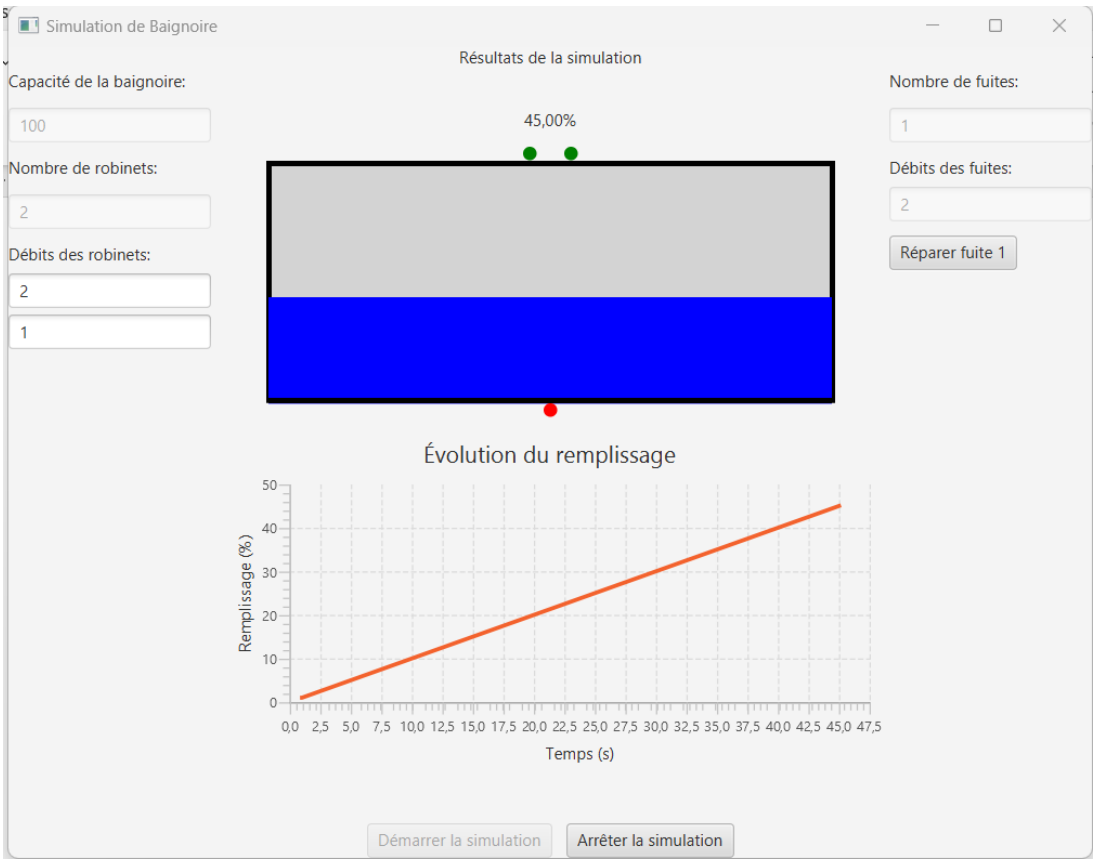
COLOMBANA Ugo

# Documentation technique

## Table des matières

1) La conception de l'interface ..... 2
2) La conception objets ..... 4
3) Les choix et les principes de parallélisations des tâches ..... 6
4) La gestion des accès concurrents ..... 6

# 1) La conception de l'interface




L'interface est organisée pour permettre à l'utilisateur de :

- Configurer les paramètres de capacité de baignoire, du nombre de robinets et des débits des robinets avant le démarrage de la simulation (côté gauche de l'interface).
- Configurer les paramètres du nombre de fuites et des débits des fuites avant le démarrage de la simulation (côté droit de l'interface).
- Réparer des fuites quand l'application est lancée en choisissant la ou les fuites qu'il veut réparer (côté droit de l'interface).
- Voir les résultats en temps réel de la simulation de remplissage incluant le niveau de remplissage, l'eau totale utilisée et le temps de simulation (au milieu de l'interface).
- Contrôler la simulation avec des boutons de démarrage et d'arrêt (en bas de l'interface).

## Description des composants de l'interface

- **BorderPane** : Composant principal permettant d'organiser les autres composants en cinq régions : top, bottom, center, left, right.

- **Top ( Label)** : affiche les résultats de la simulation.

- **Left (VBox)** : Contient les paramètres de saisies comme la capacité de la baignoire, le nombre de robinets et les débits des robinets.

  Il y a plusieurs sous composant ici :

  - **Label** : Nom du paramètre pour la capacité de la baignoire.
  - **TextField** : Champ de saisie pour la capacité de la baignoire.
  - **Label** : Nom du paramètre pour le nombre de robinets.
  - **TextField** : Champ de saisie pour le nombre de robinets.
  - **VBox** : Affiche dynamiquement les champs de débit pour chaque robinet.

- **Center (VBox)** : Contient les éléments graphiques pour visualiser la baignoire qui se remplit et la courbe de remplissage.

  Il y a plusieurs sous composant ici :

  - **Label** : Indique le niveau de remplissage actuel.
  - **Stackpane** : Affiche les composants rectangulaires de la baignoire.
    - **VBox** : Contient les graphiques des robinets et des fuites.
      - **HBox** : Affiche graphiquement les robinets.
      - **HBox** : Affiche graphiquement les fuites.
      - **TextField** : Champ de saisie pour le nombre de robinets.
  - **LineChart** : Composant affichant l'évolution du remplissage.

- **Right (VBox)** : Contient les champs de saisie pour les paramètres des fuites (nombre et débit) et les boutons permettant de réparer les fuites.

  Il y a plusieurs sous composant ici :

  - **Label** : Nom du paramètre pour le nombre de fuites.
  - **TextField** : Champ de saisie pour le nombre de fuites.
  - **VBox** : Affiche dynamiquement les champs de débit pour chaque fuite.
  - **VBox** : Affiche dynamiquement les champs de débit pour chaque fuite.
  - **VBox** : Affiche dynamiquement les boutons pour réparer chaque fuite.

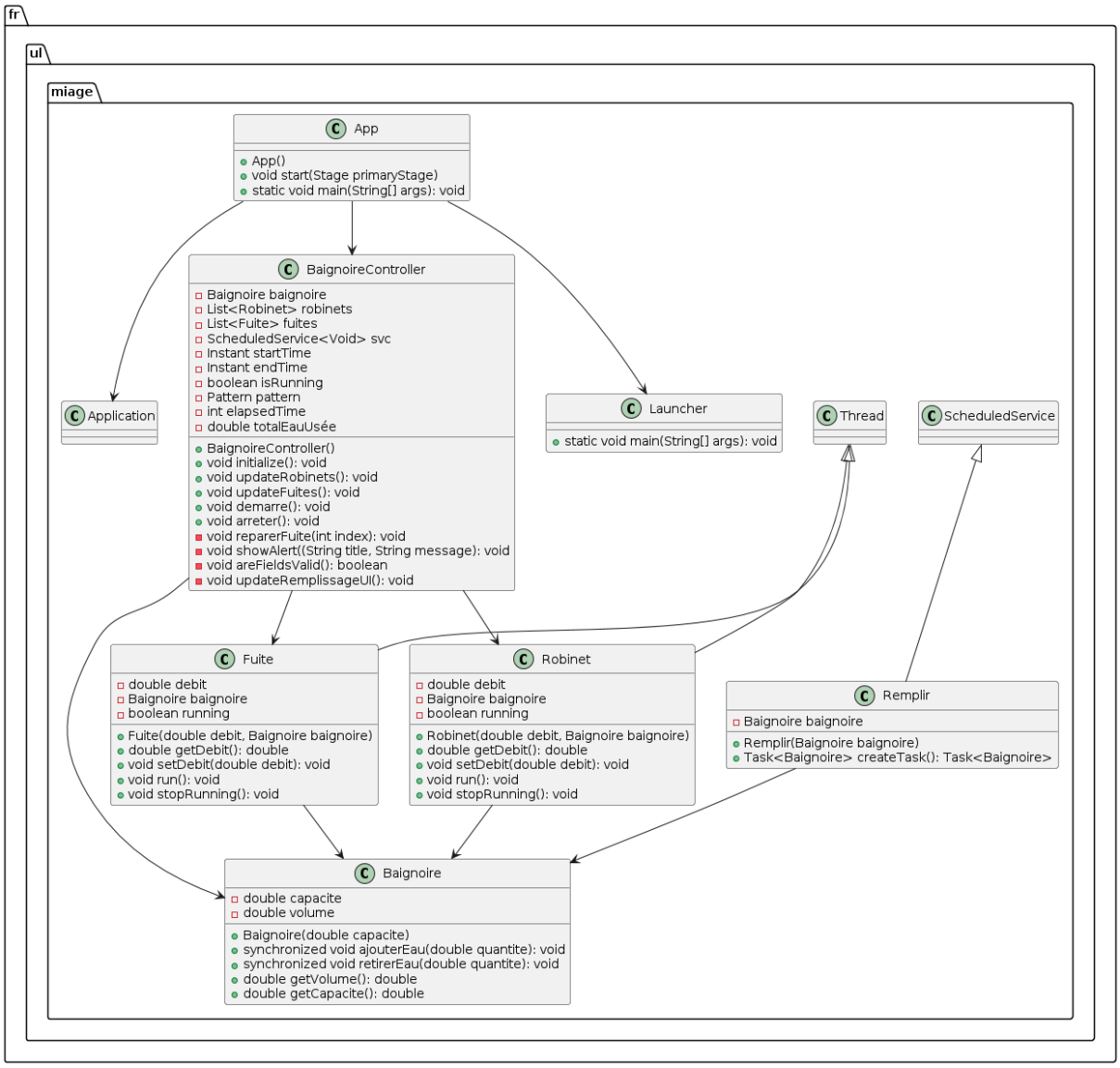
- **Bottom (HBox)** : Contient les boutons pour démarrer et arrêter la simulation.

Il y a plusieurs sous composant ici :

- **Button** : Bouton pour démarrer la situation.
- **Button** : Bouton pour arrêter la simulation.

# 2) La conception objets

## Diagramme de classe du projet



Classes :

- **App** : Initialise et lance l'interface utilisateur.
- **Launcher** : Classe permettant de lancer l'application, elle appelle simplement la méthode « main » de la classe « App ».
- **BaignoireController** : Gère les interactions entre les composants GUI et la logique de l'application.
- **Baignoire** : Représente la baignoire et gère le volume d'eau qu'elle contient.
- **Robinet** : Représente un robinet qui ajoute de l'eau dans la baignoire.
- **Fuite** : Représente une fuite qui retire l'eau de la baignore.
- **Remplir** : Permet de créer des tâches planifiés pour mettre à jour le niveau d'eau de la baignoire.
- **Thread** : Utilisée pour exécuter des opérations de manière concurrente.
- **ScheduleService** : Utilisée pour exécuter des tâches à intervalles réguliers.
- **Application** : Classe de base pour les applications JavaFX.

# 3) Les choix et les principes de parallélisations des tâches

Dans mon projet JavaFX, j'ai utilisé la parallélisation avec les classes « Thread » et « ScheduledService » pour gérer les robinets et les fuites.

- **Thread**

Mes classes « Robinet » et « Fuite » hérite de la classe « Thread » comme nous pouvons le voir sur le diagramme de classe plus haut dans ce rapport.

Ces classes sont conçues pour exécuter des tâches en parallèle comme demandé dans le sujet notamment pour remplir ou vider la baignoire à intervalle régulier (chaque seconde).

- **ScheduleService**

Cette classe est utilisée pour mettre à jour l'interface utilisateur périodiquement sans bloquer le thread principal. Cette approche garantit que l'interface utilisateur reste réactive pendant la simulation.

J'utilise la classe « ScheduleService » dans ma classe « Remplir » permettant de remplir la baignoire de façon continue.

# 4) La gestion des accès concurrents

Afin de garantir la cohésion des données, nous avons utilisé des techniques de gestion des accès concurrents.

Nous utilisons deux méthodes qui ont été déclaré en synchronisées « ajouterEau » et « retirerEau » dans la classe Baignoire. Cela permet de garantir que lorsque un thread modifie le volume d'eau alors aucun autre thread ne peut accéder à ces méthodes jusqu'à ce que la modification soit terminée.

La gestion des accès concurrents dans ce projet nous permettent de créer une application JavaFX performante et fiable, et étant capable de gérer plusieurs opérations en simultanées tout en maintenant l'intégrités des données.