

INF3034 : Programmation Orientée Objet

Interfaces graphiques

A. Gademer *

2009

Objectifs

L'objectif de cette fiche est de vous présenter les classes utilisées pour faire des interfaces graphiques.

Elle est là pour vous donner les pistes à suivre lorsque vous abordez un problème. La seule documentation exhaustive à laquelle vous devez vous référer reste l'API Java 1.5 disponible à l'url <http://java.sun.com/j2se/1.5.0/docs/api/>.

1 `java.awt` versus `javax.swing`

java.awt fut le premier paquet proposé pour faire des interfaces graphiques en Java. Il est l'ossature de toutes les interfaces.

javax.swing fut proposé a posteriori pour faciliter la création d'interfaces graphiques et permettre un rendu équivalent sur toutes les plateformes (Unix, Windows et Mac).

Aujourd'hui on utilise généralement un mixage des deux bibliothèques pour faire nos interfaces, leur richesse étant telle que je ne présenterai ici qu'une infime partie de ce qui est possible.

2 Essentiel : Le principe des poupées russes

Ce qu'il faut absolument retenir à propos des interfaces graphiques tient en une ligne :

Chaque composant contient un certain nombre d'autre composant.

Le premier, contenant tous les autres étant appelé racine.

C'est le principe des poupées russes. (cf. Fig 1)

*gademer@esiea.fr

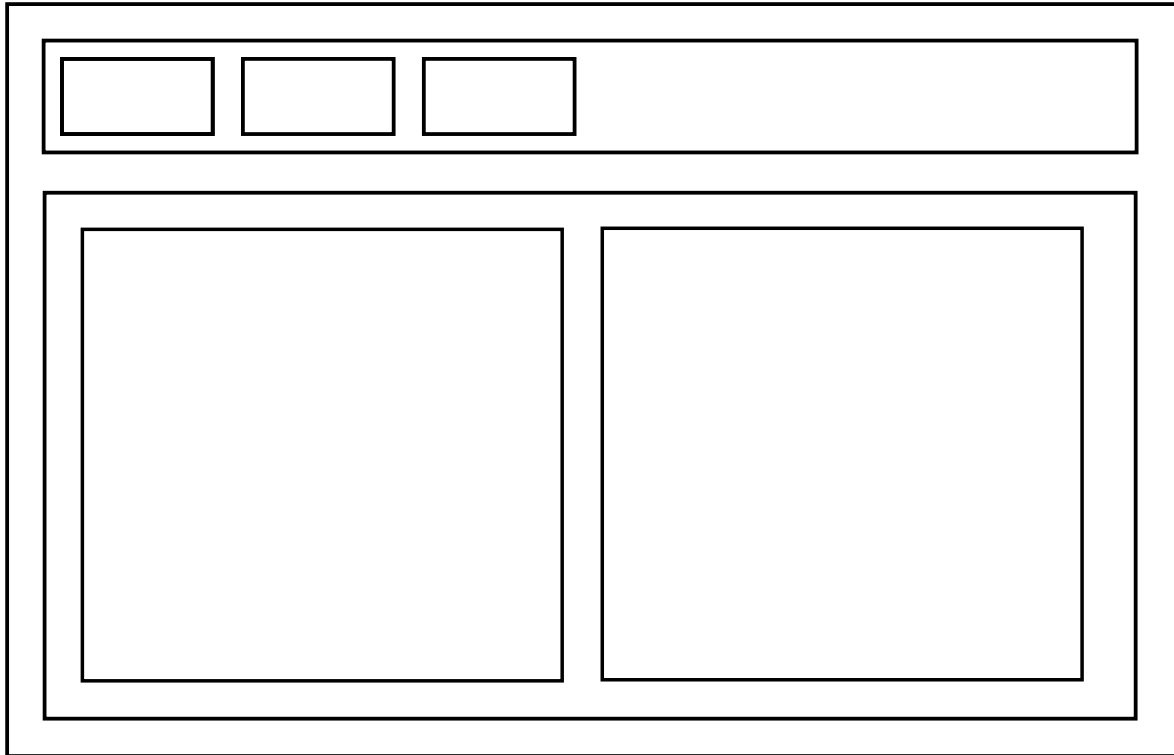


FIG. 1 – Imbrication des composants

3 À la racine : la fenêtre JFrame (javax.swing)

Généralement, le composant de plus haut niveau sera la fenêtre que l'on désire afficher. Une JFrame définit l'objet fenêtre et toutes ses propriétés.

3.1 Instanciation

On instancie généralement une JFrame de la manière suivante :

```
// Instanciation avec le nom de la fenetre
JFrame fen = new JFrame("Ma premiere interface");

// Definition de la taille par default
fen.setPreferredSize(new Dimension(640,480));

// Comportement de la croix de fermeture
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Calcul de la taille de la fenetre en fonction des composants
// qu'elle contient (ici, aucun)
fen.pack();

// Affichage de la fenetre
```

```
fen.setVisible(true);
```

Ce qui donne le résultat de la figure 2.

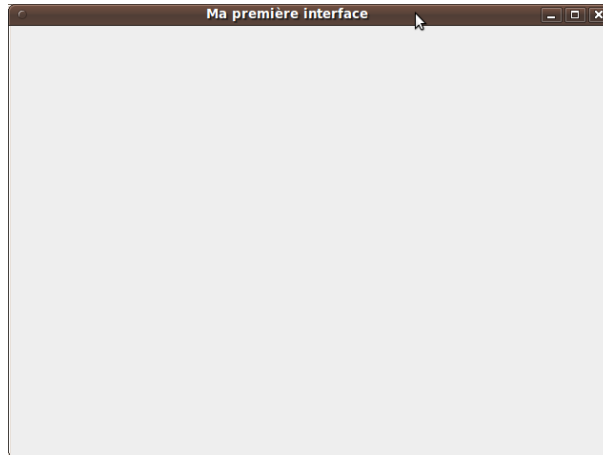


FIG. 2 – Une fenêtre toute simple

Si l'on veut cacher la fenêtre, on utilise encore la méthode `setVisible(boolean isVisible)` mais en passant `false`.

Remarque : On utilise ici la méthode `setDefaultCloseOperation(int operation)` pour choisir le comportement de la croix de fermeture de la fenêtre.

Dans le cas général où la fenêtre correspond à une application on utilise la constante `JFrame.EXIT_ON_CLOSE` qui termine le programme si l'on quitte la fenêtre.

La fenêtre est un conteneur, on va pouvoir ajouter des composants à notre fenêtre avec la méthode `add(Component comp)` ou `add(Component comp, int index)`, avant l'appel à la méthode `pack()`.

```
fen.add(new JButton("one"));
```

3.2 Agencement

Chaque conteneur se voit associé un gestionnaire d'agencement (`LayoutManager`). C'est ce `LayoutManager` qui détermine comment les composants fils sont placés à l'intérieur de celui-ci.

On ajoute un `LayoutManager` à une fenêtre avec la méthode `setLayout(LayoutManager mgr)`.

On distingue trois `LayoutManager` simples :

- `FlowLayout`
- `BorderLayout`
- `GridLayout`

3.2.1 FlowLayout

Ce `LayoutManager` est le plus simple des trois, celui qu'on utilise par défaut. Il place les composants les uns à la suite des autres alignés à gauche (`FlowLayout.LEFT`), à droite (`FlowLayout.RIGHT`) ou au centre (`FlowLayout.CENTER`).



FIG. 3 – FlowLayout

Ci-dessous le code de l'exemple affiché :

```
JFrame fen = new JFrame("Ma première interface");
fen.setPreferredSize(new Dimension(640,80));
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

fen.setLayout(new FlowLayout(FlowLayout.CENTER));

fen.add(new JButton("one"));
fen.add(new JButton("two"));
fen.add(new JButton("three"));

fen.pack();

fen.setVisible(true);
```

3.2.2 BorderLayout

Ce `LayoutManager` organise les composants selon les quatre points cardinaux (généralement au plus un dans chaque emplacement). Il est utilisé lorsque l'on a un composant principal (au centre) et des composants périphériques qui se placent autour.

`BorderLayout` s'instancie avec le constructeur par défaut. Mais on utilise les constantes `BorderLayout.NORTH`, `BorderLayout.SOUTH`, `BorderLayout.WEST`, `BorderLayout.EAST` et `BorderLayout.CENTER` comme second argument de la méthode `add(Component comp, int index)`.

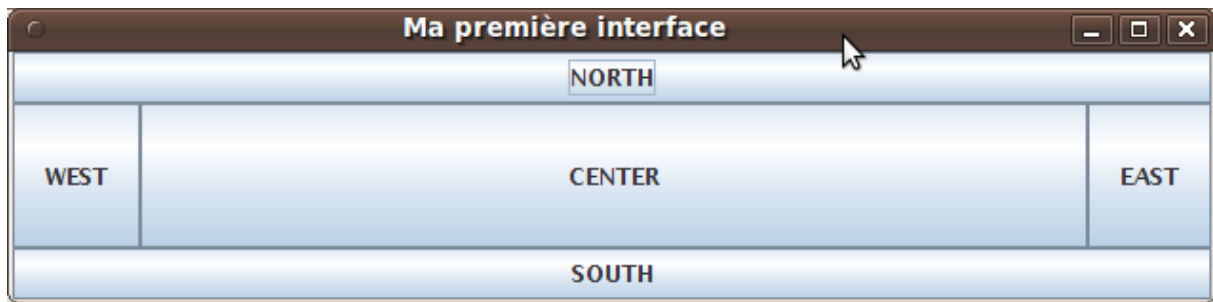


FIG. 4 – BorderLayout

Ci-dessous le code de l'exemple affiché :

```

JFrame fen = new JFrame("Ma première interface");
fen.setPreferredSize(new Dimension(640,160));
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

fen.setLayout(new BorderLayout());

fen.add(new JButton("NORTH"), BorderLayout.NORTH);
fen.add(new JButton("SOUTH"), BorderLayout.SOUTH);
fen.add(new JButton("EAST"), BorderLayout.EAST);
fen.add(new JButton("WEST"), BorderLayout.WEST);
fen.add(new JButton("CENTER"), BorderLayout.CENTER);

fen.pack();

fen.setVisible(true);

```

3.2.3 GridLayout

GridLayout permet d'afficher les composants dans un tableau à deux dimensions. Les composants sont insérés dans l'ordre en remplissant chaque ligne une par une. On l'utilise quand on veut ordonner les choses. On l'instancie avec le nombre de lignes et le nombre de colonnes.

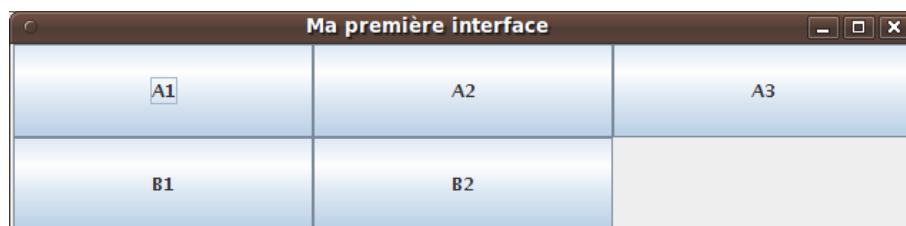


FIG. 5 – GridLayout

Ci-dessous le code de l'exemple affiché :

```
JFrame fen = new JFrame("Ma premiere interface");
fen.setPreferredSize(new Dimension(640,160));
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

fen.setLayout(new GridLayout(2,3));

fen.add(new JButton("A1"));
fen.add(new JButton("A2"));
fen.add(new JButton("A3"));
fen.add(new JButton("B1"));
fen.add(new JButton("B2"));

fen.pack();

fen.setVisible(true);
```

3.2.4 Les trois à la fois

Évidemment on utilise généralement un mélange de ces trois `LayoutManager` et les autres (<http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html>) pour obtenir ce que l'on désire. Mais pour pouvoir mélanger les `LayoutManager` il va nous falloir introduire un autre type de conteneur le `JPanel`

4 Le conteneur à tout faire : JPanel

Le `JPanel` est un conteneur, c'est-à-dire que sa fonction principale est d'être invisible et de contenir d'autres composants. On va utiliser un `JPanel` à chaque fois que l'on désire regrouper des composants. (On verra aussi dans le TP n°5 comment détourner la classe `JPanel` pour afficher le dessin de notre choix.)

On retrouve les méthodes désormais connues :

- `add(Component comp)`,
- `add(Component comp, int index)`,
- `setPreferredSize(Dimension dim)`,
- `setVisible(boolean isVisible)`
- et `setLayout(LayoutManager mgr)`.

Remarque : On peut spécifier le `LayoutManager` d'un `JPanel` au moment de l'instanciation :

```
JPanel topPanel = new JPanel(new FlowLayout());
```

Ci-dessous le code de l'exemple affiché :



FIG. 6 – Utilisation des JPanel

```
JFrame fen = new JFrame("Ma première interface");
fen.setPreferredSize(new Dimension(640,160));
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

fen.setLayout(new BorderLayout());

    JPanel topPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

    topPanel.add(new JButton("1"));
    topPanel.add(new JButton("2"));
    topPanel.add(new JButton("3"));

fen.add(topPanel, BorderLayout.NORTH);

    JPanel centerPanel = new JPanel(new GridLayout(2,2));

    centerPanel.add(new JButton("A"));
    centerPanel.add(new JButton("B"));
    centerPanel.add(new JButton("C"));
    centerPanel.add(new JButton("D"));

fen.add(centerPanel, BorderLayout.CENTER);

fen.pack();

fen.setVisible(true);
```

5 Afficher et saisir du texte : JLabel, JTextField, JTextArea

Maintenant que l'on sait placer nos composants nous allons nous intéresser aux fonctions des composants les plus courants, en commençant par ceux qui manipulent du texte.

Tout ces composants on en commun les méthode `getText()` et `setText(String str)` et un constructeur qui prend le texte par défaut.

Les JLabel servent à afficher des petits bouts de texte explicatifs.

```
JLabel label = new JLabel("Coucou !");
```

Les `JTextField` correspondent aux champs texte des formulaires. C'est dans ces composants que l'on écrit et que l'on saisit du texte.

On notera la méthode `setEditable(boolean isEditable)` qui définit si le champ peut être modifié ou non.

Enfin, les `JTextArea` correspondent aux champs de texte plus grand, sur plusieurs lignes, lorsque la saisie est plus longue.

On a encore la méthode `setEditable(boolean isEditable)` qui définit si le champ peut être modifié ou non.

Exemple :



FIG. 7 – Utilisation des `JLabel`, `JTextField`, `JTextArea`

Ci-dessous le code de l'exemple affiché :

```
JFrame fen = new JFrame("Ma première interface");
fen.setPreferredSize(new Dimension(640,320));
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

fen.setLayout(new GridLayout(4,2));

fen.add(new JLabel("JLabel :"));
fen.add(new JLabel("Mon texte a moi"));
fen.add(new JLabel("JTextField editable :"));
fen.add(new JTextField("Votre texte ici"));
fen.add(new JLabel("JTextField pas editable :"));
JTextField txtfd = new JTextField("Tout vas bien.");
txtfd.setEditable(false);
fen.add(txtfd);
fen.add(new JLabel("JTextArea :"));
fen.add(new JTextArea("Souvent, pour s'amuser, [...] ));

fen.pack();

fen.setVisible(true);
```

6 Passons à l'action : JButton

Base de toute interface graphique qui se respecte : le `JButton` permet, comme son nom l'indique d'afficher un bouton cliquable. On s'en sert pour déclencher des actions du programme. En lui-même le bouton ne fait rien mais peut être associé à un gestionnaire d'événement (`ActionListener`) qui va attendre une action de la part du bouton et qui va la traiter quand celle-là arrivera.

Un bouton est généralement instancié par le texte qui figure dessus.

```
JButton but = new JButton("Stop");
```

On lui associe ensuite un mot-clef qui va permettre de le distinguer des éventuels autres boutons avec la méthode `setActionCommand(String actionCmd)`.

```
but.setActionCommand("stopCmd");
```

Nous allons voir la suite dans la prochaine section consacrée aux gestionnaires d'événements.

7 Gestionnaires d'événements : ActionListener

Les gestionnaires d'événements sont des **interfaces** qui sont associée à des actionneurs (bouton, boîte déroulante, clavier, souris...). Ces interfaces définissent les méthodes qui seront appelée en cas d'activation de l'événement.

Généralement on implémente ces interfaces dans la classe Programme elle-même ce qui permet de définir simplement les actions à faire.

```
public class Programme implements ActionListener {  
  
    public static void main(String[] args) {  
  
        [...]  
  
        Programme p = new Programme();  
  
        JButton but = new JButton("Exploser");  
  
        but.setActionCommand("explode");  
  
        but.addActionListener(p);  
  
        [...]  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        if ( e.getActionCommand().equals("explode") )  
            System.out.println("Boom !");  
    }  
}
```

```
}  
  
}
```

7.1 ActionListener

C'est le gestionnaire d'événement le plus classique. Le composant qui s'associera à lui possédera un mot-clef unique permettant de le reconnaître et provoquera l'événement à chaque fois que l'utilisateur fera une action (cliquer sur un bouton par exemple).

La méthode associée est :

```
public void actionPerformed(ActionEvent e) { }
```

On récupère le mot-clef grâce à la méthode `getActionCommand()` de l'objet `ActionEvent` passé en argument.

7.2 MouseListener, MouseMotionListener, KeyListener

Associé à un `JPanel` ou à une `JFrame`, ces trois interfaces permettent de récupérer les événements provoqués par les périphériques de saisie (souris, clavier).

On utilise `MouseListener` pour agir lors des clics souris, `MouseMotionListener` pour agir lors des déplacements souris et `KeyListener` pour agir lors de l'appui des touches. (Attention aux interférences avec la saisie clavier dans un `JTextField`).

8 Tout le reste

Les cases à cocher, les listes déroulantes, les menus... tout cela reste à découvrir !

Si vous êtes curieux : <http://java.sun.com/docs/books/tutorial/ui/features/compWin.html>.