PROGRAMMATION ORIENTEE OBJET

AUVRAY NICOLAS

THOMAS JOHANN

RAPPORT GRAPHE-EDITOR

LICENCE INFORMATIQUE - 2EME ANNEE

Année 2022 – 2023

SOMMAIRE

I. INTRODUCTION

Au cours de la deuxième année d'informatique, l'un des modules étudiés concerne la programmation orientée objet c'est ainsi que les étudiants sont amenés à programmer en Java afin de comprendre les bases de la programmation orientée objet. Dans le cadre d'un projet, il a été demandé aux étudiants de réaliser un éditeur de graphe interactif.

L'éditeur doit être graphique, pour cela, nous avons choisi d'utiliser le package Swing de Java. Pour réaliser des graphes, notre application doit pouvoir dessiner des sommets sur la fenêtre et il doit pouvoir relier deux sommets entre eux. Le dernier point majeur est que l'application doit pouvoir enregistrer/ouvrir des graphes via son interface.

Dans un premier temps, nous présenterons la structure générale du projet qui se découpe selon deux axes que sont l'objet Graphe et ses composants ainsi que les composants de l'interface graphiques l'application. Suite à cela, nous présenterons les différentes fonctionnalités de notre projet ainsi que leurs méthodes de fonctionnement avec de terminé par une ouverture sur les potentiels améliorations du projet.

II. LA STRUCTURE DU PROJET

A) LA CLASS GRAPHE ET SES COMPOSANTS

Afin de représenter des graphes, il a été nécessaire de créer plusieurs classes. Tout d'abord, il a été indispensable de définir une classe abstraite (Element) qui permet de créer toutes les propriétés requises pour la représentation d'un élément du graphe. Chaque classe de notre programme possède des méthodes set et get, permettant de définir ou de récupérer la valeur de ses variables.

On considère alors que chaque élément présent dans le graphe est hérité de Element. Cette classe est composée de :

- *chaîne : nom*
- *entier : taille d'affichage*
- *Couleur : 3 (une couleur d'affichage, une couleur par défaut, une couleur de sélection).*

La création de la classe nous permettra de créer les deux éléments principaux d'un graphe soit : Sommet ; Arc.

La classe **Sommet** est abstraite et elle représentante pour nous un « point » du graphe représenté comme ceci :

- *Hérite* d'Élément
- Des *Coordonnées* de type entier (x,y) pour placer le point

La classe **Arc** est abstraite. Elle permet de représenter le lien entre deux sommets. Par conséquent, un objet Arc possède :

- 2 sommets
- Un entier pour la largeur du trait

Pour dessiner Les Sommets, nous avons décidé de faire hériter Sommet en plusieurs objets distinct selon leurs formes d'affichage comme leurs noms l'indiquent :

- **Rond**
- **Carre**
- **Triangle**

Nous pouvons alors ajouter pour chaque objet, les variables nécessaires afin de le dessiner comme par exemple un rayon pour le Rond ou encore une largeur pour le Carre. Puis, le plus important est de bien redéfinir les méthodes des objets pour chaque type de forme. Par exemple paint() pour les règles de dessin sur la fenêtre, ou encore getLenght() pour la taille de l'objet et bien d'autres encore.

Pour dessiner les Arc dans le graphe, nous avons pris la décision de faire leurs représentations de deux façons, un lien orienté ; un lien non orienté. De fait, interviennent les classes « Arete » et « AreteOriente » toutes deux hérité de « Arc ».

Tous ces éléments nous servent d'objet afin de représenter un Graphe. Un graphe est par conséquent représenté à l'aide d'une liste de Sommet ainsi qu'une liste d'Arc. Cette classe Graphe est bien-évidement accompagnée de toutes les méthodes nécessaires afin de manipuler ces listes.

B) L'INTERFACE GRAPHIQUE

Le but du projet est avant tout de représenter des éléments au sein d'une fenêtre graphique ainsi, il fut nécessaire de créer des classes permettant d'afficher des éléments.

Tout d'abord, il fut nécessaire de créer la classe **Windows**. La class Windows est en fait la fenêtre qui va contenir nous nos éléments graphiques.

Ensuite, il a été préférable de séparer la fenêtre en plusieurs autres objets personnalisés.

Le premier est *BarreMenu*. Cette classe est un objet hérité de *JMenuBar*. Il nous permet de créer une barre de menu en haut de la fenêtre avec plusieurs sections :

- « Fichier » Permettant de faire des traitements sur un fichier.

- « Edition » Permettant de faire des traitements sur un graphe.

- « Affichage » Permettant d'afficher ou masquer des options du programme.

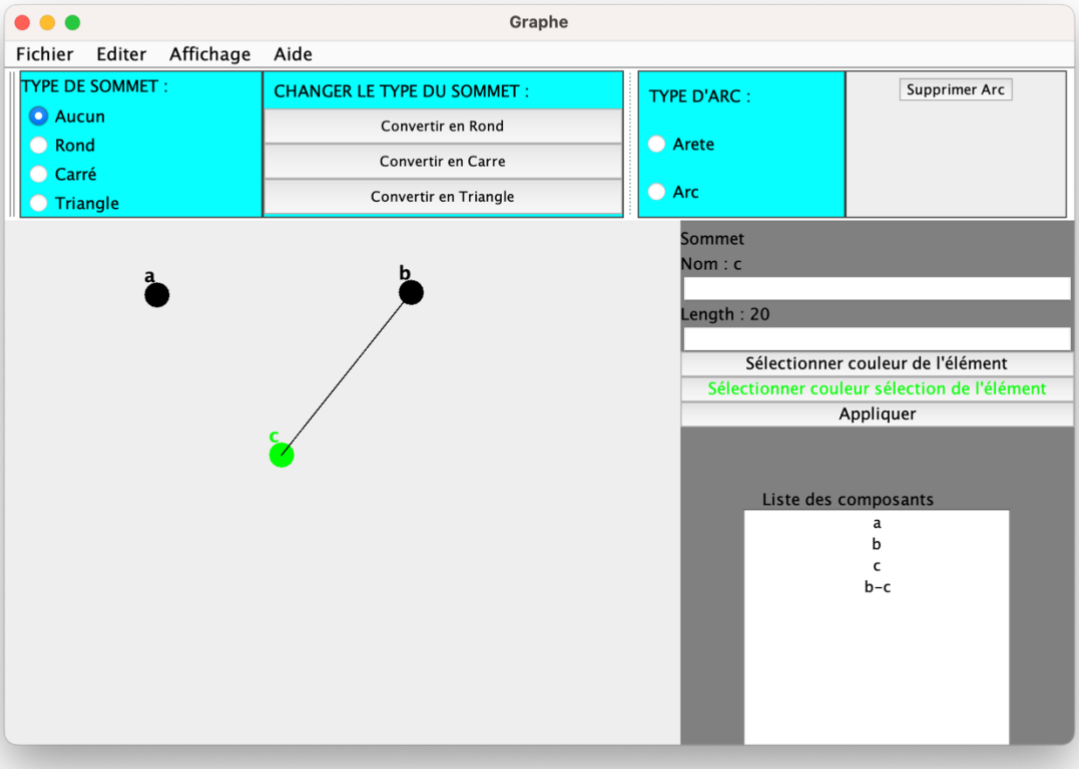
- « Aide » Permettant d'ouvrir un fichier d'aide

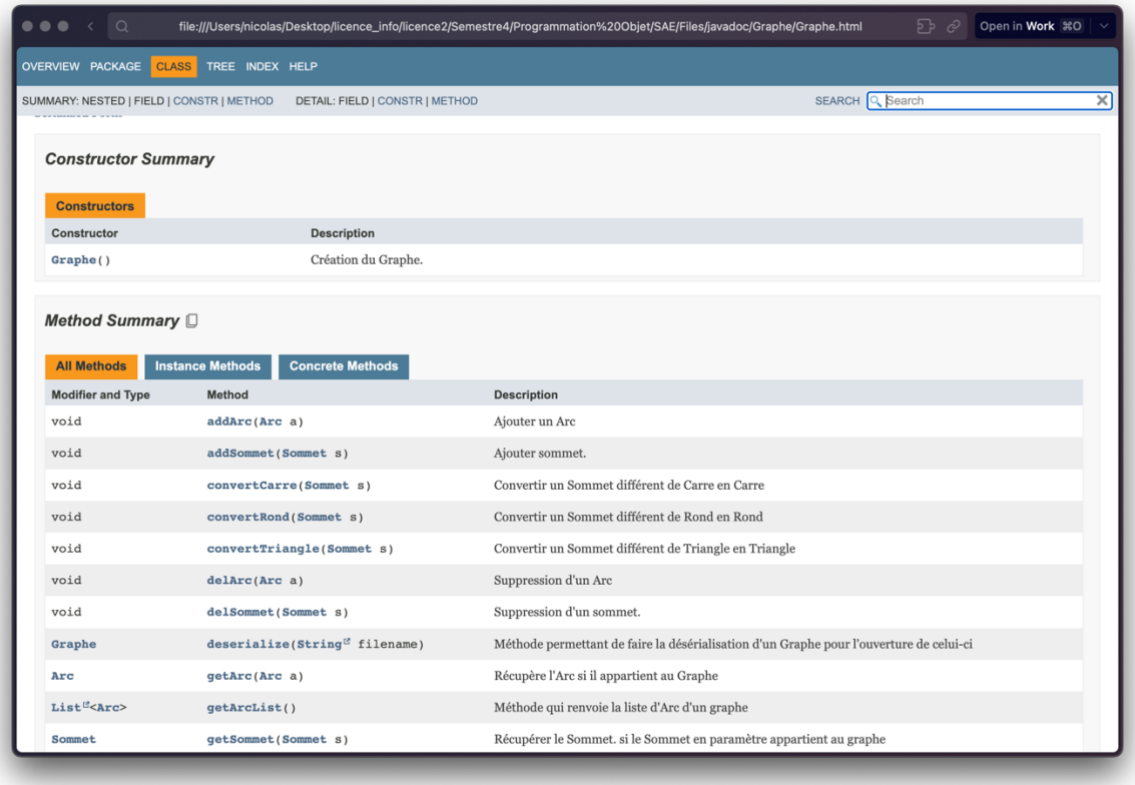
La deuxième est *BarreOutil*. Cette classe est un objet hérité de *JToolBar*. Il nous permet de créer une barre d'outils permettant le choix de l'outil d'édition de l'application.

La troisième est Dessin. Cette classe est en fait un objet hérité de JPanel qui va nous créer la fenêtre de dessin de notre graphe. Cet objet possède un graphe en attribut et implémente MouseListener afin de pouvoir effectuer des actions à l'aide de la souris.

Et enfin, la classe *InfosBar*. *InfosBar* permet d'avoir une fenêtre optionnelle sur la droite de notre application. Ce panel comporte les informations de l'élément sélectionné du graphe. Il permet également de faire toutes les modifications des éléments. Notamment les nom/tailles/couleurs… Les couleurs sont modifiées grâce à un JButton que nous avons personnalisé dans la classe boutonCouleur. En effet, cette classe permet de créer un bouton qui, lors de l'activation ouvre une fenêtre JColorChooser permettant de choisir une couleur.

Windows réunit alors tous ces objets pour nous donner une application fonctionnelle.


Graphe
Fichier
Editer
Affichage
Aide
TYPE DE SOMMET :
Aucun
Rond
Carré
Triangle
CHANGER LE TYPE DU SOMMET :
Convertir en Rond
Convertir en Carre
Convertir en Triangle
TYPE D'ARC :
Arete
Arc
Supprimer Arc
a
b
c
Sommet
Nom : c
Length : 20
Sélectionner couleur de l'élément
Sélectionner couleur sélection de l'élément
Appliquer
Liste des composants
a
b
c
b-c


file:///Users/nicolas/Desktop/licence_info/licence2/Semestre4/Programmation%20Objet/SAE/Files/javadoc/Graphe/Graphe.html
Open in Work ⌘O
OVERVIEW PACKAGE CLASS TREE INDEX HELP
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH Search
Constructor Summary
Constructors
Constructor Description
Graphe() Création du Graphe.
Method Summary
All Methods Instance Methods Concrete Methods
Modifier and Type Method Description
void addArc(Arc a) Ajouter un Arc
void addSommet(Sommet s) Ajouter sommet.
void convertCarre(Sommet s) Convertir un Sommet différent de Carre en Carre
void convertRond(Sommet s) Convertir un Sommet différent de Rond en Rond
void convertTriangle(Sommet s) Convertir un Sommet différent de Triangle en Triangle
void delArc(Arc a) Suppression d'un Arc
void delSommet(Sommet s) Suppression d'un sommet.
Graphe deserialize(String filename) Méthode permettant de faire la désérialisation d'un Graphe pour l'ouverture de celui-ci
Arc getArc(Arc a) Récupère l'Arc si il appartient au Graphe
List<Arc> getArcList() Méthode qui renvoie la liste d'Arc d'un graphe
Sommet getSommet(Sommet s) Récupérer le Sommet. si le Sommet en paramètre appartient au graphe

III. LES DIFFERENTES FONCTIONNALITES DES PARTIES

Grâce au mouseListener, nous pouvons effectuer plusieurs actions sur notre panel de dessin comme par exemple :

- Ajouter un sommet avec clic gauche
- Supprimer un sommet avec clic droit
- Déplacer un sommet avec un « Drag and drop »
- Sélectionner un sommet avec clic gauche

Grâce à l'InfosBar (affichable via CTRL+F **OU** affichage/BarreInfos), nous pouvons effectuer des actions de modifications sur les éléments du graphe comme par exemple :

- Changer le nom
- Changer les différentes couleurs (élément/sélection)
- Changer la taille

Nous pouvons également sélectionner des éléments directement grâce à la liste interactive de sélection en bas de la barre infos.

L'ouverture d'un graphe ce fait via CTRL+O **OU** fichier/ouvrir.

Les différents changements sont appliqués lors de la validation grâce au bouton « appliquer » de la fenêtre.

Les sommets peuvent être copier/couper/couper via les raccourcis usuel CTRL+C/CTRL+X/CTRL+V ou directement via le menu d'édition.

La création d'un nouveau Graphe se fait par CTRL+N **OU** fichier/nouveau fichier.

L'enregistrement du graphe est possible via CTRL+S **OU** fichier/enregistrer.

L'enregistrement du graphe dans un dossier spécifique se fait via CTRL+SHIFT+S **OU** fichier/enregistrer-sous.

L'ouverture du fichier d'aide se fait via CTRL+H **OU** aide/Read-Me

L'ouverture de la javadoc peut se faire via CTRL+J **OU** aide/Javadoc

IV. LES AMÉLIORATION POSSIBLE

Il existe plusieurs pistes d'amélioration intéressantes pour l'application en question.

La première consiste à ajouter des traitements de graphe en utilisant différents algorithmes. Cela permettrait, par exemple, de reconnaître automatiquement le type de graphe (connexe, circuit, etc.), d'utiliser des algorithmes de parcours de graphe pour retrouver des chemins particuliers, ou encore de trouver les plus courts chemins entre les différents nœuds du graphe. Ces fonctionnalités pourraient rendre l'application plus robuste et plus flexible, et permettraient à l'utilisateur de mieux visualiser les relations entre les différents éléments du graphe en changeant la couleur des chemins par exemple.

La deuxième piste d'amélioration consisterait à ajouter un paramètre poids sur les arêtes du graphe. En effet, cela permettrait de mieux représenter certaines situations où les relations entre les nœuds ont des poids différents. Par exemple, si l'on utilise un graphe pour représenter un réseau de transport, les distances entre les différentes villes pourraient être représentées par des poids sur les arêtes. Les modifications de poids pourraient être prises en compte, et un algorithme lié au poids permettrait de

retrouver les chemins les plus courts en prenant en compte ces différents poids. Cette fonctionnalité pourrait rendre l'application plus pertinente pour des domaines spécifiques, comme la logistique, l'optimisation, ou encore la cartographie.

Finalement, on pourrait envisager de sérialiser les graphes sous un format spécifique et modifiable par l'utilisateur afin de pouvoir générer des graphes plus complexes directement via un fichier texte.

CONCLUSION

En utilisant des classes pour représenter les graphes et des classes pour créer une interface graphique, nous avons pu mettre en place une solution répondant aux exigences du cahier des charges. Cette approche nous a permis de mettre en pratique les principes de la programmation objet, en nous concentrant sur l'aspect théorique du sujet. Grâce à cette modélisation, nous avons pu construire une solution modulaire, où chaque partie du programme était bien isolée et pouvait être développée de manière indépendante.

Le résultat final est une application proposant une interface graphique intuitive et ludique, permettant à l'utilisateur de créer et de manipuler facilement des graphes. L'utilisation de classes a également permis d'assurer la cohérence du code et la réutilisation de certaines parties du programme, ce qui a permis de gagner en efficacité dans le développement.

En somme, ce projet a été une occasion pour nous de mettre en pratique les principes de la programmation objet de manière créative et de construire une solution originale et ludique tout en respectant les principes de la modularité et de la réutilisation du code. Cette expérience nous a permis de consolider notre compréhension des concepts de la programmation orientée objet et de développer nos compétences en matière de conception et de développement de programmes informatiques.

PROGRAMMATION ORIENTEE OBJET

AUVRAY NICOLAS

THOMAS JOHANN

RAPPORT GRAPHE-EDITOR

LICENCE INFORMATIQUE - 2EME ANNEE

Année 2022 – 2023