

# **Java : Partie 5**

## **Interface graphique Java**

### **Client Riche**

Juin 2013

Auteur : Chouaïb LAGHLAM

Concepteur Développeur Informatique  
Module : Programmation Java  
Interface graphique : Client Riche

---

**Interface graphique Java : Sommaire**

|  |          |
|--|----------|
| <b>Préalabe.....</b>   | <b>4</b> |
| <b>Interface graphique ? .....</b>   | <b>4</b> |
| Interface graphique Client Riche, Interface graphique Client léger, Interface graphique client mobile..... | 4        |
| L'interface graphique client Riche .....   | 5        |
| L'interface graphique client léger.....  | 5        |
| L'interface graphique client mobile .....  | 5        |
| Technologies Interface graphique client Riche .....  | 5        |
| La technologie AWT : Abstract Window Toolkit.....  | 5        |
| La technologie SWING.....  | 6        |
| Technologie SWT : Standard Widget Toolkit.....   | 7        |
| <b>SWING .....</b>   | <b>7</b> |
| Les Classes usuelles pour composants graphiques SWING.....   | 7        |
| Affichage de boîtes de dialogues.....  | 9        |
| Afficher un message d'information .....  | 9        |
| Demander une confirmation.....   | 9        |
| Demande d'une information par une boîte de dialogue .....  | 10       |
| Faire un choix dans une boîte de dialogue.....   | 10       |
| Choisir une couleur .....  | 12       |
| Sélectionner un fichier .....  | 13       |
| Fabriquer ses propres fenêtres SWING .....   | 15       |
| Un premier exemple de code Java pour une fenêtre simple .....  | 15       |
| Faire une classe dédiée à la fenêtre .....   | 16       |
| Organisation d'une fenêtre.....  | 18       |
| Le panneau principal.....  | 19       |
| Les gestionnaires de placements.....   | 19       |
| Exemples de composants graphiques.....   | 30       |
| Gestion des actions utilisateur .....  | 30       |
| Listeners internes .....   | 30       |
| Un listener interne commun à plusieurs composant .....   | 33       |

Concepteur Développeur Informatique  
Module : Programmation Java  
Interface graphique : Client Riche

---

|                                    |    |
|------------------------------------|----|
| Listeners externes.....            | 36 |
| Gestion de Threads pour SWING..... | 38 |
| Thread ? .....                     | 38 |

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

## Préalabe

Avant de commencer à lire cette 5<sup>ème</sup> partie, vous êtes supposé avoir les connaissances de :

- ➔ La 1<sup>ère</sup> partie qui permet d'installer l'environnement de développement : Eclipse,
- ➔ La 2<sup>ème</sup> partie qui initie à la programmation de base en Java,
- ➔ La 3<sup>ème</sup> partie qui vous donne les connaissances nécessaires en Programmation Orientée Objets (POO),
- ➔ La 4<sup>ème</sup> partie qui vous a initié à la persistance des données

Merci de créer dans votre **Workspace**, dans votre projet « **prj\_Java\_040\_SWING**»,

- Le package « **pack\_30\_BoitesDeDialogues**» sous le dossier **src**,
- Créez un package « **pack\_20\_Fenetres**» sous le dossier **src**,

## Inteface graphique ?

### Interface graphique Client Riche, Interface graphique Client léger, Interface graphique client mobile

Aujourd'hui, une application informatique qui a une interface utilisateur est souvent graphique.

Une application peut :

- ➔ **Vouloir s'exécuter de la même façon** sur :
  - Des supports différents : Sur ordinateurs personnels, sur Smartphones, sur le Web,
  - Quel que soit le moyen de communication : via le réseau local, via le wifi, via la 3G/4G, ....
  - Quel que soit le système d'exploitation : Windows, linux, Unix, Androïd, iOS, OS/400, .....
- ➔ **Vouloir s'exécuter de façons différentes** sur :
  - Sur le réseau local et avec des identifiants : ajout et modification de données stratégiques,
  - Sur le Web et uniquement si le matériel connecté se trouve en Europe : les utilisateurs autorisés Peuvent modifier leurs propres données,
  - ....

Comme le type d'appareils connectés n'arrêtent pas d'augmenter (bientôt des montres, des lunettes, des voitures connectés, ..... ) : **il n'est pas question de faire, pour une application, une version par type d'appareil.**

Il incombe au Chef de projet de bien concevoir son application afin qu'elle puisse :

- ➔ Permettre l'ajout d'un nouveau type d'appareil aisément, sans tout remettre en question,
- ➔ De s'adapter facilement aux exigences graphiques de chaque type d'appareils,
- ➔ De partager les ressources communes à tous ces appareils connectés : les BD, les fichiers, les photos, les images, ...

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

Très souvent, le chef de projet a recours à **l'architecture MVC** pour concevoir son application afin de respecter les exigences citées ci-dessus. (Voir le cours sur le MVC).

#### L'interface graphique client Riche

Le client ici est l'appareil sur lequel va s'afficher l'interface.

Le terme « client riche » est utilisé pour **l'affichage en fenêtres** sur des ordinateurs portables ou de bureau. Les applications telles que Word ou Excel ont une interface graphique Riche.

#### L'interface graphique client léger

Ici l'affichage se fait sur un navigateur (un browser) tels qu'Internet Explorer, Chrome, Firefox, .....

Il s'agit donc d'applications Web,

#### L'interface graphique client mobile

Le client mobile désigne tout appareil mobile autre que les ordinateurs personnels : les iPhone, les Smartphones, Les tablettes, les montres connectées, les lunettes, les applications dans les voitures, .....

Ces appareils nécessitent, outre une taille réduite d'affichage, le recours à des technologies non existantes actuellement sur tous les ordinateurs :

- ➔ La reconnaissance vocale, la saisie et la sélection tactile, les technologies 3G/4G, l'interaction avec des applications d'usage généralisé : Facebook, Twitter, ....

## Technologies Interface graphique client Riche

En programmation Java, l'affichage graphique doit produire le même résultat ou presque, quel que soit le système d'exploitation utilisé sur les appareils qui exécutent l'application : Windows, Linux, Unix, Android, OS7, OS400, .....

Cela est rendu possible grâce à la machine virtuelle Java mais des retouches sont nécessaires pour adapter à 100 % Une application à certain type d'appareil ou de système d'exploitation.

La société **Sun Systems** qui est à l'origine de la création du langage java, a inventé successivement deux technologies pour l'interface graphique sous forme de fenêtres :

#### La technologie AWT : Abstract Window Toolkit

Elle a été largement utilisée jusqu'aux années 2000,

Mais les développeurs ont commencé à constater :

- Des ralentissements d'affichage lorsque les fenêtres sont riches en composants,
- Des manques de composants modernes : par exemple le composant calendrier qui permet de choisir une date,...

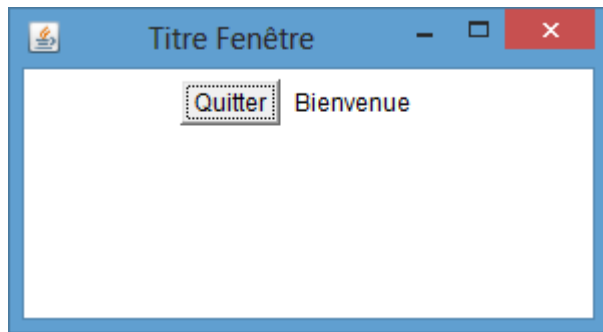
Exemple basique d'une fenêtre fabriquée selon AWT :

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---



Sun Systems va faire évoluer sa technologie AWT vers la technologie SWING

#### La technologie SWING

Les composants graphique d'AWT sont considérés comme lourds alors que ceux de SWING sont appelés composants légers,

SWING ne remplace pas AWT mais elle est une extension complémentaire d'AWT,

On parle d'API SWING : c'est-à-dire d'un ensemble de bibliothèques de classes java qui permettent de faire L'affichage de fenêtres selon cette technologie.

Le développeur java doit importer, dans son code, au minimum les espaces de noms suivants :

**import java.awt.\* ;**

Et

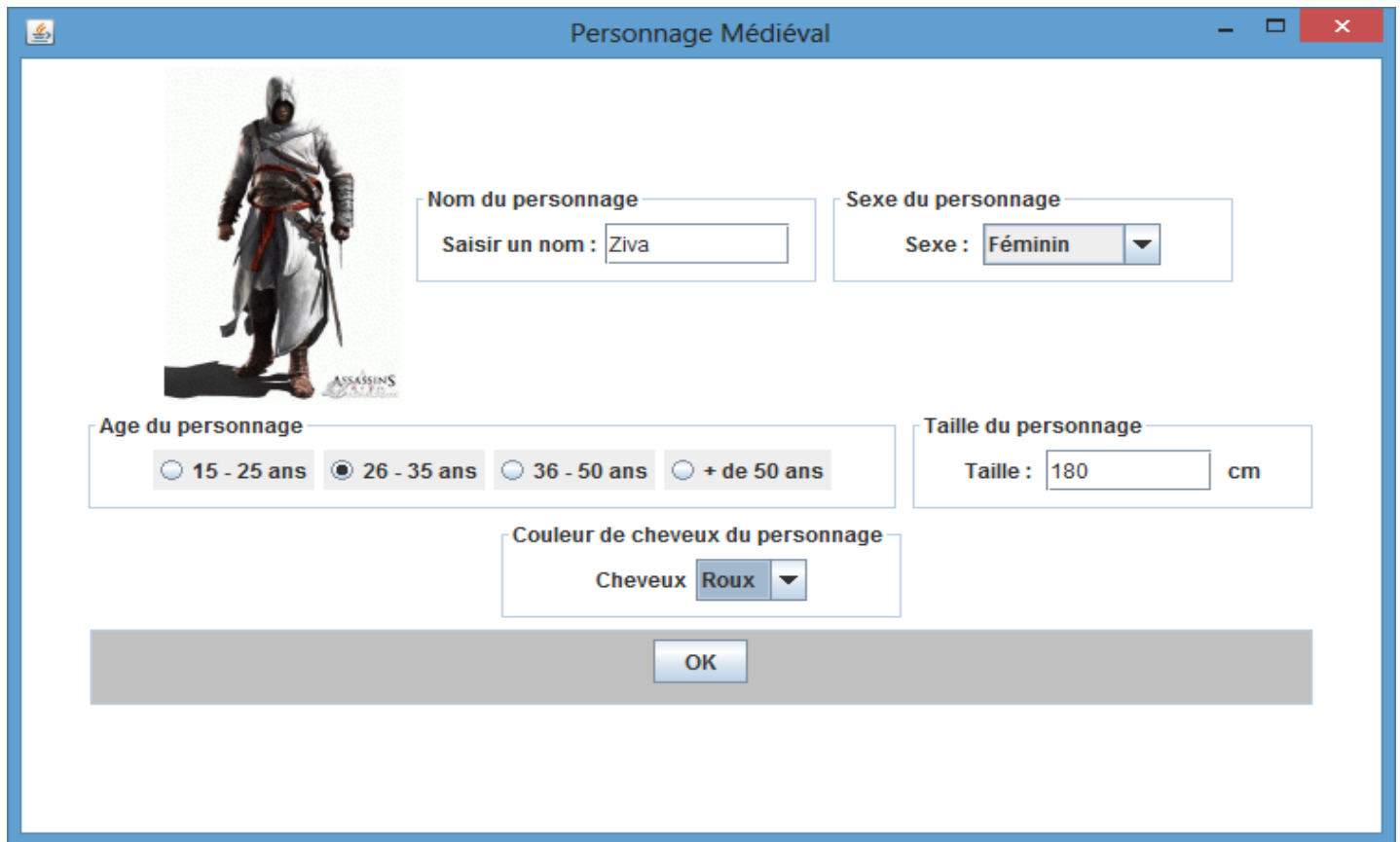
**import javax.\* ;**

Exemple de fenêtre fabriquée avec SWING :

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche



The screenshot shows a Java Swing window titled "Personnage Médiéval". Inside the window, there is a character image on the left. To the right of the image, there are several form elements: a text field for "Nom du personnage" with the value "Ziva", a dropdown menu for "Sexe du personnage" with the value "Féminin", a group of radio buttons for "Age du personnage" with "26 - 35 ans" selected, a text field for "Taille du personnage" with the value "180" and "cm" next to it, and a dropdown menu for "Couleur de cheveux du personnage" with the value "Roux". At the bottom center of the window is an "OK" button.

Technologie SWT : Standard Widget Toolkit

La compagnie IBM est à l'origine de la création d'une autre technologie graphique client riche,

Cette technologie définit une façon spécifique de fabriquer une fenêtre (une perspective) principale et d'y disposer des Petites fenêtres (des vues et des éditeurs).

L'environnement de développement **Éclipse** repose sur cette technologie SWT.

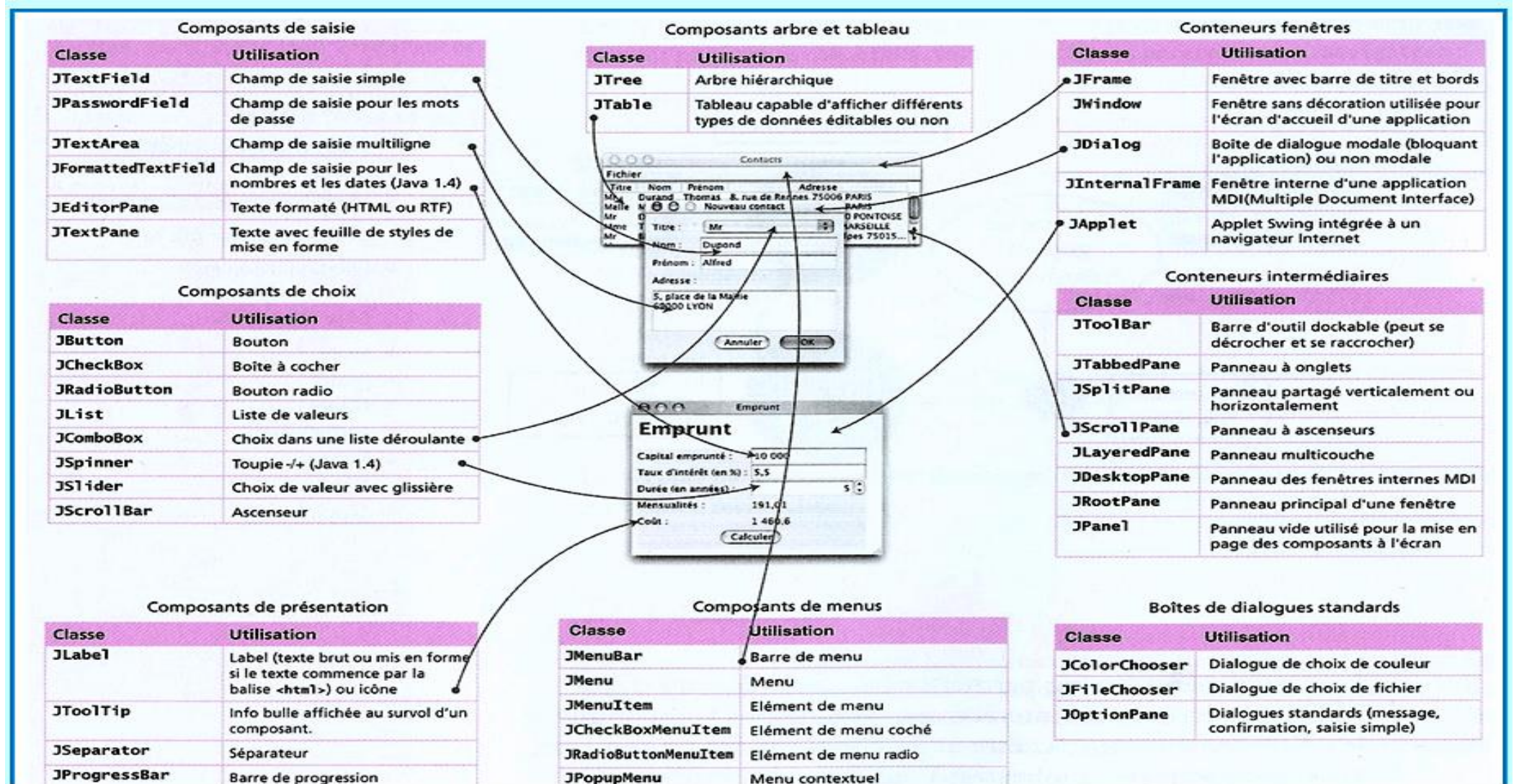
## SWING

Nous allons, dans ce cours, utiliser la technologie SWING.

### **Les Classes usuelles pour composants graphiques SWING**

Voici un tableau récapitulatif des classes graphiques courantes dans SWING :







# Concepteur Développeur Informatique

## Module : Programmation Java

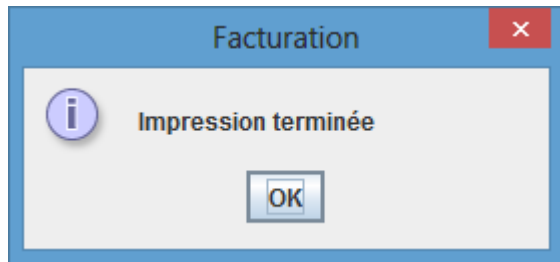
### Interface graphique : Client Riche

---

#### Affichage de boîtes de dialogues

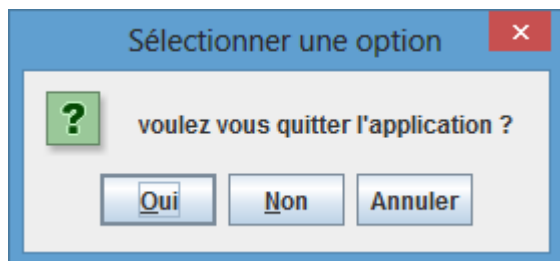
En bas à droite du tableau ci-dessus, nous avons des classes SWING qui permettent d'afficher des fenêtres prédéfinies que l'on appelle « Boîtes de dialogue ». Exemples choisis :

##### Afficher un message d'information



```
JOptionPane.showMessageDialog(null,  
    "Impression terminée",  
    "Facturation",  
    JOptionPane.INFORMATION_MESSAGE);
```

##### Demander une confirmation



```
int reponse=JOptionPane.showConfirmDialog(null,  
    "voulez vous quitter l'application ?");  
switch(reponse)  
{  
    case    JOptionPane.YES_OPTION:                // clic sur OUI de boîte de diag  
        JOptionPane.showMessageDialog(null,  
            "A bientôt",  
            "Facturation",  
            JOptionPane.INFORMATION_MESSAGE);  
        System.exit(0);  
        break;  
    case    JOptionPane.NO_OPTION:                 // clic sur NON de boîte de diag  
        JOptionPane.showMessageDialog(null,  
            "Merci de votre fidélité",  
            "Facturation",
```

# Concepteur Développeur Informatique

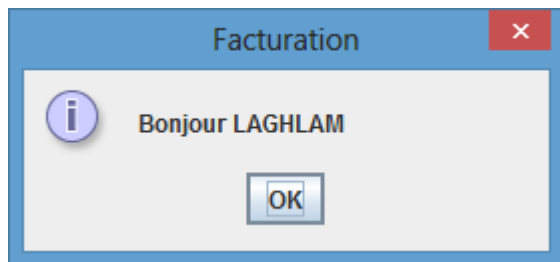
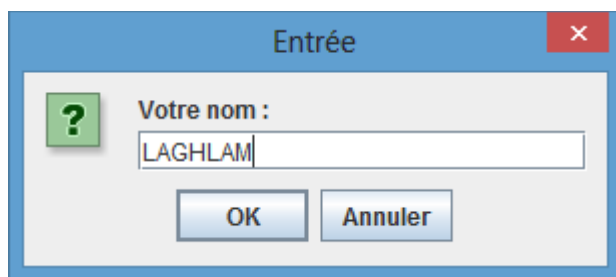
## Module : Programmation Java

### Interface graphique : Client Riche

---

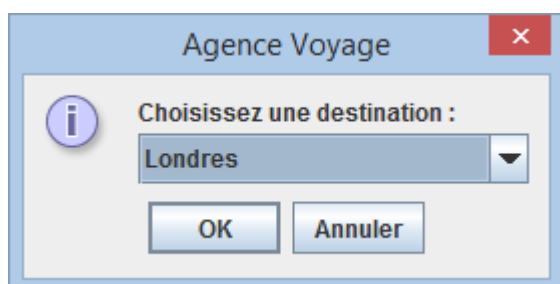
```
                                JOptionPane.INFORMATION_MESSAGE);
        break;
    case JOptionPane.CANCEL_OPTION:    // clic sur ANNULER de boîte de diag
        JOptionPane.showMessageDialog(null,
            "Décidez vous",
            "Facturation",
            JOptionPane.INFORMATION_MESSAGE);
        break;
    }
                                                                    // fin switch
```

#### Demande d'une information par une boîte de dialogue



```
String nomSaisi=JOptionPane.showInputDialog("Votre nom : ");
JOptionPane.showMessageDialog(null,
    "Bonjour "+nomSaisi,
    "Facturation", JOptionPane.INFORMATION_MESSAGE);
```

#### Faire un choix dans une boîte de dialogue

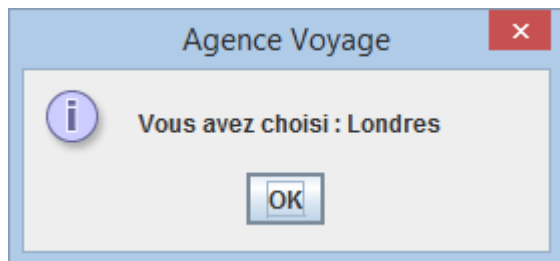


# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---



```
// affichage d'une liste dans une boîte de dialogue
String [] lesVilles={"Barcalone","Londres","Paris","Athènes"};
String couleurChoisie=(String) JOptionPane.showInputDialog(null,
    "Choisissez une destination : ",
    "Agence Voyage",
    JOptionPane.INFORMATION_MESSAGE,
    null,
    lesVilles,
    lesVilles[1]);                                // ville à afficher par défaut

// affichage de la couleur saisie
JOptionPane.showMessageDialog(null,
    "Vous avez choisi : "+couleurChoisie,
    "Agence Voyage",
    JOptionPane.INFORMATION_MESSAGE);
```

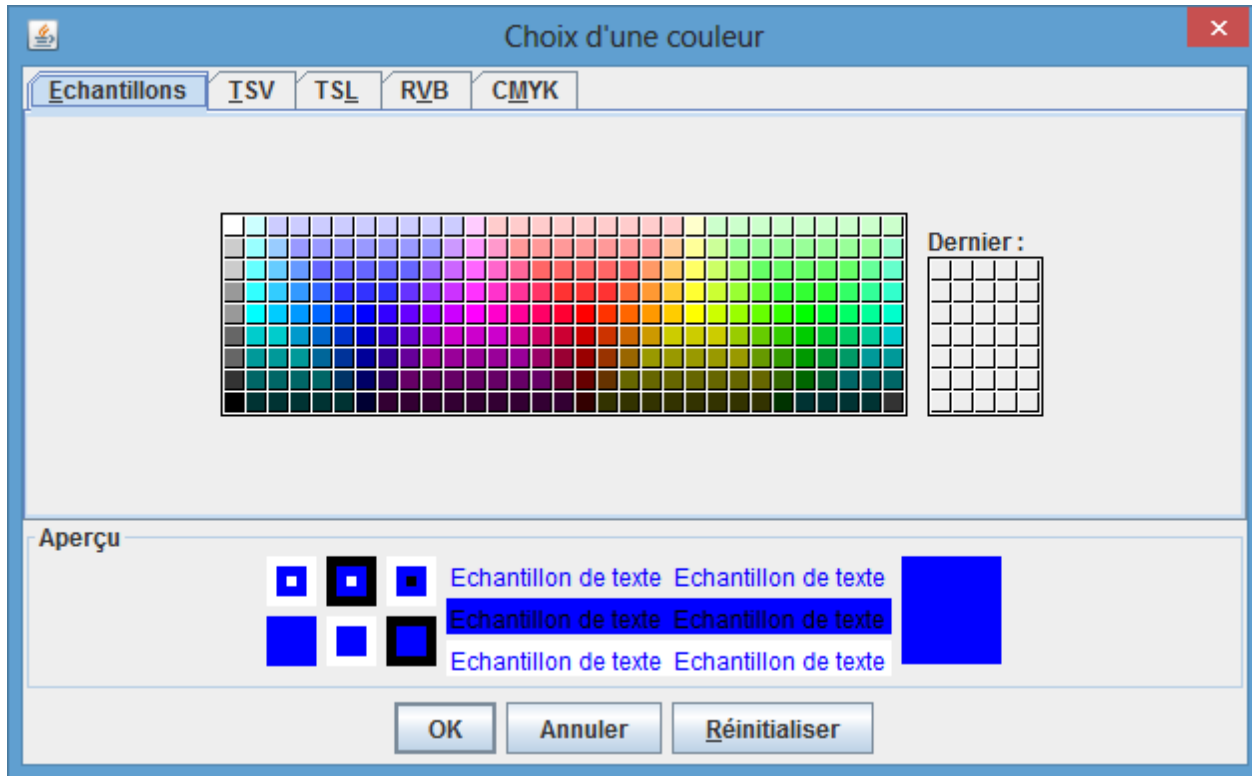
# Concepteur Développeur Informatique

## Module : Programmation Java

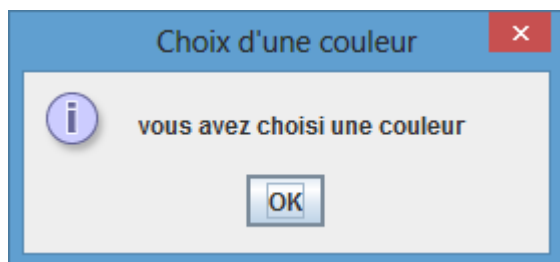
### Interface graphique : Client Riche

---

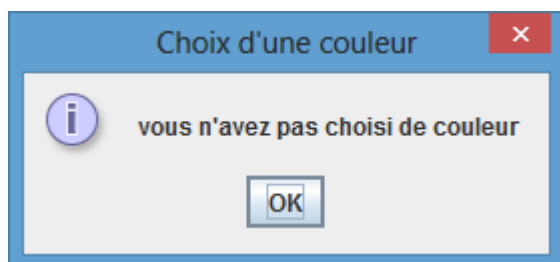
#### Choisir une couleur



Si l'utilisateur clique sur « OK » :



Si l'utilisateur clique sur « Annuler » :



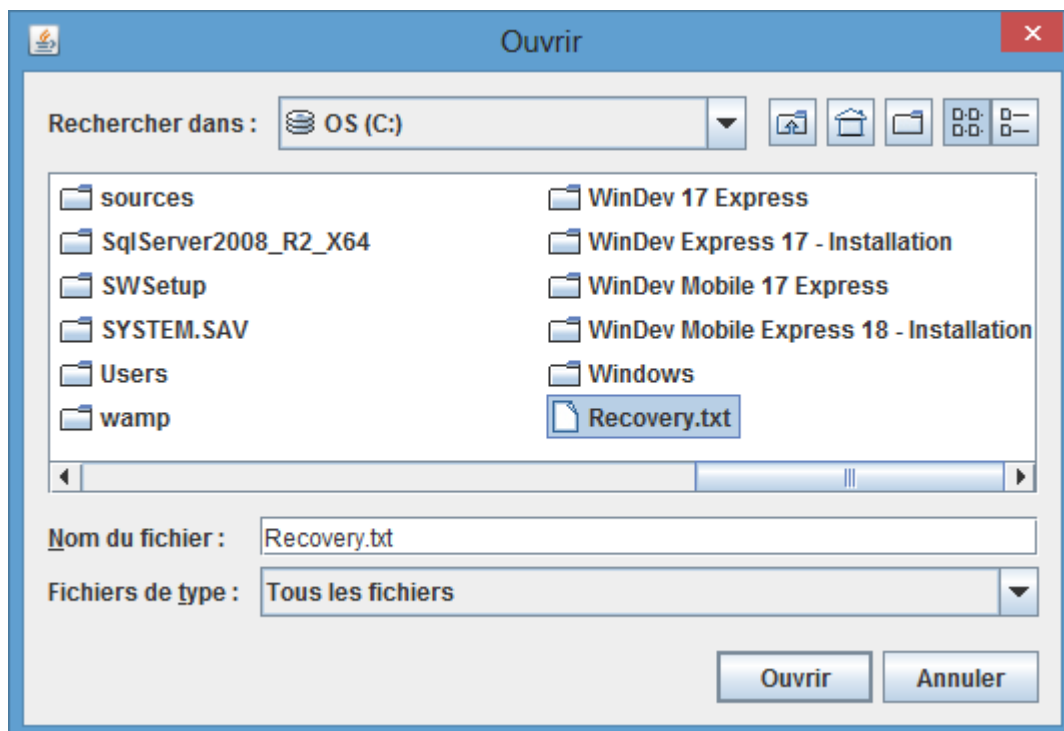
# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

```
Color couleurChoisie = JColorChooser.showDialog(  
    null,  
    "Choix d'une couleur",  
    Color.BLUE);  
if(couleurChoisie == null)  
{  
    JOptionPane.showMessageDialog(null,  
        "vous n'avez pas choisi de couleur",  
        "Choix d'une couleur",  
        JOptionPane.INFORMATION_MESSAGE);  
}  
else  
{  
    JOptionPane.showMessageDialog(null,  
        "vous avez choisi une couleur",  
        "Choix d'une couleur",  
        JOptionPane.INFORMATION_MESSAGE);  
}
```

#### Sélectionner un fichier



Si l'utilisateur clique sur «Ouvrir » :

# Concepteur Développeur Informatique

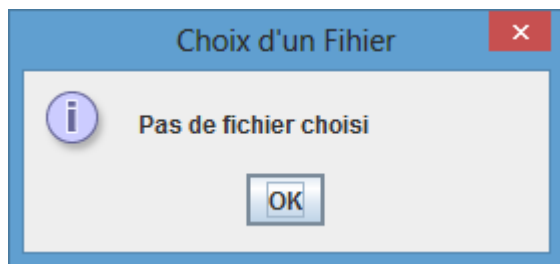
## Module : Programmation Java

### Interface graphique : Client Riche

---



Si l'utilisateur clique sur « Annuler » :



```
String fichierChoisi,chemin;
JFileChooser selecteurFichier = new JFileChooser();
int retour=selecteurFichier.showOpenDialog(null);
if(retour==JFileChooser.APPROVE_OPTION){
    // un fichier a été choisi (sortie par OK)
    // nom du fichier choisi
    fichierChoisi=selecteurFichier.getSelectedFile().getName();
    // chemin absolu du fichier choisi
    chemin=selecteurFichier.getSelectedFile().getAbsolutePath();
    JOptionPane.showMessageDialog(null,
        fichierChoisi+"\n"+chemin,
        "Choix d'un Fichier",
        JOptionPane.INFORMATION_MESSAGE);
}
else
{
    JOptionPane.showMessageDialog(null,
        "Pas de fichier choisi",
        "Choix d'un Fichier",
        JOptionPane.INFORMATION_MESSAGE);
}
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

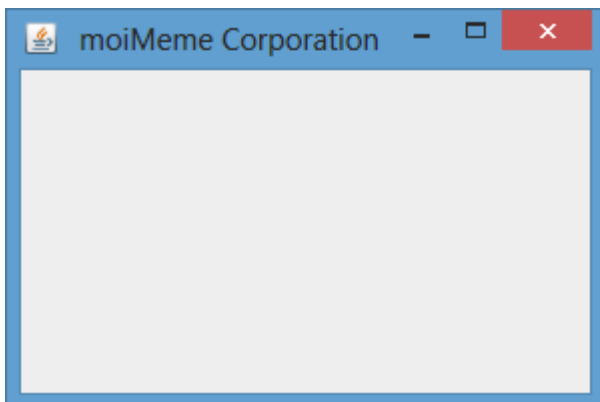
---

#### Fabriquer ses propres fenêtres SWING

Fabriquer une fenêtre graphiquement selon la technologie **SWING** dans l'environnement de développement **Éclipse** peut se faire :

- ➔ Soit à l'aide d'un outil graphique : un plugin à ajouter à Éclipse :
  - Le développeur utilise la souris pour créer la fenêtre et ses composants, l'outil génère le code Java correspondant,
  - Cette façon de faire suppose que le développeur maîtrise bien Java et plus particulièrement la technologie SWING. Car il n'est pas rare d'intervenir dans le code généré pour le modifier,
  - Quelques plugins bien utilisés : **Visual Editor**, **WindowBuilder**, ...
- ➔ Soit en codant manuellement la fabrication de la fenêtre et de ses objets :
  - Cette technique est recommandée aux débutants pour bien maîtriser la technologie SWING et Java
  - Puis par la suite, installer un outil graphique.

#### Un premier exemple de code Java pour une fenêtre simple



Code java :

```
package pack_20_Fenetres;

import javax.swing.JFrame;

public class Test_Ajeter {

    /* =====
       Fenêtre SWING Simple
       Auteur      : Chouaïb LAGHLAM
       Créé le     : Aout 2013
       Modifié le  :
       Modifié par :
       =====
    */

    public static void main(String[] args) {
```



# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
// je crée un objet fenêtre
JFrame mafen=new JFrame();
// donner une taille à la fenêtre
mafen.setSize(300, 200);
// donner un titre à la fenêtre
mafen.setTitle("moiMeme Corporation");
// centrer l'affichage de la fenêtre
mafen.setLocationRelativeTo(null);
// afficher la fenêtre
mafen.setVisible(true);
}
}
```

D'après le code ci-dessus et d'après le tableau donné précédemment, la classe qui permet de créer une fenêtre graphique est la classe **JFrame**

#### Faire une classe dédiée à la fenêtre

Le code montré ci-dessus est un code qui se trouve dans une méthode donnée de l'application.

Comment faire si plusieurs codes dans l'application (donc plusieurs méthodes) souhaitent afficher la même fenêtre ?

#### La solution :

- ➔ Créer une classe Java pour représenter la fenêtre Swing : c'est une classe qui hérite JFrame,
- ➔ Dans chaque code qui veut afficher la fenêtre : instancier la classe créée ci-dessus et l'afficher.

#### Exemple :

On souhaite afficher une fenêtre avec un texte non modifiable et un bouton :



# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

Classe dédiée à la description de la fenêtre :

```
package pack_20_Fenetres;
import java.awt.HeadlessException;
import javax.swing.*;

public class Win_20_fenêtreSimple_classeAPart extends JFrame{
    /*
        SWING :      Classe pour une fenêtre SWING
                     hérite de JFrame
    */

    /*
        //      Attributs de la fenêtre : composants graphiques de la fenêtre
        //                                             + eventuelles constantes
        JPanel panneau;           // panneau = fond d'écran
        JLabel lblNom;           // un texte non modifiable
        JButton btnQuitter;       // un bouton à cliquer
        // constructeur
        public Win_20_fenêtreSimple_classeAPart() throws HeadlessException {
            super();
            panneau = new JPanel();
            lblNom = new JLabel("Hello Les Développeurs");
            btnQuitter = new JButton("Quitter");
            // attacher les composants au panneau
            panneau.add(lblNom);
            panneau.add(btnQuitter);
            // associer le panneau à la fenêtre
            this.setContentPane(panneau);
            // régler les propriétés de la fenêtre
            this.setTitle("ma première Fenêtre SWING"); // titre de la fenêtre
            this.setSize(400, 200); // largeur et hauteur de la fenêtre
            this.setResizable(true); // empêcher le redimensionnement de la fen
            this.setLocationRelativeTo(null); // On centre la fenêtre sur l'écran
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
        }
    }
}
```

Le code qui crée et affiche la fenêtre :

```
package pack_20_Fenetres;

public class TestFenêtres {
    /*
        *      test des fenêtres SWING
    */

    public static void main(String[] args) {
        //
        // créer et afficher une fenêtre à partir d'une classe
        //
        Win_20_fenêtreSimple_classeAPart f1=
```

# Concepteur Développeur Informatique

## Module : Programmation Java

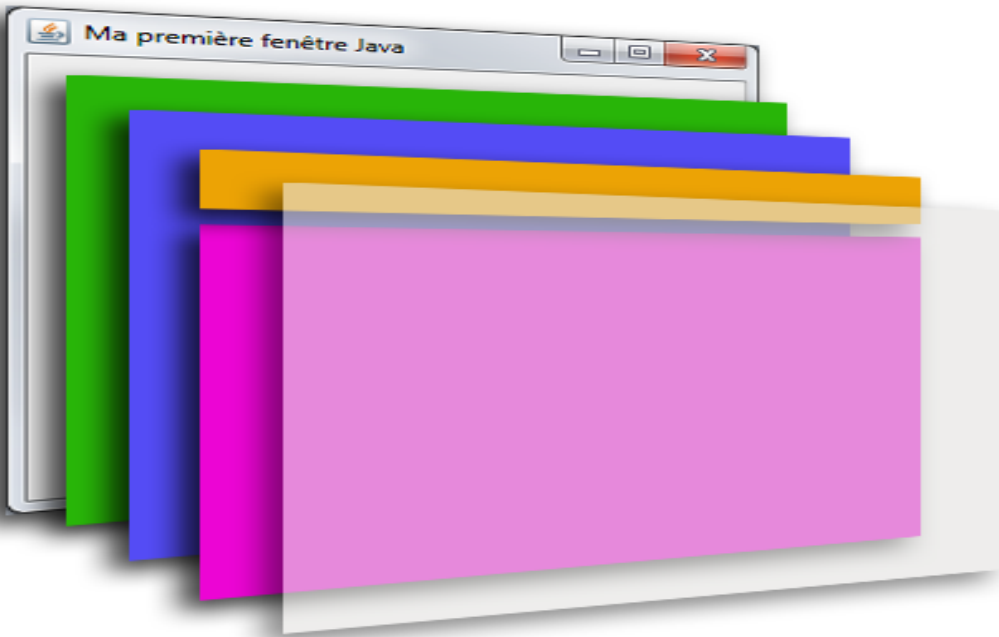
### Interface graphique : Client Riche

---

```
        new Win_20_fenêtreSimple_classeAPart();
        f1.setVisible(true);
    }
}
```

## Organisation d'une fenêtre

Une fenêtre Swing est organisée ainsi :



Structure d'une JFrame

Nous avons, dans l'ordre :

- la fenêtre ;
- le **RootPane** (en vert), le conteneur principal qui contient les autres composants ;
- le **LayeredPane** (en violet), qui forme juste un panneau composé du conteneur global et de la barre de menu (**MenuBar**) ;
- la **MenuBar** (en orange), la barre de menu, quand il y en a une ;
- le **content pane** (en rose) : c'est dans celui-ci que nous placerons nos composants ;
- le **GlassPane** (en transparence), couche utilisée pour intercepter les actions de l'utilisateur avant qu'elles ne parviennent aux composants.

**On retiendra ceci :**

Les composants doivent être posés dans un **panneau** ou **Panel** (classe JPanel) : couche content pane,

Ces composants doivent être placés selon une gestion bien choisie : on utilise un «**Gestionnaire de placement**» ou **Layout**.

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

#### Le panneau principal

Une fenêtre doit être dotée d'un fond (d'un panneau principal) : procédez ainsi :

- ➔ Créez un **panneau** : un objet de type JPanel pour créer le fond de la fenêtre sur lequel poser les composants,
- ➔ Créez les composants et les ajouter au panneau,
- ➔ Associer le panneau à la fenêtre.

Remarque : on peut mettre des panneaux secondaires dans le panneau principal pour mieux décomposer la fenêtre.

#### Les gestionnaires de placements

L'emplacement des composants, à mettre dans le panneau principal, doit être géré.

Par défaut les composants se placent de haut en bas et de gauche à droite en tenant compte de la taille de la fenêtre.

Généralement, le placement par défaut n'est pas satisfaisant.

Je peux également, préciser pour chaque composant ses dimensions (largeur, hauteur) et son emplacement (positions x et y) dans la fenêtre mais devient très compliqué pour fenêtre très riche en composant.

SWING propose le concept de «**Gestionnaire de placement**» ou «**Layout**» ;

On nous fournit pour cette gestion un ensemble de classes :

| Classe                                | Description  |
|---------------------------------------|--|
| <code>java.awt.FlowLayout</code>      | Dispose les composants d'un container les uns derrière les autres en ligne et à leur taille préférée, en retournant à la ligne si le container n'est pas assez large.  |
| <code>java.awt.GridLayout</code>      | Dispose les composants d'un container dans une grille dont toutes les cellules ont les mêmes dimensions.   |
| <code>java.awt.BorderLayout</code>    | Dispose cinq composants maximum dans un container, deux au bords supérieur et inférieur à leur hauteur préférée, deux au bords gauche et droit à leur largeur préférée, et un au centre qui occupe le reste de l'espace.   |
| <code>java.awt.CardLayout</code>      | Affiche un composant à la fois parmi l'ensemble des composants d'un container (pratique pour créer des panneaux comme ceux de la boîte de dialogue de <i>Preferences</i> d'Eclipse).   |
| <code>javax.swing.BoxLayout</code>    | Dispose les composants en ligne à leur hauteur préférée ou en colonne à leur largeur préférée.   |
| <code>java.awt.GridBagLayout</code>   | Dispose les composants d'un container dans une grille dont les cellules peuvent avoir des dimensions variables. La position et les dimensions de la cellule d'un composant varient en fonction de sa taille préférée et des contraintes de classe <code>java.awt.GridBagConstraints</code> qui lui sont associées. |
| <code>javax.swing.SpringLayout</code> | Dispose les composants d'un container en fonction de leur taille préférée et de contraintes qui spécifient comment ces composants sont rattachés les uns par rap-  |

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

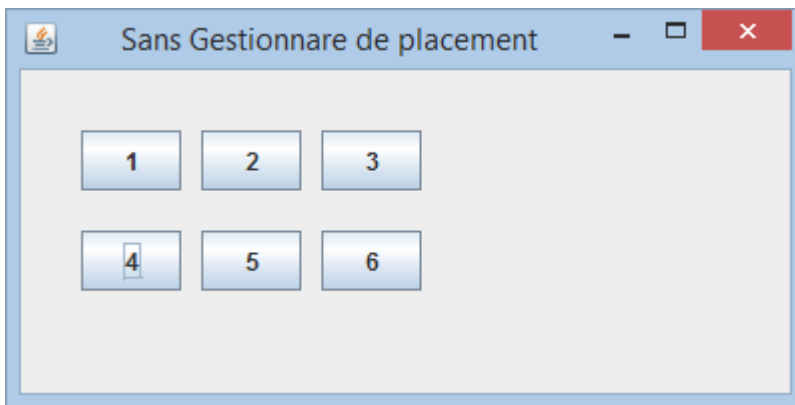
#### **Exemple de gestion d'emplacement de composants :**

Nous allons fabriquer une fenêtre avec six boutons et nous allons les placer selon différents gestionnaires de placement.

Nous allons fabriquer deux classes java :

- ➔ Une qui décrit la fenêtre,
- ➔ Une qui contient la méthode «main» et qui lance la fenêtre.

**Cas 1** : je crée mes six boutons et je les place moi-même les boutons sans gestionnaire de placement



Code de la fenêtre :

```
package pack_20_Fenetres;
import java.awt.HeadlessException;
import javax.swing.*;

public class Win_30_fenêtre_SansGestionnairesP extends JFrame{
    /*
        =====
        SWING :      Gestionnaires de placement
        =====
    */
    //
    //  Attributs de la fenêtre
    //
    JPanel panneau;
    JButton btn1,btn2,btn3,btn4,btn5,btn6;
    //
    // constructeur
    //
    public Win_30_fenêtre_SansGestionnairesP() throws HeadlessException {
        super();
        //
        panneau=new JPanel();
        btn1=new JButton("1");
        btn2=new JButton("2");
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
btn3=new JButton("3");
btn4=new JButton("4");
btn5=new JButton("5");
btn6=new JButton("6");
//      *****
//      préciser le gestionnaire de placement (layout) : ici aucun
//      *****
panneau.setLayout(null);
//      *****
//      créer les boutons manuellement et les ajouter au panneau principal
//      *****
btn1.setBounds(30,30, 50, 30);
btn2.setBounds(90,30, 50, 30);
btn3.setBounds(150,30, 50, 30);
btn4.setBounds(30,80,50, 30);
btn5.setBounds(90,80,50, 30);
btn6.setBounds(150,80,50, 30);
panneau.add(btn1);                                // associer objets au panneau
panneau.add(btn2);
panneau.add(btn3);
panneau.add(btn4);
panneau.add(btn5);
panneau.add(btn6);
//      *****
//      associer le panneau à la fenêtre
//      *****
this.setContentPane(panneau);
//      *****
//      régler les propriétés de la fenêtre
//      *****
this.setTitle("Sans Gestionnaire de placement");// titre de la fenêtre
this.setSize(400, 200);                          // largeur et hauteur de la fenêtre
this.setResizable(true);                          // empêcher le redimensionnement de la fen
this.setLocationRelativeTo(null);                  //On centre la fenêtre sur l'écran
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
```

```
}
```

```
}
```

# Concepteur Développeur Informatique

## Module : Programmation Java

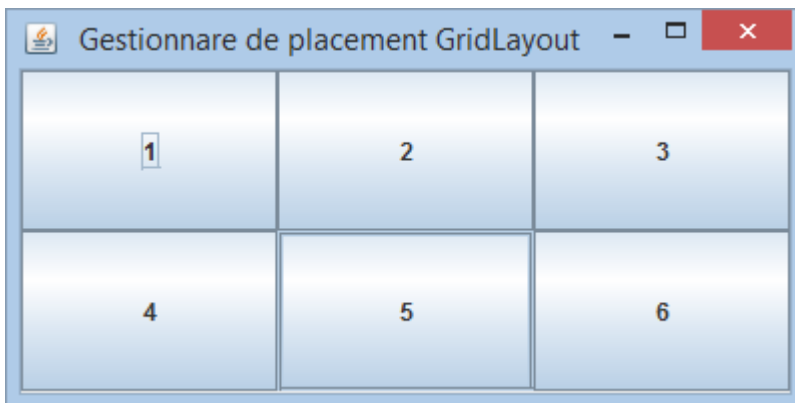
### Interface graphique : Client Riche

Le code (de la méthode main) qui crée et affiche la fenêtre (est le même pour tous les cas cités ici) :

```
20 // =====
21 // créer et afficher une fenêtre sans gestionnaire de placement
22 // =====
23 Win_30_fenêtre_SansGestionnairesP wingp1=
24     new Win_30_fenêtre_SansGestionnairesP();
25 wingp1.setVisible(true);
26 //
```

**Cas 2** : utilisation du gestionnaire de placement « GridLayout »

Il permet de placer les composants selon une grille en lignes et en colonnes



Code de la classe fenêtre :

```
package pack_20_Fenetres;
import java.awt.GridLayout;
import java.awt.HeadlessException;
import javax.swing.*.*;

public class Win_40_fenêtre_GestionnairesP_GridLayout extends JFrame{
    /* =====
       SWING :      Gestionnaires de placement GridLayout
       =====
    */
    //
    // Attributs de la fenêtre
    //
    JPanel panneau;
    JButton btn1,btn2,btn3,btn4,btn5,btn6;
    //
    // constructeur
```



# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
//
public Win_40_fenêtre_GestionnairesP_GridLayout() throws HeadlessException {
    super();
    //
    panneau=new JPanel();
    btn1=new JButton("1");
    btn2=new JButton("2");
    btn3=new JButton("3");
    btn4=new JButton("4");
    btn5=new JButton("5");
    btn6=new JButton("6");
    //      *****
    //      préciser le gestionnaire de placement (layout) : ici GridLayout
    //      *****
    panneau.setLayout(new GridLayout(2,3));
    //      *****
    //      créer les boutons manuellement et les ajouter au panneau principal
    //      *****
    panneau.add(btn1);                                     // associer objets au panneau
    panneau.add(btn2);
    panneau.add(btn3);
    panneau.add(btn4);
    panneau.add(btn5);
    panneau.add(btn6);
    //      *****
    //      associer le panneau à la fenêtre
    //      *****
    this.setContentPane(panneau);
    //      *****
    //      régler les propriétés de la fenêtre
    //      *****
    this.setTitle("Gestionnaire de placement GridLayout"); // titre de la fenêtre
    this.setSize(400, 200);                                // largeur et hauteur de la fenêtre
    this.setResizable(true);                               // empêcher le redimensionnement de la fen
    this.setLocationRelativeTo(null);                      // On centre la fenêtre sur l'écran
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
}
}
```

**Cas 3** : utilisation du gestionnaire de placement «**BorderLayout**»

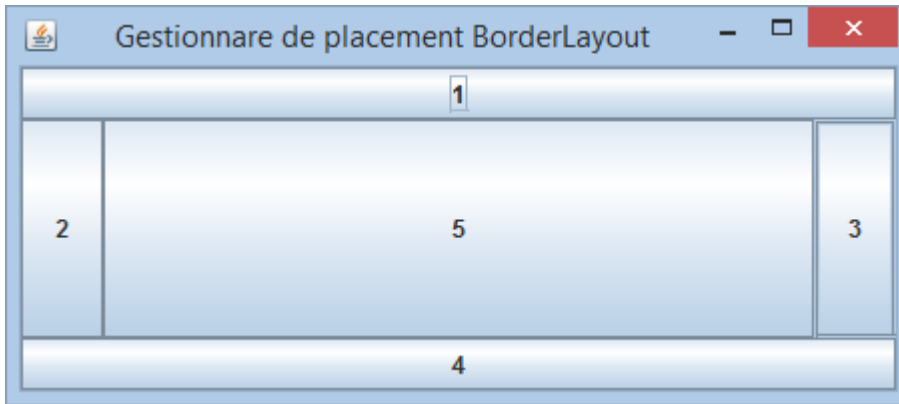
# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

Il permet de placer les composants ainsi : 5 objets maximum : nord, sud, est, ouest, centre



Code de la fenêtre :

```
package pack_20_Fenetres;
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.HeadlessException;
import javax.swing.*;

public class Win_50_fenêtre_GestionnairesP_BorderLayout extends JFrame{
    /* =====
       SWING :      Gestionnaires de placement BorderLayout
       =====
    */
    //
    // Attributs de la fenêtre
    //
    JPanel panneau;
    JButton btn1,btn2,btn3,btn4,btn5,btn6;
    //
    // constructeur
    //
    public Win_50_fenêtre_GestionnairesP_BorderLayout() throws HeadlessException {
        super();
        //
        panneau=new JPanel();
        btn1=new JButton("1");
        btn2=new JButton("2");
        btn3=new JButton("3");
        btn4=new JButton("4");
        btn5=new JButton("5");
        btn6=new JButton("6");
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
// *****
// préciser le gestionnaire de placement (layout) : ici GridLayout
// *****
panneau.setLayout(new BorderLayout());
// *****
// créer les boutons manuellement et les ajouter au panneau principal
// *****
panneau.add("North",btn1); // associer objets au panneau
panneau.add("West",btn2);
panneau.add("East",btn3);
panneau.add("South",btn4);
panneau.add("Center",btn5);
//panneau.add("Center",btn6);
// *****
// associer le panneau à la fenêtre
// *****
this.setContentPane(panneau);
// *****
// régler les propriétés de la fenêtre
// *****
this.setTitle("Gestionnaire de placement GridLayout"); // titre de la fenêtre
this.setSize(400, 200); // largeur et hauteur de la fenêtre
this.setResizable(true); // empêcher le redimensionnement de la fen
this.setLocationRelativeTo(null); //On centre la fenêtre sur l'écran
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
}
}
```

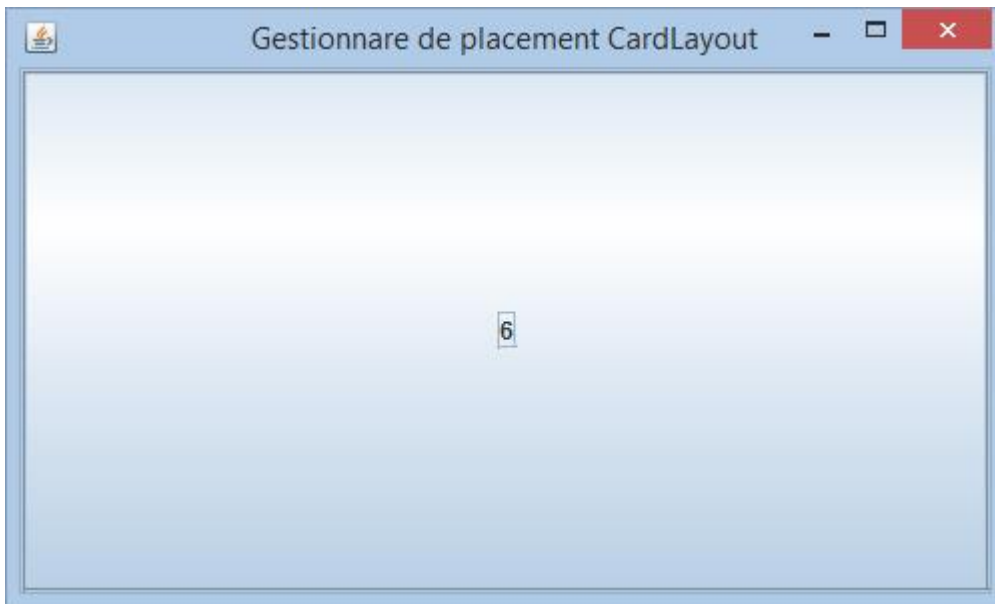
**Cas 4** : utilisation du gestionnaire de placement «**CardLayout**»  
Il place les composants ainsi : les uns au-dessus des autres,  
Il affiche un composant à la fois parmi ceux du panneau comme si on avait un jeu de carte et on choisit la carte visible

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---



Code de la fenêtre :

```
package pack_20_Fenetres;
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.GridLayout;
import java.awt.HeadlessException;
import javax.swing.*;

public class Win_60_fenêtre_GestionnairesP_CardLayout extends JFrame{
    /*
    =====
    SWING :      Gestionnaires de placement CardLayout
    =====
    */
    //
    //  Attributs de la fenêtre
    //
    JPanel panneau;
    JButton btn1,btn2,btn3,btn4,btn5,btn6;
    //
    // constructeur
    //
    public Win_60_fenêtre_GestionnairesP_CardLayout() throws HeadlessException {
        super();
        //
        panneau=new JPanel();
        btn1=new JButton("1");
        btn2=new JButton("2");
        btn3=new JButton("3");
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

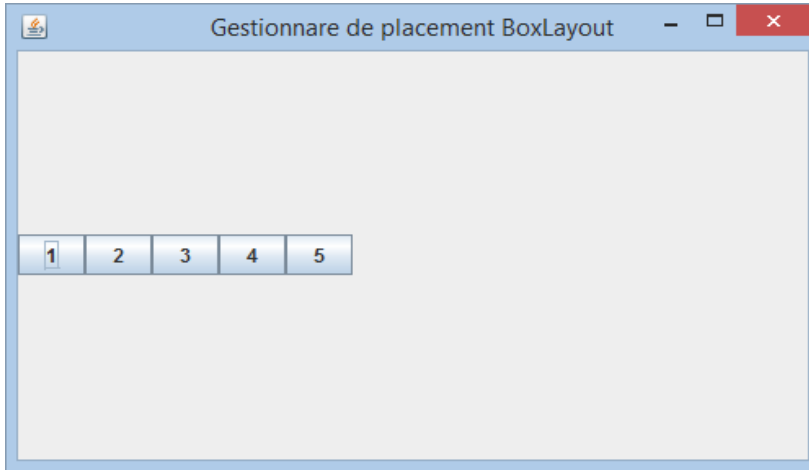
```
btn4=new JButton("4");
btn5=new JButton("5");
btn6=new JButton("6");
//      *****
//      préciser le gestionnaire de placement (layout) : ici CardLayout
//      *****
CardLayout cartes= new CardLayout();
panneau.setLayout(cartes);
//      *****
//      créer les boutons manuellement et les ajouter au panneau principal
//      *****
panneau.add(btn1,"btn1");          // associer objets au panneau
panneau.add(btn2,"btn2");          // le texte est un identifiant du bouton dans le jeu de carte
panneau.add(btn3,"btn3");
panneau.add(btn4,"btn4");
panneau.add(btn5,"btn5");
panneau.add(btn6,"btn6");
cartes.last(panneau);              // sélectionner l'objet à afficher : first, last, previous,
                                   // next, ...;
//      *****
//      associer le panneau à la fenêtre
//      *****
this.setContentPane(panneau);
//      *****
//      régler les propriétés de la fenêtre
//      *****
this.setTitle("Gestionnaire de placement CardLayout"); // titre de la fenêtre
this.setSize(500, 300);              // largeur et hauteur de la fenêtre
this.setResizable(true);              // empêcher le redimensionnement de la fen
this.setLocationRelativeTo(null);      //On centre la fenêtre sur l'écran
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
}
}
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

**Cas 5** : utilisation du gestionnaire de placement «**BoxLayout**»  
Il place les composants ainsi : en lignes ou en colonnes avec dimension demandée.



Code de la fenêtre :

```
package pack_20_Fenetres;
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.HeadlessException;
import javax.swing.*;

public class Win_70_fenêtre_GestionnairesP_BoxLayout extends JFrame{
    /* =====
       SWING :      Gestionnaires de placement BorderLayout
       =====
    */
    //
    // Attributs de la fenêtre
    //
    JPanel panneau;
    JButton btn1,btn2,btn3,btn4,btn5,btn6;
    //
    // constructeur
    //
    public Win_70_fenêtre_GestionnairesP_BoxLayout() throws HeadlessException {
        super();
        //
        panneau=new JPanel();
        btn1=new JButton("1");
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
btn2=new JButton("2");
btn3=new JButton("3");
btn4=new JButton("4");
btn5=new JButton("5");
btn6=new JButton("6");
// *****
// préciser le gestionnaire de placement (layout) : ici BoxLayout
// *****
panneau.setLayout(new BoxLayout(panneau, BoxLayout.X_AXIS));
// *****
// créer les boutons manuellement et les ajouter au panneau principal
// *****
btn1.setPreferredSize(new Dimension(100,100));
btn2.setPreferredSize(new Dimension(50,50));
panneau.add(btn1); // associer objets au panneau
panneau.add(btn2);
panneau.add(btn3);
panneau.add(btn4);
panneau.add(btn5);
//panneau.add(btn6);
// *****
// associer le panneau à la fenêtre
// *****
this.setContentPane(panneau);
// *****
// régler les propriétés de la fenêtre
// *****
this.setTitle("Gestionnaire de placement BoxLayout"); // titre de la fenêtre
this.setSize(500, 300); // largeur et hauteur de la fenêtre
this.setResizable(true); // empêcher le redimensionnement de la fen
this.setLocationRelativeTo(null); //On centre la fenêtre sur l'écran
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
}
}
```

**Autres cas** : voir les classes java : **GridBagLayout** et **SpringLayout**



# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

#### Exemples de composants graphiques

Voir les exercices proposés qui contiennent la présentation de composants SWING : zone de saisie, liste déroulée, liste à dérouler, calendrier, grille de données, cases à cocher, ....

#### **Gestion des actions utilisateur**

Lorsque l'utilisateur clique sur un bouton ou sélectionne dans une liste ou dans une grille ou encore lorsqu'il survole un composant : on dit qu'il interagit avec l'interface graphique,

Java met en place le concept du **Listener** : c'est un code que l'on associe à

- ➔ Un composant donné,
- ➔ Et à une action donnée sur ce composant : clic, sélection survol, .....

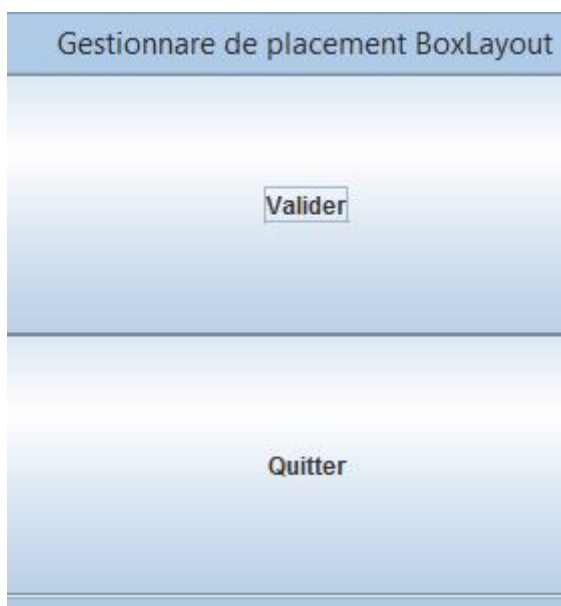
Pour Java, les listeners sont de **différents types** :

- ➔ Listeners d'action : par exemple clic,
- ➔ Listeners de sélection : par exemple choix dans une listBox,
- ➔ Listeners de souris : survol de la souris, clic droit de la souris, ....

#### Listeners internes

Un Listener est dit interne, lorsque le code du listener se trouve dans le même code (même méthode) que le code qui crée le composant.

Exemple de listener interne : on affiche une fenêtre avec deux boutons : celui du haut est sensible au clic de la souris, L'autre est sensible au survol de la souris.

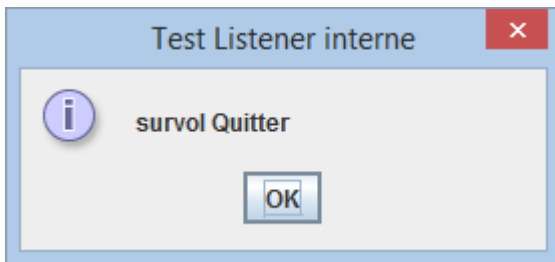
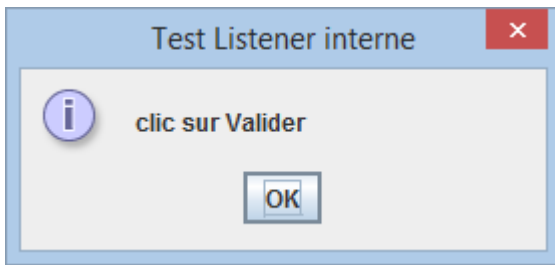


# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---



Voici le code de la fenêtre avec en rouge, le code qui implémente le listener :

```
package pack_20_Fenetres
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.HeadlessException;
import java.awt.event.MouseEvent;
import javax.swing.*;

public class Win_080_fenêtre_ListenerInterne_Clic extends JFrame{
    /* =====
       SWING :      Listener interne sur clic d'un bouton
       =====
    */
    //
    //  Attributs de la fenêtre
    //
    JPanel panneau;
    JButton btnVal,btnQuit;
    //
    // constructeur
    //
    public Win_080_fenêtre_ListenerInterne_Clic() throws HeadlessException {
        super();
        //
        panneau=new JPanel(new GridLayout(2,1));
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
btnVal=new JButton("Valider");
btnQuit=new JButton("Quitter");
//
// listener interne : clic sur un bouton
//
btnVal.addActionListener(new java.awt.event.ActionListener() {
    // événement géré ici clic gauche=actionPerformed
    public void actionPerformed(java.awt.event.ActionEvent e) {
        // afficher une boîte de dialogue
        JOptionPane.showMessageDialog(null,"clic sur Valider"
            ,"Test Listener interne",JOptionPane.INFORMATION_MESSAGE);
    }
});
//
// listener interne : survol de la souris
//
btnQuit.addMouseListener(new java.awt.event.MouseAdapter() {
    // événement géré ici clic gauche=actionPerformed
    public void mouseEntered(java.awt.event.MouseEvent e) {
        // afficher une boîte de dialogue
        JOptionPane.showMessageDialog(null,"survol Quitter"
            ,"Test Listener interne",JOptionPane.INFORMATION_MESSAGE);
    }
});
panneau.add(btnVal);
panneau.add(btnQuit);
// *****
// associer le panneau à la fenêtre
// *****
this.setContentPane(panneau);
// *****
// régler les propriétés de la fenêtre
// *****
this.setTitle("Gestionnaire de placement BorderLayout"); // titre de la fenêtre
this.setSize(500, 300); // largeur et hauteur de la fenêtre
this.setResizable(true); // empêcher le redimensionnement de la fen
this.setLocationRelativeTo(null); //On centre la fenêtre sur l'écran
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
}
}
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

Lorsqu'on cherche à créer un listener, on doit répondre à deux questions :

- ➔ On écoute quel composant (listener pour **QUI**)
- ➔ On écoute quel événement du composant (on écoute **QUOI**) ?

Dans le code ci-dessus :

| Composant                     | Événement à écouter | Type listener                                     | Méthode pour l'événement    |
|-------------------------------|---------------------|---|-----------------------------|
| Un bouton à cliquer (JBUTTON) | Clic gauche         | Listener d'action : <b>addActionListener (..)</b> | <b>actionPerformed (..)</b> |
| Un bouton à cliquer (JBUTTON) | Survol de la souris | Listener de souris : <b>addMouseListener (..)</b> | <b>mouseEntered (..)</b>    |

#### Un listener interne commun à plusieurs composant

Parfois, dans une même fenêtre, plusieurs composants requièrent un listener.

Par exemple, dans un exercice sur le loto : on doit gérer le clic sur 51 boutons : il serait éreintant de créer, avec la syntaxe utilisée ci-dessus, 51 listeners.

Une autre alternative à cela est :

- ➔ On crée des méthodes d'écoute des événements au niveau de la fenêtre et non pas au niveau de chaque composant (méthodes «**actionPerformerd**», «**mouseEntered**», .... Codées à la fin de la classe de la fenêtre,
- ➔ Lorsqu'on ajoute un listener à un composant, on fait référence au listener de la fenêtre :  
Par exemple :        btn1.addActionListener (**this**) ;

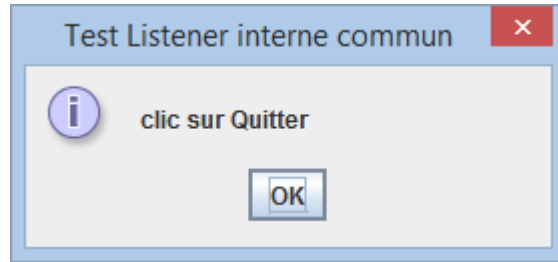
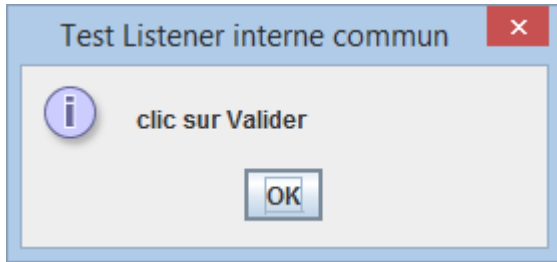
**Reprenons l'exemple vu plus haut** : fenêtre avec deux boutons et gérons le clic gauche sur chacun des boutons avec une seule méthode «**actionPerformed**»



# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche



Code :

```
package pack_20_Fenetres;
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.HeadlessException;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.*;

public class Win_085_fenêtre_ListenerInterne_Commune extends JFrame
    implements ActionListener, MouseListener {

    /*
     * =====
     * SWING :      Listener interne sur clic d'un bouton
     * =====
     */
    //
    // Attributs de la fenêtre
    //
    JPanel panneau;
    JButton btnVal, btnQuit;
    //
    // constructeur
    //
    public Win_085_fenêtre_ListenerInterne_Commune() throws HeadlessException {
        super();
        //
        panneau = new JPanel(new GridLayout(2, 1));
        btnVal = new JButton("Valider");
        btnQuit = new JButton("Quitter");
        //
        // listener interne commun : les événements à intercepter sont
        // définis à la fin de cette classe
    }
}
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
//
btnVal.addActionListener(this);
//btnQuit.addMouseListener(this);
btnQuit.addActionListener(this);
panneau.add(btnVal);
panneau.add(btnQuit);
// *****
//      associer le panneau à la fenêtre
//      *****
this.setContentPane(panneau);
// *****
//      régler les propriétés de la fenêtre
//      *****
this.setTitle("Gestionnaire de placement BorderLayout"); // titre de la fenêtre
this.setSize(500, 300); // largeur et hauteur de la fenêtre
this.setResizable(true); // empêcher le redimensionnement de la fen
this.setLocationRelativeTo(null); //On centre la fenêtre sur l'écran
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // si clique sur croix rouge
} // fin constructeur
//
// gestion clic sur un bouton
//
@Override
public void actionPerformed(ActionEvent arg0) {
    // je récupère le bouton sur lequel on a cliqué
    JButton bt=(JButton)(arg0.getSource());
    if(bt.getText().equals("Valider")){
        JOptionPane.showMessageDialog(null,"clic sur Valider"
            ,"Test Listener interne commun",JOptionPane.INFORMATION_MESSAGE);
    }
    if(bt.getText().equals("Quitter")){
        JOptionPane.showMessageDialog(null,"clic sur Quitter"
            ,"Test Listener interne commub",JOptionPane.INFORMATION_MESSAGE);
    }
}
//
// gestion survol de la souris
//
@Override
public void mouseEntered(MouseEvent e) {
    // afficher une boîte de dialogue
    // JOptionPane.showMessageDialog(null,"survol Quitter"
    //      ,"Test Listener interne",JOptionPane.INFORMATION_MESSAGE);
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
}

@Override
public void mouseClicked(MouseEvent arg0) {
    // TODO Auto-generated method stub

}
@Override
public void mouseExited(MouseEvent arg0) {
    // TODO Auto-generated method stub

}
@Override
public void mousePressed(MouseEvent arg0) {
    // TODO Auto-generated method stub

}
@Override
public void mouseReleased(MouseEvent arg0) {
    // TODO Auto-generated method stub

}
}
```

#### Listeners externes

Maintenant, on aimerait aller plus loin avec les listeners : on veut gérer les actions utilisateurs sur plusieurs fenêtres et pas seulement une seule.

On va décider de centraliser la gestion des événements dans une classe extérieure à la classe JFrame de chaque fenêtre.

On dit que :

- ➔ On confier la gestion des actions à une action externe,
- ➔ On peut dire aussi que l'on crée des listeners externes.

**Notre exemple** : on gère le clic sur les deux boutons « valider » et « Quitter » :

- ➔ On crée une classe nommée « **ActionsValidation** » qui hérite d'une classe java « **AbstractAction** »,
  - On dote cette classe de deux attributs :
    - Un objet de type **JFrame** : on saura la fenêtre d'où vient le clic,
    - Une chaîne : on saura qu'il est le texte du bouton d'où vient le clic (on aurait pu recevoir un objet composant dans une version sophistiquée)
  - On dote cette classe d'un **constructeur** qui reçoit la fenêtre et le bouton à l'origine de l'événement
  - On dote cette classe des méthodes événements à gérer (ici le clic : méthode « **actionPerformed** »)



# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

Code de cette classe de listener externe :

```
package pack_20_Fenetres;
import java.awt.event.ActionEvent;
import javax.swing.AbstractAction;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

/*
 * =====
 * Classe qui gère les actions utilisateur à
 * l'extérieur d'une classe JFrame
 * =====
 * Dans la classe de la fenêtre JFrame :
 * un bouton, par exemple, donne à son constructue
 * // un objet de cette classe ActionsValidation
 */
public class ActionsValidation extends AbstractAction{
    // attributs
    private JFrame fen;
    private String texte;
    // constructeur
    public ActionsValidation(JFrame fenAppelante, String txtComposant) {
        // le texte à mettre sur le bouton est passé à la classe mère
        // AbstractAction
        super(txtComposant);
        this.fen = fenAppelante;
        this.texte = txtComposant;
    }
    //
    // gestion du clic ou sélection d'un composant
    @Override
    public void actionPerformed(ActionEvent arg0) {
        // message sur boîte de dialogue
        JOptionPane.showMessageDialog(null,
            "Le clic vient de la fenêtre "+fen.getClass().getName()+
            " bouton : "+texte,
            "Listener Externe",
            JOptionPane.OK_OPTION);
    }
}
```

➔ Dans le code de la fenêtre qui contient les boutons : on crée les boutons de cette façon :

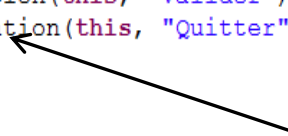
# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
30 //btnVal=new JButton("Valider");
31 //btnQuit=new JButton("Quitter");
32 //
33 // listener externe : grâce à une classe externe
34 //
35 btnVal=new JButton(new ActionsValidation(this, "Valider"));
36 btnQuit=new JButton(new ActionsValidation(this, "Quitter"));
37
38
39 panneau.add(btnVal);
40 panneau.add(btnQuit);
```



## Gestion de Threads pour SWING

### Thread ?

Plusieurs programmes sont présents sur nos ordinateurs.

Sur une machine à un seul processeur, on ne peut pas exécuter plusieurs programmes en même temps.

Si plusieurs programmes sont lancés en mémoire :

- ➔ Le processeur n'exécute pas un programme du début à la fin mais consacre un laps de temps à chacun programme présent en mémoire,
- ➔ Une place de la RAM est allouée à chacun des programmes exécutés,
- ➔ Chaque programme accède à sa propre portion de mémoire,
- ➔ Une partie de la RAM peut être, sous contrôle, partagée par des programmes différents.

Parfois, il y a un **ralentissement** de **traitements** et cela va jusqu'à être ressenti par l'utilisateur final : c'est souvent le cas pour l'affichage d'interfaces graphiques, des traitements de la vidéo.

Une solution pour éviter les ralentissements est de permettre l'**exécution en parallèle** de plusieurs traitements sur un même processeur (l'idéal est d'avoir plusieurs processeurs sur une machine),

**Un thread est donc un environnement d'exécution de code doté de sa propre portion de mémoire RAM et s'exécutant en parallèle d'autres threads.**

Quand vous codez dans la méthode **main ()** et vous exécutez ce code : l'exécution se fait dans un thread par défaut nommé **«main»** et vous ne faites rien pour le créer.

Une classe nommée «Thread» est fournie par Java. On peut l'utiliser pour exécuter un code en parallèle du code courant.

Voici comment l'utiliser :

- ➔ Créez une classe (nommant là par exemple **«ThreadSecondaire»** qui hérite de **«Thread»**,
- ➔ Dotez là de la méthode **«run»** : c'est obligatoire. Mettez dans cette méthode le code qui doit s'exécuter en parallèle.

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

➔ Dans le code principal (par exemple dans la méthode «**main**» :

- Créez une instance (un objet) à partir de la classe **ThreadSecondaire**,
- Appelez la méthode « start » de cette instance : la méthode «run» est appelée et son code exécuté automatiquement.
- 

**Exemple** :

Dans le thread principal (méthode main), on crée un 2<sup>ème</sup> thread : les deux threads « **principal** » et « **secondaire** » affichent un message sur la console.

Code de la classe Thread :

```
package pack_25_Threads;
// classe Thread : hérite de Thred
public class Thread1 extends Thread {
    // cette méthode s'exécute automatiquement dès que :
    // 1) une instance a été créée à partir de cette classe
    // 2) et on invoque la méthode start pour l'instance
    public void run() {
        long start = System.currentTimeMillis();
        //System.out.println("temps avant boucle : "+start);
        // boucle tant que la durée de vie du Thread est < à 5 secondes
        while( System.currentTimeMillis() < ( start + (1000 * 5))) {
            //System.out.println("temps pendant boucle : "+System.currentTimeMillis());
            // traitement du thread1
            System.out.println("Ligne affichée par le thread1");

            try {
                // pause du code du thread1
                Thread.sleep(700);
            }
            catch (InterruptedException ex) {}
        }
        // fin run
    }

    // on peut override autres méthodes
} // fin classe
```

Le code la méthode main qui teste les deux threads :

```
package pack_25_Threads;
public class Thr_00_TestThreads {
    /* =====
    * Gestion des Threads
    * =====
    * test
    */
}
```

# Concepteur Développeur Informatique

## Module : Programmation Java

### Interface graphique : Client Riche

---

```
*/
public static void main(String[] args) {
    // =====
    //      Test 1 : thread principal (méthode main) qui affiche un texte
    //      et un 2d thread qui affiche un texte
    //      ATTENTION : le thread principal doit faire une pause pour
    //      que l'on est le temps de voir le 2ème thread
    //      =====
    // nous sommes dans le thread principal du main()
    //
    // création d'un autre thread pour lancer un traitement parallèle
    // à partir d'une classe Thread1

    Thread1 thread = new Thread1();
    // Activation du thread qui vient d'être créé
    thread.start();           // le code de la méthode run() s'exécute en //
    // tant que le thread est en vie...
    int i=0;
    while( thread.isAlive() ) {
        i++;
        // traitement du thread principal
        System.out.println("Traitement du main() : "+i);
        try {
            // et faire une pause dans le thread principal : 1/2 seconde
            Thread.sleep(700);

        }
        catch (InterruptedException ex) {
            // code s'il y a plantage
        }
    } // fin while
}
```

#### Remarque :

Lorsque certaines fenêtres fabriquées en SWING sont très riches en composants : on constate un net ralentissement à l'affichage (effet cascade) et cela peut aller jusqu'à ralentir l'ordinateur en général.

Pensez dans ce cas, à créer et à lancer la fenêtre SWING dans un thread à part du thread principal.

