Compte rendu : TP C++ - Héritage et Polymorphisme

GUIRAUDOU Bastien et TURPIN Laurent

6 décembre 2016

1 Introduction

Dans ce TP, nous devions réaliser un logiciel pour gérer des trajets entre des villes avec différents moyens de transport. Il devait permettre à l'utilisateur d'ajouter de nouveau trajets, simples ou composés de plusieurs étapes, à un catalogue, d'afficher ce même catalogue ou encore de rechercher un trajets parmis ceux existants. Nous avons choisis comme outils de réalisation la liste chainée, méthode qui nous semblait appropriée car nous l'avions déjà vu et utilisée lors d'un précédent TP d'algorithmique.

Nous avons essayé, tout au long de notre travail, de faire en sorte que notre programme soit facilement modulable par la suite, dans le cas ou le cahier des charges viendrait à évoluer.

2 Descriptions Précises des Classes

Pour la réalisation de notre TP, nous avons créé cinq classes : Une servant à gérer notre catalogue de trajet, trois pour la création et la gestion des dit trajets, fonctionnant sous le principe de l'héritage, et enfin une classe pour définir les éléments de notre liste chainée, puisque nous avons décidé que notre catalogue fonctionnerait ainsi. Un fichier `LTIF.cpp` utilise l'ensemble de ces classes et contient le `main` du programme, c'est lui qui gère le catalogue.

2.1 LTIF - Logiciel de Transport IF

Ce fichier contient le `main`, ainsi que la gestion de l'interface avec l'utilisateur. L'interaction entre l'utilisateur et le logiciel se fait directement dans l'invite de commande.

`MAX_LENGTH`, un nombre entier constant sert à la création de nos chaines de caractères qui recevront les entrées de l'utilisateur.

Une fois exécuté, on peut ajouter des trajets au catalogue ou appeler la recherche et l'affichage de ce catalogue. Un rapide mode d'emploi est affiché au lancement du programme, pour que l'utilisateur ne soit pas perdu avec l'interface.

2.2 Catalogue

Cette classe permet la gestion de notre catalogue de trajet. Elle permet d'ajouter des trajets (à la fin du catalogue), d'afficher les trajets dans leur ordre d'ajout, ou bien de rechercher un trajet. Les trajets sont rangés en liste chainée, et la classe contient un pointeur vers le premier élément et un autre vers le dernier de la liste.

Attributs :

`first` un pointeur d'élément de liste chainée, pointant vers le premier élément du catalogue, ou vers `NULL` si le catalogue est vide.

`last` un pointeur d'élément de liste chainée, pointant vers le dernier élément du catalogue, ou vers `NULL` si le catalogue est vide.

2.3 Element

Cette classe sert a définir les éléments qui composent notre liste chaînée. Un objet de type `Element`contient deux pointeurs, un pointant vers un trajet, le deuxième vers l'élément suivant. Le premier pointe vers un objet de type `Trajet`, qui peut être indifféremment un trajet composé ou un trajet simple.

Attributs :

`next` un pointeur vers un autre objet de type `Element`, le suivant dans la liste chaînée.

`traj` un pointeur vers un objet de type Trajet.

Ces éléments servent également à créer des trajets composés, en fonctionnant de la même manière, à ceci près que lors de la création de trajets composés nous n'utilisons que des trajets simples. En revanche, en cas de volonté d'évolution, nous pourrons mettre des trajets composés dans d'autres trajets composés sans que cela pose problème.

2.4 Trajet

Cette classe est la classe mère de notre héritage des classes représentant des trajets (cf : Figure 1). C'est une classe virtuelle pure, par conséquent il est impossible de créer un objet de type `Trajet`, mais cette classe est indispensable à la création d'une chaîne de trajets polymorphiques.

Cette classe n'a pas d'attributs propres, et aucune de ses méthodes n'est implémentée, elles le sont dans les classes filles. Nous avons une méthode permettant d'afficher un trajet donné, des getteurs pour récupérer certain de leurs attributs.

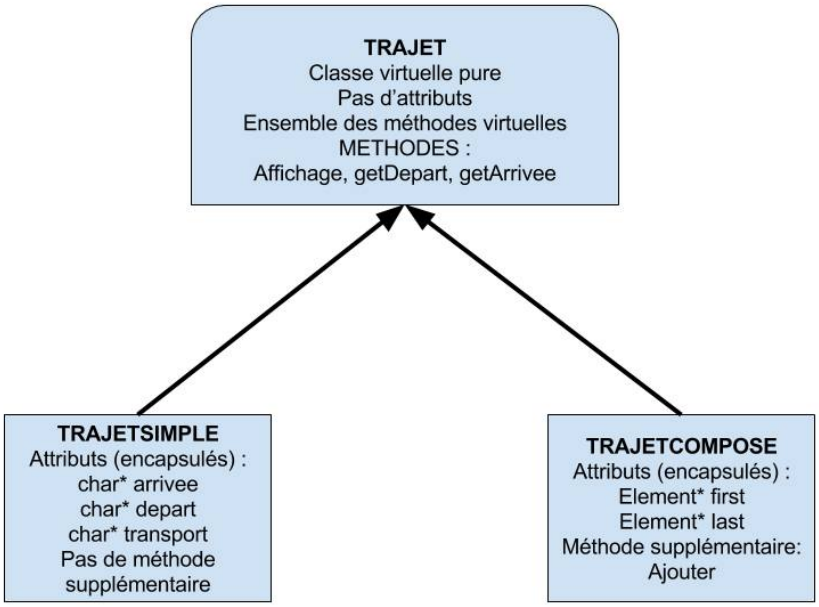

FIGURE 1 – Schéma d'héritage du projet

2.5 TrajetSimple

Cette classe hérite de la classe `Trajet`. Un objet de type `TrajetSimple` comporte une ville de départ, une ville d'arrivée et un moyen de transport. Ce sont des objets de type `TrajetSimple` qui sont utilisés pour créer des trajets composés.

Attributs :

`arrivee` une chaîne de caractères contenant la ville d'arrivée du trajet.

`depart` une chaîne de caractères contenant la ville de départ du trajet.

`transport` une chaîne de caractères contenant le moyen de transport.

Pour pouvoir récupérer ces attributs, qui sont encapsulés, nous avons créé des getteurs, qui nous permettent de réutiliser ces attributs en lecture seule sans risques qu'ils soient modifiés depuis l'extérieur de la classe.

2.6 TrajetCompose

Cette classe descend également de la classe `Trajet`. Un trajet composé fonctionne de la même manière que notre catalogue, avec une liste chaînée de trajets, qui sont ici tous des trajets simples. L'ajout se fait en bout de chaine.

Attributs :

`first` un pointeur d'Element, qui pointe vers le premier trajet simple.

`last` un pointeur d'Element, qui pointe vers le dernier trajet simple.

Tout comme pour les trajets simples, nous avons des getteurs pour récupérer le départ du premier trajet simple et l'arrivée du dernier, afin de pouvoir effectuer notre recherche de trajets correctement.

3 Structure de données

Notre structure de données est une liste chaînée. Dans notre objet `Catalogue`, nous avons deux pointeurs. Le premier, nommé `first`, pointe vers le premier élément de notre liste. Le deuxième, nommé `last`, pointe vers le dernier élément. Notre liste est basée sur le principe du First in First out, c'est à dire qu'à l'affichage, le premier élément inséré est le premier affiché, et que l'insertion se fait en bout de chaine. Nous n'avons pas de méthode pour supprimer des éléments.

Chaque élément de notre liste chainée est composé de deux pointeurs. Celui nommé `next` pointe vers l'élément suivant, et celui nommé `traj` pointe vers un objet de type trajet. Le pointeur `next` du dernier élément pointe vers `NULL`.

Grâce au système d'héritage, le trajet pointé par un élément peut être indépendamment un trajet simple ou un trajet composé.

Nos trajets composés sont formés sur le même principe que notre catalogue, et nous avons donc une nouvelle liste chaînée, avec des pointeurs `first` et `last`, et une suite d'éléments. La seule différence est que dans notre code actuel, l'utilisateur ne peux que former un trajet composé a partir de trajets simples.

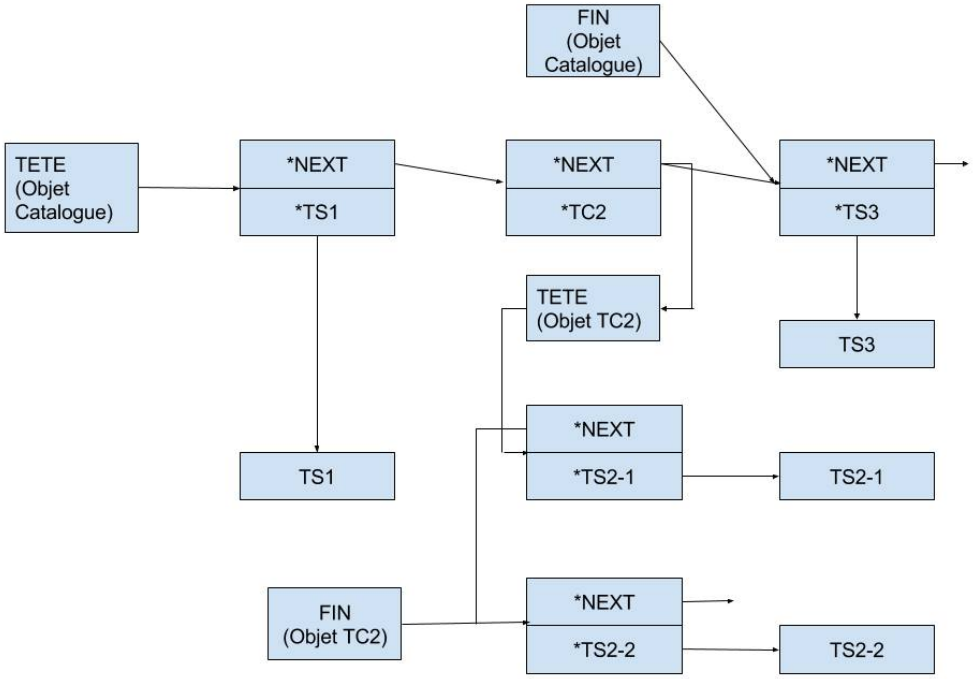

FIGURE 2 – Schéma de l'état de la mémoire sur un exemple

4 Liste des classes et méthodes

4.1 Trajet

```
class Trajet;
```

La classe `Trajet` est une classe abstraite décrivant le comportement général d'un trajet. Chaque méthode de la classe est `const` car quand un trajet est créé, il ne doit pas être modifier.

```
virtual void Afficher() const = 0;
```

La méthode `Afficher` permet d'afficher dans la sortie standard la ville de départ du trajet, la ville d'arrivée et le moyen de transport.

```
virtual const char* getDepart() const = 0;
```

La méthode `getDepart` renvoie la chaine de caractères correspondant à la ville de départ du trajet

```
virtual const char* getArrivee() const = 0;
```

La méthode `getArrivee` fait la même chose que `getDepart` mais pour la ville d'arrivée

Les constructeurs et le destructeur de cette classe ne seront pas précisés car c'est une classe abstraite donc aucun objet de ce type n'est créé.

4.2 TrajetSimple

```
class TrajetSimple : public Trajet;
```

Cette classe hérite de `Trajet`, elle définit un trajet qui n'a aucune escale, juste un ville de départ, une ville d'arrivée et un moyen de transport.

```
TrajetSimple (const char* dep, const char* arr, const char* trans);
```

Le constructeur prend en paramètre des chaines constantes de caractères pour définir chaque attribut de la classe.

```
virtual ~TrajetSimple();
```

Le destructeur est assez simple, il libère la mémoire pointée par chacun des pointeurs.

```
void Afficher() const;
```

Affiche sur la sortie standard toutes les caractéristiques de la classe de la manière suivante :
"De [Ville de départ] à [Ville d'arrivée] en [Moyen de Transport]
Il n'y a pas de retour à la ligne à la fin, le formatage de l'affichage sera mieux géré par le catalogue.

```
const char* getDepart() const;
```

Cette méthode renvoie la chaine de caractères `depart` correspondant à la ville de départ du trajet simple.

```
const char* getArrivee() const;
```

Cette méthode renvoie la chaine de caractères `arrivee` correspondant à la ville d'arrivée du trajet simple.

4.3 Element

```
classe Element;
```

Cette classe défini les objets représentant les maillons d'une liste chaînée composée d'objets de type `Trajet`.

```
Element (const Trajet* tr);
```

Le constructeur de la classe prend en paramètre un pointeur vers un `Trajet` et l'assigne au pointeur `traj`. Comme le type de ce pointeur est constant, il suffit de faire une affectation classique. Cependant, il faut faire attention à ne pas modifier le pointeur utilisé pour la construction après coup. Le pointeur vers le maillon suivant et initialisé à `NULL` ce qui nous permet de connaître la fin de la liste.

```
virtual ~Element();
```

Le Destructeur ne "détruit" que son trajet associé, c'est la classe qui gérera la liste qui fera le reste.

```
const Trajet* getTraj() const;
```

Cette méthode renvoie le pointeur `traj` du maillon. Pour ne pas compromettre l'intégrité du maillon et ne pas modifier son information utile, la méthode est `const` et surtout il faut directement utiliser une méthode associée au trajet.

```
void setNext(Element* n);
```

Cette méthode permet de modifier le pointeur `next` de la classe pour notamment ajouter `n` comme prochain maillon de la chaine.

```
Element* getNext() const;
```

Cette méthode renvoie le pointeur `next` servant à parcourir la liste chaînée.

4.4 Trajet Compose

```
class TrajetCompose : public Trajet;
```

Cette classe hérite également de `Trajet`, elle définit un trajet pouvant avoir de multiples escales. Elle est implémentée comme une liste chainée utilisant les objets de type `Element` comme maillons de la liste. Elle contient deux pointeurs, un sur le début de la liste et un sur la fin . Pour ce TP, ce sera une liste chaînées avec exclusivement des objets de type `TrajetSimple`.

```
TrajetCompse();
```

Le constructeur initialise les deux pointeurs à `NULL`, c'est donc l'état d'une liste vide.

```
virtual ~TrajetCompose();
```

Le destructeur doit donc se charger de libérer la mémoire pour chaque maillon grâce à une simple boucle.

```
void Afficher() const;
```

Cette méthode utilise la méthode `Afficher` de chaque maillon de la liste séparé par un tiret "-".

```
const char* getDepat() const;
```

Cette méthode utilise la méthode `getDepart` du premier maillon de la liste.

```
const char* getArriverr() const;
```

Cette méthode utilise la méthode `getArrivee` du dernier maillon de la liste.

```
void Ajouter(Trajet* tr);
```

Cette méthode crée un nouveau maillon avec `tr` et le place en fin de la liste. Il faut absolument que la ville de départ d'un maillon corresponde à la ville d'arrivée du maillon précédent pour que le trajet soit licite.

4.5 Catalogue

```
classe Catalogue;
```

Cette classe représente le catalogue complet des trajets. Elle est implémentée comme une liste chaînée dont les maillons sont des objets de type `Trajet`. Elle contient donc deux pointeurs, une pour le début et un pour la fin de la liste.

```
Catalogue();
```

Le constructeur initialise les deux pointeurs à `NULL`, c'est donc l'état d'un catalogue vide.

```
virtual ~Catalogue();
```

Le destructeur doit se charger de libérer la mémoire de chaque maillons.

```
void Afficher() const;
```

Cette méthode affiche à la sortie standard les trajets du catalogue ligne par ligne en utilisant la méthodes `Afficher` de chaque maillon.

```
void Ajouter(const Trajet* tr);
```

Cette méthode crée un nouveau maillon avec le paramètre `tt` et le place à la fin de la liste.

```
void Rechercher(const char* dep, const char* arr) const;
```

Cette méthode affiche ligne par ligne tous les trajets qui ont comme ville de départ le paramètre `dep` et comme ville d'arrivée le paramètre `arr`. Sachant qu'un trajet doit toujours être pris au complet, on ne peut pas interrompre un trajet composé.

5 Problèmes rencontrés et axes d'amélioration

5.1 Problèmes

Durant ce TP, nous avons eu des problèmes sur la façon de définir `const` certains attributs. En effet le fait de l'avoir fait un peu tard dans la réalisation a provoqué une réaction en chaîne où nous devions revoir les méthodes impactées par le changement. Cependant la définition en `const` de certains attributs (ceux des trajets simples) leur donnait un comportement peu désirable : on ne pouvait les définir qu'avec une affectation classique avec un autre pointeur. Donc si on changeait ce dernier pointeur, l'attribut changeait également.

De plus, réaliser la recherche avancée a été bien plus dur que prévu. L'algorithme fonctionne bel et bien grâce à une récursivité mais génère des fuites de mémoire venant de la créations de pointeurs qui pointent sur des objets déjà présents et qui ne doivent surtout pas être détruits. On de peut donc pas appliquer de `delete` sur ces nouveaux pointeurs au risque de corrompre les données du catalogue.

5.2 Améliorations

Pour améliorer ce programme, on peut déjà étendre le type des éléments contenus dans un trajet composé à des trajets de tout types (simple et composé). En réalité, l'opération est déjà faisable avec le code actuel.

Ensuite, on pourrait très bien n'avoir que des objets de types `TrajetCompose` dans le catalogue, avec des trajets composés ne comprenant qu'un seul trajet simple par exemple. Cela réduirait certaines redondances dans le code du fichier `LTIF.cpp`

On pourrait ajouter des méthodes pour retirer des trajet dans le catalogue ou filtrer les recherches suivant les moyens de transport, ou bien en trajet direct, ou bien en imposant des escale. On pourrait aussi assurer l'unicité des trajets dans le catalogue, le fait de ne pas faire de boucle avec les trajets composés.

Enfin une interface graphique sera toujours bien appréciée.

6 Annexes : code source

6.1 Trajet

Trajet.h

```
1  /*************************************************************************
2                            Trajet  -  description
3                              -------------------
4      debut                : $DATE$
5      copyright            : (C) $YEAR$ par $AUTHOR$
6      e-mail               : $EMAIL$
7  *************************************************************************/
8
9  //---------- Interface de la classe <Trajet> (fichier Trajet.h) ----------
10 #if ! defined ( TRAJET_H )
11 #define TRAJET_H
12
13 //------------------------------------------------- Interfaces utilisees
14
15 //------------------------------------------------------------ Constantes
16
17 //------------------------------------------------------------------ Types
18
19 //-------------------------------------------------------------------------
20 // Role de la classe <Trajet>
21 // Classe abstraite pure pour ses descendants : TrajetSimple et
22 // TrajetCompose
23 //-------------------------------------------------------------------------
24
25 class Trajet
26 {
27 //------------------------------------------------------------------ PUBLIC
28
29 public:
30 //----------------------------------------------------- Methodes publiques
31     virtual void Afficher () const = 0;
32     // Mode d'emploi : Affiche l'objet Trajet.
33     //
34     // Contrat : L'affichage tiens compte de la nature de l'objet, compose
35     // ou simple.
36
37     virtual const char * getDepart () const = 0;
38     // Mode d'emploi : renvoie la ville de depart du trajet.
39
40     virtual const char* getArrivee () const = 0;
41     // Mode d'emploi : renvoie la ville d'arrivee du trajet.
42
43 //-------------------------------------------- Constructeurs - destructeur
44
45     Trajet ( );
46     // Mode d'emploi : Ne jamais utiliser
47
48     virtual ~Trajet ( );
49
50 //------------------------------------------------------------------ PRIVE
```

```
51
52 protected:
53 //------------------------------------------------------ Methodes protegees
54
55 //------------------------------------------------------ Attributs proteges
56
57 };
58
59 //------------------------------ Autres definitions dependantes de <Trajet>
60
61 #endif // TRAJET_H
```

6.2 TrajetSimple

TrajetSimple.h

```
1  /*************************************************************************
2                            TrajetSimple  -  description
3                              -------------------
4      debut                : $DATE$
5      copyright            : (C) $YEAR$ par $AUTHOR$
6      e-mail               : $EMAIL$
7  *************************************************************************/
8
9  //---- Interface de la classe <TrajetSimple> (fichier TrajetSimple.h) ----
10 #if ! defined ( TRAJETSIMPLE_H )
11 #define TRAJETSIMPLE_H
12
13 //------------------------------------------------ Interfaces utilisees
14 #include "Trajet.h"
15
16 //------------------------------------------------------------ Constantes
17
18 //------------------------------------------------------------------ Types
19
20 //-------------------------------------------------------------------------
21 // Role de la classe <TrajetSimple>
22 // Trajet qui est direct. Il ne contient qu'une ville de depart, d'arrivee
23 // et un moyen de transport
24 //-------------------------------------------------------------------------
25
26 class TrajetSimple : public Trajet
27 {
28 //------------------------------------------------------------------ PUBLIC
29
30 public:
31 //------------------------------------------------------ Methodes publiques
32
33     void Afficher () const;
34     // Mode d'emploi : affiche le depart, l'arrivee et le moyen de
35     // transport du trajet.
36
37     const char* getDepart () const;
38     // Mode d'emploi : renvoie la ville de depart du trajet.
39
```

```
40      const char* getArrivee () const;
41      // Mode d'emploi : renvoie la ville d'arrivee du trajet.
42
43  //------------------------------------------ Constructeurs - destructeur
44
45      TrajetSimple (const char* dep, const char* arr, const char* trans);
46      // Mode d'emploi : Initialise les attributs avec chaque parametre
47
48      virtual ~TrajetSimple ( );
49
50  //------------------------------------------------------------------ PRIVE
51
52  protected:
53  //---------------------------------------------------- Methodes protegees
54
55  //---------------------------------------------------- Attributs proteges
56      char* arrivee;
57      char* depart;
58      char* transport;
59  };
60
61  //----------------------- Autres definitions dependantes de <TrajetSimple>
62
63  #endif // TRAJETSIMPLE_H
```

TrajetSimple.cpp

```
1   /*************************************************************************
2                              TrajetSimple  -  description
3                                -------------------
4       debut                : $DATE$
5       copyright            : (C) $YEAR$ par $AUTHOR$
6       e-mail               : $EMAIL$
7   *************************************************************************/
8
9   //-- Realisation de la classe <TrajetSimple> (fichier TrajetSimple.cpp) --
10
11  //------------------------------------------------------------------ INCLUDE
12
13  //-------------------------------------------------------- Include systeme
14  using namespace std;
15  #include <iostream>
16  #include <cstring>
17
18  //------------------------------------------------------ Include personnel
19  #include "TrajetSimple.h"
20
21  //------------------------------------------------------------------ Constantes
22
23  //------------------------------------------------------------------ PUBLIC
24
25  //----------------------------------------------------- Methodes publiques
26
27  const char* TrajetSimple::getDepart () const
28  {
29      return depart;
```

```
30 }
31
32 const char* TrajetSimple::getArrivee () const
33 {
34     return arrivee;
35 }
36
37 void TrajetSimple::Afficher () const
38 {
39     cout <<"De "<<depart<<" a "<<arrivee<<" en "<<transport;
40 }
41
42 //------------------------------------------ Constructeurs - destructeur
43
44 TrajetSimple::TrajetSimple (const char* dep, const char* arr, const char* trans)
45 {
46 #ifdef MAP
47     cout << "Appel au constructeur de <TrajetSimple>" << endl;
48 #endif
49 arrivee = new char[strlen(arr)+1];
50 strcpy(arrivee, arr);
51
52 depart = new char[strlen(dep)+1];
53 strcpy(depart, dep);
54
55 transport = new char[strlen(trans)+1];
56 strcpy(transport, trans);
57
58 } //----- Fin de TrajetSimple
59
60 TrajetSimple::~TrajetSimple ( )
61 {
62 #ifdef MAP
63     cout << "Appel au destructeur de <TrajetSimple>" << endl;
64 #endif
65
66 delete [] depart;
67 delete [] arrivee;
68 delete [] transport;
69
70 } //----- Fin de ~TrajetSimple
71
72
73 //------------------------------------------------------------------ PRIVE
74
75 //------------------------------------------------------ Methodes protegees
```

6.3 TrajetCompose

TrajetCompose.h

```
1  /*************************************************************************
2                              TrajetCompose  -  description
3                                -------------------
4      debut                : $DATE$
5      copyright            : (C) $YEAR$ par $AUTHOR$
6      e-mail               : $EMAIL$
7  *************************************************************************/
8
9  //--- Interface de la classe <TrajetCompose> (fichier TrajetCompose.h) ---
10 #if ! defined ( TRAJETCOMPOSE_H )
11 #define TRAJETCOMPOSE_H
12
13 //---------------------------------------------------- Interfaces utilisees
14
15 #include "Trajet.h"
16 #include "Element.h"
17
18 //------------------------------------------------------------- Constantes
19
20 //------------------------------------------------------------------ Types
21
22 //-------------------------------------------------------------------------
23 // Role de la classe <TrajetCompose>
24 // Trajet qui contient plusieurs trajets simples. Implementee en liste
25 // chainee.
26 //-------------------------------------------------------------------------
27
28 class TrajetCompose : public Trajet
29 {
30 //----------------------------------------------------------------- PUBLIC
31
32 public:
33 //---------------------------------------------------- Methodes publiques
34
35     void Afficher () const;
36     // Mode d'emploi : Affiche l'objet TrajetCompose, en affichant chacun
37     // des trajets
38     // simples qui le composent.
39
40     const char* getDepart () const;
41     // Mode d'emploi : renvoie la ville de depart du trajet complet.
42
43     const char* getArrivee () const;
44     // Mode d'emploi : renvoie la ville d'arrivee du trajet complet.
45
46     void Ajouter (Trajet* tr);
47     // Ajoute un trajet simple a la fin de la liste.
48     //
49     // Contrat : L'enchainement des differents trajets simples est licite
50     // c'est a dire que l'arrivee de l'un correspond correspond au depart
51     // du suivant
52
```

```
53 //------------------------------------------ Constructeurs - destructeur
54
55     TrajetCompose ( );
56
57     virtual ~TrajetCompose ( );
58
59 //------------------------------------------------------------------ PRIVE
60
61 protected:
62 //----------------------------------------------------- Methodes protegees
63
64 //----------------------------------------------------- Attributs proteges
65 Element* first;
66 Element* last;
67
68 };
69
70 //----------------------- Autres definitions dependantes de <TrajetCompose>
71
72 #endif // TRAJETCOMPOSE_H
```

TrajetCompose.cpp

```
1  /*************************************************************************
2                             TrajetCompose  -  description
3                               -------------------
4      debut                : $DATE$
5      copyright            : (C) $YEAR$ par $AUTHOR$
6      e-mail               : $EMAIL$
7  *************************************************************************/
8
9  //- Realisation de la classe <TrajetCompose> (fichier TrajetCompose.cpp) -
10
11 //------------------------------------------------------------------ INCLUDE
12
13 //-------------------------------------------------------- Include systeme
14 using namespace std;
15 #include <iostream>
16 #include <cstring>
17
18 //------------------------------------------------------ Include personnel
19 #include "TrajetCompose.h"
20
21 //------------------------------------------------------------------ Constantes
22
23 //------------------------------------------------------------------ PUBLIC
24
25 //----------------------------------------------------- Methodes publiques
26
27 void TrajetCompose::Ajouter (Trajet* tr)
28 {
29     if (last == NULL)
30     {
31         first = new Element(tr);
32         last = first;
33     }
```

```
34      else
35      {
36          last->setNext(new Element(tr));
37          last = last->getNext();
38      }
39  }
40
41  void TrajetCompose::Afficher () const
42  {
43      Element* curseur = first;
44      while (curseur != last)
45      {
46          (curseur->getTraj())->Afficher();
47          cout << " - ";
48          curseur = curseur->getNext();
49      }
50      (curseur->getTraj())->Afficher();
51  }
52
53  const char* TrajetCompose::getDepart () const
54  {
55      return (first->getTraj())->getDepart();
56  }
57
58  const char* TrajetCompose::getArrivee () const
59  {
60      return (last->getTraj())->getArrivee();
61  }
62
63  //------------------------------------------ Constructeurs - destructeur
64
65  TrajetCompose::TrajetCompose ( )
66  {
67  #ifdef MAP
68      cout << "Appel au constructeur de <TrajetCompose>" << endl;
69  #endif
70
71  first = NULL;
72  last = NULL;
73
74  } //----- Fin de TrajetCompose
75
76
77  TrajetCompose::~TrajetCompose ( )
78  // Algorithme : il doit liberer la memoir de chaque maillon de la chaine.
79  {
80  #ifdef MAP
81      cout << "Appel au destructeur de <TrajetCompose>" << endl;
82  #endif
83  Element* temp;
84  while (first != last)
85  {
86      temp = first;
87      first = temp->getNext();
88      delete temp;
89  }
```

```
90 delete first;
91
92 } //----- Fin de ~TrajetCompose
93
94 //------------------------------------------------------------------ PRIVE
95
96 //---------------------------------------------------- Methodes protegees
```

6.4 Element

Element.h

```
1  /**************************************************************************
2                            Element  -  description
3                              -------------------
4      debut                : $DATE$
5      copyright            : (C) $YEAR$ par $AUTHOR$
6      e-mail               : $EMAIL$
7  **************************************************************************/
8
9  //---------- Interface de la classe <Element> (fichier Element.h) --------
10 #if ! defined ( ELEMENT_H )
11 #define ELEMENT_H
12
13 //------------------------------------------------- Interfaces utilisees
14
15 #include "Trajet.h"
16
17 //------------------------------------------------------------ Constantes
18
19 //------------------------------------------------------------------ Types
20
21 //------------------------------------------------------------------------
22 // Role de la classe <Element>
23 // Correspond a un maillon d'une liste chainee
24 //
25 //------------------------------------------------------------------------
26
27 class Element
28 {
29 //------------------------------------------------------------------ PUBLIC
30
31 public:
32 //----------------------------------------------------- Methodes publiques
33
34     const Trajet* getTraj() const;
35     //Recuperer trajet pointe.
36     //Contrat : Ne pas modifier le retour et utiliser directement une
37     //methode associee
38
39     void setNext(Element* n);
40     //Permet de definir le prochain maillon de la chaine
41
42     Element* getNext() const;
43     //Recuperer le pointeur vers le prochain element de la liste.
```

```
44
45 //------------------------------------------ Constructeurs - destructeur
46
47     Element (const Trajet* tr);
48     // Mode d'emploi :
49     //
50     // Contrat :
51     //
52
53     virtual ~Element ( );
54     // Mode d'emploi :
55     //
56     // Contrat :
57     //
58
59 //------------------------------------------------------------------ PRIVE
60
61 protected:
62 //------------------------------------------------------ Methodes protegees
63
64 //------------------------------------------------------ Attributs proteges
65
66 Element* next;
67 const Trajet* traj;
68
69 };
70
71 //---------------------------- Autres definitions dependantes de <Element>
72
73 #endif // ELEMENT_H
```

Element.cpp

```
1  /*************************************************************************
2                             Element  -  description
3                               -------------------
4     debut                : $DATE$
5     copyright            : (C) $YEAR$ par $AUTHOR$
6     e-mail               : $EMAIL$
7  *************************************************************************/
8
9  //---------- Realisation de la classe <Element> (fichier Element.cpp) ----
10
11 //------------------------------------------------------------------ INCLUDE
12
13 //-------------------------------------------------------- Include systeme
14 using namespace std;
15 #include <iostream>
16
17 //------------------------------------------------------ Include personnel
18 #include "Element.h"
19
20 //------------------------------------------------------------------ Constantes
21
22 //------------------------------------------------------------------ PUBLIC
23
```

```
24 //------------------------------------------------- Methodes publiques
25
26 const Trajet* Element::getTraj() const
27 {
28     return traj;
29 }
30
31 void Element::setNext(Element* n)
32 {
33     next = n;
34 }
35
36 Element* Element::getNext() const
37 {
38     return next;
39 }
40
41 //-------------------------------------------- Constructeurs - destructeur
42
43 Element::Element (const Trajet * tr)
44 // Algorithme :
45 //
46 {
47 #ifdef MAP
48     cout << "Appel au constructeur de <Element>" << endl;
49 #endif
50
51 traj = tr;
52 next = NULL;
53
54 } //----- Fin de Element
55
56
57 Element::~Element ( )
58 // Algorithme :
59 //
60 {
61 #ifdef MAP
62     cout << "Appel au destructeur de <Element>" << endl;
63 #endif
64
65 delete traj;
66
67 } //----- Fin de ~Element
68
69
70 //------------------------------------------------------------------ PRIVE
71
72 //------------------------------------------------- Methodes protegees
```

6.5 Catalogue

Catalogue.h

```
1  /*************************************************************************
2                             Catalogue  -  description
3                               -------------------
4      debut                : $DATE$
5      copyright            : (C) $YEAR$ par $AUTHOR$
6      e-mail               : $EMAIL$
7  *************************************************************************/
8
9  //---------- Interface de la classe <Catalogue> (fichier Catalogue.h) ----
10 #if ! defined ( CATALOGUE_H )
11 #define CATALOGUE_H
12
13 //--------------------------------------------------- Interfaces utilisees
14 #include "Trajet.h"
15 #include "Element.h"
16
17 //------------------------------------------------------------- Constantes
18
19 //------------------------------------------------------------------ Types
20
21 //-------------------------------------------------------------------------
22 // Role de la classe <Catalogue>
23 // Gere la collection de Trajet, elle peut l'afficher, ajouter un trajet
24 // ou rechercher un trajet. Implementee en liste chainee
25 //-------------------------------------------------------------------------
26
27 class Catalogue
28 {
29 //------------------------------------------------------------------ PUBLIC
30
31 public:
32 //----------------------------------------------------- Methodes publiques
33     // type Methode ( liste des parametres );
34     // Mode d'emploi :
35     //
36     // Contrat :
37     //
38
39     void Afficher () const;
40     //Mode d'emploi : Afficher les trajets numerotes;
41
42     void Ajouter (const Trajet* tr);
43     // mode d'emploi : Ajoute un trajet simple a la fin de la liste.
44
45     void Rechercher (const char* dep, const char* arr) const;
46     // Mode d'emploi : affiche les trajets disponibles entre les deux villes
47     // saisies.
48 //------------------------------------------------- Surcharge d'operateurs
49
50 //-------------------------------------------- Constructeurs - destructeur
51
52     Catalogue ( );
```

```
53      // Mode d'emploi :
54      //
55      // Contrat :
56      //
57
58      virtual ~Catalogue ( );
59      // Mode d'emploi :
60      //
61      // Contrat :
62      //
63
64  //------------------------------------------------------------------ PRIVE
65
66  protected:
67  //----------------------------------------------------- Methodes protegees
68
69  //----------------------------------------------------- Attributs proteges
70
71  Element* first;
72  Element* last;
73
74  };
75
76  //-------------------------- Autres definitions dependantes de <Catalogue>
77
78  #endif // CATALOGUE_H
```

Catalogue.cpp

```
1   /*************************************************************************
2                             Catalogue  -  description
3                               -------------------
4       debut                : $DATE$
5       copyright            : (C) $YEAR$ par $AUTHOR$
6       e-mail               : $EMAIL$
7   *************************************************************************/
8
9   //---------- Realisation de la classe <Catalogue> (fichier Catalogue.cpp)
10
11  //------------------------------------------------------------------ INCLUDE
12
13  //-------------------------------------------------------- Include systeme
14  using namespace std;
15  #include <iostream>
16  #include <cstring>
17
18  //------------------------------------------------------ Include personnel
19  #include "Catalogue.h"
20
21  //------------------------------------------------------------------ Constantes
22
23  //------------------------------------------------------------------ PUBLIC
24
25  //----------------------------------------------------- Methodes publiques
26
27  void Catalogue::Ajouter (const Trajet* tr)
```

```
28 {
29     if (last == NULL)
30     {
31         first = new Element(tr);
32         last = first;
33     }
34     else
35     {
36         last->setNext(new Element(tr));
37         last = last->getNext();
38     }
39 }
40
41 void Catalogue::Afficher () const
42 {
43     if(first == NULL) {
44         cout<<"Le catalogue est pour le moment vide.";
45         cout<<"Il ne tiens qu'a vous de le remplir."<<endl;
46     }
47     else {
48         cout << "** CATALOGUE DES TRAJETS DISPONIBLES **"<< endl;
49
50         Element* curseur = first;
51         int compt = 1;
52         while (curseur != last)
53         {
54             cout << compt << " - ";
55             compt ++;
56             (curseur->getTraj())->Afficher();
57             cout << endl;
58             curseur = curseur->getNext();
59         }
60         cout << compt << " - ";
61         (curseur->getTraj())->Afficher();
62         cout << endl;
63     }
64 }
65
66 void Catalogue::Rechercher (const char* dep, const char* arr) const
67 {
68
69     Element* curseur = first;
70     int compt = 0;
71     cout<<endl<<"RESULTATS : "<<endl;
72     while (curseur != NULL)
73     {
74         if (strcmp((curseur->getTraj())->getDepart(), dep) == 0 &&
75         strcmp((curseur->getTraj())->getArrivee(), arr) == 0)
76         {
77             (curseur->getTraj())->Afficher();
78             cout<< endl;
79             compt ++;
80         }
81         curseur = curseur->getNext();
82     }
83     if (compt == 0)
```

```
84      {
85          cout << "Il n'y a pas de trajets disponibles pour votre requete."<<endl;
86      }
87  }
88
89  //------------------------------------------------ Surcharge d'operateurs
90
91  //-------------------------------------------- Constructeurs - destructeur
92
93  Catalogue::Catalogue ( )
94  {
95  #ifdef MAP
96      cout << "Appel au constructeur de <Catalogue>" << endl;
97  #endif
98
99      first = NULL;
100     last = NULL;
101
102 } //----- Fin de Catalogue
103
104
105 Catalogue::~Catalogue ( )
106 {
107 #ifdef MAP
108     cout << "Appel au destructeur de <Catalogue>" << endl;
109 #endif
110
111     Element* temp;
112     while (first != last)
113     {
114         temp = first;
115         first = temp->getNext();
116         delete temp;
117     }
118     delete first;
119 } //----- Fin de ~Catalogue
120
121
122 //------------------------------------------------------------------ PRIVE
123
124 //------------------------------------------------------ Methodes protegees
```

6.6 Tests unitaires

Test.cpp

```
1  using namespace std;
2
3  #include <iostream>
4  #include <cstring>
5  #include "../Source/TrajetSimple.h"
6  #include "../Source/TrajetCompose.h"
7  #include "../Source/Element.h"
8  #include "../Source/Catalogue.h"
9
10 //Fichier des tests "unitaires" (a peu pres)
11
12 void TestTrajetSimple()
13 //Test la bonne creation d'un trajet simple et les methodes de la classe
14 {
15
16     TrajetSimple ts("Lyon", "Grenoble", "Chameau");
17     ts.Afficher();
18     cout<<endl<<ts.getDepart()<<endl<<ts.getArrivee()<<endl;
19 }
20
21 void TestTrajetCompse()
22 //Test la bonne creation d'un trajet compose et ses methodes
23 {
24     TrajetSimple* ts1 = new TrajetSimple("Lyon", "Grenoble", "Chameau");
25     TrajetSimple* ts2 = new TrajetSimple("Grenoble", "Paris", "Pingouin");
26
27     TrajetCompose tc;
28     tc.Ajouter(ts1);
29     tc.Afficher();
30     cout << endl;
31     cout <<tc.getDepart()<<endl<<tc.getArrivee()<<endl;
32     tc.Ajouter(ts2);
33     tc.Afficher();
34     cout << endl;
35     cout <<tc.getDepart()<<endl<<tc.getArrivee()<<endl;
36 }
37
38 void TestCatalogue()
39 // Test la creation d'un catalogue, sa creation, son affichage et la recharche
40 {
41     TrajetSimple* ts5 = new TrajetSimple("Lyon", "Grenoble", "Chameau");
42     TrajetSimple* ts6 = new TrajetSimple("Grenoble", "Paris", "Pingouin");
43     TrajetCompose* tc2 = new TrajetCompose();
44     tc2->Ajouter(ts5);
45     tc2->Ajouter(ts6);
46
47     TrajetSimple* ts3 = new TrajetSimple("Marseille", "Strasbourg", "Lama");
48     TrajetSimple* ts4 = new TrajetSimple("Strasbourg", "Pau", "Canari");
49     TrajetCompose* tc1 = new TrajetCompose();
50     tc1->Ajouter(ts3);
51     tc1->Ajouter(ts4);
52
```

```
53      TrajetSimple* ts1 = new TrajetSimple("Quimper", "Bordeaux", "Scooter");
54      TrajetSimple* ts2 = new TrajetSimple("Montpellier", "Veau-en-Velin",
55      "Platipus");
56      TrajetSimple* ts7 = new TrajetSimple("Quimper", "Bordeaux", "Baleine");
57
58      Catalogue cat;
59      cat.Ajouter(ts1);
60      cat.Ajouter(tc1);
61      cat.Ajouter(tc2);
62      cat.Ajouter(ts2);
63      cat.Ajouter(ts7);
64      cat.Afficher();
65      cat.Rechercher("Lyon", "Paris");
66      cat.Rechercher("Quimper", "Bordeaux");
67      cat.Rechercher("Marseille", "Strasbourg");
68      cat.Rechercher("Erule", "Tatouille");
69  }
70
71  int main() {
72      //TestTrajetSimple();
73      //TestTrajetComose();
74      //TestCatalogue();
75      return 0;
76  }
```

6.7 LTIF

LTIF.cpp

```
1   using namespace std;
2
3   #include <iostream>
4   #include <cstring>
5   #include "../Source/TrajetSimple.h"
6   #include "../Source/TrajetCompose.h"
7   #include "../Source/Element.h"
8   #include "../Source/Catalogue.h"
9
10  //Fichier corespondant a l'executable
11
12  //Pour avoir une taille pour les chaine de caratere servant au cin
13  const int MAX_LENGTH = 32;
14
15  void Creer(Catalogue & c) {
16      cout<<endl<<"**CREATION DE TRAJET**"<<endl;
17      cout<<"Trajet simple (tapez 1) ou compose ";
18      cout<<"(entrez votre nombre de trajets) : ";
19      int nb;
20      cin>>nb;
21
22      char dep[MAX_LENGTH];
23      char arr[MAX_LENGTH];
24      char trans[MAX_LENGTH];
25
26      if(nb == 1) {
```

```
27          cout<<"Ville de depart : ";
28          cin>>dep;
29          cout<<"Ville d'arrivee : ";
30          cin>>arr;
31          cout<<"Moyen de transport : ";
32          cin>>trans;
33          TrajetSimple* nouv = new TrajetSimple(dep, arr, trans);
34
35          cout<<"Voulvez-vous confirmer ? (0 pour non, 1 pour oui)"<<endl;
36          cin>>nb;
37          if(nb == 0) {
38              cout<<"Annule."<<endl<<endl;
39              delete nouv;
40          }
41          else {
42              c.Ajouter(nouv);
43              cout<<"Trajet cree."<<endl<<endl;
44          }
45      }
46      else if(nb > 1) {
47          TrajetCompose* nouv = new TrajetCompose();
48          cout<<"Ville de depart : ";
49          cin>>dep;
50          for(int i=1 ; i<nb ; i++) {
51              cout<<"Ville de l'escale "<<i<<" : ";
52              cin>>arr;
53              cout<<"Moyen de transport : ";
54              cin>>trans;
55              nouv->Ajouter(new TrajetSimple(dep, arr, trans));
56
57              strcpy(dep, arr);
58          }
59          cout<<"Ville d'arrivee : ";
60          cin>>arr;
61          cout<<"Moyen de transport : ";
62          cin>>trans;
63          nouv->Ajouter(new TrajetSimple(dep, arr, trans));
64
65          cout<<"Voulvez-vous confirmer ? (0 pour non, 1 pour oui)"<<endl;
66          cin>>nb;
67          if(nb == 0) {
68              cout<<"Annule."<<endl<<endl;
69              delete nouv;
70          }
71          else {
72              c.Ajouter(nouv);
73              cout<<"Trajet cree."<<endl<<endl;
74          }
75      }
76  }
77
78  void Rechercher(Catalogue & c) {
79      cout<<endl<<"**RECHERCHE**"<<endl;
80      char dep[MAX_LENGTH];
81      char arr[MAX_LENGTH];
82
```

```
83      cout<<"Ville de depart : ";
84      cin>>dep;
85      cout<<"Ville d'arrivee : ";
86      cin>>arr;
87      c.Rechercher(dep, arr);
88  }
89
90  int main() {
91
92      cout<<"Bonjour, cher utilisateur."<<endl;
93      cout<<"**MANUEL D'UTILISATION**"<<endl;
94      cout<<"Tapez \"Quitter\" pour quitter, \"Catalogue\" pour ";
95      cout<<"afficher le catalogue, \"Ajouter\" pour ajouter ";
96      cout<<"un trajet au catalogue et \"Rechercher trajet\" ";
97      cout<<"pour une recherche"<<endl<<endl;
98      char entree[MAX_LENGTH];
99      Catalogue cat;
100
101     cin>>entree;
102     while(strcmp("Quitter", entree) !=0) {
103
104         if(strcmp("Catalogue", entree) == 0) {
105             cout<<endl;
106             cat.Afficher();
107             cout<<endl;
108         }
109
110         if(strcmp("Ajouter", entree) == 0) {
111             Creer(cat);
112         }
113
114         if(strcmp("Rechercher", entree) ==0) {
115             Rechercher(cat);
116             cout<<endl;
117         }
118         cin>>entree;
119     }
120
121     cout<<"Au revoir, et merci."<<endl;
122     return 0;
123 }
```