SAÉ 3.01 : Développement d'une Application

Génération automatique de diagramme de classes
Compte-rendu du projet

Rémi Choffat
Noah Laghlali
Tuline Leveque
Gabin Mathieu

10 janvier 2025

Table des matières

1 Liste des fonctionnalités réalisées

1.1 Ajout

— Importer une classe ;
— Importer un package ;
— *Depuis n'importe quel emplacement de stockage.*

1.2 Exportation

— Exporter le diagramme sous format UML ;
— Exporter le diagramme sous forme d'image.

1.3 Affichage

— Afficher le diagramme ;
— Afficher/masquer une/toutes les classes ;
— Afficher/masquer les classes parent d'une classe ;
— Afficher/masquer toutes les relations ;
— Afficher/masquer tous les héritages ;
— Afficher/masquer toutes les implémentations ;
— Afficher/masquer les attributs d'une/de toutes les classes ;
— Afficher/masquer les méthodes d'une/de toutes les classes.

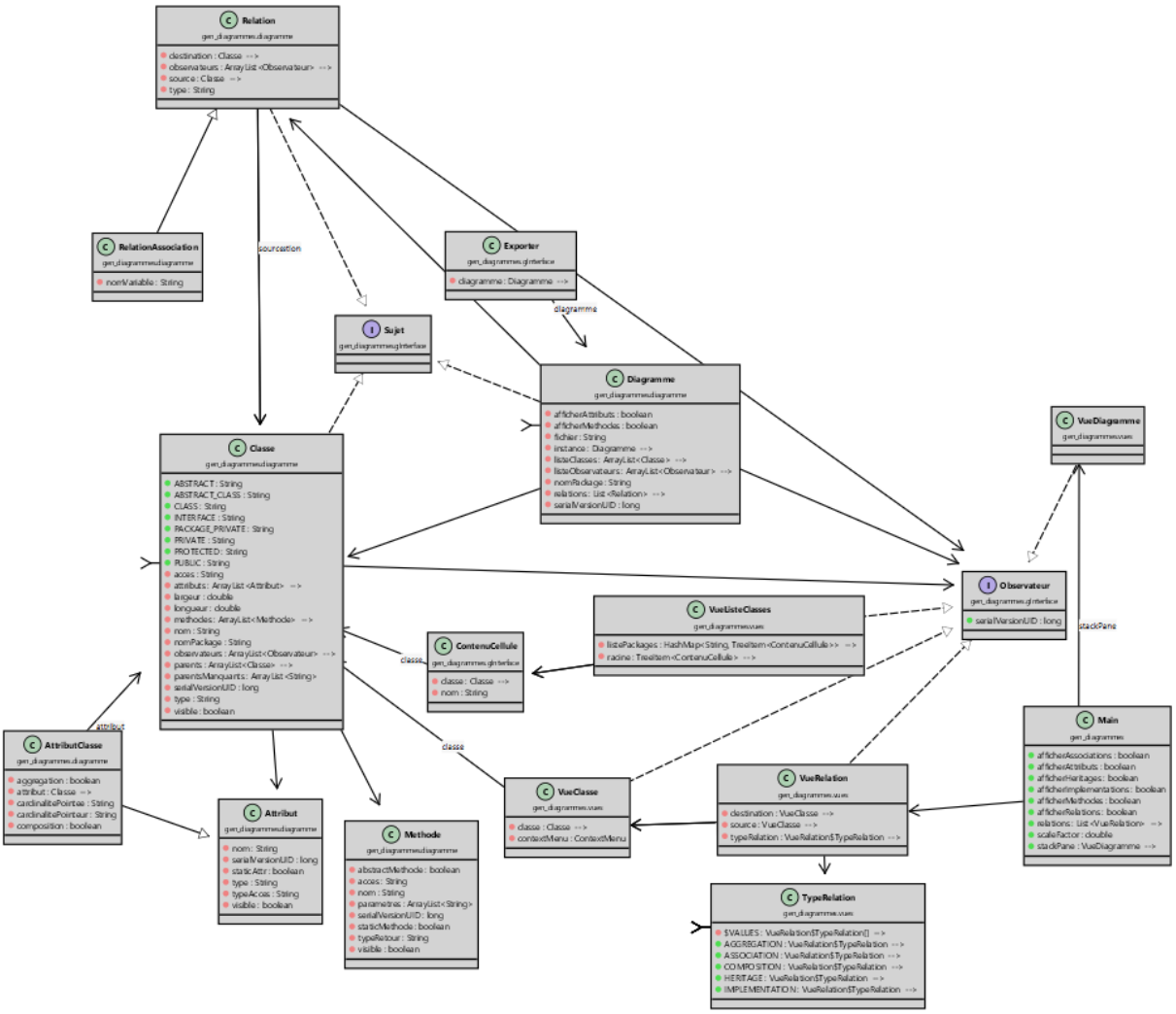
1.4 Fichiers

— Enregistrer un diagramme `.plante` ;
— Charger un diagramme `.plante` ;
— Générer un nouveau diagramme `.plante`.

1.5 Autre

— Supprimer une/toutes les classes ;
— Créer une nouvelle classe depuis le diagramme ;
— Déplacer une classe (glissement) ;
— Se déplacer dans la page (utiliser les flèches directionnelles + zoomer/dézoomer).

2 Diagramme de classes final



3 Répartition du travail

3.1 Itération 1

— **Rémi** : création classe `Classe` + gestion relations entre les classes + chargement fichiers `.class` + génération diagrammes UML ;
— **Tuline** : création classes `Methode`, `Attribut`, `Diagramme` + diagrammes de séquence de l'application + génération diagrammes UML ;
— **Gabin** : modification classe `Classe` + chargement des fichiers `.class` ;
— **Noah** : test de chaque classe de l'application + correction des classes.

3.2 Itération 2

— **Rémi** : gestion des relations entre les classes + réglage de bugs itération 1 + aide interface graphique + diagramme classe ;
— **Tuline** : création vue d'une classe + modification interface graphique + mise en place d'une position pour les classes + diagrammes de séquence ;
— **Gabin** : création en partie de l'interface graphique + squelette MVC ;
— **Noah** : interface graphique + fonctionnalité sélection fichier (recherche manuelle ou glissement) + début `VueRelation` + possibilité de sélectionner et changer la place d'une classe.

3.3 Itération 3

— **Rémi** : exporter le diagramme sous forme d'image + afficher/masquer une classe ;
— **Tuline** : afficher/masquer une classe ;
— **Noah** : afficher des flèches entre les classes + déplacer les classes.

3.4 Itération 4

— **Rémi** : afficher/masquer les attributs d'une classe + afficher/masquer les méthodes d'une classe + charger une classe de n'importe quel package + afficher/masquer les classes parent + possibilité de sauvegarder et charger le diagramme ;
— **Tuline** : afficher le package dans le menu (`VueListeClasse` + `ContenuCellule`) ;
— **Gabin** : créer une classe + nettoyage du main + réorganisation globale pour mise en place plus concrète du patron MVC ;
— **Noah** : afficher les flèches en fonction de la relation.

3.5 Itération 5

— **Rémi** : correction de bugs + graphe de scène + visibilité des sous-packages dans le menu ;
— **Tuline** : diagramme de classes global + diagrammes de classes détaillés + diagrammes de séquence + graphe de scène détaillé + compte rendu global et préparation à la soutenance ;
— **Gabin** : affichage du nom de l'attribut sur la flèche + correction de bugs ;
— **Noah** : se déplacer sur la page avec les flèches directionnelles + gérer l'écart entre les flèches qui se superposent.

4 Présentation d'un élément original

4.1 Rémi

J'ai ajouté une fonctionnalité permettant de sauvegarder l'état du diagramme en cours de modification dans un fichier à télécharger. Cela permet à l'utilisateur d'enregistrer sa progression (classes affichées, classes ajoutées), pour consulter son diagramme ultérieurement et poursuivre ses modifications. Pour arriver à ce résultat, il m'a fallu adapter plusieurs classes pour permettre une sérialisation efficace, et pour pouvoir charger de manière correcte le diagramme contenu dans le fichier de l'utilisateur.

4.2 Tuline

J'ai mis en place un *TreeView* qui permet d'afficher sous forme d'arborescence les différents packages importés dans le projet. J'ai trouvé cette partie intéressante, notamment lors de la gestion d'événements sur ce TreeView afin d'afficher ou de masquer une classe. La première difficulté rencontrée était de mettre un contrôleur sur chaque "cellule" de ce TreeView, ce qui n'était pas possible. J'ai donc résolu le problème en utilisant un "handle()" qui directement s'occupait d'afficher ou masquer la classe sur laquelle l'utilisateur venait de cliquer (changement d'un attribut "visible" dans la classe qui permet d'afficher/masquer la classe lorsque la page s'actualise). Mais afin de récupérer la classe et pas uniquement son nom dans le TreeCell, j'ai dû créer une nouvelle classe "ContenuCellule" qui avait pour attributs le nom de la classe et la classe en elle-même, ce qui m'a permis de récupérer l'attribut classe dont j'avais besoin.

4.3 Gabin

J'ai changé la structure du projet afin de mettre en place plus concrètement le patron MVC. Lors de l'itération 3, nous avions beaucoup de "setOnaction" dans le main. Nous ne respections donc pas le patron que nous voulions mettre en place. J'ai dû déplacer le code dans une vue et des contrôleurs que j'ai créés.

4.4 Noah

J'ai ajouté tout un système de déplacement dans le logiciel. J'ai également ajouté un système de zoom/dézoom selon la position de la souris pour pouvoir faire des grands déplacements et avoir une vue globale du diagramme. De plus, j'ai mis en place les déplacements avec les flèches directionnelles pour faire des déplacements plus précis. Je me suis également occupé du déplacement des classes avec la souris, en prenant en compte le zoom et en gérant les flèches des relations.

5 Éléments modifiés par rapport à l'étude préalable

Initialement, nous avions prévu d'implémenter les fonctionnalités suivantes :
— Gérer la taille des classes ;
— Gérer la place des flèches ;
— Générer les squelettes des classes construites au sein de l'application.
Cependant, ces fonctionnalités n'ont pas été mises en place car nous avons estimé que d'autres points étaient plus pertinents à travailler.

Néanmoins, nous avons ajouté des fonctionnalités non demandées telles que :
— Enregistrer/charger un diagramme (au format `.plante`) ;
— Se déplacer dans la fenêtre d'affichage.

De plus, dans notre diagramme de classe lors de l'étude préalable, nous avions initialement imaginé qu'un diagramme correspondait à un seul package. Dans notre version finale, il est désormais possible d'ajouter autant de packages que souhaité à un diagramme.

6 Patrons de conception et d'architecture mis en œuvre

Nous avons mis en place le **patron d'architecture MVC** pour l'ensemble du projet, ainsi que le **patron Singleton** pour la classe `Diagramme`.

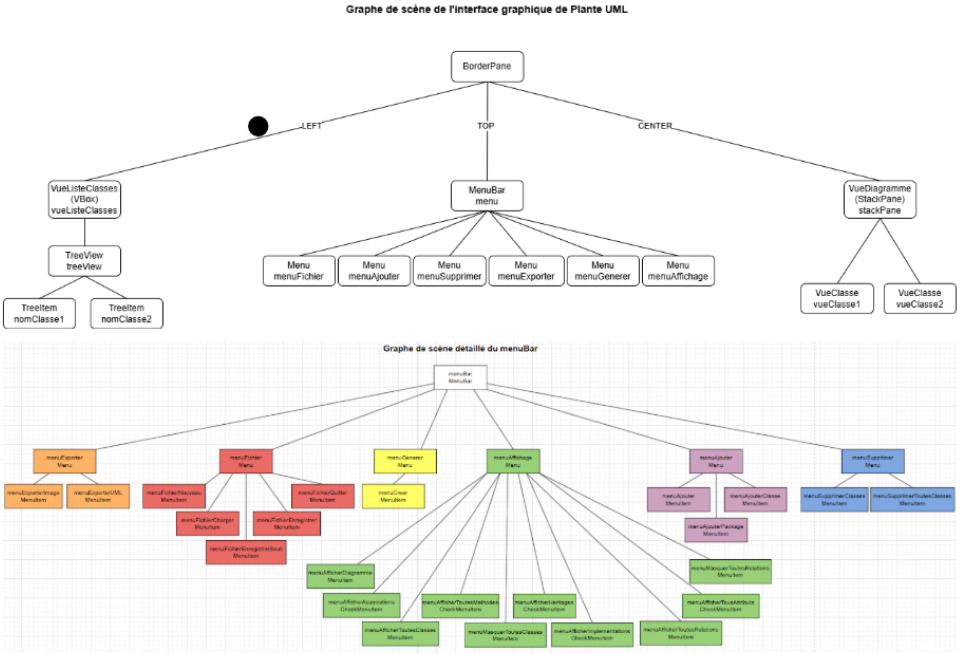
6.1 Patron MVC

Étant donné que notre application devait inclure une interface graphique et des interactions avec l'utilisateur, il était essentiel d'implémenter le patron MVC. Ce patron nous a permis de structurer le code en séparant :
— le **Modèle**, qui gère les données et la logique métier ;
— la **Vue**, qui s'occupe de l'affichage et de l'interface utilisateur ;
— le **Contrôleur**, qui relie la Vue au Modèle en gérant les interactions utilisateur.

6.2 Patron Singleton

Notre classe `Diagramme` devait être unique et accessible depuis n'importe quelle partie du projet. Pour répondre à ce besoin, nous avons implémenté le patron Singleton, qui garantit qu'une seule instance de cette classe est créée et partagée dans l'application.

7 Graphe de scène de l'interface graphique



8 Étapes pour lancer l'application

8.1 1. Cloner le dépôt GitHub

1. Ouvrez un terminal ou utilisez l'interface Git de votre IDE.
2. Clonez le dépôt avec la commande suivante :

```
git clone git@github.com:remi-choffat/S3-01.git
```

3. Dans IntelliJ IDEA, allez dans `File > Open` et sélectionnez le dossier contenant le projet cloné.

8.2 2. Configurer IntelliJ IDEA

8.2.1 Configurer le JDK

1. Allez dans `File > Project Structure > SDKs`.
2. Ajoutez le JDK 21 si ce n'est pas déjà fait.
3. Assurez-vous que le projet utilise Java 21 :
   — Allez dans `File > Project Structure > Modules`.
   — Vérifiez que le module est associé au JDK 21.

8.2.2 Ajouter JavaFX depuis Maven

1. Allez dans `File > Project Structure > Libraries > + > From Maven`.
2. Ajoutez les bibliothèques suivantes :
   — `openjfx:javafx.base:21`
   — `openjfx:javafx.controls:21`
   — `openjfx:javafx.swing:21`
3. IntelliJ téléchargera automatiquement les dépendances.

8.2.3 Configurer Lombok

1. Installez le plugin Lombok :
   — Allez dans `File > Settings > Plugins > Marketplace`.
   — Recherchez et installez `Lombok`.
2. Activez le traitement des annotations :
   — Allez dans `File > Settings > Build, Execution, Deployment > Compiler > Annotation Processors`.
   — Activez `Enable annotation processing`.

8.3 3. Exécuter l'application

1. Créez une configuration d'exécution :
   — Allez dans `Run/Debug Configurations > + > Application`.
   — Configurez les options suivantes :
      — `Name` : Plante UML.
      — `Main class` : `gen_diagrammes.Main`.
      — `Use classpath of module` : Sélectionnez le module principal.
2. Lancer l'application en cliquant sur `Run`.

8.4 Accès aux fonctionnalités

- **Menu principal :**
  - **Fichier** : Créer, ouvrir, et enregistrer un diagramme.
  - **Ajouter** : Ajouter des classes ou des packages.
  - **Exporter** : Exporter les diagrammes en image ou en PlantUML.
  - **Affichage** : Contrôler la visibilité des éléments du diagramme.
- **Zoom et navigation :**
  - Utilisez la molette pour zoomer/dézoomer.
  - Déplacez les éléments par glisser-déposer.