17/03/2022

Alexandre Jaquier et Jonathan Friedli

Rapport Laboratoire 3 : Rivière

Une image contenant extérieur, eau, nature, roche

Description générée automatiquement

# Introduction

Dans le cadre de ce laboratoire, nous devons modéliser un fameux problème de traversée de rivière.

Une famille composée d'un père, d’une mère, de deux filles et de deux garçons est accompagnée d'un policier et d'un voleur menotté. Ils doivent tous traverser une rivière à l'aide d'un bateau.

Contraintes devant être satisfaites en tout temps sur les deux rives et sur le bateau :

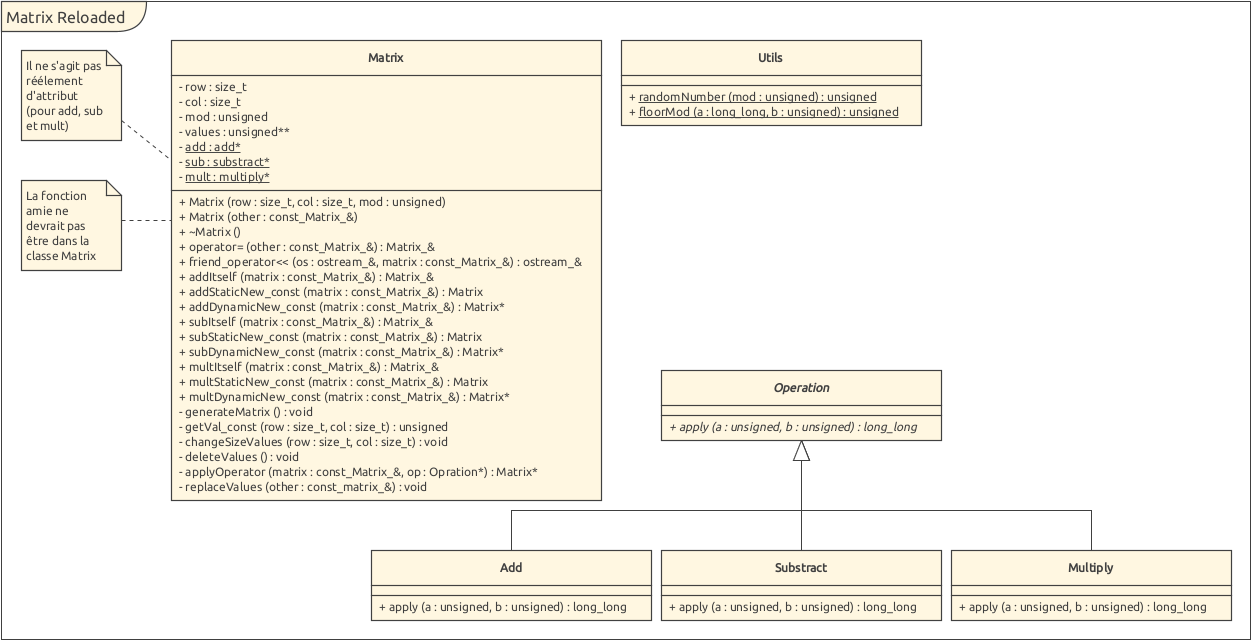
• Il ne peut y avoir que deux personnes au maximum sur le bateau.

• Les enfants et le voleur ne peuvent piloter le bateau (mais peuvent y monter lorsqu’il est à quai).

• Le voleur ne peut pas rester en contact avec un membre de la famille si le policier n'est pas présent.

• Les fils ne peuvent rester seuls avec leur mère si le père n’est pas présent (surement à cause de violence domestique) et réciproquement, les filles ne peuvent rester seules avec leur père si la mère n’est pas présente. (Nous sommes très clairement en présence de parents catastrophique mais le professeur du cours de POA a décidé que nous devrions modéliser ce problème en C++ plutôt que de faire appels aux services de protection de la jeunesse 🤬).

Le but du laboratoire est donc de créer une application en C++ en mode console permettant à l’utilisateur d’introduire les commandes pour embarquer et débarquer des personnes et déplacer le bateau.



# Diagramme UML

# Choix de modélisation et d’implémentation

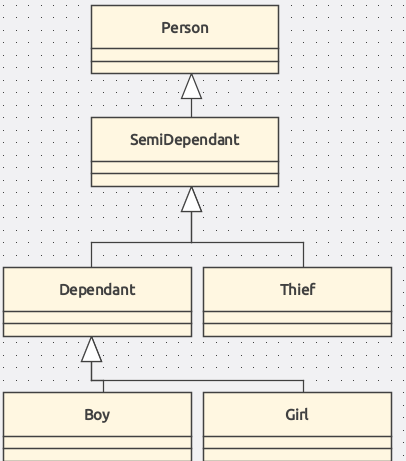
## Person

Afin de modéliser les différents personnes présentes dans le problème, nous avons décidé de créer une classe abstraite **Person** afin de factoriser au maximum le comportement des ces dernières. Ensuite 3 classes vont hériter de **Person : DependantPerson**, **Driver** et **Thief**.

## Driver

Les deux parents et le policier seront ensuite instanciés en tant que **Driver** car c’est les 3 seules personnes pouvant conduire le bateau.

## DependantPerson

Cette classe nous permet de factoriser le fonctionnement des enfants, surtout au niveau de la présence/absence de leurs parents. De cette classe, héritent les classes **Boy** et **Girl.** Ces dernières sont utiles afin de personnaliser le message d’erreur (Fille avec son père mais sans sa mère ou Garçon sans son père mais avec sa mère).

## Thief

Nous avons choisi de faire une classe séparée pour le voleur car ce dernier ne dépend que d’une seule autre personne (le policier). Nous aurions pu faire un héritage plus complexe (voir image ci-contre) mais cela aurait trop complexifié le problème.

## Container

Comme indiqué dans la consigne, nous avons créé une classe **Container** dont héritent les classes **Boat** et **Banks**. Cela nous permet de factoriser le fonctionnement de ces containers.

# Tests effectués

Nous avons effectué divers tests afin de vérifier que toutes les règles sont respectées.

|  |  |
| --- | --- |
| **Test** | **Résultat attendu** |
| Il y a au maximum 2 personnes dans le bateau. Si nous tentons d’embarquer une 3ème personne, un message d’erreur s’affiche. |  |
| Le bateau peut bouger uniquement si le père, la mère ou le policier est dedans. |  |
| Si la mère laisse une des filles avec le père, un message d’erreur s’affiche. |  |
| Si le père laisse un des fils avec la mère, un message d’erreur s’affiche. |  |
| Si le voleur n’est pas tout seul en en présence du policier, un message d’erreur s’affiche. |  |
| Il est impossible d’embarquer une personne qui ne serait pas sur la même rive que le bateau. |  |
| Si l’utilisateur entre une input ne correspondant à aucune des commandes prédéfinies, un message d’erreur s’affiche. |  |
| Si l’utilisateur entre une input sans paramètre (p, m, h, r ou q) et décide de rajouter du texte après. Cela résultera en un input invalide. |  |
| Si l’utilisateur entre l’une des input suivante (e <nom> ou d <nom>) mais donne un nom inconnu. Cela résultera en un input invalide |  |