# ADG PROJECT

Automatic Diagram Generator

## Groupe

HEUERTZ Zacharie
KNORST Valentin
KORBAN Ryan
GROS Geoffrey
FUCHS Thomas

## Sommaire

# Analyse

## I. Liste des fonctionnalités

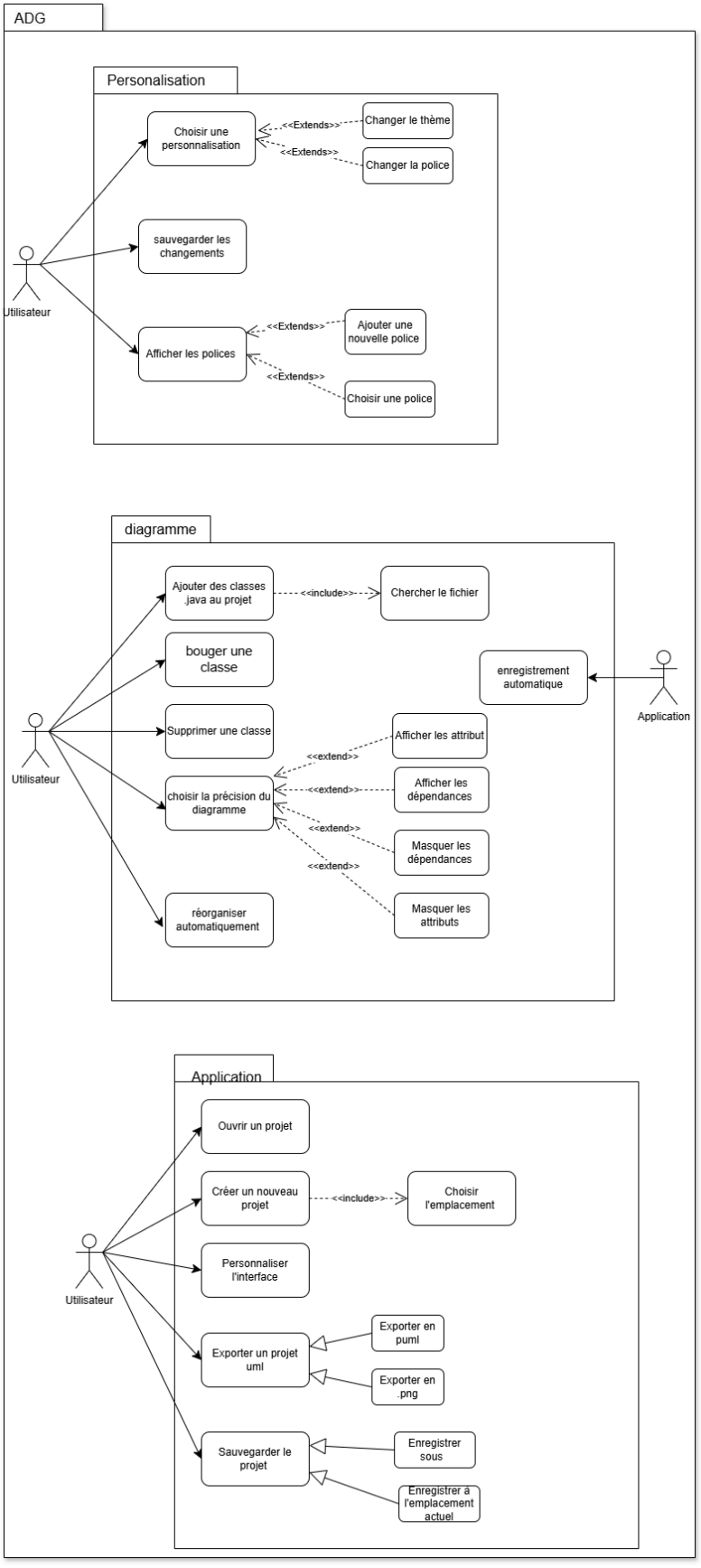
- ajouter une classe
- enlever une classe
- retirer le display l'héritage d'une classe
- bouger une classe
- déposer un fichier
- déposer un dossier
- enregistrement automatique
- exporter le projet en uml et .png
- personnalisation de l'interface
- créer un nouveau projet
- sélectionner des projets
- démasquer/masquer une classe

## II. Diagrammes

### 1. Consigne

- les cas d'utilisation de l'application et les diagrammes de cas d'utilisation si cela est nécessaire,
- des descriptions textuelles et/ou des DSS pourront les compléter,
- un diagramme d'activités de l'application,
- la conception : un diagramme de classe en y précisant les patrons de conception que vous prévoyez d'utiliser

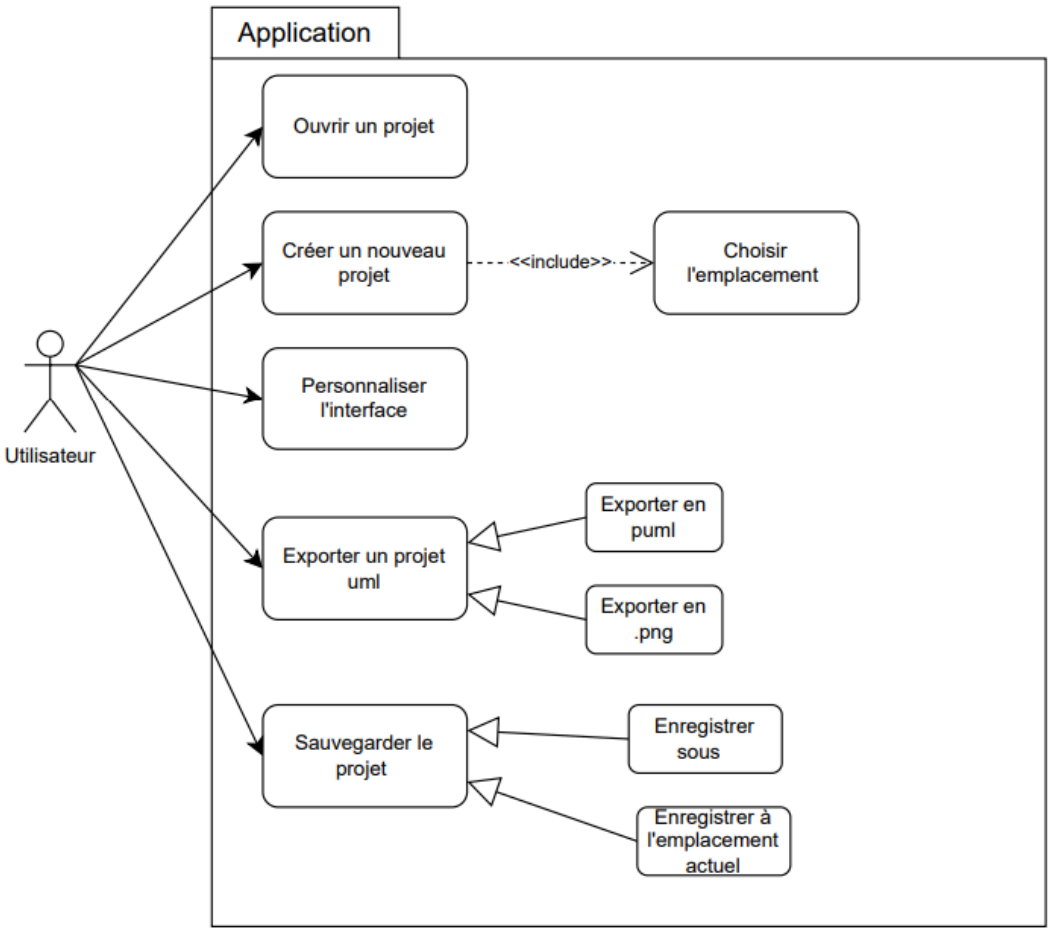
**Ci-dessous le cas d'utilisation de l'application, vous pourrez trouver en dessous les détails des différents packages**



## 2. Cas d'utilisation de l'application

**Cas d'utilisation "global":**

- L'utilisateur peut sélectionner un projet existant ou en créer un nouveau.
- Il peut aussi personnaliser son interface utilisateur selon ses envies.
- Enfin il peut exporter un projet dans le type d'export souhaité.




**Lors de la modification du diagramme l'utilisateur peut:**

- Déposer un fichier ou un dossier pour en générer le diagramme de classe associé

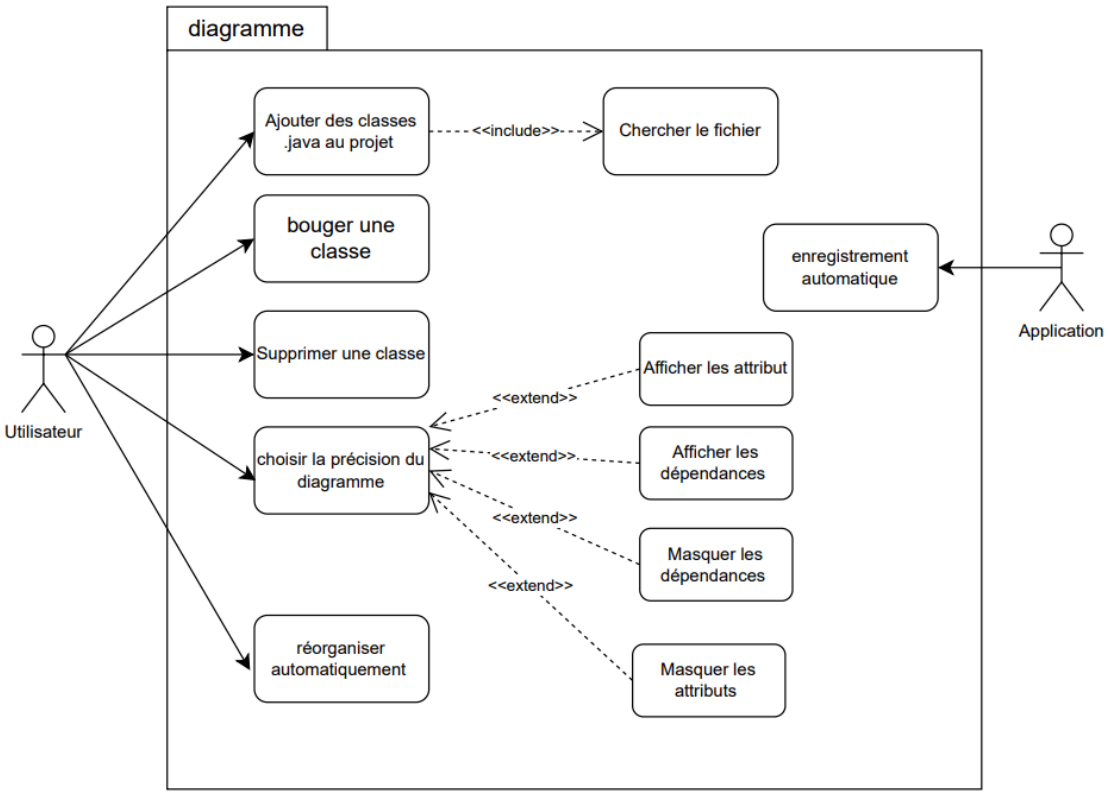
- Déplacer une classe de position en maintenant le clic sur cette dernière

- Supprimer une classe dans l'affichage

- Modifier la précision des détails des classes et des relations entre elle

- Il peut enregistrer son fichier s'il désire sauvegarder l'état d'avancement de son diagramme de classe

Chaque action entraîne un enregistrement automatique afin de ne pas perdre les données. L'utilisateur peut réorganiser manuellement ou une réorganisation aura lieu en cas d'ajout/suppression de classe.




**Lors de la personnalisation de l'interface, l'utilisateur peut:**

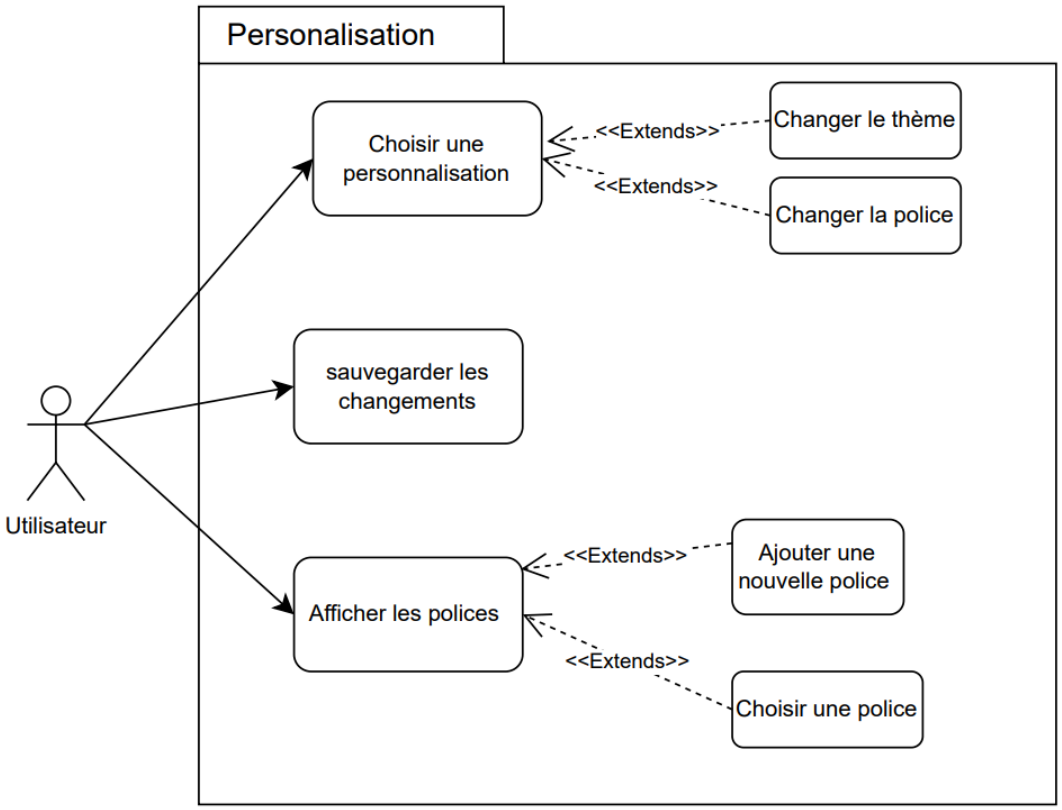
- Modifier une option.

- Sauvegarder les changements qu'il a apportés.

- Afficher les différentes options qu'il peut modifier selon ses désirs.

- Ajouter une option au choix d'une personnalisation.

- Sélectionner une option.

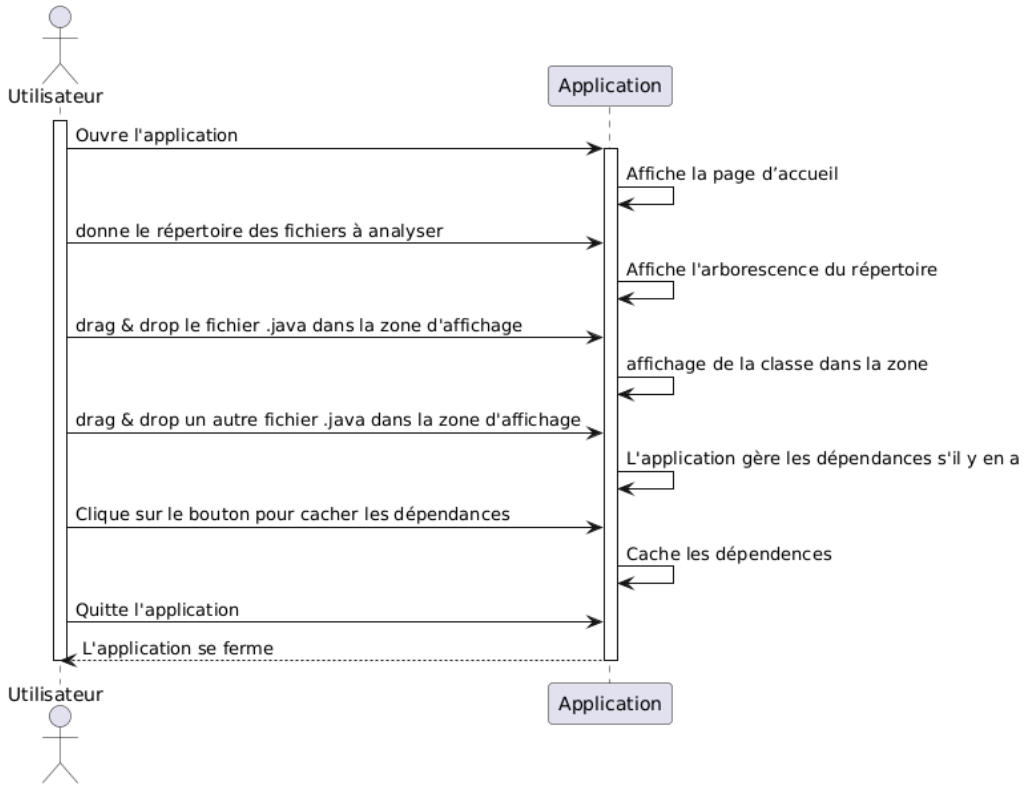



## 3. Descriptions textuelles

**Scénario d'utilisation "Création de projet":**

- l'utilisateur ouvre l'application et arrive sur la page d'accueil
- il clique sur "Créer un nouveau projet"
- l'utilisateur renseigne le répertoire source des fichiers qu'il veut analyser par la suite
- l'utilisateur arrive sur la fenêtre du projet
- l'utilisateur déroule un répertoire
- l'utilisateur drag un fichier .java
- l'utilisateur drop le fichier dans la **zone d'affichage**
- la classe apparaît
- l'utilisateur drag and drop un nouveau fichier
- la classe apparaît et ses dépendances également (si elles sont présentes sur la zone d'affichage)
- l'utilisateur clique droit sur une classe pour afficher un menu contextuel
- l'utilisateur clique sur "cacher les dépendances"
- l'utilisateur ferme la fenêtre

- l'utilisateur ferme la page d'accueil




```
@startuml
actor Utilisateur
participant "Application" as App
activate Utilisateur
Utilisateur -> App : Ouvre l'application
activate App
App -> App : Affiche la page d'accueil
Utilisateur -> App : donne le répertoire des fichiers à analyser
App -> App : Affiche l'arborescence du répertoire
Utilisateur -> App : drag & drop le fichier .java dans la zone
d'affichage
App -> App : affichage de la classe dans la zone
Utilisateur -> App : drag & drop un autre fichier .java dans la zone
d'affichage
App -> App :  L'application gère les dépendances s'il y en a
Utilisateur -> App : Clique sur le bouton pour cacher les dépendances
App -> App :  Cache les dépendences
Utilisateur -> App : Quitte l'application
App --> Utilisateur : L'application se ferme
deactivate App
deactivate Utilisateur
```

```
@enduml
```

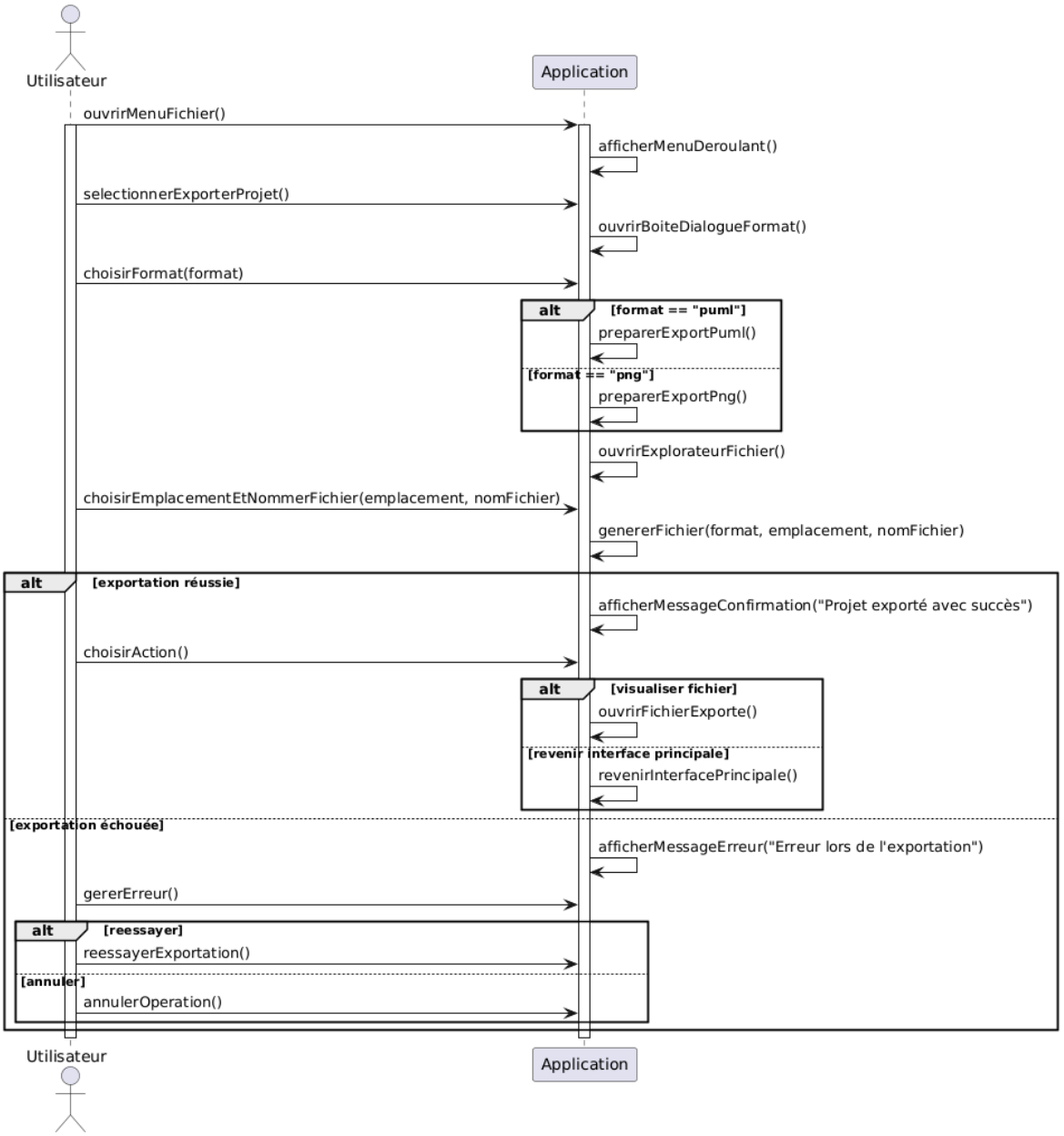
**Scénario d'utilisation "exporter un projet UML" :**

Préconditions :

- L'utilisateur a un projet ouvert dans l'application.
- Le projet contient au moins un diagramme de classe

Description des étapes :

- L'utilisateur navigue dans l'interface de l'application et localise le menu "Fichier", il clique dessus, un menu déroulant s'affiche
- L'utilisateur clique sur l'option "Exporter le projet".
- Une boîte de dialogue s'ouvre, permettant à l'utilisateur de choisir le format d'exportation (PUML ou png).
- L'utilisateur sélectionne le format souhaité et clique sur "Suivant".
- Une nouvelle boîte de dialogue (explorateur de fichier) s'ouvre, demandant à l'utilisateur de choisir l'emplacement de sauvegarde du fichier.
- L'utilisateur choisi où le sauvegarder et a la possibilité de le nommer comme il veut.
- L'application génère le fichier (puml ou png) en utilisant les données du projet (classes, relations, attributs, etc.).
- Une fois l'exportation terminée, l'application affiche un message de confirmation indiquant que le projet a été exporté avec succès.
- L'utilisateur peut choisir de visualiser le fichier exporté ou de revenir à l'interface principale de l'application.
- Si une erreur se produit pendant le processus d'exportation (par exemple, un problème d'écriture dans le fichier), un message d'erreur s'affiche, informant l'utilisateur du problème et lui offrant des options pour réessayer ou annuler l'opération.



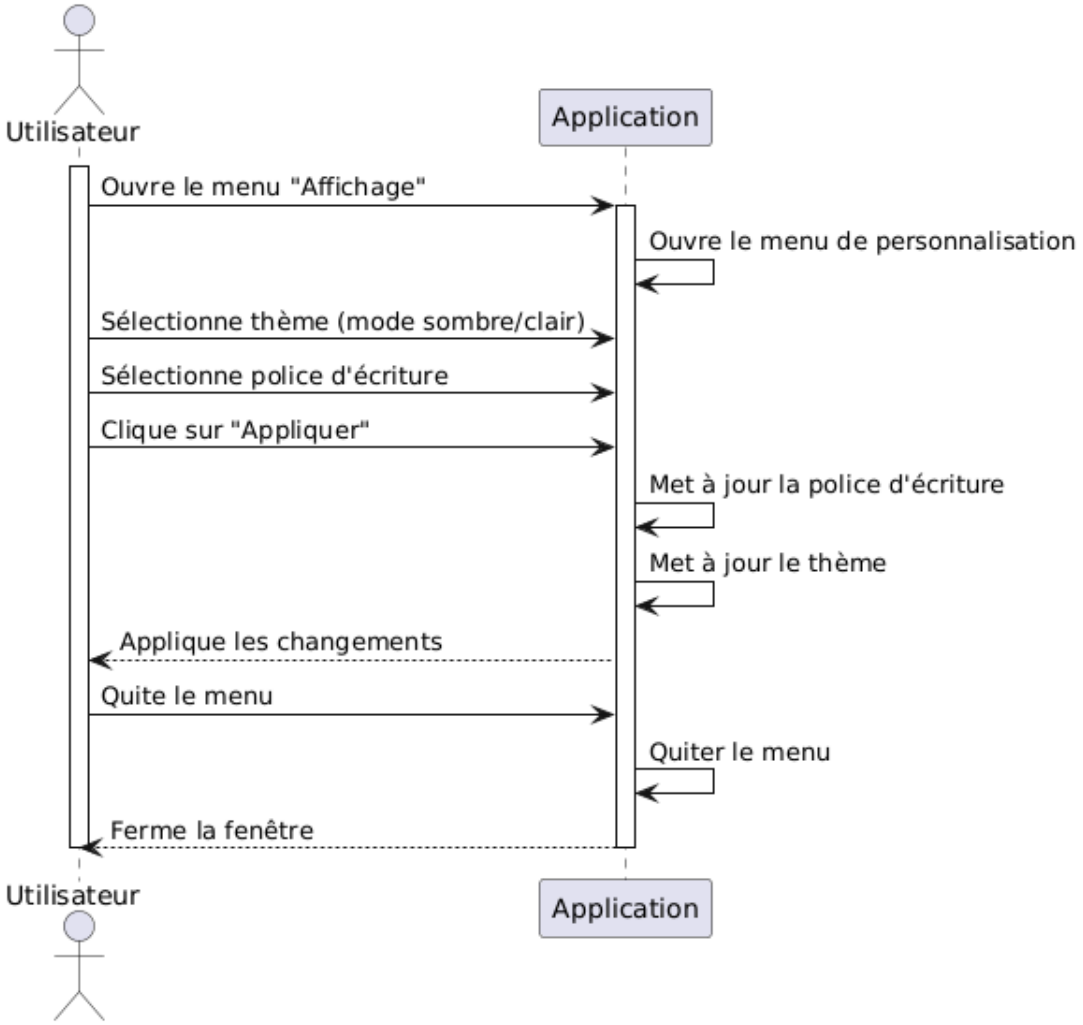

**Scénario d'utilisation "Personnalisation" :**

Préconditions :

- L'utilisateur a un projet ouvert dans l'application.
- Le projet contient au moins un diagramme de classe

Description des étapes :

- L'utilisateur navigue dans l'interface de l'application et localise le menu "Affichage", il clique dessus, des options s'affichent
- Il a le choix entre plusieurs options
- Il peut changer le thème de l'application (mode sombre/Claire)
- L'utilisateur peut choisir la police d'écriture de l'application
- Une fois les personnalisations effectuées, l'utilisateur peut les appliquer, ce qui met à jour l'état de l'application.
- Il peut quitter le menu à la fin




```
@startuml

actor Utilisateur
participant Application
activate Utilisateur

Utilisateur -> Application : Ouvre le menu "Affichage"
activate Application
Application -> Application : Ouvre le menu de personnalisation

Utilisateur -> Application : Sélectionne thème (mode sombre/clair)

Utilisateur -> Application : Sélectionne police d'écriture
Utilisateur -> Application : Clique sur "Appliquer"
Application -> Application : Met à jour la police d'écriture
Application -> Application : Met à jour le thème
```

```
Application --> Utilisateur : Applique les changements
Utilisateur -> Application : Quite le menu
Application -> Application : Quiter le menu
Application --> Utilisateur : Ferme la fenêtre
deactivate Application
deactivate Utilisateur
@enduml
```

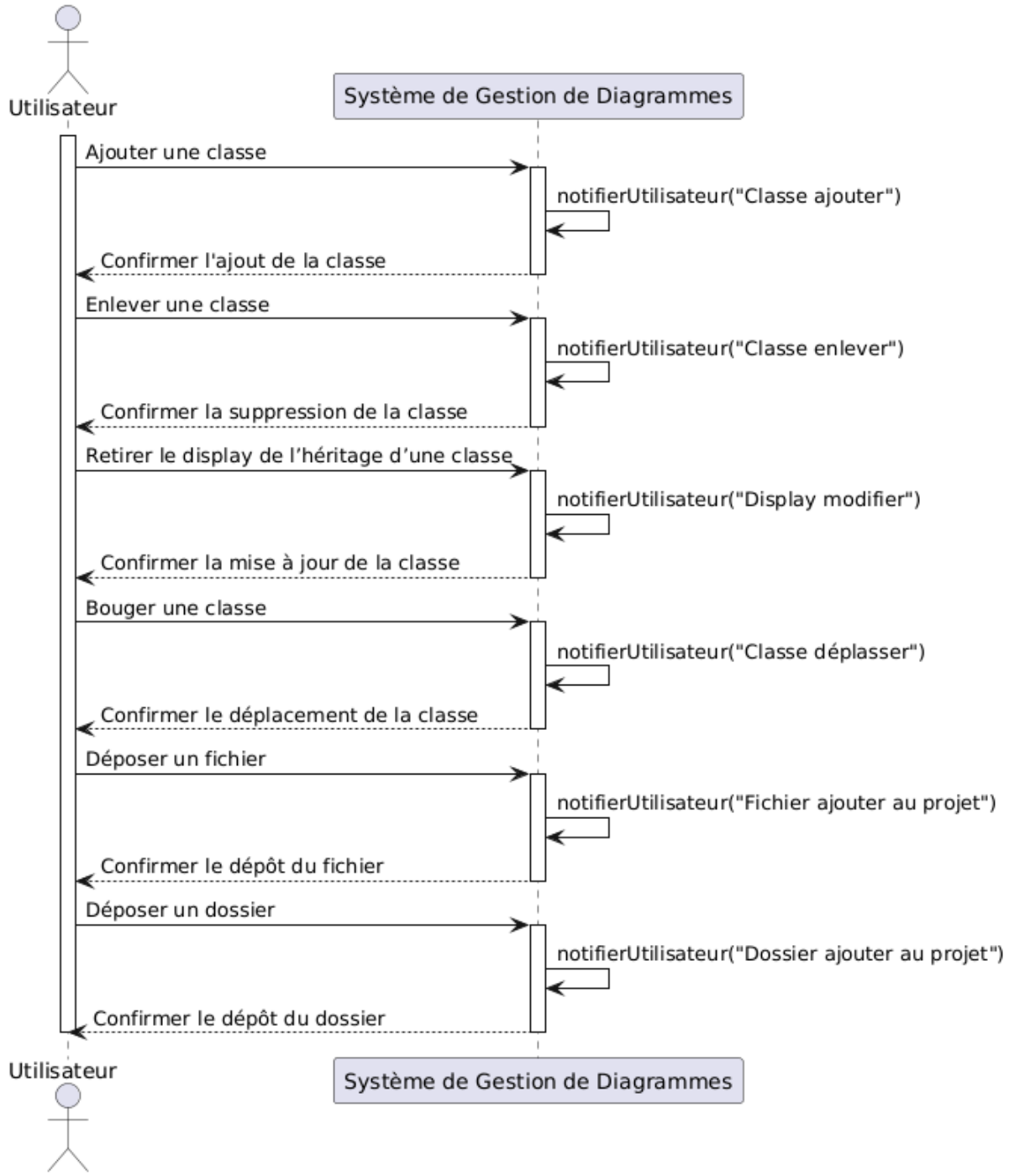
**Scénario d'utilisation 'Modification de l'affichage UML':**

Préconditions :

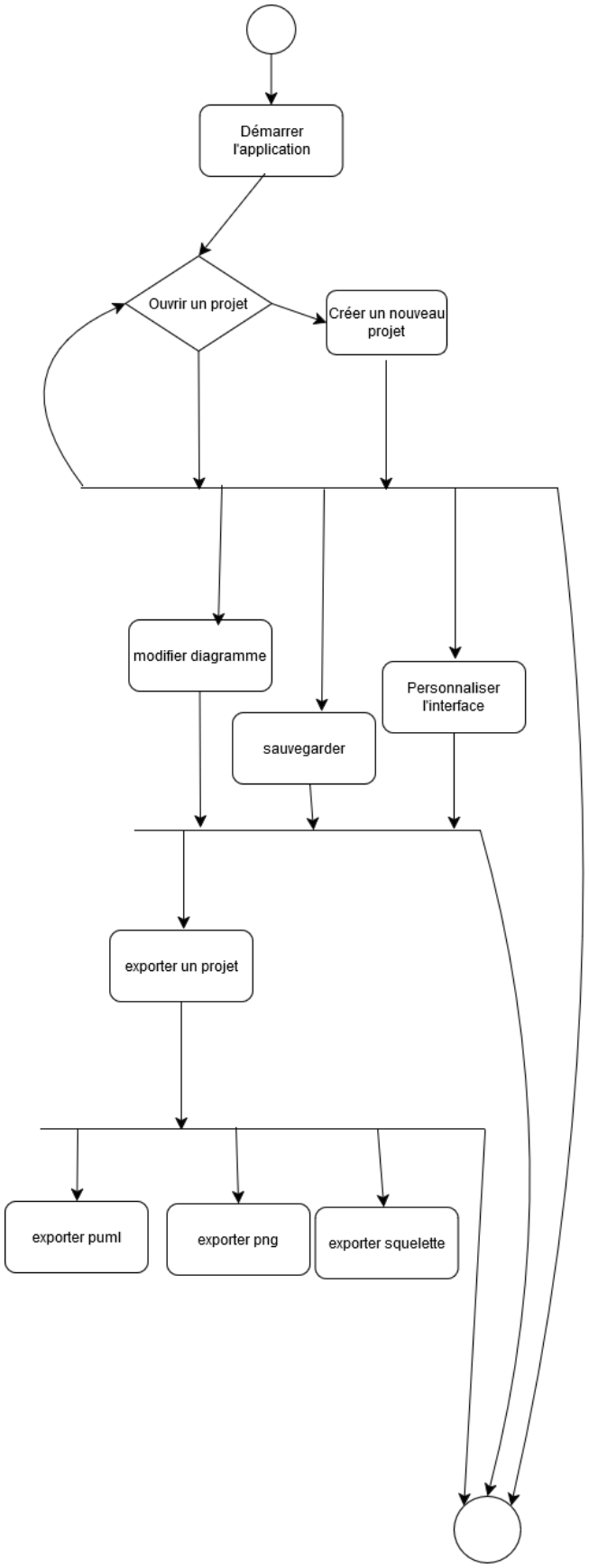
- L'utilisateur a un projet ouvert dans l'application.
- Le projet contient au moins un diagramme de classe

Description des étapes :

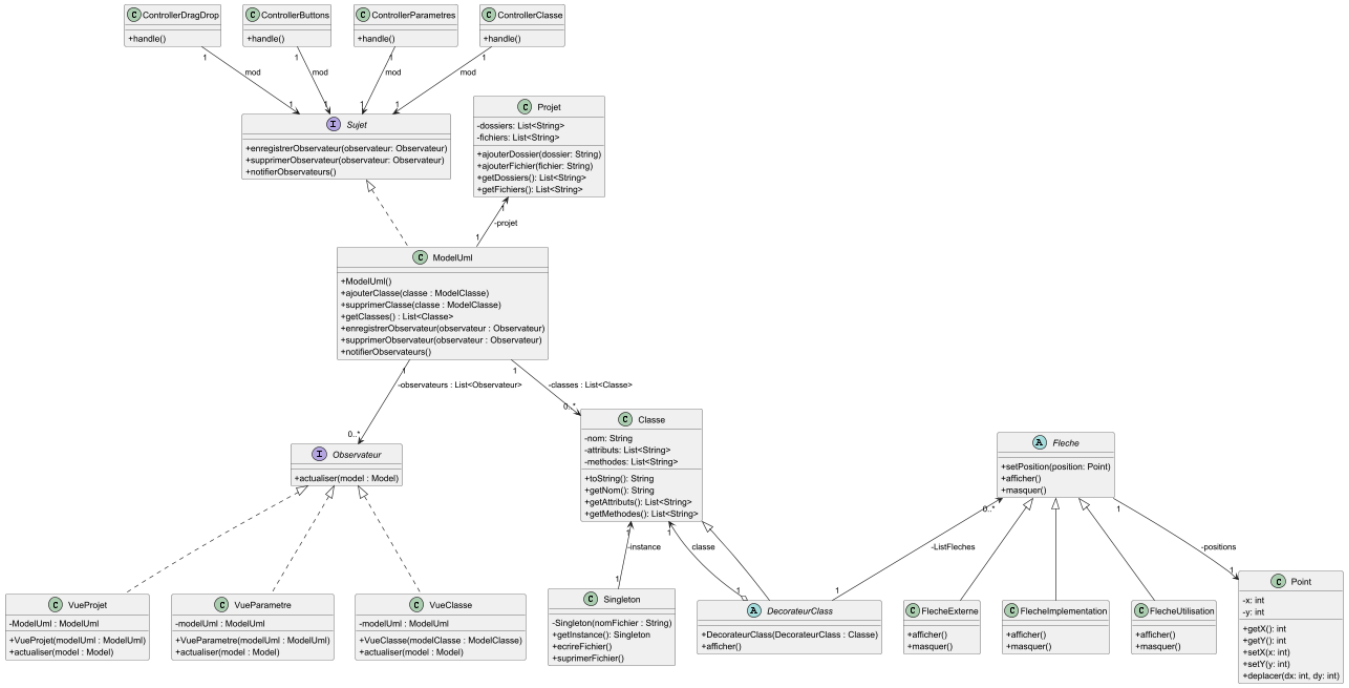
- L'utilisateur appuie sur le bouton ajouter une classe et il peut rentrer les différentes informations relatives à la classe
- L'utilisateur appuie sur le bouton supprimer pour supprimer la classe correspondante
- L'utilisateur drag and drop le fichier pour ajouter les classes contenues pour les ajouter au diagramme
- L'utilisateur drag and drop le dossier contenant des classes afin de les ajouter au projet uml


Utilisateur
Système de Gestion de Diagrammes
Ajouter une classe
notifierUtilisateur("Classe ajouter")
Confirmer l'ajout de la classe
Enlever une classe
notifierUtilisateur("Classe enlever")
Confirmer la suppression de la classe
Retirer le display de l'héritage d'une classe
notifierUtilisateur("Display modifier")
Confirmer la mise à jour de la classe
Bouger une classe
notifierUtilisateur("Classe déplasser")
Confirmer le déplacement de la classe
Déposer un fichier
notifierUtilisateur("Fichier ajouter au projet")
Confirmer le dépôt du fichier
Déposer un dossier
notifierUtilisateur("Dossier ajouter au projet")
Confirmer le dépôt du dossier
Utilisateur
Système de Gestion de Diagrammes

## 4. Diagramme d'activité de l'application



## 5. Diagramme de Classe




```
@startuml
 skinparam classAttributeIconSize 0

 interface Sujet {
   + enregistrerObservateur(observateur: Observateur)
   + supprimerObservateur(observateur: Observateur)
   + notifierObservateurs()
 }
 class ModelUml implements Sujet{

   + ModelUml()
   + ajouterClasse(classe : ModelClasse)
   + supprimerClasse(classe : ModelClasse)
   + getClasses() : List<Classe>
   + enregistrerObservateur(observateur : Observateur)
   + supprimerObservateur(observateur : Observateur)
   + notifierObservateurs()
 }

 interface Observateur {
   + actualiser(model : Model)
 }

 class VueClasse implements Observateur {
   - modelUml : ModelUml
```

```
  + VueClasse(modelClasse : ModelClasse)
  + actualiser(model : Model)
}

abstract class DecorateurClass extends Classe {
   + DecorateurClass(DecorateurClass : Classe)
  +afficher()
}

abstract class Fleche {
  +setPosition(position: Point)
  +afficher()
  + masquer()
}

class FlecheExterne extends Fleche {
  +afficher()
  + masquer()
}

class FlecheImplementation extends Fleche{
  +afficher()
  + masquer()
}

class FlecheUtilisation extends Fleche{
  +afficher()
  + masquer()
}

class Singleton {
  -Singleton(nomFichier : String)
  +getInstance(): Singleton
  + ecrireFichier()
  + suprimerFichier()
}

class ControllerDragDrop {
  +handle()
}

class ControllerButtons {
  +handle()
}

class ControllerParametres {
```

```
  +handle()
}

class ControllerClasse {
  +handle()
}

class VueProjet implements Observateur{
  - ModelUml : ModelUml
  + VueProjet(modelUml : ModelUml)
  + actualiser(model : Model)
}

class VueParametre implements Observateur {
     - modelUml : ModelUml
     + VueParametre(modelUml : ModelUml)
     + actualiser(model : Model)
}

class Projet {
  -dossiers: List<String>
  -fichiers: List<String>
  +ajouterDossier(dossier: String)
  +ajouterFichier(fichier: String)
  +getDossiers(): List<String>
  +getFichiers(): List<String>
}

class Classe {
  -nom: String
  -attributs: List<String>
  -methodes: List<String>
  +toString(): String
  +getNom(): String
  +getAttributs(): List<String>
  +getMethodes(): List<String>
}

class Point {
  -x: int
  -y: int
  +getX(): int
  +getY(): int
  +setX(x: int)
  +setY(y: int)
  + deplacer(dx: int, dy: int)
```

```
 }
 Fleche "1" --> "1" Point : -positions
 ControllerDragDrop "1" --> "1" Sujet : mod
 ControllerButtons "1" --> "1" Sujet : mod
 ControllerParametres "1" --> "1" Sujet : mod
 ControllerClasse "1" --> "1" Sujet : mod
 ModelUml "1" --> "0..*" Classe : -classes : List<Classe>
 DecorateurClass "1" o--> "1" Classe : classe
 Fleche "0..*" <-- "1" DecorateurClass : -ListFleches
 Classe "1" <-- "1" Singleton : -instance
 ModelUml "1" --> "0..*" Observateur : -observateurs : List<Observateur>
 Projet "1" <-- "1" ModelUml : -projet
 @enduml
```

## 6. Détails de Conception

Fonctionnement des fichiers de sauvegarde:

```
@startuml
skinparam classAttributeIconSize 0

class Model {
 notifierControlleur()
}

interface Vue {
  notifier()
}

Model -- Vue

@enduml


@startadg
[
    {
        "Model": {
            "coordinates": [20.1, 14.6],
            "options": {
                "display_constructor": true,
                "display_methods": true,
                "display_fields": true,
                "display_inheritance": true,
```

```
                "display_dependencies": true
            }
        }
    },
    {
        "Vue": {
            "coordinates": [0, 0],
            "options": {
                "display_constructor": true,
                "display_methods": false,
                "display_fields": true,
                "display_inheritance": true,
                "display_dependencies": false
            }
        }
    }
]
@endadg
```

Dans cet exemple, du côté UML, on stocke le nom, méthodes, attributs, les dépendances et héritages. Du côté ADG (format JSON), on stocke les coordonnées sur la grid ainsi que toutes les options de d'affichage (constructeurs, méthodes, etc…).
Ainsi en un seul fichier on peut stocker tout ce qui est nécessaire pour sauvegarder un projet.

## III. Maquette

### 1. Consigne

- une maquette balsamique de l'application

## IV. Planning

### 1. Consigne

- le planning des itérations prévues (trello) et les objectifs de chacune (en terme de cas d'utilisations) avec identification des risques)

### 2. Trello

C.f Annexes

# Développement

# Réunion 13/12/2024

Itération 1

Répertoire Github ICI
Trello ICI

## Fonctionnalités réalisées:

- Terminé Base de l'interface graphique
- Terminé Sauvegarde en .adg
- Terminé Drag and Drop *(depuis l'extérieur de la Scène)*
- Terminé Affichage d'une classe

## Travail personnel:

- **Classe, Analyser, Save, TestAnalyser, TestSave** *(Zacharie)*
  → Analyser un fichier (.java) et retranscrire les informations dans la classe "Classe" ainsi que retranscrire les données de la dite classe dans un fichier (.adg) sous forme (en premier lieu) de script PlantUml. (branche "analyser")

- **MVC : MainUML, ModelUML, VueTitre, style.css, ControleurCreateProject, VueDiagramme** *(Valentin)*
  → Conception complète de la maquette de l'interface graphique, création de cette dernière : sa page d'accueil, ses inputs windows et sa transition vers son état d'affichage diagramme avec un état vide de celui- ci.

- **Observateur, Sujet, ModelUML, ControleurCreateProject, VueDiagramme** *(Ryan)*
  → l'utilisateur peut effectuer un glisser-déposer des fichiers. Lors de l'opération, le chemin absolu des fichiers est affiché dans la console et les coordonnées exactes de l'endroit où les fichiers ont été déposés sont sauvegardées pour une utilisation ultérieure.

- **ClassUml*(Geoffrey)***
  → Affiche une classe dans la partie prévue du logiciel. Lors de l'affichage, les différents niveaux de protection sont affichés par un code couleur afin de les différencier plus simplement.

- **ControllerDragDrop, ModelUML, MainUML** *(Thomas)*
  → Drag and drop un ou **plusieurs fichiers** .java dans un StackPane dans la partie droite de l'interface de la branche ConceptionPartieGraphique. Stockage du chemin absolue du fichier dans une ArrayList<String>.

## Fonctionnalités prévues :

- Flèches implémentation, héritage et dépendances.

# Conception :

# Fiches de réunion

Itération 1 : Fonctionnalité de drag-and-drop, affichage de la classe et sauvegarde dans fichier .adg.

Itération 2 : Flèches implémentation, héritage. Début export PUML.

Itération 3 : Déplacement des classes, fin de l'exportation. Retirer l'héritage d'une classe.

Itération 4 :  Masquer et afficher une classe, création et sélection de projets.

Itération 5 : Enregistrement automatique et exportation en .png.

Itération 6 : Arborescence de fichiers et personnalisation de l'interface.

# Annexes

1. Repo Github
2. Google Doc
3. Trello