

REPUBLIQUE DE BENIN

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

UNIVERSITE D'ABOMEY-CALAVI

Institut de Formation et de Recherche en Informatique

Filières : Génie Logiciel – Internet et Multimédia

Licence II

Java Avancé

Elaboré par :

Ing. Quentin S. CHOUKPIN

Enseignant des Sciences Informatique et Technologique

E-mail : choukpinquentin92@gmail.com

Tél : (00229) 97965978

OBJECTIF GENERAL

Concevoir une application avec interface graphique avec le langage Java.

OBJECTIFS SPECIFIQUES

- Maîtriser les méthodes d'interfaces graphiques avec Java.
- Structurer un problème informatique avec les approches orientée objet
- Gérer les différents événements provenant d'une interface Java.
- Réaliser une connexion entre une application graphique Java et une base de données
- Réaliser un projet avec interface graphique en java.

MODE D'EVALUATION

- Evaluation écrite
- Travaux pratiques
- Recherches

PLAN DU COURS

Introduction générale

- 1. Interfaces graphiques Java**
- 2. Gestion des évènements**
- 3. Java et les Bases de données.**
- 4. Réalisation de projets concrets**

IFRI-UAC

Introduction Générale

Le langage Java est un langage de programmation informatique compilé et interprété. Créé par James Gosling et Patrick Naughton de Sun Microsystems, il apparaît comme un langage objet permettant le développement d'applications complètes et qui s'appuie sur les structures de données classiques (tableaux, fichiers) et utilise abondamment l'allocation dynamique de mémoire pour créer des objets en mémoire. Il est une plateforme, un environnement d'exécution. Le langage Java peut être structuré en deux parties :

- Le programme écrit
- La JVM (Java Virtuel Machine).

Le langage Java admet plusieurs avantages très importants. Ainsi, nous pouvons énumérer :

- Son caractère Orienté Objet.
- Sa capacité multiplateformes (portabilité)
- Sa sécurité aussi bien pendant le développement que pendant l'utilisation.
- La disponibilité des Bibliothèques de classes indépendantes de la plate-forme, ce qui est le point essentiel de la programmation sur internet ou plusieurs machines dissemblables sont interconnectées.
- Son Interactivité sur le Web facilitant l'insertion dans des pages Web, au milieu d'images et de textes, de programmes interactifs appelés « applets »
- Sa capacité à générer des applications qui soient indépendantes des machines et de leur système d'exploitation

Ainsi, la question qui se pose est de savoir ce qu'on peut réaliser avec JAVA ? Avec la robustesse du langage Java, nous pouvons nous en servir pour plusieurs réalisations parmi lesquelles nous pouvons retenir :

- Le développement, des applications Desktop,
- Le développement des applets pour vos sites web,
- Le développement des sites en JSP, des applications pour téléphone mobile.

A travers ce module, nous aborderons les notions de la conception graphique en Java, ensuite la connexion d'une interface graphique java aux bases de données d'une part puis la gestion des événements pour enfin terminer par la réalisation des applications utiles avec le langage java.

1. Interface graphique Java

1.1. Définition Conceptuelle

Une interface graphique ou GUI (Graphical User Interface) est une couche applicative destinée à fournir aux utilisateurs d'ordinateurs un moyen attractif et facile pour interagir avec un programme. Il s'agit de l'interface d'un programme qui profite des capacités graphiques d'un ordinateur pour le rendre plus facile d'utilisation. Des interfaces graphiques bien conçues évitent à l'utilisateur d'avoir à apprendre de complexes langages de commandes. Un ensemble d'écrans de présentations qui utilisent des éléments graphiques (tels boutons et icônes) pour rendre un programme plus facile d'utilisation. Ainsi, une même application peut offrir différentes interfaces à l'utilisateur :

- Une interface graphique stricto sensu qui s'exécute sur son ordinateur
- Une interface web qui s'exécute dans un navigateur Internet
- Une interface WAP qui s'exécute sur un téléphone
- Une interface « terminal » qui s'exécute sur des appareils spécifiques
- Des « services web » qui offrent ses fonctionnalités à d'autres applications
- Notons que l'interface peut être intégrée à l'application (elle forme un tout avec l'application) (architecture « 2-tiers »).

Voici une synoptique de ce principe :

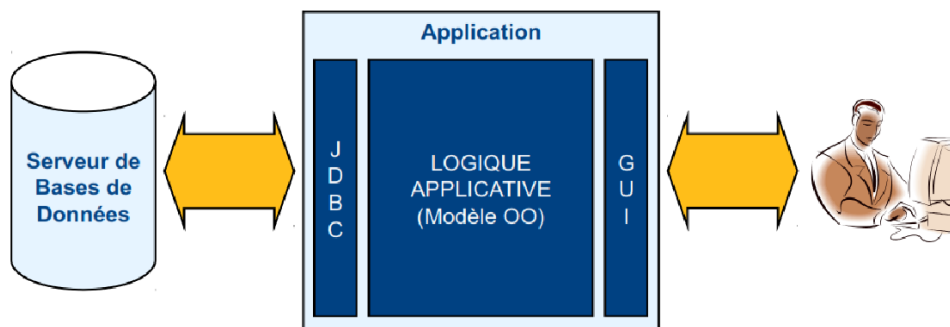


Figure 1 : Principe de l'interface Graphique

Elle peut être indépendante de l'application (différentes interfaces indépendantes utilisent la même application sous-jacente) et ne sont que la « partie visible » de l'application (architecture « 3-tiers »).

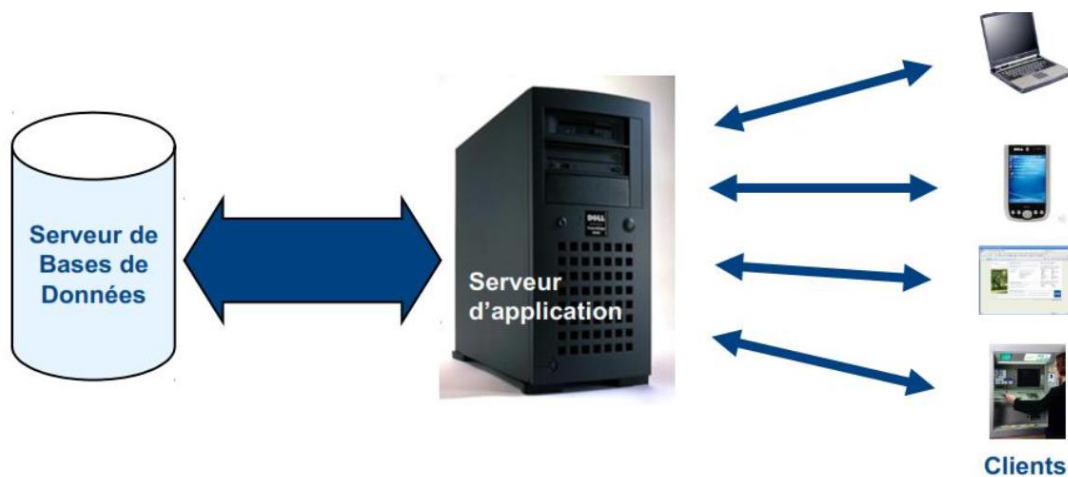


Figure 2: Architecture 3-Tiers

1.2. Principe des interfaces Graphiques

Les éléments graphiques de l'interface Java sont définis dans le paquetage (package) AWT (Abstract Window Toolkit). AWT est une librairie de classes graphiques portables s'exécutant sans recompilation sur tout ordinateur disposant d'un interpréteur de code Java (machine virtuelle Java). Cependant, la machine virtuelle s'appuie en partie sur les ressources graphiques du système d'exploitation. Ainsi, L'apparence des fenêtres et boutons diffère d'un système d'exploitation à l'autre avec AWT. En effet, il existe une autre librairie d'interfaces graphiques appelée Swing entièrement autonome, qui ne dépend pas du système d'exploitation, mais qui peut prendre l'aspect de tel ou tel système d'exploitation à la demande.

Il faut noter avant toute chose que les interfaces graphiques structurent leur constitution en deux éléments majeurs :

- **Les containers (contenants)** : qui sont eux-mêmes des composants mais sont destinés à accueillir des composants. Ils gèrent l'affichage des composants.
- **Les Components (Composants)** : constituent les différents éléments de l'affichage (boutons, barres de menus, etc.)

1.3. Interface graphique avec AWT

AWT (Abstract Window Toolkit) est un package java inclu dans toutes les versions de Java. Ce package est la base de la programmation des interfaces graphique. Il représente le seul package fonctionnant sur toutes

les générations de navigateurs. Les classes contenues dans AWT dérivent (héritent) toutes de la classe `Component`. Nous présenterons ici quelques classes pour construire une interface utilisateur standard.

1.3.1. Classes Container AWT

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. Voici à travers les représentations ci-dessous, l'organisation et la hiérarchie de ces classes :

Hiérarchie de la classe Container :

```

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

❖ Les principales classes conteneurs

| Classe | Fonction |
|---|---|
| <pre> +--java.awt.Container +--java.awt.Window </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 +--java.awt.Panel </pre> | Crée une surface sans bordure, capable de contenir d'autres éléments : boutons, panel etc... |
| <pre> +--java.awt.Container +--java.awt.ScrollPane </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 +--java.awt.Frame </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 +--java.awt.Dialog </pre> | Crée une fenêtre de dialogue avec l'utilisateur, avec une bordure, un titre et un bouton-icône de fermeture. |

Une première fenêtre peut être construite à partir d'un objet de classe « Frame » ; donc une fenêtre est donc un objet, et l'on pourra donc créer autant de fenêtres que nécessaire. Il suffira à chaque fois d'instancier un objet de la classe Frame.

Ainsi, nous pouvons aborder quelques méthodes de la classe Frame :

❖ Quelques méthodes de la classe Frame, utiles au départ

| Méthodes | Fonction |
|--|---|
| public void setSize(int width, int height) | retaille la largeur (width) et la hauteur (height) de la fenêtre. |
| public void setBounds(int x, int y, int width, int height) | retaille la largeur (width) et la hauteur (height) de la fenêtre et la positionne en x,y sur l'écran. |
| public Frame(String title) public Frame() | Les deux constructeurs d'objets Frame, celui qui possède un paramètre String écrit la chaîne dans la barre de titre de la fenêtre. |
| public void setVisible(boolean b) | Change l'état de la fenêtre en mode visible ou invisible selon la valeur de b. |
| public void hide() | Change l'état de la fenêtre en mode invisible . |
| Différentes surcharges de la méthode add : public Component add(Component comp) etc... | Permettent d'ajouter un composant à l'intérieur de la fenêtre. |

Cependant, 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 :

▪ **TP1 :**

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

- **TP2:** 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> |

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. 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 et « 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.

- **TP3 :** 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> |

1.3.2. 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.

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

- **TP4** : 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 );
    }
}
```

- **TP5** : introduisons 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 );
    }
}
```

1.3.3. Composants d'affichage et de saisie

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...

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

- **TP6** : Soit à afficher une fenêtre principale contenant le texte "fenêtre principal" 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

| 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 );
}
}
```

- 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 );
    }
}
```

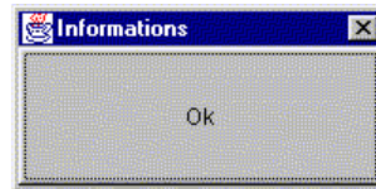
Autre exemple

La classe de dialogue sans Layout Manager

```
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);
    }
}
```

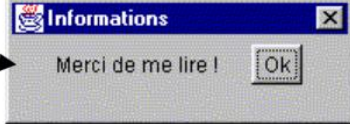
Voici ce que donne l'exécution de ce programme Java.

En fait lorsqu'aucun Layout manager n'est spécifié, c'est par défaut la classe du Layout <BorderLayout> qui est utilisée par Java. Cette classe n'affiche qu'un seul élément en une position fixée.

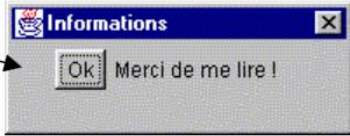


Nous remarquons que le bouton masque l'étiquette en prenant toute la place.

Autre exemple

| La classe de dialogue avec FlowLayout | |
|--|---|
| <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 l'effet visuel obtenu :</p>  |

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

| | |
|---|---|
| <pre>setLayout(new FlowLayout()); add(bout1); add(etiq);</pre> | <p>on obtient l'affichage inversé :</p>  |
|---|---|

Remarque : D'une manière générale, utilisez la méthode `< public void setLayout(LayoutManager mgr) >` pour indiquer quel genre de positionnement automatique 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

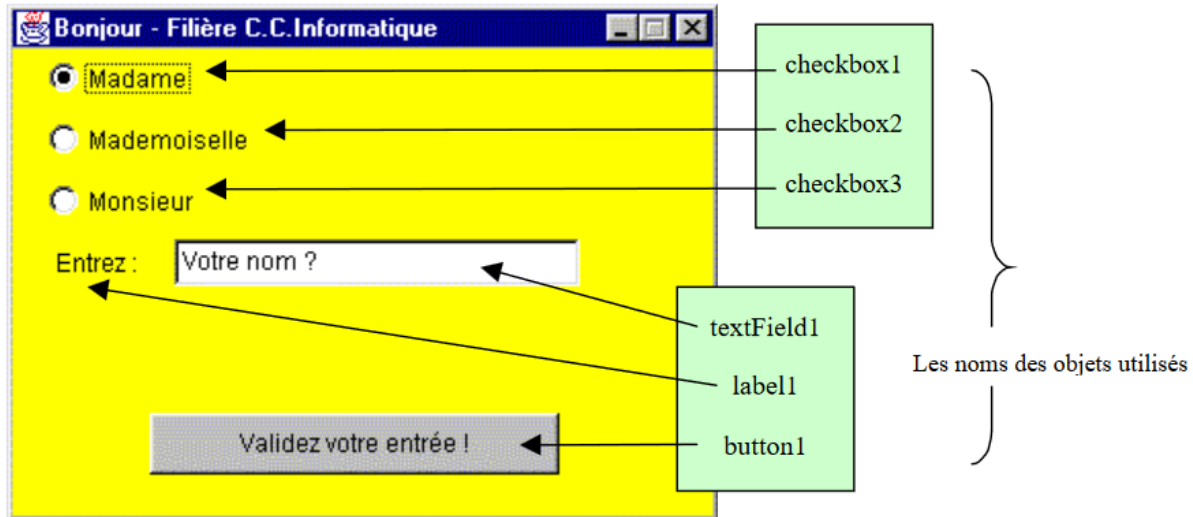
héritant de *LayoutManager* :

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.

- **TP7 :** Une fenêtre d'application



#code

```
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 button1 = 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 - Filière C.C. Informatique"); // titre de la fenêtre
        this.setForeground(Color.black); // couleur de premier plan de la fenêtre
        button1.setBounds(70, 200, 200, 30); // positionnement du bouton
        button1.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(button1); // 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
}
```

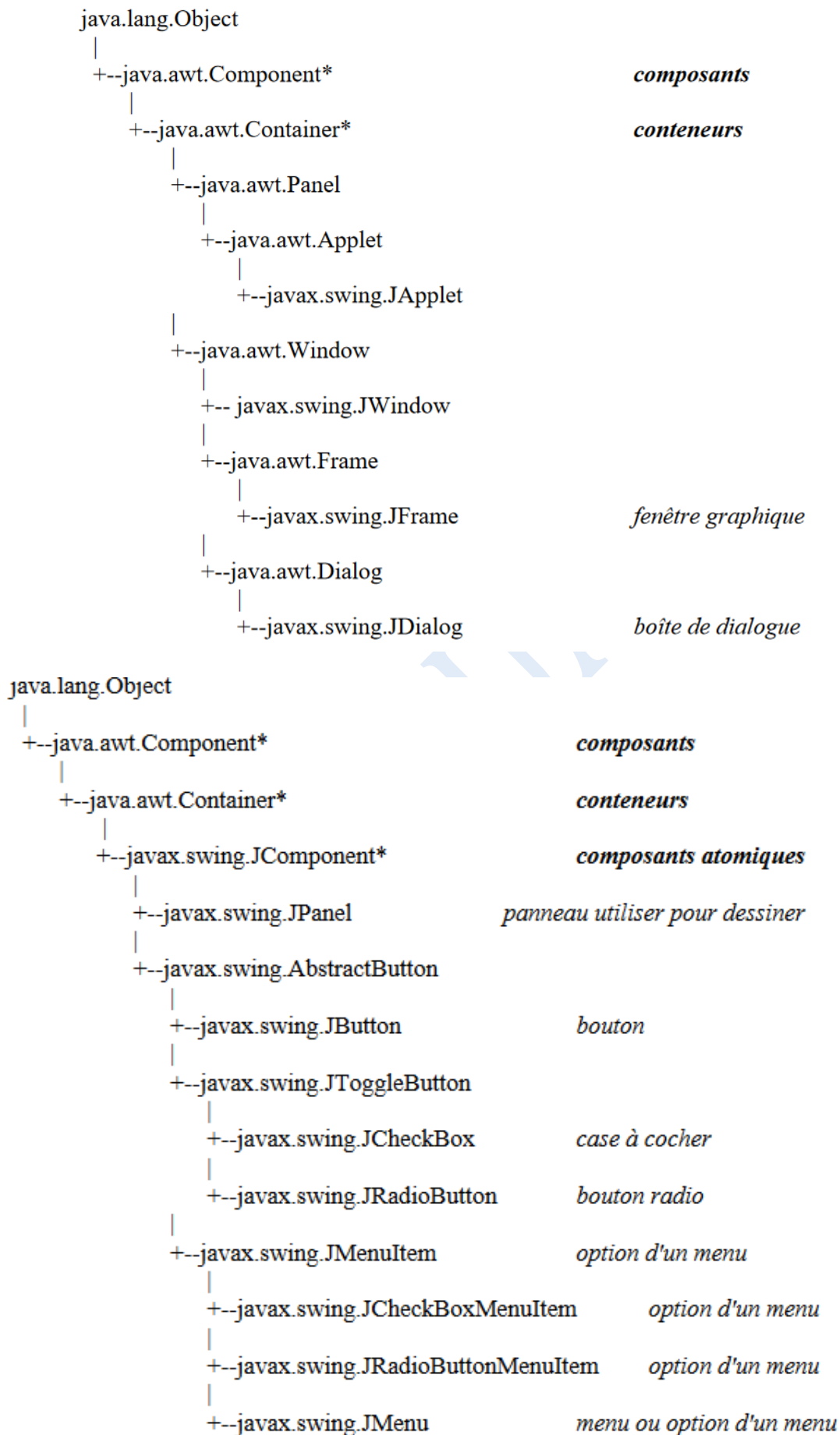
NB : Pour plus de détails sur les autres classes et méthodes de la librairie AWT, Veuillez consulter la documentation de Java ou le document « **la programmation objet en java** » mis à votre disposition de la page 152 à 256.

1.4. Interface graphique avec Swing

La librairie Swing étant totalement gérée par la machine virtuelle, elle permet de ne plus se limiter au sous-ensemble de composants graphiques communs à toutes les plates-formes. Swing définit de nouveaux composants. Cependant, la bibliothèque Swing nécessite plus de temps CPU, car la machine virtuelle doit dessiner les moindres détails des composants. Cette indépendance vis-à-vis du système d'exploitation permet même de décider, sur n'importe quelle machine, de l'apparence (le look and feel) que l'on veut donner à ses composants (style Linux, style Windows, etc.). Certains composants peuvent être complexes à mettre en œuvre si on veut les exploiter dans tous leurs détails (les listes, les arbres ou les tables par exemple). Swing est fourni en standard avec le JDK (Java Development Kit).

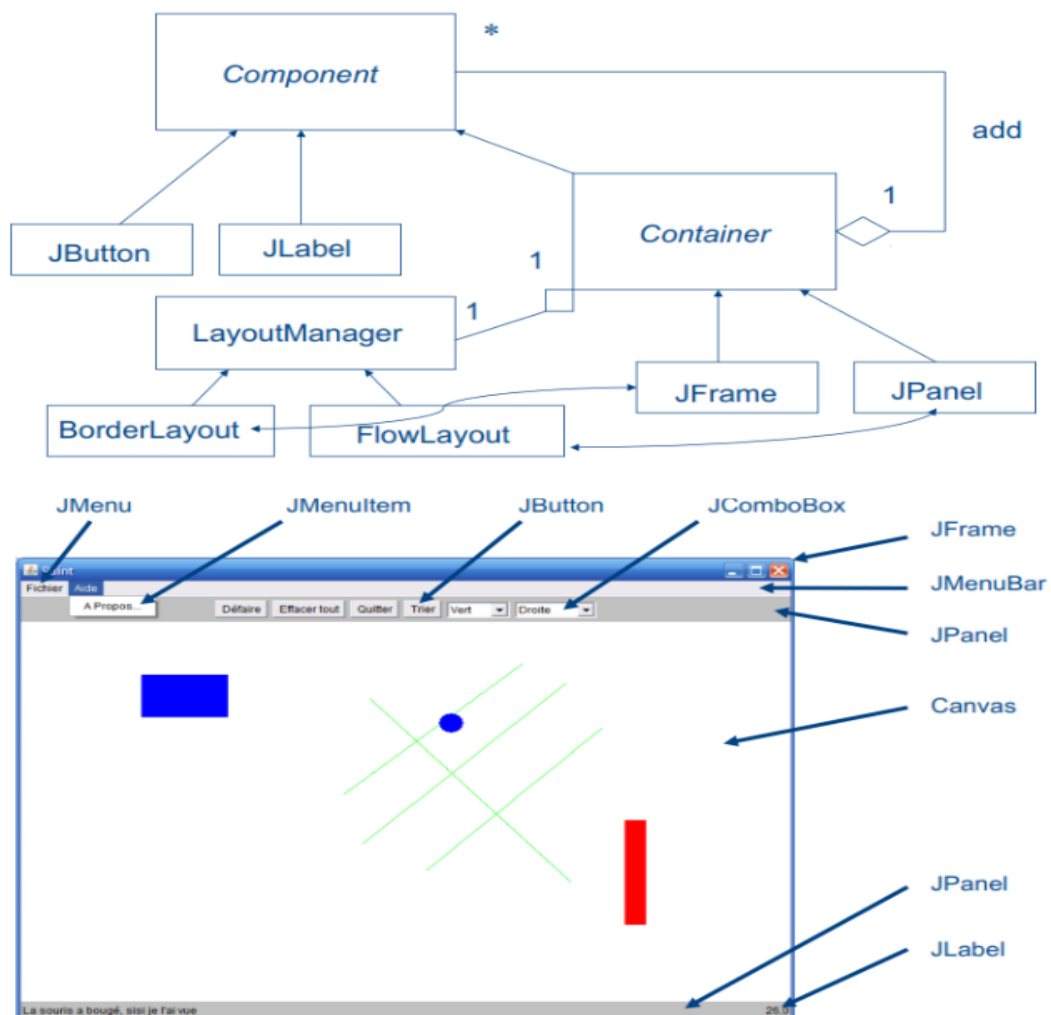
1.4.1. Comparative AWT vs SWING

Sur le plan organisationnel, nous pouvons afficher les points de ressemblance ou de dissemblance présentés aux figures ci-dessous. On retiendra de ces comparatives que les classes et méthodes de la librairie AWT semble être pour la plupart des objets lourds ; tant dis que, seules quelques classes de la librairie SWING paraissent lourdes. Il est donc généralement conseiller de miser plus sur les composants SWING.



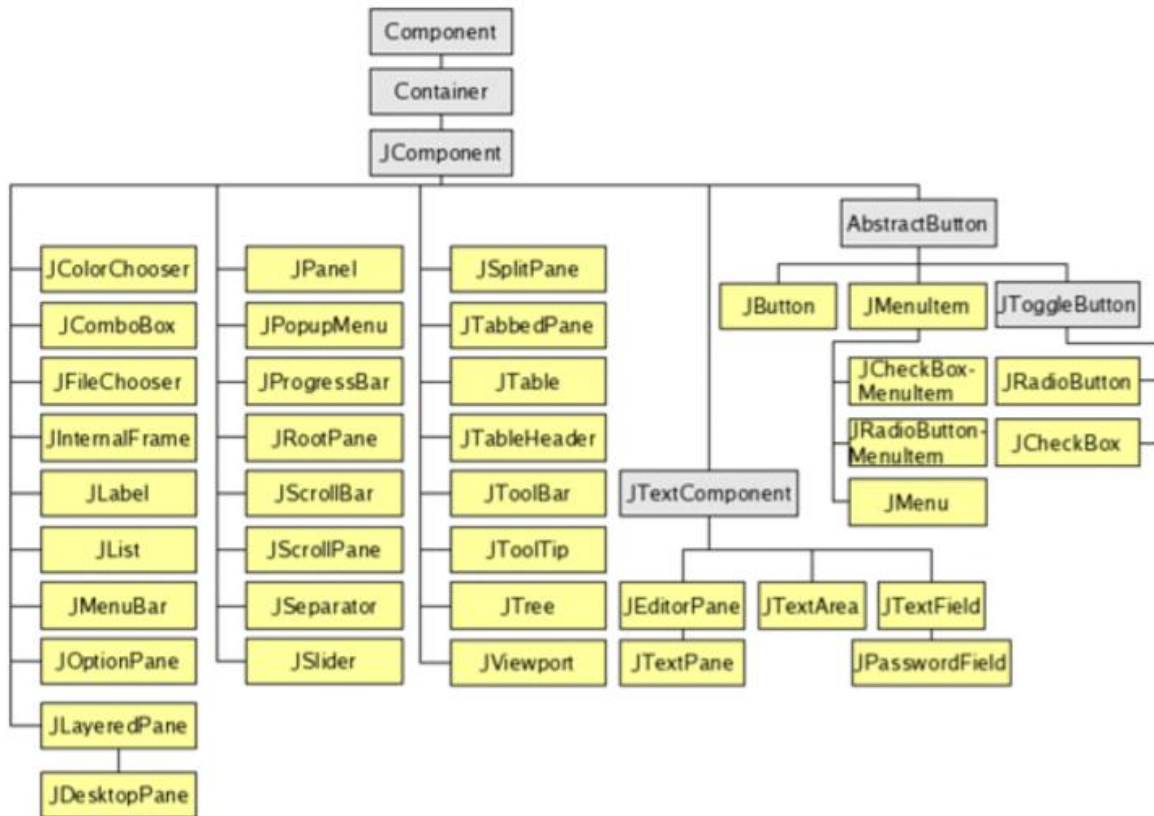
| | |
|------------------------------------|-------------------------|
| +--javax.swing.JLabel | étiquette |
| +--javax.swing.text.JTextComponent | |
| +--javax.swing.JTextField | champ de texte |
| +--javax.swing.text.JList | boîte de liste |
| +--javax.swing.text.JComboBox | boîte de liste combinée |
| +--javax.swing.JScrollPane | panneau de défilement |
| +--javax.swing.JMenuBar | barre de menu |
| +--javax.swing.JPopupMenu | menu surgissant |
| +--javax.swing.JToolBar | barre d'outils |

Une illustration de ces éléments est présentée par la figure suivante :



1.4.2. Quelques éléments de la librairie SWING

Suivant l'organisation ci-après :



nous aurons :

❖ Exemples de composants

- **JTextArea** (zone de texte)
 - Contient une chaîne de caractères
- **JTable** (table de données)
 - Contient deux Arrays ([]):
 - Les données (tableau multidimensionnel d'objets)
 - Les intitulés des colonnes (vecteur de chaînes de caractères)
 - Gère automatiquement l'affichage des données en colonnes
 - Permet le déplacement de colonnes
 - Permet ou interdit la modification des données de la table
- **JButton** (bouton)
 - Contient un intitulé
- **JCheckBox** (case à cocher)
 - Contient un intitulé
 - Peut être sélectionnée ou non
 - Peut être activée ou désactivée (on peut ou non la cocher)
- **JLabel** (étiquette/intitulé)
 - Contient un intitulé
- **JComboBox** (liste déroulante)
 - Contient une liste d'éléments sous forme de chaînes de caractères
 - Contient un entier représentant le numéro de la ligne sélectionnée

❖ Exemples de conteneurs

- JFrame
 - Composant du plus haut niveau
 - La fenêtre d'une application est une instance de cette classe
 - Le Frame contient les différents composants graphiques de l'application
 - Ne peut être intégré dans un autre conteneur
 - Peut contenir une barre de menu (JMenuBar)
- JPanel
 - Peut être lui-même intégré dans un autre conteneur (par ex. un JFrame)
 - Peut contenir lui-même autant de composants que nécessaire
 - Plusieurs moyens d'y disposer les composants
- JScrollPane
 - Peut être lui-même intégré dans un autre conteneur (par ex. un JFrame)
 - Offre une « vue » sur un seul composant ou conteneur (JTable, JPanel, etc.)
 - Permet de changer le contenu en cours de route
 - Offre automatiquement des barres de défilement si nécessaire

❖ Barres et éléments de menus

- JMenuBar
 - Une barre de menu (une seule par JFrame)
 - Contient différents menus (JMenu)
- JMenu
 - Représente un menu dans une barre de menu
 - Contient différents éléments (JMenuItem)
 - Contient un intitulé
- JMenuItem
 - Représente un élément de menu
 - Contient un intitulé
 - Peut être associé à des raccourcis clavier

❖ Constructeurs de composants

- JTextArea (zone de texte)
 - `new JTextArea(String texteAAfficher);`
- JTable (table de données)
 - `new JTable(Object[][] donnees, String[] intitules);`
 - ➔ Il faut un tableau de chaînes de caractères contenant les intitulés des colonnes
 - ➔ Et une matrice d'objets dont chaque rangée est une ligne du tableau
- JButton (bouton)
 - `new JButton(String intitulé);`
- JCheckBox (case à cocher)
 - `new JCheckBox(String intitulé);`
- JLabel (étiquette/intitulé)
 - `new JLabel(String intitulé);`
- JComboBox (liste déroulante)
 - `new JComboBox();`

❖ Constructeurs de containers et menus

- JFrame
 - new JFrame();
- JPanel
 - new JPanel();
- JScrollPane
 - new JScrollPane();
- JMenuBar
 - new JMenuBar();
- JMenu
 - new JMenu(String intitulé);
- JMenuItem
 - new JMenuItem(String intitulé);

❖ Quelques méthodes utiles

- JTextArea (zone de texte)
 - setText(String) → Modifier le contenu
- JTable (table de données)
 - setSelectionMode(int) → Définir le mode de sélection (ex: une ligne à la fois)
 - getSelectionModel() → Récupérer le modèle de sélection défini
- JButton (bouton)
 - setText(String) → Modifier l'intitulé
- JCheckBox (case à cocher)
 - setText(String) → Modifier l'intitulé
 - setEnabled(boolean) → Déterminer si la case est activée ou pas (peut être cochée ou non)
 - setSelected(boolean) → Déterminer si la case est cochée ou pas
 - isEnabled() → Renvoie true si la case est activée et false si elle ne l'est pas
 - isSelected() → Renvoie true si la case est cochée et false si elle ne l'est pas
- JLabel (étiquette/intitulé)
 - setText(String) → Modifie l'intitulé
- JComboBox (liste déroulante)
 - addItem(String) → Ajouter un élément (chaîne de caractères) dans la liste
 - getSelectedIndex() → Renvoie le numéro de la ligne sélectionnée

1.4.3. Quelques applications de base avec SWING

- TP8 : Création d'une fenêtre avec un bouton

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

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

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

Une fenêtre est un conteneur destiné à contenir d'autres composants. La méthode getContentPane de la classe JFrame fournit une référence, de type Container, au contenu de la fenêtre.

- TP 9 : Case à cocher

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

La méthode `setSelected` de la classe `AbstractButton` permet de modifier l'état d'une case à cocher.

- TP 10 : les boutons radio **JRadioButton**

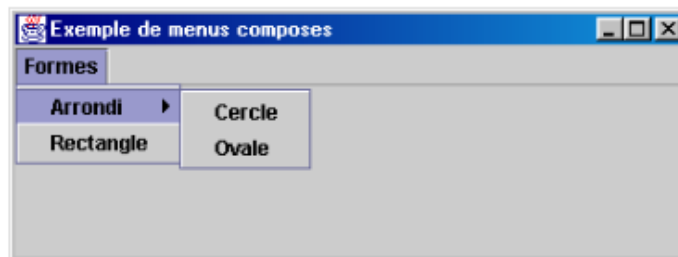
```
class MaFenetre extends JFrame {
    private JRadioButton bRouge;
    private JRadioButton bVert;
    public MaFenetre () {
        super("Une fenetre avec des boutons radio");
        setBounds(10,40,300,200);
        bRouge = new JRadioButton("Rouge");
        //création d'un bouton radio de référence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

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

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

■ TP 11 : Menu avec option

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



■ TP 12 : Bulles d'aides

Exemple

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



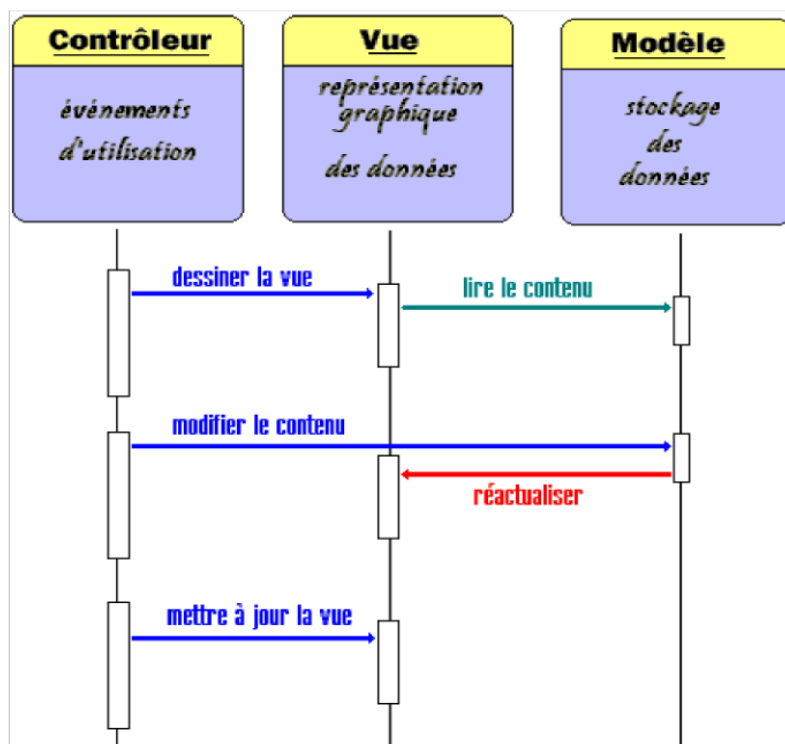
Remarque : pour plus d'applications, consultez les documents mis en compléments.

1.4.4. Le concept MVC avec SWING

En technologie de conception orientée objet il est conseillé de ne pas confier trop d'actions à un seul objet, mais plutôt de répartir les différentes responsabilités d'actions entre plusieurs objets. Par exemple pour un composant visuel (bouton, liste etc...) vous déléguez la gestion du style du composant à une classe (ce qui permettra de changer facilement le style du composant sans intervenir sur le composant lui-même), vous stockez les données contenues dans le composant dans une autre classe chargée de la gestion des données de contenu (ce qui permet d'avoir une gestion décentralisée des données) .

Si l'on recense les caractéristiques communes aux composants visuels servant aux IHM (interfaces utilisateurs), on retrouve 3 constantes générales pour un composant :

- son contenu (les données internes, les données stockées, etc...)
- son apparence (style, couleur, taille, etc...)
- son comportement (essentiellement en réaction à des événements)



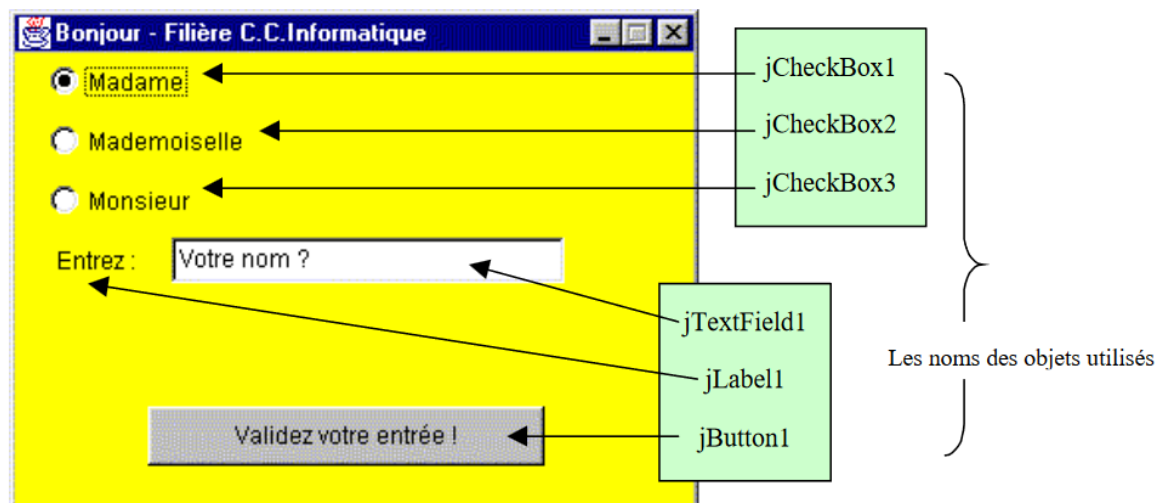
Le schéma précédent représente l'architecture Modèle-Vue-Contrôleur (ou design pattern observateur-observé) qui réalise cette conception décentralisée à l'aide de 3 classes associées à chaque composant :

- Le **modèle** qui stocke le contenu, qui contient des méthodes permettant de modifier le contenu et qui n'est pas visuel.
- La **vue** qui affiche le contenu, est chargée de dessiner sur l'écran la forme que prendront les données stockées dans le modèle.
- Le **contrôleur** qui gère les interactions avec l'utilisateur .

Avec la librairie SWING :

- Le **Modèle**, chargé de stocker les données, qui permet à la vue de lire son contenu et informe la vue d'éventuelles modifications est bien *représenté par une classe*.
- La **Vue**, permettant une représentation des données (nous pouvons l'assimiler ici à la représentation graphique du composant) peut être *répartie sur plusieurs classes*.
- Le **Contrôleur**, chargé de gérer les interactions de l'utilisateur et de propager des modifications vers la vue et le modèle peut aussi être réparti sur plusieurs classes, voir même dans des *classes communes à la vue et au contrôleur*.

- TP 13 : Réalisez l'interface suivante avec SWING



#Codes

```
import java.awt.*; // utilisation des classes du package awt
import java.awt.event.*; // utilisation des classes du package awt
import java.io.*; // utilisation des classes du package io
import javax.swing.*; // utilisation des classes du package swing
import java.util.*; // utilisation des classes du package util pour Enumeration
import javax.swing.text.*; // utilisation pour Document
import javax.swing.event.*; // utilisation pour DocumentEvent

class ficheSaisie extends JFrame { // la classe ficheSaisie hérite de la classe des fenêtres JFrame
    JButton jButton1 = new JButton( ); // création d'un objet de classe JButton
    JLabel jLabel1 = new JLabel( ); // création d'un objet de classe JLabel
    ButtonGroup Group1 = new ButtonGroup( ); // création d'un objet groupe pour AbstractButton et dérivées
```

```

JCheckBox jcheckbox1 = new JCheckBox(); // création d'un objet de classe JCheckBox
JCheckBox jcheckbox2 = new JCheckBox(); // création d'un objet de classe JCheckBox
JCheckBox jcheckbox3 = new JCheckBox(); // création d'un objet de classe JCheckBox
JTextField jTextField1 = new JTextField(); // création d'un objet de classe JTextField
Container ContentPane;

private String EtatCivil; //champ = le label du checkbox coché
private FileWriter fluxwrite; //flux en écriture (fichier texte)
private BufferedWriter fluxout; //tampon pour lignes du fichier

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

//indique quel JCheckBox est coché(réf) ou si aucun n'est coché(null) :
private JCheckBox getSelectedJCheckbox(){
    JCheckBox isSelect=null;
    Enumeration checkBenum = Group1.getElements();
    for (int i = 0; i < Group1.getButtonCount(); i++) {
        JCheckBox B =(JCheckBox) checkBenum.nextElement();
        if(B.isSelected())
            isSelect = B;
    }
    return isSelect;
}

//Active ou désactive le bouton pour sauvegarde :
private void AutoriserSave(){
    if (jtextField1.getText().length() !=0 && this.getSelectedJCheckbox()!=null)
        jbutton1.setEnabled(true);
    else
        jbutton1.setEnabled(false);
}

//rempli le champ Etatcivil selon le checkBox coché :
private void StoreEtatcivil(){
    this.AutoriserSave();
    if (this.getSelectedJCheckbox()!=null)
        this.EtatCivil=this.getSelectedJCheckbox().getLabel();
    else
        this.EtatCivil="";
}

//sauvegarde les infos étudiants dans le fichier :
public void ecrireEnreg(String record) {
    try {
        fluxout.write(record); //écrit les infos
        fluxout.newLine(); //écrit le eoln
    }
    catch (IOException err) {
        System.out.println( "Erreur : " + err );
    }
}

//Initialiser la fenêtre :
private void Initialiser() { //Création et positionnement de tous les composants
    ContentPane = this.getContentPane(); //Référencement du fond de dépôt des composants
    this.setResizable(false); // la fenêtre ne peut pas être retaillée par l'utilisateur
    ContentPane.setLayout(null); // pas de Layout, nous positionnons les composants nous-mêmes
    ContentPane.setBackground(Color.yellow); // couleur du fond de la fenêtre
    this.setSize(348, 253); // width et height de la fenêtre
    this.setTitle("Boniour - Filière C.C.Informatique"); // titre de la fenêtre

```

```

this.setForeground(Color.black); // couleur de premier plan de la fenêtre
jbutton1.setBounds(70, 160, 200, 30); // positionnement du bouton
jbutton1.setText("Validez votre entrée !"); // titre du bouton
jbutton1.setEnabled(false); // bouton désactivé
jlabel1.setBounds(24, 115, 50, 23); // positionnement de l'étiquette
jlabel1.setText("Entrez :"); // titre de l'étiquette
jcheckbox1.setBounds(20, 25, 88, 23); // positionnement du JCheckBox
Group1.add(jcheckbox1); // ce JCheckBox est mis dans le groupe Group1
jcheckbox1.setText("Madame"); // titre du JCheckBox
jcheckbox2.setBounds(20, 55, 108, 23); // positionnement du JCheckBox
Group1.add(jcheckbox2); // ce JCheckBox est mis dans le groupe Group1
jcheckbox2.setText("Mademoiselle"); // titre du JCheckBox
jcheckbox3.setBounds(20, 85, 88, 23); // positionnement du JCheckBox
Group1.add(jcheckbox3); // ce JCheckBox est mis dans le groupe Group1
jcheckbox3.setText("Monsieur"); // titre du JCheckBox
jcheckbox1.setBackground(Color.yellow); // couleur du fond du jcheckbox1
jcheckbox2.setBackground(Color.yellow); // couleur du fond du jcheckbox2
jcheckbox3.setBackground(Color.yellow); // couleur du fond du jcheckbox3
jtextField1.setBackground(Color.white); // couleur du fond de l'éditeur mono ligne
jtextField1.setBounds(82, 115, 198, 23); // positionnement de l'éditeur mono ligne
jtextField1.setText("Votre nom ?"); // texte de départ de l'éditeur mono ligne
ContentPane.add(jcheckbox1); // ajout dans la fenêtre du JCheckBox
ContentPane.add(jcheckbox2); // ajout dans le ContentPane de la fenêtre du JCheckBox
ContentPane.add(jcheckbox3); // ajout dans le ContentPane de la fenêtre du JCheckBox
ContentPane.add(jbutton1); // ajout dans le ContentPane de la fenêtre du bouton
ContentPane.add(jtextField1); // ajout dans le ContentPane de la fenêtre de l'éditeur mono ligne
ContentPane.add(jlabel1); // ajout dans le ContentPane de la fenêtre de l'étiquette
EtatCivil = ""; // pas encore de valeur
Document doc = jTextField1.getDocument(); // partie Modele MVC du JTextField
try {
    fluxwrite = new FileWriter("etudiants.txt", true); // création du fichier (en mode ajout)
    fluxout = new BufferedWriter(fluxwrite); // tampon de ligne associé
}
catch (IOException err) { System.out.println( "Problème dans l'ouverture du fichier "); }
//--> événements et écouteurs :
this.addWindowListener(
    new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            try {
                fluxout.close(); // le fichier est fermé et le tampon vidé
            }
            catch (IOException err) { System.out.println( "Impossible de fermer le fichier "); }
            System.exit(100);
        }
    });
doc.addDocumentListener( // on crée un écouteur anonyme pour le Modele (qui fait aussi le controle)
    new javax.swing.event.DocumentListener() {
        //-- l'interface DocumentListener a 3 méthodes qu'il faut implémenter :
        public void changedUpdate(DocumentEvent e) {}

        public void removeUpdate(DocumentEvent e) { // une suppression est un changement de texte
            textValueChanged(e); // appel au gestionnaire de changement de texte
        }

        public void insertUpdate(DocumentEvent e) { // une insertion est un changement de texte
            textValueChanged(e); // appel au gestionnaire de changement de texte
        }
    });
jcheckbox1.addItemListener(
    new ItemListener() {

```

```
        public void itemStateChanged(ItemEvent e){
            StoreEtatcivil( );
        }
    });
    jcheckbox2.addItemListener(
        new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                StoreEtatcivil();
            }
        });
    jcheckbox3.addItemListener(
        new ItemListener(){
            public void itemStateChanged(ItemEvent e){
                StoreEtatcivil();
            }
        });
    jButton1.addActionListener(
        new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ecrireEnreg(EtatCivil+"."+jtextField1.getText());
            }
        });
}
//Gestionnaire du changement de texte dans un document
private void textValueChanged(DocumentEvent e) {
    AutoriserSave();
}
}
```

IFRI-UNIVERSITY

2. Gestion des événements

2.1. Notion d'évènement

Un événement se produit. Sa source dans laquelle il se produit génère un objet de type événement. La source transmet l'événement à son (ses) écouteur(s). L'écouteur appelle la méthode correspondant au type d'événement et lui passe en argument l'objet événement. La méthode en question spécifie les traitements à réaliser lorsqu'un événement du type correspondant se produit. Dans ses traitements, la méthode peut examiner les caractéristiques de l'événement. (Position du curseur de la souris, code de la touche pressée au clavier...) et adapter son comportement en fonction de ces caractéristiques. Ainsi, nous avons les classifications suivantes :

- ❖ **Evénements** : `ActionEvent`, `AdjustmentEvent`, `ComponentEvent`, `ContainerEvent`, `FocusEvent`, `ItemEvent`, `KeyEvent`, `MouseEvent`, `TextEvent`, `WindowEvent`
- ❖ **Les Interfaces Ecouteurs** : `ActionListener`, `AdjustmentListener`, `ComponentListener`, `ContainerListener`, `FocusListener`, `ItemListener`, `KeyListener`, `MouseListener`, `MouseMotionListener`, `WindowListener`
- ❖ **Les Adapteurs correspondants** : `ActionAdapter`, `WindowAdapter`, `KeyAdapter`, `MouseAdapter`, etc.
- ❖ **Les Sources d'événements** : `Button`, `List`, `MenuItem`, `TextField`, `ScrollBar`, `CheckBox`, `Component`, `Container`, `Window`

2.2. Caractérisation des événements

L'utilisateur va intervenir sur le programme via le clavier ou la souris. Le programme devra associer des traitements aux actions possibles de l'utilisateur.

On distingue deux types d'événements :

- **Evénements de bas niveau** : appuyer ou relâcher un bouton de souris.
- **Evénements logiques** : clic sur une souris. Ainsi, si vous appuyez sur la lettre A, vous produisez les événements suivants :

- 4 événements de bas niveau :
 - Appuie sur la touche shift
 - Appuie sur la touche A
 - Relâchement de la touche A
 - Relâchement de la touche shift

- 1 événement logique :
Frappe du caractère A

Les événements sont représentés par des instances de sous-classes de `java.util.EventObject`. On peut alors classer les événements comme suit :

- Événements de bas niveau : `KeyEvent` (action sur une touche), `MouseEvent` (action sur la souris)
- Événements de haut niveau : `FocusEvent` (une fenêtre qui prend le focus ou la main), `WindowEvent` (fenêtre fermée, ouverte, icônifiée), `ActionEvent` (une action est déclenchée), `ItemEvent` (choisir un Item dans une liste), `ComponentEvent` (un composant caché, montré, déplacé, retaillé)

Dans le paragraphe qui suit, nous allons nous intéresser plus particulièrement à `ActionEvent`.

2.3. Principes de gestion des événements

Sur une interface graphique Java, il y a un très grand nombre d'événements qui se génèrent. En effet, il est hors de question de les gérer tous. Ainsi, il est alors indispensable d'écouter uniquement les événements qui nous intéressent sur les composants ciblés. Pour arriver à être à l'écoute de ces événements ciblés, il devient nécessaire de créer des écouteurs, pour chaque « event » (événement) désiré. Il faudra ensuite penser à associer ces écouteurs aux composants cibles. Abordons la gestion des événements de type « Action ».

2.3.1. Notion d'écouteur (« ActionListener »)

Un écouteur est un objet destiné à recevoir et à gérer les événements générés par un composant de l'interface graphique. Les écouteurs principaux se trouvent aussi dans le package `java.awt.event`. En effet, la plupart de ces écouteurs sont définis par une interface java : n'importe quel objet peut devenir un écouteur.

2.3.2. Création d'écouteur

La structure de base de la création d'écouteur est la suivante :

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MonEcouleur implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        System.out.println("click !!");
    }
}
```

La méthode « actionPerformed » représente le message qui sera envoyé à l'écouteur. On inscrit un tel écouleur auprès d'un composant nommé bouton et cela comme suit : bouton.addActionListener(ecouteur). Quand un événement, un ActionEvent, est engendré par une action de l'utilisateur sur le bouton qui envoie le message « actionPerformed() » à l'écouteur. Le bouton lui passe l'événement déclencheur : **ecouteur.actionPerformed(unActionEvent);**

- TP 14 : Application d'un listener à un bouton

```
public Appli() {
    p = new MonJPanel();
    add(p);
}
```

```
public MonJPanel() {
    JButton b = new JButton("Click me !!");
    add(b);
}
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MonEcouleur implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        System.out.println("click !!");
    }
}
```

Pour utiliser mon écouleur, il faudra modifier le contenu de mon JPanel à :

```
public MonJPanel() {
    JButton b = new JButton("Click me !!");
    add(b);
    b.addActionListener(new MonEcouleur());
}
```


- TP 15 : Utilisation d'un écouteur par objet (bouton)

```
public class MonEcouteurBis implements ActionListener {  
  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("right choice !!");  
    }  
}  
  
public MonJPanel() {  
    JButton b = new JButton("Click me !!");  
    JButton b2 = new JButton("No Click me first !!");  
    add(b); add(b2);  
    b.addActionListener(new MonEcouteur());  
    b2.addActionListener(new MonEcouteurBis());  
}
```

NB : Il est aussi possible de créer plusieurs écouteurs pour un même objet.

- TP 16 : Utilisation d'un écouteur pour plusieurs objets

```
public MonJPanel() {  
    JButton b = new JButton("Click me !!");  
    JButton b2 = new JButton("No Click me first !!");  
    add(b); add(b2);  
    MonEcouteur m = new MonEcouteur();  
    b.addActionListener(m);  
    b2.addActionListener(m);  
}
```

- TP 17 : Association de la source des événements aux écouteurs

```
public class MonEcouteur implements ActionListener {  
  
    public void actionPerformed(ActionEvent e) {  
        JButton b = (JButton)e.getSource();  
  
        if (b.getActionCommand().equals("Click me !!"))  
            System.out.println("click !!");  
        else  
            System.out.println("right choice !!");  
    }  
}
```

On peut utiliser également `getText()` à la place de `getActionCommand()`.

2.3.3. Ecouter les événements de souris

Pour la souris, elle peut avoir 7 actions possibles :

- MouseListener

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

- MouseMotionListener

Bouton de la souris est pressé souris déplacée

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

Souris déplacée et le curseur se trouve dans un composant

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

Implanter « MouseListener » signifie redéfinir toutes les méthodes! Une solution consiste à utiliser « MouseAdapter » comme suit :

```
class MouseAdapter implements MouseListener {  
    public void mousePressed(MouseEvent e) {}  
    public void mouseClicked(MouseEvent e) {}  
    public void mouseReleased(MouseEvent e) {}  
    public void mouseEntered(MouseEvent e) {}  
    public void mouseExited(MouseEvent e) {}  
    public void mouseDragged(MouseEvent e) {}  
    public void mouseMoved(MouseEvent e) {}  
}  
  
class EcouteMaSouris extends MouseAdapter {  
    public void mouseClicked(MouseEvent e) {  
        // bla bla bla bla  
    }  
}
```

▪ TP 18 : Ecouter une souris

```
public Appli() {  
    p = new JPanel();  
    add(p);  
    MonEcouteurSouris ecouteur = new MonEcouteurSouris();  
    p.addMouseListener(ecouteur);  
}
```

OU

```
public void mouseClicked(MouseEvent e) {
    System.out.println("position du clic : x="+e.getX()+";y="+e.getY());
}

public void mousePressed(MouseEvent e) {
}

public void mouseReleased(MouseEvent e) {
}

public void mouseEntered(MouseEvent e) {
    System.out.println("Entrée dans la zone de clic");
}

public void mouseExited(MouseEvent e) {
    System.out.println("Sortie dans la zone de clic");
}
```

- TP 19 : Être son propre écouteur

```
public Appli(){
    this.addMouseListener(this);
}

public void mouseClicked(MouseEvent e) {
    System.out.println("position du clic : x="+e.getX()+";y="+e.getY());
}

public void mousePressed(MouseEvent e) {
}

public void mouseReleased(MouseEvent e) {
}

public void mouseEntered(MouseEvent e) {
    System.out.println("Entrée dans la zone de clic");
}

public void mouseExited(MouseEvent e) {
    System.out.println("Sortie dans la zone de clic");
}
```

- TP 20 : Un écouteur pour plusieurs objets

```
public class MonEcouleur implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        JButton b = (JButton)e.getSource();

        if(b.getActionCommand().equals("Click me !!"))
            System.out.println("click !!");
        else
            System.out.println("right choice !!");
    }
}
```

Cette devient rapidement fastidieux et nécessite d'autres organisations. Il revient donc de :

- ✓ créer une nouvelle classe écouteur par composant ! Mais pas de manière classique (avec un fichier par classe)
- ✓ Il est possible de définir une classe à l'intérieur du code d'une autre classe
- ✓ créer la classe écouteur correspondant à un composant immédiatement après son initialisation en utilisant une classe interne anonyme !

▪ TP 21 : Classe anonyme

The diagram illustrates the creation and use of an anonymous inner class for button click events. It consists of a code block with several annotations:

- Création et ajout du bouton**: Points to the code that creates a `JButton` and adds it to the application.
- La classe interne anonyme pour définir l'écouteur du bouton**: Points to the anonymous inner class that implements `ActionListener` and calls `monBouttonActionPerformed`.
- Gestion de l'événement**: Points to the `monBouttonActionPerformed` method that handles the click event by printing a message.

```
public Appli(){
    monBoutton = new javax.swing.JButton();
    monBoutton.setText("Test");
    add(monBoutton);

    monBoutton.addActionListener(
        new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                monBouttonActionPerformed(evt);
            }
        }
    );

    private void monBouttonActionPerformed(ActionEvent evt) {
        System.out.println("le bouton Test a été cliqué");
    }
}
```

OU plus court

```
monBoutton = new javax.swing.JButton();
JButton b = new javax.swing.JButton("wow");
add(b);
monBoutton.setText("Test");
add(monBoutton);

monBoutton.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            System.out.println("le bouton Test a été cliqué");
        }
    });
```

Ou pour une souris

```
monBoutton.addActionListener(
    new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            System.out.println("le bouton Test a été cliqué");
        }
    });

addMouseListener(
    new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            System.out.println("clac : "+e.getX()+" "+e.getY());
        }
    });
```

▪ TP 22 : TP

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

class ListenerGeneral implements ActionListener
{ Label etiq;
  Frame win;
  Button bout;
  //constructeur :
  public ListenerGeneral(Button bouton, Label etiquette, Frame window)
  { this.etiq = etiquette;
    this.win = window;
    this.bout = bouton;
  }
}
```

```
public void actionPerformed(ActionEvent e)
// Actions sur l'étiquette, la fenêtre, le bouton lui-même :
{ etiq.setText("changement");
  win.setTitle ("Nouveau titre");
  win.setBackground(Color.yellow);
  bout.setLabel("Merci");
}
}
class ListenerQuitter implements ActionListener
{ public void actionPerformed(ActionEvent e)
  { System.exit(0);
  }
}

class AppliWindowEvent
{
  public static void main(String [] arg) {
    Frame fen = new Frame ("Bonjour" );
    fen.setBounds(100,100,250,120);
    fen.setLayout(new FlowLayout( ));
    Button entree = new Button("Entrez");
    Button quitter = new Button("Quitter l'application");
    Label texte = new Label("Cette ligne est du texte");
    entree.addActionListener(new ListenerGeneral( entree, texte, fen ));
    quitter.addActionListener(new ListenerQuitter( ));
    fen.add(texte);
    fen.add(entree);
    fen.add(quitter);
    fen.setVisible(true);
  }
}
```

- TP 23 :

3. Connexion de java à la base de données

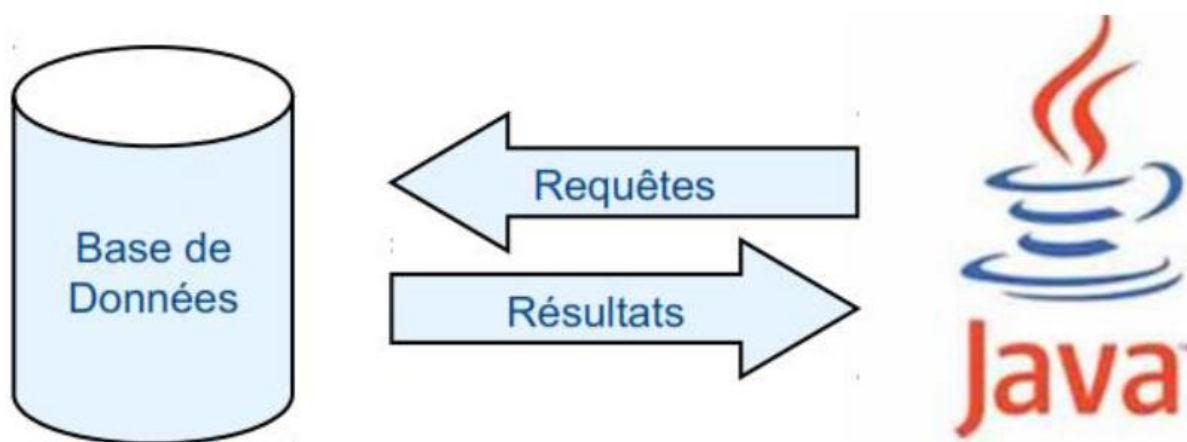
3.1. Défi du concept

Une application informatique a le plus souvent (surtout en informatique d'entreprise) besoin d'interagir avec une ou plusieurs bases de données. Or les bases de données sont en général centralisées et servent un grand nombre de «clients ». Cependant, les bases de données se contentent donc en général de stocker l'information, éventuellement de la valider, mais pas de la traiter. Ensuite, elles se mettent à la disposition des applications pour leur fournir l'information demandée ou pour stocker leurs modifications (en SQL).

Comment une application Java peut-elle interagir avec une base de données externe (par exemple php_my_admin) ?

Comment permettre à l'application de se connecter à la base de données, de lui envoyer des requêtes et d'en récupérer le résultat ?

Pour rendre possible ces interrogations, nous devons faire recours aux procédures ODBC.



3.2. Solution ODBC

Dans le monde Windows, il existe une plateforme baptisée ODBC (Open DataBase Connectivity). Il s'agit d'une interface standard permettant de mettre des bases de données à disposition des applications Windows. Chaque PC équipé de Windows possède une liste des "DataSources" ODBC qui pointent chacune vers une base de données. ODBC "connaît" chaque type de base de données Windows et sait "dans quel langage" lui parler. Autrement dit, il existe un "pilote ODBC" pour la plupart des bases de

données qui tournent sous Windows. Pour qu'un programme puisse accéder à une base de données, il suffit donc que celle-ci soit déclarée dans la liste des sources ODBC. Mais un problème demeure : Java n'est pas une plateforme Windows, donc il n'est pas lié en soi à ODBC. Les applications Java doivent en effet pouvoir dialoguer avec n'importe quelle base de données. Pour ce faire, Java possède son propre système de communication avec les bases de données : JDBC (Java DataBase Connectivity) qui est un ensemble de classes prédéfinies pour chaque type de BD.

A l'instar d'ODBC, le JDBC peut dialoguer avec un grand nombre de bases de données différentes (mais pas Access en tant que tel). En particulier, JDBC comprend un pilote qui lui permet de dialoguer avec ODBC. Pour accéder à une base de données ODBC, un programme Java doit donc établir une connexion vers ODBC en précisant la DataSource dont il a besoin. On parle en général d'un «Pont JDBC-ODBC».

3.3. Mise en œuvre du JDBC

Pour établir une connexion JDBC, quelques lignes de code suffisent :

```
public static final String jdbcdriver = "sun.jdbc.odbc.JdbcOdbcDriver";

public      static      final      String      database      =
"jdbc:odbc:NomDeLaDataSourceODBC";

private Connection conn;

try {Class.forName(jdbcdriver);

conn = DriverManager.getConnection(database);

} catch(Exception e) {e.printStackTrace(); }
```

Pour envoyer des requêtes à la BD une fois la connexion établie, il faut créer

un « Statement » : **Statement stmt = conn.createStatement();**

Une fois le Statement créé, on peut alors lui envoyer les requêtes SQL sous la forme de chaînes de caractères :

- **String requete = "DELETE * FROM clients";**
- **stmt.executeUpdate(requete);**

Mais dans le cas de requêtes SELECT, il faut récupérer le résultat. Les données issues de la requête sont rassemblées dans un jeu de résultats, appelé «ResultSet» :

- String requete = "SELECT * FROM client"

- ResultSet rs = stmt.executeQuery(requete);

On peut alors accéder aux enregistrements renvoyés un à un en demandant chaque fois au ResultSet de nous renvoyer la ligne suivante :

Exemple :

```
while(rs.next()){  
    int c_id = rs.getInt("client_id");  
    String c_n = rs.getString("client_name");  
    String c_fn = rs.getString("client_firstname");  
    String c_ad = rs.getString("client_address");  
    String c_ph = rs.getString("client_phone");  
    int c_ab = rs.getInt("client_abonne");  
    double c_c = rs.getDouble("client_credit");  
    int c_am = rs.getInt("client_anneemembre");  
}
```

Notez que l'on récupère la valeur de chaque champ en utilisant la méthode correspondant au type de données du champ (getInt, getString, getDate, etc.). Ce faisant, on a alors récupéré les données correspondant à chaque enregistrement. Il nous reste donc à utiliser ces données pour créer un objet du type souhaité :

Client c = new Client(c_id,c_n,c_fn,c_ad,c_ph,c_c,c_am);

Nous pouvons alors stocker ces clients un à un dans l'ArrayList:

clients.add(c) ; // A supposer que nous ayons une variable membre clientsde // type ArrayList<Client> dans la classe

Il faut enfin toujours veiller à fermer les connexions :

rs.close(); stmt.close(); conn.close();

4. Quelques Projets d'applications

IFRI-UAC

IFRI-UAC

IFRI-UAC