



ALG2ir-TD02 : Applications à interface graphique : Swing (Aspects statiques)

1 Présentation du TD

Le but de ce TD est de vous montrer comment écrire une application à interface graphique simple à l'aide d'un éditeur, également simple². Nous n'utiliserons donc *pas* toutes les facilités fournies par NetBeans!

Nous nous concentrerons ici sur les aspects statiques de mise en page. Actuellement, deux bibliothèques graphiques sont fournies par SUN dans son kit de développement Java³ : AWT et Swing. C'est cette dernière qui est présentée ici. Nous attaquerons les aspects dynamiques (programmation événementielle) dans le prochain TD.

Pour conclure cette présentation, cassons une idée (peut-être) reçue : *Non!* Un outil graphique de développement n'est pas nécessaire pour développer une application à interface graphique en Java. Derrière un tel outil se cache un générateur de code. Il est donc possible d'obtenir exactement le même résultat en tapant directement le code.

2 Présentation des bibliothèques graphiques en Java

2.1 AWT (Abstract Window Toolkit)

C'est l'API⁴ des débuts, créée par SUN. Elle offre tout ce qu'il faut pour développer des applications graphiques.

2.2 Swing

Deuxième réponse de SUN au problème du développement d'interfaces graphiques. Swing possède des composants plus riches et offre des composants légers, au contraire de AWT où tous les composants sont lourds (voir le cadre ci-après).

¹ Le texte de ce TD s'inspire très largement de ceux des Ateliers Logiciels Java (2^e Gestion et 3^e Industrielle et Réseaux) créés principalement par MCD et VAK, mais auxquels ont également collaboré RFS et SMB (et peut-être d'autres que j'ignore, qu'ils me pardonnent).

² En fait, nous utiliserons NetBeans mais pas son interface graphique de développement (le *NetBeans Swing GUI Builder*), uniquement son éditeur.

³ JDK, Java development kit.

⁴ Application Programming Interface ou « interface de programmation » en français : interface effectivement implémentée par les bibliothèques du JDK de SUN, par exemple.

Swing est, à l'heure actuelle, la librairie graphique Java la plus utilisée.

Composant lourd *vs* composant léger

On parle de *composant lourd* lorsque l'API du système hôte est utilisée pour le créer. C'est le cas des composants AWT. Ainsi, lorsqu'on crée un bouton AWT, un appel à la librairie graphique du système est réalisé pour le créer.

Un *composant léger*, comme ceux de Swing, est créé de toute pièce par l'API Java, *sans* faire appel à son correspondant du système hôte. Ce fut un plus gros travail pour les concepteurs de l'API Swing mais cela permet d'avoir des applications qui ont quasiment le même *look* quel que soit le système d'exploitation sur lequel elles tournent. Cette solution est toutefois généralement plus lente que celle utilisant l'API native.

2.3 SWT (Standard Widget Toolkit)

SWT⁵ est une bibliothèque graphique libre initiée par IBM et actuellement développée par la Fondation Eclipse. Elle n'est pas reconnue comme standard et constitue donc une alternative aux bibliothèques graphiques standard fournies par SUN. Elle utilise les librairies du système hôte pour dessiner ses composants graphiques.

3 Composants et conteneurs

Voyons brièvement les bases du développement d'applications graphiques sous Swing. Les notions introduites ci-dessous se retrouvent dans les autres API.

D'abord, faisons la distinction entre les composants et les conteneurs.

Les **composants** sont des éléments graphiques réutilisables. On peut les configurer et interagir avec eux. Parmi eux, on trouve les boutons, menus, champs de saisie, etc.

Signalons tout de suite qu'il est fortement déconseillé de mélanger l'utilisation de composants Swing et AWT. Notez que cette remarque ne porte que sur les *composants*. Pour ce qui concerne la gestion de la mise en page (voir la section 4) ou encore pour la gestion d'événements⁶, par exemple, l'utilisation de classes AWT et de composants Swing s'avère le plus souvent obligatoire!

Les **conteneurs**, comme leur nom l'indique, sont des éléments pouvant contenir des composants.

⁵ Page maison : <http://www.eclipse.org/swt/>.

⁶ Les aspects dynamiques de la programmation d'applications à interface graphique en Java font l'objet du prochain TD.

La grande puissance de l'architecture mise en œuvre est de poser qu'*un conteneur est un composant*. On obtient ainsi la possibilité d'imbriquer les conteneurs comme une poupée gigogne pour obtenir l'aspect graphique désiré.

Que trouve-t-on comme conteneur ?

La `JFrame` (fenêtre) est un conteneur évident et incontournable. Visuellement, elle apparaît comme une fenêtre : un rectangle avec une barre de titre et des boutons de manipulation de la fenêtre (fermer, maximiser, etc.). Elle contient des composants.

La `JDialog` (boîte de dialogue) est une fenêtre qui est liée à une autre. Elle peut être ouverte (créée) en mode modal ou non.

Fenêtre modale ou non modale

Lorsqu'une fenêtre *A* ouvre une fenêtre *B* en mode « modal », *A* n'est plus accessible tant que *B* n'est pas fermée. C'est le comportement typique des boîtes de dialogue.

À l'inverse, si *A* ouvre *B* en mode « non modal », les deux fenêtres sont actives et l'utilisateur peut passer de l'une à l'autre à son gré.

Le `JPanel` se rencontre également très souvent. Invisible, il contient d'autres composants. Il est utilisé pour la mise en page mais également lors du développement de nouveaux composants.

Exemple 1 : Écrivons une application qui affiche une fenêtre tout à fait vide.

```

1 package nvs.alg2ir.td02swingstatique.enonce;
2
3 import javax.swing.JFrame ;
4
5 /**
6  * Application qui affiche une fenêtre tout à fait vide.
7  */
8 public class AfficheFenetreVide extends JFrame {
9
10    public static void main(String [] args) {
11        AfficheFenetreVide f = new AfficheFenetreVide ();
12        f.setVisible(true);
13    }
14 }
```

Cette solution présente tout dans une seule et même classe. On peut séparer la classe de la fenêtre proprement dite de celle de la méthode de test, ce qui donne :

```

1 package nvs.alg2ir.td02swingstatique.enonce;
2
3 import javax.swing.JFrame ;
4
```

```

5  /**
6   * Classe fournissant une fenêtre complètement vide.
7   */
8  public class FenetreVide extends JFrame {
9
10     public FenetreVide () {
11     }
12 }

```

```

1  package nvs.alg2ir.td02swingstatique.enonce;
2
3  /**
4   * Classe de test de la classe FenetreVide.
5   */
6  public class TestFenetreVide {
7
8     public static void main(String [] args) {
9         FenetreVide f = new FenetreVide ();
10        f.setVisible(true);
11    }
12 }

```

Quelques remarques en vrac :

- le paquetage `swing` se trouve dans le paquetage `javax` où le `x` vient du mot « *extended* » : le paquetage `swing` est une extension au JDK présente par défaut ;
- la taille de la fenêtre doit être réglée : on y vient tout de suite ;
- que se passe-t-il suite à un clic sur l'icône de fermeture de la fenêtre, présente dans sa barre de titre (croix sous WINDOWS) ? Allez voir sous l'onglet `Runtime` de NetBeans ou examinez les processus actifs sur votre machine ! La Javadoc de `JFrame` explique ce comportement. Nous y reviendrons en détail dans le TD suivant.

Exemple 2 : Écrivons une application composée d'un bouton dans une fenêtre. Cliquer sur le bouton n'a pas d'effet pour l'instant. Spécifions un titre pour la fenêtre.

```

1  package nvs.alg2ir.td02swingstatique.enonce;
2
3  import javax.swing.JButton;
4  import javax.swing.JFrame;
5
6  /**
7   * Fenêtre avec un titre et un bouton inactif.
8   */
9  class MaFenetre extends JFrame {
10
11     private JButton bouton;
12
13     MaFenetre () {
14         super("Ma fenêtre"); // construit avec un titre
15         bouton = new JButton("<html><p>OK</p><p>KO</p></html>");
16         //this.getContentPane().add(bouton) ;
17         this.add(bouton) ;
18         // getContentPane() n'est plus nécessaire en 1.5
19     }

```

```

20
21 public static void main(String[] args) {
22     MaFenetre f = new MaFenetre();
23     f.pack(); // pour afficher tous les composants de la fenêtre
24     // Essayer d'enlever le pack(), pour voir
25     f.setVisible(true);
26 }
27 }

```

Notez que les composants (comme ici le `JButton`) ne sont pas ajoutés directement à la `JFrame` mais au conteneur de composants⁷ de celle-ci, son *content pane*, obtenu par la méthode `getContentPane()`. Jusqu'au JDK 1.4, il fallait *explicitement* récupérer le *content pane* de la fenêtre et lui ajouter les composants désirés. À partir de la version 1.5 de Java, cela se passe *implicitement*.

Ex1. Créez une fenêtre avec un texte non éditable (étiquette) et deux boutons ayant pour label « Oui » et « Non ».

Testez le comportement de l'application lorsque l'utilisateur modifie la taille de la fenêtre. Comment les composants sont-ils placés ?

4 Les gestionnaires de mise en page

Lorsqu'on utilise `JBuilder` ou `Borland C++Builder`, on place un composant à un endroit de la fenêtre et il y reste. Ce n'est *a priori* pas le cas ici. Pourquoi ? Un conteneur place ses composants en fonction de son « gestionnaire de mise en page » ou *layout manager* en anglais.

Sous `JBuilder`, lorsqu'on crée une `Frame`, un code est automatiquement inséré pour associer un gestionnaire `XYLayout`⁸ à cette `Frame`. Avec ce gestionnaire, les composants indiquent de manière *absolue* où ils veulent se placer, dans un système de coordonnées (x, y) par rapport au coin supérieur gauche du conteneur. C'est probablement simple et efficace avec un outil de développement graphique mais ce n'est pas propre⁹ voire compliqué lorsque le nombre de composants est grand ou variable.

Par défaut, le gestionnaire de mise en page d'une `JFrame` est le `BorderLayout`.

Passons en revue les principaux gestionnaires de mise en page que l'on rencontre.

Le `BorderLayout` permet de placer jusqu'à cinq composants dans le conteneur qui l'utilise. Il définit cinq zones : le nord, le sud, l'est, l'ouest et le centre. Si aucun emplacement n'est spécifié lors de l'ajout d'un composant, il est placé au centre.

Le `FlowLayout` place ses composants les uns à la suite des autres, horizontalement, en passant à la ligne si nécessaire. Il s'agit du gestionnaire de mise en page par défaut d'un `JPanel`.

⁷ Une `JFrame` est constituée de plusieurs conteneurs, dont le *content pane*. Le conteneur d'un menu (`JMenu`), par exemple, n'est pas le *content pane* de sa `JFrame`.

⁸ Ce gestionnaire de mise en page est propre à `BORLAND`.

⁹ Cela ne permet pas à l'application de s'adapter à des résolutions d'écrans différentes ou à la modification de la taille de ses fenêtres.

Le `GridLayout` place ses composants dans une grille dont la taille (nombre de lignes et de colonnes) peut être spécifiée. Toutes les cases ont la même taille.

Outre la gestion du placement initial des composants, le gestionnaire de mise en page gère le remplacement de ceux-ci lorsque la taille de leur conteneur est modifiée. De plus, la taille des composants dans leur conteneur est, en partie, déterminée par le gestionnaire de mise en page du conteneur.

Signalons enfin la possibilité de ne pas utiliser les services d'un gestionnaire de mise en page, si ce n'est pour le placement initial des composants. Pour cela, il faut spécifier le gestionnaire de mise en page `null`. Celui-ci place les composants de manière absolue, à l'endroit indiqué dans un système de coordonnées centré sur le coin supérieur gauche du conteneur. On retrouve les caractéristiques du `XYLayout` de BORLAND.

Revenons à l'Ex1 dont l'énoncé peut être reformulé comme :

Écrivez une application composée de deux boutons et d'un texte non éditable. Ne spécifiez rien quant à la mise en page.

Que se passe-t-il ?

```

1 package nvs.alg2ir.td02swingstatique.enonce;
2
3 import java.awt.Container;
4 import javax.swing.JButton;
5 import javax.swing.JFrame;
6 import javax.swing.JLabel;
7
8 /**
9  * Fenêtre avec deux boutons et un label : pas de gestionnaire
10  * de mise en page spécifié ni de position lors de l'ajout
11  * des composants.
12  */
13 class MaFenetreSGMP extends JFrame {
14
15     private JButton btOui, btNon;
16     private JLabel lbTexte;
17
18     MaFenetreSGMP () {
19         super("Ma fenêtre");
20         Container content = this.getContentPane(); // ça passe aussi en 1.4
21
22         btOui = new JButton("Oui");
23         btNon = new JButton("Non");
24         lbTexte = new JLabel("Bonjour");
25
26         content.add(btNon);
27         content.add(btOui);
28         content.add(lbTexte);
29
30         this.setDefaultCloseOperation (DISPOSE_ON_CLOSE) ;
31     }
32
33     public static void main(String [] args) {
34         MaFenetreSGMP f = new MaFenetreSGMP ();

```

```

35     f.pack();
36     f.setVisible(true);
37 }
38 }

```

Remarquez l'utilisation de la méthode `setDefaultCloseOperation` de `JFrame`. Plus de détails immédiatement dans la Javadoc ou lors du prochain TD !

Exemple 3 : Reprenons la solution proposée pour l'Ex1, mais utilisons cette fois-ci les possibilités du `BorderLayout` pour la mise en page.

Observez le placement des composants lorsque la taille de la fenêtre est modifiée.

```

1  package nvs.alg2ir.td02swingstatique.enonce;
2
3  import java.awt.BorderLayout;
4  import javax.swing.JButton;
5  import javax.swing.JFrame;
6  import javax.swing.JLabel;
7
8  /**
9   * Fenêtre avec deux boutons et un label : utilisation
10  * du gestionnaire de mise en page par défaut de la JFrame,
11  * à savoir le BorderLayout.
12  */
13  public class MaFenetreGMPDefault extends JFrame {
14
15     private JButton btOui, btNon;
16     private JLabel lbTexte;
17
18     public MaFenetreGMPDefault() {
19         super("Ma fenêtre");
20
21         btOui = new JButton("Oui");
22         btNon = new JButton("Non");
23         lbTexte = new JLabel("Bonjour");
24
25         this.add(lbTexte, BorderLayout.NORTH);
26         add(btOui, BorderLayout.WEST);
27         add(btNon, BorderLayout.EAST);
28
29         setDefaultCloseOperation(DISPOSE_ON_CLOSE);
30     }
31
32     public static void main(String[] args) {
33         MaFenetreGMPDefault f = new MaFenetreGMPDefault();
34         f.pack();
35         f.setVisible(true);
36     }
37 }

```

Exemple 4 : Reprenons la solution proposée pour l'Ex1, mais utilisons cette fois-ci les possibilités du `FlowLayout` pour la mise en page.

À nouveau, observez le placement des composants lorsque la taille de la fenêtre est modifiée.

```

1 package nvs.alg2ir.td02swingstatique.enonce;
2
3 import java.awt.Container;
4 import java.awt.FlowLayout;
5 import javax.swing.JButton;
6 import javax.swing.JFrame;
7 import javax.swing.JLabel;
8
9 /**
10  * Fenêtre avec deux boutons et un label : utilisation
11  * d'un FlowLayout comme gestionnaire de mise en page.
12  */
13 public class MaFenetreFlowlayout extends JFrame {
14
15     private JButton btOui, btNon;
16     private JLabel lbTexte;
17
18     public MaFenetreFlowlayout () {
19         super("Ma fenêtre");
20         Container content = this.getContentPane();
21
22         btOui = new JButton("Oui");
23         btNon = new JButton("Non");
24         lbTexte = new JLabel("Bonjour");
25
26         this.getContentPane().setLayout( new FlowLayout() );// passe en 1.4
27         this.getContentPane().add(lbTexte);
28         content.add(btOui);
29         add(btNon);// passe pas en 1.4
30
31         this.setDefaultCloseOperation (DISPOSE_ON_CLOSE) ;
32     }
33
34     public static void main(String [] args) {
35         MaFenetreFlowlayout f = new MaFenetreFlowlayout ();
36         f.pack ();
37         f.setVisible (true);
38     }
39 }

```

Ex2. Reprenez à nouveau l'énoncé de l'Ex1, mais arrangez-vous cette fois-ci pour placer les composants (le texte et les deux boutons) de sorte que la fenêtre (**JFrame**) ressemble à une boîte de dialogue : le texte au centre, les boutons en bas, la fenêtre non redimensionnable (voir FIG. 1).

Aide : Pensez aux poupées russes...



FIG. 1 – Fausse boîte de dialogue.