

# Java et fenêtre

## avec Awt

---

### IHM avec Java

Java, comme tout langage moderne, permet de créer des applications qui ressemblent à l'interface du système d'exploitation. Cette assurance d'ergonomie et d'interactivité avec l'utilisateur est le minimum qu'un utilisateur demande à une application. Les interfaces homme-machine (dénommées IHM) font intervenir de nos jours des éléments que l'on retrouve dans la majorité des systèmes d'exploitation : les fenêtres, les menus déroulants, les boutons, etc...

Ce chapitre traite en résumé, mais en posant toutes les bases, de l'aptitude de Java à élaborer une IHM. Nous regroupons sous le vocable d'IHM Java, les applications disposant d'une interface graphique et les applets que nous verrons plus loin.

### Le package AWT

C'est pour construire de telles IHM que le package AWT (Abstract Window Toolkit) est inclus dans toutes les versions de Java. Ce package est la base des extensions ultérieures comme **Swing**, mais est le seul qui fonctionne sur toutes les générations de navigateurs.

Les classes contenues dans AWT dérivent (héritent) toutes de la classe **Component**, nous allons étudier quelques classes minimales pour construire une IHM standard.

### Les classes Conteneurs

Ces classes sont essentielles pour la construction d'IHM Java elles dérivent de la classe **java.awt.Container**, elles permettent d'intégrer d'autres objets visuels et de les organiser à l'écran.

#### *Hierarchie de la classe Container :*

```
java.lang.Object
|
+--java.awt.Component
|
+--java.awt.Container
```

Voici la liste extraite du JDK des sous-classes de la classe **Container** autres que **Swing** : **Panel**, **ScrollPane**, **Window**.

*Les principales classes conteneurs :*

Classe	Fonction
<pre> +--java.awt.Container           +--<b>java.awt.Window</b> </pre>	Crée des rectangles simples sans cadre, sans menu, sans titre, mais ne permet pas de créer directement une fenêtre Windows classique.
<pre> +--java.awt.Container           +--<b>java.awt.Panel</b> </pre>	Crée une surface sans bordure, capable de contenir d'autres éléments : boutons, panel etc...
<pre> +--java.awt.Container           +--<b>java.awt.ScrollPane</b> </pre>	Crée une barre de défilement horizontale et/ou une barre de défilement verticale.

*Les classes héritées des classes conteneurs :*

Classe	Fonction
<pre> java.awt.Window           +--<b>java.awt.Frame</b> </pre>	Crée des fenêtres avec bordure, pouvant intégrer des menus, avec un titre, etc...comme toute fenêtre Windows classique. C'est le conteneur de base de toute application graphique.
<pre> java.awt.Window           +--<b>java.awt.Dialog</b> </pre>	Crée une fenêtre de dialogue avec l'utilisateur, avec une bordure, un titre et un bouton-icône de fermeture.

Une première fenêtre construite à partir d'un objet de classe **Frame**; une fenêtre est donc un objet, on pourra donc créer autant de fenêtres que nécessaire, il suffira à chaque fois d'instancier un objet de la classe **Frame**.

*Quelques méthodes de la classe **Frame**, utiles au départ :*

Méthodes	Fonction
<b>public void</b> setSize( <b>int</b> width, <b>int</b> height)	retaille la largeur (width) et la hauteur (height) de la fenêtre.
<b>public void</b> setBounds( <b>int</b> x, <b>int</b> y, <b>int</b> width, <b>int</b> height)	retaille la largeur (width) et la hauteur (height) de la fenêtre et la positionne en x,y sur l'écran.

<b>public</b> Frame( <b>String</b> title) <b>public</b> Frame( )	Les deux constructeurs d'objets Frame, celui qui possède un paramètre String écrit la chaîne dans la barre de titre de la fenêtre.
<b>public void</b> setVisible(boolean b )	Change l'état de la fenêtre en mode <b>visible</b> ou <b>invisible</b> selon la valeur de b.
<b>public void</b> hide( )	Change l'état de la fenêtre en mode <b>invisible</b> .
Différentes surcharges de la méthode add : <b>public</b> <b>Component</b> add( <b>Component</b> comp) etc...	Permettent d'ajouter un composant à l'intérieur de la fenêtre.

Une Frame lorsque son constructeur la crée est en mode invisible, il faut donc la rendre visible, c'est le rôle de la méthode **setVisible ( true)** que vous devez appeler afin d'afficher la fenêtre sur l'écran :

Programme Java
<pre>import java.awt.*; class AppliWindow {     public static void main(String [ ] arg) {         Frame fen = new Frame ("Bonjour" );         fen. setVisible ( true );     } }</pre>

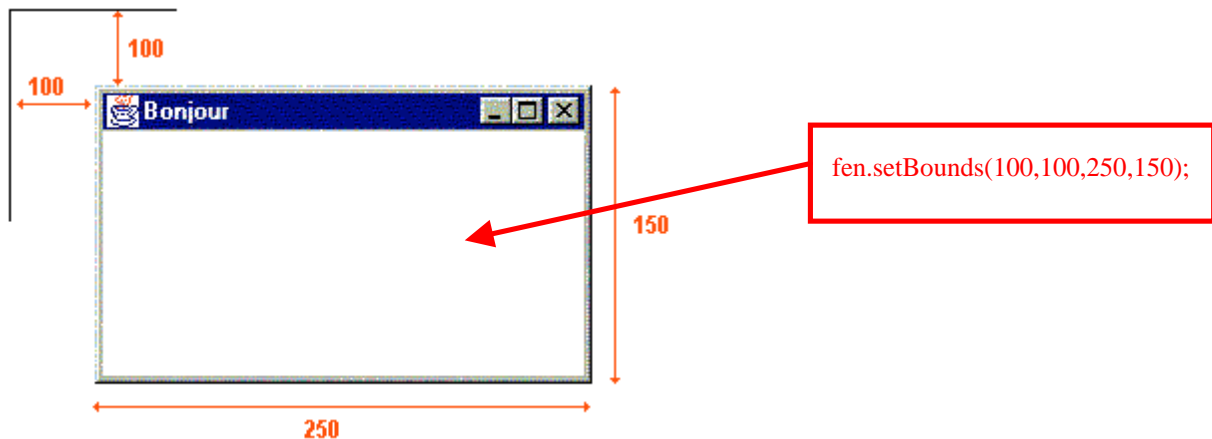
Ci-dessous la fenêtre affichée par le programme précédent :



*Cette fenêtre est trop petite, retailons-la avec la méthode **setBounds** :*

Programme Java
<pre>import java.awt.*; class AppliWindow {     public static void main(String [ ] arg) {         Frame fen = new Frame ("Bonjour" );         fen.setBounds(100,100,250,150);         fen. setVisible ( true );     } }</pre>

Ci-dessous la fenêtre affichée par le programme précédent :



Pour l'instant nos fenêtres sont repositionnables, retaillables, mais elles ne contiennent rien, comme ce sont des objets conteneurs, il est possible en particulier, d'y inclure des composants.

Il est possible d'afficher des fenêtres dites de dialogue de la classe `Dialog`, dépendant d'une `Frame`. Elles sont très semblables aux `Frame` (barre de titre, cadre,...) mais ne disposent que d'un bouton icône de fermeture dans leur titre :

#### une fenêtre de classe `Dialog` :



De telles fenêtres doivent être obligatoirement rattachées lors de la construction à un parent qui sera une `Frame` ou une autre boîte de classe `Dialog`, le constructeur de la classe `Dialog` contient plusieurs surcharges dont la suivante :

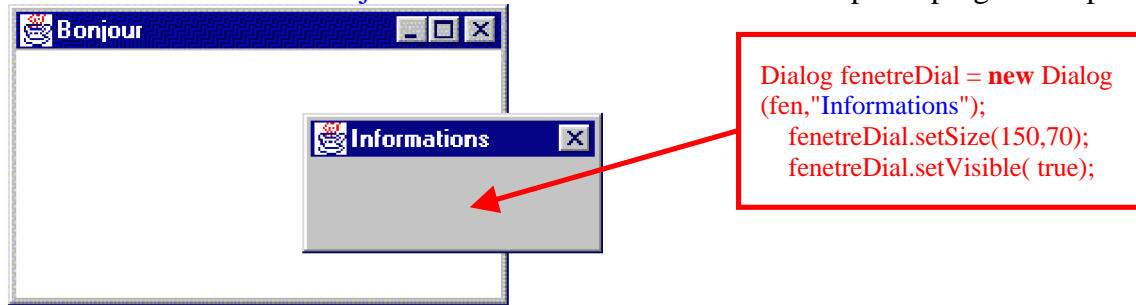
**`public Dialog(Frame owner, String title)`**

où `owner` est la `Frame` qui va appeler la boîte de dialogue, `title` est la string contenant le titre de la boîte de dialogue. Il faudra donc appeler le constructeur `Dialog` avec une `Frame` instanciée dans le programme.

*Exemple d'affichage d'une boîte informations à partir de notre fenêtre "Bonjour" :*

Programme Java
<pre>import java.awt.*; class AppliWindow {     public static void main(String [ ] arg) {         Frame fen = new Frame ("Bonjour" );         fen.setBounds(100,100,250,150);         Dialog fenetreDial = new Dialog (fen,"Informations");         fenetreDial.setSize(150,70);         fenetreDial.setVisible( true);         fen. setVisible ( true );     } }</pre>

Ci-dessous les fenêtres **Bonjour** et la boîte **Informations** affichées par le programme précédent :



## Composants déclenchant des actions

Ce sont essentiellement des classes directement héritées de la classe **java.awt.Container**. Les menus dérivent de la classe **java.awt.MenuComponent**. Nous ne détaillons pas tous les composants possibles, mais certains les plus utiles à créer une interface Windows-like.

*Composants permettant le déclenchement d'actions :*

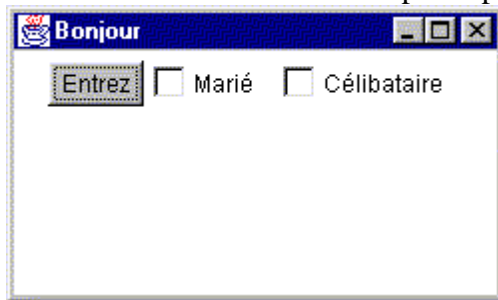
Les classes composants	Fonction
<pre>java.lang.Object   +--java.awt.MenuComponent   +--<b>java.awt.MenuBar</b></pre>	Création d'une barre des menus dans la fenêtre.
<pre>java.lang.Object   +--java.awt.MenuComponent   +--<b>java.awt.MenuItem</b></pre>	Création des zones de sous-menus d'un menu principal de la classique barre des menus.
<pre>java.lang.Object   +--java.awt.MenuComponent   +--java.awt.MenuItem   +--<b>java.awt.Menu</b></pre>	Création d'un menu principal classique dans la barre des menus de la fenêtre.
<pre>java.lang.Object   +--java.awt.Component   +--<b>java.awt.Button</b></pre>	Création d'un bouton poussoir classique (clicquable par la souris)
<pre>java.lang.Object   +--java.awt.Component   +--<b>java.awt.Checkbox</b></pre>	Création d'un radio bouton, regroupable éventuellement avec d'autres radio boutons.

Enrichissons notre fenêtre précédente d'un bouton poussoir et de deux radio boutons :

```
Programme Java


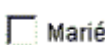

import java.awt.*;
class AppliWindow
{
    public static void main(String [ ] arg) {
        Frame fen = new Frame ("Bonjour" );
        fen.setBounds(100,100,250,150);
        fen.setLayout(new FlowLayout( ));
        Button entree = new Button("Entrez");
        Checkbox bout1 = new Checkbox("Marié");
        Checkbox bout2 = new Checkbox("Célibataire");
        fen.add(entree);
        fen.add(bout1);
        fen.add(bout2);
        fen.setVisible ( true );
    }
}
```

Ci-dessous la fenêtre affichée par le programme précédent :



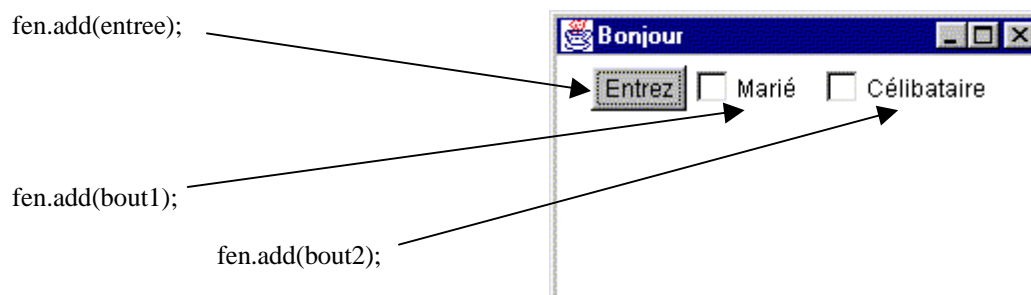
### Remarques sur le programme précédent :

#### 1) Les instructions

- `Button entree = new Button("Entrez");` → 
- `Checkbox bout1 = new Checkbox("Marié");` → 
- `jCheckbox bout2 = new Checkbox("Célibataire");` → 

servent à **créer** un bouton poussoir (classe Button) et deux boutons radio (classe CheckBox), chacun avec un libellé.

#### 2) Les instructions



servent à **ajouter** les objets créés au conteneur (la fenêtre fen de classe Frame).

### 3) L'instruction

- `fen.setLayout(new FlowLayout( ));`

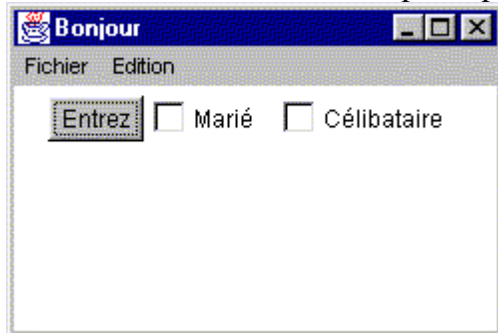
sert à **positionner** les objets visuellement dans la fenêtre les uns à côté des autres, nous en dirons un peu plus sur l'agencement visuel des composants dans une fenêtre.

Terminons la personnalisation de notre fenêtre avec l'introduction d'une barre des menus contenant deux menus : "fichier" et "édition" :

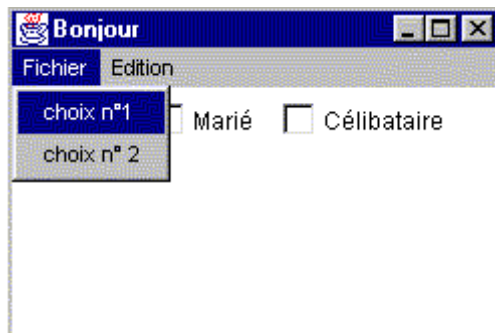
```
Programme Java

import java.awt.*;
class AppliWindow
{
    public static void main(String [ ] arg) {
        Frame fen = newFrame ("Bonjour" );
        fen.setBounds(100,100,250,150);
        fen.setLayout(new FlowLayout( ));
        Button entree = new Button("Entrez");
        Checkbox bout1 = new Checkbox("Marié");
        Checkbox bout2 = new Checkbox("Célibataire");
        fen.add(entree);
        fen.add(bout1);
        fen.add(bout2);
        // les menus :
        MenuBar mbar = new MenuBar( );
        Menu meprinc1 = new Menu("Fichier");
        Menu meprinc2 = new Menu("Edition");
        MenuItem item1 = new MenuItem("choix n° 1");
        MenuItem item2 = new MenuItem("choix n° 2");
        fen.setMenuBar(mbar);
        meprinc1.add(item1);
        meprinc1.add(item2);
        mbar.add(meprinc1);
        mbar.add(meprinc2);
        fen.setVisible ( true );
    }
}
```

Ci-dessous la fenêtre affichée par le programme précédent :



La fenêtre après que l'utilisateur clique sur le menu Fichier



## Remarques sur le programme précédent :

### 1) Les instructions

`MenuBar mbar = new MenuBar( );`

`Menu meprinc1 = new Menu("Fichier");`

`Menu meprinc2 = new Menu("Edition");`

`MenuItem item1 = new MenuItem("choix n°1");`

`MenuItem item2 = new MenuItem("choix n° 2");`



Fichier

Edition

Choix n°1

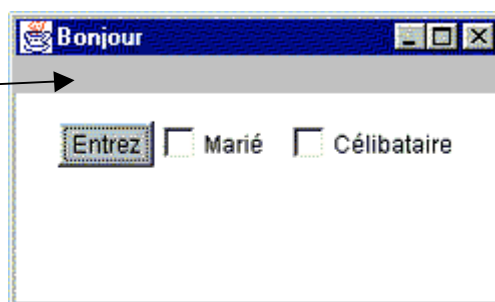
Choix n°2

servent à **créer** une barre de menus nommée *mbar*, deux menus principaux *meprinc1* et *meprinc2*, et enfin deux sous-menus *item1* et *item2*. A cet instant du programme tous ces objets existent mais ne sont pas attachés entre eux, on peut les considérer comme des objets "flottants".

### 2) Dans l'instruction

`fen.setMenuBar(mbar);`

la méthode `setMenuBar` de la classe `Frame` sert à **attacher** (inclure) à la fenêtre *fen* de classe `Frame`, l'objet barre des menus *mbar* déjà créé comme objet "flottant".



### 3) Les instructions

`meprinc1.add(item1);`

`meprinc1.add(item2);`

servent grâce à la méthode `add` de la classe `Menu`, à **attacher** les deux objets flottants de sous-menu nommés *item1* et *item2* au menu principal *meprinc1*.

Fichier

Choix n°1

Fichier

Choix n°1

Choix n°2

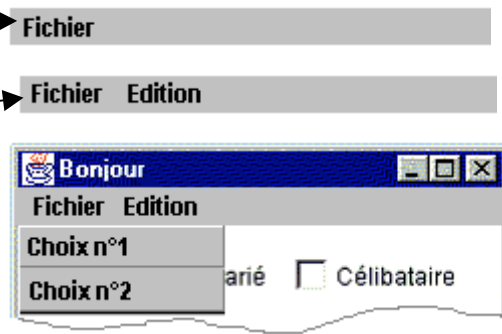


#### 4) Les instructions

`mbar.add(meprinc1);`

`mbar.add(meprinc2);`

servent grâce à la méthode **add** de la classe MenuBar, à **attacher** les deux objets flottants de catégorie menu principal nommés **meprinc1** et **meprinc2**, à la barre des menus **mbar**.



Remarquons enfin ici une application pratique du **polymorphisme dynamique (redéfinition)** de la méthode **add**, elle même **surchargée** plusieurs fois dans une même classe.

## Composants d'affichage ou de saisie

*Composants permettant l'affichage ou la saisie :*

Les classes composants	Fonction
java.awt.Component   +--java.awt.Label	Création d'une étiquette permettant l'affichage d'un texte.
java.awt.Component   +--java.awt.Canvas	Création d'une zone rectangulaire vide dans laquelle l'application peut dessiner.
java.awt.Component   +--java.awt.List	Création d'une liste de chaînes dont chaque élément est sélectionnable.
java.awt.Component   +--java.awt.TextComponent   +--java.awt.TextField	Création d'un éditeur mono ligne.
java.awt.Component   +--java.awt.TextComponent   +--java.awt.TextArea	Création d'un éditeur multi ligne.

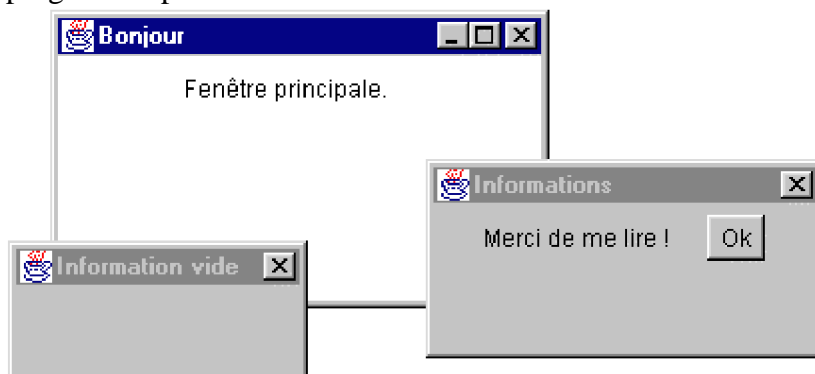
**Ces composants s'ajoutent à une fenêtre après leurs créations, afin d'être visible sur l'écran comme les composants de Button, de CheckBox, etc...**

Ces composants sont à rapprocher quant à leurs fonctionnalités aux classes Delphi de composant standards, nous en donnons la correspondance dans le tableau ci-dessous :

Les classes Java	Les classes Delphi
<code>java.awt.Label</code>	<code>TLabel</code>
<code>java.awt.Canvas</code>	<code>TCanvas</code>
<code>java.awt.List</code>	<code>TListBox</code>
<code>java.awt.TextField</code>	<code>TEdit</code>
<code>java.awt.TextArea</code>	<code>TMemo</code>
<code>java.awt.CheckBox</code>	<code>TCheckBox</code>
<code>java.awt.Button</code>	<code>TButton</code>

### *Exemple récapitulatif :*

Soit à afficher une fenêtre principale contenant le texte "fenêtre principale" et deux fenêtres de dialogue, l'une vide directement instancié à partir de la classe Dialog, l'autre contenant un texte et un bouton, instanciée à partir d'une classe de boîte de dialogue personnalisée. L'exécution du programme produira le résultat suivant :



Nous allons construire un programme contenant **deux classes**, la première servant à définir le genre de boîte personnalisée que nous voulons, la seconde servira à créer une boîte vide et une boîte personnalisée et donc à lancer l'application.

### *Première classe :*

La classe de dialogue personnalisée
<pre>import java.awt.*; class UnDialog extends Dialog {     public UnDialog(Frame mere)     {         super(mere,"Informations");     } }</pre>

```

Label etiq = new Label("Merci de me lire !");
Button bout1 = new Button("Ok");
setSize(200,100);
setLayout(new FlowLayout( ));
add(etiq);
add(bout1);
setVisible ( true );
}
}

```

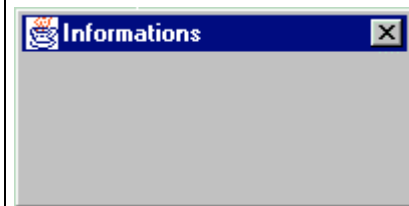
Explications pas à pas des instructions :

Cette classe **UnDialog** ne contient que le constructeur permettant d'instancier des objets de cette classe, elle dérive (hérite) de la classe Dialog < **class** UnDialog **extends** Dialog >

On appelle immédiatement le constructeur de la classe mère (Dialog) par l'instruction < **super**(mere,"Informations"); >

on lui fournit comme paramètres : la Frame propriétaire de la boîte de dialogue et le titre de la future boîte.

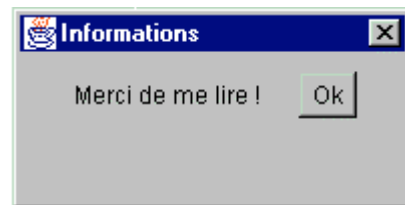
On crée une Label <Label etiq = **new** Label("Merci de me lire !");>,  
 On crée un Button <Button bout1 = **new** Button("Ok");>.  
 On définit la taille de la boîte à instancier <setSize(200,100);>  
 On indique le mode d'affichage des composants qui y seront déposés <setLayout(new FlowLayout( ));>



On ajoute la Label <add(etiq);> à la future boîte,

On ajoute le Button <add(bout1);>

La future boîte devra s'afficher à la fin de sa création <setVisible ( true );>



### Seconde classe :

Une classe principale servant à lancer l'application et contenant la méthode main :

#### La classe principale contenant main

```

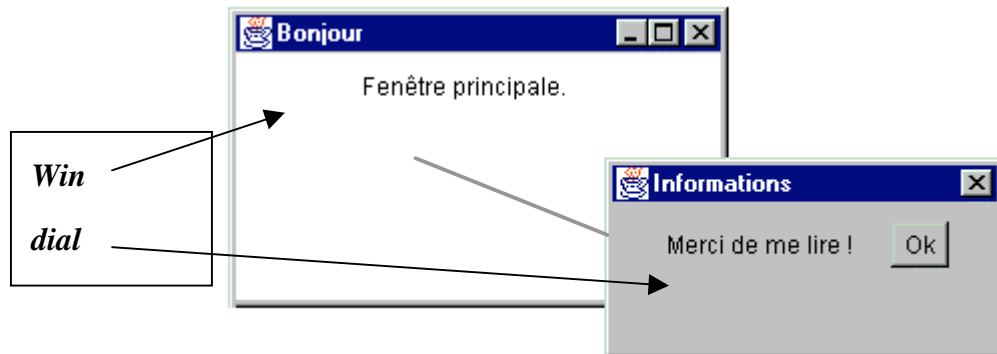
class AppliDialogue
{
    public static void main(String [] arg) {
        Frame win = new Frame("Bonjour");
        UnDialog dial = new UnDialog(win);
        Dialog dlg = new Dialog(win,"Information vide");
        dlg.setSize(150,70);
        dlg.setVisible ( true );
        win.setBounds(100,100,250,150);
        win.setLayout(new FlowLayout( ));
        win.add(new Label("Fenêtre principale."));
        win.setVisible ( true );
    }
}

```

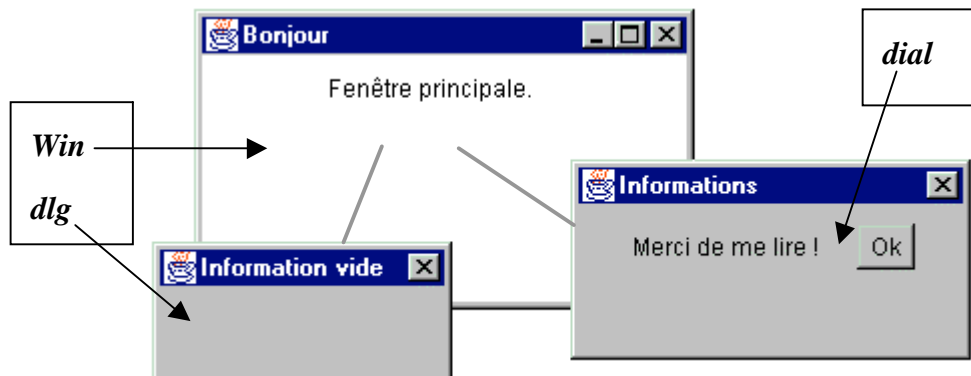
```
}
```

Toutes les instructions de la méthode main mise à part l'instruction `<UnDialog dial = new UnDialog(win);>`, correspondent à ce que nous avons écrit plus haut en vue de la création d'une fenêtre *win* de classe Frame dans laquelle nous ajoutons une Label et qui lance une boîte de dialogue *dlg* :

L'instruction `<UnDialog dial = new UnDialog(win);>` sert à instancier un objet *dial* de notre classe personnalisée, cet objet étant rattaché à la fenêtre *win* :



L'instruction `<Dialog dlg = new Dialog(win,"Information vide");>` sert à instancier un objet *dlg* de classe générale Dialog, cet objet est aussi rattaché à la fenêtre *win* :



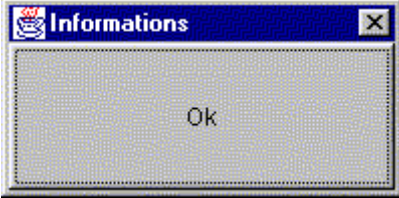
### Le programme Java avec les 2 classes

```
import java.awt.*;
class UnDialog extends Dialog {
    public UnDialog(Frame mere)
    {
        super(mere,"Informations");
        .....// instructions
        setVisible ( true );
    }
}
class AppliDialogue {
    public static void main(String [] arg) {
        Frame win = new Frame("Bonjour");
        .....// instructions
        win.setVisible ( true );
    }
}
```

```
}
```

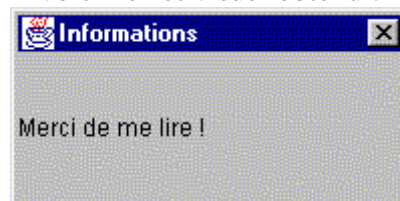
### Comment gérer la position d'un composant dans un conteneur de classe Container : Le Layout Manager

En reprenant la fenêtre de dialogue précédente, observons l'effet visuel produit par la présence ou non d'un **Layout Manager** :

La classe de dialogue sans Layout Manager	
<pre>import java.awt.*; class AppliUnDialog2 extends Dialog {     public AppliUnDialog2(Frame mere)     {         super(mere,"Informations");         Label etiq = new Label("Merci de me lire !");         Button bout1 = new Button("Ok");         setSize(200,100);         //setLayout(new FlowLayout( ));         add(etiq);         add(bout1);         setVisible ( true );     }     public static void main(String[] args) {         Frame fen = new Frame("Bonjour");         AppliUnDialog2 dlg = new AppliUnDialog2(fen);     } }</pre>	<p>Voici ce que donne l'exécution de ce programme Java.</p> <p>En fait lorsqu'aucun Layout manager n'est spécifié, c'est par défaut la classe du Layout &lt;BorderLayout&gt; qui est utilisée par Java. Cette classe n'affiche qu'un seul élément en une position fixée.</p>  <p>Nous remarquons que le bouton masque l'étiquette en prenant toute la place.</p>

Soit les instructions d'ajout des composants dans le constructeur <b>public</b> AppliUnDialog2(Frame mere)	<b>Intervertissons</b> l'ordre d'ajout du bouton et de l'étiquette, toujours en laissant Java utiliser le <BorderLayout> par défaut :
<pre>add(etiq); add(bout1); setVisible ( true );</pre>	<pre>add(bout1); add(etiq); setVisible ( true );</pre>

voici l'effet visuel obtenu :



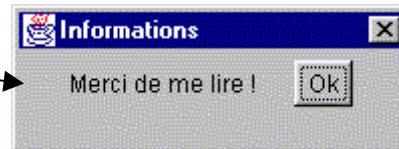
Cette fois c'est l'étiquette (ajoutée en dernier) qui masque le bouton !

Définissons un autre Layout puisque celui-ci ne nous plaît pas, utilisons la classe <FlowLayout> qui place les composants les uns à la suite des autres de la gauche vers la droite, l'affichage visuel continuant à la ligne suivante dès que la place est insuffisante. L'instruction <setLayout(new FlowLayout( ));>, assure l'utilisation du FlowLayout pour notre fenêtre de dialogue.

## La classe de dialogue avec FlowLayout

```
import java.awt.*;
class AppliUnDialog2 extends Dialog
{
    public AppliUnDialog2(Frame mere)
    {
        super(mere,"Informations");
        Label etiq = new Label("Merci de me lire !");
        Button bout1 = new Button("Ok");
        setSize(200,100);
        setLayout(new FlowLayout ());
        add(etiq);
        add(bout1);
        setVisible ( true );
    }
    public static void main(String[ ] args) {
        Frame fen = new Frame("Bonjour");
        AppliUnDialog2 dlg = new AppliUnDialog2(fen);
    }
}
```

voici l'effet visuel obtenu :



Si comme précédemment l'on échange l'ordre des instructions d'ajout du bouton et de l'étiquette :

```
setLayout(new FlowLayout ());
add(bout1);
add(etiq);
```

on obtient l'affichage inversé :



D'une manière générale, utilisez la méthode `< public void setLayout(LayoutManager mgr) >` pour indiquer quel genre de positionnement automatique (cf. aide du JDK pour toutes possibilités) vous conférez au Container (ici la fenêtre) votre façon de gérer le positionnement des composants de la fenêtre. Voici à titre d'information tirées du JDK, les différentes façons de positionner un composant dans un container.

### *héritant de LayoutManager :*

GridLayout, FlowLayout, ViewportLayout, ScrollPaneLayout,  
BasicOptionPaneUI.ButtonAreaLayout, BasicTabbedPaneUI.TabbedPaneLayout,  
BasicSplitPaneDivider.DividerLayout, BasicInternalFrameTitlePane.TitlePaneLayout,  
BasicScrollBarUI, BasicComboBoxUI.ComboBoxLayoutManager,  
BasicInternalFrameUI.InternalFrameLayout.

### *héritant de LayoutManager2 :*

CardLayout, GridBagLayout, BorderLayout, BoxLayout, JRootPane.RootLayout,  
OverlayLayout, BasicSplitPaneUI.BasicHorizontalLayoutManager.

Vous notez qu'il est impossible d'être exhaustif sans devenir assommant, à chacun d'utiliser les Layout en observant leurs effets visuels.

Il est enfin possible, si aucun des Layout ne vous convient de gérer personnellement au pixel près la position d'un composant. Il faut tout d'abord indiquer que vous ne voulez aucun Layoutmanager, puis ensuite préciser les coordonnées et la taille de votre composant.

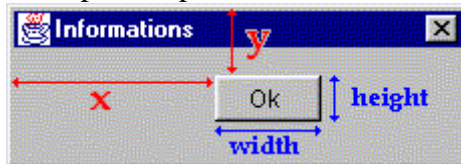
Indiquer qu'aucun Layout n'est utilisé :

```
setLayout(null); //on passe la référence null comme paramètre à la méthode de définition du Layout
```

Préciser les coordonnées et la taille du composant avec sa méthode setBounds :

```
public void setBounds(int x, int y, int width, int height)
```

Exemple, les paramètres de setBounds pour un Button :



Si nous voulons positionner nous mêmes un composant *Component comp* dans la fenêtre, nous utiliserons la méthode add indiquant le genre de façon de ranger ce composant (LayoutManager)

```
public void add(Component comp, Object constraints)
```

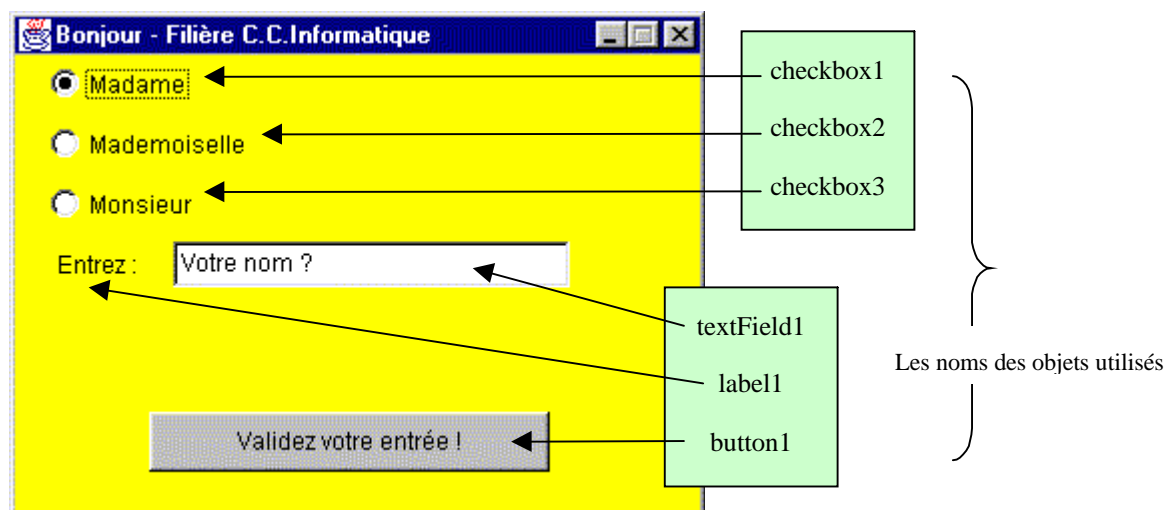
```
add(checkbox1, new FlowLayout( ));
```

ou bien

```
add(checkbox1, null);
```

## Une application fenêtrée pas à pas

Nous construisons une IHM de saisie de renseignements concernant un(e) étudiant(e) :



ci-après le code Java du programme :

```
class AppliIHM { // classe principale
    //Méthode principale
    public static void main(String[] args) { // lance le programme
        Cadre1 fenetre = new Cadre1(); // création d'un objet de classe Cadre1
        fenetre.setVisible(true); // cet objet de classe Cadre1 est rendu visible sur l'écran
    }
}

import java.awt.*; // utilisation des classes du package awt
class Cadre1 extends Frame { // la classe Cadre1 hérite de la classe des fenêtres Frame
    Button bouton1 = new Button(); // création d'un objet de classe Button
    Label label1 = new Label(); // création d'un objet de classe Label
    CheckboxGroup checkboxGroup1 = new CheckboxGroup(); // création d'un objet groupe de checkbox
    Checkbox checkbox1 = new Checkbox(); // création d'un objet de classe Checkbox
    Checkbox checkbox2 = new Checkbox(); // création d'un objet de classe Checkbox
    Checkbox checkbox3 = new Checkbox(); // création d'un objet de classe Checkbox
    TextField textField1 = new TextField(); // création d'un objet de classe TextField

    //Constructeur de la fenêtre
    public Cadre1() { //Constructeur sans paramètre
        Initialiser(); // Appel à une méthode privée de la classe
    }

    //Initialiser la fenêtre :
    private void Initialiser() { //Création et positionnement de tous les composants
        this.setResizable(false); // la fenêtre ne peut pas être retaillée par l'utilisateur
        this.setLayout(null); // pas de Layout, nous positionnons les composants nous-mêmes
        this.setBackground(Color.yellow); // couleur du fond de la fenêtre
        this.setSize(348, 253); // width et height de la fenêtre
        this.setTitle("Bonjour - Filère C.C.Informatique"); // titre de la fenêtre
        this.setForeground(Color.black); // couleur de premier plan de la fenêtre
        bouton1.setBounds(70, 200, 200, 30); // positionnement du bouton
        bouton1.setLabel("Validez votre entrée !"); // titre du bouton
        label1.setBounds(24, 115, 50, 23); // positionnement de l'étiquette
        label1.setText("Entrez :"); // titre de l'étiquette
        checkbox1.setBounds(20, 25, 88, 23); // positionnement du CheckBox
        checkbox1.setCheckboxGroup(checkboxGroup1); // ce CheckBox est mis dans le groupe checkboxGroup1
        checkbox1.setLabel("Madame"); // titre du CheckBox
        checkbox2.setBounds(20, 55, 108, 23); // positionnement du CheckBox
        checkbox2.setCheckboxGroup(checkboxGroup1); // ce CheckBox est mis dans le groupe checkboxGroup1
        checkbox2.setLabel("Mademoiselle"); // titre du CheckBox
        checkbox3.setBounds(20, 85, 88, 23); // positionnement du CheckBox
        checkbox3.setCheckboxGroup(checkboxGroup1); // ce CheckBox est mis dans le groupe checkboxGroup1
        checkbox3.setLabel("Monsieur"); // titre du CheckBox
        checkboxGroup1.setSelectedCheckbox(checkbox1); // le CheckBox1 du groupe est coché au départ
        textField1.setBackground(Color.white); // couleur du fond de l'éditeur mono ligne
        textField1.setBounds(82, 115, 198, 23); // positionnement de l'éditeur mono ligne
        textField1.setText("Votre nom ?"); // texte de départ de l'éditeur mono ligne
        this.add(checkbox1); // ajout dans la fenêtre du CheckBox
        this.add(checkbox2); // ajout dans la fenêtre du CheckBox
        this.add(checkbox3); // ajout dans la fenêtre du CheckBox
        this.add(bouton1); // ajout dans la fenêtre du bouton
        this.add(textField1); // ajout dans la fenêtre de l'éditeur mono ligne
        this.add(label1); // ajout dans la fenêtre de l'étiquette
    }
}
```