C++ : héritage et polymorphisme

TP C++

Héritage et polymorphisme

Eric Ramat
eric.ramat@univ-littoral.fr

2 novembre 2023

Durée : 6 heures

---

1 Objectifs

Le troisième TP a pour objectif d'utiliser l'héritage et le polymorphisme pour simplifier le code, généraliser ou spécialiser certains concepts.

2 Travail demandé

2.1 Première partie : simulation

Pour le moment, le code que nous avons écrit permet de définir la structure des routes et de leurs connexions via des noeuds. On peut aussi injecter un véhicule dans la structure en spécifiant l'id d'un noeud.
L'étape suivante est de faire avancer les véhicules. Lorsqu'un véhicule rentre dans un noeud, il mets un certain pour en sortir (temps de traversée). Lorsqu'il en sort, il rentre dans un tronçon `Link` et il est placé dans la première voie `Way` qui dispose d'une case `Cell` vide. Puis le véhicule va avancer de case en case si la case suivante est libre.

Etape 1. Développer la méthode `void run(unsigned int t)` dans la classe `Node`. Si un véhicule est présent et que c'est le moment que le véhicule doit sortir alors on vérifie que le tronçon suivant est libre et si c'est le cas alors on y place le véhicule. Afin de simplifier, s'il y a plusieurs tronçons de sortie, le véhicule est placé dans le premier tronçon .
Pour connaître si c'est le moment, il faut ajouter un attribut dans la classe `Node` qui stocke la date d'arrivée du véhicule dans le noeud. Modifier la méthode `push_vehicle` afin qu'elle prenne un paramètre supplémentaire : la date d'arrivée du véhicule (type : unsigned int). La date de sortie sera alors égale à la date d'arrivée + la durée de traversée.

Etape 2. Développer la méthode `void run(unsigned int t)` dans la classe `Way`. Cette méthode parcourt les cellules de la voie et déplacer les véhicules. Attention, à bien parcourir les cellules dans le bon sens. Un véhicule peut passer à la case suivante si cette dernière est libre.

Etape 3. Développer la méthode `void run(unsigned int t)` dans la classe `Link`. Cette méthode se divise en deux logiques :

— vérifier qu'un véhicule est présent à la fin d'une voie de circulation et dans ce cas, il faut déplacer le véhicule dans le noeud de sortie (s'il est libre)
— appeler la méthode run de chaque voie de circulation pour que les véhicules avancent

Etape 4. Développer la méthode `void run(unsigned int t)` dans la classe `Map`. C'est la plus simple : elle fait appel à la méthode `run` de chaque tronçon puis à la méthode `run` de chaque noeud.

Etape 5. Ajouter dans la fonction `main` après le code de création des tronçons et des noeuds et l'ajout d'un véhicule dans le noeud $N1$, le code suivant :

```
unsigned int t = 0;

while (t < 1000) {
   map.run(t);
   ++t;
}
```

Etape 6. Afin de vérifier le bon fonctionnement, ajouter les affichages suivants :
— quand un véhicule vient d'entrer dans un noeud ou un tronçon/voie
— quand un véhicule vient de sortir d'un noeud ou d'un tronçon

Vous devez obtenir :

```
0: V1 enters N1
2: V1 enters L8 on way 0
52: V1 enters N3
54: V1 enters L9 on way 0
84: V1 enters N10
86: V1 enters L10 on way 0
106: V1 enters N12
```

2.2 Deuxième partie : trajectoire

Actuellement, les véhicules empruntent toujours le premier tronçon en sortant d'un noeud. Pour éviter cette simplification, nous allons introduire la notion de trajectoire. Lors de la création d'un véhicule, on va ajouter une liste de chaînes de caractères qui contient les ID des tronçon et des noeuds empruntés.

Etape 1. Ajouter un attribut dans la classe `Vehicle` pour stocker la liste des ID et un attribut pour connaître l'indice de l'élément courant. Cet indice sera incrémenté à chaque fois que le véhicules changent de tronçon ou de noeud.

Etape 2. Modifier les méthodes `run` des classes `Node` et `Link` pour prendre en compte la trajectoire.

Etape 3. Pour tester, utiliser la nouvelle fonction `main`.

```
int main() {
   Map map;
   auto vehicle1 = std::make_unique<Vehicle>("V1", 10,
```

```
      std::vector<std::string>{"N1", "L8", "N3", "L9", "N10", "L10", "N12"});
   auto vehicle2 = std::make_unique<Vehicle>("V2", 10,
      std::vector<std::string>{"N1", "L8", "N3", "L11", "N13", "L12", "N7"});
   unsigned int t = 0;

   create_map(map);
   map.push_vehicle(t, vehicle1);
   while (t < 1000) {
      if (t == 10) {
         map.push_vehicle(t, vehicle2);
      }
      map.run(t);
      ++t;
   }
   return 0;
}
```

2.3 Troisième partie : jonction et rond-point

Les noeuds forment les connexions entre les tronçons. Un seul véhicule peut être dans le noeud. Dans cette troisième partie, nous allons introduire la notion de jonction et de rond-point. Une jonction est un noeud tel qu'on l'a défini pour le moment. Un rond-point est un noeud où plusieurs véhicules peuvent être présent en simultanément.

Etape 1. Renommer l'actuelle classe `Node` en `Junction`. Adapter le code en conséquence et d'ajouter les nouveaux fichiers dans CMakeLists.txt du répertoire src.

Etape 2. Définir une classe abstraite `Node` avec les attributs de `Junction` que l'on peut factoriser avec la future classe `RoundAbout` : id, duration, out_links et in_links.
Concernant les méthodes, il est possible de factoriser :

- les méthodes pour connecter le noeud avec les tronçons amonts et avals
- certains accesseurs

Afin de généraliser, les trois méthodes suivantes doivent être abstraites :

- `push_vehicle` qui gère l'arrivée d'un véhicle dans le noeud
- `is_free` pour indiquer que le noeud n'est pas plein
- `run` qui gère le temps de traversée du noeud

Etape 3. Après avoir défini toutes les méthodes de `Node`, faire hériter la classe `Junction` de la classe `Node` et procéder aux adaptations du reste du code (principalement la classe `Map`). La méthode `add_node` devient `add_junction`.

Etape 4. Créer une nouvelle classe `RoundAbout` pour les ronds points. On utlisera la classe queue de la STL pour représenter la liste des véhicules en train de faire le tour du rond-point. Un rond point possède une capacité limite.

Implémenter les trois méthodes abstraites (`push_vehicle`, `is_free` et `run`). Attention la méthode `run` est plus complexe que pour une jonction. Plusieurs véhicules peuvent sortir en même temps.

Etape 5. Afin de tester, on va transformer le noeud $N10$ en rond-point :

```
auto node_10 = std::make_shared<RoundAbout>("N10", 5, 2, 1, 5);
```

et ajouter de nouveaux véhicules dans le noeud $N1$ et le noeud $N8$ (attention, il faut les injecter à des dates différentes) :

```
int main() {
   Map map;
   auto vehicle1 = std::make_unique<Vehicle>("V1", 10,
      std::vector<std::string>{"N1", "L8", "N3", "L9", "N10", "L10", "N12"});
   auto vehicle2 = std::make_unique<Vehicle>("V2", 10,
      std::vector<std::string>{"N1", "L8", "N3", "L11", "N13", "L12", "N7"});
   auto vehicle3 = std::make_unique<Vehicle>("V3", 10,
      std::vector<std::string>{"N1", "L8", "N3", "L9", "N10", "L10", "N12"});
   auto vehicle4 = std::make_unique<Vehicle>("V4", 10,
      std::vector<std::string>{"N8", "L4", "N6", "L5", "N10", "L10", "N12"});
   auto vehicle5 = std::make_unique<Vehicle>("V5", 10,
      std::vector<std::string>{"N8", "L4", "N6", "L5", "N10", "L10", "N12"});

   create_map(map);

   unsigned int t = 0;

   map.push_vehicle(t, vehicle1);
   map.push_vehicle(t, vehicle4);
   while (t < 1000) {
      if (t == 3) {
         map.push_vehicle(t, vehicle3);
         map.push_vehicle(t, vehicle5);
      }
      if (t == 10) {
         map.push_vehicle(t, vehicle2);
      }
      map.run(t);
      ++t;
   }
   return 0;
}
```

Vous devez obtenir une sortie console de ce type :

```
   0: V1 enters N1
   0: V4 enters N8
   2: V1 enters L8 on way 0
   2: V4 enters L4 on way 0
   3: V3 enters N1
   3: V5 enters N8
   5: V3 enters L8 on way 0
   5: V5 enters L4 on way 0
   10: V2 enters N1
```

```
12: V2 enters L8 on way 0
52: V1 enters N3
54: V1 enters L9 on way 0
55: V3 enters N3
57: V3 enters L9 on way 0
62: V2 enters N3
64: V2 enters L11 on way 0
84: V1 enters N10 [1/5]
87: V3 enters N10 [2/5]
89: V1 enters L10 on way 0
92: V4 enters N6
92: V3 enters L10 on way 0
94: V4 enters L5 on way 0
95: V5 enters N6
97: V5 enters L5 on way 0
104: V2 enters N4
106: V2 enters L12 on way 0
109: V1 enters N12
124: V4 enters N10 [1/5]
127: V5 enters N10 [2/5]
129: V4 enters L10 on way 0
132: V5 enters L10 on way 0
176: V2 enters N7
```