POO2 - Laboratoire 3

Rivière

Cours : POO2 | Auteurs : Zwick Gaétan, Maziero Marco | 03.05.2021

Table des matières

[Introduction 3](#_Toc70929041)

[Structure du code 3](#_Toc70929042)

[La classe Controller 3](#_Toc70929043)

[Transformation de l’entrée en objets du jeu 3](#_Toc70929044)

[La validation des états de jeu 3](#_Toc70929045)

[La classe Person et ses enfants 4](#_Toc70929046)

[Le rôle de l’héritage 4](#_Toc70929047)

[La classe container 4](#_Toc70929048)

[La dépendance mutuelle Person-Container 4](#_Toc70929049)

[Diagramme de classes 4](#_Toc70929050)

[Tests 5](#_Toc70929051)

[Annexes 6](#_Toc70929052)

[Diagramme de classes UML 6](#_Toc70929053)

# Introduction

Le programme à concevoir est un petit jeu de transport de personnes d’un côté gauche d’une rivière à un côté droit à l’aide d’un bateau 2 places. Les personnes à déplaces sont une famille : père, mère, deux filles et deux fils ainsi qu’un policier et un voleur. Ces personnes sont soumises à différentes contraintes et ne peuvent rester seules entre elles.

La structure du code choisie pour ce laboratoire s’inspire du diagramme UML donné dans la donnée et permet la gestion organisée de ces différentes contraintes afin de faciliter leur modifications et l’ajout de nouvelles personnes dans le jeu.

# Structure du code

## La classe Controller

La classe gérant toutes les mécaniques du jeu est Controller. Elle offre des fonctions permettant d’afficher l’état du jeu actuel et le menu d’aide aux différentes commandes ainsi qu’une fonction nextTurn() s’occupant de jouer le prochain tour.

Au tout début, le constructeur instancie dynamiquement toutes les personnes du jeu et les place dans une liste (cette liste est utile pour réinitialiser l’état du jeu ou convertir du texte en une personne). Toutes ces personnes sont ensuite placées dans le container « rive gauche ».

nextTurn() s’occupe ensuite d’effectuer les actions suivantes :

* Récupérer l’entrée utilisateur
* Transformer l’entrée reçue en objets du jeu
* Effectuer les actions
* Afficher les éventuels messages de succès ou d’erreur

### Transformation de l’entrée en objets du jeu

L’entrée utilisateur est une valeur textuelle et lorsque l’utilisateur entre les commandes e <nom> et d <nom>, il faut convertir la partie <nom> en objet Person. La fonction findPersonByName() utilise un std::find\_if qui va itérer sur toutes les personnes du jeu et comparer leur nom a la valeur entrée en paramètre.

Une fois l’objet Person récupéré, la fonction findPersonContainer() permet de récupérer le container dans lequel se trouve la personne, le tout afin d’embarquer ou débarquer la personne selon la commande entrée.

## La validation des états de jeu

A chaque tour, il faut vérifier que l’action de l’utilisateur respecte toutes les contraintes du jeu.

C’est là qu’intervient la fonction validateState() de la classe Container. Cette fonction retourne un booléen indiquant si toutes les personnes contenues dans ce container au moment de l’appel respectent toutes les contraintes du jeu ou pas.

Pour vérifier si les contraintes sont respectées, chaque personne possède une fonction canStayWith() qui elle aussi retourne un booléen indiquant si la personne en question peut rester avec celles du container passé en paramètre.

validateState() va donc itérer sur toutes les personnes du container et pour chacune de ces personnes, elle va appeler la fonction canStayWith() avec le reste des personnes du container.

validateState() et canStayWith() sont deux fonctions présentes dans deux classes qui ont besoin l’une de l’autre. Il y a donc une dépendance mutuelle.

## La classe Person et ses enfants

La classe Person possède plusieurs enfants qui représentent les objets pouvant se déplacer au travers des différents containers. Ces personnes ont un nom et un booléen indiquant si elles sont aptes à conduire le bateau ou pas.

La fonction canStayWith() est virtuelle pure et permet d’exprimer les contraintes du jeu grâce à l’héritage.

### Le rôle de l’héritage

Chaque type de personne est représenté comme enfant de la classe Person et se doit d’implémenter la fonction canStayWith() qui parcours un container et indique si la personne peut cohabiter dans le container avec les autres.

**Attention** : la vérification faite dans la classe correspond aux contraintes touchant uniquement la personne concernée. C’est à dire que la contrainte du voleur sans le policier se fait dans la classe du voleur et non du policier ou encore la vérification que le fils est avec sa mère sans son père se fait dans la classe du fils.

## La classe container

Comme expliqué plus haut, la classe Container représente une liste de Person et offre une fonction itérant sur toutes ses personnes pour vérifier que la cohabitation respecte toutes les contraintes du jeu.

## La dépendance mutuelle Person-Container

Ces deux classes ont besoin l’une de l’autre. En effet, un container contient des personnes et les parcours pour valider son état, tandis qu’une personne prend un container en paramètre de sa fonction canStayWith() pour vérifier que les contraintes sont respectées.

Pour permette cette dépendance, les classes ont été déclarées avant les #include dans le code.

## Diagramme de classes

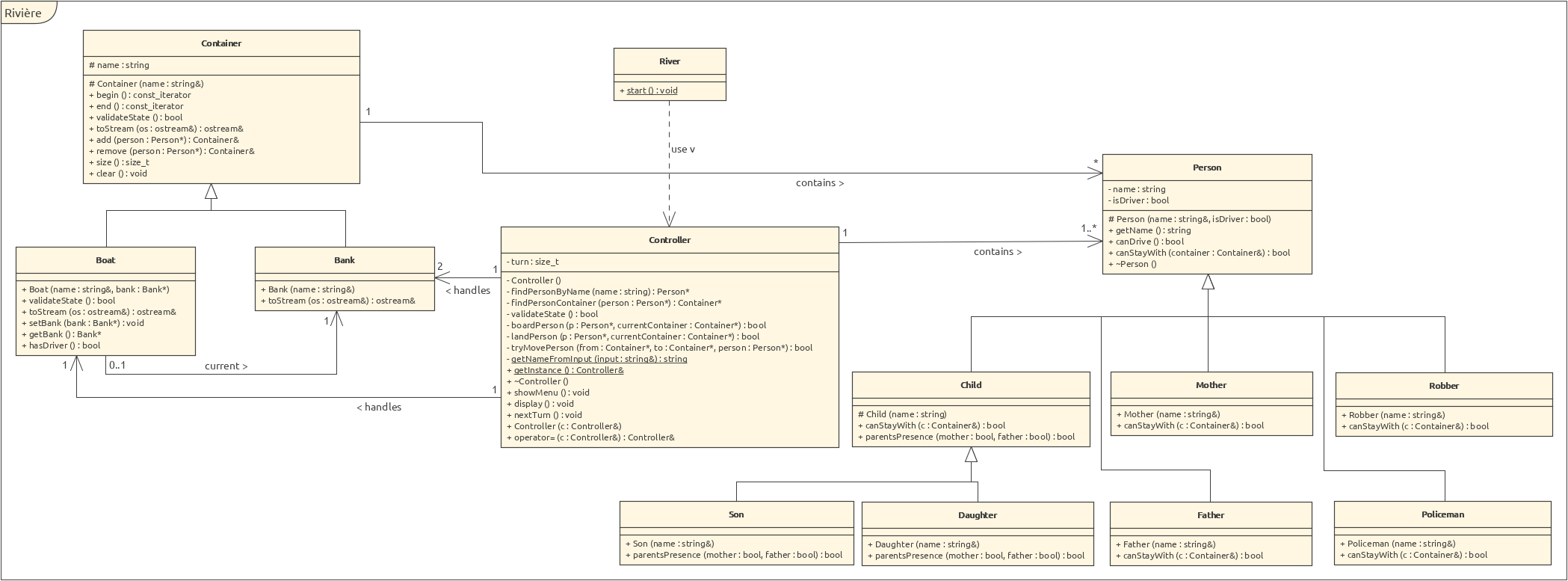
Le diagramme de classes UML est disponible en [annexe](#_Diagramme_de_classes).

# Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Description du test** | **Suite d’inputs** | **Résultat attendu** | **Réussi ?** |
| Que deux personnes au maximum sur le bateau | e policier | Sortie « Le bateau ne peut contenir que 2 personnes max » | OK |
| e voleur |
| e paul |
| Les enfants et le voleur ne peuvent piloter le bateau | e voleur | Sortie « Aucun conducteur present sur le bateau » | OK |
| m |
| d voleur |
| e paul |
| e jeanne |
| m |
| Le voleur ne peut rester avec la famille sans le policier | e policier | Sortie « Voleur seul avec la famille sans le policier » | OK |
| Les fils sans leur père avec leur mère | e pere | Sortie « Garcon avec sa mere sans son père » | OK |
| Les filles sans leur mère avec leur père | e mere | Sortie « Fille avec son pere sans sa mère » | OK |
| Embarquer une personne déjà sur le bateau | e voleur | Sortie « La personne voleur est deja sur le bateau » | OK |
| e voleur |
| Débarquer une personne déjà sur la rive | d mere | Sortie « La personne mere n'est pas sur le bateau » | OK |
| Nom de personne inexistante | e nomInconnu | Sortie « La personne nomIconnu n’existe pas » | OK |
| Commande inconnue | 1234567890 | Sortie « Commande inconnue » | OK |
| Commande vide |  | Demande l’entrée à nouveau | OK |
| Reset de l’état du jeu | r | Replace toutes les personnes sur la rive gauche et met le compteur de tours à 0 | OK |
| Commande pour afficher l’aide | h | Affiche l’aide aux commandes | OK |
| Commande pour afficher l’état du jeu | p | Affiche l’état du jeu actuel | OK |
| Gagner la partie en faisant traverser tout le monde | Faisable en ~50 entrées | Sortie « Partie terminee, felicitations ! » | OK |

# Annexes

## Diagramme de classes UML



Fichier image et Slyum disponibles dans l’archive de rendu.