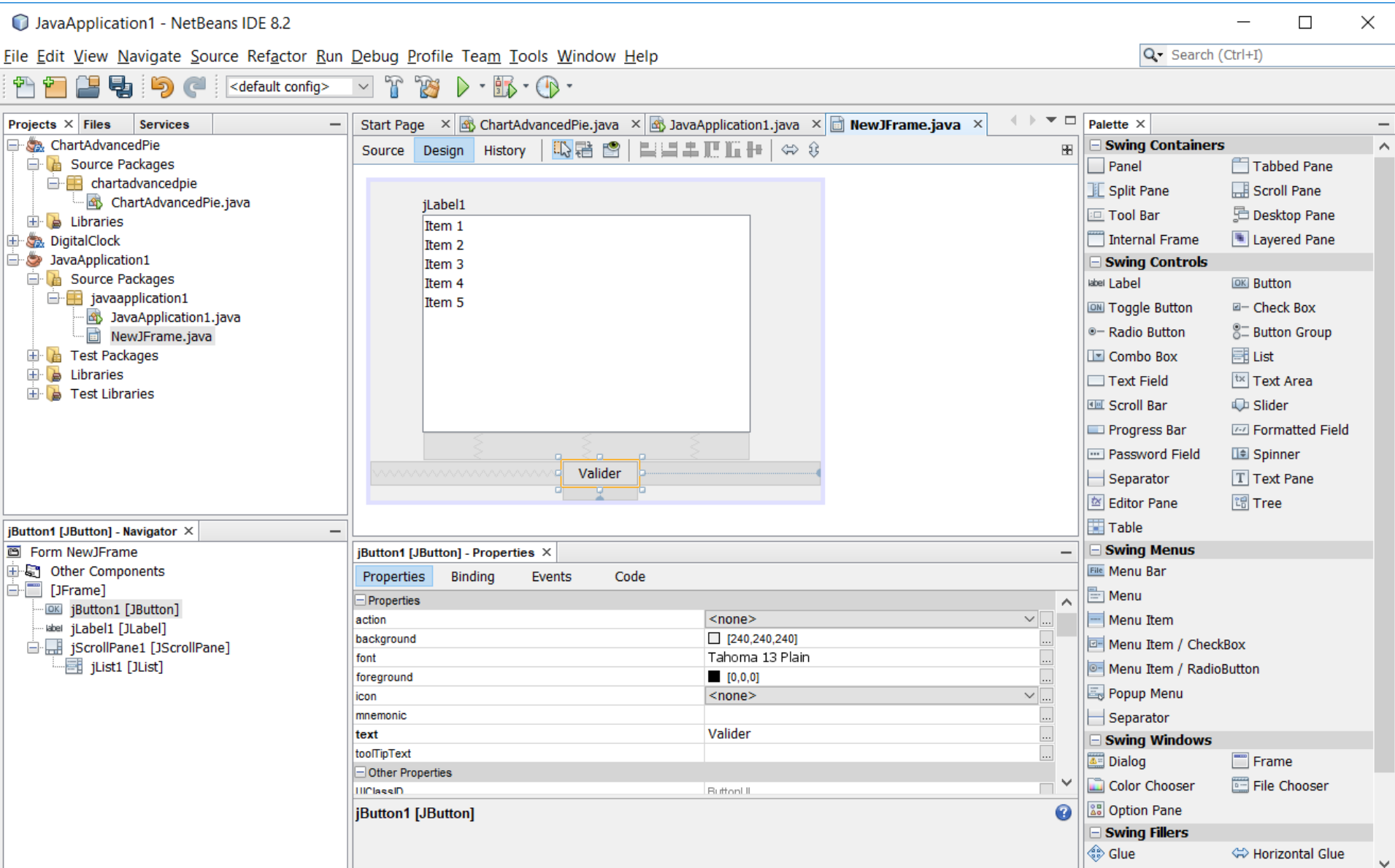
<https://netbeans.org/>

Eclipse

<https://www.eclipse.org/ide/>



Un IDE permet de construire visuellement une interface graphique.
<http://netbeans.apache.org/kb/docs/java/gui-functionality.html>



5. Boîte de message : JOptionPane

<http://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html>

<https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>

La classe **JOptionPane** permet de faire apparaître une boîte de message permettant :

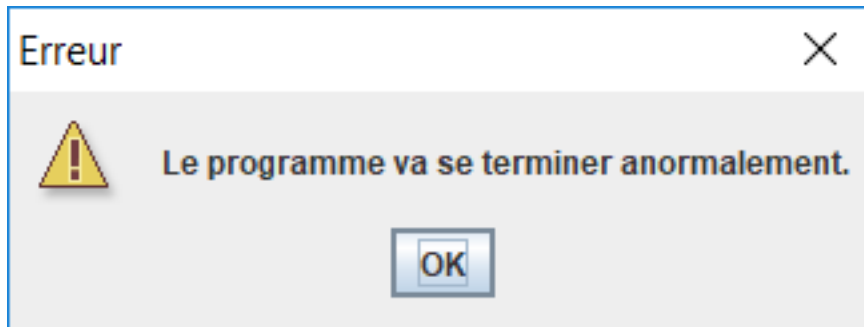
- d'informer l'utilisateur
- de lui demander de faire un choix
- de saisir une valeur
- de faire apparaître une boîte personnalisée

1) JOptionPane.showMessageDialog pour informer l'utilisateur :

```
JOptionPane.showMessageDialog(null,  
    "Le programme va se terminer anormalement.",  
    "Erreur",  
    JOptionPane.WARNING_MESSAGE);  
}
```

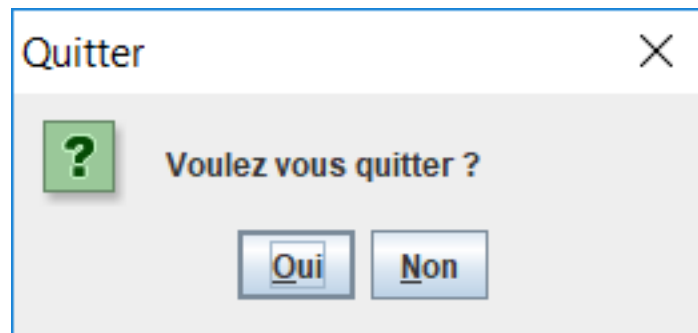
Icône :

JOptionPane.PLAIN_MESSAGE
JOptionPane.ERROR_MESSAGE
JOptionPane.INFORMATION_MESSAGE
JOptionPane.WARNING_MESSAGE
JOptionPane.QUESTION_MESSAGE



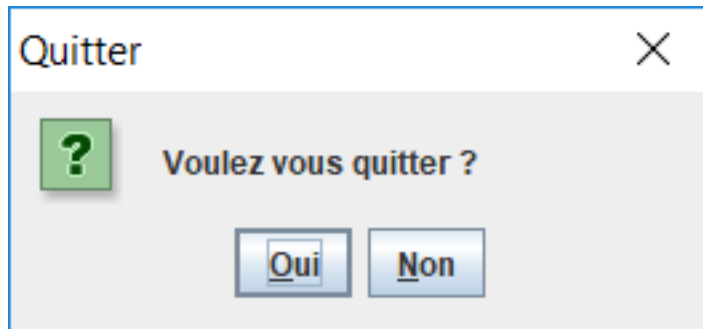
2) **JOptionPane.showConfirmDialog** pour demander à l'utilisateur de faire un choix :

```
if( JOptionPane.showConfirmDialog(null,  
    "Voulez vous quitter ?",  
    "Quitter",  
    JOptionPane.YES_NO_OPTION,  
    JOptionPane.ERROR_MESSAGE) == JOptionPane.YES_OPTION )  
{  
    System.exit(0) ;  
}
```



Paramètres :

```
if( JOptionPane.showConfirmDialog(null,  
    "Voulez vous quitter ?",  
    "Quitter",  
    JOptionPane.YES_NO_OPTION,  
    JOptionPane.ERROR_MESSAGE) == JOptionPane.YES_OPTION )  
{  
    System.exit(0);  
}
```



Boutons :

JOptionPane.DEFAULT_OPTION
JOptionPane.YES_NO_OPTION
JOptionPane.YES_NO_CANCEL_OPTION
JOptionPane.OK_CANCEL_OPTION

Code de retour :

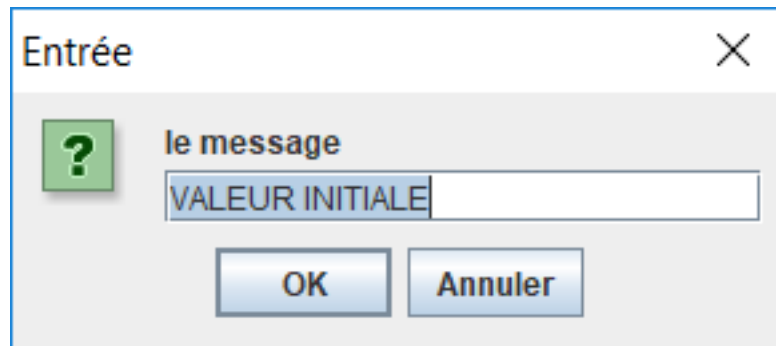
OK_OPTION
NO_OPTION
YES_OPTION
CANCEL_OPTION

Icône :

JOptionPane.PLAIN_MESSAGE
JOptionPane.ERROR_MESSAGE
JOptionPane.INFORMATION_MESSAGE
JOptionPane.WARNING_MESSAGE
JOptionPane.QUESTION_MESSAGE

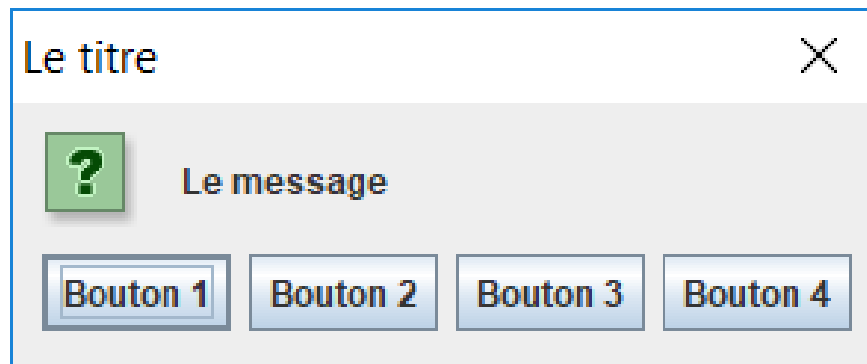
3) **JOptionPane.showInputDialog** pour demander à l'utilisateur de saisir une valeur :

```
String retour = JOptionPane.showInputDialog(null,  
    "le message",  
    "VALEUR INITIALE");
```



4) **JOptionPane.showOptionDialog** pour afficher une boîte de message personnalisée :

```
String textesBoutons[]={ "Bouton 1", "Bouton 2",  
                        "Bouton 3", "Bouton 4"};  
  
int retour = JOptionPane.showOptionDialog(null,  
    "Le message", "Le titre",  
    JOptionPane.DEFAULT_OPTION,  
    JOptionPane.QUESTION_MESSAGE, null,  
    textesBoutons, textesBoutons[0]);  
  
if( retour!=JOptionPane.CLOSED_OPTION)  
    System.out.println("Indice du bouton cliqué : " + retour);  
else  
    System.out.println("pas de bouton cliqué");
```



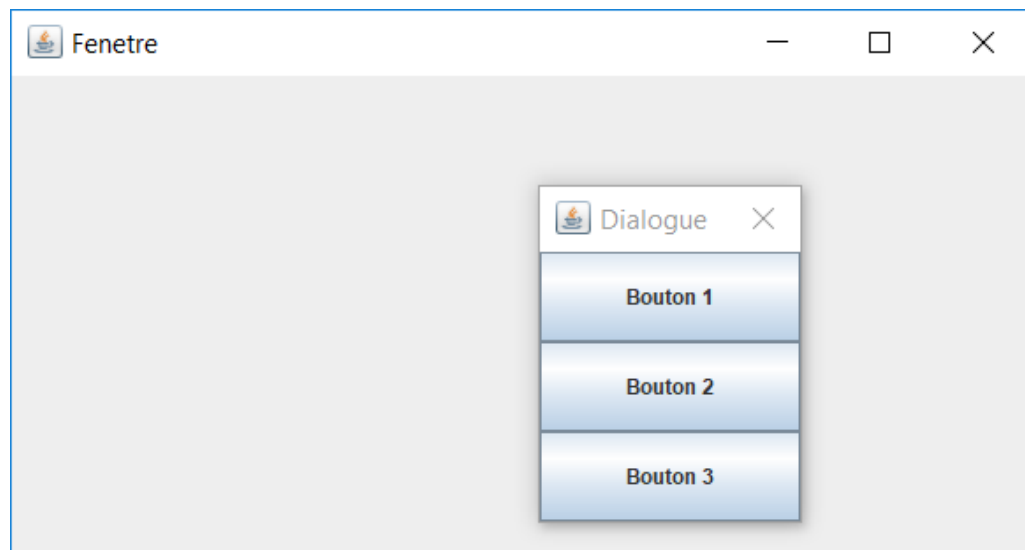
Boîte de dialogue : JDialog

<https://docs.oracle.com/en/java/javase/13/docs/api/java.desktop/javax/swing/JDialog.html>

Classe permettant de faire apparaître une boîte de dialogue dans laquelle on peut placer tout type de composants (boutons, champs éditables, labels, cases à cocher, etc.).

Deux types de comportement pour la boîte :

- **Modale** : la boîte est bloquante, on ne peut pas accéder au reste de l'interface sans fermer la boîte.
- **Non modale** : la boîte n'est pas bloquante, on peut continuer à accéder au reste de l'interface.



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

class TestJDialog
{
    public static void main(String[] args)
    {
        JFrame fenetre = new JFrame("Fenetre");
        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fenetre.setSize(600, 400);
        fenetre.setVisible(true);
    }
}
```

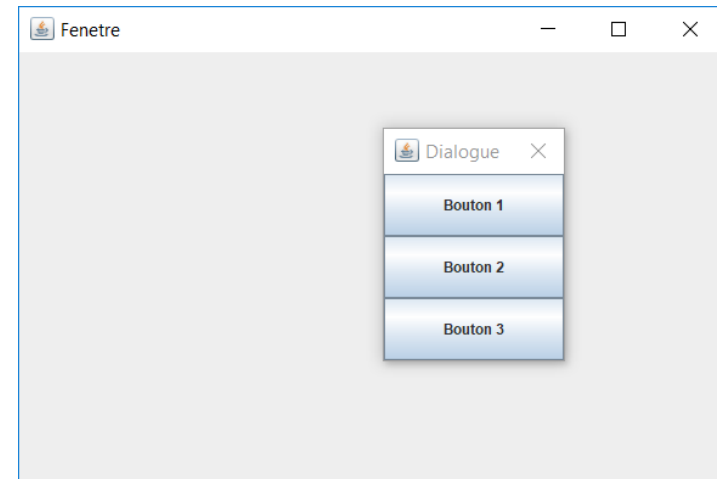
```
// Dernier argument = true : boîte modale  
//                      = false : boîte non modale
```

```
JDialog dialog = new JDialog(fenetre, "Dialogue", false);
```

```
dialog.setBounds(300, 100, 100, 200);  
JPanel dialogPanel = new JPanel();  
GridLayout layout = new GridLayout(3,1);  
dialogPanel.setLayout(layout);  
JButton button1 = new JButton("Bouton 1");  
JButton button2 = new JButton("Bouton 2");  
JButton button3 = new JButton("Bouton 3");  
dialogPanel.add(button1);  
dialogPanel.add(button2);  
dialogPanel.add(button3);
```

```
dialog.getContentPane().add(dialogPanel);  
dialog.setVisible(true);
```

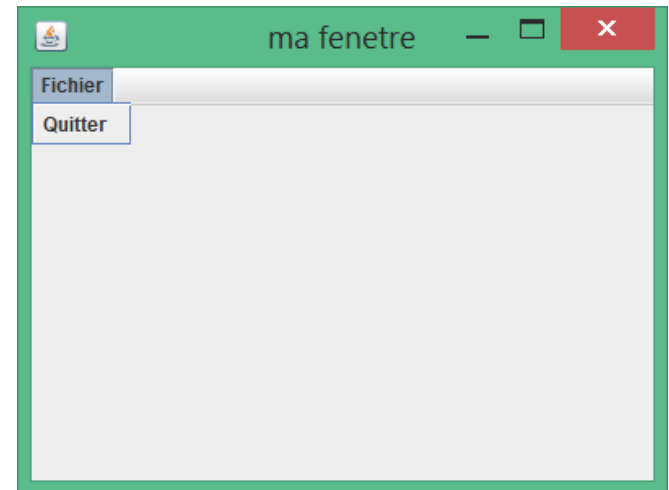
```
}  
}
```



6. Menu

Pour ajouter un menu dans une fenêtre, on utilise les classes **JMenuBar**, **JMenu** et **JMenuItem**

```
class TestFenetreSwing
{
    public static void main(String[] args)
    {
        FenetreSwing fenetre;
        fenetre = new FenetreSwing("ma fenetre");
    }
}
```



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

```
class FenetreSwing extends JFrame implements ActionListener  
{  
    FenetreSwing(String s)  
    {  
        super(s);  
        SetSize(400,300);  
    }  
}
```

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

```
JMenu menu = new JMenu("Fichier");  
menuBar.add(menu);
```

```
JMenuItem item =new JMenuItem("Quitter");  
menu.add(item);
```

```
item.setActionCommand("menu_quitter");  
item.addActionListener(this);
```

```
setVisible(true);  
}
```

```
public void actionPerformed(ActionEvent evenement)  
{  
    if (evenement.getActionCommand().equals("menu_quitter"))  
        System.exit(0);  
}  
}
```

Menu contextuel (JPopupMenu)

Permet de faire apparaître un menu contextuel, par exemple en faisant un clic droit sur un élément.

```
JPopupMenu popupMenu = new JPopupMenu();  
JMenuItem item = new JMenuItem( "Undo" );  
item.addActionListener(this);  
popupMenu.add(item);
```

Pour faire apparaître ce menu lors d'un clic droit sur un JTextArea :

```
JTextArea ta = new JTextArea();  
getContentPane().add(ta);  
ta.addMouseListener( new MouseAdapter() {  
    public void mousePressed( MouseEvent event ) {  
        if( event.getButton() == MouseEvent.BUTTON3 ) {  
            popupMenu.show( event.getComponent(), event.getX(), event.getY() );  
        }  
    }  
} );
```

Icônes (Imagelcon)

Les objets de cette classe permettent de définir des icônes à partir d'images (gif, png, jpg) qui peuvent être ajoutés au texte d'un **JLabel**, d'un **JButton**, etc.

```
Icon monIcône = new Imagelcon("Image.gif");
```

```
// Un JLabel
```

```
JLabel monLabel = new JLabel("Mon Label");
```

```
monLabel.setIcon(monIcône);
```

```
monLabel.setHorizontalAlignment(JLabel.RIGHT);
```

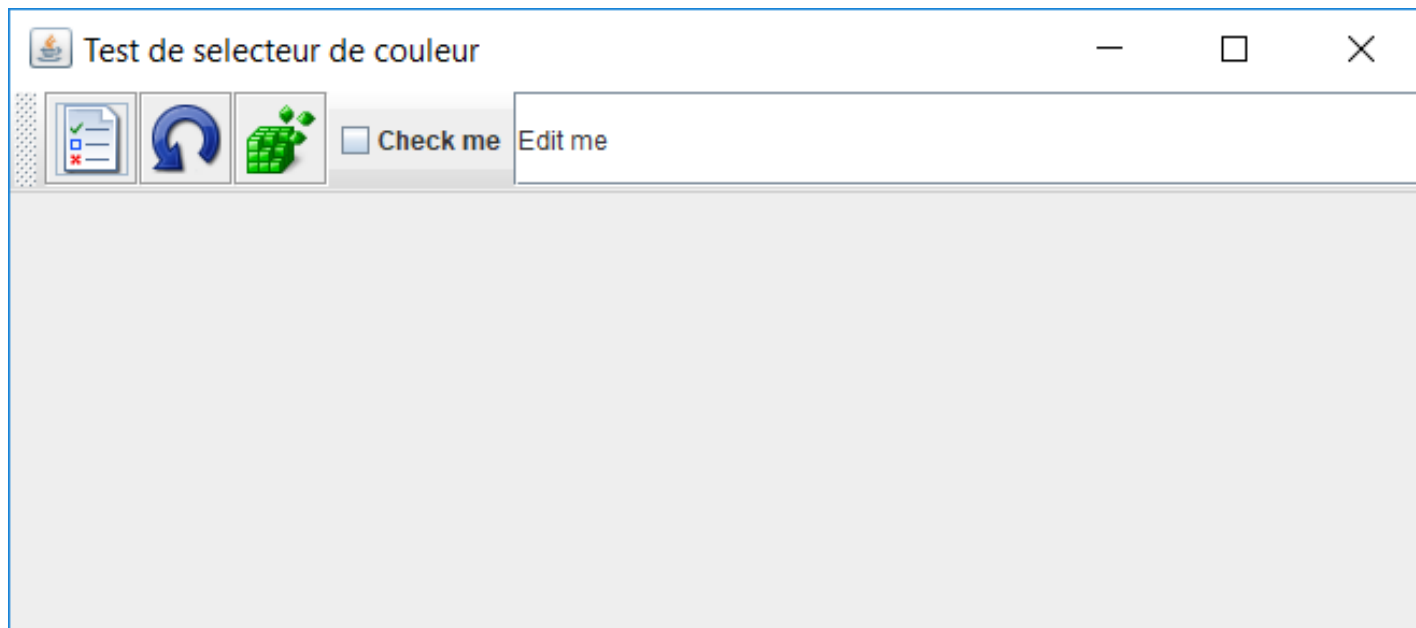
```
// Un JButton
```

```
JButton monBouton = new JButton("Mon bouton", monIcône);
```

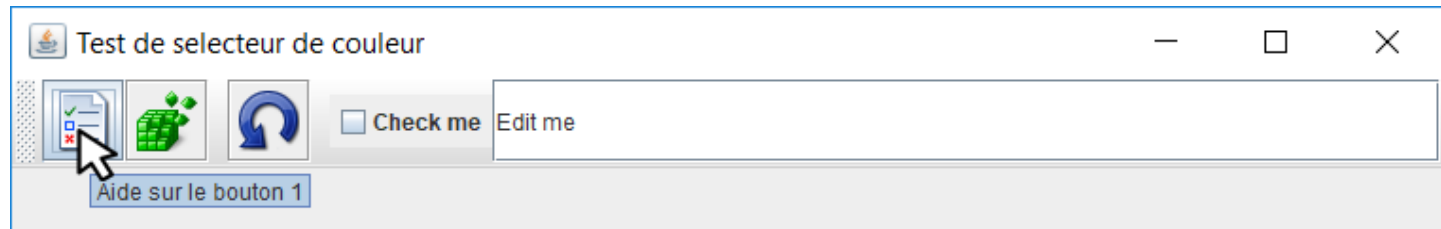

Barre d'outils (JToolBar)

Permet d'ajouter une barre d'outils pouvant être composée de boutons (**JButton**) dotés ou pas d'icône, mais aussi de n'importe quel type de composant : **JTextField**, **JCheckBox**, etc.

Cette barre est « dockable », c'est-à-dire qu'elle peut être déplacée et qu'elle peut s'accrocher directement en haut, en bas ou sur les côtés de la fenêtre.



Exemple :



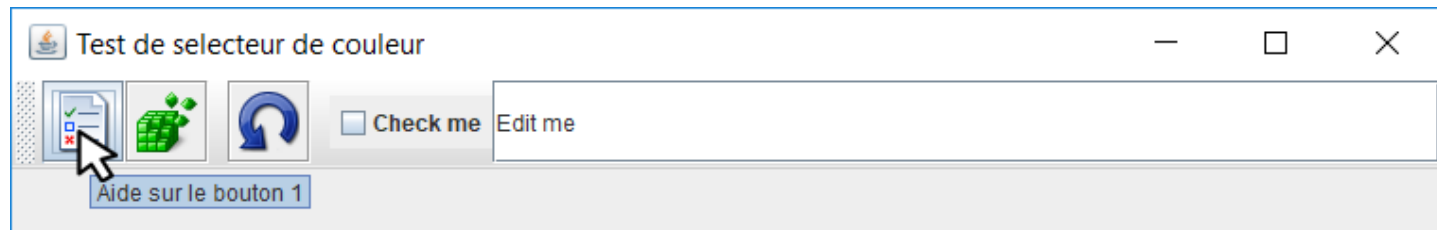
```
JToolBar toolbar = new JToolBar();
```

```
JButton btn1 = new JButton( new ImageIcon( "To_do_list.png" ) );  
btn1.setToolTipText( "Aide sur le bouton 1" );  
btn1.addActionListener( this::btn1Listener );           // Ajoute un écouteur  
toolbar.add( btn1 );
```

```
JButton btn2 = new JButton( new ImageIcon( "Registry.png" ) );  
toolbar.add( btn2 );
```

```
toolbar.addSeparator();  
JButton btn3 = new JButton( new ImageIcon( "Revert.png" ) );  
toolbar.add( btn3 );
```

```
toolbar.addSeparator();  
toolbar.add( new JCheckBox( "Check me" ) );  
toolbar.add( new JTextField( "Edit me" ) );  
getContentPane().add(toolbar, BorderLayout.NORTH);  
// Utiliser BorderLayout.SOUTH pour accrocher la toolbar en bas
```



```
JButton btn1 = new JButton( new ImageIcon( "To_do_list.png" ) );  
btn1.setToolTipText( "Aide sur le bouton 1" );  
btn1.addActionListener( this::btn1Listener );           // Ajoute un écouteur  
toolbar.add( btn1 );
```

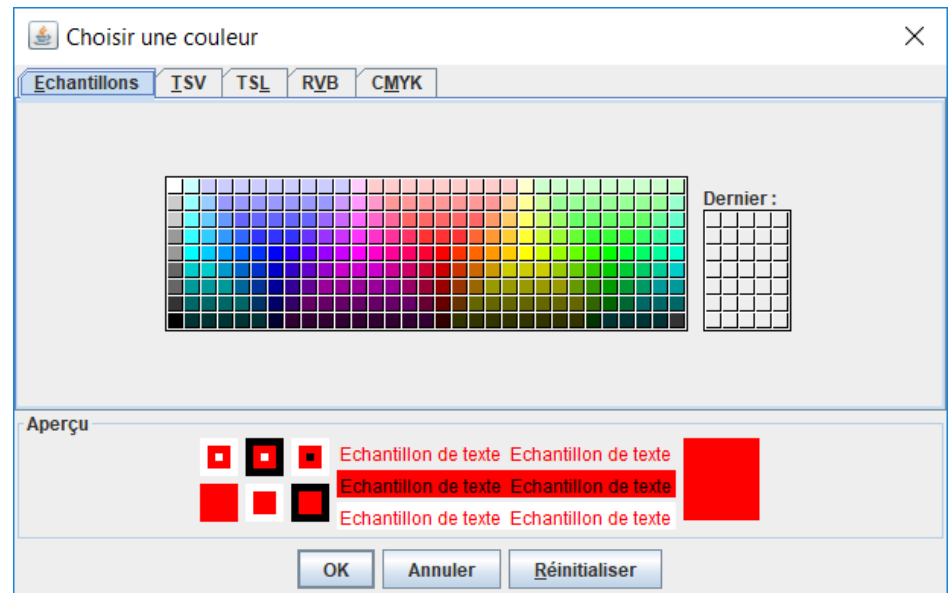
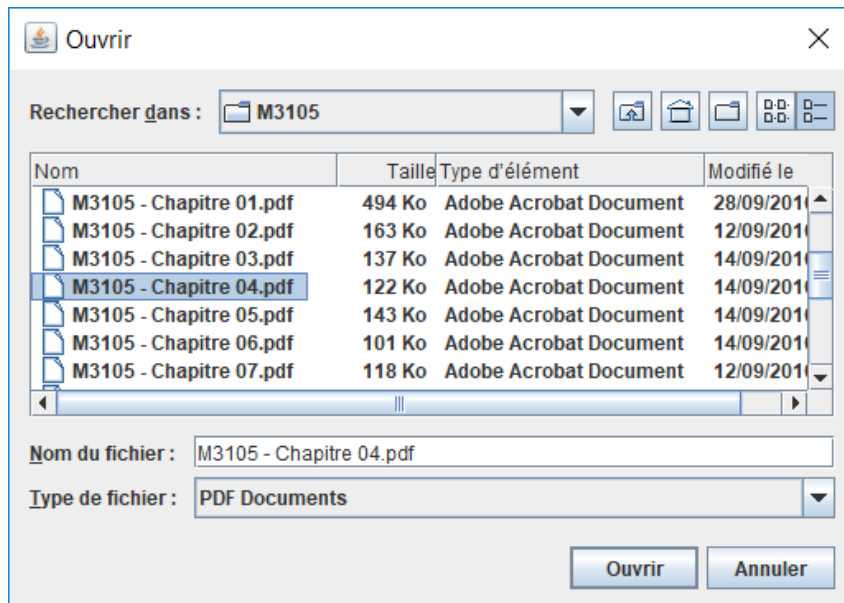
```
private void btn1Listener((ActionEvent event) )  
{  
    System.out.println("Clic sur le bouton 1");  
}
```

Si on clique sur le bouton **btn1**, la méthode **btn1Listener()** est appelée.

Note : dans **btn1.addActionListener(this::btn1Listener);**
la notation **this::btn1Listener** correspond à une référence sur la méthode **btn1Listener()**

7. Boîtes de sélection

Des classes Java permettent de faire apparaître des boîtes de sélection permettant de choisir un nom de fichier ou une couleur.

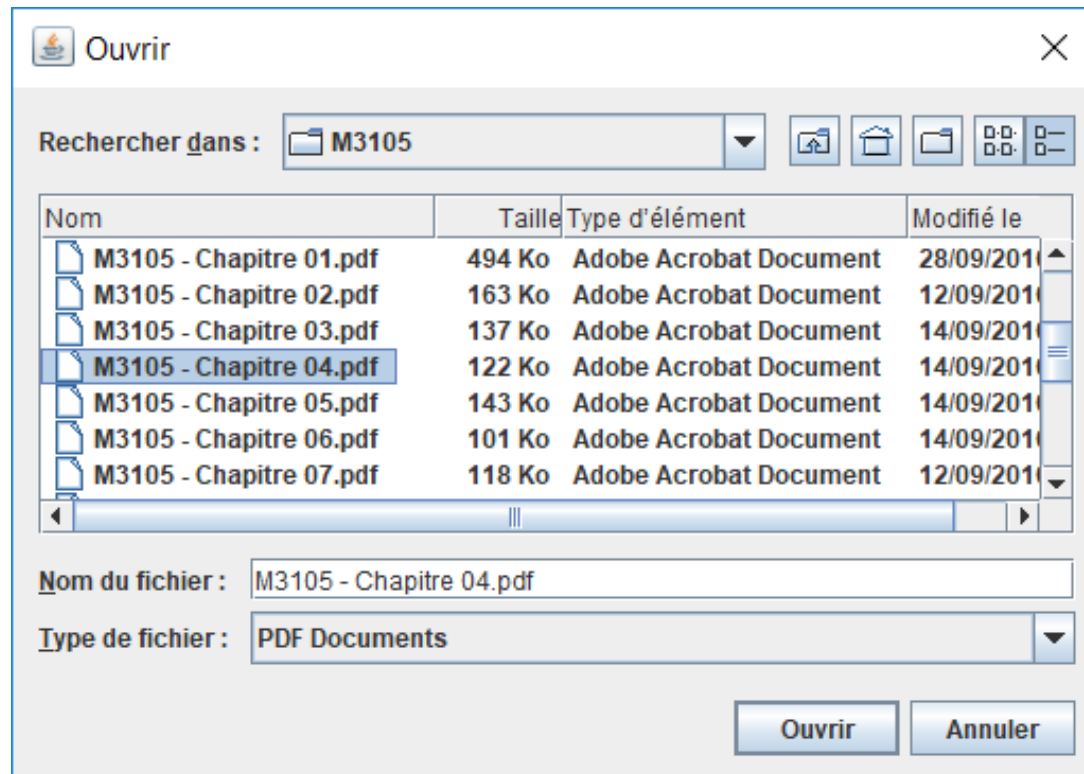


7.1 Sélecteur de fichier : JFileChooser

Permet de faire apparaître une boîte de sélection de fichier ou de répertoire, avec possibilité de filtrer les fichiers selon leur extension (ex : n'afficher que les fichiers d'extension .pdf, que les images, etc.).

<https://docs.oracle.com/en/java/javase/13/docs/api/java.desktop/javax.swing/JFileChooser.html>

<https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html>



Exemple :

```
import javax.swing.filechooser.FileNameExtensionFilter;
```

```
JFileChooser selecteur = new JFileChooser();
```

```
selecteur.addChoosableFileFilter(new FileNameExtensionFilter("PDF Documents", "pdf"));
```

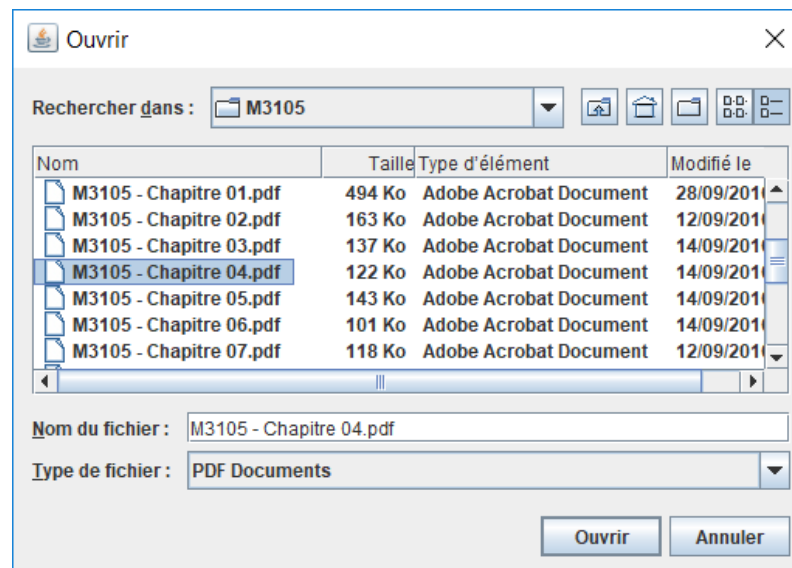
```
selecteur.addChoosableFileFilter(new FileNameExtensionFilter("Images", "jpg", "gif"));
```

```
int resultat = selecteur.showOpenDialog(null);
```

```
if(resultat == JFileChooser.APPROVE_OPTION) {
```

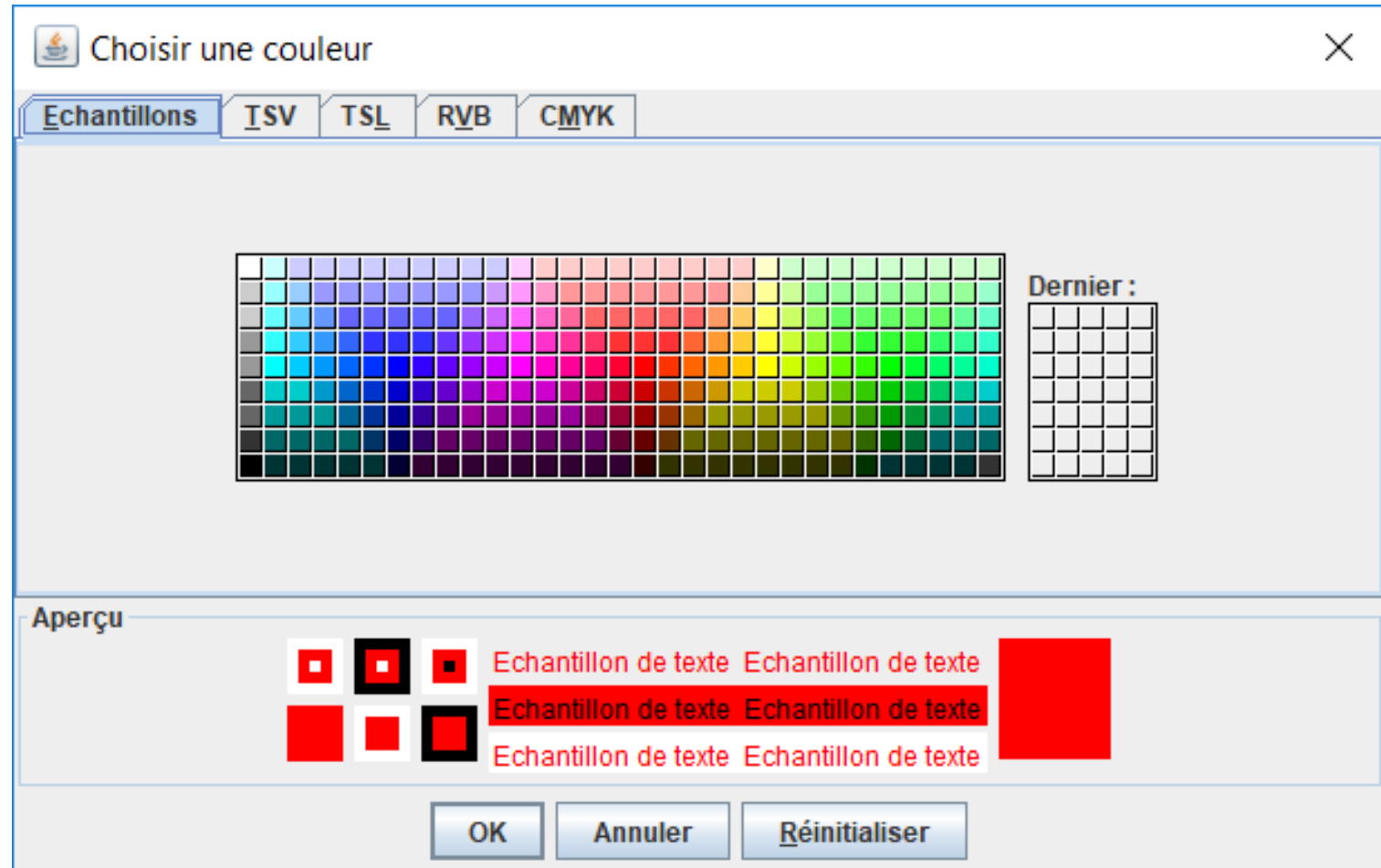
```
    System.out.println("Fichier choisi: " + selecteur.getSelectedFile().getAbsolutePath());
```

```
}
```



7.2 Sélecteur de couleur: JColorChooser

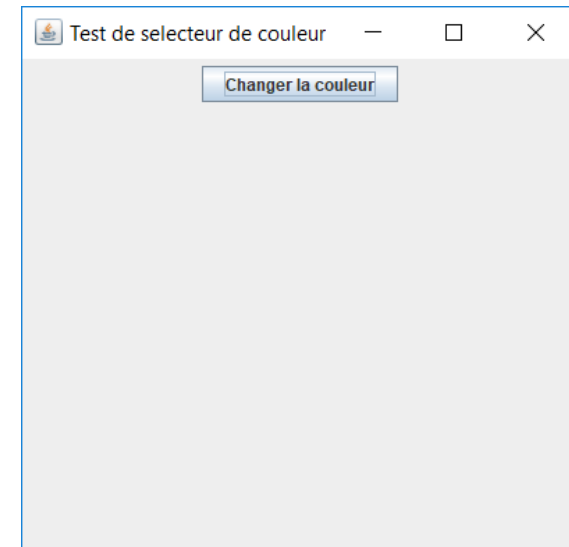
Permet de faire apparaître une boîte de sélection de couleur.



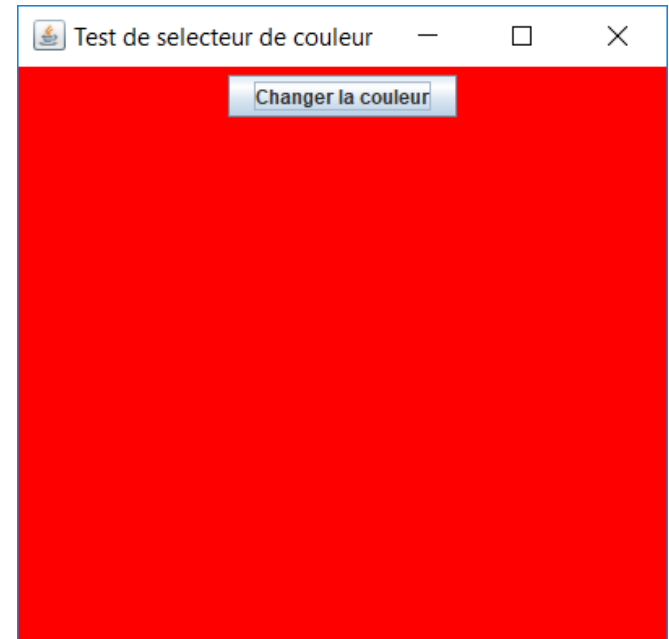
Exemple :

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

```
public class Fenetre extends JFrame implements ActionListener  
{  
    Fenetre(String titre)  
    {  
        super(titre);  
        JButton bouton = new JButton("Changer la couleur");  
        bouton.addActionListener(this);  
        getContentPane().setLayout(new FlowLayout());  
        getContentPane().add(bouton);  
  
        setSize(400,400);  
        setVisible(true);  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
    }  
}
```




```
public void actionPerformed(ActionEvent e)
{
    Color initcolor = Color.RED;
    Color color = JColorChooser.showDialog(this,"Choisir une couleur", initcolor);
    getContentPane().setBackground(color);
}
}
```



```
class Principal
{
    public static void main(String[] args)
    {
        Fenetre f = new Fenetre("Test de selecteur de couleur");
    }
}
```

8. Programmation évènementielle

8.1 Principe

Pour que des composants (boutons, champs éditables, etc.) soient fonctionnels, il faut implémenter une interface *récepteur d'évènements* (« *listener* ») qui donne à leurs conteneurs la possibilité de les « entendre ».

8.2 Gestion de l'événement de clic sur un bouton

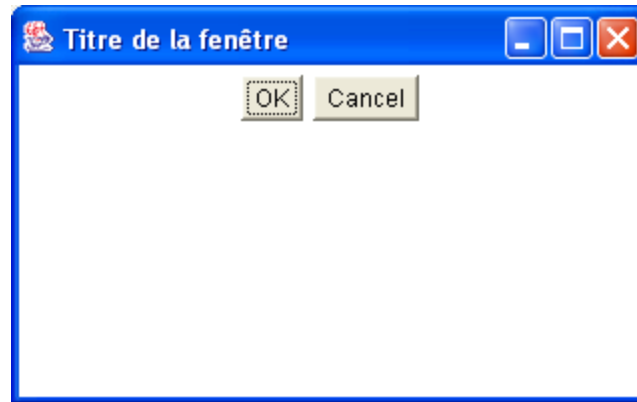
Pour gérer l'événement de clic sur un de ses boutons, la fenêtre qui le contient doit implémenter l'interface **ActionListener**.

Cette interface requiert l'implémentation de la méthode **actionPerformed()** par la classe de fenêtre.

Il faut aussi invoquer les méthodes **addActionListener()** des boutons pour indiquer que la fenêtre est un récepteur d'action.

Les composants que la classe écoute sont appelés *objets sources*, parce qu'ils peuvent générer des objets **ActionEvent** en réaction aux actions de l'utilisateur.

Dans l'exemple suivant, les objets **bouton1** et **bouton2** de la fenêtre sont les objets sources.



```
// Début du fichier BoutonFenetre.java
```

```
import java.awt.*;
```

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

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

```
class BoutonFenetre extends JFrame implements ActionListener
```

```
{
```

```
    JButton bouton1, bouton2;
```

```
BoutonFenetre(String s)
{
    super(s) ;
    setSize(320,200) ;
    setLayout(new FlowLayout()) ;
    bouton1 = new JButton("OK") ;
    bouton2 = new JButton("Cancel") ;
    bouton1.setActionCommand("bouton ok") ;
    bouton2.setActionCommand("bouton cancel") ;
    bouton1.addActionListener(this) ;
    bouton2.addActionListener(this) ;
    add(bouton1) ;
    add(bouton2) ;
    setVisible(true) ;
}
```

```
public void actionPerformed(ActionEvent evenement)
{
    if(evenement.getActionCommand().equals("bouton ok"))
        System.out.println("OK sélectionné");

    if(evenement.getActionCommand().equals("bouton cancel"))
        System.out.println("Cancel sélectionné");
}
}

// Fin du fichier BoutonFenetre.java
```

```
// Début du fichier TestBoutonFenetre.java
```

```
class TestBoutonFenetre
{
    public static void main(String[] args)
    {
        BoutonFenetre fenetre;
        fenetre = new BoutonFenetre ("Test de boutons");
    }
}
```

```
// Fin du fichier TestBoutonFenetre.java
```


Note

L'instruction `bouton1.addActionListener(this)` assigne au champ `actionListener` du bouton une référence à l'objet de classe `BoutonFenetre` qui le contient.

Ensuite, lorsque l'utilisateur clique sur le bouton, un objet `ActionEvent` est généré, dont le champ `actionCommand` contient la chaîne du bouton spécifiée par `setActionCommand()`.

Le système invoque ensuite la méthode `actionPerformed()` de l'objet récepteur identifié par `bouton1.actionListener` en lui passant cet objet `ActionEvent`.

Dans notre cas, ce récepteur est l'objet `fenetre`, qui regarde la valeur du champ `evenement.actionCommand` pour déterminer quel bouton a été cliqué.

8.3 Gestion des événements de fenêtre

Pour gérer les événements de fenêtre, la fenêtre doit implémenter l'interface **WindowListener**.

Cette interface requiert l'implémentation de 7 méthodes supplémentaires.

Dans cet exemple, nous n'utilisons que la méthode **windowClosing()**, appelée lorsque l'on clique sur la case « fermer » de la fenêtre, mais nous devons déclarer les 6 autres méthodes, sans toutefois y placer du code.

```
// Début du fichier BoutonFenetre.java
```

```
import java.awt.*;
```

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

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

```
class BoutonFenetre extends JFrame
```

```
    implements ActionListener, WindowListener
```

```
{
```

```
    JButton bouton;
```

```
BoutonFenetre(String s)
{
    super(s) ;
    setSize(320,200) ;
    setLayout(new FlowLayout()) ;
    addWindowListener(this) ;
    bouton = new JButton("OK") ;
    bouton.setActionCommand("bouton ok") ;
    bouton.addActionListener(this) ;
    add(bouton) ;
    setVisible(true) ;
}
```

```
public void windowClosed(WindowEvent e) {}  
public void windowDeiconified(WindowEvent e) {}  
public void windowIconified(WindowEvent e) {}  
public void windowActivated(WindowEvent e) {}  
public void windowDeactivated(WindowEvent e) {}  
public void windowOpened(WindowEvent e) {}  
public void windowClosing(WindowEvent e)  
{  
    System.exit(0);  
}
```

```
public void actionPerformed(ActionEvent evenement)
{
    if(evenement.getActionCommand().equals("bouton ok"))
        System.out.println("OK sélectionné");
}
}

// Fin du fichier BoutonFenetre.java
```

```
// Début du fichier TestBoutonFenetre.java
```

```
class TestBoutonFenetre
{
    public static void main(String[] args)
    {
        BoutonFenetre fenetre;
        fenetre = new BoutonFenetre ("Test de bouton");
    }
}
```

```
// Fin du fichier TestBoutonFenetre.java
```

8.4 Gestion des événements de souris

Pour gérer les événements liés à la souris, le cadre doit implémenter deux interfaces :

MouseListener

permet de connaître l'état des boutons de la souris.

MouseMotionListener

permet de connaître les déplacements de la souris.

a) L'interface `MouseListener`

`MouseListener` définit "l'écouteur" des événements souris: press, release, click, enter, exit.

`addMouseListener()` installe un écouteur **`MouseListener`** sur un composant.

Les méthodes à (toutes) définir sont :

```
public void mousePressed(MouseEvent e)
public void mouseReleased(MouseEvent e)
public void mouseClicked(MouseEvent e)
public void mouseEntered(MouseEvent e)
public void mouseExited(MouseEvent e)
```

`e.getX()` donne la coordonnée `x` de la souris au moment de l'événement.

`e.getY()` donne la coordonnée `y` de la souris au moment de l'événement.

Bouton pressé

```
public void mousePressed(MouseEvent e)
```

Définit le traitement quand survient l'événement qui indique qu'un bouton de la souris est pressé dans l'aire du composant.

Bouton relâché

```
public void mouseReleased(MouseEvent e)
```

Définit le traitement quand survient l'événement qui indique qu'un bouton de la souris est "relâché" dans l'aire du composant.

Bouton cliqué

```
public void mouseClicked(MouseEvent e)
```

Définit le traitement quand survient l'événement qui indique qu'un bouton de la souris est "pressé-relâché".

La souris entre dans la zone du composant

```
public void mouseEntered(MouseEvent e)
```

Définit le traitement quand survient l'événement qui indique que la souris "entre" dans l'aire du composant.

La souris sort de la zone du composant

```
public void mouseExited(MouseEvent e)
```

Définit le traitement quand survient l'événement qui indique que la souris "sort" de l'aire du composant.

Exemple

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

class Fenetre extends JFrame implements MouseListener
{
    Fenetre(String s)
    {
        super(s);
        setSize(600,600);
        setVisible(true);
        addMouseListener(this);
    }
}
```



```

public void mousePressed(MouseEvent e)
{
    System.out.println(e.getX());
    System.out.println(e.getY());

    int bouton = e.getButton();

    if (bouton == MouseEvent.BUTTON1)
        System.out.println("Bouton gauche");
    else if (bouton == MouseEvent.BUTTON2)
        System.out.println("Bouton du milieu");
    else if (bouton == MouseEvent.BUTTON3)
        System.out.println("Bouton droit");
}

public void mouseReleased(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
}

```

b) L'interface `MouseListener`

`MouseListener` définit "l'écouteur" des événements mouvement de souris : drag, move.

`addMouseListener()` installe un écouteur **`MouseListener`** sur un composant.

Les méthodes à (toutes) définir sont :

```
public void mouseMoved(MouseEvent e)
```

```
public void mouseDragged(MouseEvent e)
```

`e.getX()` donne la coordonnée **x** de la souris au moment de l'événement.

`e.getY()` donne la coordonnée **y** de la souris au moment de l'événement.

Déplacement de la souris (sans bouton appuyé)

```
public void mouseMoved(MouseEvent e)
```

Définit le traitement quand survient l'événement qui indique que la souris bouge dans l'aire du composant sans bouton enfoncé.

Déplacement de la souris (avec bouton(s) appuyé(s))

```
public void mouseDragged(MouseEvent e)
```

Définit le traitement quand survient l'événement qui indique que la souris bouge dans l'aire du composant avec au moins un bouton enfoncé. La souris peut sortir de l'aire du composant tant qu'un bouton est enfoncé.

Exemple

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

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

```
class Fenetre extends JFrame implements MouseMotionListener
```

```
{
```

```
    Fenetre(String s)
```

```
    {
```

```
        super(s);
```

```
        setSize(600,600);
```

```
        setVisible(true);
```

```
        addMouseMotionListener(this);
```

```
    }
```

```
public void mouseMoved(MouseEvent e)
{
    System.out.println(e.getX());
    System.out.println(e.getY());
}

public void mouseDragged(MouseEvent e) {}
}
```

8.5 Gestion des événements de clavier

Pour gérer les événements liés au clavier, la fenêtre doit implémenter l'interface **KeyListener**.

KeyListener définit "l'écouteur" d'événement clavier : press, release, type.

addKeyListener() installe un écouteur **KeyListener** sur un composant.

Les méthodes à (toutes) définir sont :

```
public void keyPressed(KeyEvent e)
public void keyReleased(KeyEvent e)
public void keyTyped(KeyEvent e)
```

`e.getKeyCode()` donne le code de la touche frappée au moment de l'événement; par exemple `KeyEvent.VK_3` pour le "3", `KeyEvent.VK_A` pour le "a" et le "A", `KeyEvent.VK_DOWN` pour la flèche bas, ...

Liste des codes :

<https://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyEvent.html>

`e.getKeyChar()` donne le caractère de la touche frappée au moment de l'événement.

Touche enfoncée

```
public void keyPressed(KeyEvent e)
```

Définit le traitement quand survient l'événement qui indique qu'une touche du clavier est enfoncée si le composant a le focus Clavier.

Touche relâchée

```
public void keyReleased (KeyEvent e)
```

Définit le traitement quand survient l'événement qui indique qu'une touche du clavier est relâchée si le composant a le focus Clavier.

Touche tapée

```
public void keyTyped(KeyEvent e)
```

Définit le traitement quand survient l'événement qui indique qu'une touche du clavier est tapée si le composant a le focus Clavier.

9. Affichage graphique

9.1 Principe

<https://docs.oracle.com/javase/8/docs/technotes/guides/2d/spec/j2d-bookTOC.html>

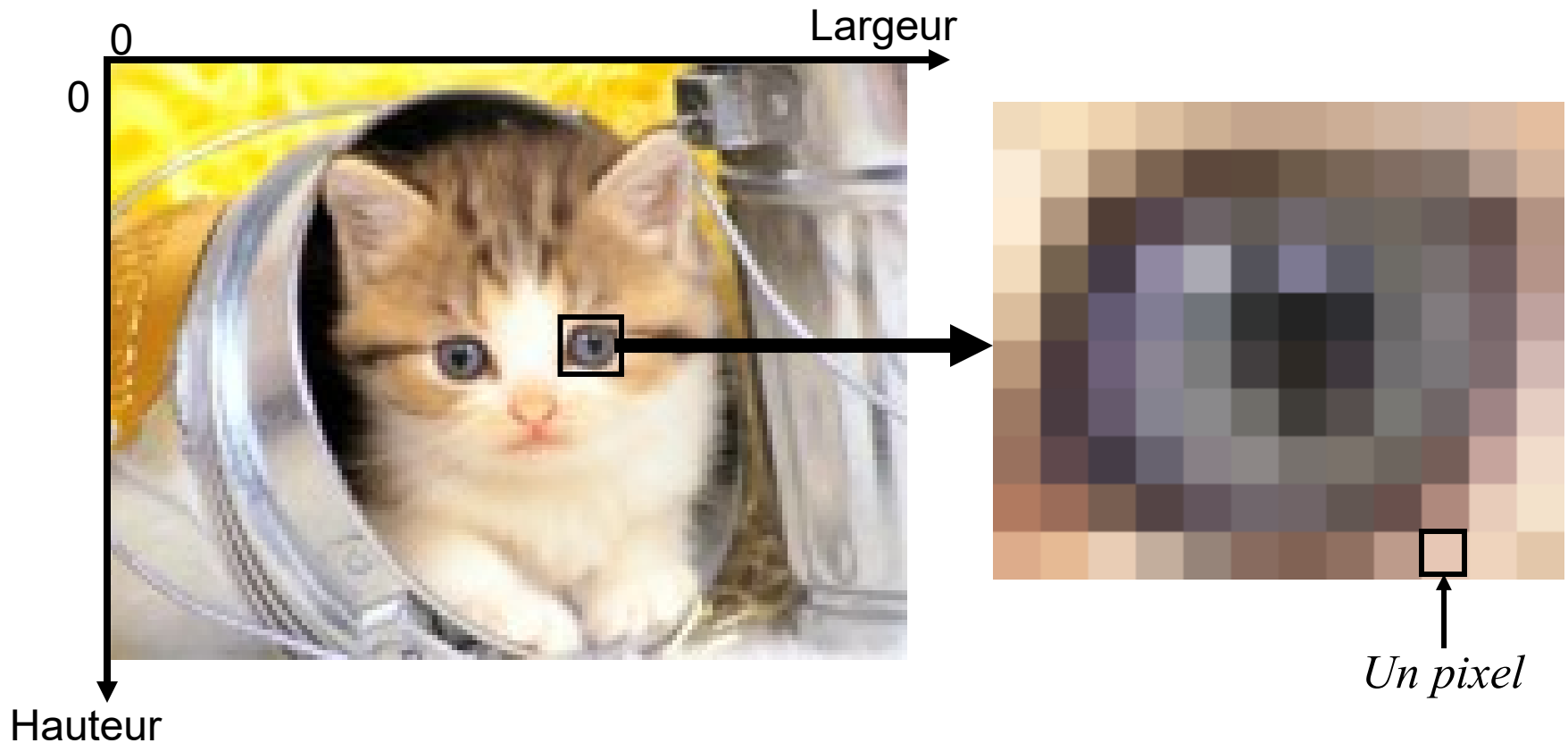
AWT et Swing permettent de réaliser des affichages graphiques :

- Primitives (point, ligne, cercle, rectangle, ...)
- Texte
- Images
- ...

L'unité utilisée pour positionner ces différents tracés est le **pixel**.

Image = matrice 2D d'éléments appelés **pixels** (contraction de « PICTure ELements »).

Chaque pixel est caractérisé par sa **couleur**.



9.2 La classe Graphics

<https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html>

L'outil de dessin est le **contexte graphique**, objet de la classe **Graphics**. Il encapsule les informations nécessaires au dessin (zone de dessin, rectangle de découpe (clipping), couleur courante)

Les méthodes de la classe **Graphics** permettent de réaliser toutes les opérations d'affichage graphique (ligne, rectangle, ellipse, texte, image, etc.)

→ Pour afficher un élément graphique il **faut** avoir un objet de classe **Graphics**.

Trois façons pour obtenir un contexte graphique :

- **Implicitement**, dans une méthode `paint()` ou `paintComponent()`
- **Explicitement**, en demandant au composant ou à une image un contexte graphique avec `getGraphics()`.
- **Explicitement**, en appelant la méthode `createGraphics()` à partir d'un `Graphics` existant.

Les objets `Graphics` qui ont été obtenus **explicitement** doivent être libérés avec la méthode `dispose()`.

9.4 Changer de couleur

```
// On suppose qu'on a un objet de classe Graphics  
// obtenu selon l'une des 3 possibilités décrites  
// auparavant :
```

```
Graphics g;
```

```
g.setColor(Color nouvelle_couleur);
```

Ex :

```
g.setColor(Color.yellow);
```

Ou :

```
Color jaune = new Color(255,255,0);
```

```
g.setColor(jaune);
```

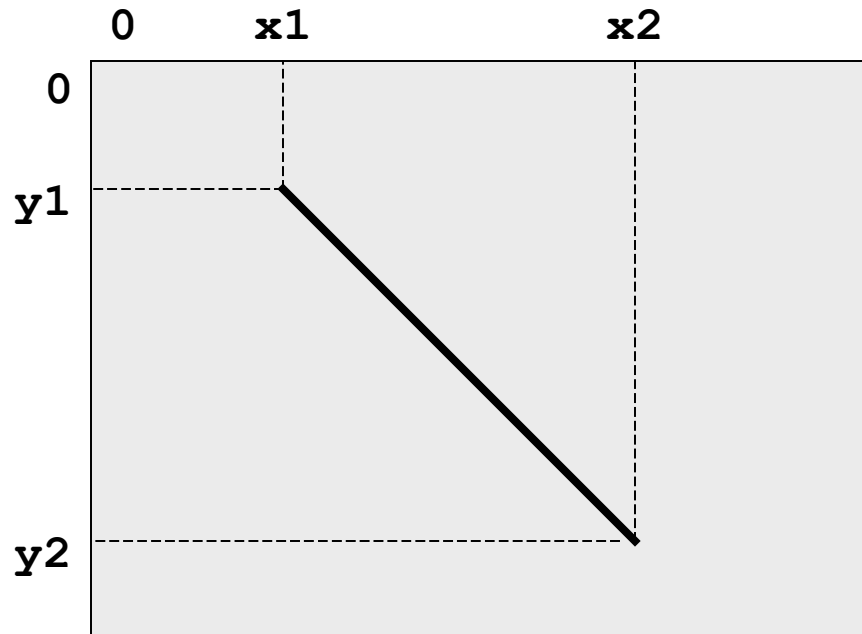
Ou :

```
g.setColor(new Color(255,255,0));
```

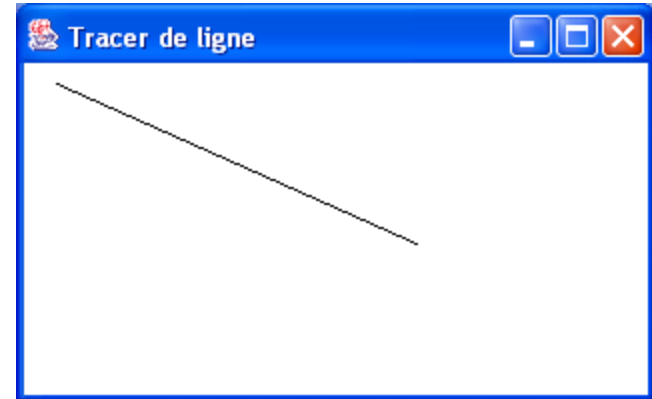
9.5 Affichage de primitives graphiques

`drawLine(int x1, int y1, int x2, int y2)`

Dessine une ligne entre les 2 points précisés.



Exemple



```
// Début du fichier TestFenetreGraphique.java
```

```
class TestFenetreGraphique
{
    public static void main(String[] args)
    {
        FenetreGraphique fen;
        fen = new FenetreGraphique("Tracer de ligne");
    }
}
```

```
// Fin du fichier TestFenetreGraphique.java
```

```
// Début du fichier FenetreGraphique.java
```

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

```
class FenetreGraphique extends JFrame
```

```
{
```

```
    FenetreGraphique(String s)
```

```
{
```

```
    super(s);
```

```
    setSize(320,200);
```

```
    setVisible(true);
```

```
    // On verra à la section 9.8 que faire cet affichage
```

```
    // ici pose problème !
```

```
    Graphics g = getGraphics();
```

```
    g.drawLine(20, 40, 200, 120);
```

```
    g.dispose();
```

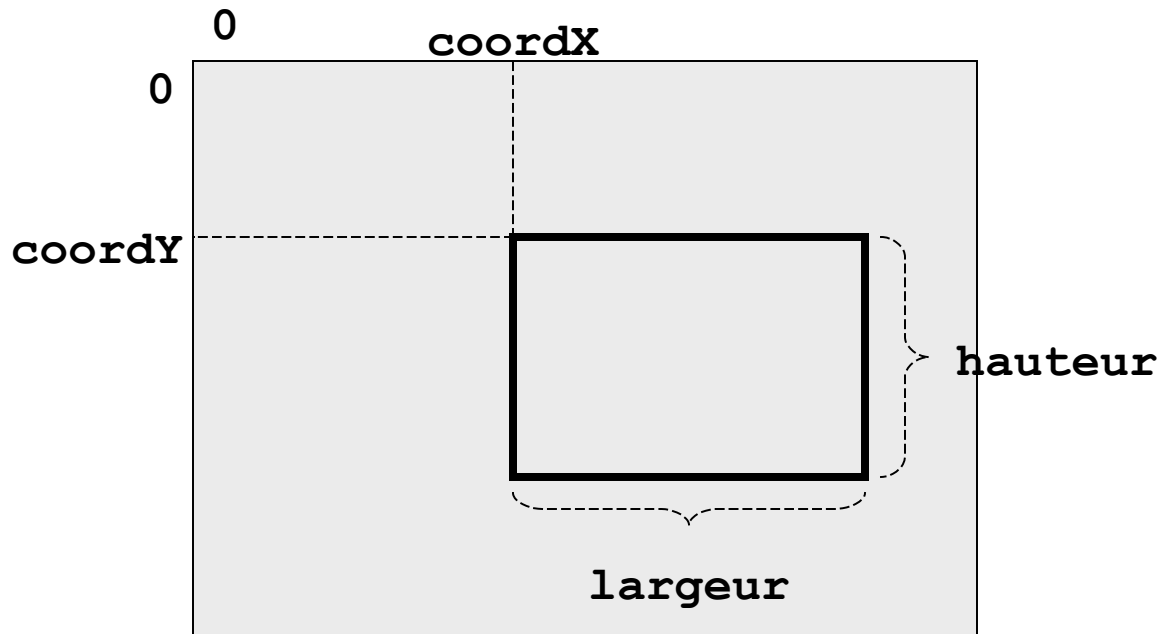
```
}
```

```
}
```

```
// Fin du fichier FenetreGraphique.java
```

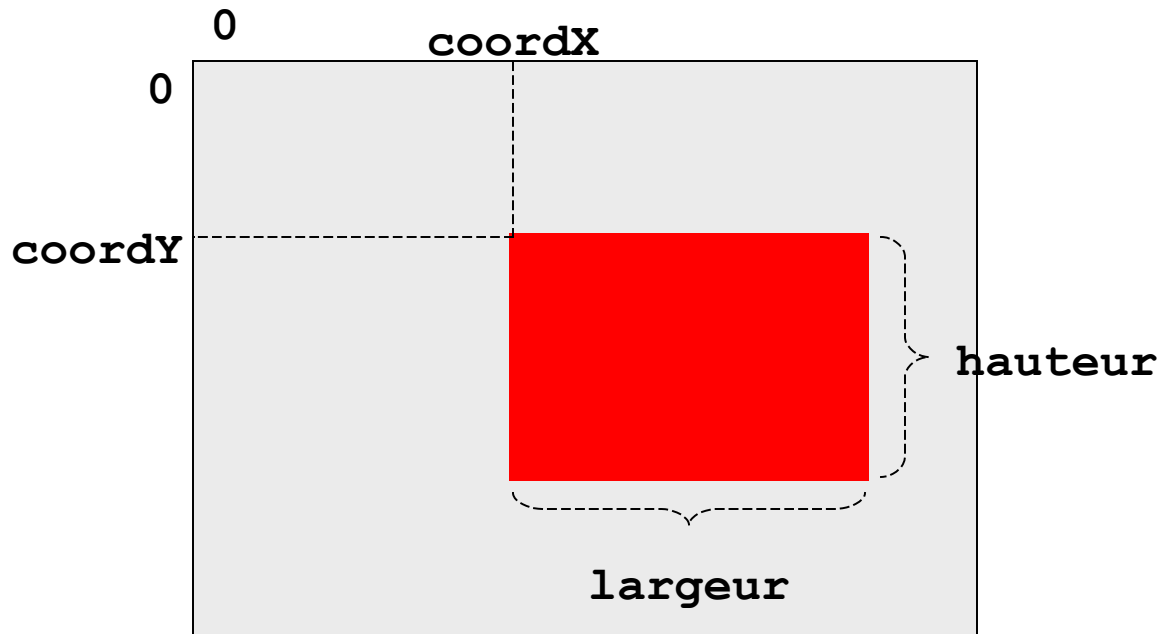
```
drawRect( int coordX, int coordY,  
          int largeur, int hauteur)
```

Dessine un rectangle en spécifiant son point supérieur gauche, sa largeur et sa hauteur (en pixels).



```
fillRect( int coordX, int coordY,  
          int largeur, int hauteur)
```

Remplit un rectangle en spécifiant son point supérieur gauche, sa largeur et sa hauteur (en pixels).



```
clearRect( int coordX, int coordY,  
           int largeur, int hauteur)
```

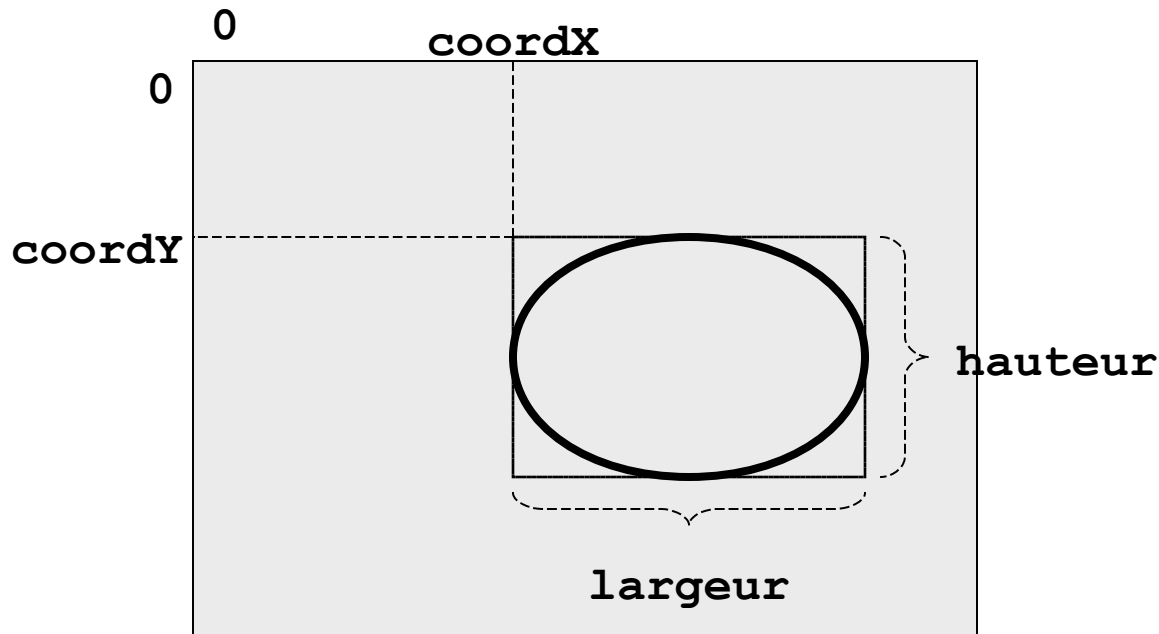
Efface le rectangle spécifié en repeignant avec la couleur de fond.

```
clipRect( int coordX, int coordY,  
          int largeur, int hauteur)
```

Restreint la zone de dessin au rectangle spécifié.

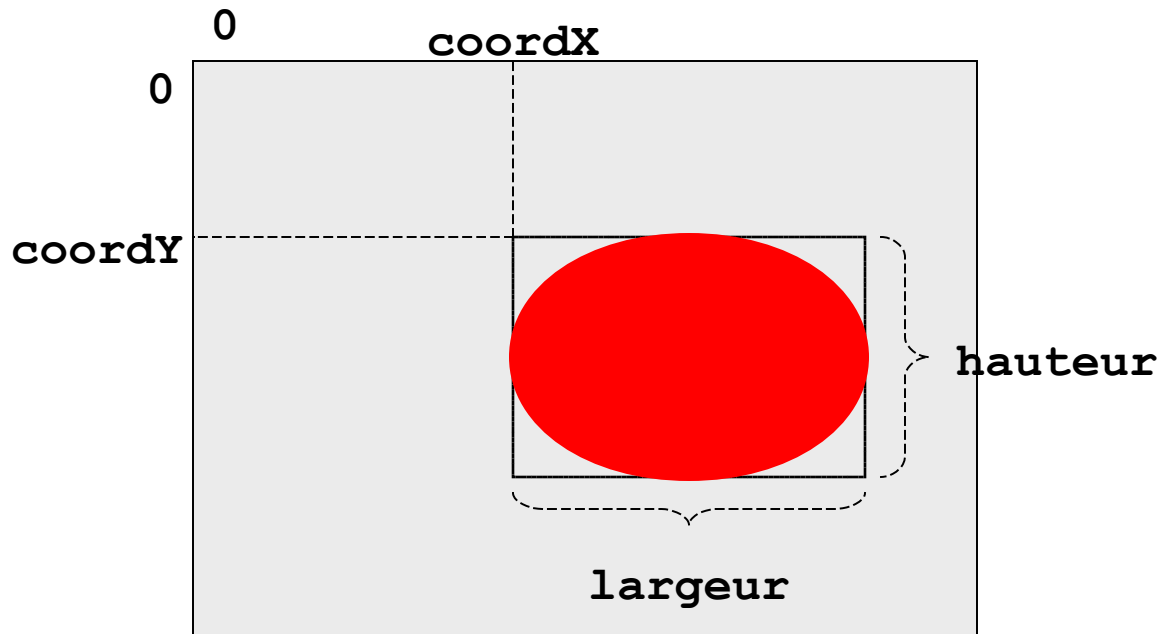
```
drawOval( int coordX, int coordY,  
          int largeur, int hauteur)
```

Dessine un ovale à l'intérieur du rectangle spécifié par son point supérieur gauche, sa largeur et sa hauteur (en pixels).




```
fillOval( int coordX, int coordY,  
          int largeur, int hauteur)
```

Remplit un ovale à l'intérieur du rectangle spécifié par son point supérieur gauche, sa largeur et sa hauteur (en pixels).



9.6 Affichage de texte

`drawString(String chaine, int coordX, int coordY)`

Dessine une chaîne de caractère à partir de la position indiquée.

```
Font font = new Font("Courier", Font.BOLD, 20);
```

```
g.setFont(font);
```

```
g.setColor(Color.red);
```

```
g.drawString("Du texte en rouge", 10, 20);
```

9.7 Affichage d'images

L'affichage d'image se fait en deux étapes :

1) Lecture : `ImageIO.read()`

On lit une image à partir d'un fichier se trouvant sur un disque, que l'on place dans un objet de classe **Image**. On peut lire des images aux formats gif, jpeg et png.

2) Affichage : `drawImage()`

L'image contenue dans un objet de classe **Image** peut ensuite être affichée.

1) Lecture

```
import java.awt.*;  
import java.io.File;  
import javax.imageio.ImageIO;  
import java.io.IOException;
```

```
Image      img;
```

```
try {  
    img = ImageIO.read(new File("une_image.jpg"));  
}  
catch (IOException e)  
{  
    e.printStackTrace();  
}
```

Remarque

On peut aussi lire une image distante en fournissant son URL :

```
URL url = new URL("http://www.site.com/images/image.jpg");  
Image image = ImageIO.read(url);
```

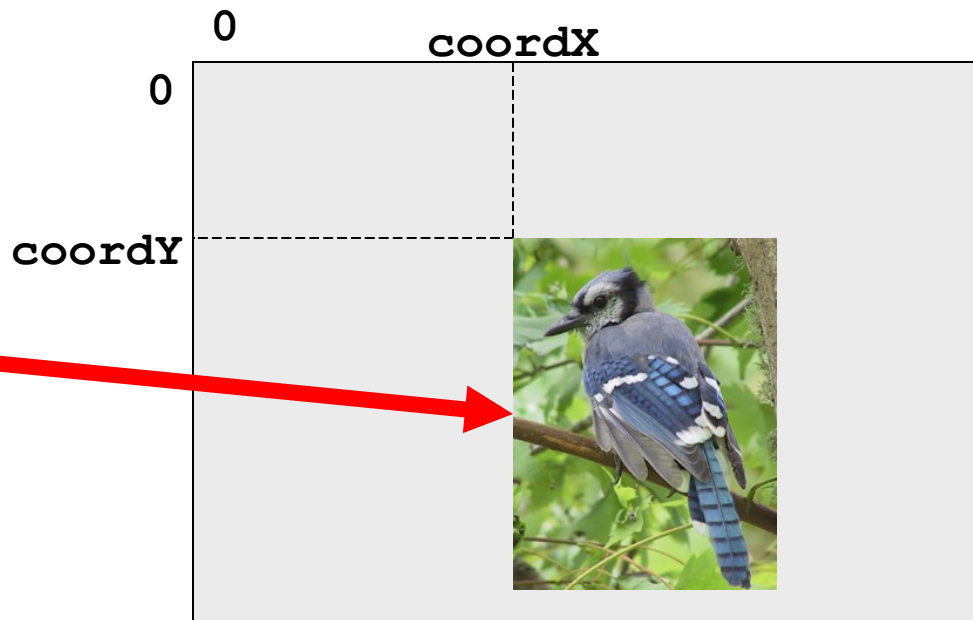
2) Affichage

Dessin de l'image, à l'aide d'un contexte graphique, dans un composant :

```
Graphics g;  
g.drawImage(img, coordX, coordY, null);
```



*Image en mémoire, dans
un objet de classe **Image***



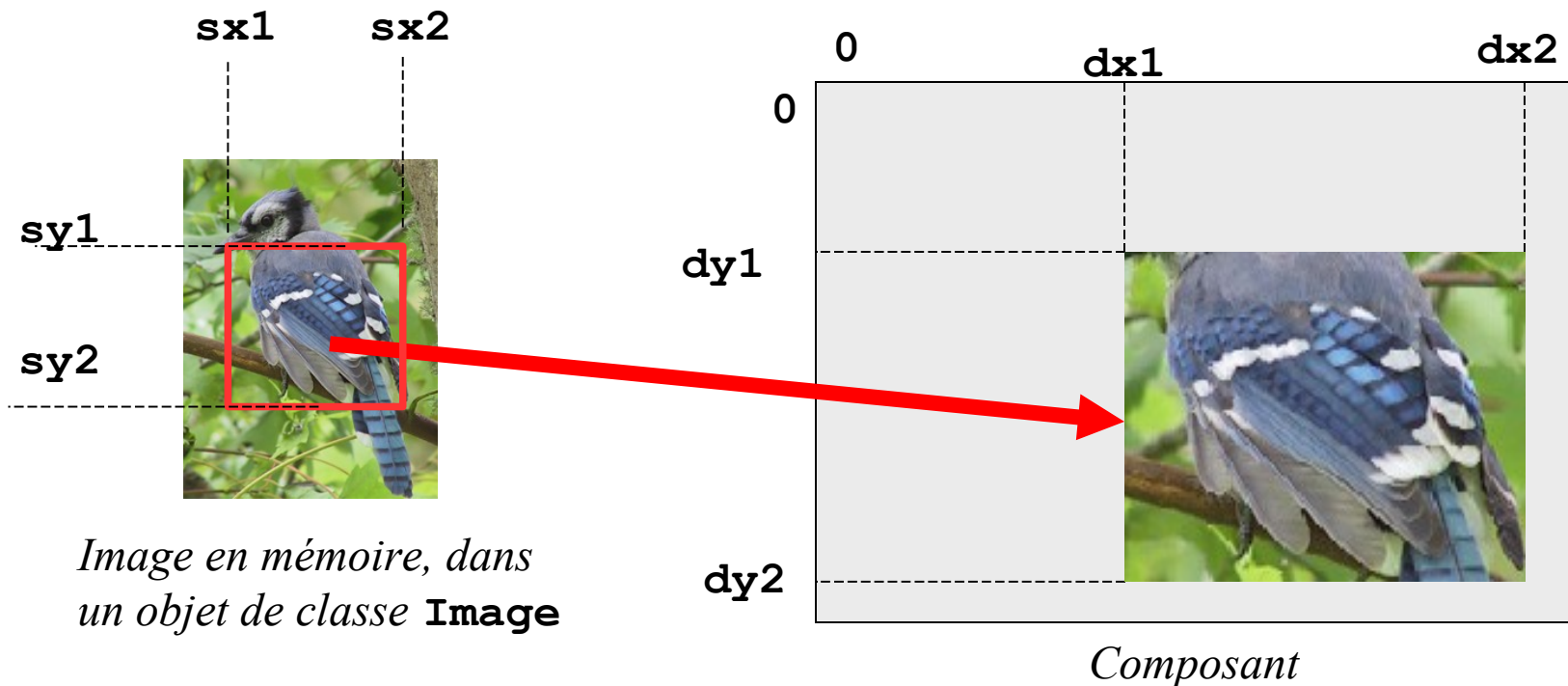
Composant

Il existe en fait 6 versions de `drawImage()` (→surcharge)

La plus complète :

```
boolean drawImage(Image img,  
                  int dx1,  
                  int dy1,  
                  int dx2,  
                  int dy2,  
                  int sx1,  
                  int sy1,  
                  int sx2,  
                  int sy2,  
                  ImageObserver observer)
```

```
boolean drawImage(Image img,  
                  int dx1,  
                  int dy1,  
                  int dx2,  
                  int dy2,  
                  int sx1,  
                  int sy1,  
                  int sx2,  
                  int sy2,  
                  ImageObserver observer)
```



Décors 2D

Le décor d'un jeu 2D est souvent constitué par un assemblage de petits blocs appelés « *tiles* » de tailles identiques (16x16, 32x32, 64x64 pixels, etc.).

Un tableau 2D (« *Tiles map* ») indique la répartition de ces tiles à l'écran. Les cases du tableau contiennent les numéros des tiles (sur 1 ou 2 octets). Ceci permet de recréer de très grands décors pour un faible coût en mémoire.

```
g.drawImage(Tiles, 0, 32, 32, 32, 64, 224, 32, 32, this);
```



Tiles

[illegible]

Tiles map

[illegible]

“**Sprites**” = éléments mobiles en 2D (personnages, objets, bonus, etc.). Images bitmap, composées de pixels.

Décomposition du mouvement sous la forme d'une planche de sprites (*sprites sheet*) dans différentes attitudes, qui seront affichés au cours du temps pour donner l'illusion du mouvement.



Zelda – A link to the past

```
g.drawImage(imgLink, 32,0, 64, 64, 300,200, 332,264, this);
```



Zelda – A link to the past

Plutôt que d'afficher successivement différentes portions d'une image, on peut aussi utiliser un sprite animé au moyen d'un fichier GIF comportant une animation (« *Animated GIF* »).

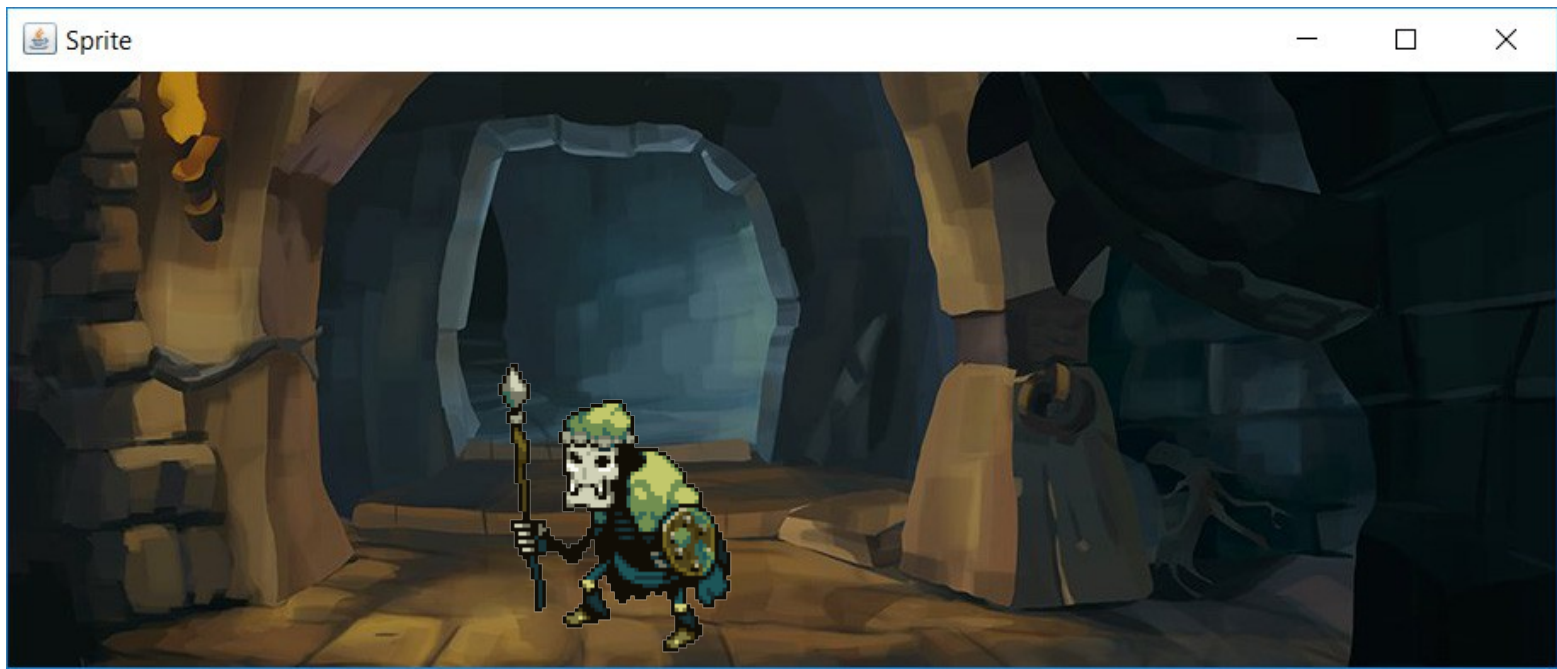
// Chargement

Image sprite ;

```
sprite = new ImageIcon("monster.gif").getImage();
```

// Affichage, au moyen d'un objet **g** de classe **Graphics** :

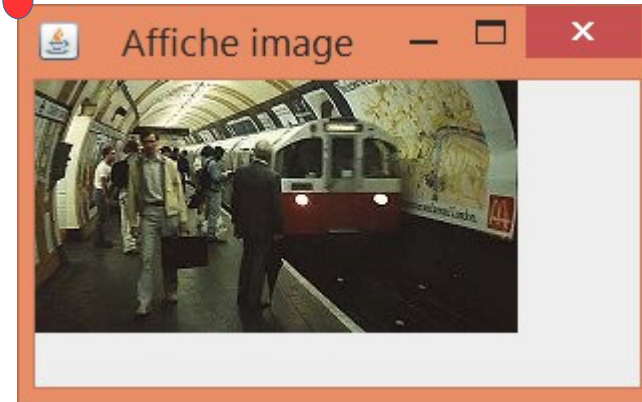
```
g.drawImage(sprite, x, y, this);
```



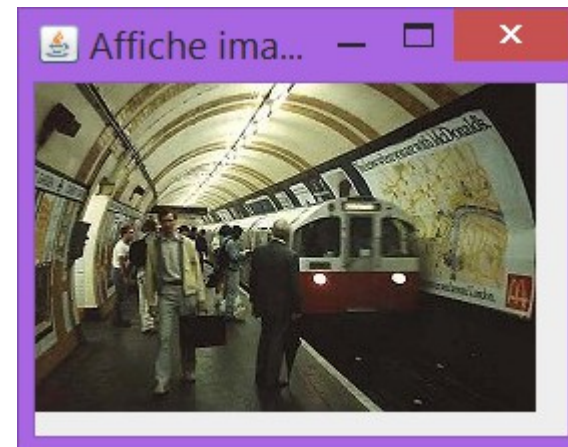
Pour afficher l'image correctement, on ne va pas l'afficher directement dans la fenêtre sinon elle sera tronquée, car l'origine de l'affichage se trouve au coin supérieur gauche de la fenêtre et non de la zone client :



origine ●

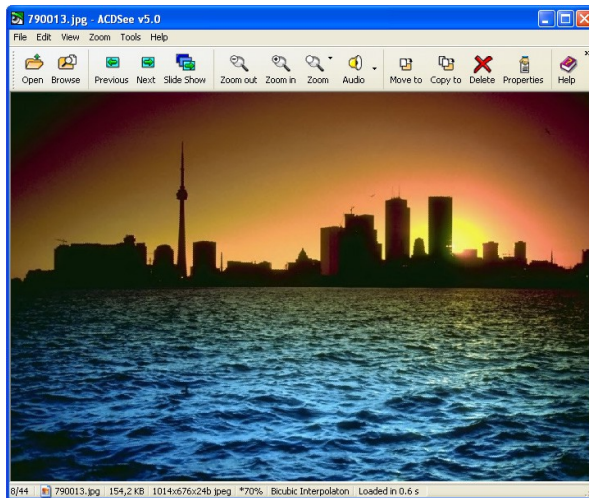


On va plutôt ajouter un objet de type JPanel dans la fenêtre, il en occupera tout le contenu, et on affichera l'image dans ce JPanel :

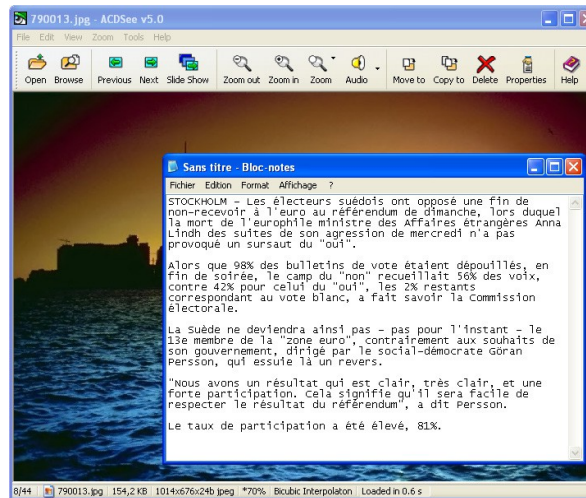


9.8 Problème de rafraîchissement de l'affichage

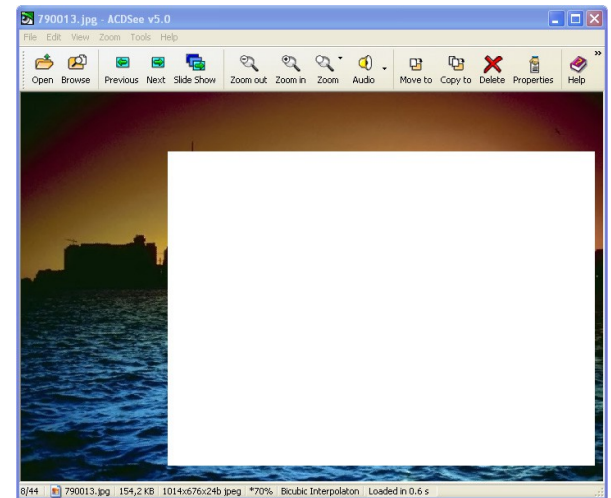
Les modifications de la fenêtre (dimensions, recouvrement par une autre fenêtre, ...) exigent le « rafraîchissement » de son contenu.



1. Fenêtre au départ



2. Recouverte par une autre application



3. Région invalide à redessiner

Méthode de la classe `Component`

`paint(Graphics g)`

Méthode appelée à chaque fois que le contenu du composant doit être redessiné (la 1ère fois, ou quand il fut masqué puis démasqué, ...); elle est à re-définir en spécifiant tout ce qui doit être dessiné.

Exemple

```
// Début du fichier TestFenetreGraphique.java
```

```
class TestFenetreGraphique
{
    public static void main(String[] args)
    {
        FenetreGraphique fen;
        fen = new FenetreGraphique("Affiche image");
    }
}
```

```
// Fin du fichier TestFenetreGraphique.java
```



```
// Début du fichier FenetreGraphique.java
```

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

```
class FenetreGraphique extends JFrame
```

```
{
```

```
    ZoneDessin  zoneDessin;
```

```
    FenetreGraphique(String s)
```

```
{
```

```
        super(s);
```

```
        setSize(320,200);
```

```
        zoneDessin = new ZoneDessin();
```

```
        setContentPane(zoneDessin);
```

```
        setVisible(true);
```

```
}
```

```
}
```

```
// Fin du fichier FenetreGraphique.java
```

```

// Début du fichier ZoneDessin.java
import java.awt.*;
import javax.swing.*;
import java.io.File;
import javax.imageio.ImageIO;
import java.io.IOException;

class ZoneDessin extends JPanel
{
    Image    img;

    ZoneDessin()
    {
        try {
            img = ImageIO.read(new File("image.jpg"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void paint(Graphics g)
    {
        g.drawImage(img, 0, 0, this);
    }
}

// Fin du fichier ZoneDessin.java

```

9.9 BufferedImage

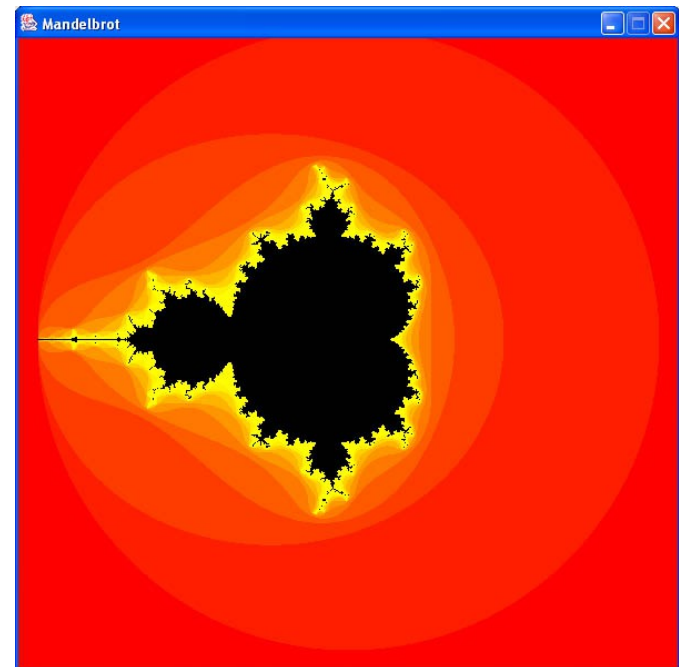
La classe **BufferedImage** est une sous-classe de la classe **Image**. Elle permet de gérer des images en mémoire (en niveaux de gris, RGB, ARGB, ...), d'en modifier les pixels, et d'afficher cette image dans une fenêtre.

<https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html>

https://www.tutorialspoint.com/java_dip/java_buffered_image.htm

Intérêt : construire une image en mémoire et s'en resservir par la suite.

Utile lorsque la construction de cette image prend du temps (ex : calcul et affichage de fractale) mais qu'on a pourtant besoin de rafraîchir rapidement la fenêtre.



Constructeur :

```
BufferedImage(int width, int height, int imageType)
```

Lire un pixel :

```
int p = getRGB(int x, int y) ;  
Color c = new Color(p);
```

Ecrire un pixel :

```
void setRGB(int x, int y, int couleur) ;
```

On peut aussi utiliser la méthode **getGraphics()** de **BufferedImage** pour obtenir un objet de type **Graphics** et utiliser toutes les méthodes de dessin que nous avons vu (`drawRect()`, `drawOval()`, `drawImage()`, `drawString()`...) pour dessiner dans cette image en mémoire.

Ex :

```
BufferedImage bufferedImage = new BufferedImage (200,200,  
                                                    BufferedImage.TYPE_INT_RGB);  
Graphics g = bufferedImage.getGraphics();  
g.drawString("Une chaîne", 10,20);
```

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

public class Test extends JPanel
{
    public void paint(Graphics g)
    {
        Image img = creationImage();
        g.drawImage(img, 50,30,this);
    }

    private Image creationImage()
    {
        BufferedImage bufferedImage = new BufferedImage (200,200,
                                                            BufferedImage.TYPE_INT_RGB);
        Graphics g = bufferedImage.getGraphics();
        g.drawString("Une chaîne", 10,20);

        return bufferedImage;
    }
}
```

```
public static void main(String[] args)
{
    JFrame frame = new JFrame();
    frame.getContentPane().add(new Test());

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(200, 200);
    frame.setVisible(true);
}
}
```

9.10 Graphics2D

La classe **Graphics2D** hérite de la classe **Graphics** et lui ajoute de nombreuses fonctionnalités (tracé de courbes, épaisseur de trait, remplissage de formes avec un dégradé ou une texture, rotation, mise à l'échelle, filtres sur les images, ...).

<https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html>

<https://docs.oracle.com/javase/tutorial/2d/overview/index.html>

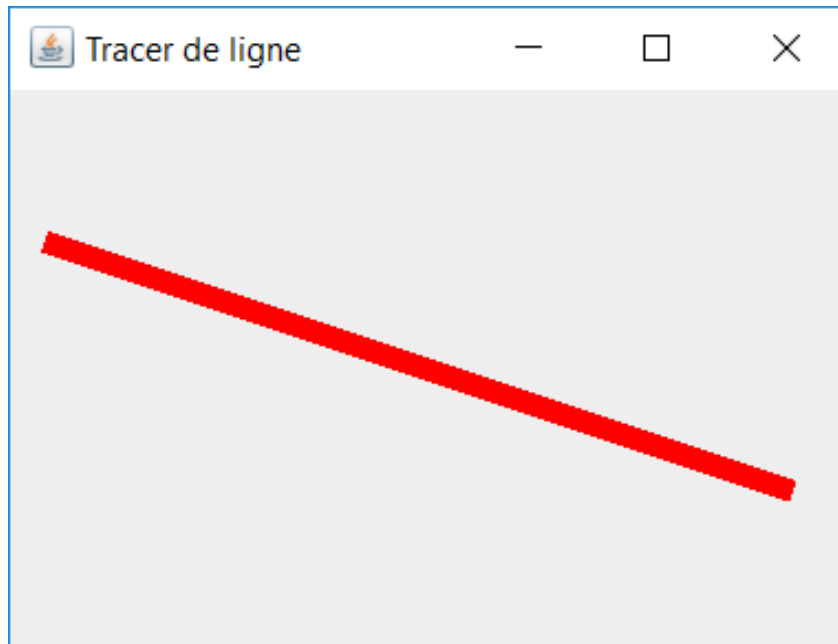
On peut obtenir un objet de type **Graphics2D** à partir d'un objet de type **Graphics** comme ceci :

```
public void paint (Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    ...
}
```


Exemple : tracé d'une ligne rouge épaisse de 10 pixels

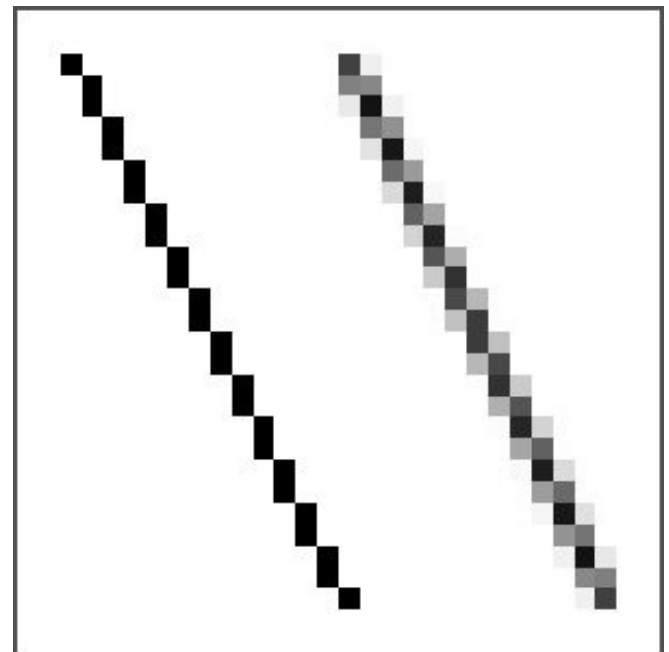
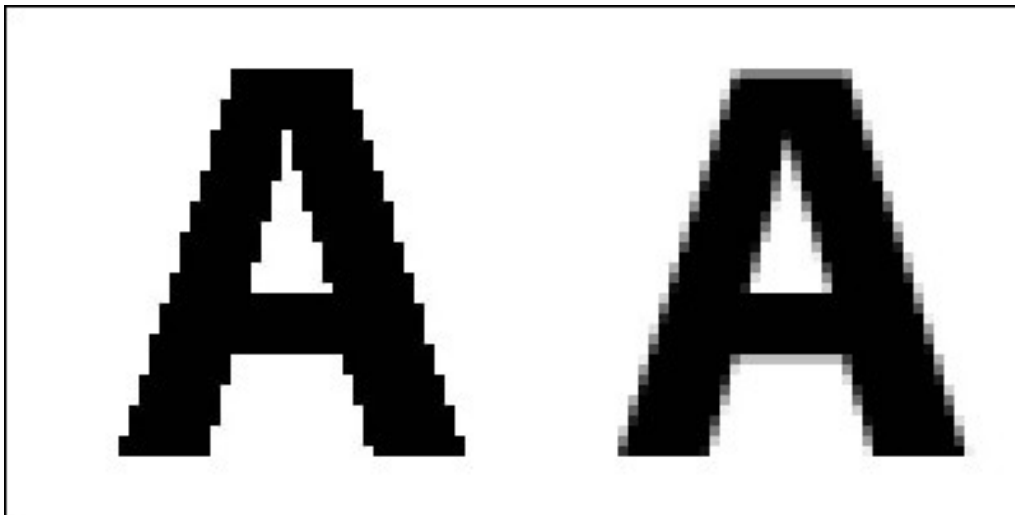
```
public void paint (Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;

    g2.setStroke(new BasicStroke(10.0f));
    g2.setColor(Color.red);
    g2.drawLine(20, 70, 350, 180);
}
```

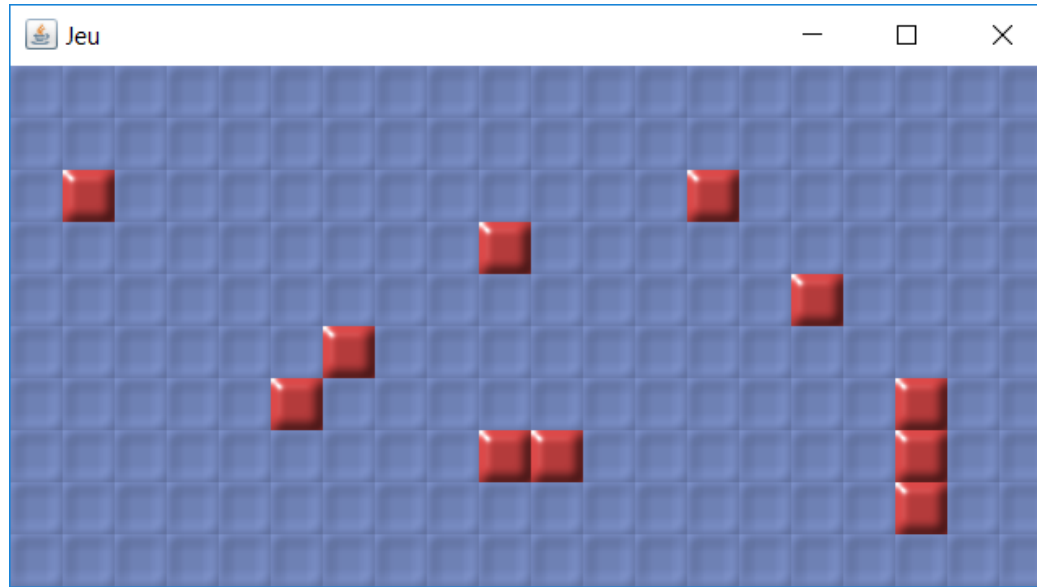


Vous pouvez constater que les tracés de primitives (lignes, cercles) et de texte souffrent de défauts graphiques de « marches d'escalier » (« *aliasing* »). Vous pouvez corriger ce défaut en activant l'« *antialiasing* » avec :

```
g2.setRenderingHint( RenderingHints.KEY_ANTIALIASING,  
                    RenderingHints.VALUE_ANTIALIAS_ON);
```



Exemple : sélectionner à la souris les cases d'une grille



// Fichier Jeu.java

```
class Jeu
{
    public static void main(String[] args)
    {
        Fenetre f = new Fenetre("Jeu");
    }
}
```

```
// Fichier Fenetre.java
```

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

```
class Fenetre extends JFrame
```

```
{  
    ZoneDessin zoneDessin;  
  
    Fenetre(String s)  
    {  
        super(s);  
        setSize(640,320);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        zoneDessin = new ZoneDessin();  
        setContentPane(zoneDessin);  
        pack();  
        setVisible(true);  
    }  
}
```

```
// Fichier ZoneDessin.java
```

```
import java.awt.*;
```

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

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

```
import java.io.File;
```

```
import javax.imageio.ImageIO;
```

```
import java.io.IOException;
```

```
class ZoneDessin extends JPanel implements MouseListener
```

```
{
```

```
    Image      bloc_bleu, bloc_rouge;
```

```
    final int dimx = 20, dimy = 10;
```

```
    final int taillex = 32, tailley = 32;
```

```
    int[][]      tableau = new int[dimx][dimy];
```

```

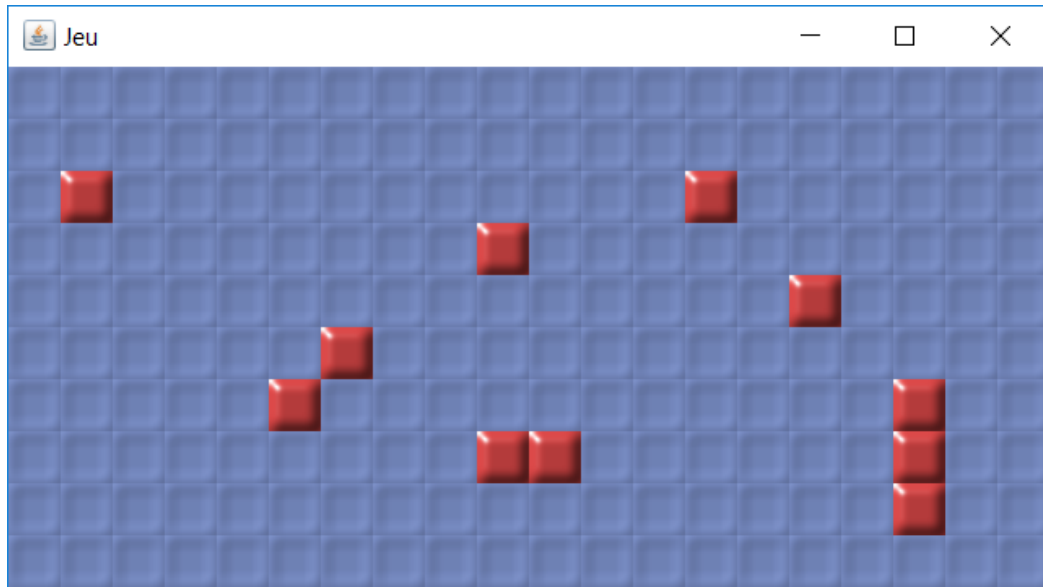
ZoneDessin()
{
    try {
        bloc_bleu = ImageIO.read(new File("bloc_bleu.gif"));
        bloc_rouge = ImageIO.read(new File("bloc_rouge.gif"));
    } catch (IOException e) {
        e.printStackTrace();
    }
    setPreferredSize(new Dimension(dimx*taillex, dimy*tailley));
    addMouseListener(this);
}

public void modifieTableau(int x, int y, int val)
{
    int xt = x/taillex;
    int yt = y/tailley;

    if( xt>=0 && xt<dimx && yt>=0 && yt<dimy )
    {
        tableau[xt][yt] = val;
        repaint();
    }
}

```

```
public void paintComponent(Graphics g)
{
    for( int y=0; y<dimy; y++ )
    {
        for( int x=0; x<dimx; x++ )
        {
            if( tableau[x][y] == 0 )
                g.drawImage(bloc_bleu, x*taille, y*taille, this);
            else if( tableau[x][y] == 1 )
                g.drawImage(bloc_rouge, x*taille, y*taille, this);
        }
    }
}
```



```
public void mousePressed(MouseEvent e)
{
    int bouton = e.getButton();

    if (bouton == MouseEvent.BUTTON1)
        modifieTableau(e.getX(), e.getY(), 1);
    else if (bouton == MouseEvent.BUTTON3)
        modifieTableau(e.getX(), e.getY(), 0);
}

public void mouseReleased(MouseEvent e) {}
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
}
```


Animation

Swing ne dispose pas de fonctionnalités permettant de faire de l'animation, mais on peut y parvenir au moyen de **timer**, permettant d'effectuer une action à intervalle de temps fixe.

<http://docs.oracle.com/javase/8/docs/api/java/util/Timer.html>

<http://docs.oracle.com/javase/8/docs/api/java/util/TimerTask.html>

<http://www.java2s.com/Code/Java/Advanced-Graphics/Animation.htm>

```

import java.util.Timer;
import java.util.TimerTask;

class Horloge
{
    Timer timer ;

    public Horloge()
    {
        TimerTask task = new TimerTask()
        {
            public void run()
            {
                fenetre.repaint();
            }
        };

        timer = new Timer();
        // Appele la méthode run() de task toutes les 1000ms (donc toutes les sec.)
        timer.scheduleAtFixedRate(task, 0, 1000);
    }
}

```