

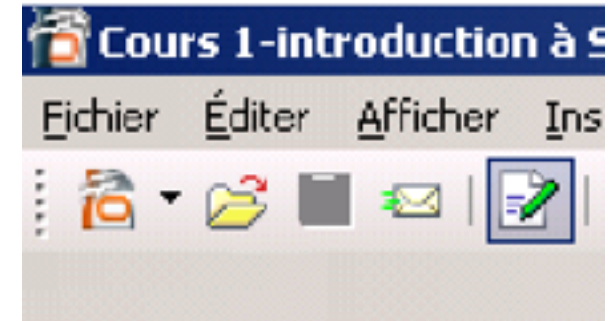
Java et les interfaces graphiques

Introduction :

Une IG : intermédiaire entre l'utilisateur et la partie «métier» d'un logiciel.

- ce qui relève de l'IG:

- * gérer le fait que le bouton soit actif ou non (idem pour la commande "Enregistrer")
- * lancer la sauvegarde quand on clique



- ce qui ne relève pas de l'IG:

- * la sauvegarde elle-même!
- Dans JAVA, l'interface utilisateur se base sur des composants
- Il en existe deux types :
 - * composants lourds (telle que la fenêtre)
 - dépendent du système
 - * composants légers
 - entièrement écrits en JAVA

- La notion de composant

De façon générale, un composant est un bout de code réutilisable. Pour cela, il doit respecter un certain nombre de conventions. Dans le cadre des interfaces graphiques, un composant est un élément graphique (qui va effectivement être réutilisé dans toutes les applications graphiques).

- Composants légers vs composants lourds

On parle de composant lourd lorsqu'on utilise l'API du système hôte pour les créer, ce que fait AWT. Ainsi, lorsqu'on crée un bouton AWT, cela entraîne un appel à la librairie graphique du système créant un bouton natif. Un composant léger, comme en Swing, est créé de toute pièce par l'API Java sans faire appel à son correspondant du système hôte.

Ce fut un plus gros travail pour les concepteurs de l'API Swing mais cela permet d'avoir des applications qui ont exactement le même «look» quelle que soit la machine sur laquelle on la tourne.

- Packages IHM en JAVA

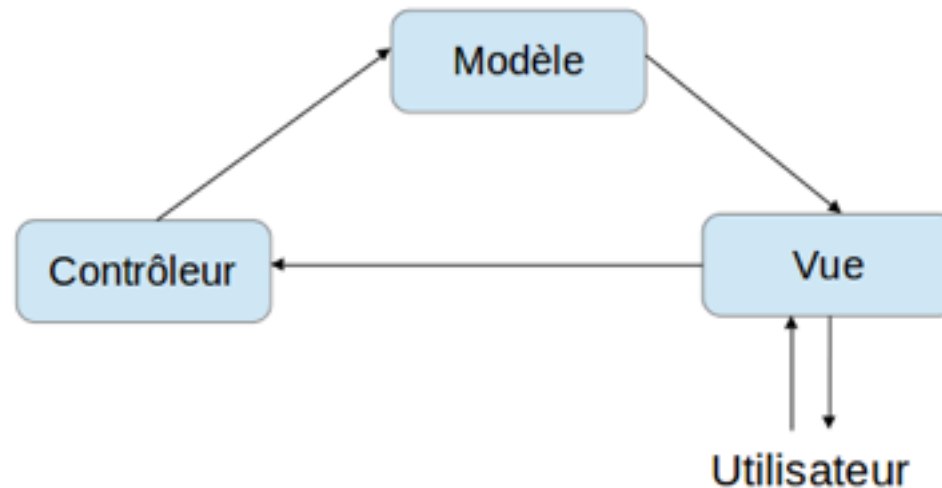
* `java.awt` : (awt : abstract window toolkit, java1.1) bas niveau. Anciens composants, dits composants lourds (ils sont opaques et affichés en dernier).

* `javax.swing` : java 1.2. Composants légers = pas de code natif (écrits entièrement en java). On préférera `JButton` à `Button` par exemple.

Modèle MVC : Modèle\Vue \Contrôleur

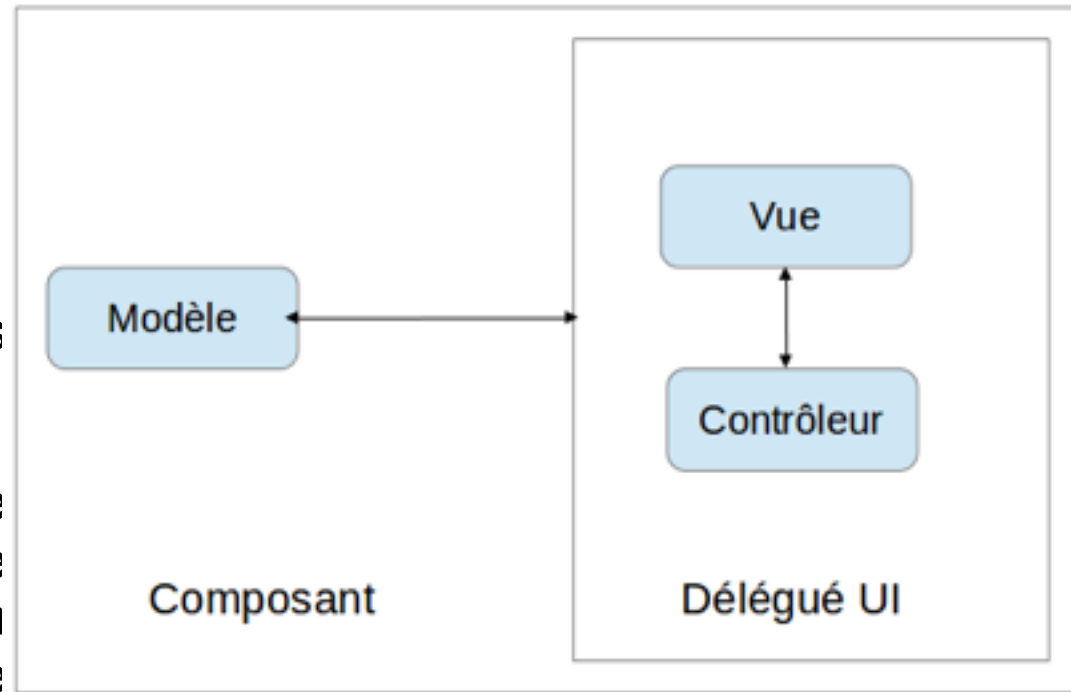
La conception des classes Swing s'appuie sur l'architecture MVC.

- On y distingue pour un composant:
 - * Le Modèle qui gère et stocke les données.
 - * Une (ou des) Vue(s) qui implante(nt) une représentation (visuelle) à partir du modèle.
 - * Le Contrôleur qui se charge des interactions avec l'utilisateur et modifie le modèle (et la ou les vues).



Architecture des composants Swing

- Les composants (légers) de Swing comprennent
 - un modèle
 - un délégué Interface Utilisateur (UI *delegate*)
- Les Contrôleurs sont ici des écouteurs d'événements
- Le délégué UI est responsable de l'apparence visuelle du composant et de ses interactions avec l'utilisateur. Avec un délégué UI différent on obtient une nouvelle apparence.



Un modèle peut comporter plusieurs vues. Un composant peut aussi avoir plusieurs types de modèles disponibles.

Pour implanter ce type d'architecture, on utilise le patron classe/interface Observable/Observer.

```
public class View implements Observer {  
    public void update(Observable object, Object arg) {  
        // méthode qui est déclenchée  
        // quand l'objet observable (le modèle) est modifié  
    }  
    // reste de la définition de la classe  
}  
  
public class Model extends Observable {  
    // définition + appels à setChanged et NotifyObservers  
}
```

1) Les classes mères : Component et JComponent

Tous les composants de Swing descendent de la classe JComponent, qui descend elle-même de la classe Component de awt. Cette hiérarchie permet de proposer de nombreuses méthodes communes à tous les éléments.

Relation entre les composants

Tous les composants sont hébergés par un **conteneur** (sauf les conteneurs primaires). Il est possible de connaître le conteneur d'un composant en utilisant la méthode getParent. On ajoute un composant dans un conteneur en utilisant la méthode add.

Les propriétés géométriques

- La position d'un composant peut être obtenue avec la méthode getLocation. Le coin supérieur gauche du composant parent est utilisé comme position de référence.
- La position peut être modifiée avec la méthode setLocation. Swing propose les gestionnaires de placement pour disposer les composants.
- La taille d'un composant peut être fixée avec la méthode setSize (et lue avec la méthode getSize).

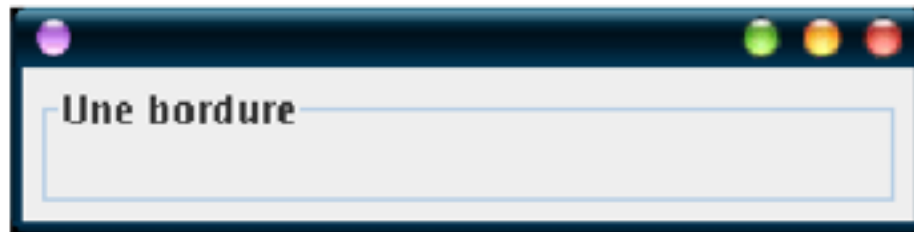
```
monComposant.setSize(monComposant.getPreferredSize());
```

Les classes mères :suite...

Les bordures

Les composants peuvent être encadrés par une bordure. Plusieurs bordures sont disponibles en creux, en relief, avec un texte,... La classe `BorderFactory` fournit de nombreuses méthodes statiques pour créer des bordures prédéfinies. Par exemple pour créer une bordure avec un titre :

```
monComposant.setBorder(BorderFactory.createTitleBorder("Une bordure"));
```



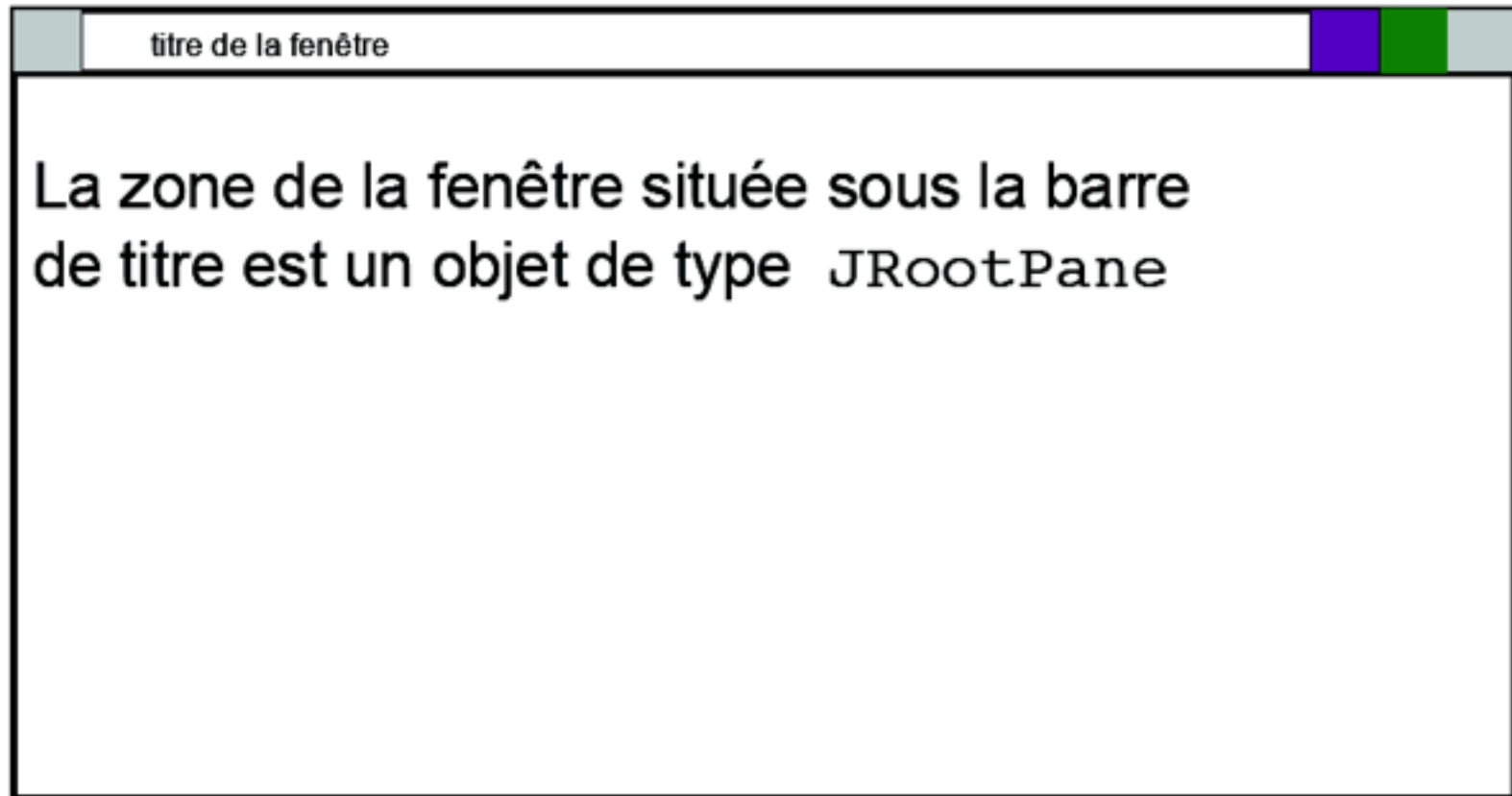
2) Les conteneurs primaires

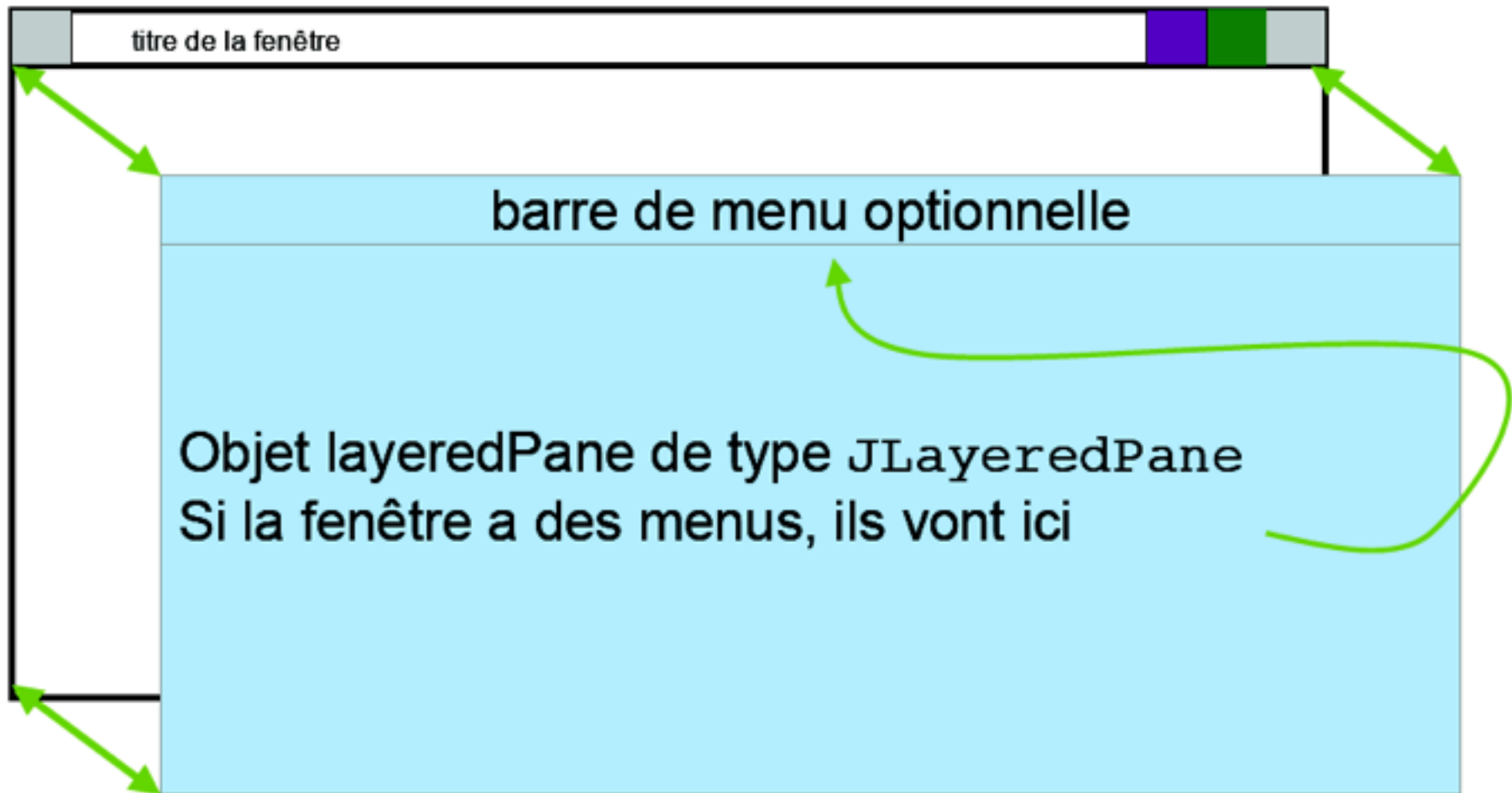
- Les seuls composants qui sont dessinés par le système d'exploitation (composants lourds). Toute application graphique doit utiliser un conteneur primaire.
- Trois types de conteneurs primaires existent : les applets (JApplet), les fenêtres (JFrame et JWindow) et les boîtes de dialogues (JDialog).

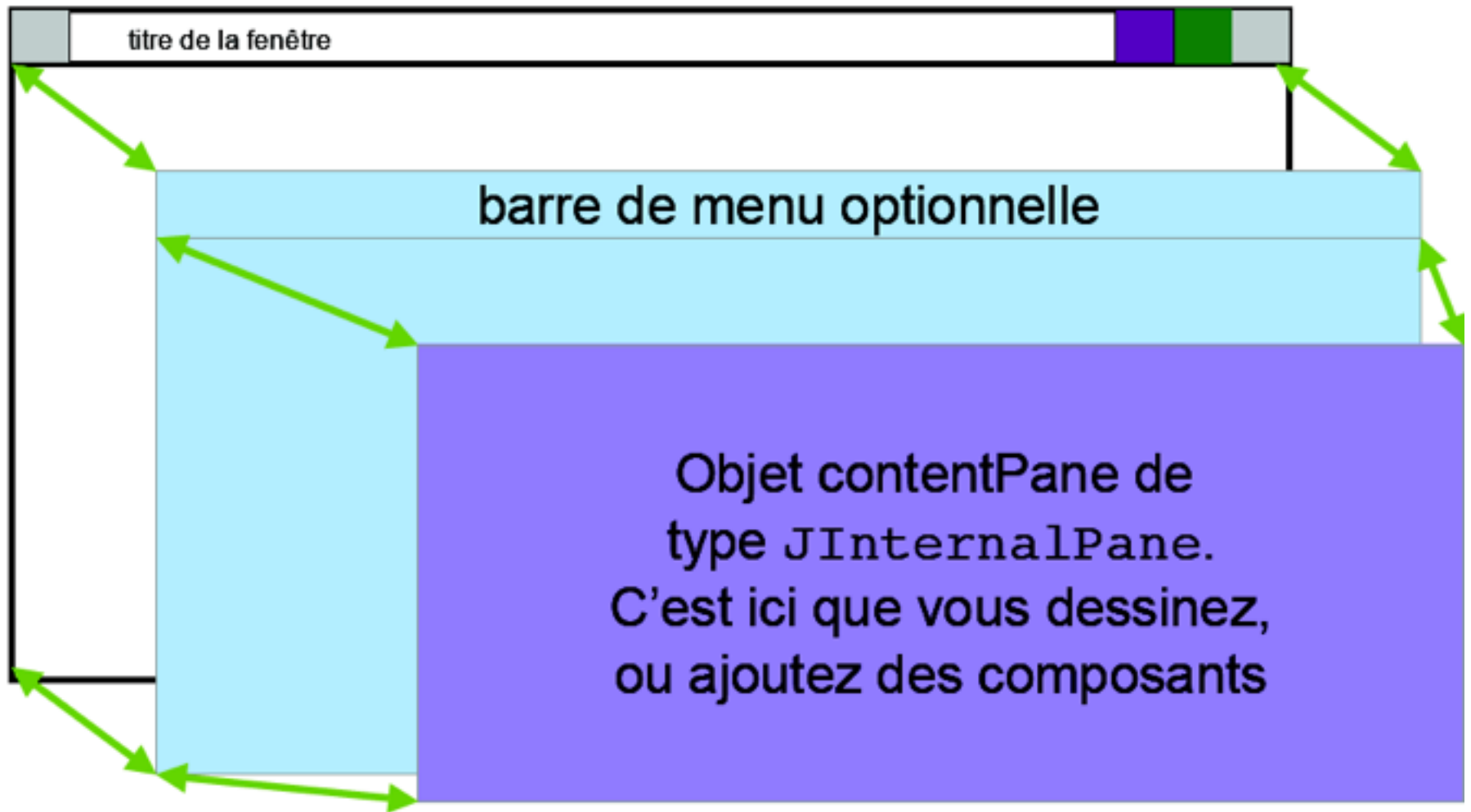
Propriétés communes

Les quatre conteneurs primaires sont construits sur le même modèle en couches.

- La couche supérieure est le GlassPane, un panneau transparent qui est utilisé pour la gestion des événements
- Sous ce panneau se situe le contentPane qui accueille les composants graphiques. Par défaut c'est un JPanel.
- Il est contenu par le LayeredPane qui peut être utilisé pour empiler des composants à des "profondeurs" différentes. Pour les JApplet, JFrame et les JDialog, il contient aussi une barre de menu, accessible via la méthode setJMenuBar. Par défaut cet élément est null.
- Tous ces éléments sont contenus dans un élément principal de type JRootPane.







Accès aux panneaux :

La classe JFrame contient les méthodes d'accès aux panneaux :

- getRootPane()
- getLayeredPane()
- getContentPane()
- getGlassPane()

Ajout de composant :

Les composants (boutons, labels,...) sont placés sur le contentPane. Par défaut ce conteneur est un JPanel. Il est possible d'accéder à ce conteneur avec la méthode getContentPane, puis d'appeler la méthode add pour ajouter des composants. Cette méthode accepte en paramètre un objet de type Jcomponent.

```
JFrame fen = new JFrame () ;  
Containercont = fen.getContentPane() ;  
cont.add(myComponent) ;
```

3) Les fenêtres : JWindow, JFrame

Swing propose deux types de fenêtres : JWindow et JFrame. Ces deux composants descendent de Window (composant awt).

Méthode communes

Pour créer une fenêtre, le constructeur est appelé. Par défaut, une fenêtre qui est créée n'est pas affichée. Pour l'afficher, il faut faire appel à la méthode setVisible.

```
JFrame fen = new JFrame() ; // identique pour JWindow  
fen.setVisible( true ) ;
```

- La taille d'une fenêtre dépend des éléments qu'elle contient. Afin d'éviter de l'estimer ou de la calculer, Swing propose une méthode (pack) qui calcule la taille de la fenêtre en fonction de la taille préférée de ses composants internes.

```
fen.pack();
```

- Lors de l'appel de la méthode pack(), la méthode getPreferredSize() est appelée sur tous les composants pour connaître leurs dimensions.

- Pour créer un programme exécutable en mode graphique, il suffit de créer une (ou plusieurs) fenêtre dans la classe qui contient la méthode statique main.

- Lorsqu'une fenêtre n'est plus utile, elle peut être cachée (avec setVisible(false)) ou détruite.

Les fenêtres : suite...

Différences entre JFrame et JWindow

La classe JWindow permet de créer une fenêtre sans aucune bordure et aucun bouton. Elle ne peut être fermée que par le programme qui l'a construite (ou en mettant fin à l'application via le système d'exploitation en tuant le processus associé). D'une manière générale, JWindow n'est utilisée que pour créer des fenêtres particulières comme des écrans d'attente.

```
import javax.swing.JWindow ;
public class TestJWindow{
    public static void main(String [] args){
        JWindow fen = new JWindow () ;
        fen.setSize(300,300) ;
        fen.setLocation(500,500) ;
        fen.setVisible(true) ;
    }
}
```

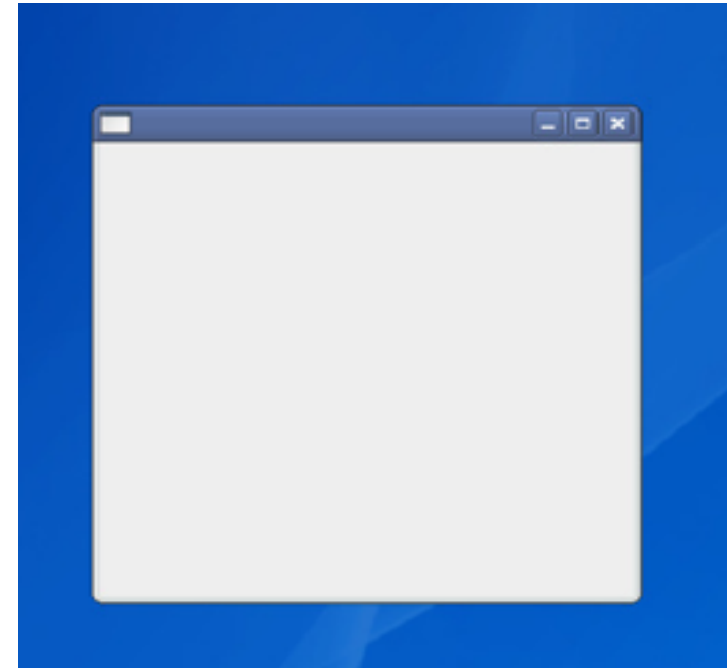


La méthode setLocation permet de positionner le composant à l'intérieur du conteneur. Dans le cas des JFrame et des JWindow, le conteneur est l'écran. Ce programme construit une fenêtre sans bordure et sans bouton, elle ne peut pas être fermée.

Les fenêtres : suite...

La plupart des applications sont construites à partir d'une (ou plusieurs) JFrame. En effet, JFrame construit des fenêtres qui comportent une bordure, un titre, des icônes et éventuellement un menu.

```
import javax.swing.JFrame ;
public class TestJFrame{
    public static void main(String [] args){
        JFrame fen = new JFrame() ;
        fen.setSize(300,300);
        fen.setVisible(true);
        fen.setLocation(500,500) ;
    }
}
```



Par défaut, une JFrame n'est pas fermée lorsque l'on clique sur l'icône de fermeture mais cachée (comme avec setVisible(false)). Ce comportement peut être modifié avec la méthode setDefaultCloseOperation. Plusieurs comportements sont définis dans la classe JFrame. Pour fermer la fenêtre, on utilise EXIT_ON_CLOSE :

```
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;
```

Pour modifier le titre d'une JFrame, on utilise la méthode setTitle(). On peut aussi utiliser la version surchargée du constructeur JFrame :

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

4) Les boîtes de dialogues

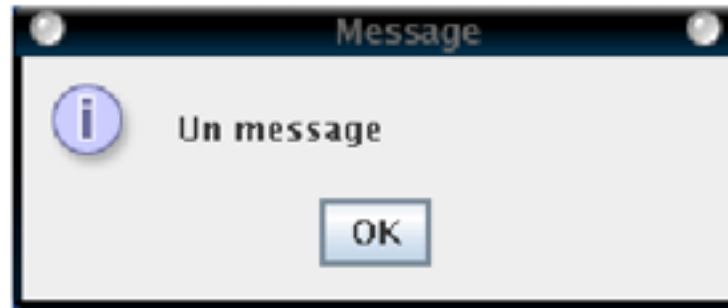
- Swing propose des boîtes de dialogues afin de communiquer rapidement avec l'utilisateur. La classe `JOptionPane` permet de créer des boîtes de dialogues génériques mais aussi de les modifier en fonction des besoins de l'application.
- Les méthodes d'affichage sont statiques (pas nécessaire d'instancier une classe pour les utiliser).

Les dialogues de message

- Les plus simple, elles informent l'utilisateur en affichant un texte simple et un bouton de confirmation. Elles restent affichée tant que l'utilisateur n'a pas cliqué sur le bouton et bloquent l'application en cours.
- Pour afficher un message on utilise la méthode `showMessageDialog` de la classe `JOptionPane`. Cette méthode propose de nombreuses options accessibles via la surcharge.

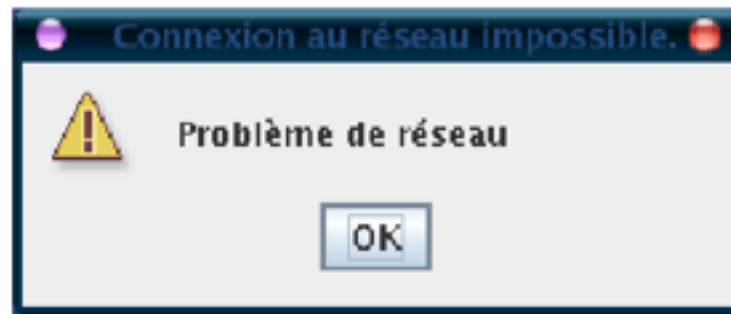

```
JOptionPane.showMessageDialog(fen,"Un message")
```

composant parent



On peut modifier l'aspect de la fenêtre en fonction du type de message, par exemple pour afficher un dialogue d'avertissement :

```
JOptionPane.showMessageDialog(fen,"Problme de erseau", "Impossible de se  
connecter",JOptionPane.WARNING_MESSAGE);
```

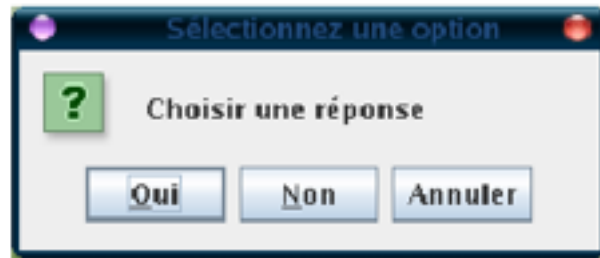


Les boites de dialogues : suite...

Les dialogues de confirmation/question

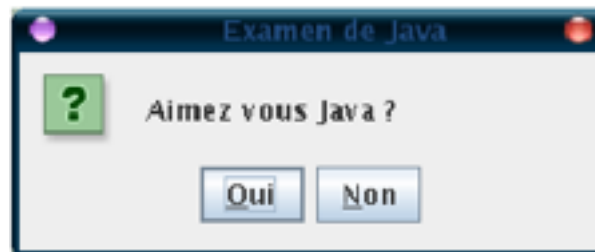
- Les boites de dialogues peuvent aussi être utilisées pour demander un renseignement à l'utilisateur.
- Le cas le plus courant est une question fermée (oui ou non). La méthode `showConfirmDialog` affiche ce type de boite. Sans paramètre particulier, elle ajoute un bouton pour annuler la question :

```
JOptionPane.showConfirmDialog(fen,"Choisir une réponse");
```



La syntaxe est la même que pour `showMessageDialog`. On peut personnaliser la boite de dialogue avec des chaînes de caractères, par exemple :

```
JOptionPane.showConfirmDialog(fen,"Aimez vous Java ?", "Examen de Java",JOptionPane.YES_NO_OPTION);
```



Les boîtes de dialogues : suite...

La boîte de dialogue renvoie un entier en fonction du choix de l'utilisateur. Cette valeur peut être utilisée pour modifier le comportement du programme. Cet entier est défini comme une constante de la classe `JOptionPane`. Par exemple pour le cas précédent :

```
int rep = JOptionPane.showConfirmDialog(fen , " Aimez-vous Java ?", "Examen de
Java",JOptionPane.YES_NO_OPTION) ;
if ( rep == JOptionPane.YES_OPTION) {
// ...
}
if ( rep == JOptionPane.NO_OPTION) {
// ...
}
```

Les dialogues de saisie

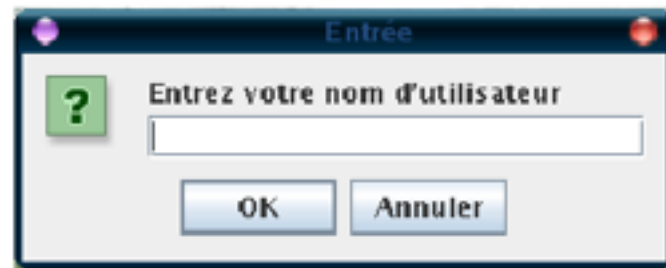
- La méthode `showInputDialog` affiche une boîte de dialogue comprenant une entrée de saisie (`JTextField`) en plus des boutons de validation. Après validation elle retourne une chaîne de caractères (`String`) si l'utilisateur a cliqué sur OK, sinon elle renvoie `null`.

Les boîtes de dialogues : suite...

Les dialogues de saisie

- La méthode `showInputDialog` affiche une boîte de dialogue comprenant une entrée de saisie (`JTextField`) en plus des boutons de validation. Après validation elle retourne une chaîne de caractères (`String`) si l'utilisateur a cliqué sur OK, sinon elle renvoie `null`.

```
String rep = JOptionPane.showInputDialog(fen,"Entrez votre nom d'utilisateur" );
```



La méthode `showInputDialog` est surchargée afin de pouvoir modifier l'aspect de la boîte de dialogue. Pour changer le titre de la boîte on va utiliser le code suivant :

```
String rep = JOptionPane.showInputDialog(fen,"Entrez votre nom  
d'utilisateur","Renseignement",JOptionPane.INFORMATION_MESSAGE);
```

```
System.out.println(rep);
```

Les boites de dialogues : suite...

Construction de dialogues personnalisées

- Pour construire des boites de dialogue plus complexes on utilise la méthode `showOptionDialog`. Cette méthode admet 8 paramètres (certains pouvant être à null). Elle renvoie un entier qui correspond au bouton qui a été cliqué.

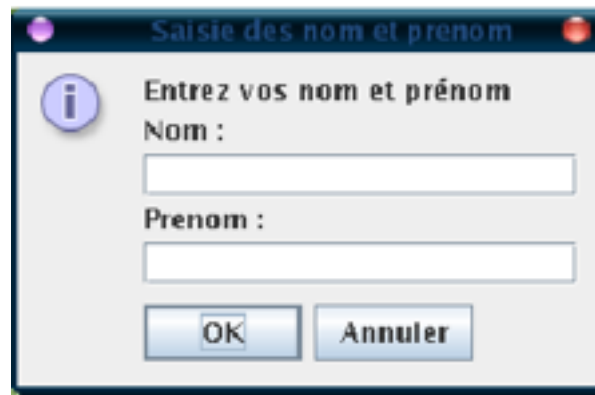
- Le prototype est le suivant :

`showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)`

Exemple : Construire une boite de dialogue qui demande à l'utilisateur de saisir ses nom et prénom. La boite contient 3 labels et 2 champs de saisie.

```
JLabel labelNom = new JLabel("Nom:");  
JLabel labelPrenom = new JLabel("Prenom:");  
JTextField nom = new JTextField();  
JTextField prenom = new JTextField();  
JLabel lab = new JLabel( " Entrez vos nom et prénom: " ) ;
```

```
Object [] tab = new Object [{ labelNom , nom , labelPrenom , prenom ,lab }];  
int rep = JOptionPane.showOptionDialog(fen,tab,"Saisie des nom  
etprenom",JOptionPane.OK_CANCEL_OPTION,      JOptionPane.INFORMATION  
_MESSAGE,null,null,null);
```



```
if (re == JOptionPane.OK_OPTION) {  
    System.out.println(" Nom : " + nom.getText() + " prenom : " + prenom.getText());  
}
```

5) Les conteneurs secondaires

- Swing propose de nombreux conteneurs secondaires pour créer des interfaces ergonomiques (composants légers).

Le panneau : JPanel

Le conteneur léger le plus simple de Swing : c'est le conteneur par défaut de JFrame et de JWindow. Il permet de grouper des composants selon une politique de placement. Pour ajouter un composant à un panneau, on utilise la méthode add (ou l'une de ses surcharges). La méthode réciproque remove permet d'enlever un composant.

Exemple : anneau comportant plusieurs composants : trois JLabel, deux JTextField et un JTextArea (formulaire de renseignements simple).

```

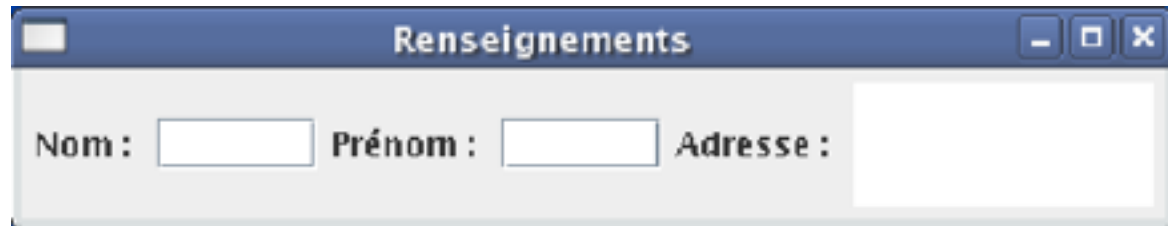
public class FichIdentite extends JPanel{
    private JLabel labelNom ;
    private JTextField nom ;
    private JTextField prenom ;
    private JLabel labelPrenom;
    private JTextArea adresse ;
    private JLabel labelAdresse;
    public FichIdentite() {
        super () ;
        labelNom = new JLabel ( " Nom : " ) ;
        labelPrenom = new JLabel ( " Prenom : " ) ;
        labelAdresse = new JLabel ( " Adresse : " ) ;
        nom = new JTextField(5) ;
        prenom = new JTextField(5) ;
        adresse = new JTextArea("",3,10) ;

        this.add(labelNom);
        this.add(nom);
        this.add(labelPrenom);
        this.add(prenom );
        this.add(labelAdresse);
        this.add(adresse);
    }
}

```

Pour pouvoir utiliser ce panneau, il faut l'inclure dans une fenêtre (par exemple une JFrame).


```
public class TestPanel{  
    public static void main ( String [] args ) {  
        JFrame fen = new JFrame () ;  
        Ficheldentite laFiche = new Ficheldentite();  
        fen.setContentPane(laFiche);  
        fen.pack () ;  
        fen.setVisible(true) ;  
        fen.setTitle("Renseignements") ;  
    }  
}
```

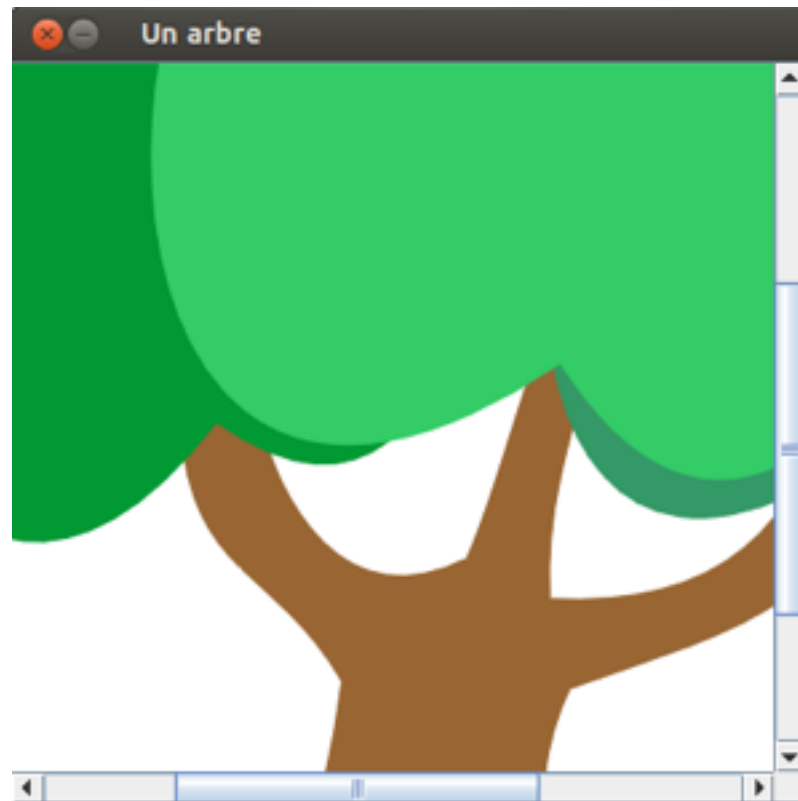


Le panneau à défilement : JScrollPane

Les applications traitant du texte ou des images n'affichent souvent qu'une partie du document pour éviter de monopoliser toute la surface d'affichage. Des glissières sont affichées sur les cotés afin de pouvoir se déplacer dans le document. Le composant JScrollPane permet d'implémenter cette fonction.

```
JFrame fra = new JFrame () ;  
JLabel monImage = new JLabel ( new ImageIcon("arbre.gif"));  
JScrollPane lePanneau = new JScrollPane(monImage) ;  
fra.setContentPane(lePanneau) ;  
fra.setTitle( " Un arbre " ) ;  
fra.setSize(400 ,400) ;  
fra.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);  
fra.setResizable(false);  
fra.setVisible(true) ;
```

Par défaut, les ascenseurs ne sont affichés que s'ils sont nécessaires. La politique d'affichage peut être modifiée à l'aide des méthodes `setVerticalScrollBarPolicy` et `setHorizontalScrollBarPolicy`.



Le panneau divisé : JSplitPane

Séparer une interface en deux volets est utilisé pour mettre en vis-à-vis deux documents, ou un document et une barre d'outils. La séparation peut être horizontale ou verticale selon les interfaces. Ce type d'interface fait appel au JSplitPane.

```
JFrame fra = new JFrame () ;  
JTextArea source = new JTextArea() ;  
JTextArea traduction = new JTextArea() ;  
JsplitPane lePanneau=new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,  
source,traduction) ;  
fra.setContentPane(lePanneau) ;  
fra.setSize(400 ,400) ;  
fra.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);  
fra.setVisible(true) ;
```



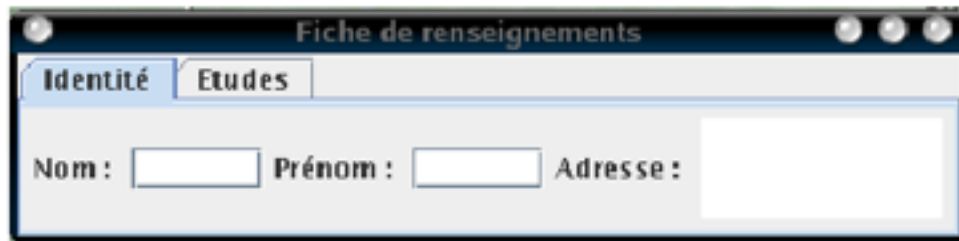
Le panneau à onglets : JTabbedPane

JTabbedPane permet de construire des interfaces en utilisant des onglets. Les composants sont regroupés de manière thématique.

La méthode `addTab` permet d'ajouter un composant dans une nouvelle feuille. Généralement on utilise un `JPanel`.

```
public class MenuOnglets extends JtabbedPane {  
    private FichIdentite identite ;  
    private FicheEtude etudes ;  
    public MenuOnglets(){  
        super() ;  
        identite = new FichIdentite() ;  
        etudes = new FicheEtude() ;  
        this.addTab( " Identit " , identite ) ;  
        this.addTab( " Etudes " , etudes ) ;  
    }  
}
```

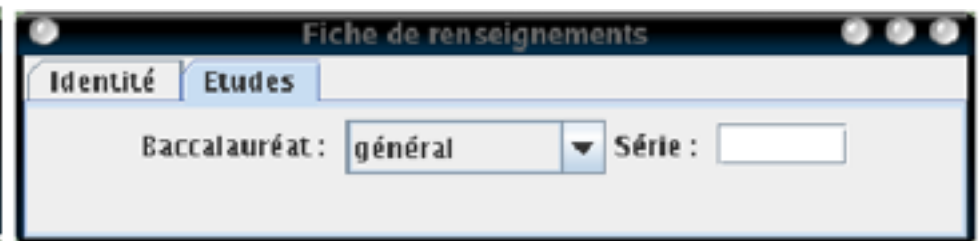
Les onglets sont numérotés à partir de 0. L'onglet affiché peut être modifié à l'aide de la méthode `setSelectedIndex`. `getSelectedIndex` retourne l'onglet visible. La position des onglets peut être modifiée en utilisant la méthode `setTabPlacement`.



Fiche de renseignements

Identité Etudes

Nom : Prénom : Adresse :



Fiche de renseignements

Identité Etudes

Baccalauréat : général ▼ Série :

6) Les composants atomiques

Les composants atomiques sont tous les composants élémentaire de Swing. Ce sont les boutons, les labels, les menus,...

Les labels : JLabel

Un label est une simple chaîne de caractères informative (il peut aussi contenir une image). Pour créer un nouveau label il suffit d'appeler le constructeur JLabel.

```
JLabel monLabel = new JLabel ( " Une chaîne de caractères" ) ;  
Ce label est ajouté à un conteneur avec la méthode add.
```

```
JFrame fen = new JFrame () ;  
JPanel pan = new JPanel () ;  
JLabel unLabel = new JLabel ( " Une chaine de caractres" ) ;  
pan.add ( unLabel ) ;  
fen.setContentPane( pan ) ;  
fen.pack () ;//pack fait en sorte que tous les composants de //l'application soient  
à leur preferredSize, ou au dessus  
fen.setVisible( true ) ;
```



Code à tester...

```
JFrame fen = new JFrame () ;  
JPanel pan = new JPanel () ;  
JLabel unLabel = new JLabel ( " Une chaine de caracteres" ) ;  
ImageIcon img = new ImageIcon("arbre.gif") ;  
JLabel unLabel2 = new JLabel ( img ) ;  
pan.add( unLabel2 ) ;  
pan.add ( unLabel ) ;  
fen.setContentPane( pan ) ;  
fen.pack () ;  
fen.setVisible( true ) ;  
fen.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
```

Les boutons : JButton et JToggleButton

Les boutons sont les composants les plus utilisés pour interagir avec l'utilisateur. Swing propose plusieurs types de boutons. Tous les boutons héritent de la classe abstraite `AbstractButton`.

Le composant JButton

Le bouton le plus utilisé est le `JButton`. Il crée un bouton qui peut être cliqué par l'utilisateur à l'aide de la souris. Généralement le texte affiché dans le bouton est passé comme paramètre au constructeur. Toutefois, il est possible de le modifier à l'aide de la méthode `setText`.

```
JPanel pan = new JPanel () ;  
JLabel unLabel = new JLabel ( " Un label " ) ;  
JButton unBouton = new JButton ( " Un Bouton " ) ;  
pan.add(unLabel);  
pan.add(unBouton);
```

Pour ajouter une icône :

```
ImageIcon img = new ImageIcon( "sun-java.png" ) ;  
JButton unBouton = new JButton ( " Un Bouton " , img ) ;
```



Le composant JToggleButton

Swing propose un type de bouton particulier, les boutons à bascule : JToggleButton. Ces boutons conservent leur état après le relâchement de la souris. Ils sont utilisés pour représenter un état booléen (comme par exemple l'effet souligné dans un traitement de texte). On peut comparer les JButton aux boutons poussoirs et les JToggleButton aux interrupteurs.

```
ImageIcon img = new ImageIcon( "sun-java.png" ) ;  
JToggleButton unBouton= new JToggleButton("Un Bouton",img);
```



Les cases à cocher : JCheckBox

Les cases à cocher permettent de matérialiser des choix binaires d'une manière plus usuelle que les boutons à bascules (JToggleButton). Comme les boutons, elles héritent de la classe abstraite AbstractButton.

```
JCheckBox casePasCochee = new JCheckBox("Une case à cocher");  
JCheckBox caseCochee = new JCheckBox("Une case cochée",true);
```



Comme pour les boutons à bascule, on lit l'état d'une case à cocher à l'aide de la méthode `isSelected()` qui renvoie une valeur booléenne et on la modifie avec `setSelected`.

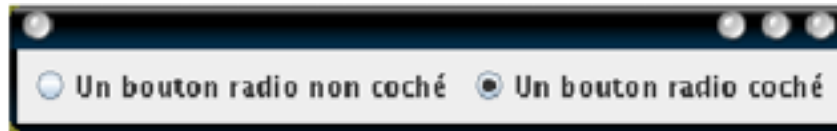
```
if (maCaseACocher.isSelected()== true) {  
    // la case est coche  
} else {  
    // la case n'est pas cochée  
}
```

Les boutons radio : JRadioButton

Les boutons radio JRadioButton sont des boutons à choix exclusif, ils permettent de choisir un (et un seul) élément parmi un ensemble. Comme les AbstractButton dont ils héritent, ils ont deux états. Pour créer un bouton radio, on peut utiliser différentes surcharges du constructeur. L'une des plus utilisées permet d'initialiser la variable de texte associée au bouton : JRadioButton(String).

Il est possible de choisir l'état du bouton de la création avec le constructeur JRadioButton(String, boolean).

```
JRadioButton bouton1 = new JRadioButton("Un bouton radio");  
JRadioButton bouton2 = new JRadioButton("Un bouton radio coché",  
true);
```



Les boutons radio doivent être regroupés dans un BoutonGroup pour avoir un comportement exclusif. Il s'agit d'un groupe logique où un et seul bouton peut être actif (true). Le BoutonGroup n'a aucune incidence sur l'affichage. Il est donc utile lors de la conception de matérialiser les groupes de boutons radio.

```

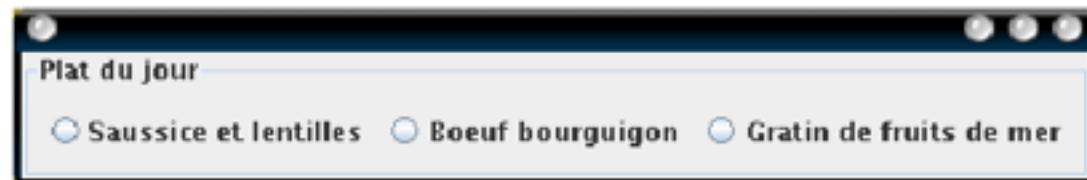
public class TestRadio extends JPanel {
    JRadioButton plat1 , plat2 , plat3;
    ButtonGroup plat=null;
    public TestRadio(){
        plat1 = new JRadioButton(" Saussice et lentilles");
        plat2 = new JRadioButton(" Boeuf bourguignon");
        plat3 = new JRadioButton(" Gratin de fruits de mer ");
        plat = new ButtonGroup();
        plat.add( plat1);
        plat.add( plat2);
        plat.add( plat3);
        this.add( plat1);
        this.add( plat2);
        this.add( plat3);
        this.setBorder(BorderFactory.createTitledBorder("Plat du jour"));
    }
}

```

```

public class PlatDuJour{
    public static void main( String [] args) {
        JFrame fen = new JFrame ();
        TestRadio panneau = new TestRadio();
        fen.setContentPane( panneau );
        fen.pack();
        fen.setVisible( true);
    }
}

```



Les listes de choix : JList

Ce composant permettent de choisir un (ou plusieurs) élément(s) parmi un ensemble prédéfini. Cet ensemble peut être un tableau ou vecteur d'objets quelconques . Les éléments sont présentés sous la forme d'une liste dans laquelle les éléments choisis sont surlignés.

```
String [] lesElements={" Guitare " , " Basse " , " Clavier " , "Batterie " , " Percussions" , " Flute" ,  
" Violon "};  
JList instruments = new JList( lesElements);  
JPanel pan = new JPanel();  
JLabel text = new JLabel(" Choisissez un(des ) instrument(s) :");  
JFrame fen = new JFrame(" Musique ");  
pan.add(text);  
pan.add(instruments);
```



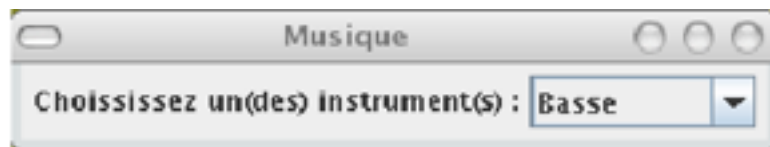
Les éléments sont numérotés à partir de 0. Les méthodes `getSelectedIndex` et `getSelectedIndices` permettent de connaître le premier indice ou tous les indices des éléments sélectionnés. De même `getSelectedValue` et `getSelectedValues` fournissent le premier élément ou tous les éléments sélectionnés.

Les boites combo : JComboBox

Les boites combo permettent de choisir un seul élément parmi une liste proposée. On les utilise quand l'ensemble des éléments à afficher n'est pas connu lors de la conception. Comme les listes de choix, on peut les construire en passant un tableau d'objet en paramètre de construction.

```
JComboBox instruments = new JComboBox( lesElements);
```

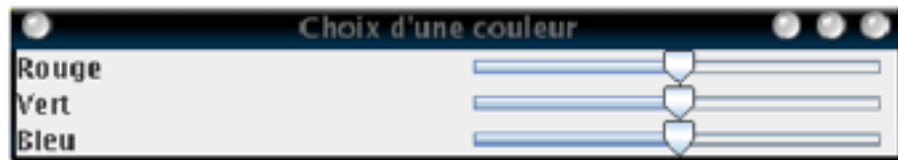
Les méthodes `getSelectedIndex` et `getSelectedItem` permettent de connaître l'indice et l'objet sélectionné.



Les glissières : JSlider

Les glissières permettent de proposer à l'utilisateur une interface de saisie plus intuitive qu'un champ de texte pour régler certains paramètres. Swing propose le composant JSlider pour représenter un réglage variable.

```
JSlider sBlue = new JSlider ();  
JSlider sRed = new JSlider ();  
JSlider sGreen = new JSlider ();  
// ...  
JPanel pan = new JPanel ();  
// ...  
pan.add(sRed);  
pan.add(sGreen );  
pan.add(sBlue);
```

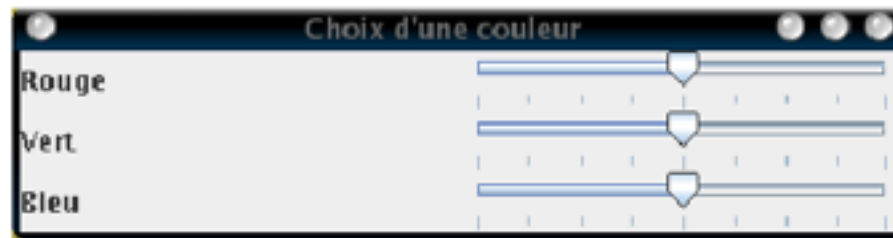


La position du curseur est fixée avec `setValue` et lue avec `getValue`. L'orientation peut être changée en utilisant la méthode `setOrientation`. Les `JSlider` peuvent être configurés pour couvrir une étendue à l'aide de `setMaximum` et `setMinimum`. Il est possible de tracer des repères, deux types sont proposés, les repères majeurs et les repères mineurs ; les intervalles de chaque type sont définis avec `setMinorTickSpacing` et `setMajorTickSpacing`.

```
sBlue.setMinimum(0);  
sBlue.setMaximum(255);  
sBlue.setValue(127);  
sBlue.setMajorTickSpacing(127);  
sBlue.setMinorTickSpacing(32);  
sBlue.setPaintTicks( true);
```

On peut configurer les bornes et la position du curseur :

```
JSlider sBlue = new JSlider(JSlider.HORIZONTAL,0,255,127);  
sBlue.setMajorTickSpacing(127);  
sBlue.setMinorTickSpacing(32);  
sBlue.setPaintTicks(true);
```



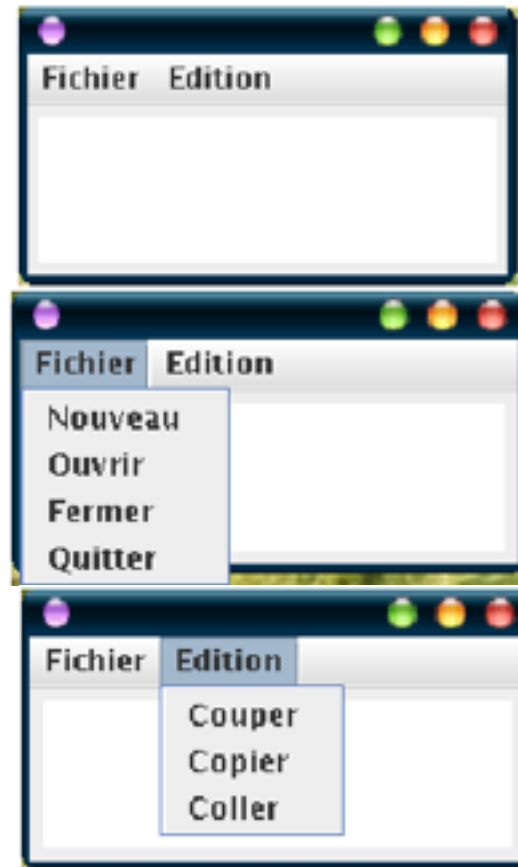
Les menus : JMenu, JMenuBar, JMenuItem, JRadioButtonMenuItem, JCheckBoxMenuItem

Les barres de menus permettent de regrouper de nombreuses fonctions d'une manière ergonomique. On utilise un composant JMenuBar pour construire une barre de menus. Les menus sont construits à partir de la classe JMenu. Ils sont placés dans la JMenuBar avec la méthode add. Ces menus sont constitués d'éléments appartenant à la classe JMenuItem ou à l'une de ses classes filles (JRadioButtonMenuItem, JCheckBoxMenuItem). Les éléments de menus sont ajoutés aux menus à l'aide de la méthode add.

```

JFrame fen = new JFrame ();
JMenu menuFichier = new JMenu(" Fichier ");
JMenuItem menuFichierNouveau = new JMenuItem(" Nouveau ");
JMenuItem menuFichierOuvrir = new JMenuItem(" Ouvrir ");
JMenuItem menuFichierFermer = new JMenuItem(" Fermer ");
JMenuItem menuFichierQuitter = new JMenuItem(" Quitter ");
menuFichier.add ( menuFichierNouveau);
menuFichier.add ( menuFichierOuvrir);
menuFichier.add ( menuFichierFermer);
menuFichier.add ( menuFichierQuitter);
JMenu menuEdition = new JMenu(" Edition ");
JMenuItem menuEditionCouper=new JMenuItem("Couper");
JMenuItem menuEditionCopier=new JMenuItem("Copier");
JMenuItem menuEditionColler=new JMenuItem("Coller");
menuEdition.add ( menuEditionCouper);
menuEdition.add ( menuEditionCopier);
menuEdition.add ( menuEditionColler);
JMenuBar barreMenu = new JMenuBar ();
barreMenu.add( menuFichier);
barreMenu.add( menuEdition);
fen.setJMenuBar( barreMenu);

```



La barre de menus (barreMenu) pourrait être ajoutée à un conteneur. Ici, comme nous utilisons une fenêtre JFrame, nous pouvons ajouter directement la barre de menus à la fenêtre à l'aide de la méthode setJMenuBar.

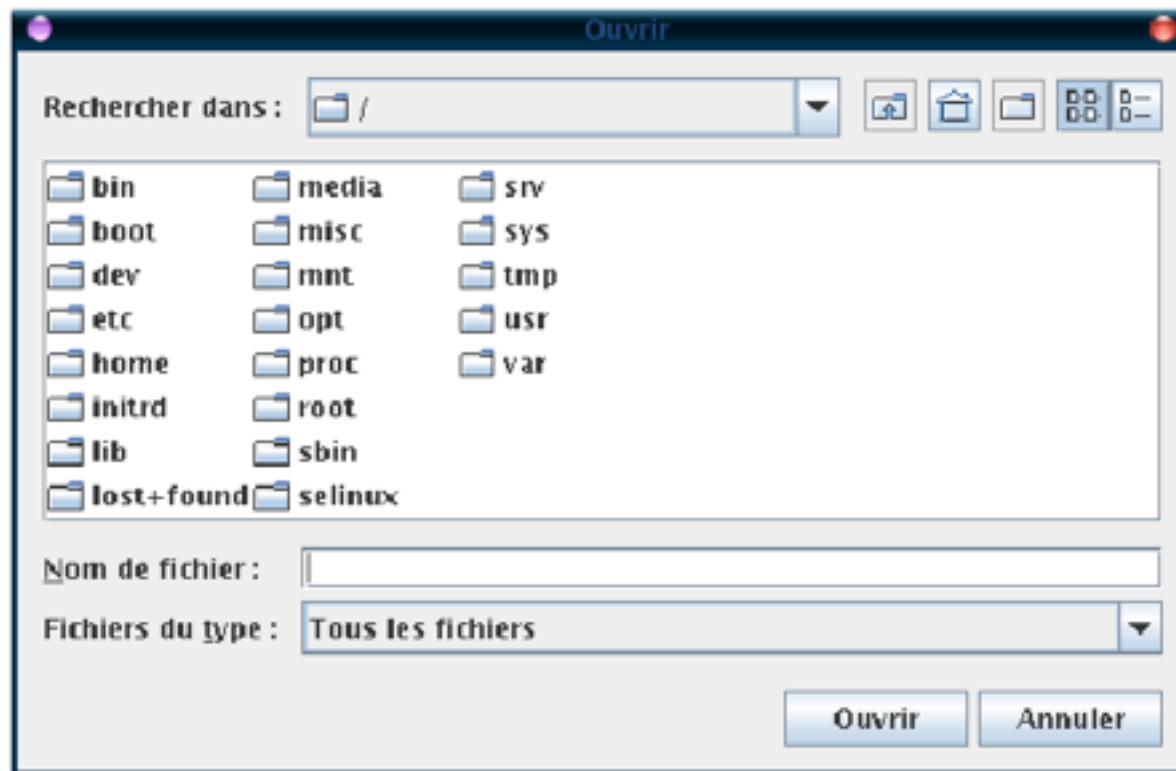
Les dialogues de sélection de fichiers : JFileChooser

- Comme toutes les interfaces graphiques, Swing propose une boîte de sélection de fichier(s) avec la classe JFileChooser. La classe propose trois méthodes pour afficher un dialogue d'ouverture de fichier.
- La première méthode (showOpenDialog) présente une boîte de dialogue pour l'ouverture d'un fichier, la seconde (showSaveDialog) pour la sauvegarde d'un fichier, enfin, la troisième méthode (showDialog) permet de spécifier des chaînes de caractères pour le bouton de validation et le titre de la fenêtre afin de créer des boîtes personnalisées.

Contrairement aux boîtes de dialogues, un objet doit être instancié :

```
JFileChooser fc = new JFileChooser();  
fc.showOpenDialog( fen );
```

- Les trois méthodes renvoient un entier dont la valeur correspond au bouton qui a été cliqué. La méthode getSelectedFile renseigne sur le nom du fichier sélectionné.
- ```
if (fc.showOpenDialog(fen)== JFileChooser. APPROVE_OPTION){
 System.out.println("Le fichier est:"+fc.getSelectedFile());
}
```



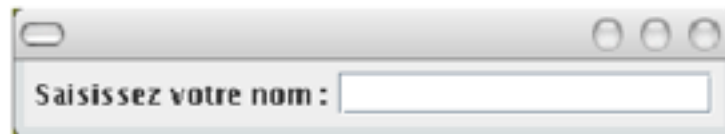
## Les composants orientés texte

- Swing propose cinq composants pour travailler avec du texte : `JTextField`, `JFormattedTextField`, `JPasswordField`, `JTextArea`, `JEditorPane` et `JTextPane`. Tous ces composants descendent (directement ou non) de `JTextComponent`.

### Le champ de saisie `JTextField`

Pour saisir une seule ligne de texte, on utilise le composant `JTextField`. Il construit un champ de saisie dont la largeur peut être fixée avec `setColumns`. Il est préférable de fixer la largeur du champ de saisie pour éviter des déformations des interfaces graphiques lors du remplissage.

```
JPanel pan = new JPanel ();
JLabel lNom = new JLabel (" Entrez votre nom :");
JTextField tfNom = new JTextField();
tfNom.setColumns(15);
pan.add (lNom);
pan.add (tfNom);
```



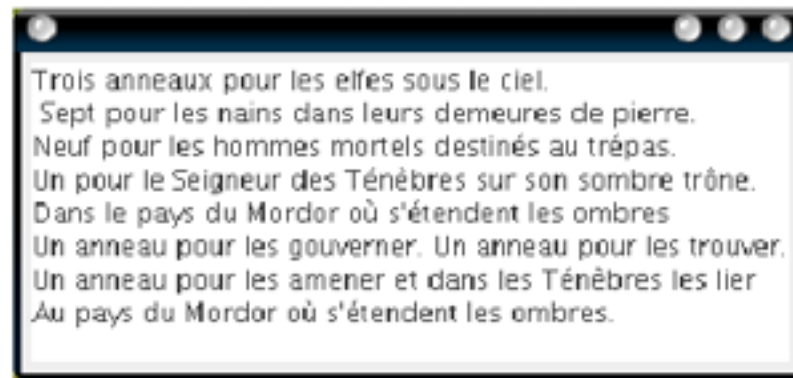
## La zone de texte JTextArea

Il est souvent nécessaire d'afficher un texte sur plusieurs lignes. Le composant JTextArea est le composant le plus simple pour effectuer cette tâche. Le texte est affiché en bloc, avec une police unique. Pour remplir le composant on utilise la méthode setText :

```
JTextArea taNotes = new JTextArea();
taNotes.setText(" Trois anneaux pour ...
```

```
...
```

```
les ombres.\n");
taNotes.setEditable(false);
pan.add(taNotes);
```



## Visualisateur de documents formatés JEditorPane

Pour afficher des mises en page plus complexes ou/et des documents plus riches les formats RTF et HTML sont souvent reconnus comme des standards. Le composant JEditorPane permet de gérer des affichages complexes notamment l'affichage de page web via la méthode setPage :

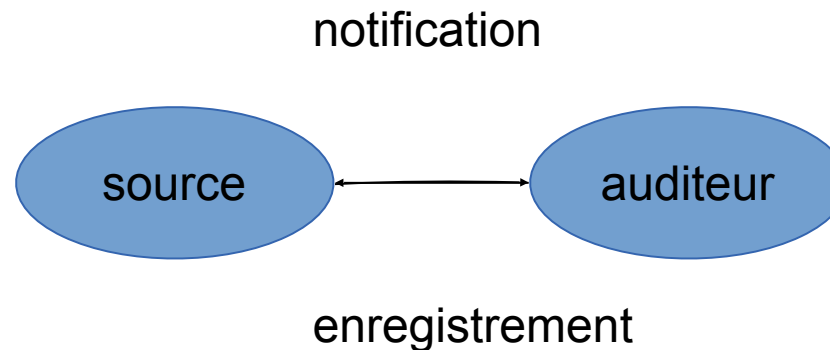
```
JEditorPane epNotes = new JEditorPane();
epNotes.setEditable(false);
epNotes.setPage(" http :// www. google .fr");
```

Ce composant permet de réaliser un navigateur simple en l'intégrant à une fenêtre



## 7) Les événements

- Java propose des mécanismes de communication entre les objets basés sur des événements.
- L'événement est émis par un objet (la source) et reçu par un ou plusieurs objets (les auditeurs). On dit que la source notifie un événement aux auditeurs. Pour pouvoir recevoir un (ou des) événement(s) les auditeurs doivent s'enregistrer au près de la (des) source(s).



### Les événements

- Les événements héritent de la classe `EventObject`. Cette classe ne propose qu'une méthode : `getSource` qui renvoie l'objet qui a créé cet événement.
- Le constructeur de `EventObject` admet l'objet qui l'a créé (source) comme seul paramètre. Par convention, les noms des objets sont de la forme `XXXEvent`.



## **La source**

- La source doit être capable de créer des événements et de les transmettre aux listeners.

## **Les listener (auditeurs)**

- Les listeners doivent implémenter une interface qui dérive de `EventListener`. En pratique, cette interface a un nom de la forme `XXXListener`. Elle doit comprendre une méthode qui admet comme argument un élément de type `XXXEvent`.
- Un listener est un objet dont la fonction est d'écouter un événement et de le traiter.
- On associe un listener à un composant pour traiter les événements qu'il reçoit.
- Il existe un type de listener pour chaque type d'événement.
- Les classes d'événement sont définies dans les paquetages `java.awt.event.*` et `javax.swing.event.*`.

## java.awt.event

| Listeners           | Événement        |            |
|---------------------|------------------|------------|
| ActionListener      | ActionEvent      |            |
| AdjustementListener | AdjustementEvent |            |
| ComponentListener   | ComponentEvent   | PaintEvent |
| ContainerListener   | ContainerEvent   |            |
| FocusListener       | FocusEvent       |            |
| InputMethodListener | InputMethodEvent |            |
| ItemListener        | ItemEvent        |            |
| KeyListener         | KeyEvent         | InputEvent |
| MouseListener       | MouseEvent       |            |
| MouseMotionListener |                  |            |
| TextListener        | TextEvent        |            |
| WindowListener      | WindowEvent      |            |

## javax.swing.event

| Listeners             | Événement          | Listeners                | Événement             |
|-----------------------|--------------------|--------------------------|-----------------------|
| AncestorListener      | AncestorEvent      | MenuListener             | MenuEvent             |
| CaretListener         | CaretEvent         | MouseListener            |                       |
| CellEditorListener    | ChangeEvent        | PopupMenuListener        | PopupMenuEvent        |
| ChangeListener        |                    | TableColumnModelListener | TableColumnModelEvent |
| DocumentListener      | DocumentEvent      | TableModelListener       | TableModelEvent       |
| HyperlinkListener     | HyperlinkEvent     | TreeExpansionListener    | TreeExpansionEvent    |
| InternalFrameListener | InternalFrameEvent | TreeModelListener        | TreeModelEvent        |
| ListDataListener      | ListDataEvent      | TreeSelectionListener    | TreeSelectionEvent    |
| ListSelectionListener | ListSelectionEvent | TreeWillExpandListener   | TreeExpansionEvent    |
| MenuDragMouseListener | MenuDragMouseEvent | UndoableEditListener     | UndoableEditEvent     |
| MenuKeyListener       | MenuKeyEvent       |                          |                       |

## **Fermer la fenêtre**

- Pour terminer le programme en fermant la fenêtre, on traite l'événement WindowEvent.
- Pour cela on crée un objet WindowListener qu'on associe à la fenêtre.
- WindowListener est une interface, on crée une classe interne implémentant l'interface.

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

```
public class Swing1 {
 static JFrame f;
 public static void main(String [] args){
 f = new JFrame("Ma première fenêtre");
 Ecouteur ec = new Ecouteur();
 f.addWindowListener (ec);
 //f.addWindowListener((new Fenêtre()).new Ecouteur());
 f.setBounds(100,100, 300, 200);
 f.setVisible(true);
 }
}
```

```
class Ecouteur implements WindowListener {

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

interface



## WindowAdapter : redéfinir uniquement les méthodes utiles

```
import javax.swing.*;
import java.awt.event.*;
public class Swing1 {
 static JFrame fenêtre;
 public static void main(String [] args){
 fenêtre = new JFrame("Ma première fenêtre");
 fenêtre.addWindowListener(new Ecouteur());
 fenêtre.setBounds(100,100, 300, 200);
 fenêtre.setVisible(true);
 }
}
```

implémentation

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

## 8) Positionnement des composants

- Le placement des composants dans un conteneur est défini par un gestionnaire de placement. Lors de la création d'un conteneur, un gestionnaire est créé en même temps et lui est associé.

| Conteneur       | Gestionnaire par défaut |
|-----------------|-------------------------|
| JPanel, JApplet | FlowLayout              |
| JFrame, JWindow | BorderLayout            |

4 principaux conteneurs

- Les gestionnaires de placement implémentent l'interface `LayoutManager`. Ils utilisent les méthodes `getPreferredSize/getMinimumSize`, pour connaître la taille préférée/minimale des composants.
- Ensuite, ils calculent les tailles et les positions des composants. Ces informations sont ensuite appliquées en utilisant les méthodes `setSize` et `setLocation`.
- Pour accéder au gestionnaire de placement d'un conteneur on utilise la méthode `getLayout`. La méthode `setLayout` permet de spécifier un nouveau gestionnaire.

# LayoutManagers

- **BorderLayout**
  - divise la zone en 5 régions : nord, sud, est, ouest, centre
  - la division est paramétrée par l'espacement horizontal et vertical des composants
  - c'est le layout par défaut pour JFrame
- **GridLayout**
  - \* dispose les composants en tableau
  - \* il nécessite les paramètres nombre de colonnes et de lignes, espacement des colonnes et des lignes
- **FlowLayout**
  - \* dispose les composants séquentiellement horizontalement
  - \* ces paramètres sont la justification (gauche, droite, centrée, justifiée); l'espacement horizontal et vertical
- **ViewportLayout**
  - \* un seul composant aligné en bas si la surface est suffisante, en haut sinon
- **ScrollPaneLayout**
  - \* le composant peut défiler dans la zone
- **BoxLayout**
  - \* semblable à GridLayout avec une seule dimension
- **OverlayLayout**
  - \* les composants sont empilés les uns sur les autres
- **CardLayout**
  - \* Concept de page
- **GridBagLayout**
  - \* disposition générale



## Le positionnement absolu

Bien que cette approche soit peu recommandée, il est possible de ne pas utiliser de gestionnaire de placement. Dans ce cas, on utilise `setLayout(null)` pour désactiver le gestionnaire par défaut du conteneur.

Les composants sont ensuite placés en utilisant les coordonnées absolues à l'aide de la méthode `setLocation`. La taille du composant peut être imposée en utilisant la méthode `setSize` et des valeurs :

```
JButton unBouton , unAutreBouton;
JLabel unLabel ;
// ...
unBouton.setSize(100,20);
unAutreBouton.setSize(150,20);
unLabel.setSize(200,50);
// ...
unBouton.setLocation(20,20);
unAutreBouton.setLocation(50,50) ;
unLabel.setLocation(100,50);
```

Il est préférable d'utiliser la méthode `getPreferredSize` pour être certain d'afficher les textes en entier.

```
unBouton.setSize(unBouton, getPreferredSize());
unAutreBouton.setSize(unBouton, getPreferredSize());
unLabel.setSize(unBouton, getPreferredSize());
```

## Le gestionnaire FlowLayout

- Le gestionnaire le plus élémentaire.
- Il dispose les différents composants de gauche à droite et de haut en bas. Pour cela il remplit une ligne de composants puis passe à la suivante. Le placement peut suivre plusieurs justifications, notamment à gauche, au centre et à droite. Les justifications sont définies à l'aide de variables statiques `FlowLayout.LEFT`, `FlowLayout.CENTER` et `FlowLayout.RIGHT`.

```
public class TestFlowLayout extends JPanel {
 public TestFlowLayout() {
 FlowLayout fl = new FlowLayout();
 this.setLayout(fl);
 this.add(new JButton ("Un"));
 this.add(new JButton (" Deux"));
 this.add(new JButton (" Trois"));
 this.add(new JButton (" Quatre "));
 this.add(new JButton (" Cinq"));
 this.add(new JButton (" Six"));
 }
}
```

Par défaut, le gestionnaire tente de mettre tous les composants sur une seule ligne.



Si la fenêtre est réduite, une nouvelle ligne de composants est créée.



Les composants qui ne peuvent plus être placés sur la première ligne sont placés sur la seconde ligne.

## Le gestionnaire GridLayout

- Le gestionnaire GridLayout propose de placer les composants sur une grille régulièrement espacée. Généralement les composants sont disposés de gauche à droite puis de haut en bas.

```
public class TestGridLayout extend JPanel {
 public TestGridLayout() {
 this.setLayout(new GridLayout(3 ,0));
 this.add (new JButton ("Un"));
 this.add (new JButton (" Deux"));
 this.add (new JButton (" Trois"));
 this.add (new JButton (" Quatre "));
 this.add (new JButton (" Cinq"));
 this.add (new JButton (" Six"));
 this.add (new JButton (" Sept"));
 }
}
```



|        |      |       |
|--------|------|-------|
| Un     | Deux | Trois |
| Quatre | Cinq | Six   |
| Sept   |      |       |

## Le gestionnaire BorderLayout

- Le gestionnaire BorderLayout définit 5 zones dans le conteneur : une zone centrale (CENTER) et quatre zones périphériques (EAST, WEST, NORTH, SOUTH).

```
public class TestBorderLayout extends JPanel {
 public TestBorderLayout() {
 this.setLayout(new BorderLayout());
 this.add (new JButton (" North"), BorderLayout. NORTH);
 this.add (new JButton (" South"), BorderLayout. SOUTH);
 this.add (new JButton (" East"),BorderLayout. EAST);
 this.add (new JButton (" West"),BorderLayout. WEST);
 this.add (new JButton (" Center "), BorderLayout. CENTER);
 }
}
```



Il n'est pas obligatoire de placer cinq composants dans le conteneur, les emplacements non pourvus seront alors ignorés.



# Le gestionnaire GridBagLayout

- La mise en page la plus complète. Comme GridLayout, elle est basée sur la définition de grilles. Par contre, les composants n'ont **pas forcément des dimensions identiques** : ils peuvent occuper une ou plusieurs cases de la grille.

- Etapes:

## 1) Créer un objet GridBagLayout

GridBagLayout

```
gbl = new GridBagLayout();
setLayout(gbl);
```

## 2) Créer un objet GridBagConstraints

```
GridBagConstraints gbc = new GridBagConstraints();
gbl.setConstraints(component, gbc);
add(component);
```

# Propriétés du GridBagConstraints:

- **gridx** et **gridy**: définir les coordonnées de la cellule dans la zone d'affichage

|     |     |     |
|-----|-----|-----|
| 0,0 | 1,0 | 2,0 |
| 0,1 | 1,1 | 2,1 |
| 0,2 | 1,2 | 2,2 |
| 0,3 | 1,3 | 2,3 |

Certaines cellules peuvent rester vides

- **gridwidth** et **gridheight**: définir respectivement le nombre de cases en colonnes et en lignes du composant courant (valeur par défaut: 1)
- **anchor**: définir le point d'ancrage d'un composant dans la ou les cellules qu'il occupe



9 points d'ancrage:

|                  |            |                |
|------------------|------------|----------------|
| FIRST_LINE_START | PAGE_START | FIRST_LINE_END |
| LINE_START       | CENTER     | LINE_END       |
| LAST_LINE_START  | PAGE_END   | LAST_LINE_END  |

- **fill** : détermine comment utiliser l'espace disponible lorsque la taille du composant est inférieure à celle qui lui est offerte.
  - GrigBagConstraints.NONE pour ne pas redimensionner le composant.
  - GrigBagConstraints.HORIZONTAL pour remplir l'espace horizontal offert.
  - GrigBagConstraints.VERTICAL pour remplir l'espace vertical offert.
  - GrigBagConstraints.BOTH pour remplir l'espace offert, horizontalement et verticalement.

- **weightx** et **weighty** déterminent comment se répartit l'espace supplémentaire entre les composants (pard éfaut: 0 —> les composants sont centrés horizontalement )

# Example:

```
public class TestGridBag extends JFrame
{ void addButton(String label,GridBagConstraints constraints,GridBagLayout layout){
 JButton button = new JButton(label);
 layout.setConstraints(button, constraints);
 getContentPane().add(button);
}
TestGridBag() {
 setTitle("TestGridBag");
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 GridBagLayout layout = new GridBagLayout();
 getContentPane().setLayout(layout);
 GridBagConstraints cons = new GridBagConstraints();
 cons.gridx = 0;
 cons.gridy = 0;
 cons.gridwidth = 1;
 cons.gridheight = 2;
 cons.fill = GridBagConstraints.NONE;
 cons.anchor = GridBagConstraints.CENTER;
 cons.weightx = 1;
 cons.weighty = 2;
 addButton("Un", cons, layout);
 cons.gridx = 1;
 cons.gridy = 0;
 cons.gridwidth = 1;
 cons.gridheight = 2;
 cons.fill = GridBagConstraints.VERTICAL;
 cons.anchor = GridBagConstraints.EAST;
 cons.weightx = 1;
 cons.weighty = 2;
 addButton("Deux", cons, layout);
 cons.gridx = 2;
 cons.gridy = 0;
 cons.gridwidth = 2;
 cons.gridheight = 2;
 cons.fill = GridBagConstraints.HORIZONTAL;
 cons.anchor = GridBagConstraints.NORTH;
 cons.weightx = 2;
 cons.weighty = 2;
```

```
 addButton("Trois", cons, layout);
 cons.gridx = 0;
 cons.gridy = 2;
 cons.gridwidth = 1;
 cons.gridheight = 1;
 cons.fill = GridBagConstraints.BOTH;
 cons.anchor = GridBagConstraints.CENTER;
 cons.weightx = 1;
 cons.weighty = 1;
 addButton("Quatre", cons, layout);
 cons.gridx = 1;
 cons.gridy = 2;
 cons.gridwidth = 1;
 cons.gridheight = 1;
 cons.fill = GridBagConstraints.NONE;
 cons.anchor = GridBagConstraints.SOUTHWEST;
 cons.weightx = 1;
 cons.weighty = 1;
 addButton("Cinq", cons, layout);
 cons.gridx = 2;
 cons.gridy = 2;
 cons.gridwidth = 1;
 cons.gridheight = 1;
 cons.fill = GridBagConstraints.BOTH;
 cons.anchor = GridBagConstraints.CENTER;
 cons.weightx = 2;
 cons.weighty = 1;
 addButton("Six", cons, layout);
```

```
 }
}
```

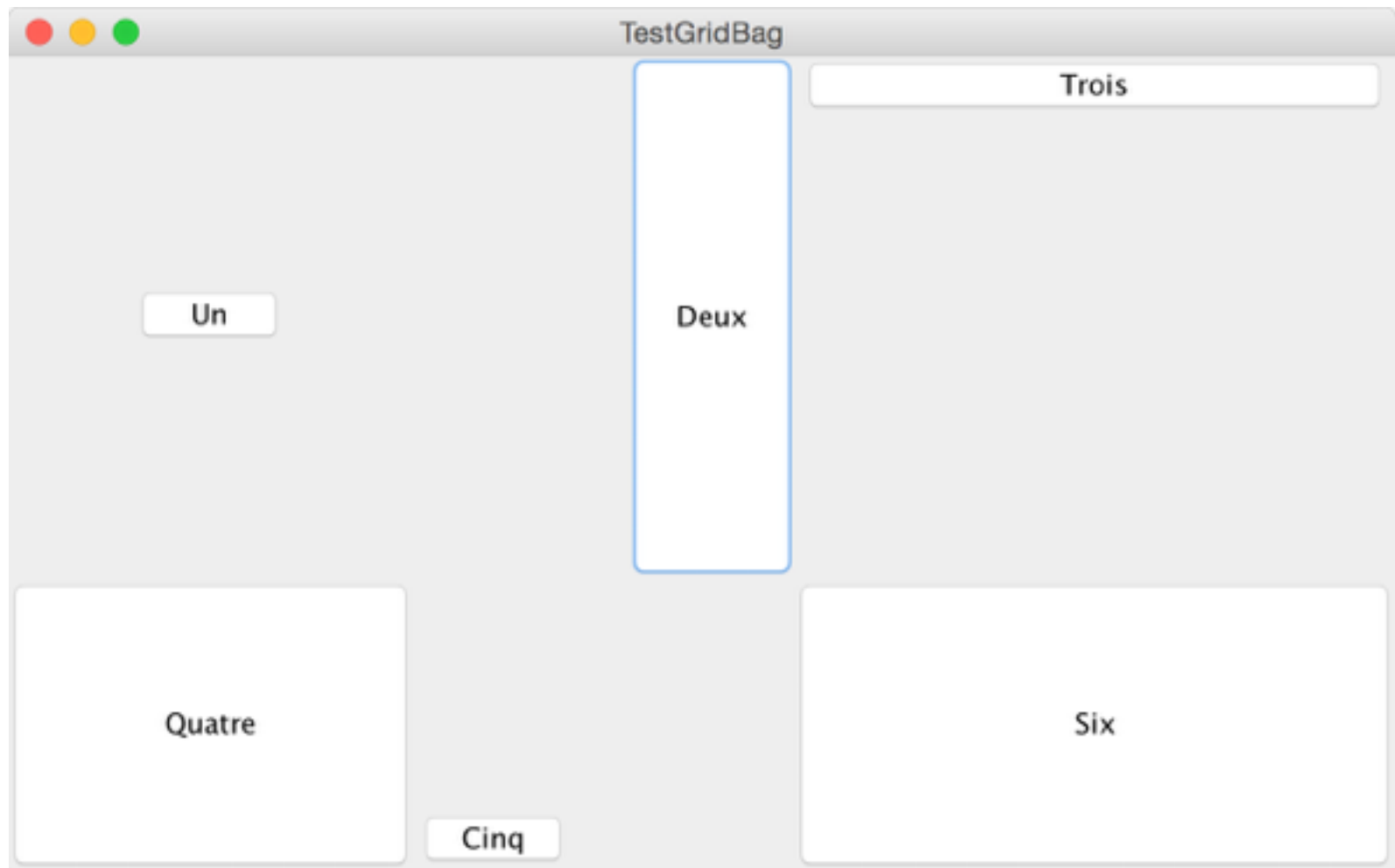


```

public class Gridbag {

 public static void main(String[] args) {
 JFrame f= new TestGridBag();
 Toolkit theKit = f.getToolkit();
 Dimension wndSize = theKit.getScreenSize();
 f.setBounds(wndSize.width/4, wndSize.height/4,wndSize.width/2, wndSize.height/2);
 f.setVisible(true);
 }
}

```



Nous souhaitons afficher le message "Ceci est ma première fenêtre"

## Fenêtre avec JLabel

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

public class Swing1 extends JFrame{
 public static void main(String [] args){
 Swing1 f = new Swing1();
 f.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent e){
 System.exit(0);
 }
 });
 f.setBounds(100,100, 300, 200);
 JLabel texte = new JLabel(" Ceci est ma première fenêtre");
 f.getContentPane().add(texte);
 f.setVisible(true);
 }
}
```

## Modifier la police des caractères

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

public class Swing1 extends JFrame{
 public static void main(String [] args){
 Swing1 f = new Swing1();
 f.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent e){
 System.exit(0);
 }
 });
 f.setBounds(100,100, 300, 200);
 JLabel texte = new JLabel(" Ceci est ma première fenêtre");

 Font fo = new Font("Times", Font.BOLD,20);
 texte.setFont(fo);

 f.getContentPane().add(texte);
 f.setVisible(true);
 }
}
```

## Ajouter un JButton

```
import javax.swing.*;
import java.awt.event.*;
public class Swing1 extends JFrame{
 public static void main(String [] args){
 Swing1 f = new Swing1();
 f.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent e){
 System.exit(0);
 }
 });
 f.setBounds(100,100, 300, 200);
 JLabel texte = new JLabel(" Ceci est ma première fenêtre");
 Font fo = new Font("Times", Font.BOLD,20);
 texte.setFont(fo);
 f.getContentPane().add(texte);
 JButton btn = new JButton("OK");
 f.getContentPane().add(btn);
 f.setVisible(true);
 }
}
```

Où est passé le label ?

```

import javax.swing.*;import java.awt.event.*;import java.awt.*;
class Fenetre1 extends JFrame{
 JLabel etq;
 JButton b1, b2;
 public Fenetre1 (String titre) {
 super(titre);
 this.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent e){
 System.exit(0); }});
 this.getContentPane().setLayout(new
FlowLayout(FlowLayout.CENTER, 30, 30));
 this.setBounds(100,100, 300, 200);
 this.etq = new JLabel(" Ceci est ma deuxième fenêtre");
 this.b1 = new JButton("OK");
 this.b2 = new JButton("Annuler");
 this.getContentPane().add(this.etq);
 this.getContentPane().add(this.b1);
 this.getContentPane().add(this.b2);
 }
}

public static void main(String[] args) {
 Fenetre1 f = new Fenetre1("Fenêtre avec FlowLayout");
 f.setVisible(true);
}
}

Redimensionner la fenêtre! Ajouter f.setResizable(false);

```

## **Ajuster la taille de la fenêtre**

- Pour un affichage efficace, l'exemple précédent nécessite l'ajustement dynamique des dimensions de la fenêtre.
- Pour cela, il faut obtenir
  - \* les dimensions de l'écran
  - \* les dimensions du JLabel
  - \* calculer les nouvelles dimensions pour ajuster la fenêtre
  - \* recentrer la fenêtre en recalculant ses coordonnées x et y
- On utilise la méthode `setBounds` de `JFrame` pour ajuster la fenêtre

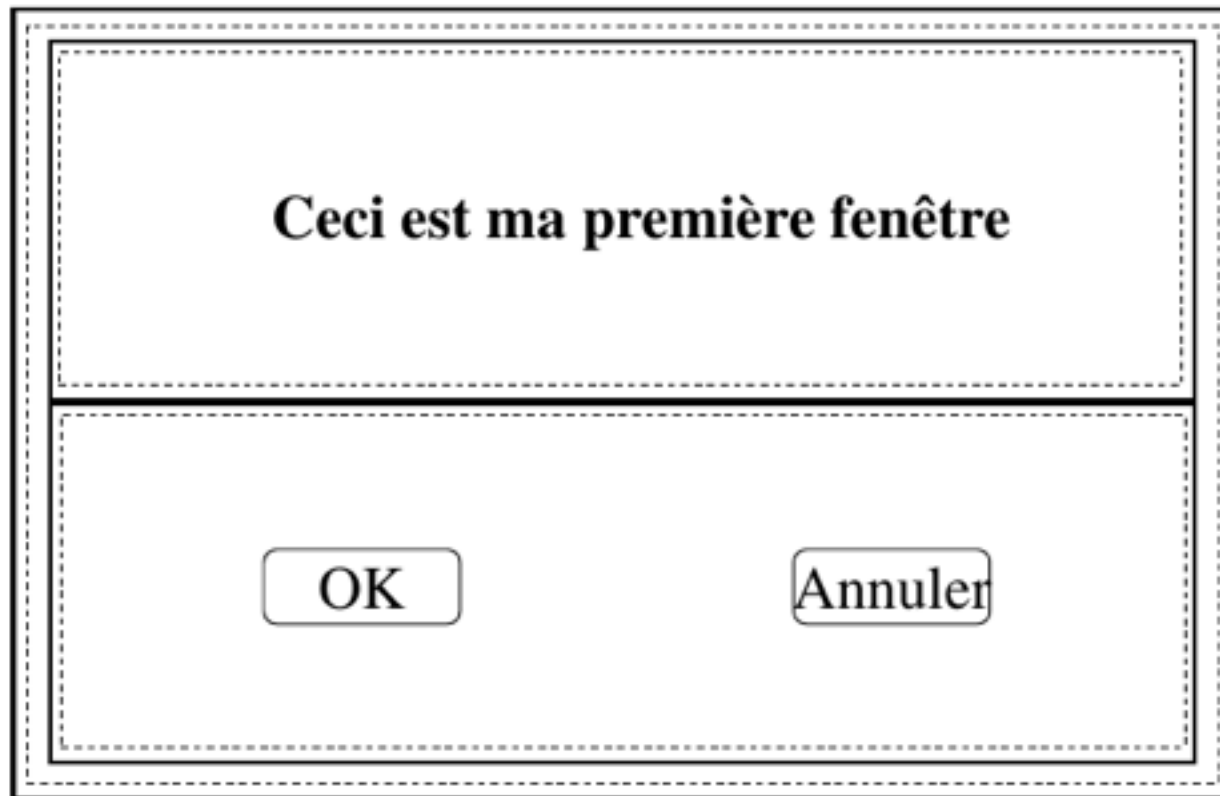
## Ajouter cette méthode à la classe Fenetre1

```
Rectangle calculerLimites(JLabel b){
 //taille de l'écran
 Dimension tailleEcran = Toolkit.getDefaultToolkit().getScreenSize();
 int sw = tailleEcran.width; //largeur
 int sh = tailleEcran.height; //hauteur
 int pw = (b.getPreferredSize()).width; //largeur du label
 int ph = (b.getPreferredSize()).height; //hauteur du label
 int w = (int) (pw+60); //nouvelle largeur= pw plus une marge 30 gauche, 30 droite
 int h = (int) (ph * 5); //la nouvelle hauteur = ph fois 5 pour laisser de l'espace
 int x = (sw - w)/2; //abscisse de la fenêtre (au centre)
 int y = (sh - h) / 2; //ordonnée de la fenêtre (au centre)
 return new Rectangle(x, y, w, h); //nouvelles dimensions
}
```

Insérer l'appel `this.setBounds(calculerLimites(etq));`

## Utiliser des Panels

- Pour remédier au problème précédent, on peut créer deux JPanel qu'on ajoute à la contentPane
- On associe à la contentPane un GridLayout créant un tableau de 2 lignes et une colonne
- Pour chaque Panel, on associe un FlowLayout





```

import javax.swing.*; import java.awt.event.*; import java.awt.*;
class Fenetre2 extends JFrame{
 public Fenetre2 (String titre) {
 super(titre);
 this.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent e){ System.exit(0);});
 this.getContentPane().setLayout(new GridLayout(2,1,0, 20));
 JPanel p1 = créerPanel1(); JPanel p2 = créerPanel2();
 this.getContentPane().add(p1); this.getContentPane().add(p2);
 ajuster(); //positionne la fenetre dans l'écran
 setSize(getLayout().preferredLayoutSize(this));
 }
 JPanel créerPanel1() {
 JPanel p = new JPanel();
 p.setLayout(new FlowLayout(FlowLayout.CENTER,0, 20));
 p.add(new JLabel("Ceci est ma première fenetre"));
 return p;
 }
 JPanel créerPanel2() {
 JPanel p = new JPanel();
 p.setLayout(new FlowLayout(FlowLayout.CENTER, 40, 0));
 p.add(new JButton("OK"));
 p.add(new JButton("Annuler"));
 return p;
 }
 void ajuster(){
 Dimension tailleEcran = Toolkit.getDefaultToolkit().getScreenSize();
 int sw = tailleEcran.width;
 int sh = tailleEcran.height;
 int w = (int) (getSize().width + 60);
 int h = (int) (getSize().height * 10);
 int x = (sw - w) / 2;
 int y = (sh - h) / 2;
 setLocation(x,y);
 }
}

```

## Ajouter de l'interactivité aux boutons

- Nous souhaitons maintenant traiter les clics de souris sur les deux boutons
  - \* un clic sur le bouton "Texte" entraîne la modification de la couleur du texte du label
  - \* un clic sur le bouton "Fond" entraîne la modification de la couleur du fond du JPanel contenant le label
- Pour cela, il faut créer et associer un listener à chaque objet
  - \* le listener sera de type MouseAdapter
  - \* l'événement à traiter sera

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

## Listeners de MouseEvent

```
import javax.swing.*; import java.awt.event.*; import java.awt.*;
class Fenetre3 extends JFrame{
 JPanel p1;
 JLabel texte;
 public Fenetre3 (String titre) {
 super(titre);
 this.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent e){ System.exit(0);}});

 this.getContentPane().setLayout(new GridLayout(2,1,10, 20));
 p1 = créerPanel1(); JPanel p2 = créerPanel2();
 this.getContentPane().add(p1); this.getContentPane().add(p2);

 p1.setSize(p2.getSize());
 ajuster();
 setSize(getLayout().preferredLayoutSize(this));
 }
 JPanel créerPanel1() {
 JPanel p = new JPanel();
 p.setLayout(new FlowLayout(FlowLayout.CENTER,0, 20));
 texte = new JLabel("Ceci est ma première fenêtre");
 p.add(texte);
 setSize(getLayout().preferredLayoutSize(p));
 return p;
 }
}
```

## Les boutons et leurs Listeners

```
JPanel créerPanel2() {
 JPanel p = new JPanel();
 p.setLayout(new GridLayout(2,1,10, 20));
 JButton b1 = new JButton ("Texte");
 b1.addMouseListener(new MouseAdapter(){
public void mouseClicked(MouseEvent e){
 texte.setForeground(new Color(255,0,0));
 Texte.repaint() ;
 }}) ;
 JButton b2 = new JButton ("Fond");
 b2.addMouseListener(new MouseAdapter(){
 public void mouseClicked(MouseEvent e){
 p1.setBackground(new Color(0,0,255));
 p1.repaint();
 });
 p.add(b1);
 p.add(b2);
 return p;
}
...
```

## Suite

```
void ajuster(){
 Dimension tailleEcran =
Toolkit.getDefaultToolkit().getScreenSize();
 int sw = tailleEcran.width;
 int sh = tailleEcran.height;
 int w = (int) (getSize().width + 60);
 int h = (int) (getSize().height * 10);
 int x = (sw - w) / 2;
 int y = (sh - h) / 2;
 setLocation(x,y);
}
}

public static void main(String [] args){
 Fenetre3 f = new Fenetre3("Cette fenêtre utilise 2 panels");
 f.setVisible(true);
}
```

## Création de menu

```
void ajouterMenus(JFrame frame){
 JMenu fich = new JMenu("Fichier",true);
 JMenuItem ouv = new JMenuItem("Ouvrir");
 ouv.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent e){
 //ouvrir();
 }});
 fich.add(ouv);
 JMenuItem ferm = new JMenuItem("Fermer");
 ferm.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent e){
 //fermer();
 }
 });
 fich.add(ferm);
 fich.add(new JSeparator());
 JMenuItem quit = new JMenuItem("Quitter");
 quit.addActionListener(new ActionListener(){
 public void actionPerformed(ActionEvent e){
 System.exit(0);
 }
 });
 fich.add(quit);
 JMenuBar mb = new JMenuBar();
 mb.add(fich);
 frame.setJMenuBar(mb);
}
```

## Création d'un JTextField

```
JTextField tf;
```

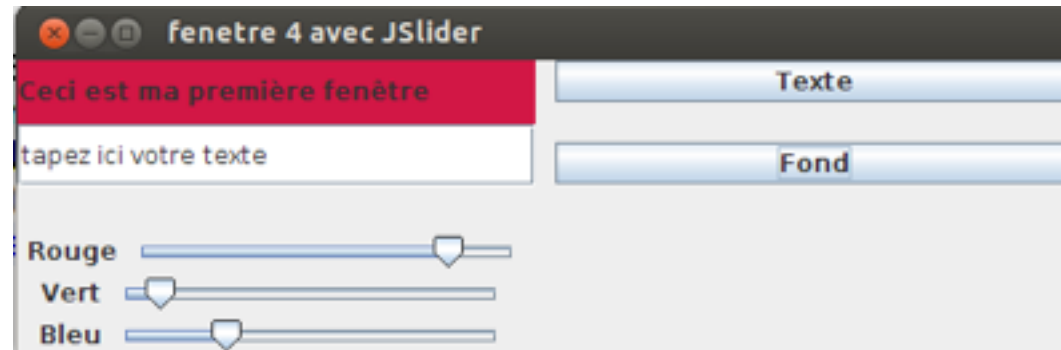
```
tf = new JTextField("texte par défaut");
p.add(tf); // ajouter la zone texte au pannel p
```

```
tf.getText() ; // récupérer le texte tapé dans la zone
```

Modifiez l'exemple de la classe Fenetre3 pour y ajouter un JTextField. Le bouton texte permettra de récupérer le contenu du JTextField et de l'afficher dans le JLabel.

## Exercice :

Réaliser la fenêtre suivante.





## **Exercice :**

Il s'agit de créer un programme qui possède une fenêtre principale où l'on tape du texte, et une fenêtre secondaire où le texte est envoyé, ligne par ligne.

La fenêtre principale contiendra une zone de texte éditable où l'utilisateur tapera la ligne à envoyer, ainsi qu'un bouton d'envoi. Elle permettra aussi de fermer le programme.

## **Exercice :**

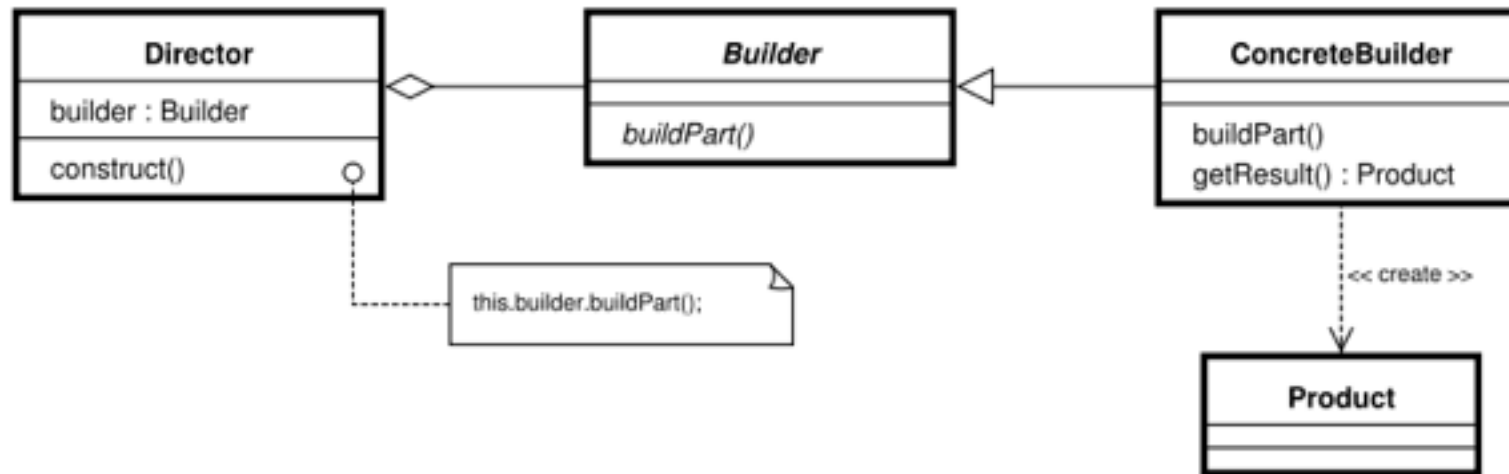
Créer une interface graphique qui permet, à l'aide d'un menu, de sélectionner un fichier. Le contenu de ce fichier sera affiché dans une zone de texte. Ensuite, le contenu du fichier peut être modifié, et le fichier peut être sauvegardé sur le disque avec le menu « sauvegarder ».

## Le pattern Monteur (*Builder*)

- Une classe offrant des moyens de construction d'un objet (complexe). Par exemple, pour construire un dessin il faut ajouter des points, des lignes, des cercles.... Il ne doit pas être confondu avec la Fabrique.
- Le problème d'une Fabrique de création (*Factory*), c'est qu'elle ne permet pas de définir comment un objet va être construit. Certes, il est toujours possible de passer plusieurs paramètres dans la méthode de création d'une fabrique mais cela s'avère souvent très délicat pour la maintenance. En plus, cela fait exploser de façon exponentielle la liste des constructeurs.
- Ce modèle est intéressant lorsque l'algorithme de création d'un objet complexe doit être indépendant des constituants de l'objet et de leurs relations, ou lorsque différentes représentations de l'objet construit doivent être possibles.

### Conséquences

- Variation possible de la représentation interne d'un produit
  - \* l'implémentation des produits et de leurs composants est cachée au Director
  - \* Ainsi la construction d'un autre objet revient à définir un nouveau Builder
- Isolation du code de construction et du code de représentation du reste de l'application
- Meilleur contrôle du processus de construction



**Builder** : interface pour créer les parties de l'objet Product

**ConcreteBuilder** : construit et assemble les parties du produit en implémentant l'interface Builder

**Director** : construit un objet en utilisant l'interface Builder

**Product** : représente l'objet complexe sous construction

## **Example : (pseudocode)**

**class** Car **is**

Can have GPS, trip computer and various numbers of seats. Can be a city car, a sports car, or a cabriolet.

**class** CarBuilder **is**

**method** getResult() **is**

output: a Car with the right options

Construct and return the car.

**method** setSeats(number) **is**

input: the number of seats the car may have.

Tell the builder the number of seats.

**method** setCityCar() **is**

Make the builder remember that the car is a city car.

**method** setCabriolet() **is**

Make the builder remember that the car is a cabriolet.

**method** setSportsCar() **is**

Make the builder remember that the car is a sports car.

**method** setTripComputer() **is**

Make the builder remember that the car has a trip computer.

**method** unsetTripComputer() **is**

Make the builder remember that the car does not have a trip computer.

**method** setGPS() **is**

Make the builder remember that the car has a global positioning system.

**method** unsetGPS() **is**

Make the builder remember that the car does not have a global positioning system.

Construct a CarBuilder called carBuilder

carBuilder.setSeats(2)

carBuilder.setSportsCar()

carBuilder.setTripComputer()

carBuilder.unsetGPS()

car := carBuilder.getResult()

## Aperçu sur les expressions Lambda

- JAVA 8, qui introduit les expressions Lambda, devient un langage ouvert à l'orientation vers les fonctions et moins regardant quant au caractère "fortement typé" qui définit ce langage.
- Les fonctions Lambda décrivent en informatique, par extension ou abus de langage, l'ensemble des fonctions anonymes (fonction qui n'est pas définie par une déclaration : nom, paramètres, valeur de retour, types ...).
- Avantages :
  - \* utilisation de la fonction à la volée
  - \* réduction des lignes de code
  - \* lecture simplifiée, aucun besoin de se référer à la déclaration de la fonction
  - \* facilitation de l'imbrication (l'appel d'une fonction dans une autre)
- L'utilisation des fonctions Lambda pourrait remettre en cause la définition de JAVA comme langage orienté objet (ce ne sont plus des objets qui sont passés en paramètres mais des fonctions) ainsi que le typage strict.

## Classes imbriquées

- Lors de l'utilisation de Swing, il est plus simple et plus lisible de décrire les actions d'un bouton comme ceci :

```
 JButton testButton = new JButton("Test Button");
 testButton.addActionListener(new ActionListener()
 {
 public void actionPerformed(ActionEvent ae)
 {
 System.out.println("Click Detected by A non Class");
 }
 });
```

- Un objet ActionListener est utilisé en redéfinissant sa méthode actionPerformed à la volée plutôt que de créer une classe à part qui implémente l'interface ActionListener. Le code reste tout de même un peu « lourd ».
- De plus l'interface ActionListener ne contient qu'une méthode abstraite, ce qui en fait une interface « fonctionnelle ». Cette implémentation dans le code peut être facilement remplacée par une expression Lambda.

## Classes imbriquées...

- Une expression Lambda se compose de paramètres, d'un symbole flèche et d'un corps (exécution).

| Liste des Arguments | Symbole Flèche | Corps |
|---------------------|----------------|-------|
| (int x, int y)      | ->             | x + y |

L'exemple précédent peut être ré-écrit comme suit :

```
JButton testButton = new JButton("Test Button");
testButton.addActionListener(e ->
{
 System.out.println("Click Detected by A non Class");
});
```

- Le type de “e” dépend du contexte. Ce qui nous permet aussi de supprimer éventuellement une ligne d'import devenue inutile.

# Le passage de valeurs

- Exemple : une simple liste d'entier de 1 à 6.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);
```

- Une méthode pour en faire la somme:

```
public int sumAll(List<Integer> numbers)
{ int total = 0;
 for (int number : numbers)
 {total += number;}
 return total;
}
```

- Finalement nous souhaitons aussi une méthode pour faire la somme des nombres pairs :

```
public int sumAllEven(List<Integer> numbers)
{ int total = 0;
 for (int number : numbers)
 {
 if (number % 2 == 0)
 {total += number;}
 }
 return total;
}
```



## Le passage de valeurs...

**Question** : Que faire dans le cas où il vous est demandé de rajouter un calcul de somme supplémentaire ?

### Réponses

- Dans un cas classique, l'idée sera d'adapter la méthode pour qu'elle puisse effectuer toutes les sommes.
- Dans le cas des expressions lambda il est aussi possible d'utiliser les « Prédicats » (Predicate<T>) : (java.util.function)

```
public int sumAll(List<Integer> numbers, Predicate<Integer> p)
{
 int total = 0;
 for (int number : numbers)
 {
 if (p.test(number))
 {
 total += number;
 }
 }
 return total;
}
```

- « Predicate » est une nouvelle interface fonctionnelle ajoutée à JAVA 8.

On utilise une méthode rendue plus générique grâce au prédicat et expressions Lambda de la manière suivante :

```
sumAll(numbers, n -> true);
sumAll(numbers, n -> n % 2 == 0);
sumAll(numbers, n -> n > 3);
```