Laboratoire 3: Rivière

Durée du laboratoire: 3 séances. A rendre pour le jeudi 12 mai.

1. Introduction

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.
- Les filles ne peuvent rester seules avec leur père si la mère n'est pas présente.

Le but du laboratoire est 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.

2. Exemple d'exécution

Remarque: dans l'exemple ci-dessous la ligne de = symbolise la rivière.

1. Situation initiale:

```
p      : afficher
e <nom>: embarquer <nom>
d <nom>: debarquer <nom>
m      : deplacer bateau
r      : reinitialiser
q      : quitter
h      : menu

------------------------------------------------------------
Gauche: pere mere paul pierre julie jeanne policier voleur
------------------------------------------------------------
Bateau: < >
============================================================

------------------------------------------------------------
Droite:
------------------------------------------------------------
```

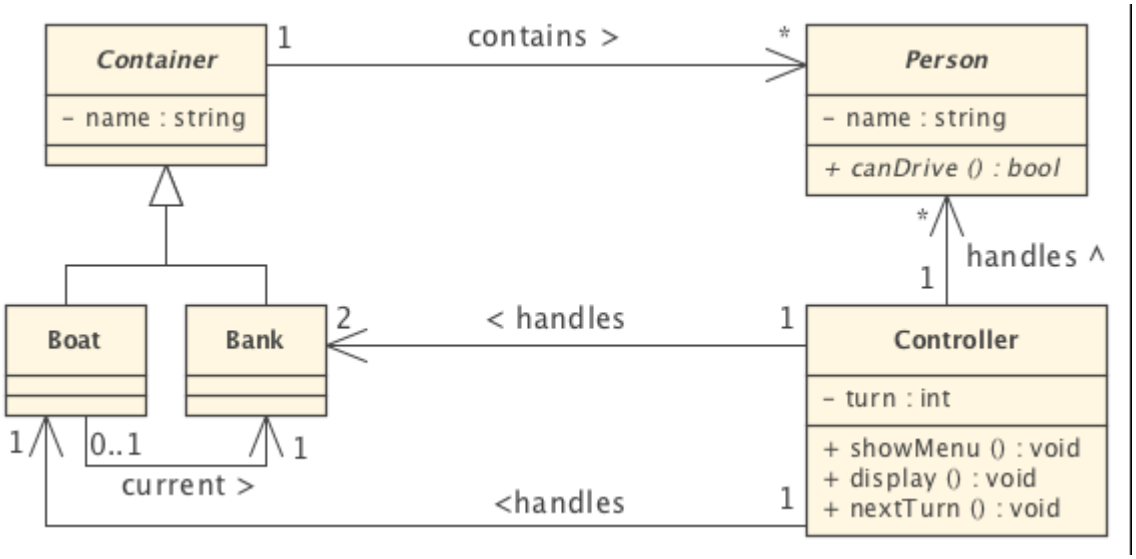
2. Etats après les commandes « e voleur », « e pere », « e policier », « m » et « d policier »:

```
0> e voleur
------------------------------------------------------------
Gauche: pere mere paul pierre julie jeanne policier
------------------------------------------------------------
Bateau: < voleur >
============================================================

------------------------------------------------------------
Droite:
------------------------------------------------------------
```

```
1> e pere
### garcon avec sa mere sans son pere

2> e policier
------------------------------------------------------------
Gauche: mere paul pierre julie jeanne pere
------------------------------------------------------------
Bateau: < voleur policier >
============================================================

------------------------------------------------------------
Droite:
------------------------------------------------------------

3> m
------------------------------------------------------------
Gauche: mere paul pierre julie jeanne pere
------------------------------------------------------------

============================================================
Bateau: < voleur policier >
------------------------------------------------------------
Droite:
------------------------------------------------------------

4> d policier
------------------------------------------------------------
Gauche: mere paul pierre julie jeanne pere
------------------------------------------------------------

============================================================
Bateau: < voleur >
------------------------------------------------------------
Droite: policier
------------------------------------------------------------
```

3. Mise en oeuvre

Modéliser le problème en utilisant une approche POO, par exemple en s'inspirant du diagramme de classes incomplet ci-dessous (en y adjoignant les sous-classes, attributs et méthodes nécessaires). En particulier, il est important de bien factoriser le code produit et de ne pas compromettre l'encapsulation des données.



4. Annexe: STL

Afin de ne pas réinventer la roue il est possible d'utiliser les classes de la STL (Standard Template Library).

1. Classe `string` (cf. http://www.cppreference.com/cppstring.html)
   Définie dans l'en-tête `<string>`. Fonctionnalités analogues à celles définies dans le laboratoire précédent.
2. Classe `list` (cf. http://www.cppreference.com/cpplist.html)
   Définie dans l'en-tête `<list>`.
   Cette classe est générique et permet ainsi d'instancier des listes spécifiques d'un type donné. Par exemple,
   - définition d'une liste d'entiers: `list<int> listeEntiers;`
   - définition d'une liste de pointeurs sur des voitures: `list<Voiture*> listeVoitures;`

   Méthodes usuelles
   - `push_back()` et `push_front()`: insère un élément en fin ou en début de liste.
   - `pop_back()` et `pock_front()`: supprimer le dernier ou le premier élément de la liste.
   - `remove(valeur)`: supprimer les éléments `valeur` de la liste.
   - `size()`: nombre d'éléments dans la liste.
   - `begin()` et `end()`: rend un itérateur référençant le début ou la fin de la liste.
3. Itérateurs (cf. http://www.cppreference.com/iterators.html)
   A chacune des collections définie dans la STL est associé un type d'itérateur permettant de la parcourir. Par exemple,
   - itérateur sur la liste d'entiers: `list<int>::iterator it = listeEntiers.begin();`

   Opérateurs usuels:
   - les opérateurs `++` et `--` permettent de passer à l'élément suivant ou précédent dans la liste.
   - l'élément courant dans la collection est obtenu en déréférençant l'itérateur (`*it`).
   - la position d'un itérateur dans la collection peut être comparée avec celle d'un autre par les opérateurs `==` et `!=` (p.ex. `it == listeEntiers.end()`).
4. Fonction `find`
   La fonction `find(debut, fin, valeur)` permet de rechercher un élément dans une collection où, `debut` et `fin` sont des itérateurs et `valeur` est l'élément recherché. Elle rend un itérateur positionné sur le premier élément correspondant dans la collection ou sur la fin de la collection (définie par la méthode `end()` de la collection).

Exemple (`stl.cpp`):

```
list<string> l; // definition d'une liste de string

// insertions
l.push_back("un");
l.push_back("deux");
l.push_back("trois");

// parcours de la liste au moyen d'un itérateur
for (list<string>::iterator it = l.begin(); it != l.end(); it++)
  cout << *it << " ";
cout << endl;

// suppression d'un élément
l.remove("deux");

// recherche d'un élément
if (find(l.begin(), l.end(), "deux") == l.end())
  cout << "La liste ne contient plus d'élément \"deux\"" << endl;
```