Bilan de l'application :

L'application permet de gérer et manipuler des classes dans un diagramme UML avec génération automatique de code Java.

Fonctionnalités principales :

Générer un diagramme de classe:

- **Glissement d'une classe de l'arborescence dans l'espace d'affichage du diagramme**
- **Import d'un dossier contenant des fichier .class**
- **Export en format PNG**
- **Export en format PlantUML**
- **Sauvegarder un diagramme généré via l'interface**
- **Chargement d'un diagramme déjà existant**

Gestion des attributs

- **Affichage des attributs avec leur nom, type et accessibilité (public/private).**
- **Ajout d'attributs dans les fichiers .java avec mise à jour automatique.**

Gestion des méthodes

- **Affichage des méthodes avec leur nom, type de retour, accessibilité et paramètres.**
- **Ajout de méthodes avec génération et insertion dans les fichiers .java Gestion des dépendances**
- **Affichage ou masquage des relations entre classes avec des flèches.**
- **Gestion de la cardinalité des dépendances.**

Gestion des classes

- **Suppression de classes du diagramme.**
- **Déplacement des classes directement sur le diagramme.**

Génération de code

- **Mise à jour automatique et bien structurée des fichiers .java après chaque modification.**

Interface contextuelle

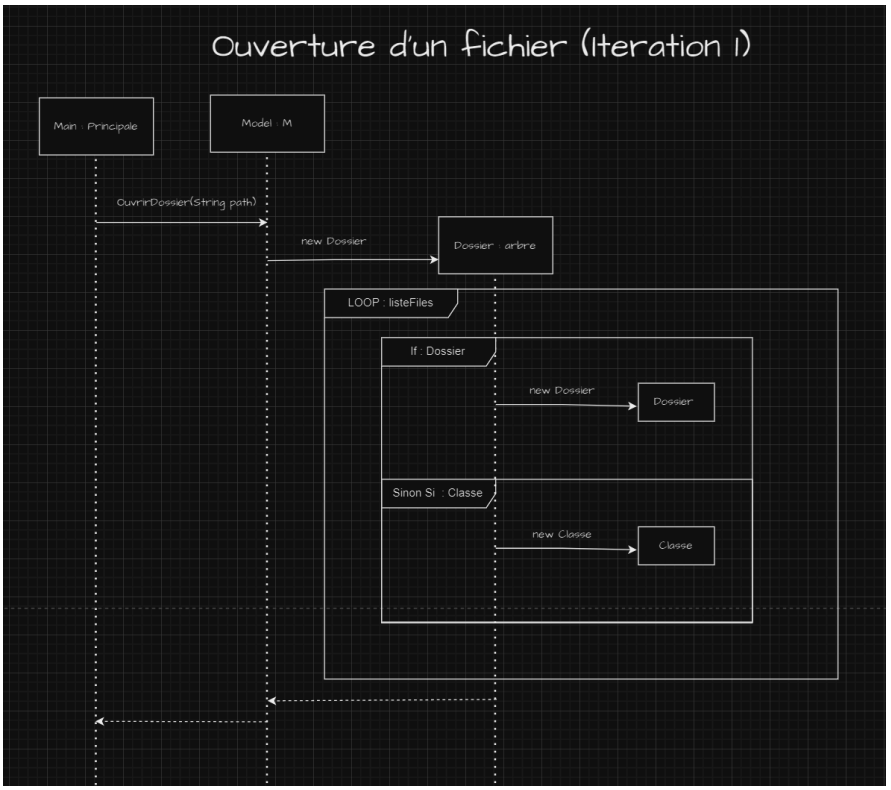
- **Menu contextuel (clic droit) pour accéder aux actions principales sur les classes.**
- **Création rapide d'une classe via le menu clic droit sur le diagramme.**

Autres fonctionnalités

- **Changement de couleur des éléments.**
- **Fonctionnalité d'annulation (retour arrière).**
- **Effacement complet du diagramme.**

Itération 1 :

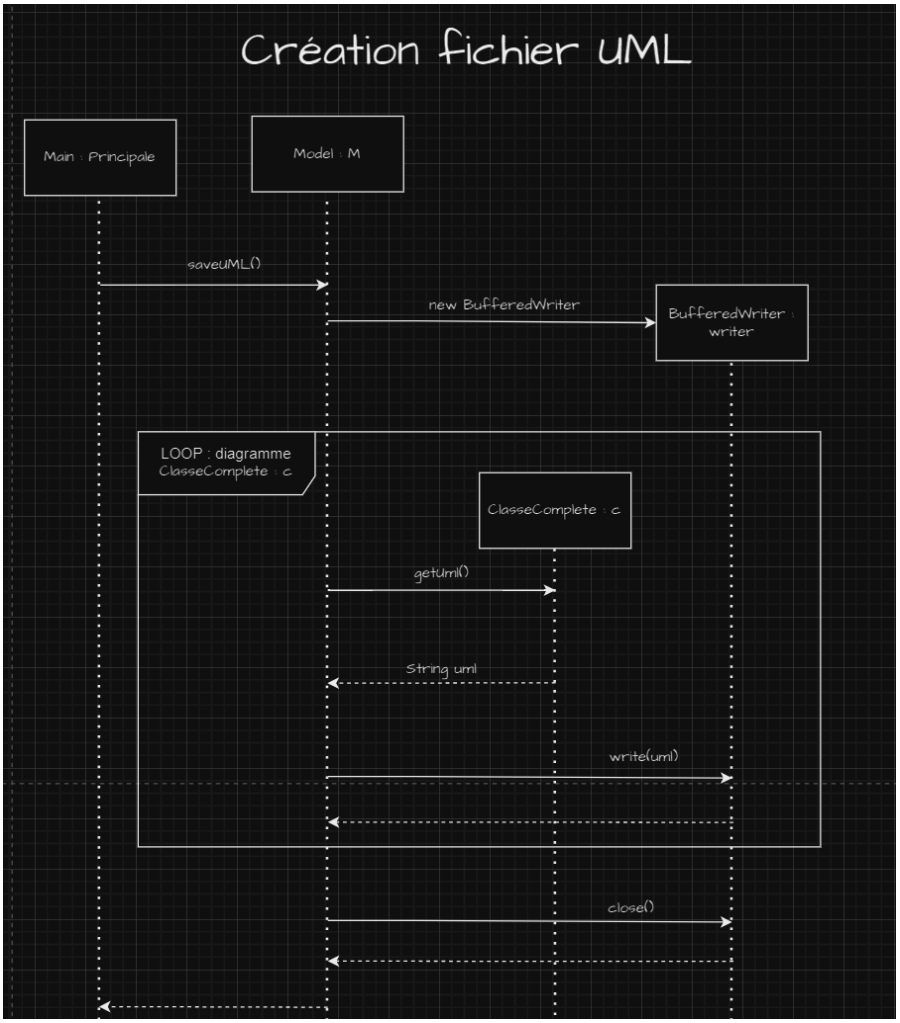
- **Génération d'une arborescence de classes à partir d'un fichier composite. (Hugo R)**
- **Sauvegarde d'une classe en format PlantUML dans un fichier texte et affichage dans la console. (Hugo R)**
- **Utilisation de l'introspection pour accéder aux informations d'un objet classe. (Arthur)**

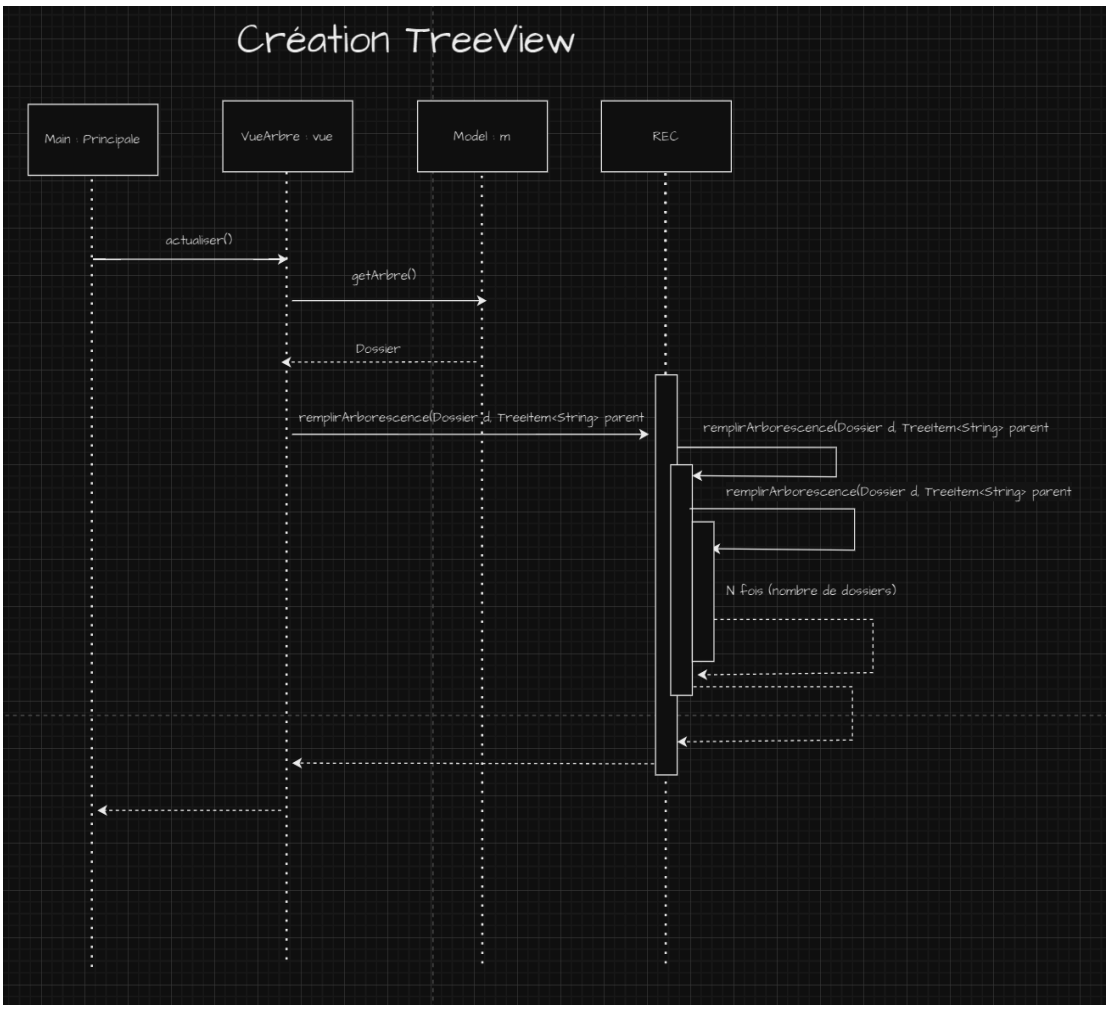


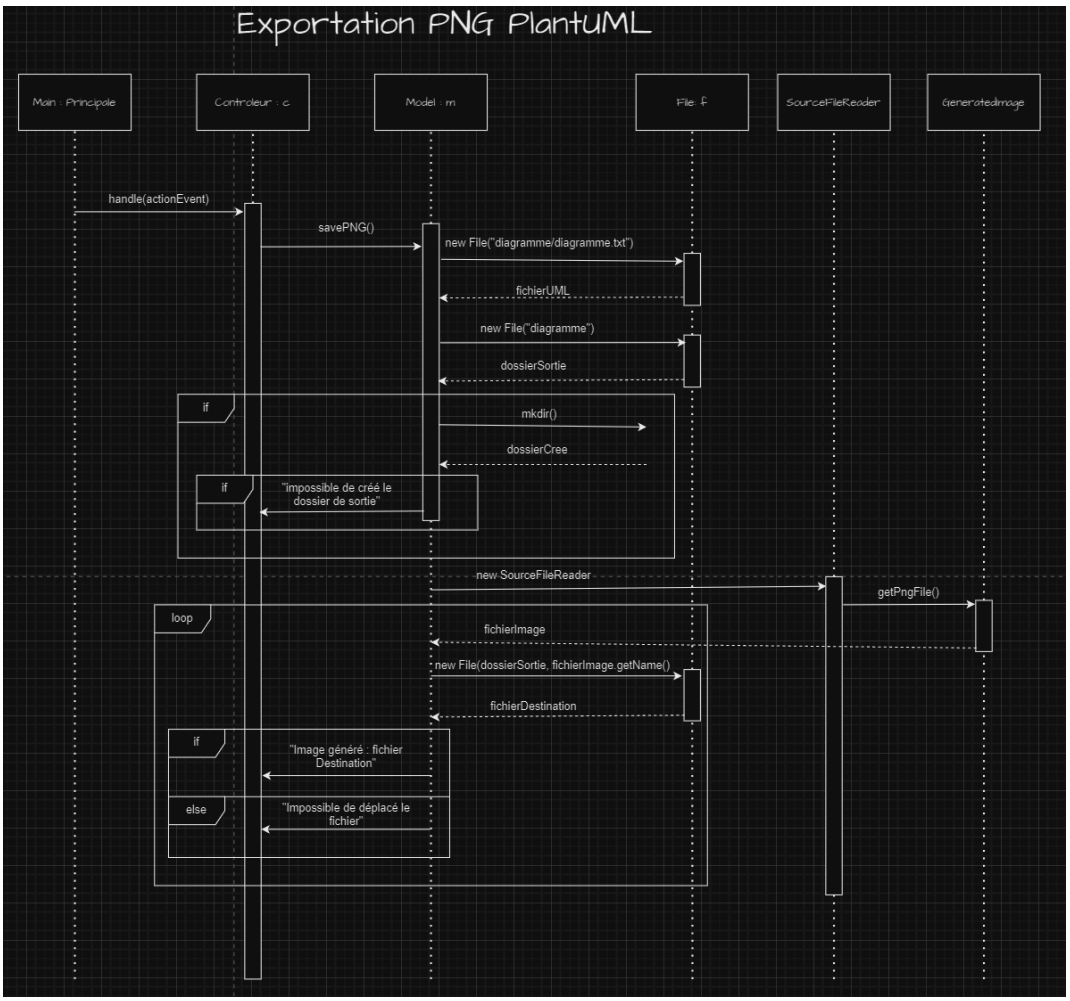
Itération 2 :

- **Exportation des classes au format source PlantUML. (Hugo R)**
- **Exportation des classes au format PNG à partir du PlantUML. (Kévin)**
- **Import d'un dossier contenant des sous-dossiers et fichiers .class depuis n'importe quel répertoire. (Hugo R + Arthur)**

- **Visualisation d’une arborescence de fichiers créée à partir du dossier racine importé. (Hugo B)**
- **Import d’un fichier .class sur le diagramme, visible sous forme de rectangle avec le nom de la classe. (Hugo R + Arthur)**




Création TreeView
Main : Principale
VueArbre : vue
Model : m
REC
actualiser()
getArbre()
Dossier
remplirArborescence(Dossier d, TreeItem<String> parent
remplirArborescence(Dossier d, TreeItem<String> parent
remplirArborescence(Dossier d, TreeItem<String> parent
N fois (nombre de dossiers)



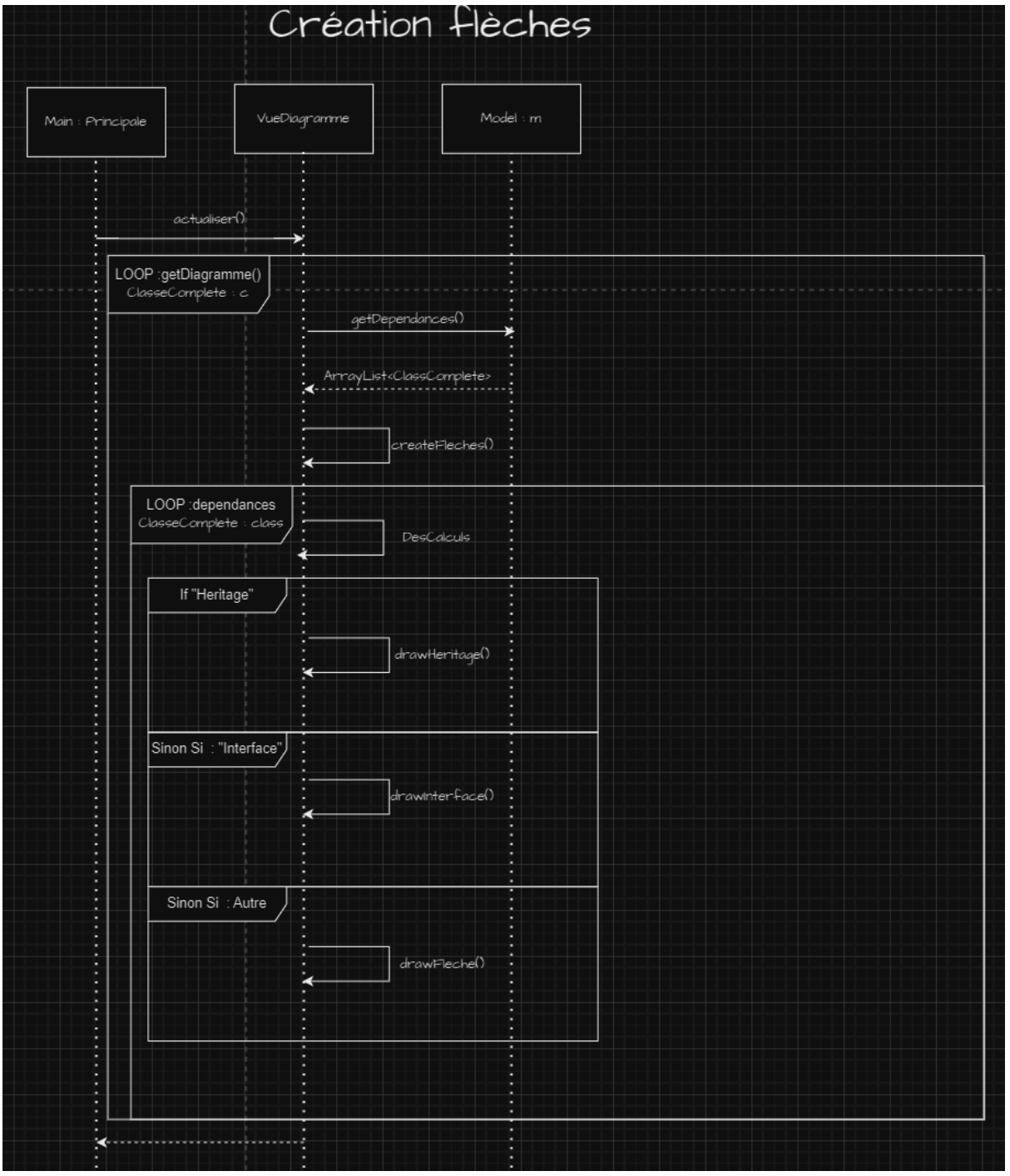
Itération 3 :

- **Affichage des logs dans l'interface pour suivre les actions récentes de l'utilisateur. (Hugo B + Hugo R)**
- **Import d'un fichier .class en glissant-déposant directement sur le diagramme. (Arthur)**
- **Effacement complet du diagramme. (Arthur)**
- **Menu contextuel clic droit. (Kévin + Hugo B)**
- **Affichage des attributs et des méthodes des classes sur le diagramme. (Hugo R)**

Itération 4 :

- **Gestion de la visibilité des attributs, méthodes et dépendances des classes (afficher/masquer). (Hugo B)**
- **Déplacement dynamique des classes sur le diagramme via glisser-déposer. (Arthur)**
- **Visualisation des flèches représentant les dépendances entre classes (héritage, implémentation, association), actualisées en temps réel lors du déplacement des classes. (Hugo R)**

- **Suppression d’une classe spécifique du diagramme. (Hugo B)**

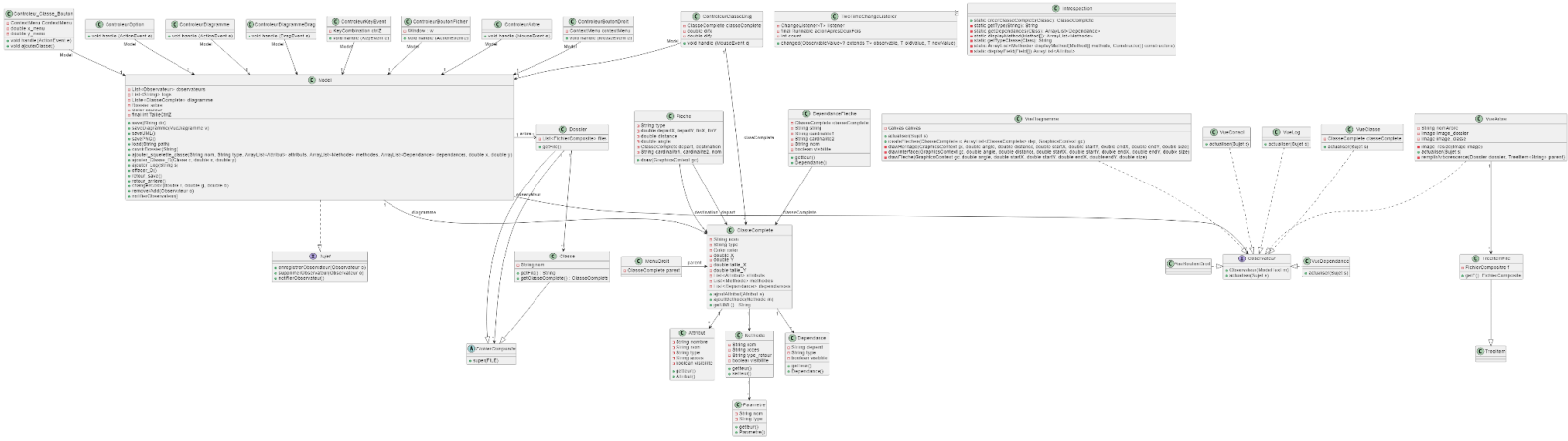


Itération 5 :

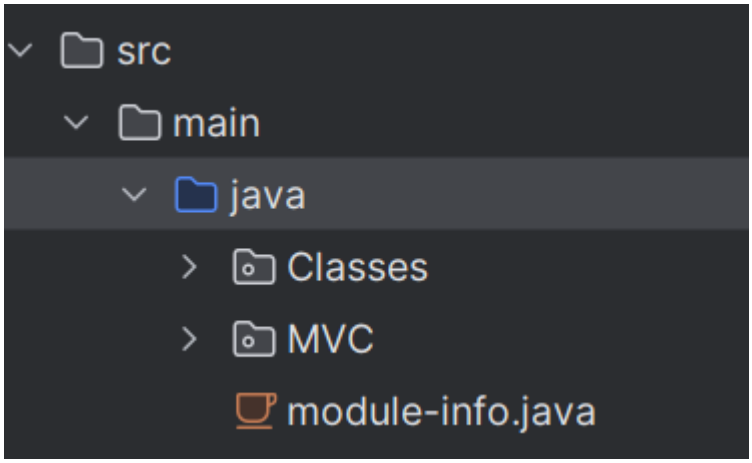
- **Génération automatique du squelette de classes en fichiers .java. (hugo B)**
- **Ajout de méthodes et d’attributs directement dans les fichiers .java. (Hugo B)**
- **Personnalisation de la couleur du diagramme. (Bonus) (Arthur)**
- **Sauvegarde du diagramme en capture d’écran. (Arthur)**

- **Sauvegarde et chargement d'un diagramme dans un format compatible avec l'application. (Hugo R + Arthur)**
- **Ajout d'attributs et de méthodes dans l'affichage des classes sur le diagramme. (Kévin)**
- **Fonctionnalité de retour arrière sur une pile de 20 actions. (Bonus) (Hugo R)**
- **Ajout des constructeurs à l'affichage des classe du diagramme (Hugo R)**

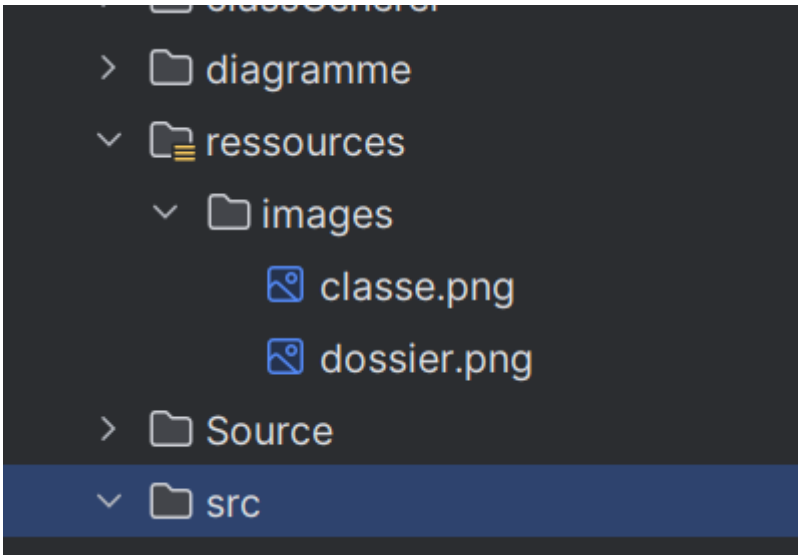
Diagramme de classe : (Le code PlantUML est sur une autre page)



Utilisation de l'application :



Mettre le répertoire java en "Sources".



Mettre le répertoire ressources en “Ressources”.

Code PlantUML du diagramme final :

```
@startuml
Interface Sujet{
+ enregistrerObservateur(Observateur o)
+ supprimerObservateur(Observateur o)
+ notifierObservateur()
}

Interface Observateur{
+ Observateur(ModelText m)
+ actualiser(Sujet s)
}

'Model
Class Model
{
- List<Observateur> observateurs
- List<String> logs
- Liste<ClasseComplete> diagramme
- Dossier arbre
- Color couleur
- final int TailleCtrlZ

+ save(String dir)
+ saveDiagramme(VueDiagramme v)
+ saveUML()
+ savePNG()
+ load(String path)
+ ouvrirDossier(String)
```

```
+ ajouter_squelette_classe(String nom, String type, ArrayList<Attribut> attributs,
ArrayList<Methode> methodes, ArrayList<Dependance> dependances, double x, double y)
+ ajouter_Classe_D(Classe c, double x, double y)
+ ajouter_Log(String s)
+ effacer_D()
+ retour_save()
+ retour_arriere()
+ changerColor(double r, double g, double b)
+ remove/Add(Observateur o)
+ notifierObservateur()
}

Model "1" --> "*" ClasseComplete : diagramme
Model "1" -right-> "1" Dossier : arbre

Class VueLog {
+ actualiser(Sujet s)
}
Class VueClasse {
- ClasseComplete classeComplete
+ actualiser(Sujet s)
}
Class VueDependance{
+ actualiser(Sujet s)
}
Class VueArbre
{
- String nomArbre
- Image image_dossier
- Image image_classe
- image_resize(Image image)
+ actualiser(Sujet s)
- remplirArborescence(Dossier dossier, TreeItem<String> parent)
}
Class VueDiagramme {
- Canvas canvas
+ actualiser(Sujet s)
+ createFleches(ClasseComplete c, ArrayList<ClasseComplete> dep, GraphicsContext gc)
- drawHeritage(GraphicsContext gc, double angle, double distance, double startX, double
startY, double endX, double endY, double size)
- drawInterface(GraphicsContext gc, double angle, double distance, double startX, double
startY, double endX, double endY, double size)
- drawFleche(GraphicsContext gc, double angle, double startX, double startY, double endX,
double endY, double size)
}
```

Class VueConsol{
+ actualiser(Sujet s)
}

Class ControleurBoutonFichier
{
- Window : w
+ void handle (ActionEvent e)
}
Class ControleurArbre{
+ void handle (MouseEvent e)
}
Class ControleurBoutonDroit {
- ContextMenu contextMenu
+ void handle (MouseEvent e)
}
Class Controleur_Classe_Bouton{
- ContextMenu ContextMenu
- double x_menu
- double y_menu
+ void handle (ActionEvent e)
+ void ajouterClasse()
}
Class ControleurOption{
+ void handle (ActionEvent e)
}
Class ControleurDiagramme{
+ void handle (ActionEvent e)
}
Class ControleurDiagrammeDrag {
+ void handle (DragEvent e)
}
Class ControleurClasseDrag {
- ClasseComplete classeComplete
- double difx
- double dify
+ void handle (MouseEvent e)
}
Class ControleurKeyEvent{
- KeyCombination ctrlZ
+ void handle (KeyEvent e)
}

class TreeItemFile

```
{
- FichierCompositre f
+ getF(): FichierComposite
}
TreeItemFile --|>  TreeItem

VueArbre "1" --> "*" TreeItemFile

class TwoTimeChangeListener<T>{
- ChangeListener<T> listener
- final Runnable actionApresDeuxFois
- int count
+ changed(ObservableValue<? extends T> observable, T oldValue, T newValue)
}

Model ..|> Sujet
Model "observateur"--> "*" Observateur


VueConsol ..|> Observateur
VueClasse ..|> Observateur
VueBoutonDroit .right.|> Observateur
VueDependance .left.|> Observateur
VueLog ..|> Observateur
VueArbre ..|> Observateur
VueDiagramme ..|> Observateur


ControleurArbre "Model" --> "1" Model
ControleurBoutonDroit "Model" --> "1" Model
ControleurBoutonFichier "Model" --> "1" Model
Controleur_Classe_Bouton "Model" --> "1" Model
ControleurDiagramme "Model" --> "1" Model
ControleurOption "Model" --> "1" Model
ControleurClasseDrag "Model" --> "1" Model
ControleurDiagrammeDrag "Model" --> "1" Model
ControleurKeyEvent "Model" --> "1" Model

' MVC ^ ----------------------------------------



class Classe{
- String nom

+ getFile() : String
+ getClasseComplete() : ClasseComplete
}
```

```
class ClasseComplete {
- String nom
- String type
- Color color
- double X
- double Y
- double taille_X
- double taille_Y
- List<Attribut> attributs
- List<Methode> methodes
- List<Dependance> dependances
+ ajoutAttribut(Attribut a)
+ ajoutMethode(Methode m)
+ getUML() : String
}

class Attribut {
-String nombre
- String nom
- String type
- String acces
- boolean visibilite
+ getteur()
+ Attribut()
}

class Methode {
- String nom
- String acces
- String type_retour
- boolean visibilite
+ getteur()
+ setteur()
}

class Parametre{
- String nom
- String type
+ getteur()
+ Parametre()
}

Methode "1" --> "*" Parametre

class Dependance{
- String depend
- String type
- boolean visibilite
```

```
+ getteur()
+ Dependance()
}

class DependanceFleche{
- ClasseComplete classeComplete
- String string
- String cardinalite1
- String cardinalite2
- String nom
- boolean visibilite
+ getteur()
+ Dependance()
}

class Dossier {
- List<FichierComposite> files
+ getFile()
}

abstract class FichierComposite {
+ super(FILE)
}

class Fleche{
- String type
- double departX, departY, finX, finY
- double distance
- double angle
- ClasseComplete depart, destination
- String cardinalite1, cardinalite2, nom
+ draw(GraphicsContext gc)
}

Dossier -right-|> FichierComposite
Classe --|> FichierComposite

Dossier "1" --> "*" Classe
Dossier "1" --> "*" FichierComposite

Fleche --> ClasseComplete : depart
Fleche --> ClasseComplete : destination
DependanceFleche --> ClasseComplete : classeComplete

ControleurClasseDrag "1" --> "*" ClasseComplete : claseComplete

ClasseComplete "1" --> "*"Attribut
ClasseComplete "1"--> "*"Methode
```

```
ClasseComplete "1"--> "*"Dependance

class MenuDroit{
- ClasseComplete parent
}

MenuDroit -right-> ClasseComplete : parent

class Introspection {
  + static creerClasseComplete(Classe): ClasseComplete
  - static getType(String): String
  - static getDependances(Class): ArrayList<Dependance>
  - static displayMethod(Method[]): ArrayList<Methode>
  - static getTypeClasse(Class): String
  - static ArrayList<Methode> displayMethod(Method[] methods, Constructor[] constructors)
  - static displayField(Field[]): ArrayList<Attribut>
}
@enduml
```