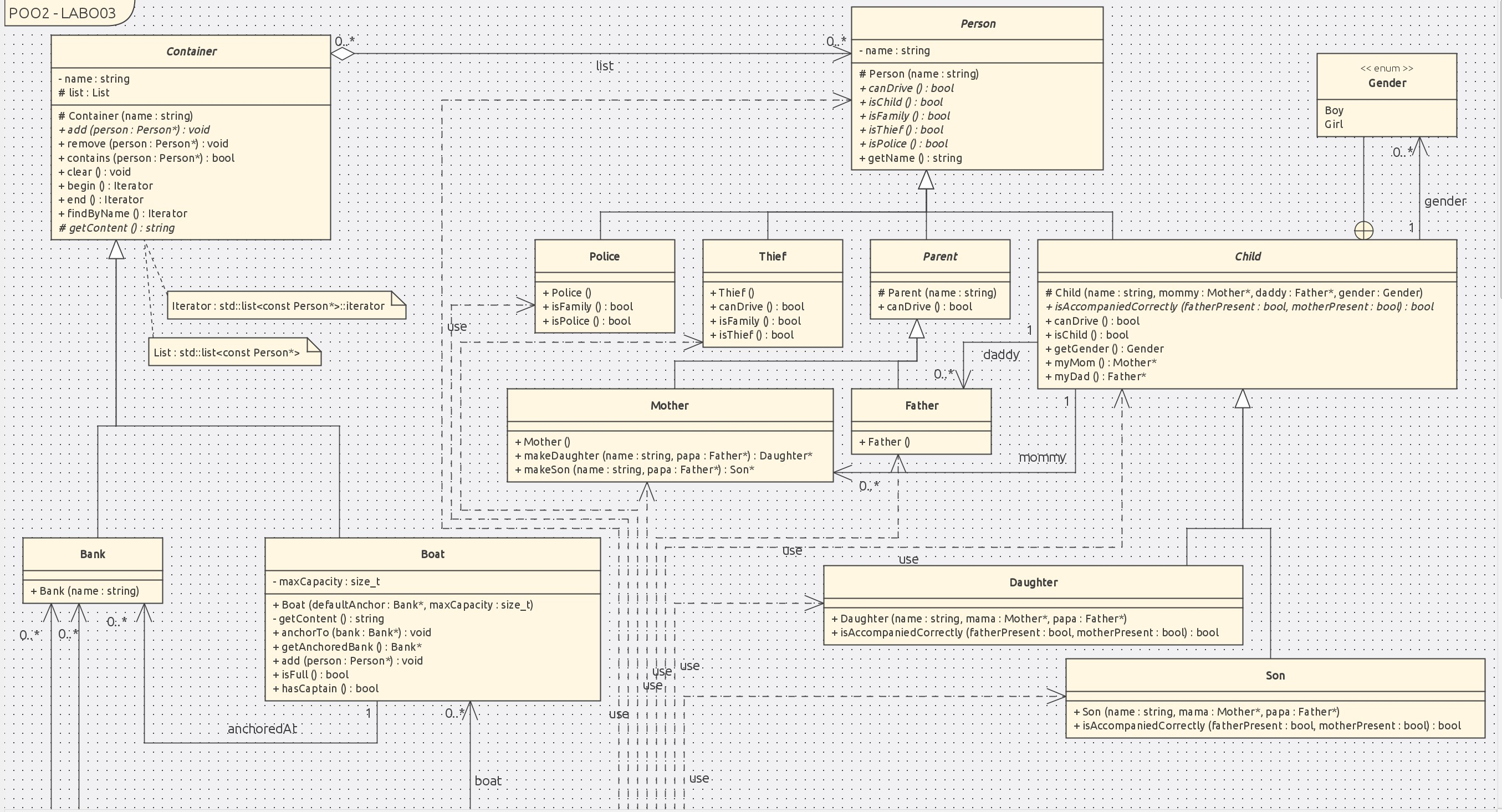
POO2 – Laboratoire 3

Rivière

## UML

Une image contenant table

Description générée automatiquement

## Conception

Une brève description des particularités de cette implémentation ainsi que des décisions prises.

Containers

Les rives et le bateau sont conçus comme des conteneurs de personnes. Chaque conteneur possède un nom (Gauche, Droite, Bateau) et une liste de Person\*. La classe Container est donc spécialisée à contenir des Person\* et n’est pas compatible avec d’autres types d’éléments.

La seule méthode abstraite est la méthode « add ». Le comportement de « add » change dépendant s’il s’agit d’une rive ou du bateau. Le reste des méthodes reste inchangé par les sous-classes de conteneur mais le bateau implémente davantage de fonctionnalités.

La liste de personnes contenues dans le conteneur est stockée comme objet, ainsi pas de destructeur, de constructeur de copies et d’opérateur d’affectation particuliers sont nécessaires. L’allocation et la désallocation des Person sont gérées par la classe client (« Controller »). Les constructeurs, destructeurs et opérateurs par défaut vont s’appuyer sur ceux définies dans la std::list.

Bank

Cette classe ne change rien dans le comportement de la super classe, mais il semble judicieux et plus propre (dans le sens d’une future évolution) que la classe « Container » reste abstraite et d’avoir un nom plus parlant pour le conteneur rive (Bank).

Boat

Cette classe redéfinit le comportement de la méthode « add » de la superclasse pour gérer la contrainte de capacité.

Elle implémente un ensemble de fonctionnalités propre au bateau. Pour permettre l’embarquement et désembarquement sur la rive, elle garde le pointeur sur la rive sur laquelle le bateau est ancré. Pour sa navigation dans la rivière, elle offre des fonctionnalités permettant de vérifier que le capitaine est présent à bord. Le capitaine est un « Person » qui « canDrive ».

La classe Boat modifie son affichage en redéfinissant la méthode « getContent ».

Person

Les personnes dans le programme sont représentées par un ensemble de classes qui permet leur distinction. La classe abstraite Person contient un nom et un ensemble de méthodes abstraites que les classes enfants vont redéfinir par rapport à leurs caractéristiques.

L’ensemble des méthodes virtuelles : isChild, isFamily, isPolice, isThief servent à pouvoir distinguer les types Person dans la classe « Controller » afin d’implémenter les contraintes fonctionnelles décrites dans l’énoncé. En effet, les classes « Container » contiennent la liste de type Person\*, ces méthodes servent pour éviter le transtypage dynamique au moment où on veut savoir la vraie nature de la Person derrière le pointeur.

Précédemment, nous avions implémenté un enum interne PersonType dans Person qui énumère les différents types qui nous intéressent {Family, Police, Thief}, l’utilisation des méthodes virtuelles semble mieux respecter les principes OCP (du fait qu’on utilise une enum interne dans la classe du client). L’utilisation des méthodes virtuelles est aussi conseillée dans les slides du cours pour ce genre de besoins.

Child

Les classes concrètes de Child, Daughter et Son sont fabriquées par la classe Mother en passant le pointeur sur Father en argument. Il est nécessaire de savoir qui sont les parents (Father et Mother) d’un enfant pour implémenter les besoins fonctionnels décrits.

La clase Child déclare une méthode purement abstraite « isAccompaniedCorrectly » qui va différer dépendant s’il s’agit de la fille ou du fils. Cette méthode reçoit 2 boolean qui indiquent si le papa et maman sont présents dans le même conteneur. La classe « Controller » est responsable de fournir ces informations en utilisant les getters « child.getMom() » et « child.getDad() » pour les rechercher dans le conteneur. L’enfant décide s’il est accompagné correctement ou pas, mais la classe « Controller » décide quoi faire avec cette information. Rien n’empêche la classe « Controller » de mettre un enfant dans un conteneur même si l’enfant n’est pas content (pas de lancement d’exceptions de la part de Child).

Controller

La cohérence du programme est donc gérée par la classe « Controller » en s’appuyant sur les données obtenues avec des méthodes publiques disponibles dans des objets gérés.

Cette classe s’occupe de la coordination du programme en se basant sur les inputs du l’utilisateur. Elle est responsable de l’allocation / désallocation des Person, initialisation des Containers, gestions des entrée/sorties utilisateur et de la gestion des tours.

Initialisation

La classe « Controller » instancie 3 Conteneurs que sont la rive gauche, la rive droite et le bateau ancré sur la rive gauche par défaut. Ensuite, elle initialise les Person nécessaires, affiche l’état actuel et les commandes possibles à l’utilisateur.

La méthode « initPersons » fait l’allocation des Person nécessaires pour le programme. « destroyPeople » fait la désallocation de Person présents dans les 3 containers (rive gauche, rive droite et bateau). Elle est utilisée pendant la fin du programme mais aussi pour la réinitialisation demandée par l’utilisateur. De ce fait, elle n’est pas en relation directe avec les Person dans les conteneurs mais possède des dépendances vers chacun des types de Person. La dépendance vers les types « Person » peut être évité en implémentant une fabrique de chacun des types de Person.

Commandes

La méthode « handleCommand » permet d’effectuer une action par rapport à l’entrée utilisateur.

Les méthodes utilitaires statiques split et trim sont utilisées pour préparer les entrées utilisateur pour le traitement. La méthode split va « tokeniser » l’entrée utilisateur et est particulièrement utile pour des commandes avec arguments.

Le token à l’indice 0 identifie la commande. Chaque commande possible est représentée par une entrée dans l’enum « MenuOptions », la méthode « resolveMenu » permet de mapper un string sur une de ces options.

Gestion des contraintes

Les contraintes fonctionnelles sont vérifiées par la classe « Controller ». Elles sont vérifiées à chaque commande (embark et disembark). Toutes les contraintes concernent uniquement les commandes embark et disembark.

La méthode embark vérifie d’abord que le bateau n’est pas déjà plein avant l’embarquement. La méthode disembark n’a pas besoin de vérifier ceci car les rives ont des capacités infinies.

Ensuite on recherche la personne par nom dans la rive (si embark) ou dans le bateau (si disembark).

Une fois la personne trouvée, on contrôle les règles d’accompagnement sur le bateau et la rive concernée. La méthode utilitaire statique « controlAccompanimentRules » va vérifier toutes les règles d’accompagnement (papa, maman, enfant) mais aussi (famille, policier, voleur). Cette méthode fait appel à plusieurs autres méthodes utilitaires statiques pour le contrôle d’accompagnement.

La méthode « controlAccompanimentRules » fait une copie des conteneurs concernés, déplace la Person dans ces copies et vérifie que les règles sont respectées après le déplacement. En gros, on déplace d’abord (dans la copie) puis on vérifie si c’est bon.

## Tests

Pour réaliser ces tests ci-dessous, nous avons automatisé la saisie de l’utilisateur au moyen de l’outil **std::istringstream** qui nous permet dès lors d’utiliser le flux d’entrée avec des commandes prédéfinies, sans devoir les saisir nous-même.

Test afin de résoudre le problème de River

Afin de tester notre programme, nous avons essayé de faire passer toutes les personnes d’une rive à une autre tout en respectant les contraintes mentionnées dans l’énoncé. Un détail des commandes pour chacun des tests est disponible en annexe.

Une image contenant table

Description générée automatiquementUne image contenant table

Description générée automatiquement

Tests des fils avec leur mère sans leur père et des filles avec leur père sans leur mère

Les tests suivants correspondent aux contraintes #4 et #5 de la donnée indiquant que les fils ne peuvent être seuls avec leur mère si le père n’est pas présent et inversement avec les filles.

Dans le cas des fils, nous sommes partis du problème initial puis avons embarqué le père (provoquant une erreur), puis un des deux fils et ensuite le père (provoquant aussi une erreur puisque le second fils se retrouve sans père mais avec sa mère)

Dans le cas des filles, nous avons réitéré le même procédé mais en embarquant d’abord la mère, puis une des deux filles puis la mère.

**Commandes :**

0> e pere 0> e mere

1> e paul 1> e jeanne

2> e pere 2> e mere

Une image contenant table

Description générée automatiquementUne image contenant table

Description générée automatiquement

Test de capacité maximale du bateau

Nous avons veillé à ce qu’il n’y ait pas plus de deux personnes sur le bateau. Nous avons tenté d’ajouter, en plus du voleur et du policier, la fille Jeanne.

**Commandes :**

0> e voleur

1> e policier

2> e jeanne

Une image contenant table

Description générée automatiquement

Test d’embarquement ou de désembarquement d’une personne deux fois

Une image contenant table

Description générée automatiquementDans ce test, nous avons tenté d’embarquer deux fois le voleur sur le bateau, correspondant aux tours 0 et 1, et de le désembarquer à nouveau deux fois, aux tours 4 et 5. Comme le programme ne le trouve pas dans les conteneurs d’origine, c’est-à-dire respectivement sur la rive gauche et sur le bateau, il va simplement dire qu’il n’a pas trouvé le voleur.

**Commandes :**

0> e voleur

1> e voleur

2> e policier

3> m

4> e voleur

5> e voleur

Test du voleur qui ne peut être avec la famille sans la présence du policier

Il existe trois cas où le voleur ne peut se trouver seul avec la famille :

* Quand le policier embarque en premier, laissant le voleur seul avec la famille
* Quand le voleur embarque en premier ou second, puis un membre de la famille en second ou premier
* Quand le voleur débarque et qu’un membre de la famille est présent ou vice versa

**Commandes :**

0> e policier 5> m

1> e voleur 6> e jeanne

2> e policier 7> m

3> m 8> e jeanne

4> e voleur (e policier puis e jeanne fonctionnent)

Une image contenant table

Description générée automatiquementUne image contenant table

Description générée automatiquement

**Commande (suite)**

1>e voleur

Une image contenant table

Description générée automatiquement 2>e jeanne

Test des enfants et du voleur qui ne peuvent piloter le bateau

Selon la contrainte #2, les enfants et le voleur ne peuvent être les capitaines du bateau, ce qui est testé ci-dessous pour respectivement le voleur (à gauche) et les enfants (avec les garçons, à droite). Pour les filles, cette contrainte est aussi respectée ainsi que s’il n’y a personne sur le bateau.

**Commandes :**

0> e voleur 0> e paul

1> m 1> e pierre

2> m

Une image contenant table

Description générée automatiquementUne image contenant table

Description générée automatiquement

Test de la réinitialisation du programme

La réinitialisation doit permettre de remettre toutes les personnes sur la rive gauche et de rétablir la position du bateau sur cette même rive. Le tour n°9 représente la commande ‘r’

Une image contenant table

Description générée automatiquement

Une image contenant table

Description générée automatiquementTest d’inputs invalides

Enfin, nous avons testé des commandes farfelues contenant des erreurs de syntaxe ou d’orthographe.

**Commandes :**

0> e paire

1> e pierre et paul

2> erer pere

3>

4> e voleur

5> e policier

6>

7> m

8> d valeur

## Annexes

**Commandes pour faire passer tout le monde sur l’autre rive**

1>e voleur

2>e policier

3>m

d voleur

m

e jeanne

m

d policier

d jeanne

e voleur

e policier

m

d policier

d voleur

e julie

e mere

m

d julie

m

e pere

m

d mere

m

d pere

e voleur

e policier

m

d policier

d voleur

e mere

m

e pere

m

d mere

m

e paul

m

d pere

d paul

e voleur

e policier

m d policier

d voleur

e pierre

e policier

m

d pierre

m

e voleur

m

d policier

52>d voleur

53>q