

Rapport de stage  
Licence Informatique 3<sup>ème</sup> année

stagiaire : Athman Mekhzoumi  
tuteur : Olivier Baudon

11 juin 2018

# Table des matières

<b>1</b>	<b>A modifier</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.1.1	Sujet du stage . . . . .	2
1.1.2	Le contexte du stage . . . . .	2
1.2	Déroulement du stage . . . . .	3
1.2.1	première semaine . . . . .	3
1.2.2	Deuxième semaine . . . . .	4
1.2.3	Troisième semaine . . . . .	4
1.2.4	Quatrième semaine . . . . .	6
1.3	Bilan du stage . . . . .	6
1.4	Référence . . . . .	7

# Chapitre 1

## A modifier

### 1.1 Introduction

Dans cette partie, nous allons présenter le thème du stage, les prérequis exigés, le travail demandé et dans quel contexte ce sujet a été posé.

#### 1.1.1 Sujet du stage

Le sujet du stage est *Publier une bibliothèque sur les graphes et la compléter*, une version beta de la bibliothèque était déjà implémenté avant le début du stage, les tâches principales étaient de compléter le code là où il y manquait, améliorer le code en s'inspirant des nouvelles versions du langage *java* et aussi documenter ainsi que commenter la bibliothèque.

Des connaissances en "Programmation orienté objet" et en "Théorie des graphes" m'étaient nécessaires pour comprendre la conception et l'implémentation de la bibliothèque, heureusement pour j'avais déjà acquis les notions de base sur ces deux domaines au cours de mes études universitaires dans mon pays natal, mais cela ne m'était pas suffisant car il me fallait des connaissances plus profondes, alors j'ai lu quelques cours en ligne et codé programmes avant le début du stage pour être au niveau exigé.

#### 1.1.2 Le contexte du stage

Ce stage rentre dans le cadre du cycle licence, c'est un stage obligatoire de 4 semaines minimum, il représente l'élément pédagogique 4TTVP18U et il fait partie du semestre 6 du parcours informatique, l'obtention de ce stage a été faite avec une candidature spontanée adressée au Maître de Conférences Mr BAUDON Olivier qui fait partie de l'équipe des chercheurs du LaBRI.

## 1.2 Déroulement du stage

Dans cette partie nous allons parler du travail fait par l'étudiant, des compétences qui ont été mobilisées pour le réaliser ainsi que les connaissances et compétences acquises et maîtrisées.

### 1.2.1 première semaine

#### Les classes abstraites et les interfaces

Une classe est définie comme abstraite avec le mot clé **abstract**. Les classes abstraites sont à utiliser lorsqu'une classe mère ne doit pas être instanciée, cette dernière peut t'avoir ce qu'on appelle une méthode abstraite qui est caractérisée par le fait qu'elle n'a pas de corps, comme elle peut t'avoir des méthodes normales, par contre si une classe contient une méthode abstraite, cette classe doit alors être déclarée abstraite.

Une interface n'est qu'une classe abstraite à 100%, ce qui signifie aucune méthode d'une interface n'a de corps, elle sert à définir un supertype et à utiliser le polymorphisme, pour implémenter dans une classe il faut utiliser le mot clé **implements**, et on peut implémenter autant qu'on veut dans la même classe, pour que cela soit fait correctement il faut redéfinir toutes les méthodes de l'interface (ou des interfaces) dans la classe concernée.

#### Les exceptions

Lorsqu'un événement que la JVM ne sait pas gérer apparaît, une exception est levée (exemple : division par zéro). une exception correspond donc à une erreur, la superclasse qui gère les exceptions s'appelle **Exception**. C'est possible aussi de créer une classe d'exception personnalisée qui génère ce que le programmeur considère comme exception mais peut être pas par la JVM, pour y procéder il faut lui hériter de la classe `Exception`.

L'instruction qui permet de capturer des exceptions est le bloc **try...catch**, si une exception est levée dans le bloc `try`, les instructions figurant dans le bloc `catch` seront exécutées pour autant que celui-ci capture la bonne exception levée. On peut aussi prévenir la JVM qu'une méthode est risquée de générer une exception grâce au mot clé **throws**. Une instantiation d'une exception est lancée par le biais de l'instruction **throw**.

#### Les varargs

Les varargs permettent de passer un nombre variable d'arguments à une méthode (pourvu qu'ils soient tous du type déclaré dans la signature de la méthode), sans créer explicitement de tableau, en revanche, dans le corps de la méthode, la variable `var-arg` est considérée comme un banal tableau et peut donc être parcourue par un « `foreach` », fournir sa taille avec la propriété `length`. Il y a toutefois deux petites limitations à cette syntaxe :

- Il ne peut y avoir qu'un seul `var-arg` par signature de méthode.
- Le `var-arg` doit toujours être le dernier paramètre.

## 1.2.2 Deuxième semaine

### Les collection

Les collection permet de stocker un nombre variable d'objets, il y a principalement trois types de collection : **List** et **Set** qui hérite de l'interface **Collection** et aussi le type **Map**. chaque type a ses avantages et ses inconvénients et fournies des fonctionnalités propre à lui.

Voici quelques différences les différents types de collections :

- Les **Collection** stockent des objets alors que les **Map** stockent un couple (clé-valeur).
- Si on insère fréquemment des données en milieu de liste, **LinkedList** est plus approprié.
- Si on veut rechercher ou accéder à une valeur via une clé de recherche, vaut mieux opter pour une collection de type **Map**.
- Si on veut traiter une grande quantité de données, le type **Set** convient le plus.

### La généricité

La généricité est un concept très utile pour développer des objets travaillant avec plusieurs types de données, ça nous fait gagné du temps au lieu de développer des classes qui traite de façon identique des données de type différent.

On peut utiliser la généricité sur les objets servant à gérer des collections, et avec l'outil **wildcard** (?) on indiquer que n'importe quel type peut être traité, mais cela revient à rendre ladite collection en lecture seule ce qui peut être désavantageant.

### Git

Pour Faciliter la communication et entre le stagiaire et l'encadrant pendant tout le stage, même les weekend, la plateforme github a été utilisée par pour sauvegarder les changements fait sur la bibliothèque et aussi pour encadrer et guider le stagiaire au fur et à mesure de son apprentissage.

## 1.2.3 Troisième semaine

### Diagramme UML

La modélisation UML permet de vulgariser les aspects liés à la conception et à l'architecture, propres au logiciel (dans ce cas la bibliothèque), au client (dans ce cas les utilisateurs de cette dernière). Aussi, elle apporte une compréhension rapide du programme à d'autres développeurs externes en cas de reprise du logiciel et facilite sa maintenance.

Pour ce projet j'ai utilisé **ObjectAid UML Explorer** qui est un outil de visualisation de code souple et léger pour l'IDE Eclipse. Il affiche le code source Java et les bibliothèques dans des diagrammes de classes et de séquences UML dynamiques qui se mettent automatiquement à jour lorsque mon code change. La figure suivante représente les diagrammes UML obtenu à partir d'ObjectAid et en précisant les paramètres souhaités :

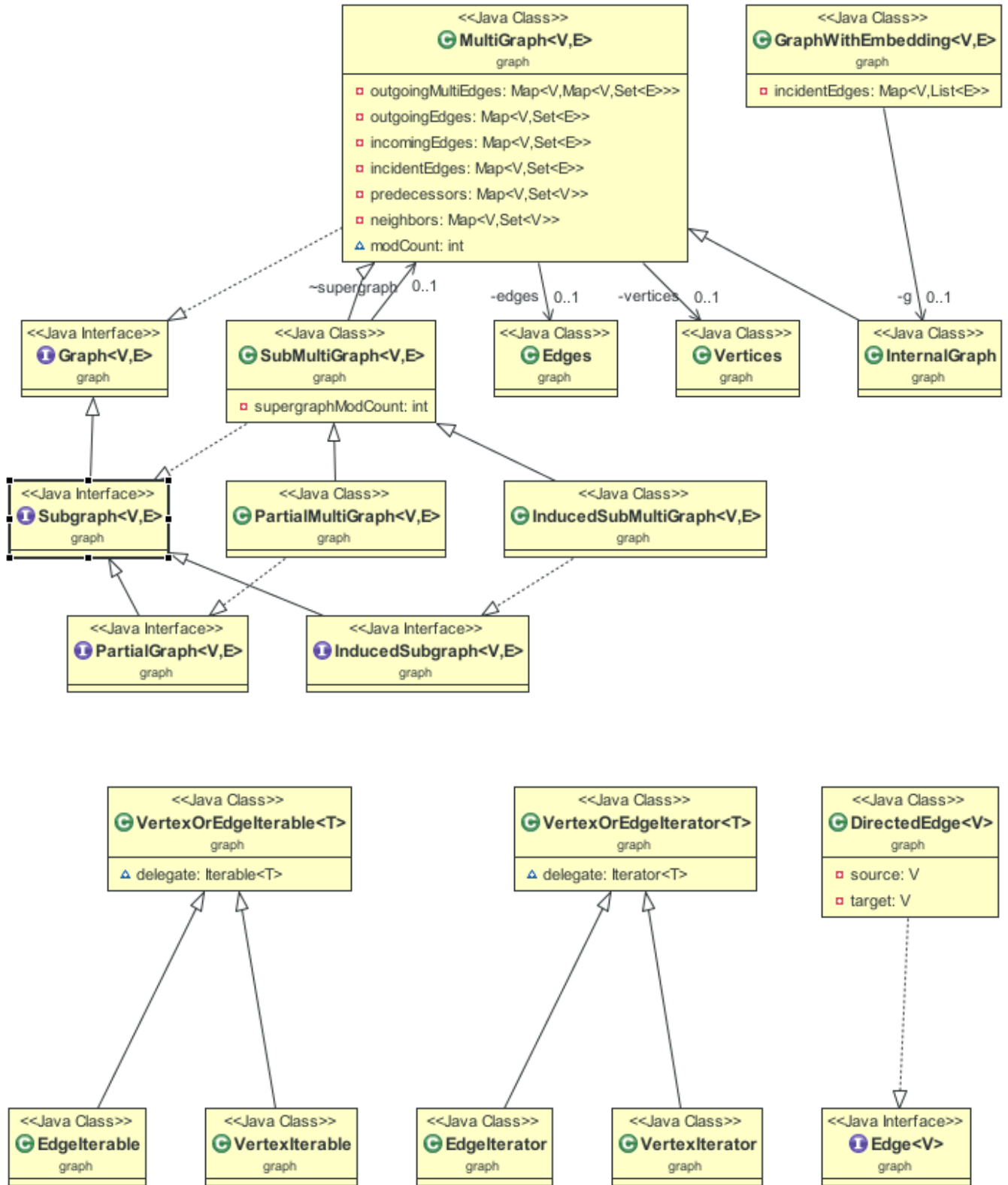


Figure 1 : Diagrammes UML du package "graphe"

## **Classe interne, locale et anonyme**

Une classe interne est déclarée à l'intérieur d'une autre classe, Elle peut donc accéder aux membres de la classe externe. Il y'a deux modèles principales de cette dernière, Les classes internes statiques qui ne peuvent accéder qu'aux membres statiques de leurs classes englobantes respectives, Les classes internes non statiques qui peuvent accéder aux membres statiques de leurs classes ainsi qu'aux membres des objets respectives qui les ont créées.

Une classe interne définie dans un bloc est une classe interne dont la portée est limitée au bloc : c'est une classe interne locale. Une classe locale ne peut pas être statique. Une classe locale peut accéder aux attributs de la classe englobante ainsi qu'aux paramètres et variables locales de la méthode où elle est définie, à condition que ceux-ci soient spécifiés final.

Il est possible de définir une classe interne, sans lui donner de nom par dérivation d'une super classe, ou par implémentation d'une interface, c'est ce qu'on appelle une classe anonyme. Elles sont utiles lorsqu'on a besoin de déclarer une classe pour l'utiliser une seule fois.

## **Correction des warnings**

Au début du stage la bibliothèque contenait multiples warnings, pour toutes sortes de dysfonctionnement ou de problème éventuelle il faut retirer ces warnings. Pour cela chaque warning a été traité individuellement selon le problème qui le génère. Pour procéder à cette opération j'ai eu recours à deux solutions : les solutions proposées sur internet et l'utilitaire de réparation des warnings intégré à l'IDE Eclipse.

## **1.2.4 Quatrième semaine**

**Optimisation du code**

**Lambda expression**

**Entête et licence**

## **1.3 Bilan du stage**

## 1.4 Référence

[https://openclassrooms.com/courses/apprenez-a-programmer-en-java/  
les-classes-abstraites-et-les-interfaces](https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-classes-abstraites-et-les-interfaces)

[https://openclassrooms.com/courses/apprenez-a-programmer-en-java/  
les-exceptions](https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-exceptions)

<http://thecodersbreakfast.net/index.php?post/2008/07/23/77-java-les-var-args>

[https://openclassrooms.com/courses/apprenez-a-programmer-en-java/  
les-collections-d-objets](https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-collections-d-objets)

<http://objectaid.com/>

<http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/classes3.html>