

كلية العلوم
ⵜⴰⴳⴷⴰⵢⵜ ⵜⴰⵎⴻⵎⴰⵙⵜ
FACULTÉ DES SCIENCES



UNIVERSITÉ IBN ZOHR

FACULTÉ DES SCIENCES

Département Informatique

Filière Sciences Mathématiques et Informatique

PFE

Présenté par : Aicha AIT HAMMOUCH

Khadija NID SAID

Mohammed OUAICHA

Pour l'obtention de la

Licence en Sciences Mathématiques et Informatique

Mapping O/R et génération de code Java/C++ à partir de modèles UML

soutenu le : 31/05/2023

Encadré par : AYOUB SABRAOUI

Année universitaire 2022-2023



Dédicaces

A Dieu tout puissant notre créateur

Je dédie ce modeste travail à mes très chers parents, mes plus grands soutiens et sources de respect, pour leur inépuisable support et leurs nombreux sacrifices.

Je souhaite également exprimer ma profonde gratitude envers M. Sabraoui, mon encadrant, dont la collaboration et l'accompagnement ont été essentiels pour la réalisation de ce travail.

À mes camarades Khadija et Aïcha de projet, qui ont partagé avec moi cette aventure, je vous remercie pour notre collaboration fructueuse et notre dévouement mutuel.

À tous mes proches, qui ont constamment encouragé ma progression, et à mes amis, qui m'ont soutenu et cru en moi, je leur exprime ma reconnaissance sincère.

Enfin, je rends hommage à tous les professeurs qui ont marqué ma vie et m'ont inspiré à contribuer de manière significative à notre société. Puissent-ils tous trouver un bonheur immense tant dans cette vie que dans l'au-delà.

OUAICHA MOHAMMED





Dédicaces

Je dédie humblement ce mémoire à ma très chère mère, à mon père et à Dieu, qui m'ont soutenu tout au long de mon parcours académique.

À ma mère, je suis profondément reconnaissante pour ses sacrifices, son amour inconditionnel et son soutien constant.

À mon père, je suis reconnaissante pour son dévouement, ses conseils avisés et son exemple inspirant. Que Dieu les bénisse pour tout ce qu'ils ont fait pour moi.

Je souhaite également exprimer ma profonde gratitude envers M. Sabraoui, mon encadrant, dont la collaboration et l'accompagnement ont été essentiels pour la réalisation de ce travail.

À ma chère sœur et mes chers frères, je vous remercie pour votre encouragement et votre soutien sans faille. Votre présence dans ma vie a été une source de motivation et de bonheur.

Je tiens à exprimer ma reconnaissance à mes camarades Mohamed et Aïcha de projet pour notre collaboration fructueuse et notre travail d'équipe. Vos contributions ont grandement enrichi cette expérience académique.

Enfin, je suis reconnaissante envers toute ma famille et tous mes amis qui ont été présents et m'ont apporté leur soutien lorsque j'en avais le plus besoin. Vos encouragements et votre présence ont été des piliers essentiels dans ma vie.

Merci infiniment à tous ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire. Votre soutien et vos encouragements ont été d'une valeur inestimable pour moi.

NID SAID KHADIJA





Dédicaces

Je tiens à exprimer ma profonde gratitude envers Dieu, mes parents et mes frères, pour leur amour inconditionnel et leurs sacrifices inestimables. Leur soutien indéfectible et leurs encouragements ont été une source d'inspiration et de motivation tout au long de ce parcours. Je ne saurais assez les remercier pour leur confiance en moi et leur soutien sans faille.

Je souhaite également exprimer ma gratitude envers mon encadrant, M. Sabraoui, dont la collaboration, les conseils précieux et l'accompagnement ont été d'une importance capitale pour la réalisation de ce projet. Sa guidance éclairée a contribué à façonner ma réflexion et à atteindre des résultats de qualité. Je lui suis sincèrement reconnaissante pour son implication et sa disponibilité.

Je n'oublie pas non plus mes camarades de projet, Mohamed et Khadija, avec qui j'ai partagé cette aventure. Leur collaboration, leur dévouement et leur esprit d'équipe ont été essentiels pour surmonter les défis et mener ce projet à bien. Leur soutien mutuel et leur engagement ont créé une dynamique de travail positive et productive.

Enfin, je tiens à remercier tous mes amis, qui ont été présents à mes côtés tout au long de ce projet. Leur soutien moral, leurs encouragements et leurs précieux conseils ont été d'une grande valeur. Leur présence dans ma vie a été une source de joie et de réconfort.

C'est avec une profonde gratitude que je dédie ce projet à toutes ces personnes exceptionnelles qui ont contribué à sa réalisation. Leur impact positif dans ma vie restera à jamais gravé dans mon cœur.

AIT HAMMOUCH AICHA





Remerciements

Nous tenons à exprimer nos sincères remerciements à tous les membres du jury pour avoir accepté de juger ce travail. Nous espérons que cette première expérience répondra à leurs attentes et sera de qualité.

Nous adressons également nos remerciements les plus chaleureux à Monsieur Ayoub Sabraoui pour avoir accepté de nous encadrer tout au long de ce projet. Sa disponibilité, ses conseils avisés et son soutien constant ont été d'une importance capitale pour notre réussite. Nous lui exprimons notre admiration et notre profonde reconnaissance pour son investissement et son accompagnement.

Nous souhaitons également exprimer notre gratitude envers nos chers professeurs, qui ont joué un rôle essentiel dans notre formation tout au long de ces trois années passées à la Faculté des sciences. Leur enseignement de qualité et leur engagement envers notre réussite académique ont été une source d'inspiration et de motivation.

Enfin, nous tenons à remercier toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce travail. Leur soutien moral, leurs encouragements et leurs conseils précieux ont été d'une grande valeur.

C'est grâce à la collaboration et à l'appui de tous ces acteurs que ce projet a pu voir le jour. Nous leur sommes infiniment reconnaissants et nous espérons avoir répondu à leurs attentes.








Table des matières


Dédicaces	i
Remerciements	ii
Table des matières	iii
Liste des figures	iv
Liste des abréviations.....	v
Résumé	vi
Abstract.....	vii
Introduction générale.....	1
Chapitre 1 : Contexte général du projet	3
I. INTRODUCTION :	3
II. PRÉSENTATION DU PROJET	3
1. Contexte du projet	3
2. Description du projet.....	3
3. Problématique du projet	4
4. Objectifs	5
4.1 Objectifs personnels	5
4.2 Objectifs professionnels	5
5. Conduite de projet	5
6. Planification du projet	6
7. Conclusion.....	7
Chapitre 2 : Analyse et conception	8
I. INTRODUCTION :	8
II. ANALYSES ET CONCEPTION :	8
1. Spécification des Besoins :.....	8
1.1. Besoins fonctionnels :	8
1. Saisie des informations des classes :	8
2. Gestion des associations entre les classes :	9
3. Sérialisation des données en XML :	9
4. Génération des classes Java, CPP et du code SQL :	9
5. Personnalisation des classes :	9
6. Mapping objet-relationnel :	9
1.2. Besoins non fonctionnels :	10
1. Portabilité :	10



2. Performances :	10
3. Convivialité :	10
2. Spécification de MOR :	10
2.1. Solution MOR proposée :	10
1. Présentation des classes en SQL :	11
2. Présentation de l'héritage en SQL :	12
3. Présentation des associations en SQL :	13
- agrégation :	14
- Composition :	15
3. UML :	16
3.1. Diagramme de cas d'utilisation :	16
3.2. Diagramme de classes :	17
3.3. Diagramme de séquences :	19
3.4. Diagramme de classes pour la DTD :	21
Chapitre 3 : Etude technique et environnement :	22
I. INTRODUCTION :	22
II. CHOIX DES TECHNOLOGIES :	22
1. JDOM :	22
2. XML ET DTD :	23
2.1 Le Modèle et le Métamodèle :	23
2.2 XML et DTD :	24
III. LANGAGES ET OUTILS DE DÉVELOPPEMENT :	25
1. Les langages de programmation :	25
• JAVA :	25
2. Les langages de présentation :	25
• JAVA SWING :	25
3. Les outils de développement :	26
• Visual Studio Code :	26
• GITHUB :	27
• GIT :	28
4. Les outils de Conception :	28
• LUCIDCHART :	28
5. Les outils de Design :	30
• FREEPIK :	30
6. Conclusion :	31
Chapitre 4 : Réalisation :	32
I. INTRODUCTION :	32
II. LES INTERFACES DE L'APPLICATION :	32




1. La page de garde :	32
2. saisir les informations des classes	34
1.1. Le nombre de classes et leurs noms	34
1.2. Les attributs d'une classe	35
1.3. Les méthodes d'une classe	35
1.4. Les associations d'une classe	37
1.5. Affichage des résultats	38
Conclusion Générale et perspectives	41
Bibliographie.....	42
Annexes	43
I. Les Relations Entre les classes en JAVA	43
1- Association Simple :	43
2-Agrégation:	44
3 - Composition:.....	45
II. les fonctions présentant l'héritage et les associations en SQL	46
- La fonction "printExtendIfWeHaveIt":	46
- La fonction " printLesVariableDautreClasses " :	48
- La fonction " printTablesFromXmlIntoSqlFile " :	49
- La fonction "printlesChampsduTable" :	50





Liste des figures

Figure 1 : Diagramme de Gantt pour la gestion de projet.....	7
Table 1 : présentation d'une classe en SQL.....	11
Figure 2 : Notion d'héritage	12
Figure 3 :L'héritage dans un DCL.....	13
Figure 4 : Représentation de l'héritage en SQL	13
Figure 5 : Association simple dans un DCL.....	14
Figure 6 : Représentation d'association simple en SQL.....	14
Figure 7 : Agrégation dans un DCL.....	15
Figure 8 : Représentation d'une Agrégation en SQL.....	15
Figure 9 : Composition dans un DCL.....	16
Figure 10 : exemple Composition en SQL.....	16
Figure 11 : Diagramme de cas d'utilisations.....	17
Figure 12 : Diagrammes de classes.....	18
Figure 13 : Diagrammes de séquences.....	20
Figure 14 : diagramme de classes pour la DTD.....	21
Figure 15 : la bibliothèque Java JDOM.....	22
Figure 16 : XML ET DTD.....	23
Figure 17: Visual Studio Code	26
Figure 18 : GITHUB.....	27
Figure 19 : GIT.....	27
Figure 20 : LUCIDCHART.....	28
Figure 21 : FREEPIK.....	29







Figure 22 : page d'accueil.....	31
Figure 23 : nouveau projet.....	32
Figure 24 : ouvrir un projet existant	32
Figure 25 : nombre des classes.....	33
Figure 26 : Les noms des classes.....	33
Figure 27 : Les attributs d'une classe.....	34
Figure 28 : Les méthodes d'une classe.....	35
Figure 29 : Les paramètres d'une méthode	35
Figure 30 : Les associations d'une classe.....	36
Figure 31 : Le fichier XML crée.....	37
Figure 32 : La DTD.....	37
Figure 33 : Exemple d'une classe JAVA générée.....	38
Figure 34 : Exemple d'une classe CPP générée	38
Figure 35 : Exemple du code SQL généré	39





Liste des abréviations


MOR/ORM : *Le Mapping Objet Relationnel.*

XML : *Extensible Markup Language.*

DTD : *Description de Type de Document.*

UML: *Unified Modeling Language.*

DCL: *Diagramme De Classes.*






Résumé

En résumé, notre projet vise à simplifier et accélérer le processus de développement de logiciels en automatisant la tâche fastidieuse de création de code source et en établissant une correspondance entre le modèle objet et le modèle relationnel. Il s'adresse aux développeurs qui cherchent à optimiser leur processus de développement, en générant du code objet à partir de modèles UML dans un premier temps, et du code SQL correspondant aux modèles objets dans un second temps.

Pour cela nous avons suivi les différentes phases de développement d'un logiciel, à savoir l'analyse des besoins, la conception et l'implémentation. L'application a été modélisée selon le langage UML et développée en se basant sur les langages de programmation JAVA et XML.






Abstract

In summary, our project aims to simplify and speed up the software development process by automating the tedious task of creating source code and establishing a correspondence between the object model and the relational model. It is intended for developers who seek to optimize their development process, by generating object code from UML models first, and SQL code corresponding to object models second.

To do this, we followed the different phases of software development, namely needs analysis, design and implementation. The application was modeled according to the UML language and developed based on the JAVA and XML programming language



Introduction générale


Lors de la conception d'une application, les développeurs sont souvent confrontés au défi de stocker des données de manière persistante tout en les organisant de manière à pouvoir être récupérées et traitées ultérieurement. Les développeurs ont recours à deux approches principales : le modèle objet et le modèle relationnel.

Le modèle objet permet de représenter les entités de l'application sous forme d'objets, avec des attributs et des méthodes associés. Cette approche est largement utilisée dans la programmation orientée objet et permet de modéliser les entités de manière intuitive.

En revanche, le modèle relationnel stocke les données dans des tables, où chaque table représente une entité et chaque ligne représente une instance de cette entité. Cette approche est basée sur la théorie des ensembles et est largement utilisée dans les systèmes de gestion de bases de données relationnelles.

Bien que ces deux modèles aient des avantages distincts, leur incompatibilité est un problème majeur.

Pour résoudre ce problème, les développeurs ont développé une technique appelée mapping objet-relationnel (ORM), qui permet de faire correspondre les objets d'un modèle objet avec les tables d'un modèle relationnel. Cela facilite la persistance des données et permet aux développeurs de manipuler les données sous forme d'objets dans le code de l'application, tout en les stockant de manière persistante dans une base de données relationnelle. Cette technique a été largement adoptée par les développeurs de logiciels et de systèmes d'information pour faciliter l'utilisation de ces deux modèles de données ensemble.



Un autre concept très important dans le processus de développement logiciel est la génération de code, qui permet de créer automatiquement du code natif à partir de modèles. La génération de code source présente plusieurs avantages qui la rendent attrayante pour ses utilisateurs. Parmi les avantages d'une telle approche, on peut citer : le gain de temps, la réduction d'erreurs humaines et la réutilisation du code.

Pour mettre en œuvre ces concepts, nous avons développé une application desktop, permettant à ses utilisateurs de concevoir graphiquement des diagrammes de classes UML, qui seront automatiquement codés en XML. Ensuite, l'utilisateur peut générer des classes Java et C++ à partir de ce diagramme de classes et du code SQL correspondant au modèle objet déjà conçu selon un ensemble de règles de mapping qu'on a défini.

Le présent rapport s'articule donc autour des chapitres suivants :

- Chapitre 1 : Contexte général du projet
- Chapitre 2 : Analyse et conception
- Chapitre 3 : Etude technique et environnement
- Chapitre 4 : Réalisation



Chapitre 1 : Contexte général du projet

I. INTRODUCTION :

Ce premier chapitre décrit le contexte général du projet, ainsi qu'une description du projet commençant par la problématique, les objectifs fixés, la conduite et la planification des tâches à réaliser.


II. PRÉSENTATION DU PROJET

1. Contexte du projet

Le projet consiste à créer un outil de génération de code qui simplifie le processus de développement de logiciels en permettant aux développeurs de spécifier les fonctionnalités souhaitées et de générer automatiquement le code source correspondant. L'outil est conçu pour être facile à utiliser et pour permettre une génération rapide et précise du code source pour les langages de programmation JAVA et C++ et du code SQL selon un mapping O/R qu'on a défini.

2. Description du projet

Le projet consiste en la création d'un générateur de code qui permet de créer automatiquement du code Java et C++ à partir des diagrammes de classes UML, et du code SQL à partir des modèles objets.



L'interface utilisateur de l'outil est conçue pour permettre aux développeurs de spécifier facilement les concepts liés à la création d'un diagramme de classes, à savoir : les classes, les méthodes, les attributs, etc., ainsi que les associations possibles entre les classes.

Ensuite, L'outil génère automatiquement des classes Java et C++ correspondant au diagramme de classes conçu, ce qui permet d'économiser du temps et des efforts.

En outre, On a implémenté un autre générateur de code qui assure le Mapping O/R entre le diagramme de classes conçu et le modèle relationnel correspondant sous forme d'un code SQL.

Le projet est motivé par la nécessité de simplifier et d'accélérer le processus de développement de logiciels en automatisant la tâche fastidieuse de la création de code source et de faire une correspondance entre le modèle objet et le modèle relationnel. Il est destiné aux développeurs qui cherchent à optimiser leur processus de développement de logiciels.

3. Problématique du projet

La problématique de ce projet porte sur la correspondance entre les modèles conçus et le code qu'il faut implémenter, en particulier le Mapping O/R. En effet, l'implémentation du code n'est rien qu'une traduction des modèles conçus dans la phase de modélisation, et qui est une tâche complexe et fastidieuse à réaliser manuellement, ce qui peut entraîner des erreurs et des incohérences entre les modèles et le code écrit. Notre projet permet l'automatisation de cette tâche de manière efficace et fiable, en garantissant la cohérence et l'intégrité des données générées.



4. Objectifs

Parmi les objectifs fixés et les résultats attendus du projet, on peut mentionner :


4.1 Objectifs personnels

- Maîtriser et apprendre des nouvelles technologies et des nouveaux langages.
- Travailler en groupe et gérer un projet de manière professionnelle.
- Profiter des différentes techniques permettant de faciliter le travail

4.2 Objectifs professionnels

- Fournir un outil convivial et efficace pour la génération de classes Java à partir d'un diagramme de classes UML..
- Permettre aux utilisateurs de spécifier facilement les informations sur les classes telles que les attributs, les méthodes et les associations.
- Utiliser le concept de mapping objet relationnel pour garantir la cohérence entre les modèles conçus et le code généré.
- Prendre en charge la génération du code SQL pour faciliter l'interaction avec les bases de données.
- Prendre en charge la génération de code pour d'autres langages de programmation tels que CPP.
- Améliorer la qualité du code généré en garantissant la conformité aux bonnes pratiques de programmation.

5. Conduite de projet



La conduite de notre projet se déroule en trois phases. Dans un premier temps, nous avons défini le cadre global de l'application avec l'aide de notre encadrant. Nous avons ainsi spécifié les objectifs du projet et procédé à l'analyse des besoins, à la planification du travail et à la conception des interfaces souhaitables.

Ensuite, nous avons entamé la deuxième phase, qui consiste à la réalisation de tout ce qu'on a planifié dans la première phase. Nous avons commencé par l'écriture des premières lignes de code avec l'accompagnement de notre encadrant. Nous avons également organisé des réunions hebdomadaires pour faire le point sur le projet et planifier les prochaines tâches à réaliser

Enfin, nous avons procédé à la vérification du travail réalisé, en ajoutant les éléments manquants pour assurer la qualité et la complétude de notre projet. Tout au long de ces trois phases, nous avons travaillé en équipe en répartissant les tâches pour mener à bien ce projet.

6. Planification du projet

La planification est un processus qui consiste à anticiper le déroulement du projet en définissant les différentes phases qui vont constituer son cycle de vie. Elle permet de tracer les échéances temporelles tout en respectant les contraintes de délais et de ressources. Grâce à cette approche, toutes les tâches à mettre en œuvre au cours du projet ont été planifiées.

La figure ci-dessous représente l'enchaînement des différentes phases et étapes du projet.



Figure 1 : Diagramme de Gantt pour la gestion de projet

7. Conclusion

Ce chapitre introductoire nous a donné une idée sur le travail qu'il faut réaliser ainsi que les objectifs attendus, et qui sont en relation avec la génération de code qui constitue une approche primordiale dans le processus de développement logiciel. Dans le chapitre suivant, nous allons voir une analyse plus approfondie sur la problématique et le sujet traité.



Chapitre 2 : Analyse et conception

I. INTRODUCTION :

Ce chapitre vise à présenter la problématique, la solution MOR proposée, les besoins fonctionnels et non fonctionnels, ainsi que les modèles UML.

II. ANALYSES ET CONCEPTION :

1. Spécification des Besoins :

L'analyse des besoins est la phase de départ de toute application à développer dans laquelle on va identifier les besoins de l'application. On précise des besoins fonctionnels qui présentent les fonctionnalités attendues de l'application et les besoins non fonctionnels pour pouvoir clarifier les besoins des utilisateurs.

1.1. Besoins fonctionnels :

Cette partie arrive après une étude détaillée du système, elle est réservée à la description des exigences fonctionnelles des différents acteurs de l'application.

1. Saisie des informations des classes :

Les utilisateurs doivent pouvoir entrer les informations relatives aux classes telles que les noms des classes, les attributs et les méthodes associées.



2. Gestion des associations entre les classes :

L'application doit permettre de spécifier les associations et les relations entre les classes, telles que les relations d'héritage, de "composition", "d'agrégation" et " les associations simples".

3. Sérialisation des données en XML :

Une fois les informations des classes saisies, l'application doit être en mesure de générer un fichier XML correspondant à ce diagramme de classes, et qui stocke ces informations de manière structurée.

4. Génération des classes Java, CPP et du code SQL :

Le générateur de code doit utiliser le fichier XML créé, pour générer les classes Java et C++ conformément au diagramme de classes conçu. De plus, il doit être capable de générer du code SQL en respectant les règles de Mapping O/R qu'il faut définir.

5. Personnalisation des classes :

Le générateur doit permettre à l'utilisateur la mise à jour des classes générées en ajoutant des fonctionnalités supplémentaires, des annotations ou des commentaires.

6. Mapping objet-relationnel :

Le générateur doit prendre en charge le mapping objet-relationnel, en traduisant les informations des classes et des associations en structures de données compatibles avec une base de données relationnelle.



1.2. Besoins non fonctionnels :

1. Portabilité :

L'application doit être compatible avec différents environnements et systèmes d'exploitation, tels que Windows, Linux et macOS.

2. Performances :

La génération des classes Java, C++ et du code SQL doit être effectuée de manière efficace et rapide pour optimiser l'expérience de l'utilisateur.

3. Convivialité :

L'interface utilisateur de l'application doit être intuitive, conviviale et facile à utiliser, afin que les utilisateurs puissent facilement saisir les informations des classes et générer les résultats souhaités.

En spécifiant clairement ces besoins fonctionnels et non fonctionnels, nous nous assurons de développer une application qui répond aux attentes des utilisateurs tout en respectant les contraintes techniques et les exigences de performance.

2. Spécification de MOR :

Le mapping objet-relationnel (ORM) est une technique de programmation qui permet de faire correspondre des données stockées dans une base de données relationnelle avec des objets en programmation orientée objet (POO). Cela permet de travailler avec les données stockées dans la base de données en utilisant des objets en POO, simplifiant ainsi la manipulation de ces données.

2.1. Solution MOR proposée :

Il existe plusieurs techniques de MOR, mais nous-mêmes avons trouvé une solution simple et efficace pour établir une correspondance entre les classes UML et les tables SQL qui doivent être générées par notre générateur de code.



1. Présentation des classes en SQL :

CLASSE : En programmation orientée objet (POO), une classe est un concept fondamental qui permet de définir un modèle ou un plan pour la création d'objets. Une classe est une structure qui encapsule les données (attributs) et les comportements (méthodes) associés à un certain type d'objet. Elle sert de modèle à partir duquel des instances individuelles, appelées objets, peuvent être créées.

TABLE : En SQL, une table est une structure de données tabulaire qui permet de stocker et d'organiser des informations de manière structurée. Une table est composée de lignes et de colonnes, où chaque ligne représente un enregistrement individuel et chaque colonne représente un attribut ou une propriété de cet enregistrement.

Nous avons utilisé une fonction appelée "**printTablesFromXmlIntoSqlFile**" pour créer des tables correspondant aux classes générées. Veuillez consulter l'annexe pour voir la fonction

Modèle Object	Modèle Relationnel (SQL)
Une classe	Une table
Un attribut	Un champ
String	Varchar (50)
Int	Int
Boolean	Int
Double	Float
Char	Char

Table 1 : présentation d'une classe en SQL

2. Présentation de l'héritage en SQL :

L'héritage : est un concept fondamental en programmation orientée objet (POO) qui permet à une classe d'hériter les attributs et les méthodes d'une autre classe, appelée classe mère ou superclasse. La classe qui hérite de la classe mère est appelée classe enfant ou sous-classe.

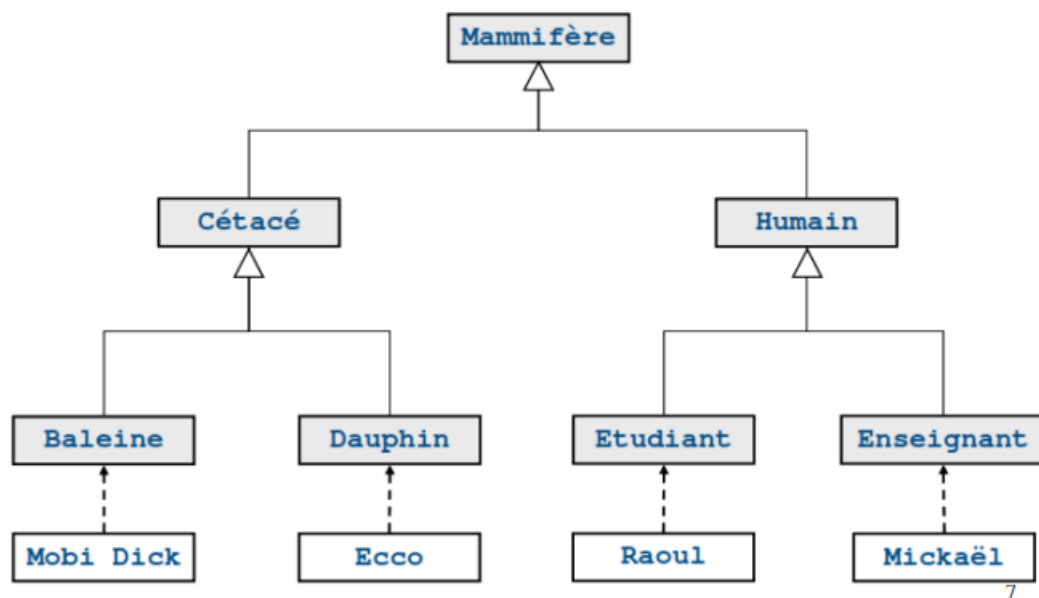


Figure 2 : Notion d'héritage

Dans une relation d'héritage, les deux classes seront représentées par deux tables, tandis que la relation d'héritage sera traduite par une clé étrangère au niveau de la table fille, comme illustré dans la figure 3. La fonction nommée "**printExtendIfWeHaveIt**" permet d'exprimer l'héritage en SQL. Veuillez consulter l'annexe pour plus de détails.

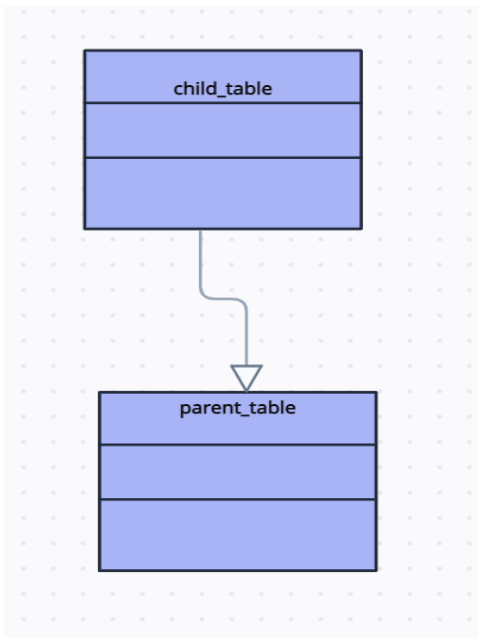


Figure 3 :L'héritage dans un DCL

```
create table Parent_table (  
    parentId int,  
    Primary key(parentId)  
);  
  
create table child_table (  
    childeId int references Parent,  
    Primary key(childtId)  
);  
Alter table child_table add constraint  
fk_child Foreign key(childeId)  
references Parent_table(parentId);
```

Figure 4 : Représentation de l'héritage en SQL

3. Présentation des associations en SQL :

La présentation des associations dans SQL était un vrai défi pour nous, mais après une analyse et une recherche approfondie, nous avons trouvé une solution simple et efficace pour faire correspondre nos classes générées avec les tables SQL sans aucun problème.

Les associations entre deux classes signifient qu'il existe une relation entre ces deux classes. Au niveau du code, cela signifie que l'une de ces classes, voire les deux, est déclarée comme attribut dans l'autre classe.

Cela nous permet de représenter ces attributs comme des champs des tables en SQL. En utilisant en plus la contrainte de clé étrangère, nous avons réussi à présenter les associations entre les classes en SQL en utilisant une fonction appelée **"printLesVariablesDautreTables"**. Veuillez consulter l'annexe pour voir plus de détails sur cette fonction.

- Association simple :

Supposons que nous ayons deux tables "Clients" et "Commandes". Un client peut passer plusieurs commandes, mais une commande ne peut être passée que par un seul client. Dans ce cas, l'association simple peut être représentée par la clé étrangère "client_id" dans la table "Commandes" qui fait référence à la clé primaire "id" dans la table "Clients".

Cette association est représentée de la manière suivante :

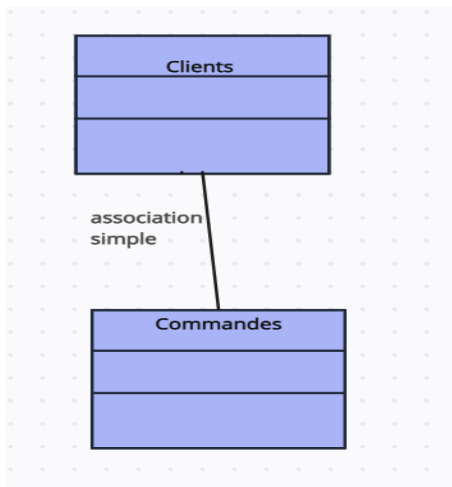


Figure 5 : Association simple dans un DCL

```
CREATE TABLE Clients (  
    id INT PRIMARY KEY,  
    nom VARCHAR(50),  
    adresse VARCHAR(100)  
);  
  
CREATE TABLE Commandes (  
    id INT PRIMARY KEY,  
    date_commande DATE,  
    client_id INT,  
    FOREIGN KEY (client_id) REFERENCES Clients(id)  
);
```

Figure 6 : Représentation d'association simple en SQL

- agrégation :

Supposons que nous ayons deux tables "Universités" et "Étudiants". Une université peut avoir plusieurs étudiants, mais un étudiant ne peut être inscrit que dans une seule université. Dans ce cas, l'agrégation avec la multiplicité peut être représentée par une troisième table "Inscriptions" qui fait référence à la fois à la table "Universités" et à la table "Étudiants".

Cette association est représentée de la manière suivante :

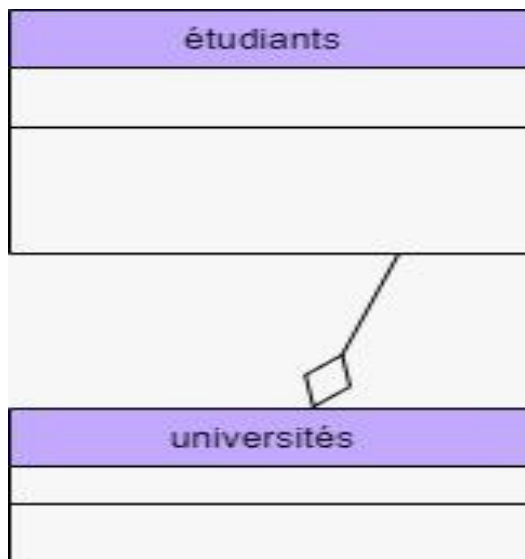


Figure 7 : Agrégation dans un DCL

```
CREATE TABLE Universités (
    id INT PRIMARY KEY,
    nom VARCHAR(50),
    ville VARCHAR(50)
);

CREATE TABLE Étudiants (
    id INT PRIMARY KEY,
    nom VARCHAR(50),
    date_naissance DATE
);

CREATE TABLE Inscriptions (
    universite_id INT,
    etudiant_id INT,
    date_inscription DATE,
    PRIMARY KEY (universite_id, etudiant_id),
    FOREIGN KEY (universite_id) REFERENCES Universités(id),
    FOREIGN KEY (etudiant_id) REFERENCES Étudiants(id)
);
```

Figure 8 : Représentation d'une Agrégation en SQL

- Composition :

Supposons que nous ayons deux tables "Voitures" et "Moteurs". Une voiture est composée d'un moteur, et un moteur ne peut être utilisé que dans une seule voiture. Dans ce cas, la composition avec la multiplicité peut être représentée par la clé étrangère "moteur_id" dans la table "Voitures" qui fait référence à la clé primaire "id" dans la table "Moteurs".

Cette association est représentée de la manière suivante :

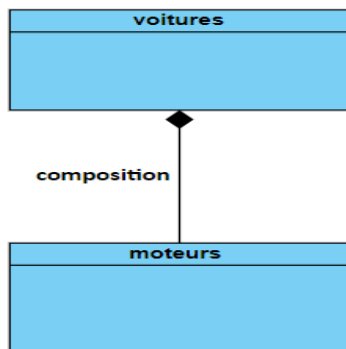


Figure 9 : Composition dans un DCL

```

CREATE TABLE Moteurs (
  id INT PRIMARY KEY,
  puissance INT,
  cylindres INT
);

CREATE TABLE Voitures (
  id INT PRIMARY KEY,
  modele VARCHAR(50),
  annee INT,
  moteur_id INT,
  FOREIGN KEY (moteur_id) REFERENCES Moteurs(id)
);
  
```

Figure 10 : exemple Composition en SQL

3. UML

UML : (Unified Modeling Language) est un langage de modélisation graphique utilisé pour représenter visuellement les concepts et les relations dans un système logiciel. Il fournit une notation standardisée pour faciliter la communication et la conception de logiciels.

2.2. Diagramme de cas d'utilisation :

Un diagramme de cas d'utilisation est une représentation visuelle des interactions entre les acteurs (utilisateurs) et le système logiciel. Il met en évidence les différentes actions ou fonctionnalités que les acteurs peuvent effectuer dans le système, ainsi que les relations entre ces actions. Le diagramme de cas d'utilisation permet de comprendre les besoins et les exigences des utilisateurs, et sert de base pour la conception du système.

- Identification des acteurs :

Un acteur désigne une personne, un matériel ou un logiciel qui interagit avec le système pour accomplir une ou plusieurs fonctionnalités liées aux cas d'utilisation. Dans le cadre de notre application nous avons identifié un seul acteur :

Utilisateur : L'utilisateur est l'acteur principal de notre générateur. Il interagit avec l'interface utilisateur pour fournir les informations nécessaires pour la conception de son diagramme de classes, telles que les attributs, les méthodes et les associations entre ces classes.

Après l'identification de l'acteur principal de notre système, nous allons représenter les différentes fonctionnalités que doit fournir notre application sous forme de cas d'utilisation, comme représenté dans la figure 11.

- **La présentation du diagramme:**

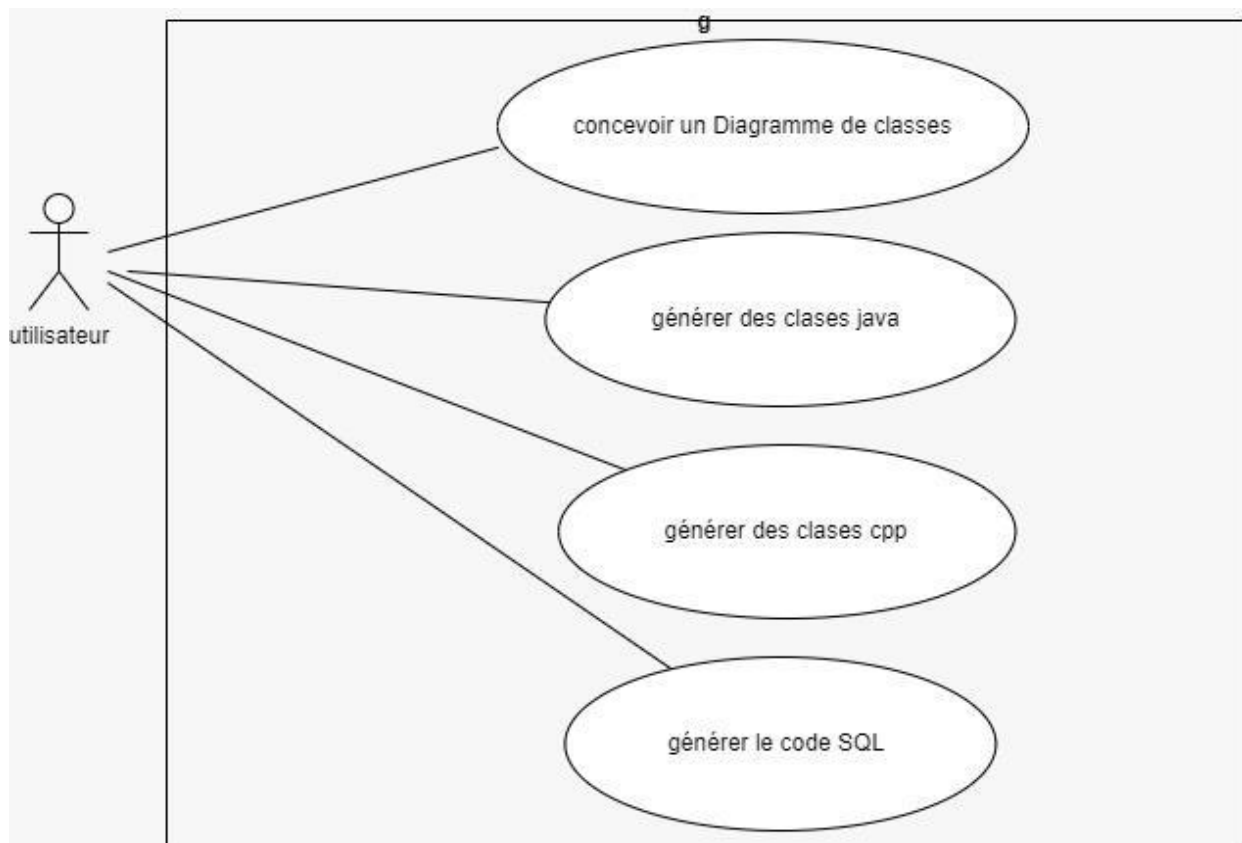


Figure 11 : Diagramme de cas d'utilisations

2.3. Diagramme de classes :

Le diagramme de classes est un outil de modélisation qui représente les classes, les attributs, les méthodes et les relations entre les classes d'un système logiciel. Il permet de visualiser la structure et les interactions entre les différentes entités du système, facilitant ainsi la conception, la communication et la compréhension du système.

- **La présentation du diagramme:**

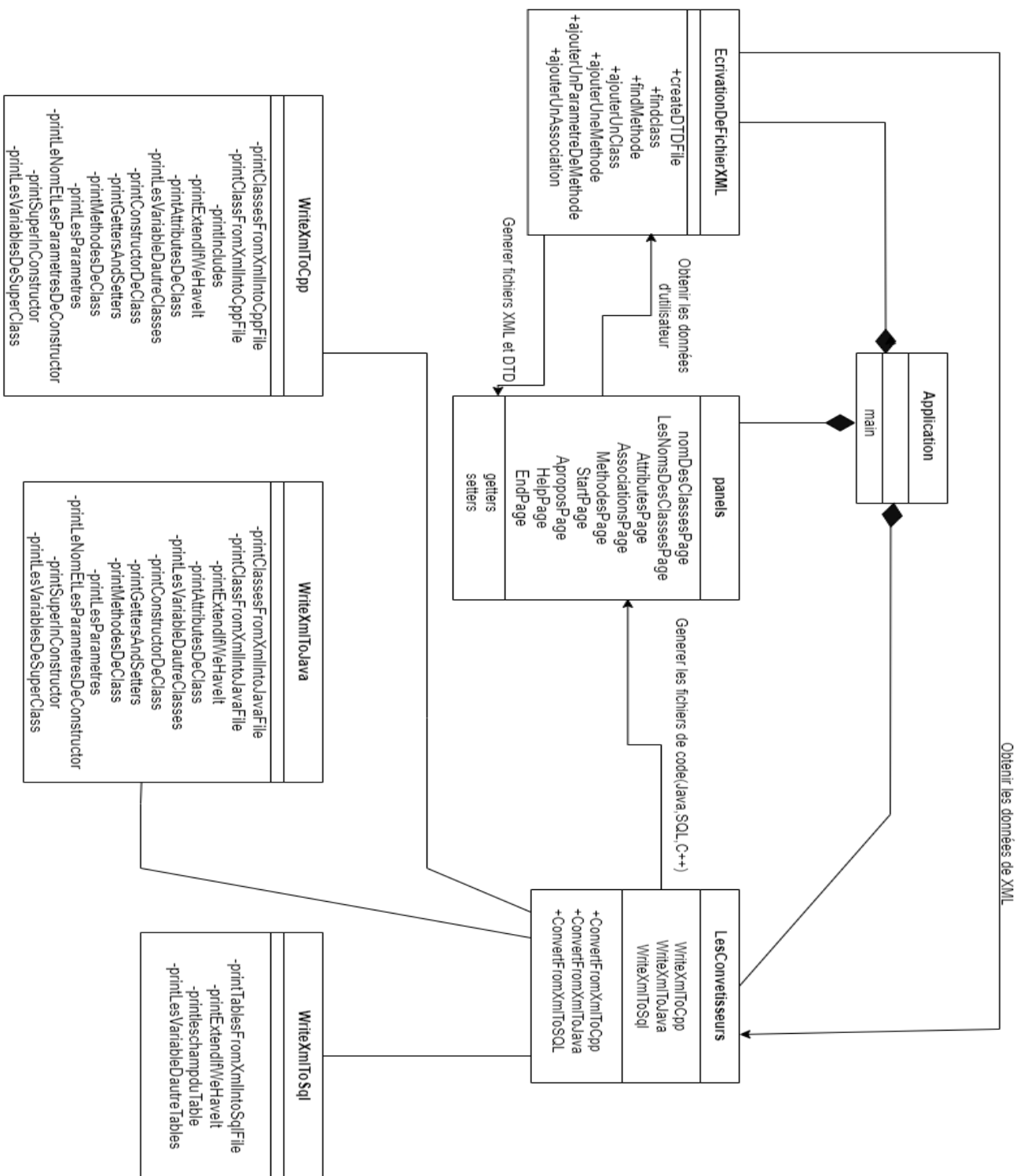


Figure 12 : Diagrammes de classes



2.4. Diagramme de séquences :

Un diagramme de séquence est un type de diagrammes UML (Unified Modeling Language) qui représente l'ordre chronologique des interactions entre les différents objets ou acteurs dans un système. Il illustre la façon dont ces objets ou acteurs communiquent entre eux en échangeant des messages dans un scénario donné.

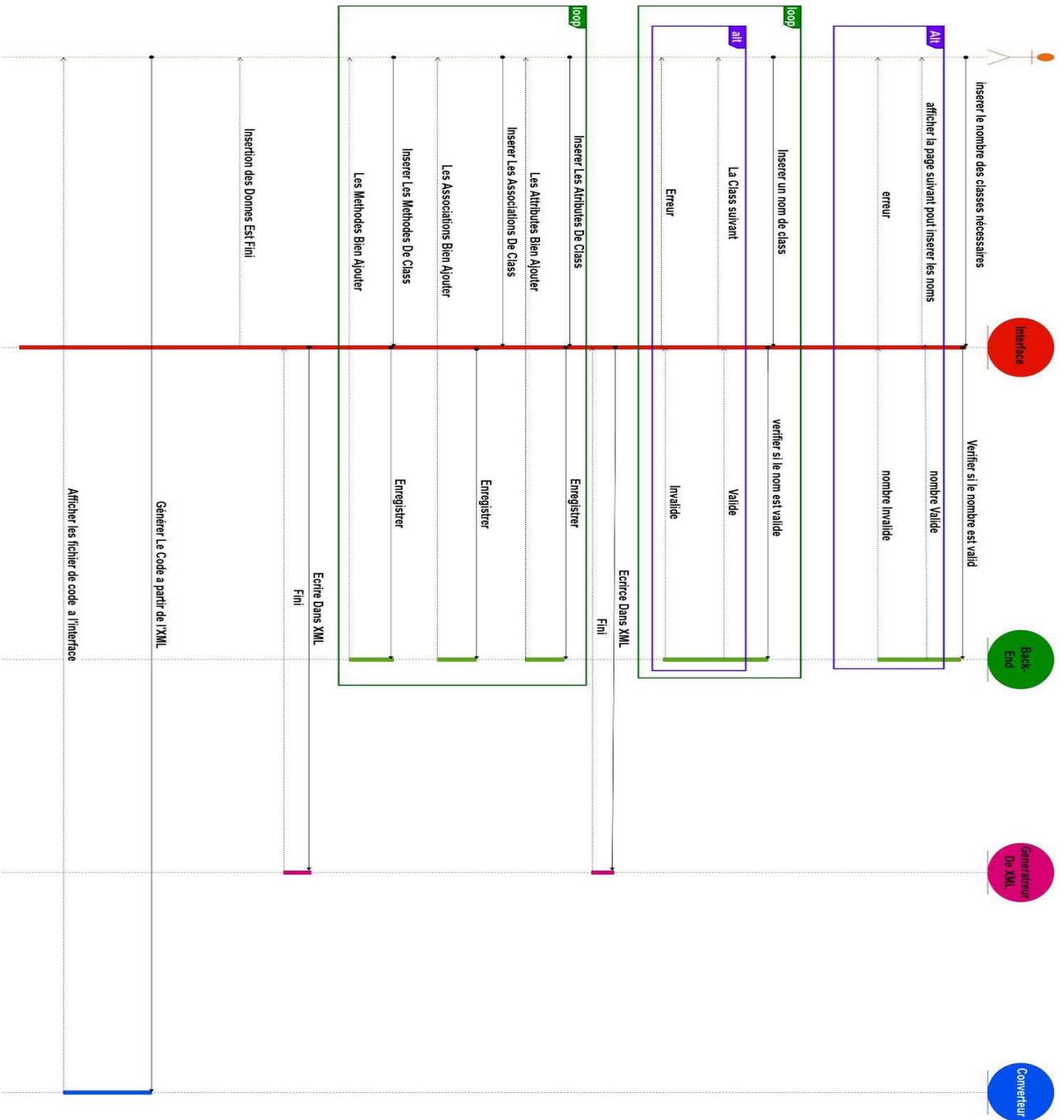


Figure 13 : Diagrammes de séquences

2.5. Diagramme de classes pour la DTD

La DTD définit la structure et les règles de validation pour un document XML spécifique. Elle spécifie les éléments qui peuvent être utilisés, les attributs associés à ces éléments, et les relations hiérarchiques entre eux.

Lorsqu'un document XML est créé, il sera associé à une DTD en utilisant une déclaration de type de document (DOCTYPE) dans l'en-tête du document XML. Cette déclaration indique au parseur XML quel type de DTD ou de schéma utiliser pour valider le document XML.

Lors de la validation, le parseur XML compare le document XML par rapport à la DTD associée. Il vérifie si le document respecte les règles de la DTD, notamment en termes de structure, d'éléments autorisés et d'attributs.

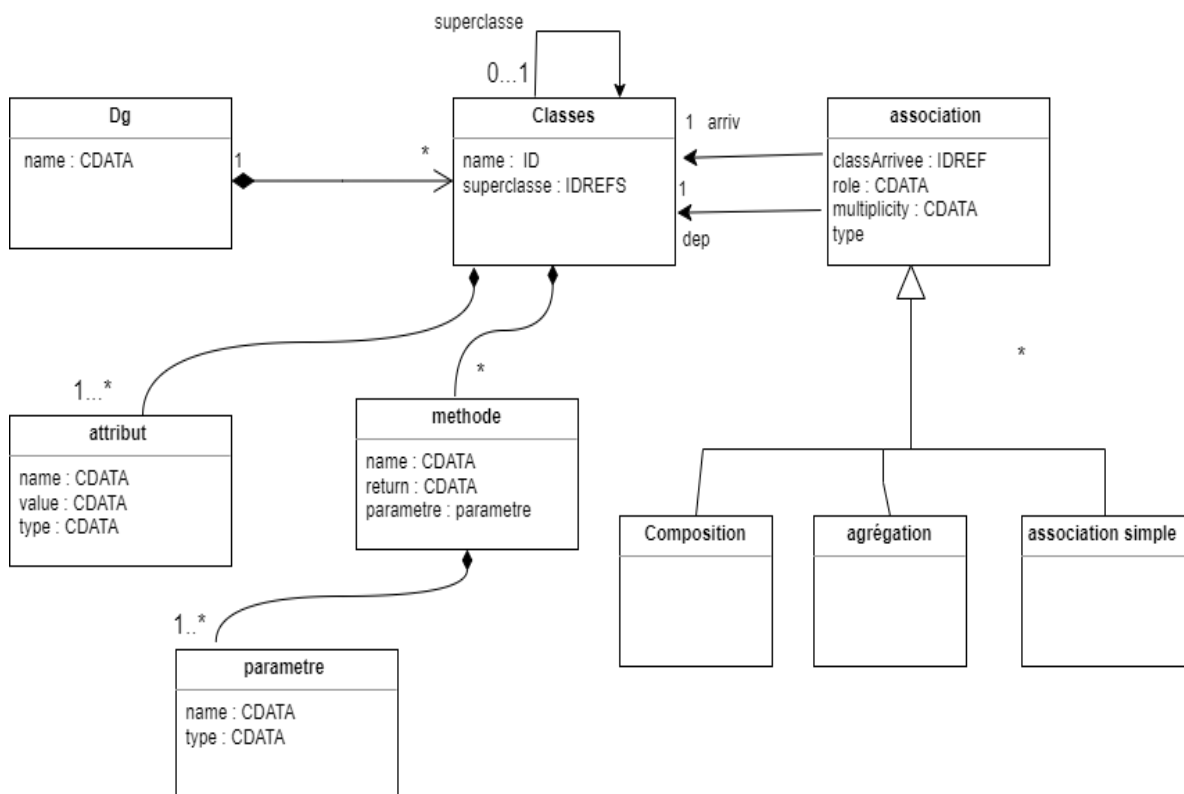


Figure 14 : diagramme de classes pour la DTD

Chapitre 3 : Etude technique et environnement

I. INTRODUCTION

Avant de passer à la phase d'implémentation de notre solution, il fallait présenter l'environnement de travail, ainsi que les technologies utilisées. Ce chapitre contient une description de l'ensemble des outils utilisés tout au long du projet.

II. CHOIX DES TECHNOLOGIES

1. JDOM



Figure 15 : la bibliothèque Java JDOM

Java JDOM est une bibliothèque open-source qui permet de manipuler des documents XML en Java de manière simple et efficace. JDOM offre une interface de programmation orientée objet pour créer, modifier et parcourir des documents XML.

Les avantages de JDOM par rapport à d'autres bibliothèques XML Java incluent sa simplicité, sa facilité d'utilisation, sa flexibilité et sa performance.

2. XML ET DTD

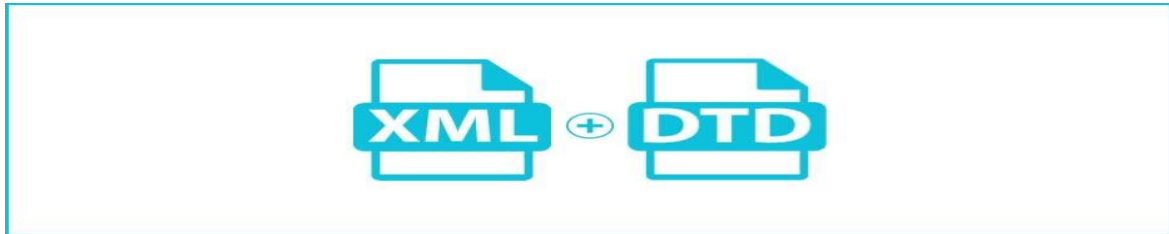


Figure 16 : XML ET DTD

2.1 Le Modèle et le Métamodèle

DTD (Document Type Definition) est une spécification utilisée pour définir la structure d'un document XML. Une DTD définit les éléments et les attributs que peut contenir un document XML, ainsi que les relations entre ces éléments. Les DTD permettent de valider la syntaxe et la structure des documents XML et d'assurer que les documents respectent un certain format.

Un modèle : est une représentation abstraite d'une réalité. Il peut être utilisé pour représenter des informations, des concepts ou des processus. Un modèle peut être défini comme un ensemble de règles qui définissent la structure et le comportement d'un système. Il peut être utilisé pour représenter des informations, des concepts ou des processus.

Un métamodèle : est une représentation abstraite d'un modèle. Il sert à décrire les caractéristiques générales d'un modèle et à spécifier comment le modèle est construit et utilisé. Un métamodèle est un outil puissant qui permet aux concepteurs de créer des modèles plus efficaces et plus précis.



2.2 XML et DTD

La DTD (Document Type Definition) : est un métamodèle qui définit la structure et le contenu d'un document XML (Extensible Markup Language). La DTD spécifie les éléments XML autorisés, leur ordre et leur contenu, ainsi que les attributs associés à chaque élément. La DTD est un outil très utile pour vérifier la validité du format XML d'un document avant son traitement par un programme informatique.

XML (Extensible Markup Language) : est un langage de balisage conçu pour stocker et transporter des données structurées entre différents systèmes informatiques. XML est considéré comme un modèle car il fournit une syntaxe standard pour la description de données structurée, ce qui permet aux applications logicielles de communiquer entre elles sans avoir à connaître les détails techniques du format de donnée utilisé par l'auteur



III. LANGAGES ET OUTILS DE DÉVELOPPEMENT

1. Les langages de programmation

- **JAVA :**

Java est un langage de programmation populaire créé par Sun Microsystems (maintenant acquis par Oracle Corporation). Il est conçu pour être portable, c'est-à-dire qu'il peut être exécuté sur différents types d'ordinateurs et de systèmes d'exploitation sans avoir besoin de modifications importantes. Java utilise une machine virtuelle Java (JVM) qui interprète le code Java et le compile en code natif pour le système d'exploitation spécifique. Cette approche rend le code Java portable et sécurisé, car le code est exécuté dans un environnement isolé qui ne peut pas accéder directement au système d'exploitation. Java est utilisée pour développer une variété d'applications, y compris des applications de bureau, des applications web et des applications mobiles. Il est également largement utilisé pour développer des applications d'entreprise en raison de sa sécurité et de sa capacité à gérer de grands volumes de données.

2. Les langages de présentation

- **JAVA SWING :**

Java Swing est une bibliothèque graphique populaire et robuste pour le développement d'interfaces utilisateur dans les applications Java. Elle offre une grande flexibilité, une grande portabilité et une grande personnalisation, et elle est soutenue par une grande communauté de développeurs qui peuvent aider les utilisateurs à résoudre les problèmes et à améliorer leurs compétences en matière de développement d'interfaces utilisateur. Java Swing a été introduit avec la version 1.2 de Java et est disponible dans toutes les versions ultérieures.

3. Les outils de développement

- Visual Studio Code

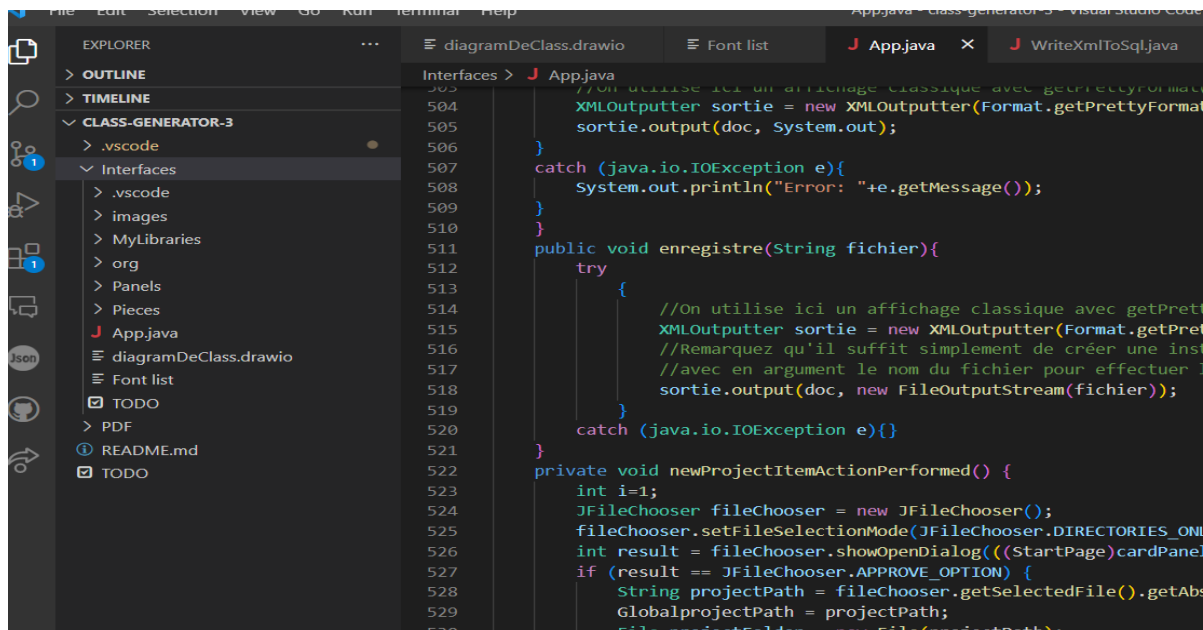


Figure 17 : Visual Studio Code

Visual Studio Code est un environnement de développement intégré puissant et complet créé par Microsoft. Il offre une variété de fonctionnalités et de capacités pour faciliter le développement d'applications pour une variété de plateformes. Visual Studio est également livré avec une variété de langages de programmation et de frameworks, et est soutenu par une grande communauté de développeurs.

- **GITHUB**

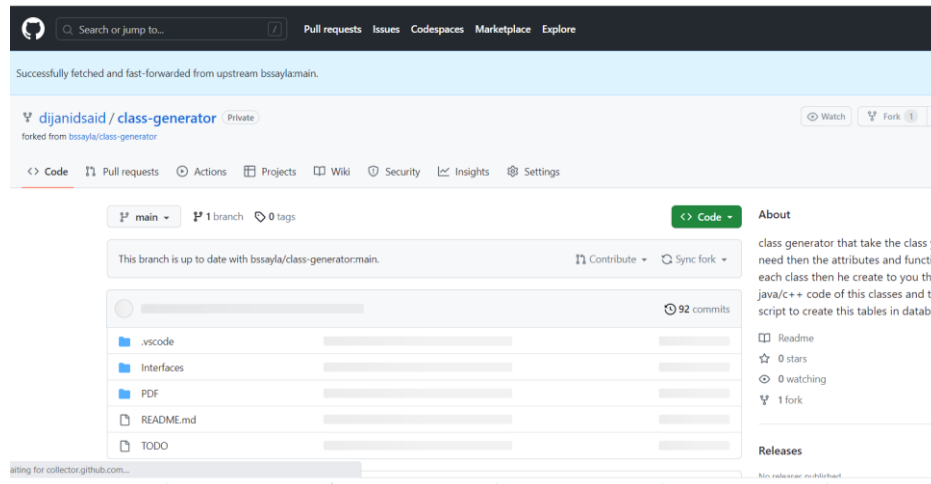


Figure 18 : GITHUB

GitHub est une plateforme de développement collaborative de logiciels basée sur Git, qui permet aux développeurs de travailler ensemble sur des projets de manière transparente et de gérer leur code source. La plateforme offre une variété d'outils pour aider les développeurs à travailler plus efficacement, et est également une communauté active de développeurs open source qui partagent des projets, des idées et des conseils sur la programmation et le développement de logiciels.

- **GIT**



Figure 19 : GIT

Nous avons utilisé GITHUB pour échanger du code entre nous de manière efficace et professionnelle, en particulier grâce à la technique Git. Git est facile à utiliser et donne les résultats souhaités, ce qui a grandement facilité la collaboration et la gestion du code source.

4. Les outils de Conception

Pour créer les diagrammes UML ainsi le diagramme de Gantt nous avons utilisés Des sites en ligne par exemple

- **LUCIDCHART**

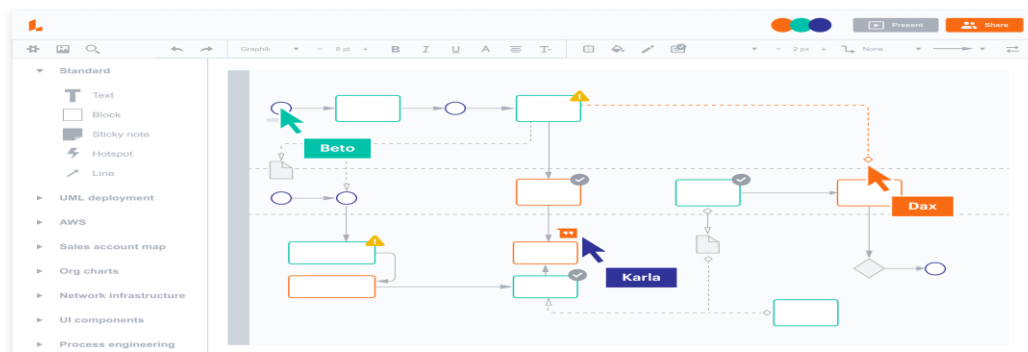



Figure 20 : LUCIDCHART



Lucidchart est une plateforme en ligne de création de diagrammes professionnels qui offre une interface utilisateur intuitive, des fonctionnalités avancées de création de diagrammes, une collaboration en temps réel, des options d'exportation de diagrammes et une intégration transparente avec d'autres outils de productivité

5. Les outils de Design

Pour choisir les couleurs de notre application et pour créer notre logo ainsi pour créer des icônes nous avons utilisés des sites en ligne par exemple :

- **FREEPIK**

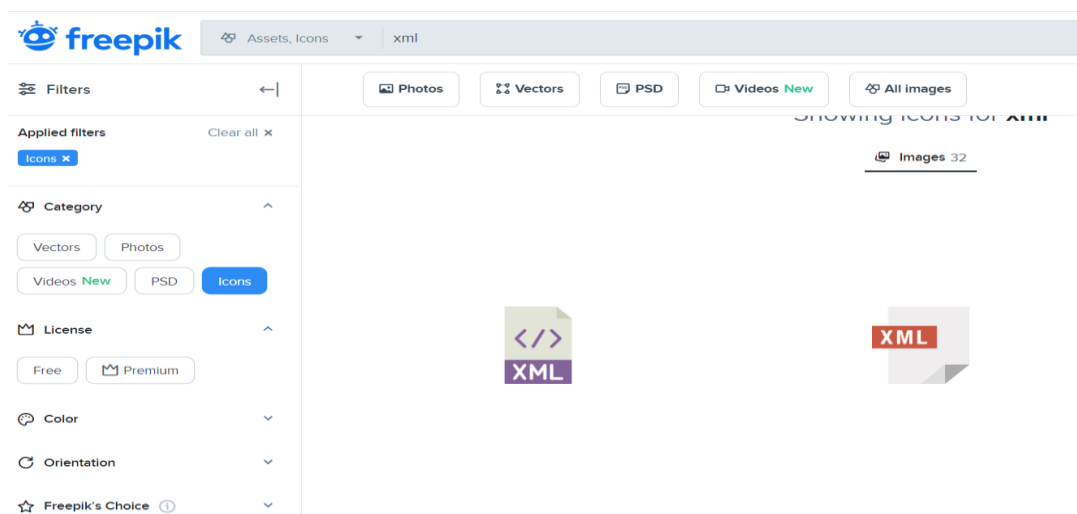


Figure 21 : FREEPIK

Freepik est une plateforme de téléchargement de ressources graphiques tels que des images, des illustrations, des icônes et qui offre des ressources gratuites et des abonnements payants, une grande variété de ressources créées par des designers du monde entier, ainsi qu'une plateforme de conception en ligne appelée "Editor". La plateforme est conçue pour aider les utilisateurs à trouver des ressources graphiques de qualité, à créer des designs professionnels et à économiser du temps et des ressources dans leur travail quotidien.



6. Conclusion

En résumé, ce chapitre a été consacré à la présentation des outils et langages qu'on a utilisés dans la réalisation de notre projet, Ceux-ci vont permettre par la suite de développer une application forte, pertinente et fiable dans le long terme. L'utilisation des bonnes méthodes et outils est la clé pour réaliser un bon projet.

Chapitre 4 : Réalisation

I. INTRODUCTION

Une fois la partie de la conception achevée, tous les éléments nécessaires au développement de l'application deviennent disponibles. Dans ce chapitre nous allons illustrer le détail de réalisation et l'implémentation des différentes interfaces de notre application.

II. LES INTERFACES DE L'APPLICATION

Cette partie présente l'application réalisée dans le cadre de notre projet de fin d'études. On va la présenter sous forme de captures d'écrans.

1. La page de garde :

Voici la page d'accueil qui propose deux options : créer un nouveau projet ou ouvrir un projet déjà existant.

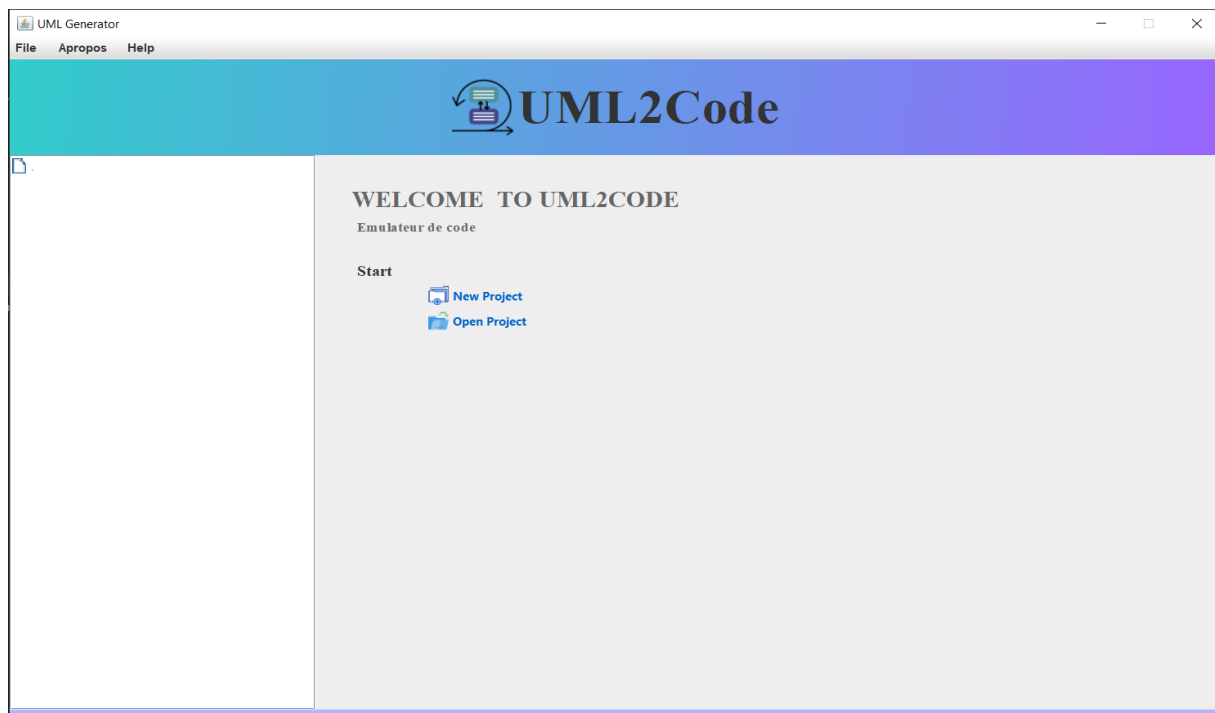


Figure 22 : page d'accueil

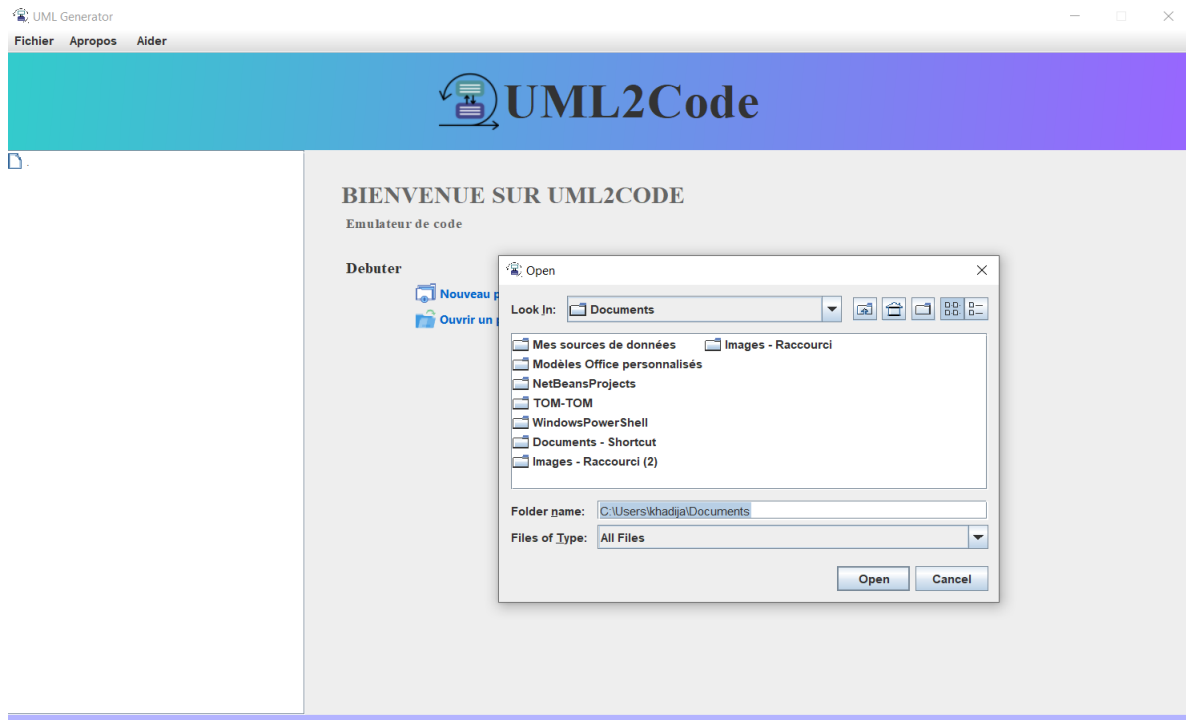


Figure 23 : nouveau projet

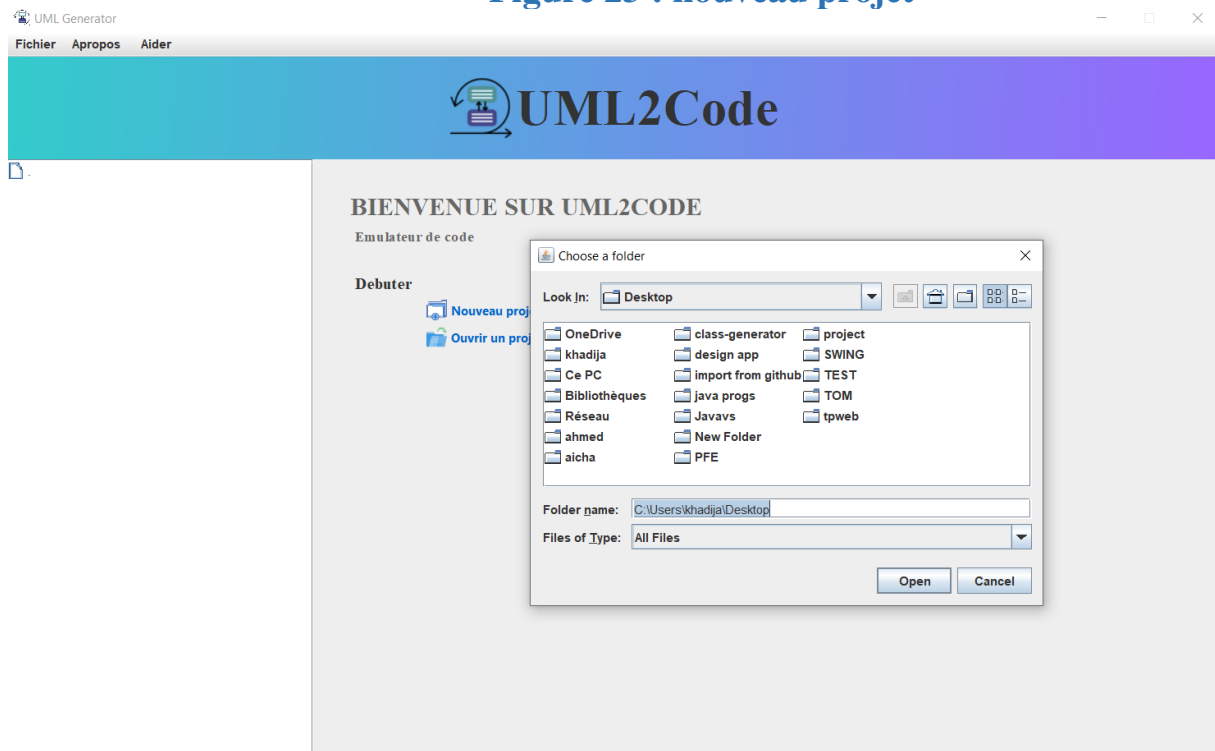


Figure 24 : ouvrir un projet existant

2. Saisir les informations des classes

L'utilisateur doit remplir toutes les informations concernant les classes, y compris leurs attributs, leurs méthodes et leurs associations avec d'autres classes, en précisant tous les détails obligatoires.

1.1. Le nombre de classes et leurs noms

L'utilisateur doit entrer le nombre de classes qu'ils souhaitent créer ainsi que leurs noms :



The screenshot shows the UML2Code application window. The title bar reads 'UML Generator' with menu items 'Fichier', 'Apropos', and 'Aider'. The main header is a blue bar with the 'UML2Code' logo. On the left, a sidebar shows a file explorer with 'project'. The main area has a light blue background with the text 'Nombre de classes' and a text input field containing the number '2'. A green 'soumettre' button is at the bottom right. A status bar at the bottom left says 'Nombre valide' and 'Next pour continue'. A green 'Suivant' button is at the bottom right of the status bar.

Figure 25 : nombre des classes



The screenshot shows the UML2Code application window at the 'Les noms du classes' screen. The title bar and header are the same as in Figure 25. The sidebar shows 'project-6'. The main area has a light blue background with the text 'Les noms du classes'. It features two input fields: 'Nom du classe :' with the text 'classe2', and 'Super Class' with a dropdown menu showing 'classe1'. A green 'Ajouter' button is at the bottom right. A status bar at the bottom left contains the text: 'Le nombre des Classes est 2', 'la classe classe1 a été ajoutée :', and 'la classe classe2 a été ajoutée, sa Classe parent est :classe1'. A green 'Suivant' button is at the bottom right of the status bar.

Figure 26 : Les noms des classes

1.2. Les attributs d'une classe

L'utilisateur doit entrer le nom de l'attribut, son type et sa valeur s'il existe

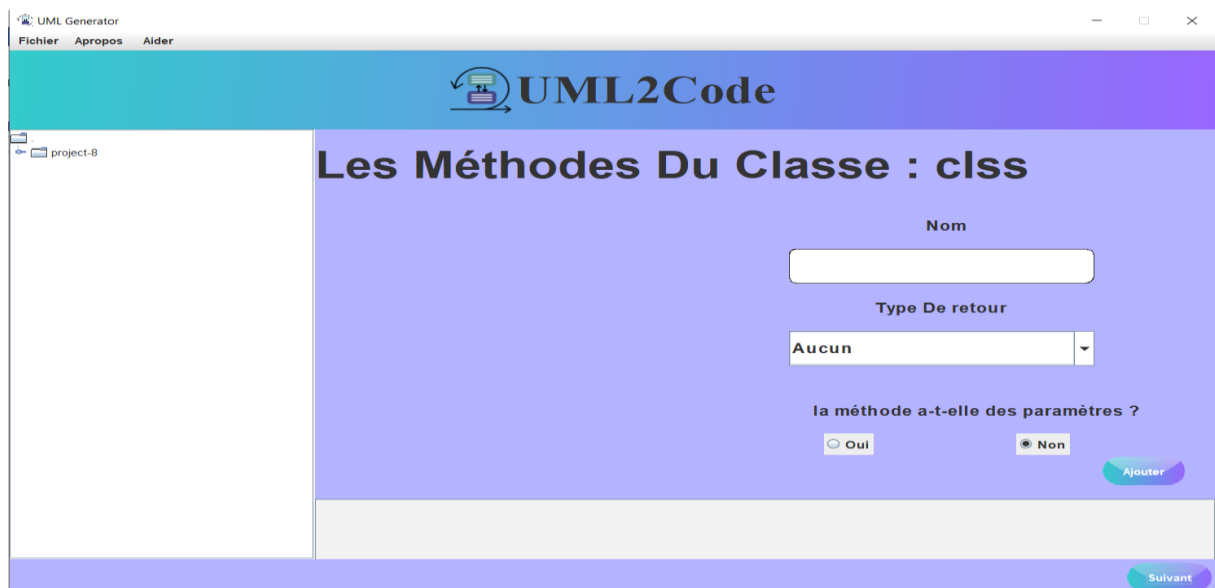


The screenshot shows the 'UML2Code' application window. The title bar reads 'UML Generator' with menu items 'Fichier', 'Apropos', and 'Aider'. The main interface has a teal header with the 'UML2Code' logo. On the left, a sidebar shows a file explorer with 'project-6'. The main area is titled 'Les Attributs Du Classe : classe1'. It contains three input fields: 'Nom' (containing 'attribut1'), 'Type' (a dropdown menu showing 'Entier'), and 'Valeur' (empty). There is an 'Ajouter' button at the bottom right of the main area and a 'Suivant' button at the bottom right of the window.

Figure 27 : Les attributs d'une classe

1.3. Les méthodes d'une classe

L'utilisateur doit saisir le nom de la méthode, son type de retour, en précisant s'il comporte des paramètres ou non. Si oui, l'utilisateur doit ensuite les saisir.



UML Generator
Fichier Apropos Aider

UML2Code

project-8

Les Méthodes Du Classe : clss

Nom

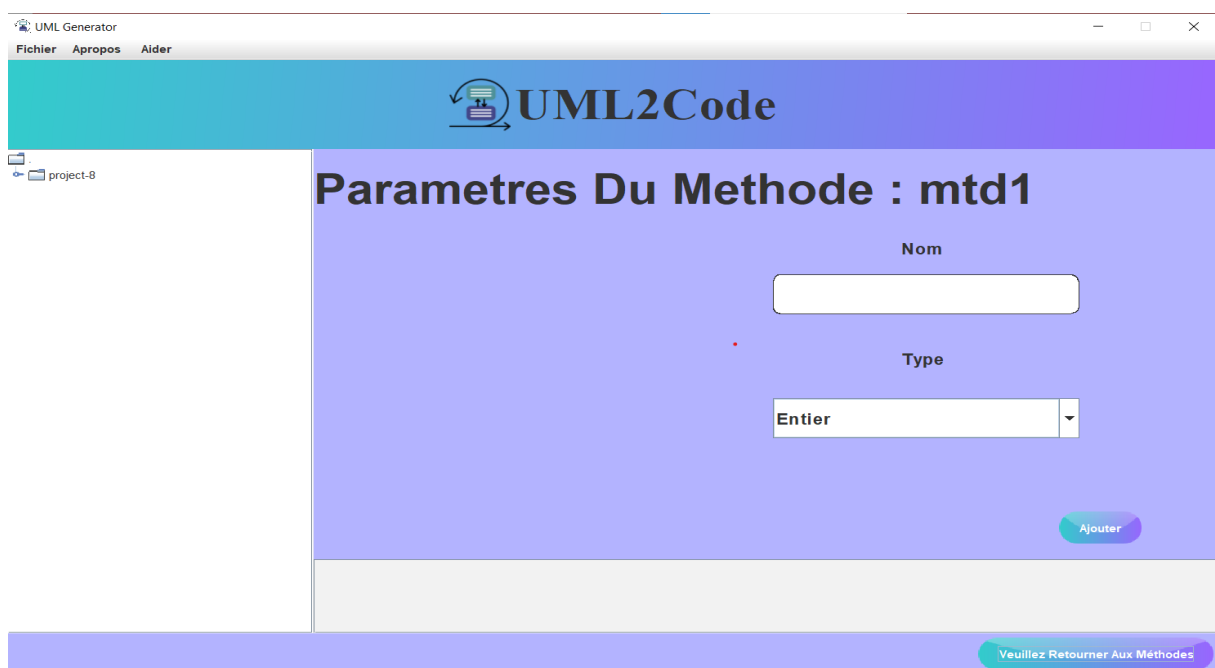
Type De retour
Aucun

la méthode a-t-elle des paramètres ?
☐ Oui ☒ Non

Ajouter

Suivant

Figure 28 : Les méthodes d'une classe



UML Generator
Fichier Apropos Aider

UML2Code

project-8

Parametres Du Methode : mtd1

Nom

Type
Entier

Ajouter

Veillez Retourner Aux Méthodes

Figure 29 : Les paramètres d'une méthode

1.4. Les associations d'une classe

L'utilisateur doit spécifier le type de l'association, la classe d'arrivée (puisque la classe de départ est celle en cours), ainsi que le rôle et la multiplicité.

The screenshot shows the 'UML2Code' web application interface. The title bar indicates 'UML Generator' with menu options 'Fichier', 'Apropos', and 'Aider'. The main header features the 'UML2Code' logo. On the left, a sidebar shows a project tree with 'project-6'. The main content area is titled 'Les Associations Du Classe : classe1'. It contains several form fields: a 'Type' dropdown menu set to 'aggregation', a 'Class D'arriver' dropdown menu set to 'classe1', an empty 'Role' text input field, and a 'Multiplicity' section with a radio button for '1' and a '1' value. At the bottom right, there are two buttons: 'Ajouter' and 'Suivant'.

Figure 30 : Les associations d'une classe

1.5.Affichage des résultats

Une fois que toutes les informations ont été saisies, le générateur crée les fichiers correspondants, à savoir le fichier XML et sa DTD, ainsi que les fichiers Java, C++, et SQL

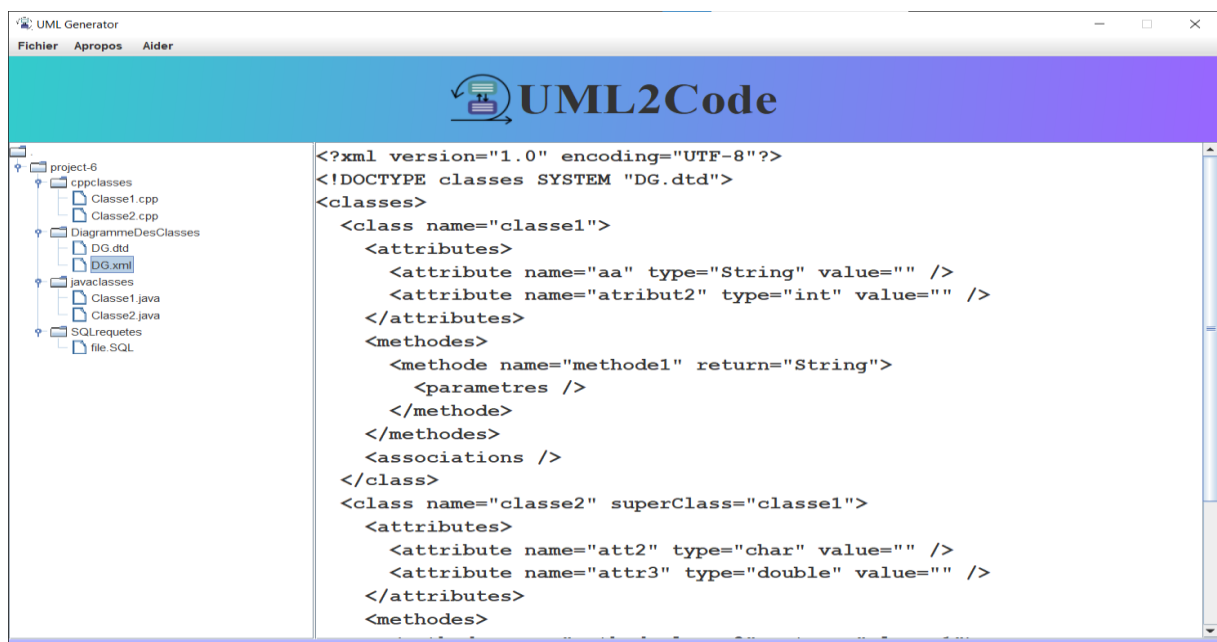


Figure 31 : Le fichier XML créé

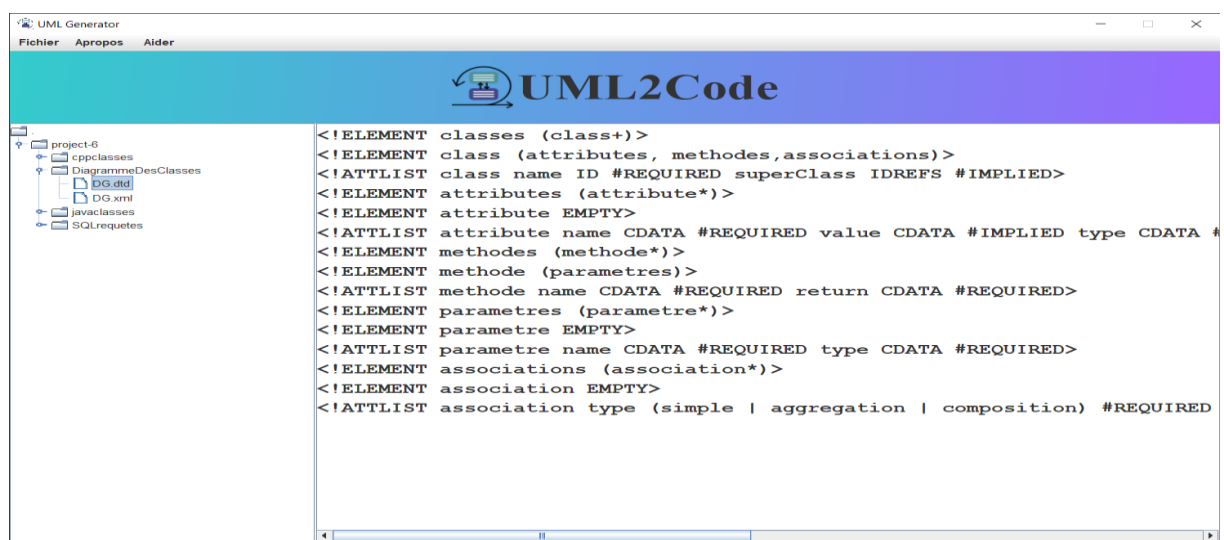


Figure 32 : La DTD

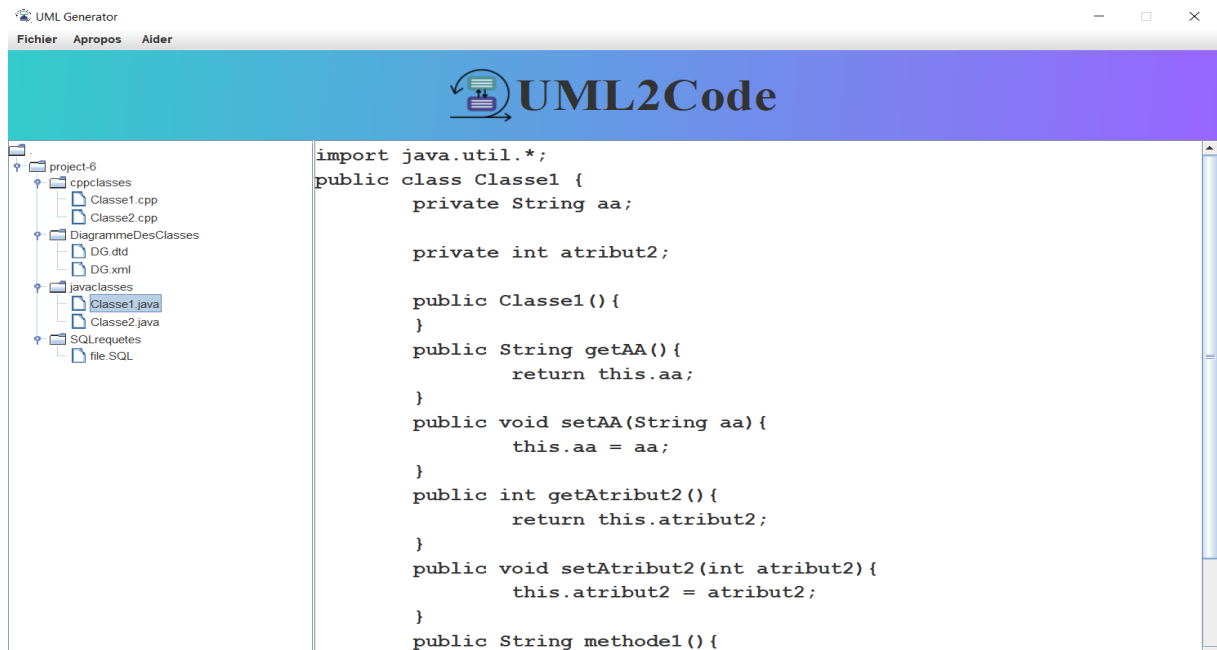


Figure 33 : Exemple d'une classe JAVA générée

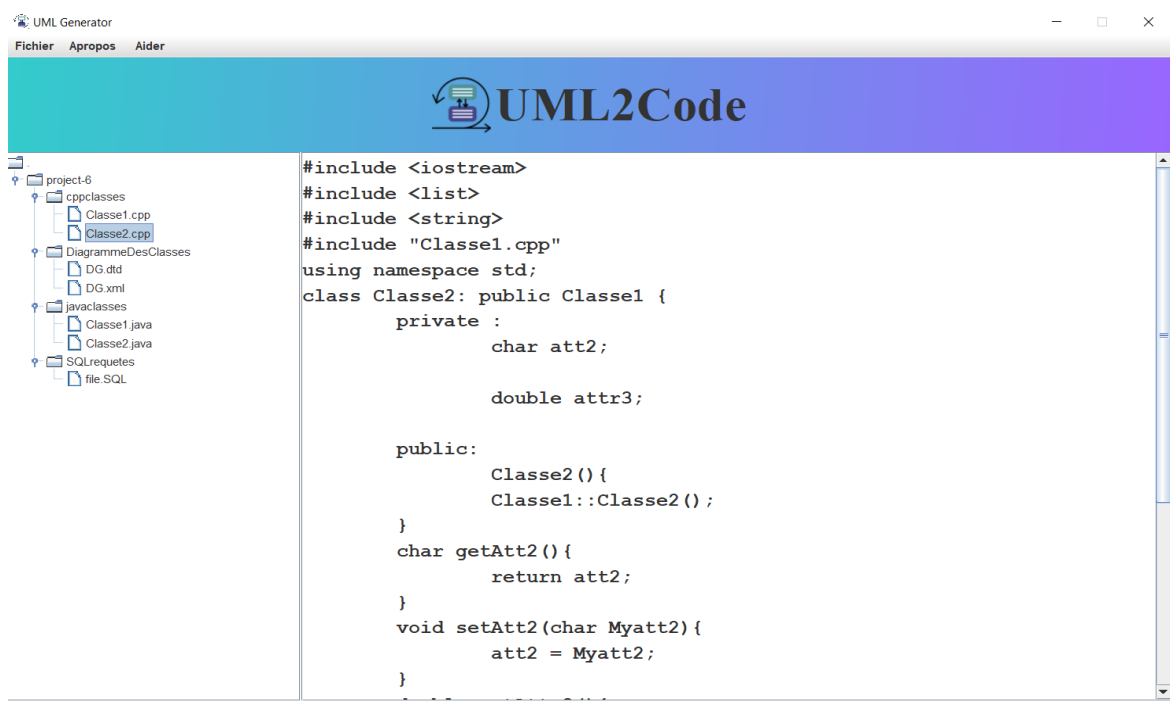


Figure 34 : Exemple d'une classe CPP générée

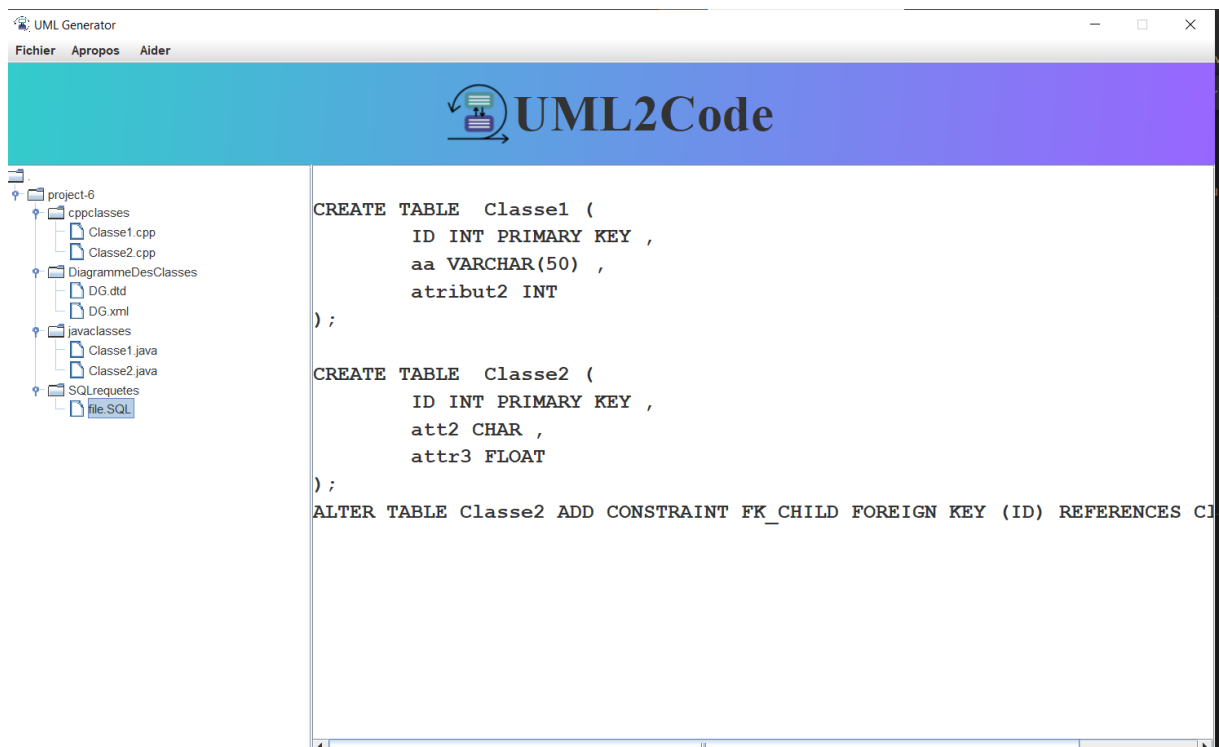


Figure 35 : Exemple du code SQL généré



Conclusion Générale et perspectives

En conclusion, le générateur de code développé dans ce projet offre une solution pratique et efficace pour simplifier et accélérer le processus de développement de logiciels. En automatisant la création de code source et en établissant une correspondance entre le modèle objet et le modèle relationnel, cet outil permet aux développeurs de générer automatiquement, à partir des diagrammes de classes conçus, des classes Java et C++, ainsi du code SQL conformément au mapping O/R qu'on a défini. Cela permet d'économiser du temps et des efforts.

Grâce à une interface utilisateur conviviale, les développeurs peuvent créer leurs diagrammes de classes facilement, offrant ainsi une personnalisation et une flexibilité accrues.

Pour les perspectives futures, il serait intéressant d'envisager l'extension de cet outil pour prendre en charge d'autres langages de programmation couramment utilisés. En élargissant la portée du générateur de code, il deviendrait encore plus polyvalent et répondrait aux besoins d'un plus grand nombre de développeurs.

De plus, l'amélioration continue de l'interface utilisateur, l'ajout de fonctionnalités avancées telles que la génération automatique de tests unitaires ou l'intégration avec des frameworks de développement populaires sont également des pistes à explorer. Cela permettrait d'optimiser davantage le processus de développement de logiciels et d'offrir une expérience utilisateur encore plus fluide.



Bibliographie

<https://www.base-de-donnees.com/orm/>

<https://javabydeveloper.com/orm-object-relational-mapping/>

<https://cynober.developpez.com/tutoriel/java/xml/jdom/#LII-C>

<https://www.tresfacile.net/le-mapping-objet-relationnel-orm/>

<https://chat.openai.com/>

<https://lucid.app/user/182708968#subscriptionLevel/lucidchart>

<https://fr.freepik.com/>

<https://www.geeksforgeeks.org/association-composition-aggregation-java/>

<https://blog.devgenius.io/association-composition-and-aggregation-in-c-925465987061>

<https://bard.google.com/>

Annexes

I. Les Relations Entre les classes en JAVA

1- Association Simple :

Supposons nous avons deux classes, la classe "Bank" (banque) et la classe "Employee" (employé).

L'association entre les deux classes, la classe "Bank" a une liste d'objets "Employee" comme l'un de ses attributs. Cela représente une relation de un à plusieurs(1..n), où une banque peut avoir plusieurs employés. L'attribut "employees" dans la classe "Bank" vous permet de stocker et de gérer une collection d'employés associés à une banque particulière.

La solution en code java pour présenter cette relation sera comme suite :

```
// Class 1
// Class Bank
class Bank {
    // Attributes de bank
    private String name;
    private list<Employee> employees;
    // Constructeur de bank
    Bank(String name)
    {
        this.name = name;
    }
    //getters and setters
    public String getBankName()
    {
        return this.name;
    }
    public void setEmployees(list<Employee> employees)
    {
        this.employees = employees;
    }
    public list<Employee>
    getEmployees(list<Employee> employees)
    {
        return this.employees;
    }
}
```



```
// Class 2
// Employee class
class Employee {
    // Attributs de Employee
    private String name;
    //constructeur de Employee
    Employee(String name)
    {
        this.name = name;
    }
    //getters and setters
    public String getEmployeeName()
    {
        return this.name;
    }
}
```

2-Agrégation:

Supposons nous avons deux classes, la classe « Department » et la classe « Etudiant ». L'agrégation entre les deux classes, la classe "Department" agrège un ensemble d'objets "Etudiant". Cela signifie qu'un département contient plusieurs étudiants. L'attribut "Etudiants" dans la classe "Department" vous permet de stocker et de gérer une collection d'objets "Etudiant" associés à un département particulier.

Et la classe "Department" a un constructeur qui prend l'attribut "Etudiants" dans ses paramètres pour initialiser

```
// Class 1
// Etudiant class
class Etudiant {
    // les attributs de la classe Etudiant
    String name;
    int id;
    String dept;
    // Constructor
    Etudiant(String name, int id, String dept)
    {
        this.name = name;
    }
}
```

```

        this.id = id;
        this.dept = dept;
    }
}
// Class 2
// un département contient plusieurs étudiants

class Department {
    // les attributs de classe Departement
    String name;
    private List<Etudiant> Etudiants;
    Department(String name, List<Etudiant> Etudiants)
    {

        this.name = name;
        this.Etudiants = Etudiants;
    }
    // getters and setters
    public List<Etudiant> getEtudiants()
    {
        return Etudiants;
    }
}

```

3 - Composition:

Supposons nous avons deux classes, la classe « Car » et classe « Engine » La composition entre les deux classes, la classe "Car" possède un objet "Engine" en tant qu'attribut. Cela signifie que la voiture est composée d'un moteur. L'objet "Engine" est créé et géré par la classe "Car". Dans ce cas, lorsque vous créez une instance de la classe "Car", un moteur de type "V8" est automatiquement créé et associé à cette instance.

Dans le constructeur, il y a une composition entre la classe "Car" et la classe "Engine". Un moteur de type "V8" est créé en interne et assigné à l'attribut "engine" de la voiture.

```

class Engine{
    string type;

    public Engine(string type) {
        this.type = type;
    }
}

```

```

    public string getType() {
        return type;
    }
}
class Car{
    string name;
    Engine engine;

    public Car(string name) {
        this.name = name;
        this.engine = new Engine("V8");
    }
    public string getName() {
        return name;
    }
    public Engine getEngine() {
        return engine;
    }
}

```

II. Les fonctions présentant l'héritage et les associations en SQL


- La fonction “printExtendIfWeHaveIt”:

```

private boolean printExtendIfWeHaveIt(List<Element> listDesClasses ,FileWriter
JavaFile){
    try{
        for (Element monClass : listDesClasses) {
            String superClass = monClass.getAttributeValue("superClass");
            String className = monClass.getAttributeValue("name");

            if(superClass == null){
                continue;
            }
            if(!superClass.equals("")){
                JavaFile.write("ALTER TABLE "+capitalize(className)+" ADD
CONSTRAINT FK_CHILD FOREIGN KEY (ID) REFERENCES " +capitalize(superClass)+"(ID);
\n");
            }
        }
    }
}

```



```
        return true;
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```

Explication De la Fonction:

La méthode “printExtendIfWeHaveIt” prend une liste d'objets “Element” et un objet FileWriter comme paramètres. Le but de cette méthode est de générer du code SQL qui ajoute des contraintes de clé étrangère à une table de base de données en fonction de la hiérarchie d'héritage d'un ensemble de classes.

La méthode parcourt chaque objet “Element” de la liste des classes et récupère le nom de la classe et sa superclasse. Si la superclasse est nulle ou une chaîne vide, la méthode passe à la classe suivante. Sinon, la méthode génère du code SQL qui ajoute une contrainte de clé étrangère à la table correspondant à la sous-classe. La contrainte de clé étrangère fait référence à la colonne ID de la table de la superclasse.

Si la méthode génère avec succès le code SQL, elle renvoie true. Si une exception est levée pendant le processus, la méthode intercepte l'exception, affiche le message d'erreur sur la console et renvoie false.

- La fonction “ printLesVariableDautreClasses ” :


```
private void printLesVariableDautreClasses(Element monClass ,FileWriter JavaFile){
    List<Element> listDesAssociation = monClass.getChildren(cname:"associations").get(0).getChildren(cname:"association");
    String classDArrivee,multiplicity;
    if(listDesAssociation.size() == 0){
        return;
    }
    for (Element association : listDesAssociation) {
        classDArrivee = association.getAttributeValue(attname:"classArrivee");
        multiplicity = association.getAttributeValue(attname:"multiplicity");

        if(multiplicity.equals("1")){
            try{
                JavaFile.write("\t"+"private "+capitalize(classDArrivee)+" "+classDArrivee.toLowerCase()+";"+"\\n");
            }catch(Exception e){
                System.out.println(e.getMessage());
            }
        }else{
            try{
                JavaFile.write("\t"+"private List<"+capitalize(classDArrivee)+"> "+classDArrivee.toLowerCase()+"s;"+"\\n");
            }catch(Exception e){
                System.out.println(e.getMessage());
            }
        }
    }
}
```

Explication De la Fonction:

La méthode `printLesVariableDautreClasses` qui prend en paramètres un objet `Element` et un objet `FileWriter`. Le but de cette méthode est de générer du code Java qui déclare des variables d'instance pour les associations entre classes dans un schéma XML.

La méthode récupère une liste d'objets `Element` qui représentent les associations entre la classe actuelle et d'autres classes. Si la liste est vide, la méthode revient sans générer de code. Sinon, la méthode parcourt chaque association et récupère le nom de la classe de destination et la multiplicité de l'association.



Si la multiplicité est "1", la méthode génère du code Java qui déclare une variable d'instance privée de type classe destination avec le même nom que la classe destination. Si la multiplicité n'est pas "1", la méthode génère du code Java qui déclare une variable d'instance privée de type "Liste" avec le type de classe de destination comme paramètre générique et le nom de la classe de destination au pluriel comme nom.

Si la méthode génère avec succès le code Java, elle passe à l'association suivante. Si une exception est levée pendant le processus, la méthode intercepte l'exception, affiche le message d'erreur sur la console et passe à l'association suivante.

Pour améliorer la lisibilité de ce code, il serait utile d'ajouter des commentaires qui expliquent le but de chaque section du code. De plus, le nom de la méthode pourrait être amélioré pour mieux refléter sa fonctionnalité. En termes de performances, il peut être plus efficace d'utiliser un objet `StringBuilder` pour concaténer le code Java au lieu d'écrire dans l'objet `FileWriter` pour chaque association.

- La fonction “ `printTablesFromXmlIntoSqlFile` ” :

```
private boolean printTablesFromXmlIntoSqlFile(List<Element> listdesclasses, FileWriter JavaFile ) {

    try{

        //FileWriter JavaFile = new FileWriter("file.sql" );
        for (int i = 0; i < listdesclasses.size(); i++) {
            Element monClass = (Element) listdesclasses.get(i);

            String className = monClass.getAttributeValue(attname:"name");

            JavaFile.write("\nCREATE TABLE "+capitalize(className));
            JavaFile.write(" ("+"\\n");
            JavaFile.write("\\tID INT PRIMARY KEY ,\\n");
            printlesChampsDuTable(monClass, JavaFile);

            JavaFile.write("\\n");
            JavaFile.write(");"+"\\n");
        }
    } catch (Exception e )
    {
        System.out.println(e.getMessage());
        return false;
    }
    return true;
}
```

Explication De la Fonction:

Il s'agit d'une méthode Java appelée `printTablesFromXmlIntoSqlFile` qui prend une liste de classes et un objet `FileWriter` comme paramètres. Le but de cette méthode est de générer du code SQL qui crée des tables de base de données basées sur la structure d'un schéma XML. La méthode parcourt chaque objet `Element` de la liste des classes et récupère le nom de la classe. Il génère ensuite du code SQL qui crée une table portant le même nom que le nom de la classe. La table a une colonne ID qui sert de clé primaire et des colonnes supplémentaires sont générées en fonction des champs de la classe. La méthode appelle une autre méthode appelée `printlesChampsDuTable` pour générer le code SQL des champs.

Si la méthode génère avec succès le code SQL, elle renvoie `true`. Si une exception est levée pendant le processus, la méthode intercepte l'exception, affiche le message d'erreur sur la console et renvoie `false`.

- La fonction “`printlesChampsduTable`” :

```
private void printlesChampsDuTable(Element monClass, FileWriter JavaFile) {  
  
    List<Element> listDesAtributtes = monClass.getChildren(cname:"attributes").get(0).getChildren(cname:"attribute");  
    for (int i = 0; i < listDesAtributtes.size(); i++) {  
        Element monAtributte = (Element) listDesAtributtes.get(i);  
        String name = monAtributte.getAttributeValue(attname:"name");  
        String type = monAtributte.getAttributeValue(attname:"type");  
        try{  
            if(type.equals("String")){  
                JavaFile.write("\t" +name+" " + "VARCHAR(50) ");  
            }else if(type.equals("int") || type.equals("boolean")){  
                JavaFile.write("\t" +name+" " + "INT ");  
            }else if(type.equals("char")){  
                JavaFile.write("\t" +name+" " + "CHAR ");  
            }else if(type.equals("double")){  
                JavaFile.write("\t" +name+" " + "FLOAT ");  
            }  
            if(i<listDesAtributtes.size()-1){  
                JavaFile.write(", \n");  
            }  
        }catch(Exception e){  
        }  
    }  
}
```



Explication De la Fonction:

La méthode `printlesChampsDuTable` prend en paramètres un objet `Element` et un objet `FileWriter`. Le but de cette méthode est de générer du code SQL qui crée des colonnes pour une table de base de données en fonction des champs d'une classe dans un schéma XML.

La méthode récupère une liste d'objets `Element` qui représentent les attributs de la classe. Il parcourt ensuite chaque attribut et récupère son nom et son type. En fonction du type de l'attribut, la méthode génère un code SQL qui crée une colonne portant le même nom que l'attribut et un type de données correspondant au type de l'attribut.

Par exemple, si le type de l'attribut est "String", la méthode génère du code SQL qui crée une colonne avec le nom de l'attribut et un type de données `VARCHAR(50)`. Si le type de l'attribut est "int" ou "booléen", la méthode génère du code SQL qui crée une colonne avec le nom de l'attribut et un type de données `INT`. Si le type de l'attribut est "char", la méthode génère du code SQL qui crée une colonne avec le nom de l'attribut et un type de données `CHAR`. Si le type de l'attribut est "double", la méthode génère du code SQL qui crée une colonne avec le nom de l'attribut et un type de données `FLOAT`.

Si la méthode génère avec succès le code SQL, elle passe à l'attribut suivant. Si une exception est levée pendant le processus, la méthode intercepte l'exception et passe à l'attribut suivant.

Pour améliorer la lisibilité de ce code, il serait utile d'ajouter des commentaires qui expliquent le but de chaque section du code. De plus, le nom de la méthode pourrait être amélioré pour mieux refléter sa fonctionnalité. En termes de performances, il peut être plus efficace d'utiliser un objet `StringBuilder` pour concaténer le code SQL au lieu d'écrire dans l'objet `FileWriter` pour chaque attribut.