

Interface graphique en Java

API swing

Institut National Agronomique Paris-Grignon

Juliette Dibie-Barthélemy
mai 2005



U.E.R. d'Informatique

Plan

| | | |
|------------|--------------------------------------------------------------|-----------|
| I. | INTERFACE GRAPHIQUE..... | 2 |
| I.1. | INTRODUCTION | 2 |
| I.2. | UN PREMIER EXEMPLE | 5 |
| I.3. | LES FENETRES GRAPHIQUES : LA CLASSE JFRAME..... | 7 |
| I.4. | DES METHODES UTILES DE LA CLASSE COMPONENT | 8 |
| I.5. | LES COMPOSANTS ATOMIQUES..... | 10 |
| I.5.1. | <i>Les cases à cocher</i> | 11 |
| I.5.2. | <i>Les boutons radio</i> | 12 |
| I.5.3. | <i>Les étiquettes</i> | 14 |
| I.5.4. | <i>Les champs de texte</i> | 15 |
| I.5.5. | <i>Les boîtes de liste</i> | 16 |
| I.5.6. | <i>Les boîtes de liste combinée</i> | 18 |
| I.6. | LES MENUS ET LES BARRES D'OUTILS | 20 |
| I.6.1. | <i>Les menus déroulants</i> | 21 |
| I.6.2. | <i>Les menus surgissants</i> | 27 |
| I.6.3. | <i>Les barres d'outils</i> | 28 |
| I.6.4. | <i>Les bulles d'aide</i> | 30 |
| I.7. | LES BOITES DE DIALOGUE | 31 |
| I.7.1. | <i>Les boîtes de message</i> | 32 |
| I.7.2. | <i>Les boîtes de confirmation</i> | 34 |
| I.7.3. | <i>Les boîtes de saisie</i> | 36 |
| I.7.4. | <i>Les boîtes d'options</i> | 38 |
| I.7.5. | <i>Les boîtes de dialogue personnalisées</i> | 40 |
| I.8. | LES GESTIONNAIRES DE MISE EN FORME | 42 |
| I.8.1. | <i>Le gestionnaire BorderLayout</i> | 43 |
| I.8.2. | <i>Le gestionnaire FlowLayout</i> | 45 |
| I.8.3. | <i>Le gestionnaire CardLayout</i> | 47 |
| I.8.4. | <i>Le gestionnaire GridLayout</i> | 48 |
| I.8.5. | <i>Un programme sans gestionnaire de mise en forme</i> | 50 |
| I.8.6. | <i>Une classe Insets pour gérer les marges</i> | 51 |
| I.9. | DESSINONS AVEC JAVA..... | 53 |
| I.9.1. | <i>Création d'un panneau</i> | 54 |
| I.9.2. | <i>Dessin dans un panneau</i> | 55 |
| I.9.3. | <i>La classe Graphics</i> | 57 |
| I.9.4. | <i>Affichage d'images</i> | 61 |
| II. | LA GESTION DES EVENEMENTS..... | 63 |
| II.1. | INTRODUCTION | 63 |
| II.2. | TRAITER UN EVENEMENT | 64 |
| II.3. | INTERCEPTER UN EVENEMENT | 65 |
| II.4. | UN PREMIER EXEMPLE | 66 |
| II.4.1. | <i>Première version</i> | 66 |
| II.4.2. | <i>Deuxième version</i> | 67 |
| II.5. | LA NOTION D'ADAPTATEUR | 68 |
| II.6. | RECAPITULONS | 69 |
| II.7. | UN EXEMPLE AVEC DES BOUTONS | 71 |
| II.8. | UN EXEMPLE DE CREATION DYNAMIQUE DE BOUTONS..... | 74 |
| II.9. | LES CLASSES INTERNES ET ANONYMES | 76 |
| II.9.1. | <i>Les classes internes</i> | 76 |
| II.9.2. | <i>Les classes anonymes</i> | 78 |
| | BIBLIOGRAPHIE..... | 79 |
| | INDEX..... | 80 |

I. Interface graphique

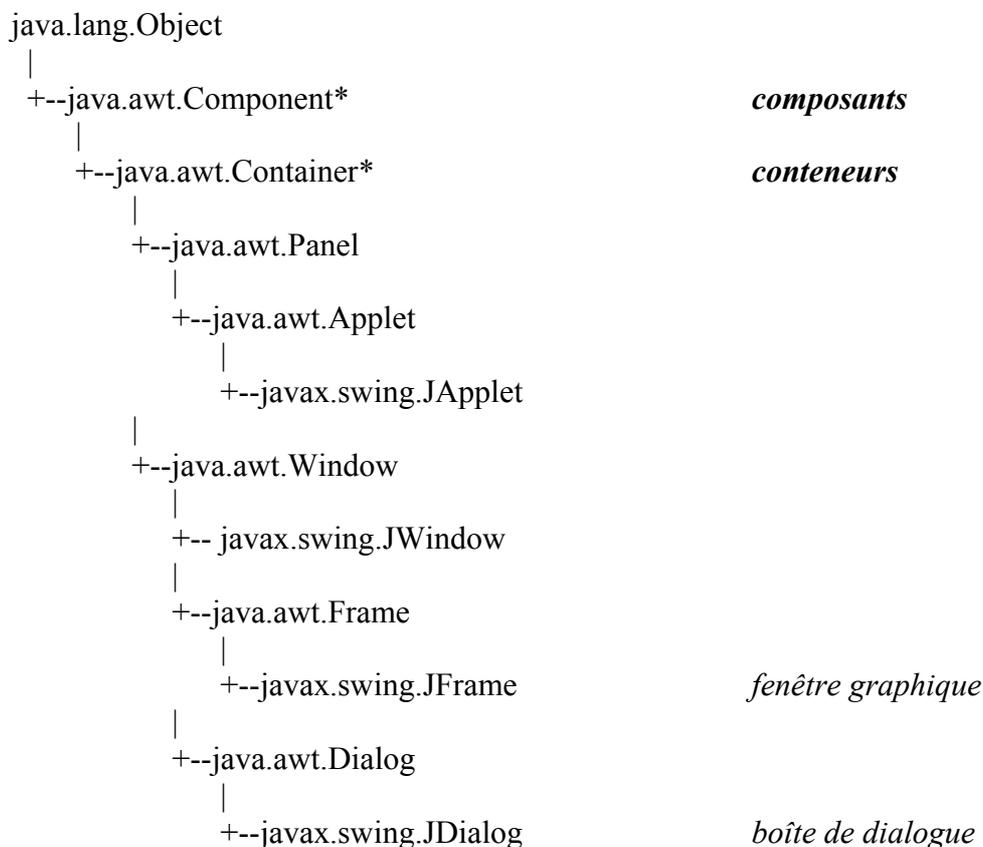
I.1. Introduction

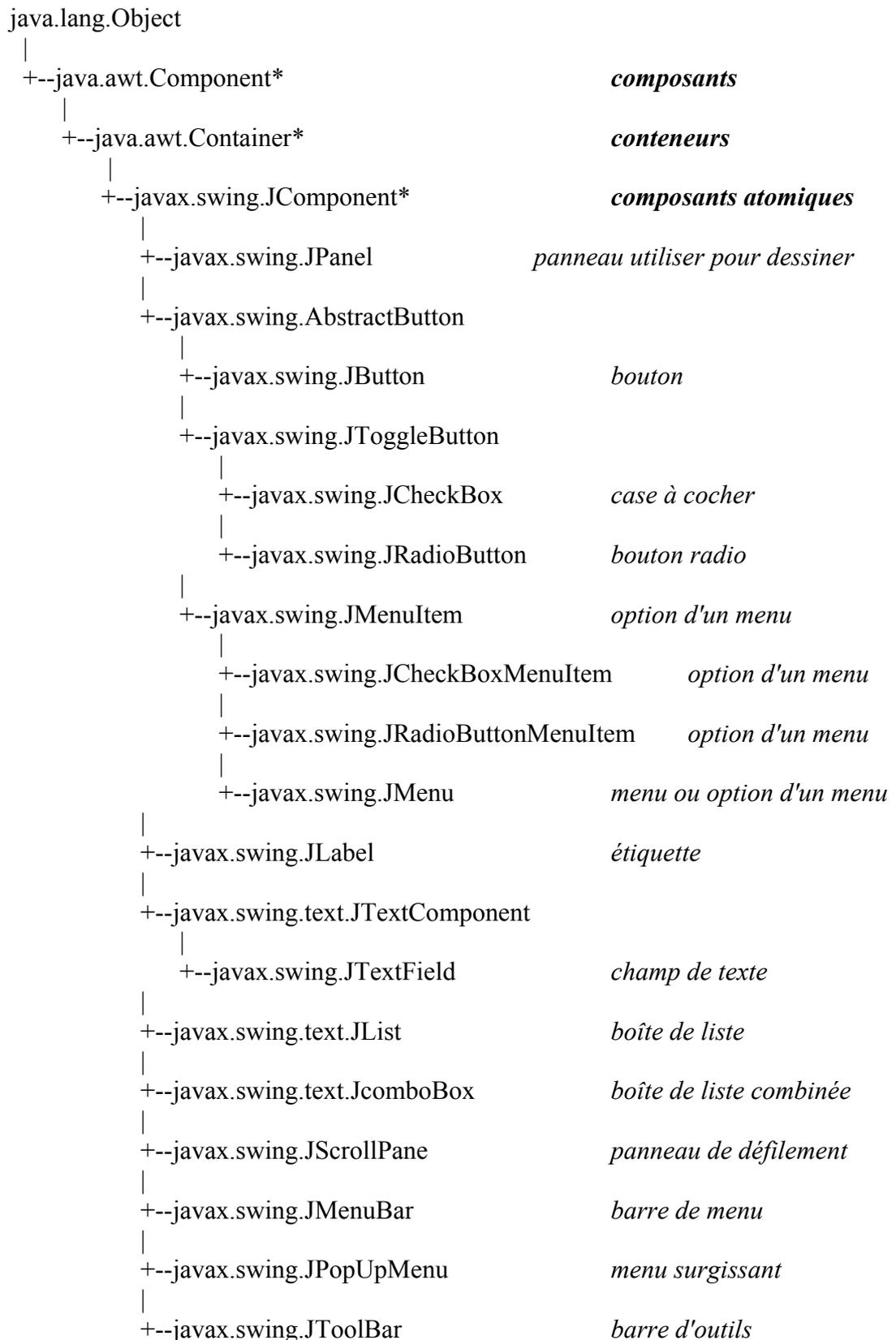
Il existe deux principaux types de **composants** susceptibles d'intervenir dans une interface graphique :

les **conteneurs** qui sont destinés à contenir d'autres composants, comme par exemple les fenêtres ;

les **composants atomiques** qui sont des composants qui ne peuvent pas en contenir d'autres, comme par exemple les boutons.

Les classes suivies d'un astérisque (*) sont abstraites.





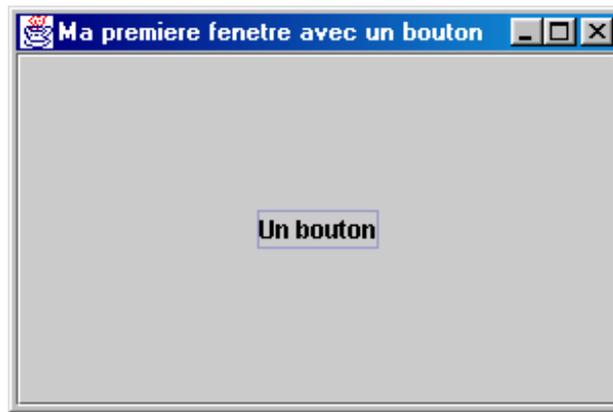
✱ *Afin d'éviter d'avoir à s'interroger sans cesse sur la répartition dans les paquetages des différentes classes utilisées dans les interfaces graphiques, nous importerons systématiquement toutes les classes des paquetages `java.awt` et `javax.swing`.*

I.2. Un premier exemple

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

class MaFenetre extends JFrame {
    private JButton MonBouton ;
    public MaFenetre () {
        super() ;
        //appel du constructeur par défaut de la classe JFrame
        //qui peut être omis
        setTitle("Ma premiere fenetre avec un bouton") ;
        //initialisation du titre de la fenêtre
        setBounds(10,40,300,200) ;
        //le coin supérieur gauche de la fenêtre est placé au pixel
        //de coordonnées 10, 40 et ses dimensions seront de 300 *
        //200 pixels
        MonBouton = new JButton("Un bouton") ;
        //création d'un bouton de référence MonBouton portant
        //l'étiquette Un bouton
        getContentPane().add(MonBouton) ;
        //Pour ajouter un bouton à une fenêtre, il faut incorporer
        //le bouton dans le contenu de la fenêtre. La méthode
        //getContentPane de la classe JFrame fournit une
        //référence à ce contenu, de type Container.
        //La méthode add de la classe Container permet
        //d'ajouter le bouton au contenu de la fenêtre.
    }
}

public class MonProg {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ;
        //rend visible la fenêtre de référence fen
    }
}
```



Contrairement à une fenêtre, un bouton est visible par défaut.

Une fenêtre de type `JFrame` peut être retaillée, déplacée et réduite à une icône. **Attention**, la fermeture d'une fenêtre de type `JFrame` ne met pas fin au programme, mais rend simplement la fenêtre invisible (comme si on appelait la méthode `setVisible(false)`). Pour mettre fin au programme, il faut fermer la fenêtre console.

I.3. Les fenêtres graphiques : la classe JFrame

La classe `JFrame` du paquetage `javax.swing` permet de créer des **fenêtres graphiques**.

La classe `JFrame` dispose de deux constructeurs : un constructeur sans argument et un constructeur avec un argument correspondant au titre de la fenêtre.

Remarque

Les instructions

```
JFrame fen = new JFrame() ;  
fen.setTitle("Une fenêtre") ;
```

et l'instruction

```
JFrame fen = new JFrame("Une fenêtre") ;
```

sont équivalentes.

Une fenêtre est un conteneur destiné à contenir d'autres composants.

La méthode `getContentPane` de la classe `JFrame` fournit une référence, de type `Container`, au contenu de la fenêtre.

Les méthodes `add` et `remove` de la classe `Container` permettent respectivement d'ajouter et de supprimer un composant d'une fenêtre.

```
Component add(Component compo)  
void remove(Component compo)
```

I.4. Des méthodes utiles de la classe Component

La classe Component est une classe abstraite dérivée de la classe Object qui encapsule les composants d'une interface graphique.

Les méthodes suivantes de la classe Component permettent de gérer l'aspect d'un composant et donc en particulier d'une fenêtre.

Montrer et cacher un composant
void setVisible(boolean b)

Exemple :

`compo.setVisible(true)` *rend visible le composant compo*

Activer et désactiver un composant
void setEnabled(boolean b)

Exemple :

`compo.setEnabled(true)` *active le composant compo*

Connaître l'état (activé ou non) d'un composant
boolean isEnabled()

Exemple :

`compo.isEnabled()` *retourne true si le composant compo est activé*

Modifier la couleur de fond d'un composant
void setBackground(Color c)

Modifier la couleur du premier plan d'un composant
void setForeground(Color c)

Modifier la taille d'un composant
void setSize(int largeur, int hauteur)

Interface graphique

Des méthodes utiles de la classe
Component

Modifier la position et la taille d'un composant

`void setBounds(int x, int y, int largeur, int hauteur)`

Gérer les dimensions d'un composant

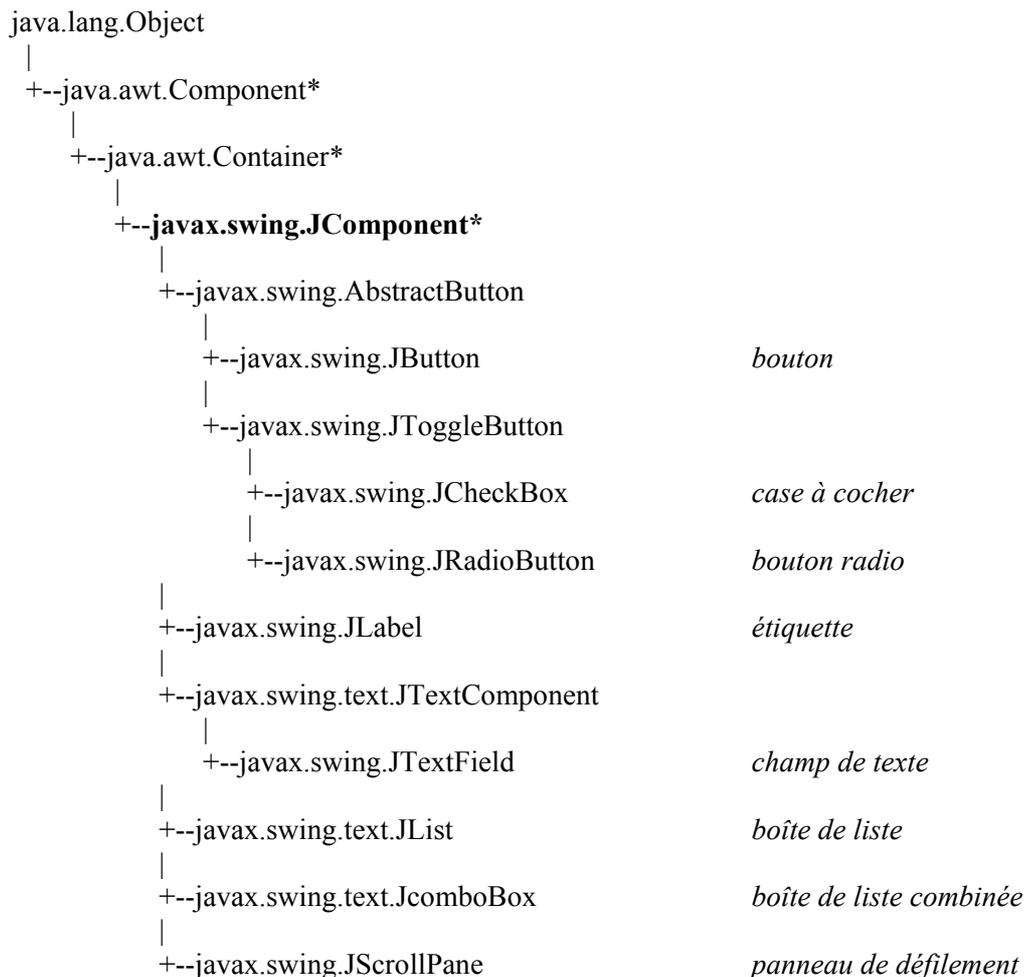
Dimension `getSize()`

`void setSize(Dimension dim)`

*//La classe Dimension du paquetage java.awt contient
deux champs publics : un champ height (hauteur) et un
champ width (largeur).*

I.5. Les composants atomiques

La classe `JComponent` est une classe abstraite dérivée de la classe `Container` qui encapsule les composants atomiques d'une interface graphique. Les principaux composants atomiques offerts par Java sont les boutons, les cases à cocher, les boutons radio, les étiquettes, les champs de texte, les boîtes de liste et les boîtes de liste combinée.



La méthode `setPreferredSize` de la classe `JComponent` permet d'imposer une taille à un composant. Elle prend en argument un objet de type `Dimension`.

Exemple

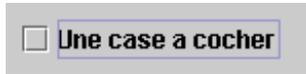
```

JButton bouton = new JButton("Un bouton") ;
bouton.setPreferredSize(new Dimension(10, 20)) ;
//bouton de largeur 10 et de hauteur 20

```

I.5.1. Les cases à cocher

La case à cocher de type **JCheckBox** permet à l'utilisateur d'effectuer un choix de type oui/non.



Exemple

```
class MaFenetre extends JFrame {
    private JCheckBox MaCase;
    public MaFenetre () {
        super("Une fenetre avec une case") ;
        setBounds(10,40,300,200) ;
        MaCase = new JCheckBox("Une case") ;
        //création d'une case à cocher de référence MaCase
        //portant l'étiquette Une case
        getContentPane().add(MaCase) ;
    }
}
```

Par défaut, une case à cocher est construite dans l'état non coché (*false*). La méthode `isSelected` de la classe `AbstractButton` permet de connaître l'état (*true* ou *false*) d'une case à cocher à un moment donné. La méthode `setSelected` de la classe `AbstractButton` permet de modifier l'état d'une case à cocher.

Exemple

```
MaCase.setSelected(true) ;
//coche la case de référence MaCase
```

I.5.2. Les boutons radio

Le 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.



Exemple

```
class MaFenetre extends JFrame {
    private JRadioButton bRouge;
    private JRadioButton bVert;
    public MaFenetre () {
        super("Une fenetre avec des boutons radio") ;
        setBounds(10,40,300,200) ;
        bRouge = new JRadioButton("Rouge") ;
        //création d'un bouton radio de référence bRouge
        //portant l'étiquette Rouge
        bVert = new JRadioButton("Vert") ;
        //création d'un bouton radio de référence bVert
        //portant l'étiquette Vert
        ButtonGroup groupe = new ButtonGroup() ;
        groupe.add(bRouge) ; groupe.add(bVert) ;
        //Un objet de type ButtonGroup (classe du
        //paquetage javax.swing, dérivée de la classe Object)
        //sert uniquement à assurer la désactivation
        //automatique d'un bouton 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.
        Container contenu = getContentPane() ;
        contenu.setLayout(new FlowLayout()) ;
        //un objet de type FlowLayout est un gestionnaire de
        //mise en forme qui dispose les composants les uns à la
        //suite des autres
        contenu.add(bRouge) ;
    }
}
```

```
contenu.add(bVert) ;  
    //Ajout de chaque bouton radio dans la fenêtre. Un  
    objet de type ButtonGroup n'est pas un composant et  
    ne peut pas être ajouté à un conteneur.  
    }  
}
```

Par défaut, un bouton radio est construit dans l'état non sélectionné (*false*) (utilisation des méthodes `isSelected` et `setSelected` de la classe `AbstractButton`).

I.5.3. Les étiquettes

Une étiquette de type **JLabel** permet d'afficher dans un conteneur un texte (d'une seule ligne) non modifiable, mais que le programme peut faire évoluer.



texte initial

Exemple

```
class MaFenetre extends JFrame {
    private JLabel MonTexte;
    public MaFenetre () {
        super("Une fenetre avec une etiquette") ;
        setBounds(10,40,300,200) ;
        MonTexte = new JLabel ("texte initial") ;
        //création d'une étiquette de référence MonTexte
        //contenant le texte texte initial
        getContentPane().add(MonTexte) ;
        MonTexte.setText("nouveau texte") ;
        //modification du texte de l'étiquette de référence
        MonTexte
    }
}
```

I.5.4. Les champs de texte

Un champ de texte (ou boîte de saisie) de type **JTextField** est une zone rectangulaire dans laquelle l'utilisateur peut entrer ou modifier un texte (d'une seule ligne).



Exemple

```
class MaFenetre extends JFrame {
    private JTextField MonChamp1 ;
    private JTextField MonChamp2 ;
    public MaFenetre () {
        super("Une fenetre avec deux champs de texte") ;
        setBounds(10,40,300,200) ;
        MonChamp1 = new JTextField(20) ;
        //champ de taille 20 (la taille est exprimée en nombre
        //de caractères standard affichés dans le champ)
        Monchamp2 = new JTextField("texte initial", 10) ;
        //champ de taille 10 contenant au départ le texte texte
        //initial
        getContentPane().add(MonChamp1) ;
        getContentPane().add(MonChamp2) ;
    }
}
```

Aucun texte n'est associé à un tel composant pour l'identifier. On pourra utiliser un objet de type **JLabel** qu'on prendra soin de disposer convenablement.

La méthode `getText` permet de connaître à tout moment le contenu d'un champ de texte.

Exemple

```
String ch= Monchamp2.getText() ;
```

I.5.5. Les boîtes de liste

La boîte de liste de type **JList** permet de choisir une ou plusieurs valeurs dans une liste prédéfinie. Initialement, aucune valeur n'est sélectionnée dans la liste.



Exemple

```
String[] couleurs = {"rouge", "bleu", "gris", "vert",
"jaune", "noir"};
JList MaListe = new JList(couleurs) ;
MaListe.setSelectedIndex(2) ;
//sélection préalable de l'élément de rang 2
```

Il existe trois sortes de boîtes de liste, caractérisées par un paramètre de type :

| Valeur du paramètre de type | Type de sélection correspondante |
|-----------------------------|-------------------------------------------------------|
| SINGLE_SELECTION | sélection d'une seule valeur |
| SINGLE_INTERVAL_SELECTION | sélection d'une seule plage de valeurs (contiguës) |
| MULTIPLE_INTERVAL_SELECTION | sélection d'un nombre quelconque de plages de valeurs |

Par défaut, le type d'une boîte de liste est MULTIPLE_INTERVAL_SELECTION.

La méthode `setSelectionMode` permet de modifier le type d'une boîte de liste.

Exemple

```
MaListe.setSelectionMode(SINGLE_SELECTION);
```

Pour une boîte de liste à sélection simple, la méthode `getSelectedValue` fournit la valeur sélectionnée. Son résultat est de type `Object`.

Exemple

```
String ch = (String) MaListe.getSelectedValue();  
//cast obligatoire
```

Pour les autres types de boîte de liste, la méthode `getSelectedValue` ne fournit que la première des valeurs sélectionnées. Pour obtenir toutes les valeurs, il faut utiliser la méthode `getSelectedValues` qui fournit un tableau d'objets.

Exemple

```
Object[] valeurs = MaListe.getSelectedValues();  
for (int i=0 ;i<valeurs.length ;i++)  
    System.out.println((String) valeurs[i] );  
//cast obligatoire
```

Par défaut une boîte de liste ne possède pas de **barre de défilement**. On peut lui en ajouter une en l'introduisant dans un panneau de défilement de type **JScrollPane** (classe dérivée de la classe `JComponent`). Dans ce cas, on n'ajoute pas au conteneur la boîte de liste, mais le panneau de défilement.

Exemple

```
JFrame fen = new JFrame() ;  
JScrollPane defil = new JScrollPane(MaListe) ;  
//la liste de référence MaListe affiche 8 valeurs. Si elle en  
contient moins, la barre de défilement n'apparaît pas.  
MaListe.setVisibleRowCount(4) ;  
//seules 4 valeurs de la liste sont visibles à la fois  
fen.getContentPane().add(defil) ;  
//ajoute le panneau de défilement au contenu de la fenêtre
```

I.5.6. Les boîtes de liste combinée

La boîte de liste combinée (boîte combo) de type **JComboBox** associe un champ de texte et une boîte de liste à sélection simple. Tant que le composant n'est pas sélectionné, seul le champ de texte s'affiche :



Lorsque l'utilisateur sélectionne le champ de texte, la boîte de liste s'affiche :



L'utilisateur peut choisir une valeur dans la boîte de liste qui s'affiche alors dans le champ de texte. Initialement, aucune valeur n'est sélectionnée dans la liste.

Exemple

```
String[] couleurs = {"rouge", "bleu", "gris", "vert",  
"jaune", "noir"};  
JComboBox MaCombo = new JComboBox(couleurs) ;  
MaCombo.setSelectedIndex(2) ;  
//sélection préalable de l'élément de rang 2
```

La boîte combo est dotée d'une barre de défilement dès que son nombre de valeurs est supérieur à 8. On peut modifier le nombre de valeurs visibles avec la méthode `setMaximumRowCount`.

Exemple

```
MaCombo.setMaximumRowCount(4) ;  
//au maximum 4 valeurs sont affichées
```

Par défaut, le champ de texte associé à une boîte combo n'est pas éditable, ce qui signifie qu'il sert seulement à présenter la sélection courante de la liste. Mais il peut être rendu éditable par la méthode `setEditable`. L'utilisateur peut alors y entrer soit une valeur de la liste (en la sélectionnant), soit une valeur de son choix (en la saisissant).

Exemple

```
MaCombo.setEditable(true) ;
```

La méthode `getSelectedItem` fournit la valeur sélectionnée. Son résultat est de type `Object`.

Exemple

```
String ch = (String) MaCombo.getSelectedItem();
```

La méthode `getSelectedIndex` fournit le rang de la valeur sélectionnée. Elle fournit la valeur `-1`, si l'utilisateur a entré une nouvelle valeur (c'est à dire une valeur qui n'a pas été sélectionnée dans la liste).

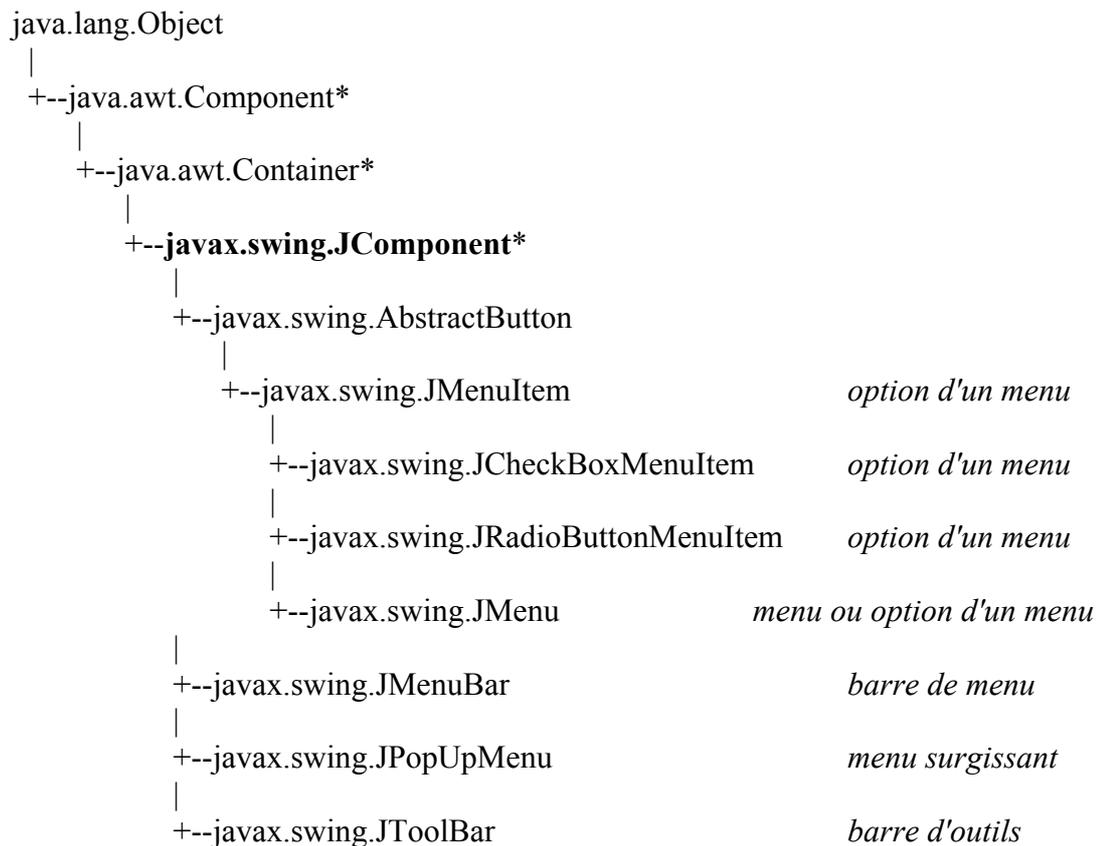
La boîte combo dispose de méthodes appropriées à sa modification.

Exemple

```
MaCombo.addItem("orange") ;  
    //ajoute la valeur orange à la fin de la liste combo  
MaCombo.addItemAt("orange", 2) ;  
    //ajoute la valeur orange à la position 2 de la liste combo  
MaCombo.removeItem("gris") ;  
    //supprime la valeur gris de la liste combo
```

I.6. Les menus et les barres d'outils

Une fenêtre de type JFrame est composée de composants atomiques, mais aussi de composants qui lui sont propres comme les menus et les barres d'outils.



I.6.1. Les menus déroulants

Les menus déroulants usuels font intervenir trois sortes d'objets:
un objet **barre de menu** de type `JMenuBar` ;
différents objets **menu**, de type `JMenu`, visibles dans la
barre de menu ;
pour chaque menu, les différentes **options**, de type
`JMenuItem`, qui le constituent.

La méthode `setEnabled` de la classe `Component` permet
d'activer ou de désactiver un menu ou une option.

a. Un exemple

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

class MaFenetre extends JFrame {
    private JMenuBar barreMenus ;
    private JMenu couleur, dimensions ;
    private JMenuItem rouge, vert, hauteur, largeur ;
    public MaFenetre () {
        super("Une fenetre avec un menu") ;
        setSize(300, 200) ;
        //création d'une barre de menu
        barreMenus = new JMenuBar() ;
        setJMenuBar(barreMenus) ;
        //ajout de la barre de menu dans la fenêtre
        //création d'un menu Couleur et de ses options Rouge et
        Vert
        couleur = new JMenu("Couleur") ;
        barreMenus.add(couleur) ;
        rouge = new JMenuItem("Rouge") ;
        couleur.add(rouge) ;
        couleur.addSeparator() ;
        //ajout d'une barre séparatrice avant l'option suivante
        vert = new JMenuItem("Vert") ;
        couleur.add(vert) ;
        //création d'un menu Dimensions et de ses options
        Hauteur et Largeur
        dimensions = new JMenu("Dimensions") ;
        barreMenus.add(dimensions) ;
        hauteur = new JMenuItem("Hauteur") ;
        dimensions.add(hauteur) ;
        dimensions.addSeparator() ;
        largeur = new JMenuItem("Largeur") ;
        dimensions.add(largeur) ;
    }
}
```

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

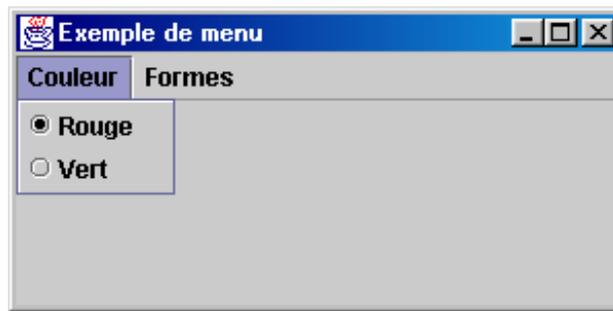


b. Les différentes sortes d'options

Les options de type `JMenuItem` sont les options les plus courantes dans un menu, mais il existe deux autres types d'options : des options cases à cocher de type `JCheckBoxMenuItem` et des options boutons radio de type `JRadioButtonMenuItem`.

Exemple

```
JMenuBar barreMenus = new JMenuBar() ;
setJMenuBar(barreMenus) ;
//création d'un menu Couleur et de son groupe de deux
options Rouge et Vert
JMenu couleur = new JMenu("Couleur") ;
barreMenus.add(couleur) ;
JRadioButtonMenuItem rouge = new
JRadioButtonMenuItem("Rouge") ;
JRadioButtonMenuItem vert = new
JRadioButtonMenuItem("Vert") ;
couleur.add(rouge) ; couleur.add(vert) ;
ButtonGroup groupe = new ButtonGroup() ;
groupe.add(rouge) ; groupe.add(vert) ;
//les options boutons radio sont placées dans un groupe de
type ButtonGroup afin d'assurer l'unicité de la sélection à
l'intérieur du groupe (cf paragraphe I.5.2.)
//création d'un menu Formes et de ses cases à cocher
Rectangle et Ovale
JMenu formes = new JMenu("Formes") ;
barreMenus.add(formes) ;
JCheckBoxMenuItem rectangle = new
JCheckBoxMenuItem("Rectangle") ;
JCheckBoxMenuItem ovale = new
JCheckBoxMenuItem("Ovale") ;
formes.add(rectangle) ; formes.add(ovale) ;
```



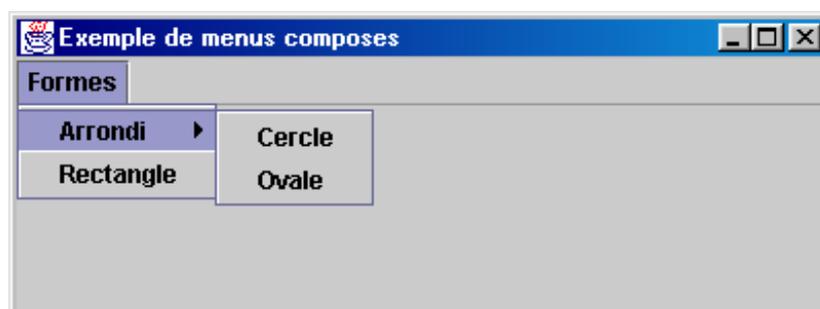
La méthode `isSelected` de la classe `AbstractButton` permet de savoir si une option est sélectionnée. Elle retourne *true* si l'option est sélectionnée, *false* sinon.

c. Composition des options

Jusqu'à présent, un menu était formé d'une simple liste d'options. En fait, une option peut à son tour faire apparaître une liste de sous-options. Pour ce faire, il suffit d'utiliser dans un menu une option de type `JMenu` (et non plus de type `JMenuItem`). Cette démarche peut être répétée autant de fois que voulu.

Exemple

```
JMenuBar barreMenus = new JMenuBar() ;  
setJMenuBar(barreMenus) ;  
//création d'un menu Formes composé d'une option Arrondi  
composée elle même de deux options Cercle et Ovale, et,  
d'une option Rectangle  
JMenu formes = new JMenu("Formes") ;  
barreMenus.add(formes) ;  
JMenu arrondi = new JMenu("Arrondi") ;  
formes.add(arrondi) ;  
JMenuItem cercle = new JMenuItem("Cercle") ;  
arrondi.add(cercle) ;  
JMenuItem ovale = new JMenuItem("Ovale") ;  
arrondi.add(ovale) ;  
JMenuItem rectangle = new JMenuItem("Rectangle") ;  
formes.add(rectangle) ;
```



I.6.2. Les menus surgissants

Les menus usuels, que nous venons d'étudier, sont des menus rattachés à une barre de menu et donc affichés en permanence dans une fenêtre Java. Java propose aussi des **menus surgissants** de type `JPopupMenu` qui sont des menus (sans nom) dont la liste d'options apparaît suite à une certaine action de l'utilisateur.

Exemple

```
//création d'un menu surgissant comportant deux options  
Rouge et Vert  
JPopupMenu couleur = new JPopupMenu() ;  
rouge = new JMenuItem("Rouge") ;  
couleur.add(rouge) ;  
vert = new JMenuItem("Vert") ;  
couleur.add(vert) ;
```

Un menu surgissant doit être affiché explicitement par le programme, en utilisant la méthode `show` de la classe `JPopupMenu`. Cette méthode a deux arguments : le composant concerné (en général, il s'agira de la fenêtre) et les coordonnées auxquelles on souhaite faire apparaître le menu (coordonnées du coin supérieur gauche). Le menu sera affiché jusqu'à ce que l'utilisateur sélectionne une option ou ferme le menu en cliquant à côté.

Exemple

```
JFrame fen = new JFrame() ;  
fen.setVisible(true) ;  
couleur.show(fen, x, y) ;  
//affiche le menu aux coordonnées x et y
```

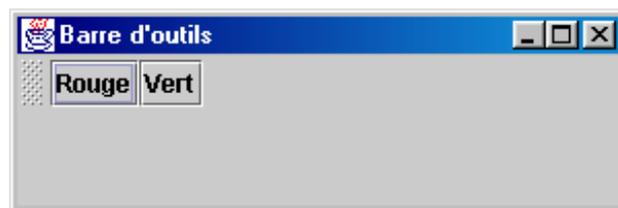


I.6.3. Les barres d'outils

Une barre d'outils de type `JToolBar` est un ensemble de boutons regroupés linéairement sur un des bords de la fenêtre. En général, ces boutons comportent plutôt des icônes que des libellés.

Exemple

```
class MaFenetre extends JFrame {
    JToolBar barreOutils ;
    JButton boutonRouge, boutonVert ;
    public MaFenetre () {
        super("Une fenetre avec une barre d'outils") ;
        setSize(300, 200) ;
        //création d'une barre d'outils composée de deux
        //boutons: un bouton Rouge et un bouton Vert
        barreOutils = new JToolBar() ;
        boutonRouge = new JButton("Rouge") ;
        barreOutils.add(boutonRouge) ;
        boutonVert = new JButton("Vert") ;
        barreOutils.add(boutonVert) ;
        getContentPane().add(barreOutils) ; }
}
```



Par défaut, une barre d'outils est **flottante**, c'est à dire qu'on peut la déplacer d'un bord à un autre de la fenêtre, ou à l'intérieur de la fenêtre. On peut interdire à une barre d'outils de flotter grâce à la méthode `setFloatable`.

Exemple

```
barreOutils.setFloatable(false) ;
```

Un bouton peut être créé avec une icône de type `ImageIcon` (classe du paquetage `javax.swing` dérivée de la classe `Object`) au lieu d'un texte.

Exemple

```
JToolBar barreOutils = new JToolBar() ;  
ImageIcon iconeVert = new ImageIcon("vert.gif") ;  
    //création d'une icône à partir d'un fichier "vert.gif"  
    //contenant un dessin d'un carré de couleur vert  
JButton boutonVert = new JButton(iconeVert) ;  
barreOutils.add(boutonVert) ;
```

I.6.4. Les bulles d'aide

Une bulle d'aide est un petit rectangle (nommé "tooltip" en anglais) contenant un bref texte explicatif qui apparaît lorsque la souris est laissée un instant sur un composant (boutons, menus, ...). Java permet d'obtenir un tel affichage pour n'importe quel composant grâce à la méthode `setToolTipText` de la classe `JComponent`.

Exemple

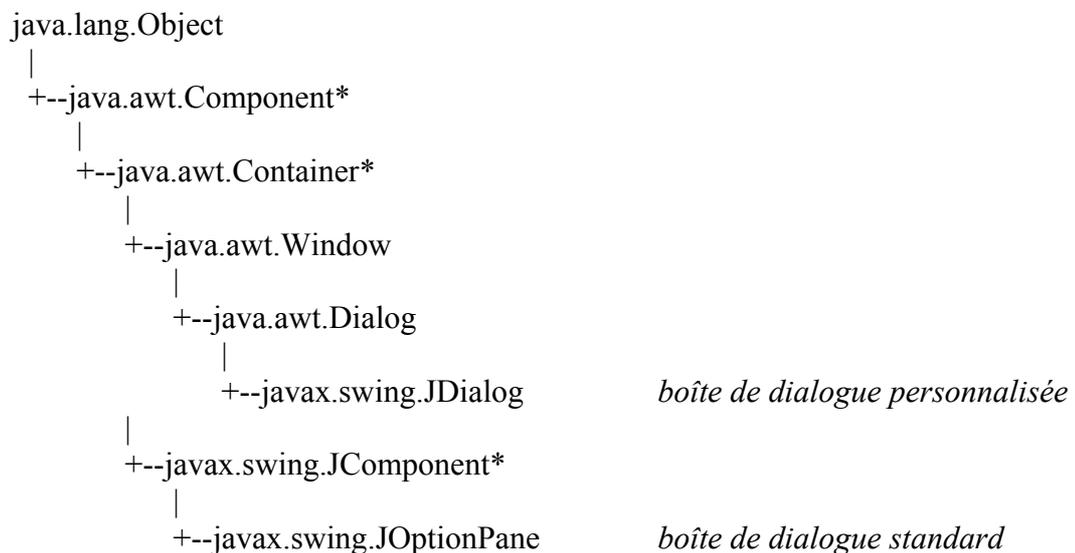
```
barreMenus = new JMenuBar() ;  
setJMenuBar(barreMenus) ;  
//création d'un menu Couleur et de ses options Rouge et Vert  
couleur = new JMenu("Couleur") ;  
barreMenus.add(couleur) ;  
rouge = new JMenuItem("Rouge") ;  
rouge.setToolTipText("fond rouge") ;  
couleur.add(rouge) ;  
couleur.addSeparator() ;  
vert = new JMenuItem("Vert") ;  
vert.setToolTipText("fond vert") ;  
couleur.add(vert) ;
```



I.7. Les boîtes de dialogue

La boîte de dialogue est un conteneur. Elle permet de regrouper n'importe quels composants dans une sorte de fenêtre qu'on fait apparaître ou disparaître.

Java propose un certain nombre de boîtes de dialogue standard obtenues à l'aide de méthodes de classe de la classe `JOptionPane` : boîtes de message, boîtes de confirmation, boîtes de saisie et boîtes d'options. La classe `JDialog` permet de construire des boîtes de dialogue personnalisées.



I.7.1. Les boîtes de message

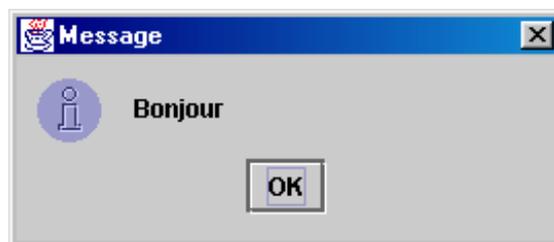
Une boîte de message fournit à l'utilisateur un message qui reste affiché tant que l'utilisateur n'agit pas sur le bouton OK. Elle est construite à l'aide de la méthode de classe `showMessageDialog` de la classe `JOptionPane`.

Exemple

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

class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ; setSize(300, 200) ; }
}

public class BoiteMess {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ;
        JOptionPane.showMessageDialog(fen, "Bonjour") ;
        //le premier argument de la méthode
        showMessageDialog correspond à la fenêtre parent
        de la boîte de message, c'est à dire la fenêtre dans
        laquelle la boîte de message va s'afficher. Cet
        argument peut prendre la valeur null.
    }
}
```



Dans l'exemple précédent, nous n'avons défini que le contenu du message. Il existe une variante de la méthode `showMessageDialog` qui permet aussi de choisir le titre de la

boîte et le type d'icône parmi la liste suivante (les paramètres sont des constantes entières de la classe `JOptionPane`).

| Paramètre | Type d'icône |
|----------------------------------------------|---------------|
| <code>JOptionPane.ERROR_MESSAGE</code> | Erreur |
| <code>JOptionPane.INFORMATION_MESSAGE</code> | Information |
| <code>JOptionPane.WARNING_MESSAGE</code> | Avertissement |
| <code>JOptionPane.QUESTION_MESSAGE</code> | Question |
| <code>JOptionPane.PLAIN_MESSAGE</code> | Aucune icône |

Exemple

```
JOptionPane.showMessageDialog(fen, "Mauvais  
choix", "Message d'avertissement",  
JOptionPane.WARNING_MESSAGE) ;
```



I.7.2. Les boîtes de confirmation

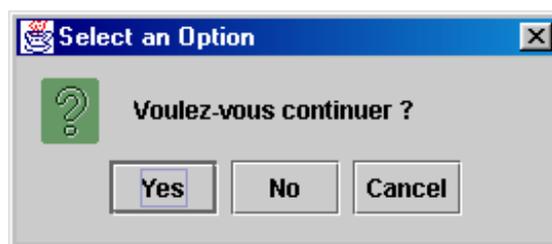
Une boîte de confirmation offre à l'utilisateur un choix de type oui/non. Elle est construite à l'aide de la méthode de classe `showConfirmDialog` de la classe `JOptionPane`.

Exemple

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

class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ; setSize(300, 200) ; }
}

public class BoiteConf {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ;
        JOptionPane.showConfirmDialog(fen, "Voulez-vous
        continuer ?") ;
    }
}
```

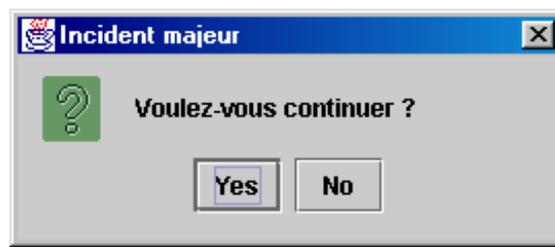


Dans l'exemple précédent, nous n'avons défini que la question posée à l'utilisateur. Il existe une variante de la méthode `showConfirmDialog` qui permet aussi de choisir le titre de la boîte et la nature des boutons qui s'y trouvent parmi la liste suivante (les paramètres sont des constantes entières de la classe `JOptionPane`).

| Paramètre | Type de boîte |
|--------------------------------------|---------------------------|
| JOptionPane.DEFAULT_OPTION (-1) | Erreur |
| JOptionPane.YES_NO_OPTION (0) | boutons YES et NO |
| JOptionPane.YES_NO_CANCEL_OPTION (1) | boutons YES, NO et CANCEL |
| JOptionPane.OK_CANCEL_OPTION (2) | boutons OK et CANCEL |

Exemple

```
JOptionPane.showConfirmDialog(fen, "Voulez-vous continuer ?", "Incident majeur",
JOptionPane.YES_NO_OPTION) ;
```



La valeur de retour de la méthode `showConfirmDialog` précise l'action effectuée par l'utilisateur sous la forme d'un entier ayant comme valeur l'une des constantes suivantes de la classe `JOptionPane` :

- YES_OPTION (0),
- OK_OPTION (0),
- NO_OPTION (1),
- CANCEL_OPTION (2),
- CLOSED_OPTION (-1).

I.7.3. Les boîtes de saisie

Une boîte de saisie permet à l'utilisateur de fournir une information sous la forme d'une chaîne de caractères. Elle est construite à l'aide de la méthode de classe `showInputDialog` de la classe `JOptionPane`.

Exemple

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

class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ; setSize(300, 200) ; }
}

public class BoiteSaisie {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ;
        JOptionPane.showInputDialog (fen, "Donnez un
        texte") ;
    }
}
```

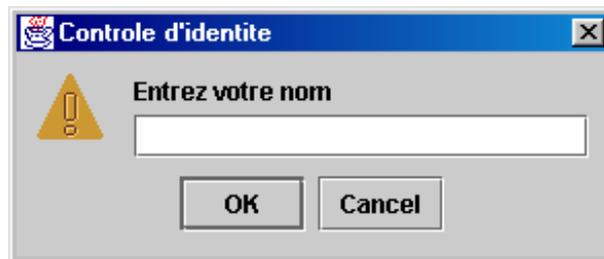


La valeur de retour de la méthode `showInputDialog` est soit un objet de type `String` contenant le texte fourni par l'utilisateur, soit la valeur `null` si l'utilisateur n'a pas confirmé sa saisie par le bouton OK.

Dans l'exemple précédent, nous n'avons défini que la question posée à l'utilisateur. Il existe une variante de la méthode `showInputDialog` qui permet aussi de choisir le titre de la boîte et le type d'icône (suivant les valeurs fournies au paragraphe I.7.1.).

Exemple

```
JOptionPane.showInputDialog(fen, "Entrez votre nom",  
"Controle d'identite",  
JOptionPane.WARNING_MESSAGE) ;
```



I.7.4. Les boîtes d'options

Une boîte d'options permet à l'utilisateur de choisir une valeur parmi une liste de valeurs, par l'intermédiaire d'une boîte combo. Elle est construite à l'aide de la méthode de classe `showInputDialog` de la classe `JOptionPane`.

Exemple

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

class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ; setSize(300, 200) ; }
}

public class BoiteOptions {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ;
        String[] couleurs = {"rouge", "bleu", "gris", "vert",
            "jaune", "noir"};
        JOptionPane.showInputDialog (fen, "Choisissez
            une couleur",
            "Boite d'options", //titre de la boîte
            JOptionPane.QUESTION_MESSAGE,
            //type d'icône suivant les valeurs du paragraphe I.7.1.
            null,
            //icône supplémentaire (ici aucune)
            couleurs,
            //liste de valeurs représentée dans la boîte combo
            couleurs[1]) ;
            //valeur sélectionnée par défaut
    }
}
```



La valeur de retour de la méthode `showInputDialog` est soit un objet de type `Object` contenant la valeur sélectionnée par l'utilisateur, soit la valeur *null* si l'utilisateur n'a pas confirmé sa saisie par le bouton OK.

I.7.5. Les boîtes de dialogue personnalisées

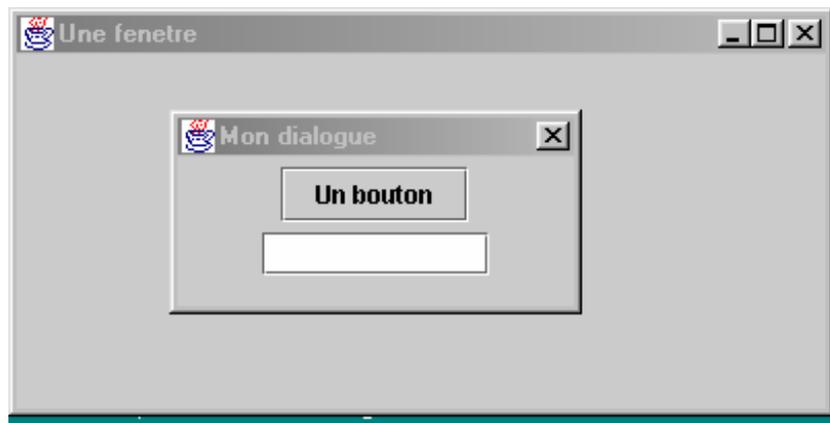
La classe `JDialog` permet de créer ses propres boîtes de dialogue.

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

```
class MaFenetre extends JFrame {  
    public MaFenetre () {  
        super("Une fenetre") ;  
        setSize(300, 200) ; }  
}
```

```
class MonDialog extends JDialog {  
    private JButton MonBouton ;  
    private JTextField MonChamp ;  
    public MonDialog (JFrame fen) {  
        super(fen,           //fenêtre parent  
              "Mon dialogue", //titre  
              true) ;      //boîte modale : l'utilisateur ne peut agir que  
                           //sur les composants de la boîte de dialogue et  
                           //ceci tant qu'il n'a pas mis fin au dialogue  
        setSize(200, 100) ;  
        //il est nécessaire d'attribuer une taille à une boîte de  
        //dialogue avant de l'afficher  
        MonBouton = new JButton("Un bouton") ;  
        MonChamp = new JTextField(10) ;  
        Container contenu = getContentPane() ;  
        contenu.setLayout(new FlowLayout()) ;  
        //un objet de type FlowLayout est un gestionnaire de  
        //mise en forme qui dispose les composants les uns à la  
        //suite des autres  
        contenu.add(MonBouton); contenu.add(MonChamp) ;  
        //ajout d'un bouton et d'un champ de texte dans la boîte  
        //de dialogue  
    }  
}
```

```
public class MonProgBoiteDialogPers {  
    public static void main(String args[]) {  
        JFrame fen = new MaFenetre() ;  
        fen.setVisible(true) ;  
        JDialog bd = new MonDialog(fen) ;  
        bd.setVisible(true) ;  
        // affiche la boîte de dialogue de référence bd  
    }  
}
```



I.8. Les gestionnaires de mise en forme

Pour chaque conteneur (fenêtre, boîte de dialogue, ...), Java permet de choisir un gestionnaire de mise en forme (en anglais "Layout manager") responsable de la disposition des composants.

Les gestionnaires de mise en forme proposés par Java sont les gestionnaires BorderLayout, FlowLayout, CardLayout, GridLayout, BoxLayout et GridBagLayout. Ce sont tous des classes du paquetage java.awt dérivées de la classe Object et qui implémentent l'interface LayoutManager.

La méthode setLayout de la classe Container permet d'associer un gestionnaire de mise en forme à un conteneur. Le gestionnaire BorderLayout est le gestionnaire par défaut des fenêtres et des boîtes de dialogue.

Exemple

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

class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ;
        setSize(300, 200) ;
        getContentPane().setLayout(new FlowLayout()) ;
        //changement de gestionnaire de mise en forme
    }
}

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

I.8.1. Le gestionnaire BorderLayout

Le gestionnaire de mise en forme BorderLayout permet de placer chaque composant dans une zone géographique.

L'emplacement d'un composant est choisi en fournissant en argument de la méthode add de la classe Container l'une des constantes entières suivantes (on peut utiliser indifféremment le nom de la constante ou sa valeur) :

| Constante symbolique | Valeur |
|----------------------|----------|
| BorderLayout.NORTH | "North" |
| BorderLayout.SOUTH | "South" |
| BorderLayout.EAST | "East" |
| BorderLayout.WEST | "West" |
| BorderLayout.CENTER | "Center" |

Si aucune valeur n'est précisée à la méthode add, le composant est placé au centre.

La classe BorderLayout dispose de deux constructeurs :

```
public BorderLayout() ;
```

```
public BorderLayout(int hgap, int vgap) ;
```

//hgap et vgap définissent respectivement l'espace horizontal et l'espace vertical (en nombre de pixels) entre les composants d'un conteneur. Par défaut, les composants sont espacés de 5 pixels.

Exemple

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

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

```
class MaFenetre extends JFrame {
```

```
    public MaFenetre () {
```

```
        super("Une fenetre") ; setSize(300, 200) ;
```

```
        Container contenu = getContentPane() ;
```

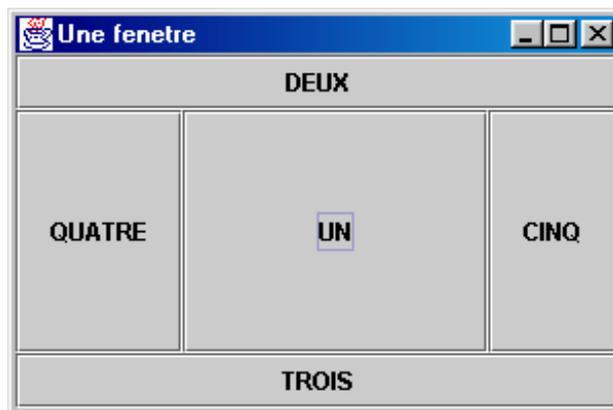
```
        contenu.setLayout(new BorderLayout()) ; //inutile
```

```
        contenu.add(new JButton("UN")) ;
```

```
        //bouton placé au centre par défaut
```

```
    contenu.add(new JButton("DEUX"), "North") ;  
    contenu.add(new JButton("TROIS"), "South") ;  
    contenu.add(new JButton("QUATRE"), "West") ;  
    contenu.add(new JButton("CINQ"), "East") ;  
  }  
}
```

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



✱ *Le gestionnaire de mise en forme BorderLayout ne tient pas compte de la taille souhaitée des composants, qui peut être imposée par la méthode `setPreferredSize` de la classe `JComponent` (cf. paragraphe VII.5.).*

I.8.2. Le gestionnaire FlowLayout

Le gestionnaire de mise en forme FlowLayout permet de disposer les composants les uns à la suite des autres, de gauche à droite.

La classe FlowLayout dispose de trois constructeurs :

```
public FlowLayout() ;
```

```
public FlowLayout(int align) ;
```

//align est un paramètre d'alignement d'une ligne de composants par rapport aux bords verticaux de la fenêtre. Ce paramètre peut prendre ses valeurs parmi les constantes entières suivantes (on peut utiliser indifféremment le nom de la constante ou sa valeur) : FlowLayout.LEFT("Left"), FlowLayout.RIGHT("Right") ou FlowLayout.CENTER("Center"). Par défaut les composants sont alignés à gauche.

```
public FlowLayout(int align, int hgap, int vgap) ;
```

//hgap et vgap définissent les espaces entre les composants.

Exemple

```
class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ; setSize(300, 200) ;
        Container contenu = getContentPane() ;
        contenu.setLayout(new FlowLayout()) ;
        //changement de gestionnaire de mise en forme
        contenu.add(new JButton("UN")) ;
        contenu.add(new JButton("DEUX")) ;
    }
}
```



- ✱ *Le gestionnaire de mise en forme `FlowLayout` tient compte, dans la mesure du possible, de la taille souhaitée des composants, qui peut être imposée par la méthode `setPreferredSize` de la classe `JComponent`.*

I.8.3. Le gestionnaire CardLayout

Le gestionnaire de mise en forme `CardLayout` permet de disposer les composants suivant une pile, de telle façon que seul le composant supérieur soit visible à un moment donné.

Exemple

```
class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ; setSize(300, 200) ;
        Container contenu = getContentPane() ;
        CardLayout pile = new CardLayout(30, 20) ;
        //les arguments précisent les retraits entre le
        //composant et le conteneur : 30 pixels de part et d'autre
        //et 20 pixels en haut et en bas
        contenu.setLayout(pile) ;
        //changement de gestionnaire de mise en forme
        contenu.add(new JButton("UN"), "bouton1") ;
        //la chaîne "bouton1" permet d'identifier le composant
        //au sein du conteneur
        contenu.add(new JButton("DEUX"), "bouton2") ;
        contenu.add(new JButton("TROIS"), "bouton3") ;
    }
}
```

Par défaut, le composant visible est le premier ajouté au conteneur (dans l'exemple précédent, c'est le bouton UN). On peut faire apparaître un autre composant de la pile de l'une des façons suivantes :

```
pile.next(contenu) ; //affiche le composant suivant
pile.previous(contenu) ; //affiche le composant précédent
pile.first(contenu) ; //affiche le premier composant
pile.last(contenu) ; //affiche le dernier composant
pile.show(contenu, "bouton3") ;
//affiche le composant identifié par la chaîne "bouton3"
```

I.8.4. Le gestionnaire GridLayout

Le gestionnaire de mise en forme GridLayout permet de disposer les composants les uns à la suite des autres sur une grille régulière, chaque composant occupant une cellule de la grille.

La classe GridLayout dispose de deux constructeurs :

```
public GridLayout(int rows, int cols) ;  
    //rows et cols définissent respectivement le nombre de  
    //lignes et de colonnes de la grille.  
public GridLayout(int rows, int cols, int hgap, int vgap) ;  
    //hgap et vgap définissent les espaces entre les  
    //composants.
```

Les dernières cases d'une grille peuvent rester vides. Toutefois, si plus d'une ligne de la grille est vide, le gestionnaire réorganisera la grille, de façon à éviter une perte de place.

Exemple

```
class MaFenetre extends JFrame {  
    public MaFenetre () {  
        super("Une fenetre") ; setSize(300, 200) ;  
        Container contenu = getContentPane() ;  
        contenu.setLayout(new GridLayout(2, 2)) ;  
        //changement de gestionnaire de mise en forme  
        contenu.add(new JButton("UN")) ;  
        contenu.add(new JButton("DEUX")) ;  
        contenu.add(new JButton("TROIS")) ;  
        contenu.add(new JButton("QUATRE")) ;  
    }  
}
```



Le gestionnaire de mise en forme `BoxLayout` permet de disposer des composants suivant une même ligne ou une même colonne, mais avec plus de souplesse que le gestionnaire `GridLayout`.

Le gestionnaire de mise en forme `GridBagLayout`, comme le gestionnaire `GridLayout`, permet de disposer les composants suivant une grille, mais ceux-ci peuvent occuper plusieurs cellules ; en outre, la taille des cellules peut être modifiée au cours de l'exécution.

I.8.5. Un programme sans gestionnaire de mise en forme

Il est possible de n'associer aucun gestionnaire de mise en forme à un conteneur. Les composants sont alors ajoutés au conteneur à l'aide de la méthode `setBounds` de la classe `Component`.

Exemple

```
class MaFenetre extends JFrame {  
    public MaFenetre () {  
        super("Une fenetre") ; setSize(300, 200) ;  
        Container contenu = getContentPane() ;  
        contenu.setLayout(null) ;  
        //changement de gestionnaire de mise en forme  
        JButton bouton1 = new JButton("UN") ;  
        contenu.add(bouton1) ;  
        bouton1.setBounds(40, 40, 80, 30) ;  
        //le coin supérieur gauche du bouton est placé au pixel  
        //de coordonnées 40, 40 par rapport au coin supérieur  
        //gauche de la fenêtre et les dimensions du bouton sont  
        //de 80 * 30 pixels  
    }  
}
```



I.8.6. Une classe Insets pour gérer les marges

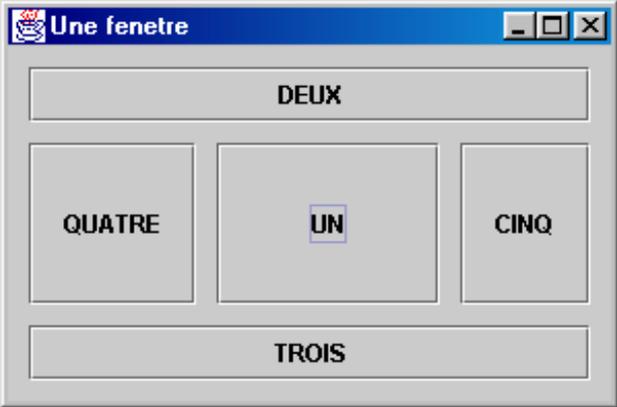
La méthode `getInsets` de la classe `Container` permet de définir les quatre marges (haut, gauche, bas et droite) d'un conteneur. Elle retourne un objet de type `Insets` (classe du package `java.awt` dérivée de la classe `Object`) défini par quatre champs publics de type entier initialisés par le constructeur suivant :

```
public Insets(int top, int left, int bottom, int right)
```

Exemple

```
class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre") ; setSize(300, 200) ;
        Container contenu = getContentPane() ;
        contenu.setLayout(new BorderLayout(10, 10)) ;
        contenu.add(new JButton("UN")) ;
        contenu.add(new JButton("DEUX"), "North") ;
        contenu.add(new JButton("TROIS"), "South") ;
        contenu.add(new JButton("QUATRE"), "West") ;
        contenu.add(new JButton("CINQ"), "East") ;
    }

    //redéfinition de la méthode getInsets afin de définir de
    //nouvelles marges pour la fenêtre
    public Insets getInsets() {
        Insets normal = super.getInsets() ;
        //récupération des marges par défaut de la fenêtre
        return new Insets(normal.top+10, normal.left+10,
            normal.bottom+10, normal.right+10) ;
        //création d'un nouvel objet de type Insets pour
        //modifier les marges de la fenêtre
    }
}
```



I.9. Dessignons avec Java

Java permet de dessiner sur n'importe quel composant grâce à des méthodes de dessin. Cependant, en utilisant directement ces méthodes, on obtient le dessin attendu mais il disparaîtra en totalité ou en partie dès que le conteneur du composant aura besoin d'être réaffiché (par exemple en cas de modification de la taille du conteneur, de déplacement, de restauration après une réduction en icône...). Pour obtenir la **permanence d'un dessin**, il est nécessaire de placer les instructions du dessin dans la méthode `paintComponent` du composant concerné. Cette méthode est automatiquement appelée par Java chaque fois que le composant est dessiné ou redessiné.

✱ *Ce problème de permanence ne se pose pas pour les composants d'un conteneur qui sont créés en même temps que le conteneur et restent affichés en permanence.*

Pour dessiner, on utilise, en général, un conteneur particulier appelé **panneau** de type `JPanel`.

```
java.lang.Object
|
+--java.awt.Component*
|
+--java.awt.Container*
|
+--javax.swing.JComponent*
|
+--javax.swing.JPanel
```

panneau utiliser pour dessiner

I.9.1. Création d'un panneau

Un panneau de type `JPanel` est un composant intermédiaire qui peut être contenu dans un conteneur et qui peut contenir d'autres composants.

Un panneau est une sorte de "sous-fenêtre", sans titre ni bordure, qui doit obligatoirement être associé par la méthode `add` (de la classe `Container`) à un autre conteneur, généralement une fenêtre.

Exemple

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

class MaFenetre extends JFrame {
    private JPanel panneau ;
    public MaFenetre () {
        super("Une fenetre avec un panneau jaune") ;
        setSize(300, 200) ;
        panneau = new JPanel();
        panneau.setBackground(Color.yellow) ;
        //Color.yellow est une constante de la classe Color
        (classe du paquetage java.awt dérivée de la classe
        Object) correspondant à la couleur jaune
        getContentPane().add(panneau) ;
        //le panneau de couleur jaune occupe toute la fenêtre
    }
}

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

Le gestionnaire de mise en forme `FlowLayout` est le gestionnaire par défaut des panneaux.

I.9.2. Dessin dans un panneau

Pour obtenir un dessin permanent dans un composant, il faut redéfinir sa méthode `paintComponent` (méthode de la classe `JComponent`), qui sera appelée chaque fois que le composant aura besoin d'être redessiné. Cette méthode a l'en-tête suivant :

```
void paintComponent (Graphics g)
```

Son unique argument `g` de type `Graphics` est le **contexte graphique** du composant qui a appelé la méthode. Il sert d'intermédiaire entre les demandes de dessin et leur réalisation effective sur le composant.

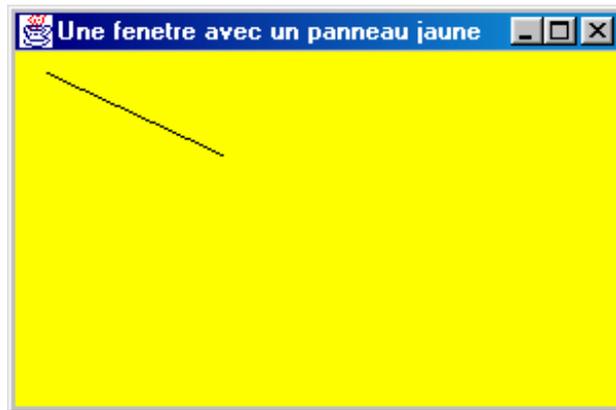
Exemple

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

```
class MaFenetre extends JFrame {  
    private JPanel pan ;  
    public MaFenetre () {  
        super("Une fenetre avec un panneau jaune") ;  
        setSize(300, 200) ;  
        pan = new Panneau();  
        pan.setBackground(Color.yellow) ;  
        getContentPane().add(pan) ;  
    }  
}
```

```
class Panneau extends JPanel {  
    public void paintComponent(Graphics g) {  
        super.paintComponent(g) ;  
        //appel explicite de la méthode paintComponent de la  
        classe de base JPanel , qui réalise le dessin du  
        panneau.  
        g.drawLine(15, 10, 100, 50) ;  
        //méthode de la classe Graphics qui trace un trait du  
        point de coordonnées (15, 10) au point de coordonnées  
        (100, 50)  
    }  
}
```

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



La méthode `repaint` (de la classe `Component`) permet d'appeler la méthode `paintComponent`.

Exemple

```
Panneau pan ;
```

```
pan.repaint() ;
```

//appel de la méthode `PaintComponent` de l'objet `pan` de type `Panneau`

I.9.3. La classe Graphics

La classe `Graphics` est une classe abstraite du paquetage `java.awt` dérivée de la classe `Object`. Cette classe dispose de nombreuses méthodes pour dessiner sur un composant et gère des paramètres courants tels que la couleur de fond, la couleur de trait, le style de trait, la police de caractères, la taille des caractères...

La méthode `paintComponent` fournit automatiquement en argument le contexte graphique du composant qui l'a appelée.

a. Paramétrer la couleur du contexte graphique

En Java, une couleur est représenté par un objet de type `Color` (classe du paquetage `java.awt` dérivée de la classe `Object`). Les constantes prédéfinies de la classe `Color` sont : `Color.black`, `Color.white`, `Color.blue`, `Color.cyan`, `Color.darkGray`, `Color.gray`, `Color.lightGray`, `Color.green`, `Color.magenta`, `Color.orange`, `Color.pink`, `Color.red`, `Color.yellow`.

Par défaut, dans la méthode `paintComponent` d'un composant, la couleur courante du contexte graphique correspondant est la couleur d'avant plan du composant (éventuellement modifiée par la méthode `setForeground` de la classe `Component` du composant). Dans la méthode `paintComponent` d'un composant, on peut :

 récupérer la couleur du contexte graphique correspondant

 public abstract `Color` `getColor()`

 modifier la couleur du contexte graphique correspondant

 public abstract `setColor(Color c)`

Les méthodes `getColor` et `setColor` sont des méthodes de la classe `Graphics`.

b. Paramétrer la police du contexte graphique

D'une manière générale, à un instant donné, un composant dispose d'une **police** courante de type **Font** (classe du paquetage `java.awt` dérivée de la classe `Object`) qui se définit par :

- un nom de famille de police (`SansSerif`, `Serif`, `Monospaced`, `Dialog`, `DialogInput`) ;
- un style : style normal (`Font.PLAIN`), style gras (`Font.BOLD`), style italique (`Font.ITALIC`), et style gras et italique (`Font.BOLD+Font.ITALIC`) ;
- une taille exprimée en points typographiques et non pas en pixels.

Exemple

```
Font f = new Font("Serif", Font.BOLD, 18) ;
```

Dans la méthode `paintComponent` d'un composant, on peut :

- recupérer la police du contexte graphique correspondant
`public abstract Font getFont()`
- modifier la police du contexte graphique correspondant
`public abstract setFont(Font f)`

Les méthodes `getFont` et `setFont` sont des méthodes de la classe `Graphics`.

c. Méthodes de dessin de la classe Graphics

```
public abstract void drawLine(int x1, int y1, int x2, int y2)
    //trace un trait du point de coordonnées (x1, y1) au point de
    //coordonnées (x2, y2)
```

```
public void drawRect(int x, int y, int largeur, int hauteur)
    //dessine un rectangle de largeur largeur et de hauteur
    //hauteur au point de coordonnées (x, y)
```

```
public void drawOval(int x, int y, int largeur, int hauteur)
    //dessine un ovale de largeur largeur et de hauteur hauteur
    //au point de coordonnées (x, y)
```

```
public abstract void drawstring(String str, int x, int y)
    //dessine le texte str au point de coordonnées (x, y)
```

✱ *Toutes ces méthodes tracent des lignes en utilisant la couleur du contexte graphique correspondant.*

I.9.4. Affichage d'images

Java sait traiter deux formats de stockage d'images : le format GIF (Graphical Interface Format) avec 256 couleurs disponibles et le format JPEG (Joint Photographic Expert Group) avec plus de 16 millions de couleurs disponibles.

Le chargement d'une image se fait en trois étapes.

Tout d'abord, le constructeur de la classe `ImageIcon` (classe du paquetage `javax.swing` dérivée de la classe `Object`) permet de charger une image.

Ensuite, la méthode `getImage` permet d'obtenir un objet de type `Image` à partir d'un objet de type `ImageIcon`. La méthode `getImage` de la classe `Toolkit` (classe du paquetage `java.awt` dérivée de la classe `Object`) permet de charger une image depuis un fichier local. La méthode `getImage` de la classe `Applet` permet de charger une image depuis un site distant. L'argument de cette méthode est un objet de type `URL` (classe du paquetage `java.net` dérivée de la classe `Object`), qui permet de représenter une adresse dans le réseau Internet (`URL` pour Uniform Reference Location).

Enfin, la méthode `drawImage` de la classe `Graphics` permet d'afficher une image de type `Image` (classe du paquetage `java.awt` dérivée de la classe `Object`).

La méthode `getImage` n'attend pas que le chargement de l'image soit effectué pour rendre la main au programme. Il se pose alors le problème de l'affichage de l'image.

Afin de ne pas voir afficher qu'une partie de l'image, Java fournit en quatrième argument de la méthode `drawImage` de la classe `Graphics` la référence à un objet particulier appelé **observateur**. Cet objet implémente l'interface `ImageObserver` comportant une méthode `imageUpdate` qui est appelée chaque fois qu'une nouvelle portion de l'image est disponible.

Tous les composants implémentent l'interface `ImageObserver` et fournissent une méthode `imageUpdate` qui appelle, par défaut, la méthode `repaint`. Ainsi, pour régler le problème de l'affichage de l'image, il suffira de fournir `this` en quatrième argument de la méthode `drawImage`.

Exemple

```
class MaFenetre extends JFrame {
    private JPanel pan ;
    public MaFenetre () {
        super("Une fenetre avec une image") ;
        setSize(300, 200) ;
        pan = new Panneau();
        getContentPane().add(pan) ; }
}

class Panneau extends JPanel {
    private ImagemIcon rouge;
    public Panneau() {
        rouge = new ImagemIcon("rouge.gif") ;
        //chargement d'une image dans l'objet de référence
        rouge, à partir du fichier "rouge.gif"
    }
    public void paintComponent(Graphics g) {
        super.paintComponent(g) ;
        Image imRouge = rouge.getImage();
        //la méthode getImage retourne une référence à un
        objet de type Image à partir d'un objet de type
        ImagemIcon
        g.drawImage(imRouge, 15, 10, this) ;
        //affiche l'image imRouge au point de coordonnées
        (15, 10)
    }
}
```

II. La gestion des événements

II.1. Introduction

Un clic souris, la frappe d'une touche au clavier ou le changement de la taille d'une fenêtre sont des exemples d'événements.

✱ *Java classe les événements en deux niveaux : les événements de bas niveau (par exemple, un clic dans une fenêtre) et les événements de haut niveau (par exemple, une action sur un bouton qui peut provenir d'un clic souris ou d'une frappe au clavier).*

En Java, les événements n'ont pas une valeur physique, mais logique. Un événement dépend du composant qui l'a généré. On appelle **source** d'un événement l'objet qui l'a généré.

Exemple

L'événement émis suite à un clic souris dans une fenêtre est de type `MouseEvent`. .

L'événement émis suite à un clic souris sur un bouton est de type `ActionEvent`. ...

Tout événement qui peut se produire dans une interface graphique est de type `XXXEvent`, classe du paquetage `java.awt.event` ou du paquetage `javax.swing.event` dérivée de la classe `EventObject` (classe du paquetage `java.util` dérivée de la classe `Object`). ..

✱ *Afin d'éviter d'avoir à s'interroger sur la répartition dans les paquetages des différentes classes utilisées dans les interfaces graphiques, nous importerons systématiquement toutes les classes des paquetages `java.awt`, `java.awt.event`, `javax.swing` et `javax.swing.event`.*

II.2.Traiter un événement

Un composant ne traite pas forcément lui même les événements qu'il génère. Il délègue ce traitement à des objets particuliers appelés **écouteurs** (un composant peut être son propre écouteur).

En fonction des événements qu'ils traitent, un écouteur doit implémenter une interface particulière, dérivée de l'interface `EventListener`, qui correspond à une catégorie d'événements. Pour traiter un événement de type `XXXEvent`, un écouteur doit implémenter l'interface `XXXListener`.

Exemple

L'interface `MouseListener` correspond à une catégorie d'événements **souris** de type `MouseEvent`. Elle comporte cinq méthodes correspondant chacune à un événement souris particulier..

```
public interface MouseListener extends EventListener {
    public void mousePressed(MouseEvent e) ;
        //appelé lorsqu'un bouton de la souris est pressé sur un
        //composant
        //l'argument e de type MouseEvent correspond à
        //l'objet événement généré
    public void mouseReleased(MouseEvent e) ;
        //appelé lorsqu'un bouton de la souris est relâché sur un
        //composant
    public void mouseClicked(MouseEvent e) ;
        //appelé lors d'un clic souris sur un composant (la souris
        //n'a pas été déplacée entre l'appui et le relâchement du
        //bouton)
    public void mouseEntered(MouseEvent e) ;
        //appelé lorsque la souris passe de l'extérieur à
        //l'intérieur d'un composant
    public void mouseExited(MouseEvent e) ; }
        //appelé lorsque la souris sort d'un composant (la souris
        //passe de l'intérieur à l'extérieur du composant)
```

II.3. Intercepter un événement

Lorsqu'un composant veut intercepter un événement de type `XXXEvent`, il doit le préciser dans son constructeur en appelant la méthode `addXXXListener(XXXListener objetEcouteur)`, où l'argument `objetEcouteur` correspond à l'objet écouteur chargé de traiter l'événement.

Pour savoir quels événements sont susceptibles d'être générés par un composant donné, il faut rechercher toutes les méthodes de la forme `addXXXListener` définies dans la classe du composant et dans ses classes ascendantes.

Exemple

La classe `Component` définit notamment les méthodes suivantes :

```
public void addFocusListener(FocusListener l) ;  
    //prise et perte du focus (à un moment donné, un seul  
    //composant est sélectionné, on dit qu'il a le focus)  
public void addKeyListener(KeyListener l) ;  
    //événements clavier  
public void addMouseListener(MouseListener l) ;  
    //événements souris  
public void addMouseMotionListener(MouseMotionListener l) ;  
    //événements liés au déplacement de la souris
```

II.4.Un premier exemple

II.4.1. Première version

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

class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre qui traite les clics souris") ;
        setSize(300, 200) ;
        addMouseListener(new EcouteurSouris());
        //la fenêtre fait appel à un écouteur d'événements souris
        pour traiter les clics souris
    }
}

//L'écouteur d'événements souris doit implémenter l'interface
MouseListener qui correspond à une catégorie
d'événements souris.
class EcouteurSouris implements MouseListener {
    //redéfinition de la méthode appelée lors d'un clic souris
    public void mouseClicked(MouseEvent e) {
        System.out.println("clic dans la fenetre"); }
    //la redéfinition des autres méthodes est "vide"
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
}

public class MonProgEvtClic1 {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ; }
}
```

II.4.2. Deuxième version

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

class MaFenetre extends JFrame implements
MouseListener {
    public MaFenetre () {
        super("Une fenetre qui traite les clics souris") ;
        setSize(300, 200) ;
        addMouseListener(this);
        //la fenetre est son propre écouteur d'événements souris
    }
    //L'argument e de type MouseEvent correspond à l'objet
    événement généré dans la fenetre lors d'un clic souris. On
    peut utiliser les informations qui lui sont associées.
    public void mouseClicked(MouseEvent e) {
        int x = e.getX() ;
        int y = e.getY() ;
        //coordonnées du curseur de la souris au moment du clic
        System.out.println("clic dans la fenetre au point de
        coordonnees " + x + ", " + y);
    }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
}

public class MonProgEvtClic2 {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ;
    }
}
```

II.5. La notion d'adaptateur

Pour chaque interface XXXListener **possédant plusieurs méthodes**, Java fournit une classe particulière XXXAdapter, appelée **adaptateur**, qui implémente toutes les méthodes de l'interface avec un corps vide.

Pour définir un écouteur d'événements de type XXXEvent, il suffit alors de dériver l'écouteur de la classe XXXAdapter et de redéfinir uniquement les méthodes voulues.

Exemple

```
class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre qui traite les clics souris") ;
        setSize(300, 200) ;
        addMouseListener(new EcouteurSouris());
    }
}

class EcouteurSouris extends MouseAdapter {
    //redéfinition uniquement de la méthode appelée lors
    //d'un clic souris
    public void mouseClicked(MouseEvent e) {
        System.out.println("clic dans la fenetre") ;
    }
}
```

II.6. Récapitulons

Pour traiter un événement de type inconnu (par exemple la fermeture d'une fenêtre) généré par une source, les étapes à suivre sont :

Rechercher les méthodes de la forme `addXXXListener` définies dans la classe de la source ou une de ses classes ascendantes.

Identifier l'interface `XXXListener` qui convient en regardant ses méthodes ; ce qui permet d'identifier le type `XXXEvent` de l'événement à traiter.

Exemple

L'interface `WindowListener` définit la méthode `windowClosing` appelée lors de la fermeture d'une fenêtre (la méthode `addWindowListener` est une méthode de la classe `Window`, classe ascendante de la classe `JFrame`).

Définir un écouteur pour traiter l'événement.

L'objet source est son propre écouteur, il doit implémenter l'interface `XXXListener` adéquate.

L'écouteur est une classe indépendante qui implémente l'interface `XXXListener` adéquate.

L'écouteur est une classe indépendante qui dérive de la classe `XXXAdapter` associée à l'interface `XXXListener` adéquate.

Implémenter la ou les méthodes de l'interface `XXXListener` qui nous intéressent. Les informations contenues dans l'événement passé en paramètre de ces méthodes pourront être exploitées.

Exemple

```
class MaFenetre extends JFrame {  
    public MaFenetre () {  
        super("Une fenetre qui gere sa fermeture") ;  
    }  
}
```

```
        setSize(300, 200) ;  
        addWindowListener(new EcouteurFermer());  
    }  
}
```

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

II.7. Un exemple avec des boutons

Un bouton ne peut déclencher qu'un seul événement de type `ActionEvent`. L'interface `ActionListener` ne comporte qu'une seule méthode `actionPerformed`.

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

class MaFenetre extends JFrame implements
ActionListener {
    private JButton MonBouton1, MonBouton2 ;
    public MaFenetre () {
        super("Une fenetre avec deux boutons") ;
        setSize(300, 200) ;
        Container contenu = getContentPane() ;
        MonBouton1 = new JButton("Bouton 1") ;
        contenu.add(MonBouton1) ;
        MonBouton2 = new JButton("Bouton 2") ;
        contenu.add(MonBouton2) ;
        //un même événement peut être traité par plusieurs
        //écouteurs : deux écouteurs sont associés à l'action de
        //l'utilisateur sur le bouton MonBouton1
        MonBouton1.addActionListener(this);
        //gère l'action de l'utilisateur sur le bouton
        //MonBouton1
        MonBouton1.addActionListener(new EcouteurFermer());
        //gère la fermeture de la fenêtre lors d'une action de
        //l'utilisateur sur le bouton MonBouton1
        MonBouton2.addActionListener(this);
        //gère l'action de l'utilisateur sur le bouton
        //MonBouton2
    }
    public void actionPerformed(ActionEvent e) {
        //utilisation de la méthode getSource de la classe
        //EventObject qui fournit une référence de type Object sur
        //la source de l'événement concerné
    }
}
```

```

    if(e.getSource() == MonBouton1)
        //conversion implicite du type JButton en un type
        //ascendant Object
        System.out.println("action sur le bouton 1") ;
    if(e.getSource() == MonBouton2)
        System.out.println("action sur le bouton 2") ;
    }
}

class EcouteurFermer implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}

public class MonProgEvtBouton1 {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ; }
}

```

La méthode `getSource` de la classe `EventObject` permet d'identifier la source d'un événement. Elle s'applique à tous les événements générés par n'importe quel composant.

La méthode `getActionCommand` de la classe `ActionEvent` permet d'obtenir la **chaîne de commande** associée à la source d'un événement. Les composants qui disposent d'une chaîne de commande sont les boutons, les cases à cocher, les boutons radio et les options menu. Par défaut, la chaîne de commande associée à un bouton est son étiquette.

Exemple

La méthode `actionPerformed` de la classe `MaFenetre` peut s'écrire :

```

public void actionPerformed(ActionEvent e) {
    String nom = e.getActionCommand() ;
    System.out.println("action sur le " + nom) ;}

```

La méthode `setActionCommand` de la classe `JButton` permet d'associer une autre chaîne de commande à un bouton.

Exemple

```
MonBouton1.setActionCommand ("premier bouton").
```

II.8. Un exemple de création dynamique de boutons

Jusqu'à présent, les composants d'un conteneur étaient créés en même temps que le conteneur et restaient affichés en permanence. Cependant, on peut, à tout moment, ajouter un nouveau composant à un conteneur (grâce à la méthode `add` de la classe `Container`) ou supprimer un composant d'un conteneur (grâce à la méthode `remove` de la classe `Container`). Si l'une de ses opérations est effectuée après l'affichage du conteneur, il faut forcer le gestionnaire de mise en forme à recalculer les positions des composants du conteneur (1) soit en appelant la méthode `validate` (de la classe `Component`) du conteneur, (2) soit en appelant la méthode `revalidate` (de la classe `JComponent`) des composants du conteneur.

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

class MaFenetre extends JFrame {
    private JButton MonBouton ;
    public MaFenetre () {
        super("Une fenetre avec des boutons dynamiques") ;
        setSize(300, 200) ;
        Container contenu = getContentPane() ;
        contenu.setLayout(new FlowLayout()) ;
        MonBouton = new JButton("Creation de boutons") ;
        contenu.add(MonBouton) ;
        MonBouton.addActionListener(new EcouteurBouton
            (contenu)) ;
    }
}
```

```
class EcouteurBouton implements ActionListener {
    private Container contenu ;
    public EcouteurBouton (Container contenu) {
        this.contenu = contenu ;
    }
    public void actionPerformed(ActionEvent e) {
        JButton NvBouton = new JButton("Bouton") ;
        contenu.add(NvBouton) ;
        contenu.validate() ;
        //recalcul les positions des composants du conteneur
    }
}

public class MonProgEvtBouton2 {
    public static void main(String args[]) {
        JFrame fen = new MaFenetre() ;
        fen.setVisible(true) ;
    }
}
```

II.9. Les classes internes et anonymes

Les classes internes et les classes anonymes sont souvent utilisées dans la gestion des événements des interfaces graphiques.

II.9.1. Les classes internes

Une **classe** est dite **interne** lorsque sa définition est située à l'intérieure d'une autre classe (ou d'une méthode).

Exemple

```
class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre qui gere sa fermeture") ;
        setSize(300, 200) ;
        addWindowListener(new EcouteurFermer());
    }
    //classe interne à la classe MaFenetre
    class EcouteurFermer extends WindowAdapter {
        public void windowClosing(WindowEvent e) {
            System.exit(0); }
    } //fin classe interne
}
```

Un objet d'une classe interne est toujours associé, au moment de son instanciation, à un objet d'une classe externe dont on dit qu'il lui a donné naissance.

Un objet d'une classe interne a toujours accès aux champs et méthodes (même privés) de l'objet externe lui ayant donné naissance.

Un objet d'une classe externe a toujours accès aux champs et méthodes (même privés) de l'objet interne auquel il a donné naissance.

Exemple

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

class Cercle {
    private Centre c ;
    private double r ;
    class Centre {          //classe interne à la classe Cercle
        private int x, y ;
        public Centre(int x, int y) {
            this.x = x ; this.y = y ; }
        public void affiche() {
            System.out.println("( " + x + ", " + y + ")") ; }
    }          //fin classe interne
    public Cercle(int x, int y, double r) {
        c = new Centre(x, y) ;
        //création d'un objet de type Centre, associé à l'objet
        //de type Cercle lui ayant donné naissance (celui qui a
        //appelé le constructeur)
        this.r = r ; }
    public void affiche() {
        System.out.println("Cercle de rayon " + r + " et de
        centre ") ;
        c.affiche() ; } // méthode affiche de la classe Centre
    public void deplace(int dx, int dy) {
        c.x += dx ; c.y += dy ; }
        //accès aux champs privés x et y de la classe Centre
    }

    public class MonProgCercle {
        public static void main(String args[]) {
            Cercle c1 = new Cercle(1, 3, 2.5) ; c1.affiche() ;
            c1.deplace(4, -2) ; c1.affiche() ; }
    }
```

Cercle de rayon 2.5 et de centre (1, 3)

Cercle de rayon 2.5 et de centre (5, 1)

II.9.2. Les classes anonymes

Une **classe anonyme** est une classe sans nom. Elle peut dériver d'une autre classe.

Exemple

```
class MaFenetre extends JFrame {
    public MaFenetre () {
        super("Une fenetre qui gere sa fermeture") ;
        setSize(300, 200) ;
        addWindowListener(new WindowAdapter() {
            //classe anonyme dérivant de la classe
            WindowAdapter
            public void windowClosing(WindowEvent e) {
                System.exit(0); }
        }); //fin classe anonyme
    }
}
```

Une classe anonyme peut également implémenter une interface.

Exemple

```
class MaFenetre extends JFrame {
    private JButton UnBouton ;
    public MaFenetre () {
        super("Une fenetre avec un bouton") ;
        setSize(300, 200) ;
        UnBouton = new JButton("Un bouton") ;
        getContentPane().add(UnBouton) ;
        UnBouton.addActionListener(new ActionListener(){
            //classe anonyme implémentant l'interface
            ActionListener
            public void actionPerformed(ActionEvent e) {
                System.exit(0); }
        }); //fin classe anonyme
    }
}
```

Bibliographie

Claude Delannoy. *Programmer en Java*. Eyrolles, 2000.

Cay S. Horstmann et Gary Cornell. *Au cœur de Java 2. Volume II : Fonctions avancées*. CampusPress France, 2000.

Gilles Clavel, Nicolas Mirouze, Emmanuel Pichon et Mohamed Soukal. *Java, la synthèse*. InterEditions, 1997.

Laura Lemay et Charles L. Perkins. *Le programmeur Java*. Simon & Schuster Macmillan (France), 1996.

[http ://www.java.sun.com](http://www.java.sun.com)

Index

C

classe

| | |
|------------------------------|----------------|
| ActionEvent | 63, 72, 73 |
| BorderLayout | 42, 43 |
| BoxLayout | 49 |
| ButtonGroup | 12, 24 |
| CardLayout | 47 |
| Color | 58 |
| Component | 8, 66 |
| Container | 5, 7, 42, 51 |
| Dimension | 9 |
| EventObject | 63, 73 |
| FlowLayout | 45, 54 |
| Font | 59 |
| Graphics | 55, 57 |
| GridBagLayout | 49 |
| GridLayout | 48 |
| Image | 61 |
| ImageIcon | 29, 61 |
| Insets | 51 |
| JButton | 5, 10, 72, 74 |
| JCheckBox | 11 |
| JCheckBoxMenuItem | 24 |
| JComboBox | 18 |
| JComponent | 10 |
| JDialog | 31, 40 |
| JFrame | 5, 6, 7 |
| JLabel | 14 |
| JList | 16 |
| JMenu | 21, 22, 26 |
| JMenuBar | 21, 22 |
| JMenuItem | 21, 22, 24, 26 |
| JOptionPane | 31 |
| JPanel | 53, 54 |
| JPopupMenu | 27 |
| JRadioButton | 12 |
| JRadioButtonMenuItem | 24 |
| JScrollPane | 17 |
| JTextField | 15 |
| JToolBar | 28 |
| MouseEvent | 63 |
| URL | 61 |
| classe (interface graphique) | |
| ActionEvent | 63, 72, 73 |
| BorderLayout | 42, 43 |
| BoxLayout | 49 |
| ButtonGroup | 12, 24 |
| CardLayout | 47 |
| Color | 58 |
| Component | 8, 66 |
| Container | 5, 7, 42, 51 |
| Dimension | 9 |

Index

| | |
|----------------------------|----------------|
| EventObject | 63, 73 |
| FlowLayout | 45, 54 |
| Font | 59 |
| Graphics | 55, 57 |
| GridBagLayout | 49 |
| GridLayout | 48 |
| Image | 61 |
| ImageIcon | 29, 61 |
| Insets | 51 |
| JButton | 5, 10, 72, 74 |
| JCheckBox | 11 |
| JCheckBoxMenuItem | 24 |
| JComboBox | 18 |
| JComponent | 10 |
| JDialog | 31, 40 |
| JFrame | 5, 6, 7 |
| JLabel | 14 |
| JList | 16 |
| JMenu | 21, 22, 26 |
| JMenuBar | 21, 22 |
| JMenuItem | 21, 22, 24, 26 |
| JOptionPane | 31 |
| JPanel | 53, 54 |
| JPopupMenu | 27 |
| JRadioButton | 12 |
| JRadioButtonMenuItem | 24 |
| JScrollPane | 17 |
| JTextField | 15 |
| JToolBar | 28 |
| MouseEvent | 63 |
| URL | 61 |
| classe anonyme | 79 |
| classe interne | 77 |
| contexte graphique | 55, 57 |

E

| | |
|-------------------------------------------------|------------|
| événement | 63, 70 |
| adaptateur | 69 |
| écouteur | 65, 69 |
| intercepter | 66 |
| source | 63 |
| traiter | 65 |
| événement bouton | |
| exemple | 72, 75 |
| événement fenêtre | |
| exemple | 71 |
| interface WindowListener | 70 |
| méthode addWindowListener(WindowListener) | 70 |
| méthode windowClosing(WindowEvent) | 70 |
| événement souris | |
| classe MouseEvent | 63 |
| exemple | 67, 68, 69 |
| interface MouseListener | 65 |
| méthode addMouseListener(MouseListener) | 66 |

G

| | |
|-------------------------------------|----|
| gestionnaire de mise en forme | 42 |
|-------------------------------------|----|

I

| | |
|-----------|--|
| interface | |
|-----------|--|

Index

| | |
|---------------------------------|----|
| ActionListener | 72 |
| EventListener | 65 |
| ImageObserver | 61 |
| LayoutManager | 42 |
| MouseListener | 65 |
| WindowListener | 70 |
| interface (interface graphique) | |
| ActionListener | 72 |
| EventListener | 65 |
| ImageObserver | 61 |
| LayoutManager | 42 |
| MouseListener | 65 |
| WindowListener | 70 |

M

méthode

| | |
|---------------------------------------------------|--------------------|
| actionPerformed(ActionEvent) | 72, 73 |
| add(Component) | 7, 43, 75 |
| add(Component,int) | 43 |
| add(Component,String) | 43, 47 |
| addFocusListener(FocusListener) | 66 |
| addItem(Object) | 19 |
| addItemAt(Object,int) | 19 |
| addKeyListener(KeyListener) | 66 |
| addMouseListener(MouseListener) | 66 |
| addMouseMotionListener(MouseMotionListener) | 66 |
| addSeparator() | 22 |
| addWindowListener(WindowListener) | 70 |
| drawImage(Image,int,int,ImageObserver) | 61 |
| drawLine(int,int,int,int) | 55, 60 |
| drawOval(int,int,int,int) | 60 |
| drawRect(int,int,int,int) | 60 |
| drawString(String,int,int) | 60 |
| getActionCommand() | 73 |
| getColor() | 58 |
| getContentPane() | 5, 7 |
| getFont() | 59 |
| getImage() | 61 |
| getImage(String) | 61 |
| getImage(URL) | 61 |
| getInsets() | 51 |
| getSelectedIndex() | 19 |
| getSelectedItem() | 19 |
| getSelectedValue() | 17 |
| getSelectedValues() | 17 |
| getSize() | 9 |
| getSource() | 73 |
| getText() | 15 |
| isEnabled() | 8 |
| isSelected() | 11, 13, 25 |
| mouseClicked(MouseEvent) | 65 |
| mouseEntered(MouseEvent) | 65 |
| mouseExited(MouseEvent) | 65 |
| mousePressed(MouseEvent) | 65 |
| mouseReleased(MouseEvent) | 65 |
| paintComponent(Graphics) | 53, 55, 57, 58, 59 |
| remove(Component) | 7, 75 |
| removeItem(Object) | 19 |
| repaint() | 56, 62 |
| revalidate() | 75 |

Index

| | |
|---------------------------------------------|------------|
| setActionCommand(String) | 74 |
| setBackground(Color) | 8 |
| setBounds(int,int,int,int) | 5, 9, 50 |
| setColor(Color) | 58 |
| setEditable(boolean) | 19 |
| setEnabled(boolean) | 8, 21 |
| setFloatable(boolean) | 28 |
| setFont(Font) | 59 |
| setForeground(Color) | 8, 58 |
| setJMenuBar(JMenuBar) | 22 |
| setLayout(LayoutManager) | 42, 50 |
| setMaximumRowCount(int) | 18 |
| setPreferredSize(Dimension) | 10, 44, 46 |
| setSelected(boolean) | 11, 13 |
| setSelectedIndex(int) | 16, 18 |
| setSelectionMode(int) | 16 |
| setSize(Dimension) | 9 |
| setSize(int,int) | 8 |
| setText(String) | 14 |
| setTitle(String) | 5, 7 |
| setToolTipText(String) | 30 |
| setVisible(boolean) | 5, 8 |
| setVisibleRowCount(int) | 17 |
| show(Component,int,int) | 27 |
| showConfirmDialog | 34 |
| showInputDialog | 36, 38 |
| showMessageDialog | 32 |
| validate() | 75 |
| windowClosing(WindowEvent) | 70 |
| méthode (interface graphique) | |
| actionPerformed(ActionEvent) | 72, 73 |
| add(Component) | 7, 43, 75 |
| add(Component,int) | 43 |
| add(Component,String) | 43, 47 |
| addFocusListener(FocusListener) | 66 |
| addItem(Object) | 19 |
| addItemAt(Object,int) | 19 |
| addKeyListener(KeyListener) | 66 |
| addMouseListener(MouseListener) | 66 |
| addMouseMotionListener(MouseMotionListener) | 66 |
| addSeparator() | 22 |
| addWindowListener(WindowListener) | 70 |
| drawImage(Image,int,int,ImageObserver) | 61 |
| drawLine(int,int,int,int) | 55, 60 |
| drawOval(int,int,int,int) | 60 |
| drawRect(int,int,int,int) | 60 |
| drawString(String,int,int) | 60 |
| getActionCommand() | 73 |
| getColor() | 58 |
| getContentPane() | 5, 7 |
| getFont() | 59 |
| getImage() | 61 |
| getImage(String) | 61 |
| getImage(URL) | 61 |
| getInsets() | 51 |
| getSelectedIndex() | 19 |
| getSelectedItem() | 19 |
| getSelectedValue() | 17 |
| getSelectedValues() | 17 |
| getSize() | 9 |

Index

| | |
|-----------------------------|--------------------|
| getSource() | 73 |
| getText() | 15 |
| isEnabled() | 8 |
| isSelected() | 11, 13, 25 |
| mouseClicked(MouseEvent) | 65 |
| mouseEntered(MouseEvent) | 65 |
| mouseExited(MouseEvent) | 65 |
| mousePressed(MouseEvent) | 65 |
| mouseReleased(MouseEvent) | 65 |
| paintComponent(Graphics) | 53, 55, 57, 58, 59 |
| remove(Component) | 7, 75 |
| removeItem(Object) | 19 |
| repaint() | 56, 62 |
| revalidate() | 75 |
| setActionCommand(String) | 74 |
| setBackground(Color) | 8 |
| setBounds(int,int,int,int) | 5, 9, 50 |
| setColor(Color) | 58 |
| setEditable(boolean) | 19 |
| setEnabled(boolean) | 8, 21 |
| setFloatable(boolean) | 28 |
| setFont(Font) | 59 |
| setForeground(Color) | 8, 58 |
| setJMenuBar(JMenuBar) | 22 |
| setLayout(LayoutManager) | 42, 50 |
| setMaximumRowCount(int) | 18 |
| setPreferredSize(Dimension) | 10, 44, 46 |
| setSelected(boolean) | 11, 13 |
| setSelectedIndex(int) | 16, 18 |
| setSelectionMode(int) | 16 |
| setSize(Dimension) | 9 |
| setSize(int,int) | 8 |
| setText(String) | 14 |
| setTitle(String) | 5, 7 |
| setToolTipText(String) | 30 |
| setVisible(boolean) | 5, 8 |
| setVisibleRowCount(int) | 17 |
| show(Component,int,int) | 27 |
| showConfirmDialog | 34 |
| showInputDialog | 36, 38 |
| showMessageDialog | 32 |
| validate() | 75 |
| windowClosing(WindowEvent) | 70 |