

TD2¹

Interface utilisateur graphique (aspects statiques)

1 Préalables

L'objectif de ce TD est de vous familiariser avec la programmation d'interfaces graphiques avec la bibliothèque **Swing**. Dans cette première approche, nous nous intéresserons essentiellement aux aspects statiques d'une interface graphique, nos interfaces ne seront pas capable d'interagir avec l'utilisateur².

Netbeans propose un outil de génération d'interface graphique . . . que nous n'utiliserons pas³ ! En effet, il est tout à fait possible de générer soi-même son interface en écrivant son code, le générateur d'interface graphique n'étant qu'un générateur de code.

Enfin, sachez qu'il existe différentes bibliothèques graphiques en Java

AWT *Abstract Window Toolkit* est l'API⁴ proposée par **Sun**. Elle offre ce qu'il faut comme composants, caractérisés comme **lourds**.

Un composant est dit lourd lorsqu'il fait appel à la bibliothèque du système hôte pour s'afficher.

Swing est la seconde API graphique proposée par Sun, elle propose des composants **légers** et plus nombreux.

Un composant est dit léger lorsqu'il est créé de toute pièce par l'API.

SWT *Standard Widget Toolkit* est une API initiée par Ibm et proposée par **Eclipse**.

Elle n'est pas un standard, simplement une alternative à l'API de Sun.

¹Le texte de ce TD s'inspire très largement de ceux des ateliers logiciels écrits jadis par MCD et VAK auxquels ont collaborés par la suite RFS, SMB et NVS. Je (PBT) laisse mon empreinte et en profite pour remercier tous ceux qui aiment être remerciés.

²Ce sera l'objet du TD3.

³Nous sommes convaincus que le lecteur intéressé ne manquera pas d'essayer par lui-même.

⁴*Application Programming Interface* ou interface de programmation.

2 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,
- ↪ les menus,
- ↪ les menus déroulants,
- ↪ les champs de saisie,
- ↪ etc.

Remarque. Il est fortement déconseillé de mélanger l'utilisation de **composants** Swing et AWT. Par contre, pour ce qui concerne la gestion de la mise en page (voir la section 3) ou encore pour la gestion d'événements⁵, par exemple, l'utilisation de classes AWT et de composants Swing sera le plus souvent obligatoire !

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

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.

⁵ Les aspects dynamiques de la programmation d'applications à interface graphique en Java font l'objet du prochain TD.

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

Écrivons une application qui affiche une fenêtre tout à fait vide.

```
$ cat AfficheFenetreVide.java
```

```
package nvs.alg2ir;

import javax.swing.JFrame ;

/**
 * Application qui affiche une fenêtre tout à fait vide.
 */
public class AfficheFenetreVide extends JFrame {

    public static void main(String[] args) {
        AfficheFenetreVide f = new AfficheFenetreVide();
        f.setSize(420,420);
        f.setVisible(true);
    }
}
```

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 :

```
$ cat FenetreVide.java
```

```
package nvs.alg2ir;

import javax.swing.JFrame ;

/**
 * Classe fournissant une fenêtre complètement vide.
 */
public class FenetreVide extends JFrame {

    public FenetreVide() {
        setSize(420,420);
    }
}
```

```
$ cat TestFenetreVide.java
```

```
package nvs.alg2ir;

/**
 * Classe de test de la classe FenetreVide.
 */
public class TestFenetreVide {

    public static void main(String[] args) {
        FenetreVide f = new FenetreVide();
        f.setVisible(true);
    }
}
```

Remarques

- ↪ 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;
- ↪ que se passe-t-il suite à un clic sur l'icône de fermeture de la fenêtre, présente dans sa barre de titre (généralement une croix à droite)? 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 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.

```
package nvs.alg2ir;

import javax.swing.JButton;
import javax.swing.JFrame;

/**
 * Fenêtre avec un titre et un bouton inactif.
 */
class MaFenetre extends JFrame {

    private JButton bouton;

    MaFenetre() {
        super("Ma_fenêtre"); // construit avec un titre
        bouton = new JButton("<html><p>OK</p><p>KO</p></html>");
        this.add(bouton);
    }

    public static void main(String[] args) {
        MaFenetre f = new MaFenetre();
        f.pack();
        f.setVisible(true);
    }
}
```

Remarques

- ↪ Notez que les composants (comme par exemple un `JButton`) ne sont pas ajoutés directement à une `JFrame` mais au conteneur de composants⁶ 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*.

```
// Extrait de code JDK 1.4  
this.getContentPane().add(bouton)
```

- ↪ La méthode `pack` permet d'ajuster la taille de la fenêtre en fonction des composants qu'elle possède. Observez les différences en fonction de la présence ou non de la méthode `pack` et de la méthode `setSize` (absente ici).

Exercice 1 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 ?

3 Les gestionnaires de mise en page

Certains IDE tels `JBuilder` ou `Borland C++Builder` placent les composants à un endroit de la fenêtre de manière absolue, si le composant est placé à la position (x, y) 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`⁷ 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 propre⁸ voire compliqué lorsque le nombre de composants est grand ou variable.

⁶ 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`.

⁷ Ce gestionnaire de mise en page est propre à `BORLAND`.

⁸ 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.

Par défaut, le gestionnaire de mise en page d'une `JFrame` est le `BorderLayout` mais il en existe d'autres. 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`.
- ↪ 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'exercice 1 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 ?

\$ MaFenetreSGMP

```
package nvs.alg2ir;

import java.awt.Container;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 * Fenêtre avec deux boutons et un label : pas de gestionnaire
 * de mise en page spécifié ni de position lors de l'ajout
 * des composants.
 */
class MaFenetreSGMP extends JFrame {

    private JButton btOui;
    private JButton btNon;
    private JLabel lbTexte;
```

```

MaFenetreSGMP() {
    super ("Ma_fenêtre");

    btOui = new JButton("Oui");
    btNon = new JButton("Non");
    lbTexte = new JLabel("Bonjour");

    this.add(btNon);
    this.add(btOui);
    this.add(lbTexte);
    this.add(btOui);

    this.setDefaultCloseOperation(DISPOSE_ON_CLOSE) ;
}

public static void main(String[] args) {
    MaFenetreSGMP f = new MaFenetreSGMP();
    f.pack();
    f.setVisible(true);
}
}

```

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

Exemple Reprenons la solution proposée pour l'exercice 1, 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.

\$ MaFenetreGMPDefault.java

```

package nvs.alg2ir;

import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 * Fenêtre avec deux boutons et un label : utilisation
 * du gestionnaire de mise en page par défaut de la JFrame,
 * à savoir le BorderLayout.
 */
public class MaFenetreGMPDefault extends JFrame {

    private JButton btOui;
    private JButton btNon;
    private JLabel lbTexte;

    public MaFenetreGMPDefault() {
        super ("Ma_fenêtre");

        btOui = new JButton("Oui");
        btNon = new JButton("Non");
        lbTexte = new JLabel("Bonjour");
    }
}

```

```

    this.add(lbTexte, BorderLayout.NORTH);
    this.add(btOui, BorderLayout.WEST);
    this.add(btNon, BorderLayout.EAST);

    setDefaultCloseOperation(DISPOSE_ON_CLOSE);
}

public static void main(String[] args) {
    MaFenetreGMPDefaut f = new MaFenetreGMPDefaut();
    f.pack();
    f.setVisible(true);
}
}

```

Exemple Reprenons la solution proposée pour l'exercice 1, 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.

\$ MaFenetreFlowLayout.java

```

package nvs.alg2ir;

import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 * Fenêtre avec deux boutons et un label : utilisation
 * d'un FlowLayout comme gestionnaire de mise en page.
 */
public class MaFenetreFlowLayout extends JFrame {

    private JButton btOui;
    private JButton btNon;
    private JLabel lbTexte;

    public MaFenetreFlowLayout() {
        super("Ma_fenêtre");

        btOui = new JButton("Oui");
        btNon = new JButton("Non");
        lbTexte = new JLabel("Bonjour");

        this.setLayout(new FlowLayout());
        this.add(lbTexte);
        this.add(btOui);
        this.add(btNon);

        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }

    public static void main(String[] args) {
        MaFenetreFlowLayout f = new MaFenetreFlowLayout();
        f.pack();
        f.setVisible(true);
    }
}

```

Exercice 2

Reprenez à nouveau l'énoncé de l'exercice 1, 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.

Aide : Pensez aux poupées russes...



Exercice 3

Reprenez l'exercice du *poulailler* du TD1, nous allons y ajouter une ébauche de GUI.

Écrivez une classe `GUIpoulailler` représentant le poulailler. Pour ce faire, vous utiliserez un `GridLayout` (de 20 lignes, 30 colonnes par exemple). Votre *content pane* contient des `JLabel`. En lisant la doc, on voit que l'on peut ajouter une **icône** à ce composant.



Un emplacement du poulailler sera donc soit une icône "vide" soit une icône représentant une poule. Nous vous fournissons deux images pouvant faire l'affaire. Pour convertir votre image (au format PNG dans notre cas) en icône, vous pouvez utiliser la classe `ImageIcon` et écrire.

```
public final ImageIcon MA_POULE =  
    new ImageIcon(getClass().getResource("poule.png"), "Ma_poule");
```

ou à votre meilleure convenance,

```
public final ImageIcon MA_POULE =  
    new ImageIcon("poule.png");
```