Chapitre 5
Application graphique

1. Introduction

Jusqu'à présent, tous les exemples de code que nous avons réalisés fonctionnent exclusivement en mode caractères. Les informations sont affichées dans une console et également saisies à partir de celle-ci. La simplicité de ce mode de fonctionnement est un atout indéniable pour l'apprentissage d'un langage. Cependant, la plupart des utilisateurs de vos futures applications s'attendent certainement à avoir une interface un petit peu moins austère qu'un écran en mode caractères. Nous allons donc étudier dans ce chapitre comment fonctionnent les interfaces graphiques avec Java. Vous allez rapidement vous apercevoir que la conception d'interfaces graphiques en Java n'est pas très simple et nécessite l'écriture de nombreuses lignes de code. Dans la pratique, de nombreux outils de développement sont capables de prendre en charge la génération d'une grande partie de ce code en fonction de la conception graphique de l'application que vous dessinez. Il est cependant important de bien comprendre les principes de fonctionnement de ce code pour éventuellement intervenir dessus et l'optimiser. Nous n'utiliserons pas dans ce chapitre d'outil spécifique.

1.1 Les bibliothèques graphiques

Le langage Java propose deux bibliothèques dédiées à la conception d'interfaces graphiques. La bibliothèque AWT et la bibliothèque SWING. Les principes d'utilisation sont quasiment identiques pour ces deux bibliothèques. L'utilisation simultanée de ces deux bibliothèques dans une même application peut provoquer des problèmes de fonctionnement et doit être évitée. Il existe une troisième bibliothèque, plus récente, nommée Java FX. Elle ne fait cependant plus partie de la plateforme Java SE standard. Ce chapitre n'abordera donc pas cette bibliothèque.

1.1.1 La bibliothèque AWT

Cette bibliothèque est la toute première disponible pour le développement d'interfaces graphiques. Elle contient une multitude de classes et interfaces permettant la définition et la gestion d'interfaces graphiques. Cette bibliothèque utilise en fait les fonctionnalités graphiques du système d'exploitation. Ce n'est donc pas le code présent dans cette bibliothèque qui assure le rendu graphique des différents composants. Ce code sert uniquement d'intermédiaire avec le système d'exploitation. Son utilisation est de ce fait relativement économe en ressources. En revanche, elle souffre de plusieurs inconvénients.

– L'aspect visuel de chaque composant étant lié à la représentation qu'en fait le système d'exploitation, il est parfois délicat de développer une application ayant une apparence cohérente sur tous les systèmes. La taille et la position des différents composants étant les deux éléments principalement affectés par ce problème.
– Pour que cette bibliothèque soit compatible avec tous les systèmes d'exploitation, les composants qu'elle contient sont donc limités aux plus courants (boutons, zone de texte, listes...).

1.1.2 La bibliothèque Swing

Cette bibliothèque a été conçue pour pallier les principales insuffisances de la bibliothèque AWT. Cette amélioration a été obtenue en écrivant entièrement cette bibliothèque en Java sans pratiquement faire appel aux services du système d'exploitation. Seuls quelques éléments graphiques (fenêtres et boîtes de dialogue) sont encore en relation avec le système d'exploitation. Pour les autres composants, c'est le code de la bibliothèque Swing qui détermine entièrement leurs aspects et comportements.

La bibliothèque Swing contient donc une quantité impressionnante de classes servant à redéfinir les composants graphiques. Il ne faut cependant pas penser que la bibliothèque Swing rend complètement obsolète la bibliothèque AWT. Beaucoup d'éléments de la bibliothèque AWT sont d'ailleurs repris dans la bibliothèque Swing. Nous utiliserons principalement cette bibliothèque dans le reste de ce chapitre.

1.2 Constitution de l'interface graphique d'une application

La conception de l'interface graphique d'une application consiste essentiellement à créer des instances des classes représentant les différents éléments nécessaires, modifier les caractéristiques de ces instances de classe, les assembler et prévoir le code de gestion des différents événements pouvant intervenir au cours du fonctionnement de l'application. Une application graphique est donc constituée d'une multitude d'éléments superposés ou imbriqués. Parmi ces éléments, l'un d'entre eux joue un rôle prépondérant dans l'application. Il est souvent appelé conteneur de premier niveau. C'est lui qui va interagir avec le système d'exploitation et contenir tous les autres éléments. En général ce conteneur de premier niveau ne contient pas directement les composants graphiques mais d'autres conteneurs sur lesquels sont placés les composants graphiques. Pour faciliter la disposition de ces éléments les uns par rapport aux autres, nous pouvons utiliser l'aide d'un gestionnaire de mise en page. Cette superposition d'éléments peut être assimilée à une arborescence au sommet de laquelle nous avons le conteneur de premier niveau et dont les différentes branches sont constituées d'autres conteneurs. Les feuilles de l'arborescence correspondant aux composants graphiques.

Le conteneur de premier niveau étant l'élément indispensable de toute application graphique, nous allons donc commencer par étudier en détail ces caractéristiques et son utilisation puis nous étudierons les principaux composants graphiques.

2. Conception d'une interface graphique

Nous avons vu un petit peu plus haut que toute application graphique est au moins constituée d'un conteneur de premier niveau. La bibliothèque Swing dispose de trois classes permettant de jouer ce rôle :

`JApplet` : représente une fenêtre graphique embarquée à l'intérieur d'une page html pour être prise en charge par un navigateur. Cette classe n'a plus d'utilité sur la plateforme Java 11 car la technologie des applets y a été abandonnée.

`JWindow` : représente une fenêtre graphique la plus rudimentaire qui soit. Celle-ci ne dispose pas de barre de titre, de menu système, pas de bordure, c'est en fait un simple rectangle sur l'écran. Cette classe est rarement utilisée sauf pour l'affichage d'un écran d'accueil lors du démarrage d'une application (*splash screen*).

`JFrame` : représente une fenêtre graphique complète et pleinement fonctionnelle. Elle dispose d'une barre de titre, d'un menu système, d'une bordure, elle peut facilement accueillir un menu, c'est bien sûr cet élément que nous allons utiliser dans la très grande majorité des cas.

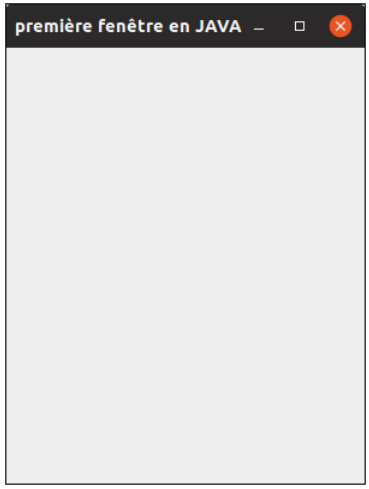
2.1 Les fenêtres

La classe `JFrame` est l'élément indispensable de toute application graphique. Comme pour une classe normale nous devons créer une instance, modifier éventuellement les propriétés et utiliser les méthodes. Voici donc le code de la première application graphique.

```
package fr.eni.editions.ihml;
import javax.swing.JFrame;
public class Principale {
     public static void main(String[] args)
```

```
        {
            JFrame fenetre;
            // création de l'instance de la classe JFrame
            fenetre=new JFrame();
            // modification de la position et de la
            // taille de la fenêtre
            fenetre.setBounds(0,0,300,400);
            // modification du titre de la fenêtre
            fenetre.setTitle("première fenêtre en JAVA");
            // affichage de la fenêtre
            fenetre.setVisible(true);
        }
}
```

Et le résultat de son exécution :



Pour que ce programme compile et s'exécute avec le fichier `module-info.java` dans votre projet, il est nécessaire d'y ajouter la référence au module contenant le package `java.swing`. L'instruction à ajouter est donc la suivante :

```
requires java.desktop;
```

En revenant au programme, il est possible d'admettre que c'est simple d'utilisation et très efficace. C'est d'ailleurs tellement efficace que vous ne pouvez pas arrêter l'application. En effet même si la fenêtre est fermée par l'utilisateur, cette fermeture ne provoque pas la suppression de l'instance de la `JFrame` de la mémoire. La seule solution pour arrêter l'application est de stopper la machine virtuelle Java avec la combinaison de touches [Ctrl] **C**. Il faut bien sûr prévoir une autre solution pour terminer plus facilement l'exécution de l'application et faire en sorte que celle-ci s'arrête à la fermeture de la fenêtre.

La première solution consiste à gérer les événements se produisant lors de la fermeture de la fenêtre et dans un de ces événements provoquer l'arrêt de l'application. Cette solution sera étudiée dans la section consacrée à la gestion des événements.

La deuxième solution utilise des comportements prédéfinis pour la fermeture de la fenêtre. Ces comportements sont déterminés par la méthode `setDefaultCloseOperation`. Plusieurs constantes sont définies pour déterminer l'action entreprise à la fermeture de la fenêtre.

`DISPOSE_ON_CLOSE` : cette option provoque l'arrêt de l'application lors de la fermeture de la dernière fenêtre prise en charge par la machine virtuelle.

`DO_NOTHING_ON_CLOSE` : avec cette option, il ne se passe rien lorsque l'utilisateur demande la fermeture de la fenêtre. Il est dans ce cas obligatoire de gérer les événements pour que l'action de l'utilisateur provoque un effet sur la fenêtre ou l'application.

`EXIT_ON_CLOSE` : cette option provoque l'arrêt de l'application même si d'autres fenêtres sont encore visibles.

`HIDE_ON_CLOSE` : avec cette option, la fenêtre est simplement masquée par un appel à sa méthode `setVisible(false)`.

La classe `JFrame` se trouve située à la fin d'une hiérarchie de classes assez importante et implémente de nombreuses interfaces. De ce fait elle possède donc de nombreuses méthodes et attributs.