OUTIL DE RÉTRO CONCEPTION UML

Dossier programmeur

Année universitaire 2025-2026

- Donovan Prévost
- Paul Gricourt
- Nicolas Delpech
- William Millereux Bienvault
- Erwan Martin

Table des matières

1. Introduction **5**
1.1 Contexte du projet 5
1.2 Objectifs pédagogiques 5
1.3 Technologies utilisées 5
2. Architecture Générale **6**
2.1 Fonctionnalités principales 6
Niveaux d'analyse implémentés 6
Fonctionnalités clés 6
2.2 Contraintes techniques 6
3. Architecture technique **7**
3.1 Diagramme d'architecture globale 7
3.2 Structure des packages 8
4. Lecture et analyse de fichiers Java **8**
Principe général 8
Fichier principal impliqué 9
Logique de lecture mise en place 9
1. Point d’entrée unique via une factory 9
2. Initialisation de la structure UML cible 9
3. Lecture séquentielle du fichier 10
4. Identification de la déclaration de classe 10
5. Extraction des membres de la classe 11
7. Préparation des relations UML 11
Résultat de la fonctionnalité 11
5. Gestion des fichiers et Calcul des liens/multiplicités **12**
Objectif général 12
Principe global de fonctionnement 12
Lecture et centralisation des fichiers Java 12
Fichier concerné : LectureRepertoire 12
Rôle de LectureRepertoire 13
Logique de traitement 13
Représentation métier d’une classe Java 13
Fichier concerné : CreeClass 13
Responsabilité de CreeClass 13
Création des liens UML entre classes **14**
Fichier concerné : Lien 14
Principe général 14
Types de relations détectées 14
1. Associations par attributs 14

2. Relations d'héritage 15
3. Implémentation d'interfaces 15
Calcul des multiplicités UML 15
Fichier concerné : Multiplicite 15
Principe général 15
Logique de calcul 16
Détermination d'une multiplicité 16
Construction des paires de multiplicités 16
Stockage et exploitation 17
6. Mise en place de l'affichage (GUI) 17
Objectif général 17
Fichiers impliqués et responsabilités 17
PanneauPrincipal – Support central de rendu graphique 17
Rôle : 17
Responsabilités principales : 18
Logique globale de fonctionnement 18
Étape 1 – Récupération du modèle UML 18
Étape 2 – Calcul et dessin des classes 18
Étape 3 – Construction des relations graphiques 19
Héritage 19
Interface 19
Associations (avec multiplicités) 19
Étape 4 – Dessin des flèches 20
7. Affichage du Dessin des relations UML (flèches, multiplicités, rôles) 21
Logique globale de construction des flèches 21
1. Héritage et interfaces 21
2. Associations avec multiplicités 22
Logique suivie 22
Résultat 22
Logique détaillée du dessin des flèches 22
Calcul des points d'ancrage 22
Gestion du décalage 23
Dessin selon le type UML 23
Affichage des multiplicités 23
Gestion des rôles 24
8. Fonctionnalité : Modification / Édition des associations UML 24
1. But de la fonctionnalité 24
2. Organisation générale de la fonctionnalité 25
3. Sélection de la classe à modifier (Panneau de choix) 25
4. Affichage et modification des associations (Panneau d'information) 26

4.1 Compréhension des relations affichées 26
4.2 Construction dynamique de l'interface 26
4.3 Modification des multiplicités 27
5. Validation et enregistrement des modifications 27
Ce qui se passe à ce moment-là 27
5.1 Mise à jour des relations inverses 28
6. Modification du rôle d'une association 28
8. Fonctionnalité : Enregistrement et chargement des projets UML 29
1. Objectif de la fonctionnalité 29
2. Emplacement dans l'architecture du projet 29
3. Principe général de la persistance 29
4. Sauvegarde et chargement binaire (.ser) 30
Rôle du format binaire 30
Logique générale 30
5. Sauvegarde et chargement texte (.uml) 30
Pourquoi un format texte ? 30
6. Chargement d'un projet depuis un fichier texte 31
Logique globale du chargement 31
6.1 Reconstruction des classes 31
6.2 Héritage et interfaces 31
6.3 Attributs et méthodes 32
6.4 Chargement des rôles 32
7. Reconstruction des relations et multiplicités 32
8. Sauvegarde dans le fichier texte 33
Logique de la sauvegarde 33

1. Introduction

1.1 Contexte du projet

Ce projet s'inscrit dans le cadre de la SAÉ 3.01 (Situation d'Apprentissage et d'Évaluation) dont l'objectif est de concevoir et réaliser un outil de rétro-conception UML à partir de code source Java.

L'application permet d'analyser des fichiers ou répertoires Java pour en extraire la structure des classes (attributs, méthodes, relations) et générer automatiquement un diagramme de classes UML correspondant.

1.2 Objectifs pédagogiques

- Maîtriser l'analyse syntaxique de code source
- Implémenter les concepts du modèle UML
- Développer une interface graphique interactive
- Gérer la persistance des données
- Travailler en équipe avec gestion de version

1.3 Technologies utilisées

- **Langage** : Java 17
- **Interface graphique** : Swing
- **Parsing** : Scanner et expressions régulières
- **Persistance** : Sérialisation Java + format texte personnalisé
- **Outils** : Git

2. Architecture Générale

2.1 Fonctionnalités principales

Niveaux d'analyse implémentés

1. **Niveau 1** : Analyse simple d'une classe (attributs et méthodes)
2. **Niveau 2** : Affichage avec formalisme UML
3. **Niveau 3** : Analyse multiple avec associations
4. **Niveau 4** : Héritage et interfaces
5. **Niveau 5** : Interface graphique complète

Fonctionnalités clés

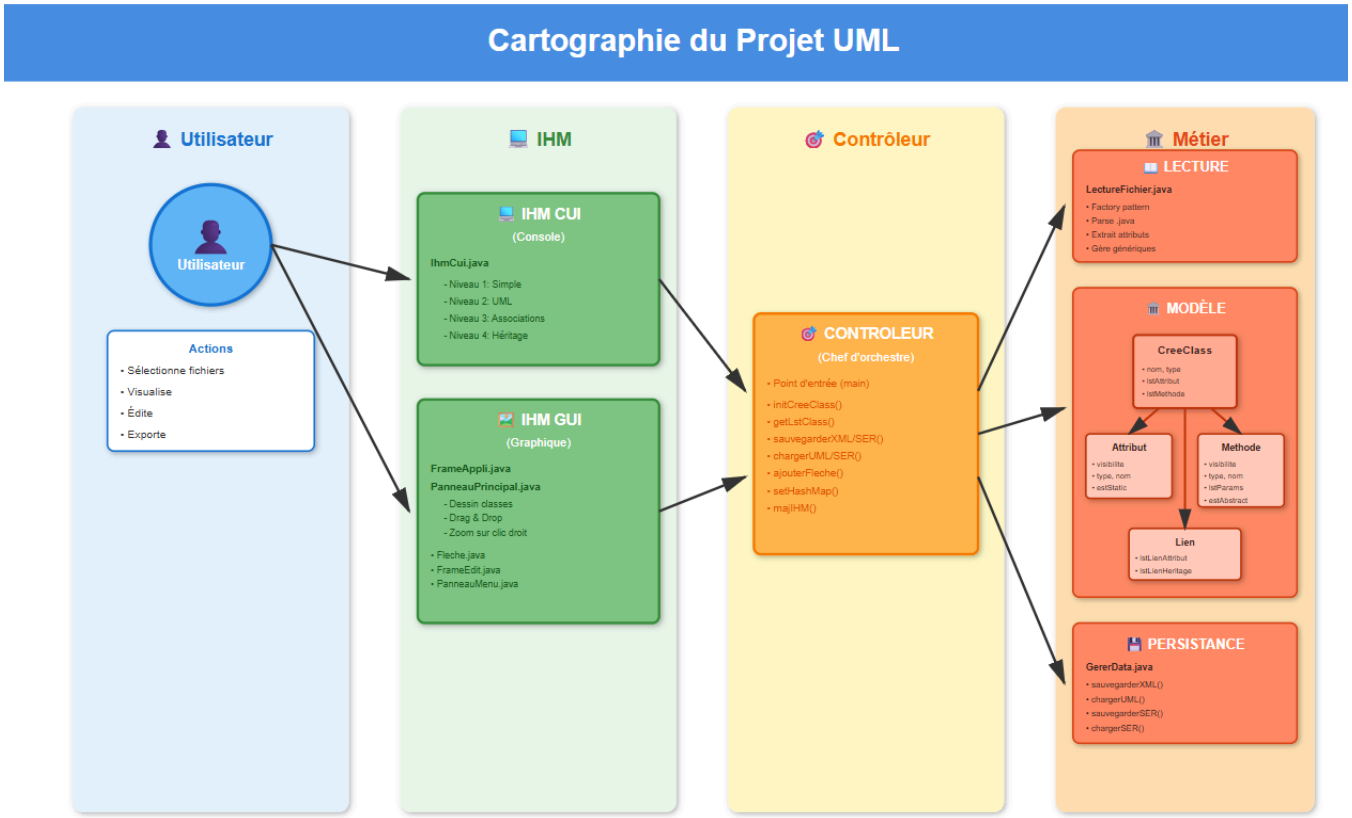
- ✅ Analyse de fichiers .java individuels
- ✅ Détection automatique des relations UML
- ✅ Calcul des multiplicités
- ✅ Interface graphique interactive
- ✅ Édition des multiplicités et rôles
- ✅ Sauvegarde/chargement de projets
- ✅ Export en image PNG
- ✅ Zoom et déplacement de classes

2.2 Contraintes techniques

- Gestion des modificateurs Java (public, private, protected, static, final, abstract)
- Support des types génériques (List<T>, Map<K,V>)
- Gestion des records
- Ignorer les énumérations (non supportées)

3. Architecture technique

3.1 Diagramme d'architecture globale



3.2 Structure des packages

```
src/
├── Controleur.java              # Point d'entrée principal
├── metier/                      # Classes métier
│   ├── Attribut.java
│   ├── CreeClass.java
│   ├── Lien.java
│   ├── Methode.java
│   ├── Multiplicite.java
│   ├── LectureFichier.java
│   ├── LectureRepertoire.java
│   ├── GererData.java
│   └── Couleur.java
├── ihm/                         # Interface graphique
│   ├── FrameAppli.java
│   ├── PanneauPrincipal.java
│   ├── PanneauMenu.java
│   ├── PanneauFichier.java
│   ├── Fleche.java
│   ├── IhmCui.java
│   └── edit/                    # Sous-package édition
│       ├── FrameEdit.java
│       ├── PanneauChoix.java
│       ├── PanneauInfo.java
│       └── PopUp.java
└── data/                        # Données (fichiers .java d'exemple)
```

4. Lecture et analyse de fichiers Java

Principe général

Afin de permettre la rétro-conception UML à partir de code Java, nous avons mis en place une chaîne de lecture et de transformation dont l'objectif est de convertir un fichier source .java en une représentation métier exploitable par le reste de l'application.

Cette fonctionnalité repose sur une séparation claire des responsabilités :

- Le contrôleur déclenche la lecture,
- La couche métier se charge de l'analyse syntaxique,
- Les informations extraites sont stockées dans une structure interne (CreeClass).

Fichier principal impliqué

LectureFichier.java (package src.metier)

Ce fichier constitue **le point d'entrée de la lecture de fichiers** pour la rétro-conception.

Son rôle est de :

- prendre en charge un chemin de fichier fourni par le contrôleur,
- lire le contenu du fichier Java,
- extraire les éléments structurels nécessaires à la modélisation UML,
- produire un objet métier représentant la classe analysée.

Logique de lecture mise en place

1. Point d'entrée unique via une factory

La lecture d'un fichier est toujours initiée via la méthode :

LectureFichier.factoryLectureFichier(String chemin)

Cette approche permet :

- d'unifier l'accès à la lecture de fichiers,
- de masquer la logique de création au contrôleur,
- de rendre le système extensible (ajout d'autres formats sans modifier le contrôleur).

Dans le cas d'un fichier .java, la factory délègue la lecture au mécanisme d'analyse du code source.

---

2. Initialisation de la structure UML cible

Dès le début du traitement, une instance de CreeClass est créée.

Cette classe métier représente :

- une classe UML,
- son nom,
- son type (class, interface, record, abstract),

- ses attributs, méthodes et constructeurs,
- ses relations potentielles.

L'objectif est de traduire progressivement le contenu du fichier Java vers cette structure, sans dépendance à l'IHM.

---

3. Lecture séquentielle du fichier

Le fichier est lu ligne par ligne, ce qui permet :

- un contrôle précis de l'analyse,
- une détection progressive des éléments UML,
- une logique simple et compréhensible pour un développeur reprenant le projet.

Avant toute analyse, chaque ligne est nettoyée :

- suppression des commentaires (//, /* */),
- exclusion des lignes vides,
- évitement des faux positifs liés aux chaînes de caractères.

---

4. Identification de la déclaration de classe

Lorsque la ligne correspond à une déclaration de type :

- class
- interface
- record
- abstract class

elle est analysée mot par mot afin d'extraire :

- le type de la structure,
- le caractère abstrait,
- la classe mère (extends),
- les interfaces implémentées (implements).

Ces informations sont directement stockées dans l'objet CreeClass.

5. Extraction des membres de la classe

Une fois la classe identifiée, les lignes suivantes sont analysées pour distinguer :

- les constructeurs (nom de la classe + parenthèses),
- les méthodes (visibilité + parenthèses),
- les attributs (visibilité + ;).

Chaque élément détecté est ajouté à la structure métier via :

- ajouterConstructeur
- ajouterMethode
- ajouterAttribut

Cette logique permet de reconstruire fidèlement la signature UML de la classe.

/!\ Les fichiers contenant une déclaration enum sont explicitement ignorés.

---

7. Préparation des relations UML

Une fois la lecture terminée, la méthode initLienMulti() est appelée sur l'objet CreeClass.

Cette étape prépare :

- les futures relations inter-classes,
- les multiplicités associées,
- les traitements ultérieurs liés aux flèches UML.

Le calcul détaillé de ces relations est volontairement délégué à la couche métier concernée.

Résultat de la fonctionnalité

À l'issue de la lecture :

- Le fichier Java est transformé en une structure UML interne,
- Cette structure est indépendante de l'interface graphique,
- Elle peut être exploitée pour l'affichage, l'édition ou la sauvegarde.

Le contrôleur récupère ensuite cet objet afin de l'intégrer au reste du projet.

5. Gestion des fichiers et Calcul des liens/multiplicités

Objectif général

Pour permettre la rétro-conception UML à partir de plusieurs fichiers Java, le projet met en place une chaîne de traitement capable de :

- parcourir un répertoire contenant des fichiers .java
- analyser chaque fichier individuellement
- centraliser toutes les classes détectées
- établir les relations UML entre ces classes
- calculer les multiplicités associées à ces relations

Cette fonctionnalité est essentielle pour produire un diagramme cohérent à l'échelle d'un projet, et non d'un simple fichier isolé.

Principe global de fonctionnement

La gestion multi-fichiers repose sur une séparation claire des responsabilités :

1. Un composant charge et orchestre l'analyse d'un répertoire
2. Chaque fichier Java produit une représentation métier de classe
3. Les relations sont calculées une fois toutes les classes connues
4. Les multiplicités sont déduites à partir des attributs et collections

Cette logique garantit que les associations UML sont correctes et complètes, car elles sont calculées après l'analyse de l'ensemble du projet.

Lecture et centralisation des fichiers Java

Fichier concerné : LectureRepertoire

Pour gérer plusieurs fichiers Java simultanément, le projet utilise une classe dédiée à la lecture d'un répertoire complet.

Rôle de LectureRepertoire

- Parcourir un dossier fourni par l'utilisateur
- Identifier uniquement les fichiers .java
- Déléguer l'analyse de chaque fichier à un lecteur spécialisé
- Conserver une liste centralisée de toutes les classes détectées
- Déclencher ensuite la création des liens et des multiplicités

Logique de traitement

1. Le répertoire est parcouru via l'API File
2. Chaque fichier .java est transmis au mécanisme de lecture de fichier
3. Chaque lecture produit un objet métier représentant une classe Java
4. Tous ces objets sont stockés dans une liste unique
5. Une fois la collecte terminée :
    - les relations UML sont créées
    - les multiplicités sont calculées

Ce choix évite les incohérences : aucune classe n'est liée tant que toutes ne sont pas connues.

Représentation métier d'une classe Java

Fichier concerné : CreeClass

Chaque fichier Java analysé est transformé en un objet métier indépendant, capable de représenter fidèlement une classe UML.

Responsabilité de CreeClass

CreeClass est le noyau métier du projet. Il contient :

- les informations structurelles de la classe (nom, type, abstraction)
- ses attributs et méthodes
- ses relations UML (héritage, interface, association)
- ses multiplicités

- les informations nécessaires à l'affichage graphique (position, taille)

Cette centralisation permet à l'IHM d'aller chercher toutes les informations métier au même endroit.

Création des liens UML entre classes

Fichier concerné : Lien

Principe général

Une fois l'ensemble des fichiers Java analysés et transformés en objets CreeClass, le projet met en place une phase dédiée à la détection des relations UML entre ces classes. Pour cela, la logique de détection des relations n'est pas intégrée directement dans CreeClass, mais déléguée à un composant spécialisé : la classe Lien.

Lien est capable de déterminer les relations entre les différentes classes du projet.

Types de relations détectées

La classe Lien est responsable de l'identification de trois types de relations UML, toutes calculées à partir du code Java analysé :

1. Associations par attributs

Une association est détectée lorsqu'un attribut d'une classe possède pour type une autre classe du projet.

Concrètement :

- les attributs de la classe courante sont analysés
- leur type est comparé au nom des autres classes connues
- si un attribut référence une autre classe (directement ou sous forme de tableau), une relation est créée

Les attributs impliqués dans une association sont ensuite déplacés vers une liste dédiée (lstClassAttribut), afin de :

- distinguer les attributs simples des attributs participant à des relations UML
- faciliter le calcul des multiplicités

2. Relations d’héritage

Les relations d’héritage sont détectées à partir du nom de la classe mère enregistré lors de la lecture du fichier Java.

Le mécanisme repose sur :

- la comparaison entre le nom de la classe mère (mere)
- et les noms des autres classes du projet

Lorsqu’une correspondance est trouvée, la classe mère est ajoutée à la liste des liens d’héritage.

---

3. Implémentation d’interfaces

Les interfaces implémentées par une classe sont stockées sous forme de noms lors de l’analyse du fichier Java.

Pour chaque interface déclarée :

- le système vérifie si une classe du projet correspond à ce nom
- si c’est le cas, une relation d’implémentation est créée

Lorsqu’une correspondance est trouvée, la classe est ajoutée à la liste des liens d'implémentations.

Calcul des multiplicités UML

Fichier concerné : Multiplicite

Principe général

Le calcul des multiplicités UML est effectué après la création des liens, afin de s’appuyer sur des associations déjà identifiées.

Les multiplicités sont déduites automatiquement à partir :

- des attributs participant aux relations
- de leur type (simple ou collection)
- du nombre de références entre deux classes

Cette phase ne modifie pas les liens existants : elle les complète avec une information de multiplicité.

Logique de calcul

Chaque CreeClass possède un objet Multiplicite, chargé de calculer ses multiplicités vis-à-vis des autres classes.

Le calcul repose sur une analyse dans les deux sens attributs de la classe courante vers une autre classe est inversement

Cela permet de représenter correctement :

- les associations unidirectionnelles
- les associations bidirectionnelles

Détermination d'une multiplicité

La multiplicité associée à un attribut est déduite uniquement à partir de son type :

- si le type correspond à une collection ou un tableau
  → multiplicité 1..*
- sinon
  → multiplicité 1..1

Construction des paires de multiplicités

Pour chaque paire de classes liées :

1. Le nombre de références est compté dans chaque sens
2. Les listes de multiplicités sont équilibrées si nécessaire
3. Chaque association est stockée sous forme de paire :
    - multiplicité côté classe courante
    - multiplicité côté classe liée

Chaque paire reçoit également un identifiant unique indépendant de l'ordre de création utilisé ensuite par l'IHM pour l'édition des multiplicités.

Stockage et exploitation

Les multiplicités sont stockées dans une structure indépendante des liens :

- une map associant une classe à ses multiplicités
- une liste d'identifiants uniques pour les associations

---

6. Mise en place de l'affichage (GUI)

Objectif général

L'Affichage graphique a pour rôle de traduire le modèle métier (classes, relations, multiplicités) en une représentation UML interactive.

Elle constitue la couche IHM du projet et permet :

- l'affichage visuel des classes Java analysées,
- la représentation des relations UML (association, héritage, interface),
- l'interaction utilisateur (sélection, déplacement, zoom),
- l'export du diagramme sous forme d'image.

Cette fonctionnalité repose sur une séparation stricte entre le métier (src.metier) et l'affichage (src.ihm).

Fichiers impliqués et responsabilités

PanneauPrincipal – Support central de rendu graphique

Rôle

PanneauPrincipal est le composant graphique principal chargé de dessiner l'intégralité du diagramme UML.

Il agit comme :

- un conteneur graphique (hérite de JPanel),
- un gestionnaire de rendu

- un gestionnaire d'interactions souris.

Responsabilités principales :

- Maintenir la liste des classes à afficher (List<CreeClass>)
- Calculer dynamiquement :
    - les dimensions des classes,
    - leur position à l'écran,
- Générer les représentations graphiques des relations UML
- Centraliser les événements souris :
    - sélection d'une classe,
    - déplacement par glisser-déposer,
    - zoom contextuel par clic droit
- Déclencher l'export du diagramme en image

PanneauPrincipal ne crée pas les données UML, il les consomme via le contrôleur.

Logique globale de fonctionnement

Étape 1 – Récupération du modèle UML

Le PanneauPrincipal récupère les classes UML depuis le contrôleur :

- soit lors du chargement d'un dossier,
- soit lors de l'ajout d'un fichier,
- soit lors du chargement d'un diagramme sauvegardé.

Chaque CreeClass contient déjà :

- ses attributs,
- ses méthodes,
- ses relations (Lien),
- ses multiplicités (Multiplicite).

Étape 2 – Calcul et dessin des classes

Pour chaque CreeClass :

- les dimensions sont calculées dynamiquement :
    - hauteur en fonction du nombre d'attributs et méthodes,
    - largeur en fonction du texte le plus long que soit un attribut ou une méthode,
- le rectangle UML est dessiné,
- le contenu textuel est affiché :
    - nom,
    - stéréotype (classe / interface),
    - attributs (avec visibilité et propriétés),
    - méthodes (avec paramètres et type de retour).

Cette logique permet :

- un affichage lisible,
- une adaptation automatique aux tailles variables des classes.

Étape 3 – Construction des relations graphiques

Les relations sont préparées après le dessin des classes, car leurs positions sont nécessaires.

Trois catégories sont traitées :

Héritage

- basé sur les liens d'héritage stockés dans Lien,
- affiché avec une flèche à triangle fermé.

Interface

- basé sur les interfaces implémentées,
- affiché avec un trait en pointillés et triangle fermé.

Associations (avec multiplicités)

- calculées à partir de Multiplicite,
- marquées comme bidirectionnelles si nécessaire,
- associées à un identifiant unique.

Chaque relation est transformée en une instance de Fleche.

Étape 4 – Dessin des flèches

Une fois toutes les flèches construites :

- elles sont dessinées une par une,
- un décalage progressif est appliqué pour éviter le chevauchement,
- les multiplicités sont affichées avec rotation automatique,
- Les rôles sont ajoutés si définis.

Étape 5 – Interaction utilisateur

Le panneau gère directement :

- Sélection
    - clic gauche sur une classe pour le déplacement,
- Déplacement
    - glisser-déposer avec contraintes de bord,
- Zoom
    - clic droit sur une classe permet de visualiser l'affichage complet.

7. Affichage du Dessin des relations UML (flèches, multiplicités, rôles)

La classe Fleche est volontairement conçue comme une unité graphique autonome :

- elle connaît :
    - sa classe source,
    - sa classe cible,
    - son type UML,
    - ses multiplicités,
    - son rôle éventuel,
- elle calcule elle-même :
    - ses points d'ancrage,
    - son orientation,
    - la rotation du texte,
    - la forme de sa pointe.

PanneauPrincipal ne fait jamais de calcul géométrique de flèche : il délègue entièrement cette responsabilité à Fleche.

Logique globale de construction des flèches

La construction des relations se fait en plusieurs passes distinctes, chacune correspondant à un concept UML.

1. Héritage et interfaces

Les relations d'héritage et d'implémentation sont :

- extraites directement depuis les liens métier de CreeClass,
- transformées en flèches sans multiplicité,
- ajoutées immédiatement à la liste globale des flèches.

Ces relations sont :

- non éditables par l'utilisateur,
- toujours directionnelles,
- représentées par un triangle fermé, conformément au standard UML.

2. Associations avec multiplicités

Les associations sont construites à partir de la classe métier Multiplicite.

Logique suivie

Pour chaque classe :

1. on récupère la table des multiplicités calculées lors de l'analyse Java,
2. chaque association est identifiée par :
    - une classe source,
    - une classe cible,
    - une multiplicité de chaque côté,
3. un mécanisme de vérification empêche :
    - la duplication de liens (A → B et B → A),
    - l'affichage incohérent des associations bidirectionnelles.

Résultat

Chaque association valide devient une instance unique de Fleche, avec :

- un identifiant,
- un marqueur bidirectionnel si nécessaire,
- deux multiplicités textuelles.

Logique détaillée du dessin des flèches

Calcul des points d'ancrage

Chaque flèche commence par :

- le calcul du centre géométrique de la classe source,

- le calcul du centre géométrique de la classe cible.

À partir de ces centres :

- la flèche est automatiquement attachée :
    - soit sur un bord vertical,
    - soit sur un bord horizontal.

Ceci assure que la flèche est attachée au bord du rectangle.

Gestion du décalage

Lorsque plusieurs flèches relient les mêmes classes :

- un vecteur perpendiculaire à la direction principale est calculé,
- un décalage progressif est appliqué (ESPACE_FL, ESPACE_Y),
- un seuil maximal empêche les décalages excessifs.

Cette approche garantit aucune superposition.

Dessin selon le type UML

Type UML	Représentation
Association	Trait simple + pointe ouverte
Association bidirectionnelle	Trait simple sans pointe
Auto-association	Carré de rappel
Héritage	Trait plein + triangle fermé
Interface	Trait pointillé + triangle fermé

Chaque type est centralisé dans Fleche.dessiner().

Affichage des multiplicités

Les multiplicités sont affichées selon une logique précise :

- positionnées à ~10 % et ~90 % de la longueur de la flèche,

- automatiquement pivotées selon l'angle de la flèche,
- jamais affichées à l'envers (correction de l'angle > ±90°).

Cas particulier :

- Pour une auto-association, les multiplicités sont placées autour du carré.

Cette rotation dynamique garantit la lisibilité, même après déplacement des classes.

Gestion des rôles

Les rôles :

- ne sont pas stockés directement dans Multiplicite,
- sont associés dynamiquement à une flèche via son identifiant.

Le panneau :

- conserve une correspondance classe → (id → rôle),
- Réinsére les rôles dans les flèches à chaque nouvel affichage.

Cela permet :

- l'édition utilisateur sans modifier le modèle métier,
- une sauvegarde indépendante du rendu graphique.

---

8. Fonctionnalité : Modification / Édition des associations UML

1. But de la fonctionnalité

La fonctionnalité de modification permet à l'utilisateur de changer les relations entre les classes UML déjà chargées dans l'application.

Grâce à cette fonctionnalité, l'utilisateur peut :

- choisir une classe UML
- voir toutes les classes qui lui sont liées
- modifier les multiplicités des relations

- ajouter ou modifier le rôle associé à une relation
- enregistrer les changements de manière cohérente

---

2. Organisation générale de la fonctionnalité

La fonctionnalité est regroupée dans le package ihm.edit.
Ce package correspond à la partie édition de l'interface graphique.

Elle est composée de trois éléments principaux :

- un panneau pour choisir la classe à modifier
- un panneau pour afficher et modifier les informations
- une fenêtre de saisie pour les rôles

Chaque élément a un rôle précis et limité.

---

3. Sélection de la classe à modifier (Panneau de choix)

Le panneau de choix est la **première étape** de la modification.

Il affiche une liste contenant toutes les classes UML actuellement chargées dans le projet.

Fonctionnement général

Quand l'application charge ou modifie les classes UML :

- le panneau récupère la liste des classes auprès du contrôleur
- chaque classe est affichée sous forme de nom dans une liste

L'utilisateur peut cliquer sur un nom de classe pour la sélectionner.

Logique du panneau

Ce panneau :

- ne modifie aucune donnée
- ne connaît pas les détails des relations
- sert uniquement à transmettre le nom de la classe sélectionnée

Lorsqu'une classe est choisie :

- le panneau informe le panneau d'information

- le panneau d'information devient responsable de l'affichage et de la modification

Cela permet de garder une séparation claire entre la sélection et l'édition.

---

4. Affichage et modification des associations (Panneau d'information)

Le panneau d'information est **le cœur de la fonctionnalité d'édition**.

Il s'affiche uniquement lorsqu'une classe est sélectionnée.

Rôle principal

Ce panneau permet de :

- afficher toutes les relations de la classe sélectionnée
- montrer les relations dans les deux sens (sortantes et entrantes)
- permettre la modification des multiplicités
- lancer la modification du rôle d'une relation
- enregistrer les changements

4.1 Compréhension des relations affichées

Pour une classe donnée, il existe deux types de relations :

- les relations où la classe pointe vers une autre (unidirectionnel)
- les relations où une autre classe pointe vers elle (bidirectionnel)

Le panneau d'information :

- rassemble ces deux types de relations
- les affiche de manière uniforme à l'utilisateur

Ainsi, l'utilisateur voit l'ensemble des associations, sans avoir à se soucier de leur sens technique.

4.2 Construction dynamique de l'interface

L'interface du panneau n'est jamais fixe.

À chaque fois qu'une nouvelle classe est sélectionnée :

- le panneau efface l'affichage précédent
- il reconstruit entièrement les champs en fonction des relations existantes
- chaque relation génère un bloc visuel avec :
    - un titre indiquant les deux classes liées
    - deux champs pour les multiplicités
    - un bouton pour le rôle

Cette approche permet :

- d'éviter les erreurs d'affichage
- de garantir que l'IHM reflète toujours l'état réel du modèle

---

4.3 Modification des multiplicités

Les multiplicités sont affichées sous forme de champs de texte.

L'utilisateur peut :

- modifier la multiplicité côté cible
- visualiser la multiplicité côté source (non modifiable)

Avant d'enregistrer :

- le programme vérifie que tous les champs sont remplis
- le format des multiplicités est contrôlé
- certaines incohérences sont détectées et signalées

Si une erreur est détectée :

- un message explicatif est affiché
- les modifications ne sont pas enregistrées

---

5. Validation et enregistrement des modifications

Une fois les modifications terminées, l'utilisateur clique sur le bouton **Valider**.

Ce qui se passe à ce moment-là

Le panneau :

1. relit toutes les valeurs saisies
2. reconstruit une structure complète des relations
3. transmet cette structure au contrôleur
4. met à jour la classe sélectionnée

Le panneau ne modifie jamais directement les données métier.

5.1 Mise à jour des relations inverses

Une règle importante est respectée automatiquement :

Si une relation est modifiée dans un sens, elle est aussi mise à jour dans l'autre sens.

Par exemple :

- si la relation entre deux classes est modifiée
- alors la classe liée est également mise à jour

Cela garantit :

- la cohérence du diagramme UML
- l'absence de contradictions entre les classes

---

6. Modification du rôle d'une association

Chaque relation possède un bouton permettant de modifier son rôle.

Lorsque l'utilisateur clique sur ce bouton :

- une petite fenêtre de saisie s'ouvre
- l'utilisateur entre le rôle souhaité
- la valeur est envoyée au contrôleur

La fenêtre ne stocke aucune donnée, ne fait aucune vérification métier et sert uniquement d'interface de saisie.

8. Fonctionnalité : Enregistrement et chargement des projets UML

1. Objectif de la fonctionnalité

La fonctionnalité d'enregistrement permet de sauvegarder l'état complet du projet UML afin de pouvoir le rouvrir plus tard sans perdre les informations calculées ou modifiées.

Elle permet notamment de conserver :

- les classes UML analysées
- leur position à l'écran
- leurs attributs et méthodes
- les relations entre les classes
- les multiplicités modifiées
- les rôles ajoutés par l'utilisateur

Cette fonctionnalité est essentielle pour transformer l'application en outil de travail persistant, et non en simple analyseur temporaire.

---

2. Emplacement dans l'architecture du projet

La gestion de l'enregistrement est regroupée dans la classe GererData, située dans le package metier.

Ce choix est volontaire :

- la sauvegarde concerne les données, pas l'interface
- aucune dépendance avec l'IHM
- réutilisable quel que soit le mode d'utilisation du programme

La classe ne contient aucun élément graphique et ne dépend pas de Swing.

---

3. Principe général de la persistance

Le projet propose deux modes de sauvegarde :

1. Un format binaire automatique
2. Un format texte lisible et modifiable

Ces deux formats répondent à des besoins différents.

---

4. Sauvegarde et chargement binaire (.ser)

Rôle du format binaire

Le format binaire permet :

- une sauvegarde rapide
- une restauration fidèle de l'état interne
- une implémentation simple

Il est principalement utilisé pour :

- les sauvegardes rapides
- les tests
- les restaurations internes

Logique générale

Lors de la sauvegarde :

- la liste complète des classes UML est récupérée
- elle est écrite telle quelle dans un fichier
- toutes les informations sont conservées automatiquement

Lors du chargement :

- la liste est relue depuis le fichier
- les objets sont recréés exactement comme ils étaient
- l'application peut reprendre immédiatement son état précédent

Ce mécanisme repose sur le fait que les classes métier sont conçues pour être sauvegardables.

---

5. Sauvegarde et chargement texte (.uml)

Pourquoi un format texte ?

Le format texte a été ajouté pour :

- permettre la lecture humaine
- faciliter le débogage
- autoriser des modifications manuelles
- mieux comprendre la structure UML sauvegardée

Ce format joue un rôle clé dans un **projet pédagogique de rétro-conception**.

---

6. Chargement d'un projet depuis un fichier texte

Logique globale du chargement

Le chargement se fait en plusieurs étapes successives :

1. lecture du fichier ligne par ligne
2. reconstruction progressive des classes
3. ajout des attributs, méthodes et rôles
4. création des relations entre classes
5. recalcul des multiplicités

Cette séparation garantit que toutes les classes existent avant de créer les liens.

6.1 Reconstruction des classes

Chaque classe est décrite dans le fichier par un ensemble de lignes.

Lors du chargement :

- une nouvelle classe est créée
- ses propriétés générales sont renseignées
- sa position et sa taille sont restaurées

Cela permet de retrouver exactement le diagramme tel qu'il était affiché.

6.2 Héritage et interfaces

Le chargement gère également :

- l'héritage entre classes

- l'implémentation d'interfaces

Ces informations sont stockées sous forme de noms, puis reconnectées une fois toutes les classes connues.

6.3 Attributs et méthodes

Les attributs et méthodes sont ajoutés progressivement à chaque classe.

Le programme distingue :

- les attributs d'instance
- les attributs de classe
- les méthodes statiques
- les méthodes abstraites

Cela permet de conserver une représentation UML complète et fidèle.

6.4 Chargement des rôles

Les rôles définis par l'utilisateur lors de l'édition sont également restaurés.

Chaque rôle est associé :

- à une relation
- à un identifiant de flèche
- à une classe source

Cela garantit que les informations ajoutées manuellement ne sont jamais perdues.

---

7. Reconstruction des relations et multiplicités

Une fois toutes les classes chargées :

- les liens UML sont recréés
- les associations entre classes sont reconnectées
- les multiplicités sont re calculées
- les relations inverses sont rétablies

Cette étape est volontairement effectuée **après** la lecture complète du fichier pour éviter toute incohérence.

8. Sauvegarde dans le fichier texte

Logique de la sauvegarde

Lors de l'enregistrement :

- chaque classe est écrite séparément
- ses propriétés sont enregistrées
- ses relations sont listées
- ses rôles sont ajoutés

Une ligne vide sépare chaque classe afin de rendre le fichier plus lisible.