

# Les interfaces graphiques (Swing)

## Introduction

---

D'une façon générale, une interface utilisateur graphique (GUI) est composée d'éléments graphiques de base comme des boutons, des étiquettes (labels), des champs de texte, des cases à cocher (check box), des ascenseurs, des menus déroulants, etc. Ces éléments de base sont appelés « Composants » ou « Contrôles ».

Dès la version Java 1.0, SUN a introduit une bibliothèque de classes nommée **AWT** (Abstract Window Toolkit) pour la programmation de base de l'interface graphique utilisateur (*Graphical User Interface*).

L'inconvénient majeur de la bibliothèque AWT est qu'elle n'est pas 100% Java. En effet, AWT s'appuie entièrement sur le gestionnaire graphique du système d'exploitation pour représenter les composants à l'écran ; ce qui fait que d'un système d'exploitation à un autre, l'interface graphique AWT peut changer énormément au niveau de la taille et de la disposition des composants.

Depuis Java SE 1.2, Sun a intégré une nouvelle bibliothèque d'interfaces utilisateur portant le nom de code **SWING**.

## I. Structure d'une interface graphique (GUI)

---

Il existe deux principaux types de composants susceptibles d'intervenir dans une interface graphique :

- Les conteneurs qui sont destinés à contenir d'autres composants, comme par exemple les fenêtres (JFrame, JWindow), les boîtes de dialogue (JDialog, JOptionPane), les panneaux (JPanel), etc. ;
- Les composants atomiques comme les boutons (JButton), les champs de texte (JTextField), les cases à cocher (JCheckBox), les boutons radio (JRadioButton), etc.

Pour pouvoir utiliser ces composants, nous devons importer leurs classes des packages spécifiques **javax.swing** et **java.awt**.

## II. Création d'une fenêtre

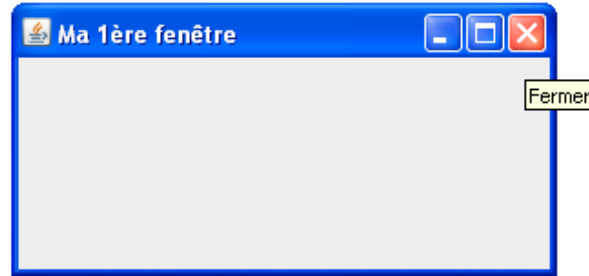
---

Une interface graphique swing est un arbre qui part d'un objet système lourd (fenêtre, boîte de dialogue, applet, ...).

Le programme suivant montre comment créer une fenêtre (**JFrame**) sous Java :

```
import javax.swing.JFrame;
public class Fenetre {
    public static void main(String[] args) {
        JFrame maFenetre = new JFrame();
        maFenetre.setTitle("Ma lère fenêtre");
        maFenetre.setSize(300, 150);
        maFenetre.setLocationRelativeTo(null);
        maFenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        maFenetre.setVisible(true);
    }
}
```

## Output



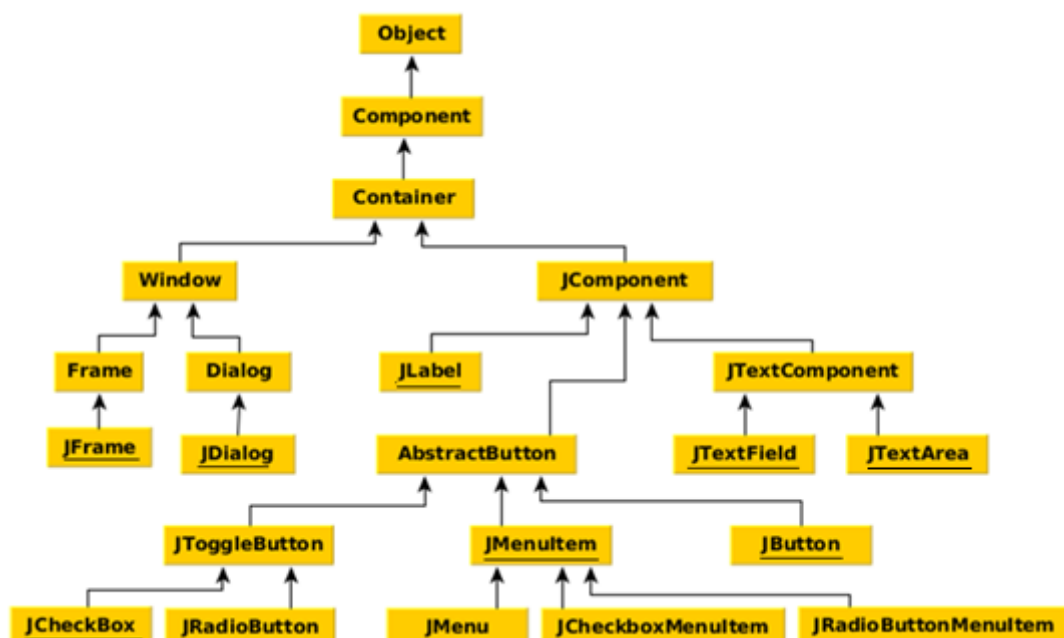
## Remarques

- Par défaut, une fenêtre n'a pas de titre. La méthode **setTitle(String)** permet d'ajouter un titre à notre fenêtre dans la barre de titre.

On aurait pu écrire directement : `JFrame maFenetre = new JFrame("Ma 1ère fenêtre");`

- Par défaut, une fenêtre a une taille de  $0 \times 0$  pixel. La méthode **setSize(largeur, hauteur)** permet d'affecter une taille à notre fenêtre ( $300 \times 150$  pixels dans l'exemple).
- L'instruction **maFenetre.setLocationRelativeTo(null)** permet d'afficher la fenêtre au milieu de l'écran (par défaut la fenêtre est affichée au coin supérieur gauche de l'écran).
- Par défaut, l'appui sur le bouton de fermeture de la fenêtre permet de la cacher (hide). Les options possibles sont :
  - DISPOSE\_ON\_CLOSE** : fermer la fenêtre.
  - EXIT\_ON\_CLOSE** : fermer la fenêtre et arrêter le programme (le processus).
  - HIDE\_ON\_CLOSE** : cacher la fenêtre (option par défaut).
  - DO\_NOTHING\_ON\_CLOSE** : ne rien faire.
- Par défaut, une fenêtre est invisible. La méthode **setVisible(true/false)** d'afficher ou masquer la fenêtre. Elle doit être utilisée après le remplissage de la fenêtre par les différents composants graphiques.

## III. Hiérarchie des classes



Les classes dont le nom commence par « J » comme JFrame, JLabel, JTextField, JTextArea, JButton, JCheckBox, JRadioButton, JMenu, ... font partie du package **javax.swing** alors que les autres font partie de package **java.awt**.

### III.1. La classe JFrame

---

Un objet JFrame représente une fenêtre principale qui possède un titre, une taille modifiable et éventuellement un menu.

La classe JFrame possède plusieurs constructeurs :

- JFrame() : Création d'un fenêtre invisible et sans titre
- JFrame(String) : Création d'une fenetre invisible avec un titre.

La classe JFrame propose différentes méthodes pour la personnalisation d'une fenêtre :

Méthode	Rôle
setTitle(titre)	définir le titre de la fenêtre
getTitle()	récupérer le titre de la fenêtre
setIconImage(image)	ajouter une icône dans la barre de titre
setSize(largeur, hauteur)	définir la taille de la fenêtre
setLocation (int x, int y)	définir la position de la fenêtre
setBounds(int x, int y, int w, int h)	définir à la fois la position et la taille de la fenêtre
setResizable(booléen)	autoriser ou interdire le redimensionnement de la fenêtre.
toBack()/toFront()	placer la fenêtre derrière/devant les autres.
dispose()	fermer la fenêtre.
getContentPane()	fournir une référence, de type « Container », au conteneur associé à la fenêtre. Si on ne l'a pas modifié, ce contenant est de type JPanel.
setContentPane()	redéfinir le conteneur associé à la fenêtre.
pack()	remettre en page l'ensemble des composants de la fenêtre (donne à chacun sa taille préférentielle).
setVisible(booléen)	afficher/masquer la fenêtre.

### III.2. La classe JPanel

---

Un panel (panneau) est essentiellement un objet de rangement pour d'autres composants. La classe JPanel possède plusieurs constructeurs parmi eux :

- JPanel() : panel avec un gestionnaire de placement par défaut (FlowLayout)
- JPanel(LayoutManager l) : panel avec le gestionnaire de placement l.

Un panel seul ne peut pas être affiché, il doit être inséré dans un composant lourd (JFrame, JDialog, JWindow, etc.).

Pour ajouter des composants à un panel p, on utilise la syntaxe suivante :

```
JPanel p = new JPanel() ;    // création du panel
p.add(composant1);
.   p.add(composant2);
...
```

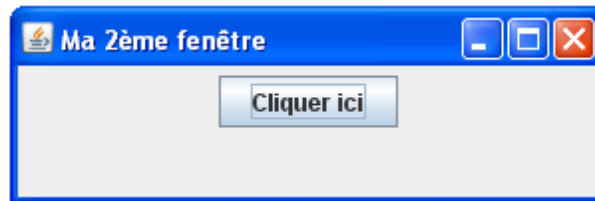
Par la suite, il faut rattacher ce panel à la fenêtre dans laquelle il s'affichera. Deux solutions sont possibles :

```
fenetre.setContentPane(p) ;
ou
fenetre.getContentPane().add(p) ;    // une fenêtre peut contenir plusieurs panels.
```

### Exemple 1 : Ajout d'un bouton à une fenêtre

```
import javax.swing.*;
public class Fenetre extends JFrame {
    public Fenetre() {
        this.setTitle("Ma 2ème fenêtre");
        this.setSize(300, 100);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel pano = new JPanel();
        JButton monBouton = new JButton("Cliquer ici");
        pano.add(monBouton);
        this.setContentPane(pano);
        this.setVisible(true);
    }
    public static void main(String[] args) {
        new Fenetre();
    }
}
```

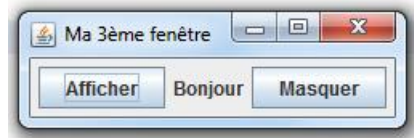
### Output



Exemple 2 : Le programme suivant permet de créer une fenêtre contenant : deux boutons et un label au milieu qui contient le texte « Bonjour ».

```
import javax.swing.*;
public class Fenetre extends ..... {
    ..... show, hide;
    JLabel message;
    public Fenetre () {
        setTitle("Ma 3ème fenêtre");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        show = new JButton("Afficher");
        message = new ..... ("Bonjour");
        hide = new JButton("Masquer");
        JPanel pano = new ..... ();
        pano.add(show);
        pano.add(message);
        pano.add(hide);
        this. .... (pano);
        pack(); // Ajuster la taille des composants
        this. .... (true);
    }
    public static void main(String[] args) {
        ..... Fenetre();
    }
}
```

### Output



**Remarque :** Dans le programme précédent, un clic sur les boutons n'a aucun effet, pour activer ces boutons de façon qu'ils affichent ou masquent respectivement le message « Bonjour », il faut associer un écouteur (**ActionListener**) à chaque bouton et développer une méthode « actionPerformed » qui gère les événements appropriés.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class FenetreAvecEcouteur extends JFrame implements
ActionListener {
    JButton show, hide;
    JLabel message;
    public FenetreAvecEcouteur() {
        setTitle("Ma 4ème fenêtre");
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        show = new JButton("Afficher");
        show.addActionListener(this);    // ajout d'un écouteur
        message = new JLabel("Bonjour");
        hide = new JButton("Masquer");
        hide.addActionListener(this);    // ajout d'un écouteur
        JPanel pano = new JPanel();
        pano.add(show);
        pano.add(message);
        pano.add(hide);
        setContentPane(pano);
        pack();
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == show){
            message.setVisible(true);
        } else if(e.getSource() == hide){
            message.setVisible(false);
        }
    }

    public static void main(String[] args){
        new FenetreAvecEcouteur();
    }
}
```

### III.3. La classe JLabel (étiquette)

Un label affiche une seule ligne de texte (étiquette) non modifiable. En général, les étiquettes ne traitent pas d'événements.

#### *Principales méthodes*

Méthode	Rôle
void setText(String)	Définir le texte du label
String getText()	Récupérer le texte du label

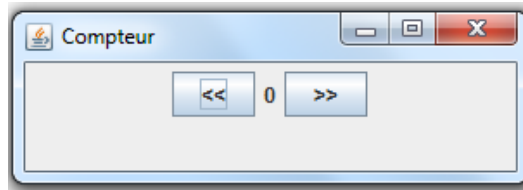
<code>void setFont(Font)</code>	Changer la mise en forme du texte du label
<code>void setVisible(true/false)</code>	Afficher/Masquer le label

### Exemple

```
JLabel l = new JLabel("Menu Général");
l.setFont(new Font("TimesRoman", Font.BOLD, 14)) ;
```

### Exercice (Compteur)

Ecrire un programme en Java qui affiche un compteur initialisé à zéro et dont la valeur s'incrémente après chaque clic sur le bouton *plus* (>>) et se décrémente après chaque clic sur le boutons *moins* (<<).



### III.4. La classe JTextField (champ de texte)

Le champ de texte est un dispositif d'entrée de texte sur une seule ligne. L'appui sur la touche « Enter » est notifié par un événement de type **ActionEvent** envoyé à un écouteur de type **ActionListener**.

On peut définir un champ de texte comme étant éditable ou non.

#### *Principales méthodes*

Méthode	Rôle
<code>void setText(String)</code>	Mettre un texte dans le champ
<code>String getText()</code>	Récupérer le texte du champ
<code>void setEditable(boolean)</code>	Rendre le texte éditable ou non éditable
<code>void setFont(Font)</code>	Changer la mise en forme du texte

### Exemple

```
JTextField t = new JTextField("Texte non modifiable");
t.setEditable(false) ;
t.setFont(new Font("Arial", Font.PLAIN, 18)) ; // texte simple
```

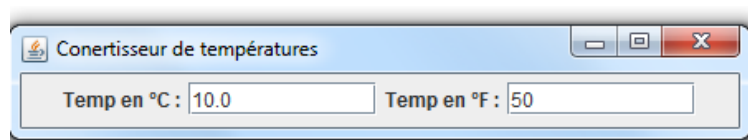
### Exercice (conversion de températures)

Ecrire un programme qui convertit une température en °C vers le Fahrenheit ou l'inverse en utilisant la formule :

$$F = (9/5) C + 32.$$

L'opération de conversion sera déclenchée par un appui sur la touche « enter » dans l'un des champs de texte.

### Exemple



### III.5. La classe JTextArea (zone de texte)

La zone de texte est un dispositif d'entrée de texte multi-lignes, multi-colonnes avec éventuellement la présence de barres de défilement (scrollBars) horizontale et/ou verticale. Elle peut être éditable ou non.

#### Principales méthodes

Méthode	Rôle
void setText(String)	Mettre un texte dans la zone de texte
Void append(String)	Ajouter un texte
String getText()	Récupérer le texte de la zone de texte
void setEditable(boolean)	Rendre le texte éditable ou non éditable
void setFont(Font)	Changer la mise en forme du texte

#### Exemple

```
JTextArea a = new JTextArea("Première ligne");
a.append("\nDeuxième ligne");
String s = a.getText();
```

### III.6. Volets de défilement (scroll bars)

Par défaut, une zone de texte (JTextArea) ne dispose pas de barres de défilement. Si on souhaite en ajouter, il faut mettre la zone de texte dans un panneau avec barres de défilement (JScrollPane).

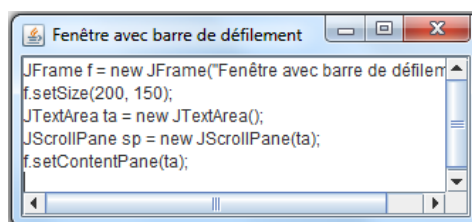
Les barres de défilement apparaissent automatiquement lorsque le contenu du composant devient plus grand que la vue. Elles disparaissent si, suite à une suppression, le texte restant tient dans la zone de texte.

#### Exemple

```
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class BarreDéfilement {
    public static void main(String[] args){
        JFrame f = new JFrame("Fenêtre avec barre de défilement");
        f.setSize(200, 150);
        JTextArea ta = new JTextArea();
        JScrollPane sp = new JScrollPane(ta);
        f.setContentPane(sp);
        f.setLocationRelativeTo(null);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

Output



### III.7. La classe JCheckBox (case à cocher)

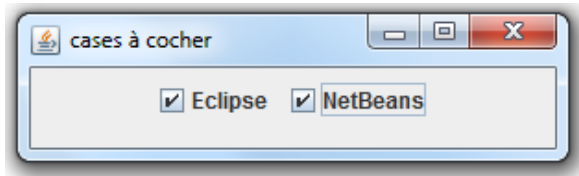
La case à cocher fournit un dispositif d'entrée « actif/inactif » accompagné d'une étiquette de texte. La sélection ou la désélection est notifiée par un **ItemEvent** envoyé à un écouteur de type **ItemListener**.

#### Principales méthodes

Méthode	Rôle
boolean isSelected()	Retourner true lorsque la case est sélectionnée
Boolean setSelected(boolean)	Mettre la case dans l'état « sélectionnée/désélectionnée »
void setText(String)	Mettre un texte dans l'étiquette
String getText()	Renvoyer le texte de l'étiquette

#### Exemple

```
JCheckBox one = new JCheckBox("Eclipse",true) ;
JCheckBox two = new JCheckBox("NetBeans") ;
two.setSelected(true) ;
```



### III.8. La classe JRadioButton (bouton Radio)

Un bouton radio de type JRadioButton permet à l'utilisateur d'effectuer un choix de type oui/non. Mais sa vocation est de faire partie d'un groupe de boutons dans lequel une seule option peut être sélectionnée à la fois.

#### Constructeurs

- `JRadioButton(String label)` : Construit un bouton radio, non sélectionné au départ.
- `JRadioButton(String label, boolean state)` : Construit un bouton radio avec le libellé et l'état initial donnés.

Par défaut, un bouton radio est construit dans l'état non sélectionné (false). On peut utiliser les méthodes **isSelected()** et **setSelected()** de la classe **AbstractButton** pour tester ou modifier l'état d'un bouton radio.

Les boutons radio doivent être mis en groupe (**ButtonGroup**)

- `void add(AbstractButton b)` : Ajoute le bouton au groupe.
- `ButtonModel getSelection()` : Renvoie le modèle du bouton sélectionné.



Un objet de type **ButtonGroup** sert uniquement à assurer la désactivation automatique des autres boutons lorsqu'un bouton du groupe est activé. Un bouton radio qui n'est pas associé à un groupe, exception faite de son aspect, se comporte exactement comme une case à cocher.



## Syntaxe :

```
ButtonGroup nomGroupe = new ButtonGroup() ;  
nomGroupe.add(bouton1) ;  
...
```

## Exemple

```
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.*;  
  
class FeuxSignalisation extends JFrame implements ActionListener {  
  
    private JRadioButton bRouge;  
    private JRadioButton bVert;  
  
    public FeuxSignalisation () {  
        super("fenetre avec boutons radio") ;  
        bRouge = new JRadioButton("Rouge") ;  
        bRouge.setBackground(Color.red);  
        bRouge.addActionListener(this);  
        bVert = new JRadioButton("Vert", true) ;  
        bVert.setBackground(Color.green);  
        bVert.addActionListener(this);  
        ButtonGroup groupe = new ButtonGroup() ;  
        groupe.add(bRouge) ;  
        groupe.add(bVert) ;  
        JPanel monPanel = new JPanel();  
        monPanel.add(bRouge) ;  
        monPanel.add(bVert) ;  
        setContentPane(monPanel);  
        setLocationRelativeTo(null);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        pack();  
        this.setVisible(true);  
    }  
  
    public void actionPerformed(ActionEvent e){  
        if (e.getSource() == bRouge)  
            JOptionPane.showMessageDialog(null,"Attendez");  
        else if (e.getSource() == bVert)  
            JOptionPane.showMessageDialog(null,"Passez");  
    }  
    public static void main(String[] args) {  
        new FeuxSignalisation ();  
    }  
}
```

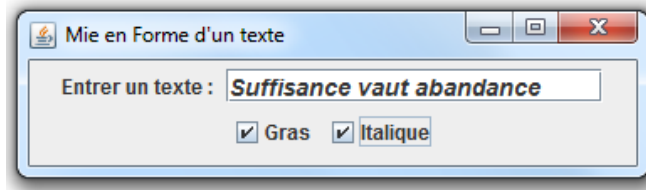
## Output



### Exercice 1 (mise en forme d'un texte)

Ecrire un programme qui fait la mise en forme d'un texte selon le style choisi par l'utilisateur (simple, **gras**, *italique*) à partir de deux cases à cocher.

#### Exemple



Remarque : la police utilisée pour afficher du texte dans un composant peut être définie avec la méthode `setFont(Font)` qui prend comme argument une instance de `java.awt.Font` comme dans l'exemple suivant :

```
Font f = new Font("Arial", Font.PLAIN, 16) ;  
monTexte.setFont(f) ;
```

Il est également possible d'utiliser les styles `Font.BOLD` et `Font.ITALIC`.

#### Solution

```
import javax.swing.*.*;  
import javax.swing.event.ChangeListener;  
import java.awt.*.*;  
import java.awt.event.*.*;  
  
public class MiseEnForme ..... JFrame ..... ItemListener {  
    private JLabel etiquette;  
    private JTextField texte;  
    private ..... gras, italique;  
  
    public MiseEnForme() {  
        JPanel pan = new JPanel();  
        etiquette = new JLabel("Entrer un texte : ");  
        texte = new JTextField(20);  
        gras = new JCheckBox("Gras");  
        gras. .... (this);  
        italique = new JCheckBox("Italique");  
        italique.addItemListener(this);  
        pan.add(etiquette);  
        pan.add(texte);  
        pan.add(gras);  
        pan.add(italique);  
        this. .... (pan);  
        this.setTitle("Mie en Forme d'un texte");  
        this.setSize(340,100);  
        this.setLocationRelativeTo(null);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setVisible(true);  
    }  
  
    public void itemStateChanged(ItemEvent e) {  
        int valGras = Font.PLAIN;  
        int valItalique = Font.PLAIN;
```

```

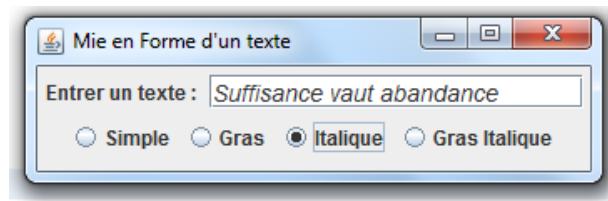
        if(gras. .... ())
            valGras = Font.BOLD;
        if (italique. .... ())
            valItalique = Font.ITALIC;
        texte.setFont(new Font("TimesRoman", valGras+valItalique,14));
    }

    public static void main(String[] args){
        new MiseEnForme();
    }
}

```

## Exercice 2

Réécrire le programme mise en forme d'un texte en utilisant 4 boutons radio rassemblés dans un même groupe (Simple / Gras / Italique / Gras Italique) à la place des cases à cocher.



```

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

public class BoutonsRadio extends JFrame implements ItemListener {
    private JLabel etiquette;
    private JTextField texte;
    private JRadioButton simple;
    private JRadioButton gras;
    private JRadioButton italique;
    private JRadioButton gras_italique;
    private ..... bg;

    public BoutonsRadio() {
        JPanel pan = new JPanel();
        etiquette = new JLabel("Entrer un texte : ");
        texte = new JTextField(20);
        gras = new JRadioButton("Gras");
        gras.addItemListener(this);
        italique = new JRadioButton("Italique");
        italique.addItemListener(this);
        gras_italique = new JRadioButton("Gras Italique");
        gras_italique.addItemListener(this);
        simple = new JRadioButton("Simple");
        simple.addItemListener(this);
        pan.add(etiquette);
        pan.add(texte);
        bg = new ButtonGroup();
        bg. .... (simple);
        bg.add(gras);
        bg.add(italique);
        bg.add(gras_italique);
        pan.add(normal);
        pan.add(gras);
        pan.add(italique);
        pan.add(gras_italique);
    }
}

```

```

        this.setContentPane(pan);
        this.setTitle("Mie en Forme d'un texte");
        this.setSize(340,100);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setVisible(true);
    }
    public void ..... (ItemEvent e){
        int valGras = Font.PLAIN, valItalique = Font.PLAIN;
        if (simple. .... ()) {
            valGras = Font.PLAIN;
            valItalique = Font.PLAIN;
        }
        if(gras.isSelected())
            valGras = Font.BOLD;
        if (italique.isSelected())
            valItalique = Font.ITALIC;
        if (gras_italique.isSelected()){
            valGras = Font.BOLD;
            valItalique = Font.ITALIC;
        }
        texte.setFont(new Font("TimesRoman",valGras+valItalique,14));
    }
    public static void main(String[] args){
        new BoutonsRadio();
    }
}

```

### III.9 Les Champs Mot de Passe : JPasswordField

Un champ de mot de passe (JPasswordField) est un type spécial de champ de texte dont l’affichage sera crypté.

#### Principales méthodes

- JPasswordField(int taille) : Construit un nouveau champ de mot de passe.
- void setEchoChar(char echo) : Définit le caractère d’écho pour le champ de mot de passe (par défaut «●»). La valeur 0 rétablit le caractère d’écho utilisé par défaut.
- char[] getPassword() : Renvoie le texte contenu dans le champ de mot de passe. Pour renforcer la sécurité, on doit écraser le contenu du tableau renvoyé après utilisation. Le mot de passe n’est pas retourné en tant que String, car une chaîne resterait dans la machine virtuelle jusqu’à ce qu’elle soit éliminée par le processus de nettoyage de mémoire (le ramasse-miettes ou garbage collector).

#### Exemple

Créer une fenêtre d’authentification qui possède l’aspect suivant :



Si le mot de passe entré est correct (par exemple « java »), le programme doit fermer la fenêtre actuelle et ouvrir la fenêtre de l'exercice précédent (BoutonsRadio); sinon, un message d'erreur doit être affiché.

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

class Authentification extends JFrame implements ActionListener {
    private JLabel login;
    private JTextField user;
    private JLabel password;
    private JPasswordField pwd;
    private JButton seConnecter;

    public Authentification () {
        setTitle("fenêtre d'authentification") ;
        setSize(250,130);
        login = new JLabel("Utilisateur :");
        user = new JTextField(10);
        password = new JLabel("Mot de passe :");
        pwd = new JPasswordField(10);
        seConnecter = new JButton("Se connecter");
        seConnecter.addActionListener(this);
        JPanel pano = new JPanel();
        pano.add(login); pano.add(user);
        pano.add(password); pano.add(pwd);
        pano.add(seConnecter);
        this.setContentPane(pano);
        this.setLocationRelativeTo(this);
        this.setVisible(true);
    }

    public void actionPerformed(ActionEvent e){
        String s = new String(pwd.getPassword());
        if (s.equals("java")){
            this.dispose(); // fermeture de la fenêtre
            new BoutonsRadio();
        }
        else
            JOptionPane.showMessageDialog(null, "erreur d'authentification");
    }

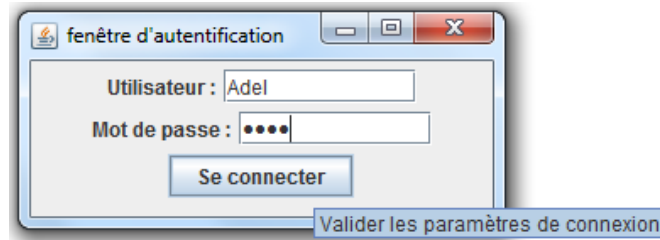
    public static void main(String[] args) {
        new Authentification();
    }
}
```

### III.10 Ajout d'une bulle d'aide (info bulle ou ToolTipText)

Pour créer une bulle d'aide qui apparaît lorsque la souris passe sur un bouton ou n'importe quel autre composant (gain de focus), il faut procéder comme suit :

```
JButton seConnecter = new JButton("Se Connecter") ;
seConnecter.setToolTipText("Valider les paramètres de connexion") ;
```

#### *Trace d'exécution*



### III.11 Les listes déroulantes (JComboBox)

La classe JComboBox permet de créer une liste dans laquelle on peut sélectionner un seul élément.

#### Constructeurs

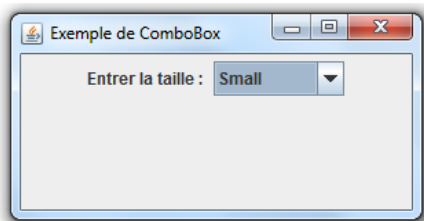
```
String[] tailles = new String[] {"Small", "Medium", "Large", "eXtra Large"};
JComboBox<String> listeTailles = new JComboBox(tailles);
```

#### Principales methods

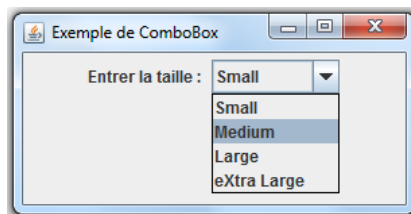
- addItem(Object item) : ajouter un objet à la liste
- removeItem(Object item) : enlever un objet de la liste
- removeAllItems() : vider la liste déroulante
- getSelectedItem() : retourne l'objet qui est actuellement sélectionné.
- getSelectedIndex() : retourne l'index de l'objet sélectionné (les indices commencent à partir de 0).

#### Exemple

Avant sélection



Après sélection



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

public class Combo extends JFrame implements ActionListener {
    private JLabel etiquette;
    private JComboBox<String> listeTailles;
    public Combo() {
        ..... tailles = {"Small", "Medium", "Large", "eXtra Large"};
        listeTailles = new ..... (tailles) ;
        listeTailles. .... (this);
        etiquette = new JLabel("Entrer la taille : ");
        JPanel p = new JPanel();
        p.add(etiquette);
        p.add(listeTailles);
        this.setContentPane(p);
        this.setTitle("Exemple de ComboBox");
        this.setSize(300,150);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(this);
        this.setVisible(true);
    }
}
```

```

public void actionPerformed(ActionEvent e){
    System.out.println(listeTailles.getSelectedIndex() + " . " +
        listeTailles.getSelectedItem());
}

public static void main(String[] args) {
    new Combo();
}
}

```

Exercice : Ecrire un programme Java qui simule un calculateur simple avec 4 opérateurs (+, -, \*, /).



Votre interface graphique est composée des éléments suivants :

- Une zone de texte (JTextField) que vous appelez « nombre1 »
- Une liste de type JComboBox que vous appelez « listeOperateurs » et qui contient 4 options (+, -, \*, /). Il est conseillé de commencer par mettre ces opérateurs dans un tableau de String.
- Une zone de texte (JTextField) que vous appelez « nombre2 »
- 3 boutons que vous appelez respectivement « calculer », « nettoyer » et « quitter »
- Un label que vous appelez « resultat »

### III.12. Les listes de choix (JList)

La classe JList permet d'afficher une liste d'objets, et offre à l'utilisateur la possibilité de sélectionner un ou plusieurs éléments de la liste.

Un JList peut être créé à partir d'un tableau d'objets ou à partir d'un Vector, mais si on veut une liste qui peut être mise à jour par programme, il faut utiliser un *ListModel*.

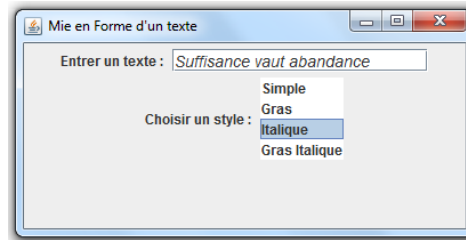
#### Constructeurs

JList()	Crée un <i>JList</i> vide.
JList(Object[] donnees)	Crée un <i>JList</i> qui affiche les données contenues dans le tableau <i>donnees</i> .
JList(Vector<?> v)	Crée un <i>JList</i> qui affiche les données contenues dans le vecteur <i>v</i> .
JList (ListModel l)	Crée un <i>JList</i> qui affiche les données contenues dans le <i>ListModel l</i> .

#### Principales méthodes

void clearSelection()	Plus aucun item sélectionné.
boolean isSelectionEmpty()	Retourne true s'il n'y a aucun item sélectionné.
int getSelectedIndex()	Retourne l'indice de l'item sélectionné.
Object getSelectedValue()	Retourne la valeur de l'item sélectionné.
Object[] getSelectedValues()	Retourne un tableau des items sélectionnés.
void setSelectedValue(Object v, boolean b)	L'item sélectionné devient v. Si b vaut true, il est rendu visible dans le JList.

Exemple : Ecrire un programme qui fait la mise en forme d'un texte selon le choix effectué par l'utilisateur (Gras, italique, ...) à partir d'une liste de choix :



## Solution

```
import javax.swing.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.awt.*;
import java.util.Vector;

public class MEFJList extends ..... ListSelectionListener {
    private JLabel etiquettel, etiquette2;
    private JTextField texte;
    private ..... listeStyles;
    public MEFJList() {
        etiquettel = new JLabel("Entrer un texte : ");
        etiquette2 = new JLabel("Choisir un style : ");
        texte = new JTextField(20);
        .....<String> listeItems = new Vector<>();
        listeItems.add("Simple"); listeItems.add("Gras");
        listeItems.add("Italique"); listeItems.add("Gras Italique");
        listeStyles = new ..... (listeItems);
        listeStyles. .... (this);
        JPanel pano = new JPanel();
        pano.add(etiquettel); pano.add(texte); pano.add(etiquette2);
        pano.add(listeStyles);
        this.getContentPane().add(pano);
        this.setTitle("Mie en Forme d'un texte");
        this.setSize(340,200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(this);
        this.setVisible(true);
    }

    public void ..... (ListSelectionEvent e ) {
        int valGras = Font.PLAIN;
        int valItalique = Font.PLAIN;
        if (listeStyles. .... () == 1)
            valGras = Font.BOLD;
        if (listeStyles. .... () == 2)
            valItalique = Font.ITALIC;
        if (listeStyles. .... () == 3) {
            valGras = Font.BOLD;
            valItalique = Font.ITALIC;
        }
        texte. .... (new Font("TimesRoman",valGras+valItalique, 14));
    }

    public static void main(String[] args) {
        new MEFJList();
    }
}
```

## III.13. Contrôle des couleurs d'un composant



Deux méthodes permettent de contrôler les couleurs d'un composant :

- `setForeground(Color c)` : définit la couleur de l'encre avec laquelle on écrira sur le composant.
- `setBackground(Color c)` : définit la couleur de fond.

Ces deux méthodes utilisent un argument instance de la classe `java.awt.Color`.

Il est également possible de créer une couleur spécifique (RGB).

### Exemple

```
texte = new JTextField(20);
texte.setForeground(Color.GRAY);
texte.setBackground(new Color(0,255,255)); //cyan
```

## III.14. La classe JTable

---

Le composant `JTable` permet d'afficher des tables de données, en permettant éventuellement l'édition de ces données. Un `JTable` ne contient pas ses données mais les obtient à partir d'un tableau d'objets à 2 dimensions ou à partir d'un modèle de données.

### Exemple

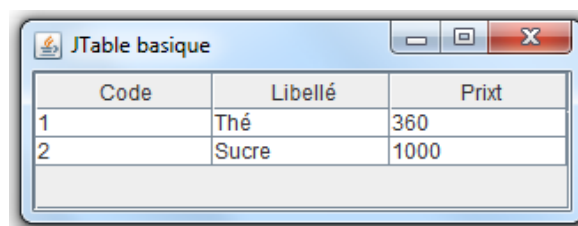
```
import java.awt.BorderLayout;
import javax.swing.*;

public class ExpleJTable extends JFrame {

    public ExpleJTable() {
        String[] entetes = {"Code", "Libellé", "Prix"};
        Object[][] donnees = {{1, "Thé", 360}, {2, "Sucre", 1000}};
        JTable tableau = new JTable(donnees, entetes);
        this.getContentPane().add(new JScrollPane(tableau));
        this.setTitle("JTable basique");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(this);
        this.pack();
    }

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

### Output



Code	Libellé	Prix
1	Thé	360
2	Sucre	1000

## IV. Catégories d'événements graphiques

Plusieurs types d'événements sont définis dans le package **java.awt.event**. Pour chaque catégorie d'événements, il existe une *interface* qui doit être implémentée par toute classe souhaitant recevoir cette catégorie d'événements. Par conséquent, la classe doit implémenter les méthodes définies dans cette interface. Ces méthodes définissent la réaction du programme lorsque certains événements particuliers se produisent.

Catégorie	Exemples d'événements	Nom de l'interface (écouteur)	Méthodes
Action	Clic sur un bouton, retour chariot dans une zone de texte, etc.	ActionListener	actionPerformed(ActionEvent)
Item	Changement d'état d'une case à cocher, d'un bouton radio, etc.	ItemListener	itemStateChanged(ItemEvent)
List	Sélection d'un élément dans un Jlist	ListSelectionListener	valueChanged(ListSelectionEvent)
Text	Changement de valeur dans une zone de texte	TextListener	textValueChanged(TextEvent)
Mouse	Déplacement de la souris, drag&drop, etc.	MouseMotionListener	mouseMoved(MouseEvent)
			mouseDragged(MouseEvent)
	Clic, enfoncement, relâchement des boutons de la souris, etc.	MouseListener	mousePressed(MouseEvent)
			mouseReleased(MouseEvent)
			mouseEntered(MouseEvent)
			mouseExited(MouseEvent)
Focus	Gain ou perte de focus par un élément	FocusListener	mouseClicked(MouseEvent)
			focusGained(FocusEvent)
Key	Pression d'une touche, relachement, etc.	KeyListener	focusLost(FocusEvent)
			keyPressed(KeyEvent)
			keyReleased(KeyEvent)
Window	Ouverture d'une fenêtre, fermeture, iconisation, etc.	WindowListener	keyTyped(KeyEvent)
			windowOpened(WindowEvent)
			windowClosed(WindowEvent)
			windowIconified(WindowEvent)
			windowDeiconified(WindowEvent)
			windowActivated(WindowEvent)
Container	Ajout, suppression d'un composant dans un conteneur	ContainerListener	windowDeactivated(WindowEvent)
			componentAdded(ContainerEvent)
Component	Affichage d'un composant, masquage, déplacement, etc.	ComponentListener	componentRemoved(ContainerEvent)
			componentMoved(ComponentEvent)
			componentHidden(ComponentEvent)
			componentShown(ComponentEvent)
Adjustment	Déplacement dans une zone de défilement	AdjustmentListener	componentResized(ComponentEvent)
			adjustmentValueChanged(adjustmentEvent)