SAE 3.01 : Développement d'applications

Génération automatique de diagramme de classes

Valentino Lambert,
André Jules,
Comte Gabriel,
Segard Mathis

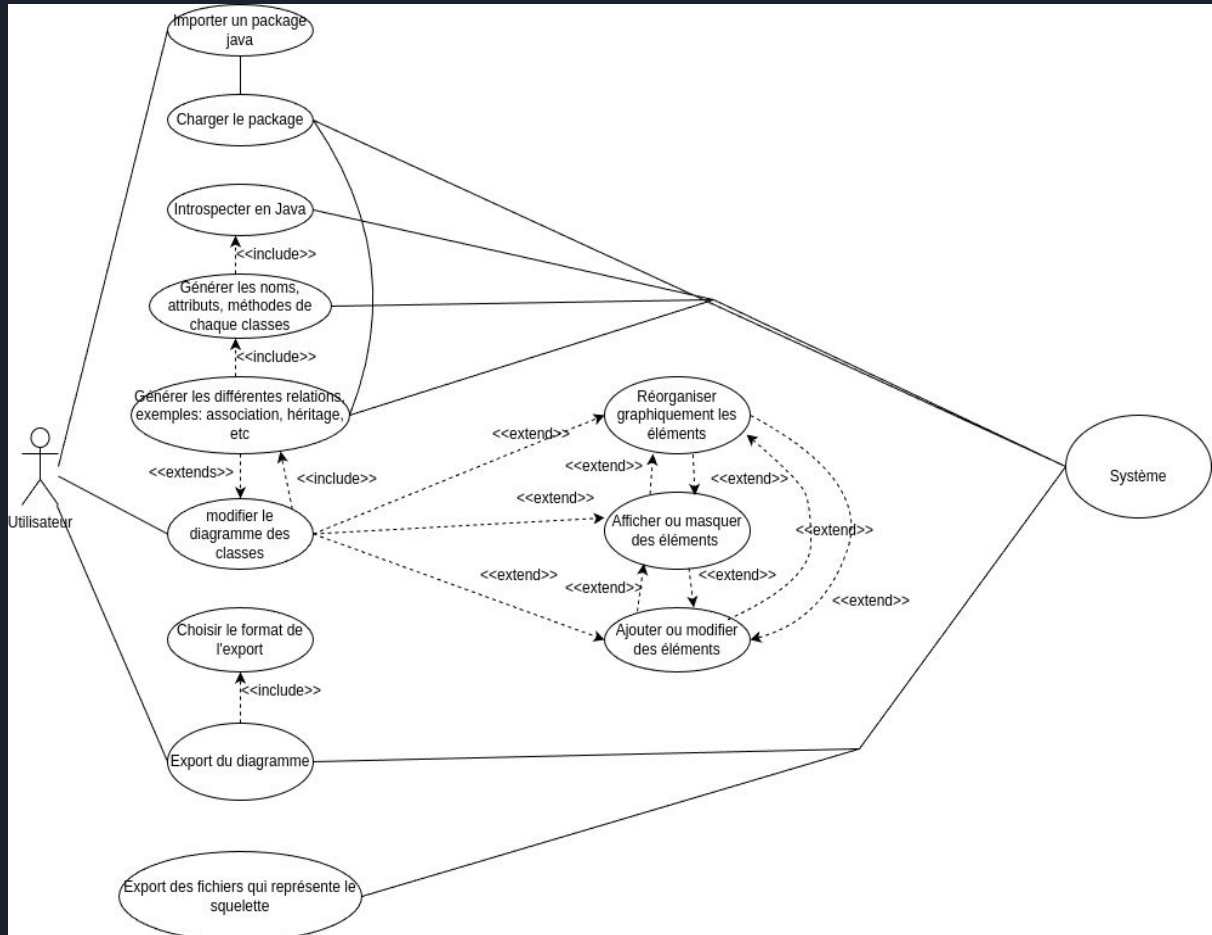
Sommaire :

- Contexte
- Présentations des itérations
- Présentation de l’application
- Présentation individuel
- Conclusion

Contexte :

Une application :

- Importer un package
- Générer un diagramme de classe
- Modifier manuellement ce diagramme
- Afficher / Masquer des éléments
- Exporter ce diagramme



Itération 1 et 2 :

Fonctionnalités prévues	Fonctionnalités réalisées
- Importation et Lecture d'un package .class - Chargement d'un .class grâce à un chemin absolu. - Vérifier que le fichier est un .class - Étendre l'introspection - Afficher les informations de la classe - Créer les éléments nécessaires pour la génération du diagramme - Lecture d'un package n'importe où sur l'ordinateur (classLoader externe).	- Lecture uniquement sur des .class présent dans le projet - Vérifier que le fichier est un .class - Étendre l'introspection - Afficher les informations de la classe (terminal) - Créer les éléments nécessaires pour la génération du diagramme

Itération 3 à 5 :

Fonctionnalités prévues	Fonctionnalités réalisées
- Créer l'interface graphique. - Faire une vue pour les flèches - Faire une vue pour les textes méthodes et attributs en utilisant la fabrique - Faire une vue pour une classe (faire une fabrique de vue pour créer les différentes classes/interfaces) - Faire une vue pour un package - Introspection détaillée et définitive (ajouts modificateurs et héritage) - Générer un diagramme de classe basic graphiquement (sans relations) - Génération automatique de diagrammes de classes à partir de packages Java. - Export en Puml et Png - Déplacer les classes sur l'écran - Faire une méthode pour ajouter des Package dans un Diagramme et une méthode pour ajouter une classe dans un package déjà existant - Empêcher la superposition des classes - Ajout graphique de la gestion des relations - Visibilité contrôleur	- Créer l'interface graphique. - Faire une vue pour les flèches - Faire une vue pour les textes méthodes et attributs en utilisant la fabrique - Faire une vue pour une classe (faire une fabrique de vue pour créer les différentes classes/interfaces) - Faire une vue pour un package - Introspection détaillée et définitive (ajouts modificateurs et héritage) - Générer un diagramme de classe basic graphiquement (sans relations) - Génération automatique de diagrammes de classes à partir de packages Java. - Export en Puml et Png - Déplacer les classes sur l'écran - Faire une méthode pour ajouter des Package dans un Diagramme et une méthode pour ajouter une classe dans un package déjà existant - Visibilité contrôleur

Bilan :

Partie conception :

- Certaines parties de la conceptions négligées et mal appliquées.
- Problème de communication au sein du group

Présentation :

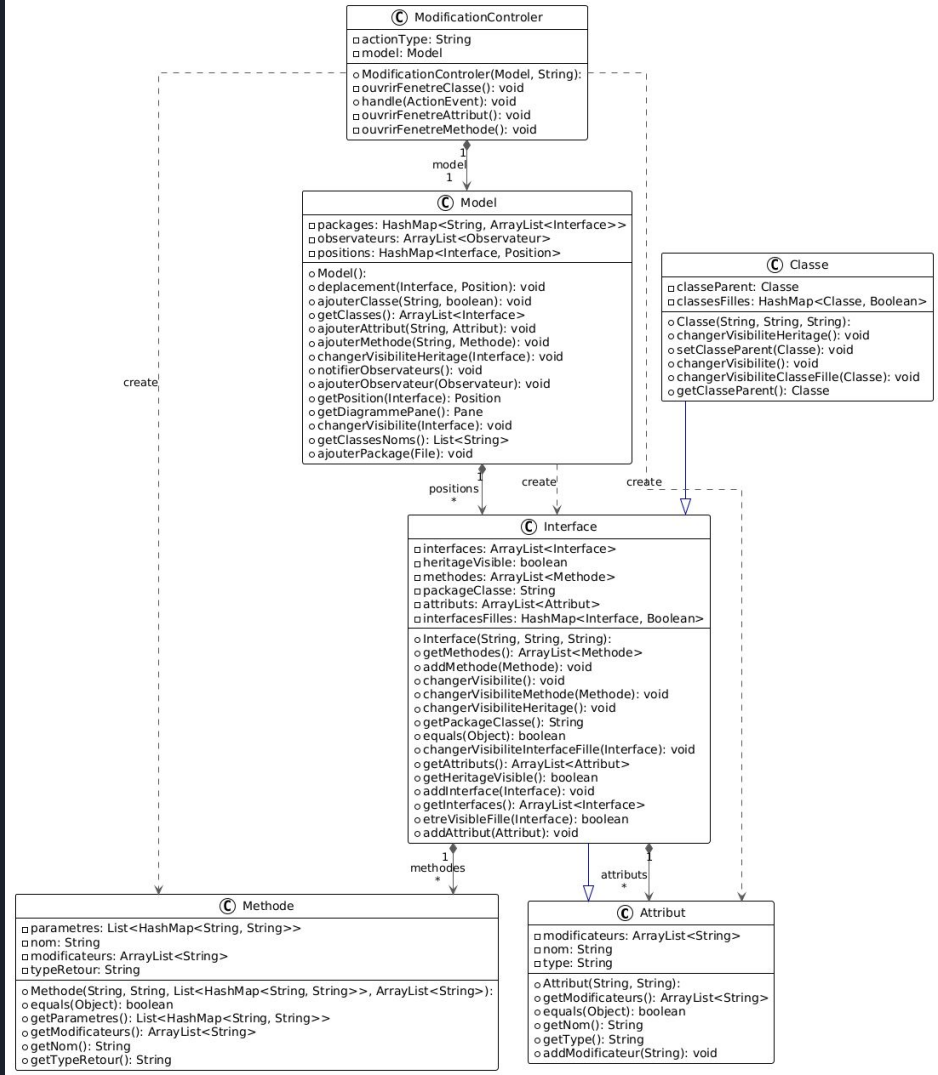
Présentation individuel :

- Valentino Lambert : Ajout Manuel de classes / méthodes / attributs
- Gabriel Comte : MVC
- Mathis Segard Loader et Visibilité

Ajout manuel de classe / méthodes / attributs :

Architecture générale

- **`Model`** :
  - Gère les données, comme les classes, les méthodes et les attributs.
  - Centralise les modifications et notifie la vue lorsqu'un changement est effectué.
- **`VuePrincipale`** :
  - Se met à jour automatiquement lorsque le modèle change.
- **`ModificationControler`** :
  - Intercepte les actions utilisateur et effectue les modifications nécessaires dans le modèle.



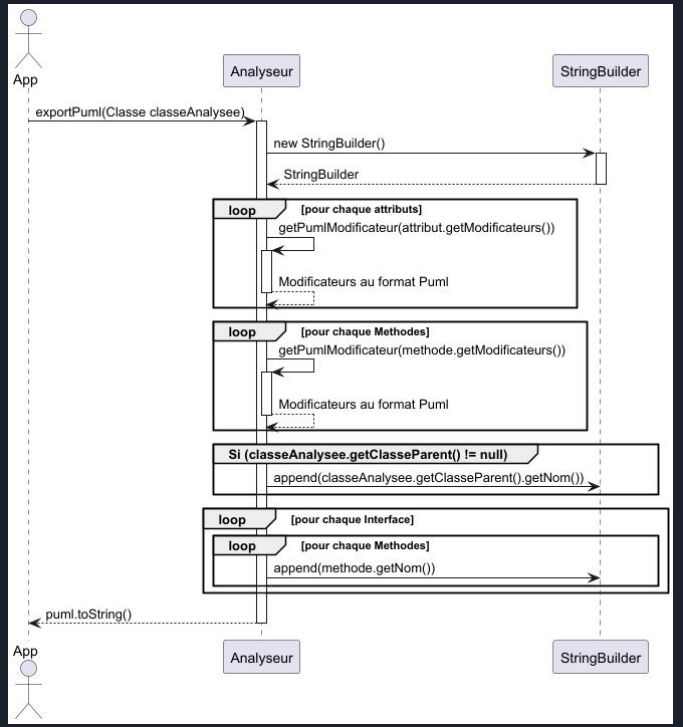
Système d’export en Puml:

Particularité :

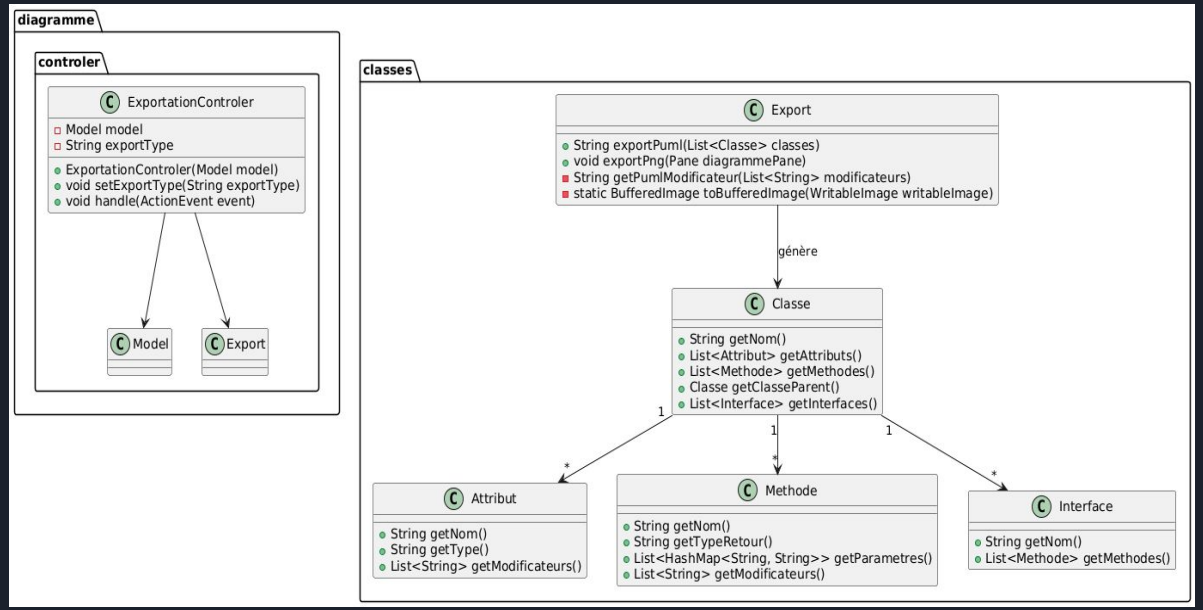
Cette partie a complètement été négligée dans la conception, nécessitant de la réaliser durant l’itération.

Fonctionnement :

- StringBuilder
- Boucle sur les classes, attributs et les méthodes
- Ajout des modificateurs et traduction en Puml
- Gestion des relations
- Création d’un fichier et écriture

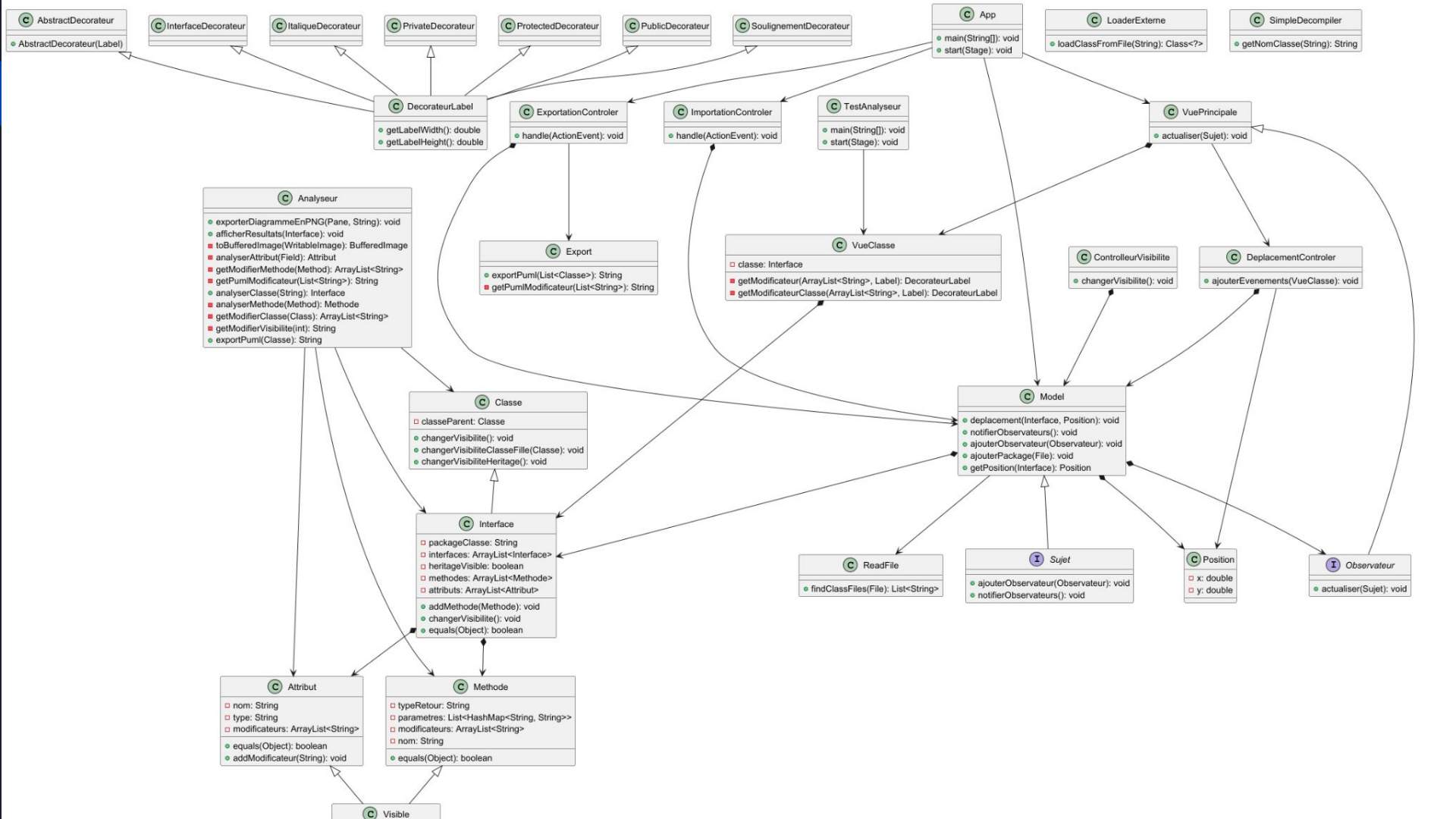


Système d’export en Puml:



Fonctionnement MVC

- l’actualisation et la construction se fait dans la vuePrincipale.
  - => puis la construction est passé au Model.
- Les autres vues ne sont pas des “vrais vues”.
- mise en place d’un décorateur pour les label.
- mise en place d’une fabrique afin de créer les différents type de flèches.


AbstractDecorateur
+ AbstractDecorateur(Label)
InterfaceDecorateur
ItaliqueDecorateur
PrivateDecorateur
ProtectedDecorateur
PublicDecorateur
SoulignementDecorateur
App
+ main(String[]): void
+ start(Stage): void
LoaderExterne
+ loadClassFromFile(String): Class<?>
SimpleDecompiler
+ getNomClasse(String): String
DecorateurLabel
+ getLabelWidth(): double
+ getLabelHeight(): double
ExportationControler
+ handle(ActionEvent): void
ImportationControler
+ handle(ActionEvent): void
TestAnalyseur
+ main(String[]): void
+ start(Stage): void
VuePrincipale
+ actualiser(Sujet): void
Analyseur
+ exporterDiagrammeEnPNG(Pane, String): void
+ afficherResultats(Interface): void
- toBufferedImage(WritableImage): BufferedImage
- analyserAttribut(Field): Attribut
- getModifierMethode(Method): ArrayList<String>
- getPumlModificateur(List<String>): String
+ analyserClasse(String): Interface
- analyserMethode(Method): Methode
- getModifierClasse(Class): ArrayList<String>
- getModifierVisibilite(int): String
+ exportPuml(Classe): String
Export
+ exportPuml(List<Classe>): String
- getPumlModificateur(List<String>): String
VueClasse
- classe: Interface
- getModificateur(ArrayList<String>, Label): DecorateurLabel
- getModificateurClasse(ArrayList<String>, Label): DecorateurLabel
ControlleurVisibilite
+ changerVisibilite(): void
DeplacementControler
+ ajouterEvenements(VueClasse): void
Classe
- classeParent: Classe
+ changerVisibilite(): void
+ changerVisibiliteClasseFille(Classe): void
+ changerVisibiliteHeritage(): void
Model
+ deplacement(Interface, Position): void
+ notifierObservateurs(): void
+ ajouterObservateur(Observateur): void
+ ajouterPackage(File): void
+ getPosition(Interface): Position
Interface
- packageClasse: String
- interfaces: ArrayList<Interface>
- heritageVisible: boolean
- methodes: ArrayList<Methode>
- attributs: ArrayList<Attribut>
+ addMethode(Methode): void
+ changerVisibilite(): void
+ equals(Object): boolean
ReadFile
+ findClassFiles(File): List<String>
Sujet
+ ajouterObservateur(Observateur): void
+ notifierObservateurs(): void
Position
- x: double
- y: double
Observateur
+ actualiser(Sujet): void
Attribut
- nom: String
- type: String
- modificateurs: ArrayList<String>
+ equals(Object): boolean
+ addModificateur(String): void
Methode
- typeRetour: String
- parametres: List<HashMap<String, String>>
- modificateurs: ArrayList<String>
- nom: String
+ equals(Object): boolean
Visible

Loader fonctionnement

Particularité :

Peut charger une classe même quand la classe qu'elle que soit l'emplacement du fichier

Visibilité

Particularité :

masque les classes et les méthodes

héritage non masquable

Conclusion