Compte rendu TP 1

Auteur: Jan Bayer, Abderrazak Chaki, Alimata Djiré

11 février 2019

Table des matières

1 Introduction

Durant le cours de Génie logiciel, en programmation Java, nous avons eu pour projet la mise en place d'un programme qui nous permettra de visualiser et de modifier le contenu une base de données à travers l'interface Java Database Connectivity. Le programme en langage Java devant être conforme au modèle MVC (modèle, vue, contrôleur) et utilisant la bibliothèque graphique Swing. Dans ce compte rendu, nous allons définir rapidement les principes de la programmation orientée objet, puis on effectuera une analyse sur les différentes démarche que l'on aura suivi et on terminera par décrire les solutions adoptées pour l'implémentation du programme.

2 Analyse

La mise en œuvre de ce projet est basée sur les principes de la programation orientée objet. Dans cette section, on va introduire et expliquer les principes de ce paradigme et les spécificités en Java.

2.1 Programmation Orienté Objet

Pour faciliter la simulation des objets réels, on a introduit la programmation orientée objet (POO). Ce paradigme permet de traduire des objets réels en objet informatique manipulable d'une manière plus logique et les regroupe dans une structure appelée classe. Une classe peut contenir plusieurs attributs et méthodes. Les attributs devront être initialisés et typés pour être correctement exploitable. Les méthodes quant à elles, peuvent utiliser ou non ces attributs, et si besoin créer ses variables locales.

La programmation orientée objet met en place ces trois principes de base :

— **Encapsulation** : c'est le principe de regrouper des données et des méthodes au seins d'une structure.

— **Polymorphisme** : c'est le fait de pouvoir reprendre une structure existante et la modifier pour pouvoir l'adapter à notre besoin.

— **Héritage** : c'est la possibilité de créer une nouvelle classe à partir d'une classe déjà existante. Ainsi la classe fraîchement créée peut utiliser les attributs et les méthodes de sa classe superieur.

2.2 Java

Java est un langage compilé (en bytecode) et interprété (par une JVM) qui est basé sur la programmation orientée objet. En effet, la particularité de Java est que tous ses composants sont

des objets. Cela implique que les développeurs doivent créer une architecture la mieux adaptée à leur application.

Dans Java, on retrouve la notion du packaging, c'est un concept qui lui est spécifique. Cette notion permet de regrouper des classes de facon organisée afin de faciliter la gestion des modules. Le fait de découper le programme en différents packages facilite son développement (en isolant les parties complexe), cela permet aussi de favoriser sa réutilisabilité c'est-à-dire l'aptitude à réutiliser tout le programme ou en partie pour alimenter une nouvelle architecture.

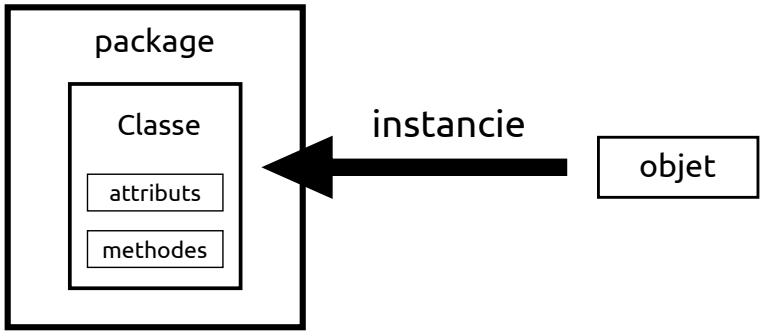

FIGURE 1 – Rélation entre les classes, objets et packages.

Les classes doivent être dans des fichiers portant le même nom qu'elles (le nom doit toujours commencer par une majuscule) et chaque classe peut être englobé dans un package (figure 1). Chaque attribut, methodes et classes ont obligatoirement l'une des propriétés de visibilité suivantes :

- *privée* - visible uniquement dans la classe qui l'englobe
- *protected* - visible aux classes descendante
- *package-private* - visible aux classe dans le même package
- *public* - visible aux seins d'une méthode)

Pour qu'une classe puisse réutiliser une autre d'un package different, il faut spécifier le chemin d'accès [1] vers le répertoir racine de ce package.

2.3 Analyse de projet

Pour implémenter notre application nous utilisons des principes de la programmation orientée objets et des divers motifs de conception. Cette section décrit successivement des concepts utilisés dans notre projets dont MVC, des événements asynchrones et des principes d'interfaces graphiques.

1. On peut définir ce chemin en tant qu'un variable de l'environement ou l'option de script.

2.3.1 MVC

La majorité des interfaces graphiques est basée sur le motif de Model-View-Controller (MVC), il devenait un standard pour l'implémentation des interfaces graphiques et des applications web. MVC nous permet de séparer les fonctions et de créer une architecture robuste pour gérer les données ainsi que l'interface graphique d'une manière indépendante. Le MVC est composé de trois éléments fondamentaux :

— M - Model - est un composant qui manipule les données et souvent représente un accès au stockage de données (une base de donné, un disque dur ...)

— V - View - est un composant qui s'occupe des interactions avec un utilisateur et donc contient les éléments graphiques (fenêtre, panel, button ...)

— C - Controller - est un composant de la logique de l'application.

2.3.2 Interfaces graphiques en Java

Les interfaces de base des interfaces graphiques sont déjà implementées dans la bibliothèque AWT qui se trouve dans le package *java.awt* et donc fait partie de la version officielle de Java. C'est la bibliothèque AWT qui intéragit avec le système d'exploitation où l'interface graphique est gérées d'une facon asyncrhone [2]. La biblothèque AWT permet de signaler une arrivée d'un évenement comme le click de souris ou l'entrée du clavier. L'évenement est ensuite mis dans une liste d'attente pour les évenements graphiques et attend jusqu'à ce qu'un listeneur le récupere et procède à son traitement.

2. Les évènements arrivent de facon indepandante du programme

3 Implémentation

Dans cette partie, nous allons expliquer les choix que nous avons adaptés pour la construction de notre programme.

3.1 Packaging

On a composé la structure de l'application afin que chaque package répresente une fonnctionalité (voir la figure 2). Le package *controller* porte la logique de l'application et gère le comportement que vont adopter le modèle et la vue. Le package *model* correspond au modèle de MVC et s'occupe de la gestion des entitées au sein du système grâce aux classes *Model* et *Person*. Il contient aussi la classe PersonDAO qui a pour but de créer une interface permettant la communication avec la base de donnée. Le package *view* permet de faire l'affichage de l'interface graphique pour utilisateur. Il inclut la classe *View* qui est l'executable de l'application.

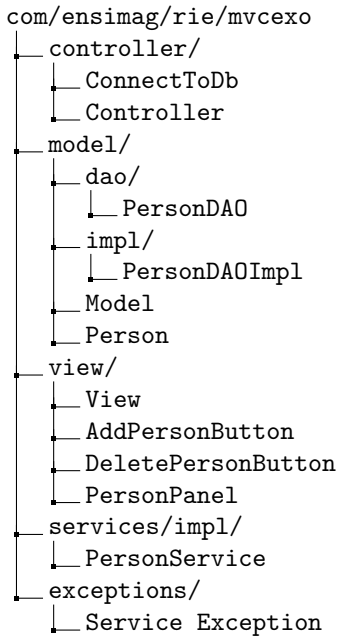

FIGURE 2 – Arboresance des packages

Le package *exceptions* est composé des exceptions qui peuvent être jetées par l'application. Le package *service* offre une couche intermediaire entre le controleur et le modèle.

3.2 Connexion à la base de donnée

Le langage Java dispose d'une interface qui facilite la connexion à la base de donnée, notamment les BDD relationnelles. Cette interface s'appelle *Java Database Connectivity* et se trouve dans le package *java.sql*. Elle définit la manière de la programmation des pilotes pour les divers BDD et

en même temps elle met en jeu l'interface de la connexion pour les applications (figure 3). JDBC ne fournit que les interfaces sans implementation, pour cette raison on a dû télécharger les pilots JDBC Oracle (en forme d'une archive JAR). On a inclut cette archive dans notre Classpath et on a indiqué à JDBC le fourniseur des pilotes à travers l'URL (qui est spécifique pour chaque implementation de JDBC).

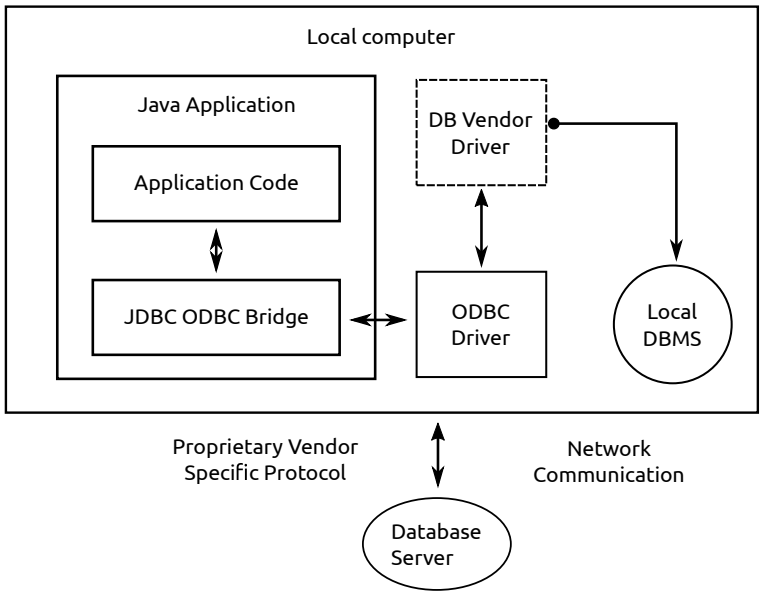

FIGURE 3 – Rélation entre les classes, objets et packages. [1] La figure répresente la communication entre l'application Java, JDBC et le serveur de la BDD. Le code d'applicatione échange avec le pilote ODBC qui crée une copie local du schéma présent dans le serveur (Local DBMS). Les changements sont mergé après la validation (*COMMIT* en SQL).

3.3 Action Listener

Le principe décrit dans la section 2.3.2 est basé sur les arrivée des évènements asynchronnes auxquels on assigne une classe qui gère leurs traitement : Action Listener. Cette classe doit impérativement implementer l'interface *ActionListener* qui est disponible dans la bibliothèque AWT. Dans notre application, ce composant est présent dans la classe *Controller*, il définit les actions pour le modèle et la vue.

3.4 Swing et Layouts

ATW est une bibliothèque qui accède directement à l'interafce graphique native du système d'exploitation ce qui peut empecher les applications d'être portable. Pour pallier à ce problème, Java a mis en place une autre bibliothèque graphique qui est partialement basée sur AWT mais qui permet de porter les applications entre les differents plateformes.

Notre application crée la fênetre en utilisant la classe *JFrame* et utilse les layouts suivants :

— **BorderLayout** - la mise en page des élément basé sur l'orientation (nord, sud, est, ouest et centre)

— **BoxLayout** - la mise en page qui a le même comportement qu'une cascade et on l'utilise pour afficher les buttons pour supprimer de chaque perssonne.

— **FlowLayout** - la mise en page qui empile les éléments en ligne.

4 Problèmes rencontrés

1. **URL de la BDD** Le premier problème était de trouver le bon URL pour pouvoir accéder à la base de donner de l'ENSIMAG. Comme le système de l'URL est propre à Oracle, il était difficile de comprendre son sense et choisir le bon.
2. **JDBC par Oracle** Comme les pilotes d'Oracle ne sont pas instalés par defaut, il fallait les télécharger sur leur site-web en forme d'une archive JAR. On devait l'inclure dans notre Classpath.
3. **Layouts** AWT et Swing offrent beaucoup de possibilités d'implementation de mise en page en utilisant les *LayoutManager* dont principes ne sont pas évident et donc il fallait essayer presque tous.

5 Conclusion

Ce projet nous a permis de mettre en pratique les notions apprises en cours, de pouvoir les développer et de comprendre ou découvrire de nouvelles solutions lors qu'on a été confrontés à des problèmes. Nous avons aprécier l'apprentissage à travers la documentation officielle Java.

Le code source est disponible sur ce repository Git : .

Références

[1] TutorialsPoint. *JDBC Tutorial*. URL : https://www.tutorialspoint.com/jdbc/index.htm (visité le 11/02/2019).