INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

CERBA Guilhem
RAUDRANT Sophie

Compte Rendu TP2

Héritage et Polymorphisme
INSA de Lyon - POO1

Contents

1 Description Détaillée des Classes de l'Application **3**
- 1.1 Diagramme de Classe UML . . . 3
- 1.2 Description des Classes . . . 4

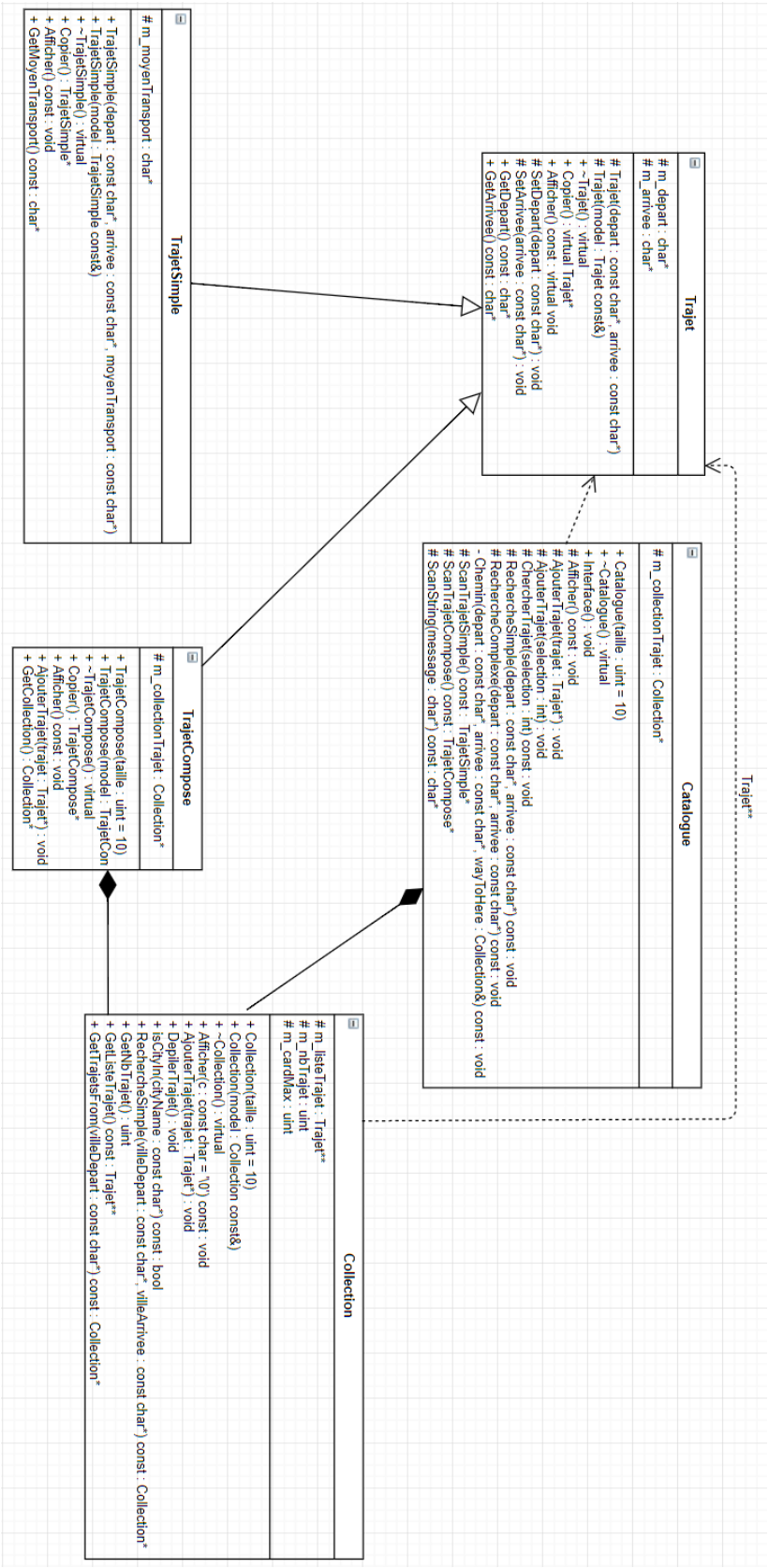
2 Description de la Structure de Données **5**

3 Listing des Classes Définies **6**
- 3.1 Main . . . 6
- 3.2 Collection . . . 7
  - 3.2.1 Interface . . . 7
  - 3.2.2 Réalisation . . . 10
- 3.3 Catalogue . . . 13
  - 3.3.1 Interface . . . 13
  - 3.3.2 Réalisation . . . 16
- 3.4 Trajet . . . 21
  - 3.4.1 Interface . . . 21
  - 3.4.2 Réalisation . . . 24
- 3.5 TrajetSimple . . . 27
  - 3.5.1 Interface . . . 27
  - 3.5.2 Réalisation . . . 29
- 3.6 TrajetCompose . . . 31
  - 3.6.1 Interface . . . 31
  - 3.6.2 Réalisation . . . 33
- 3.7 Makefile . . . 36

4 Conclusion **37**
- 4.1 Problèmes Rencontrés . . . 37
- 4.2 Axes d'Evolution et d'Amélioration . . . 37

1 Description Détaillée des Classes de l'Application

1.1 Diagramme de Classe UML



1.2 Description des Classes

.

Rôle de la classe Collection :
Structure de données permettant de stocker des Trajets. L'ajout dynamique est possible et est géré. Propose plusieurs services sur les Trajets.

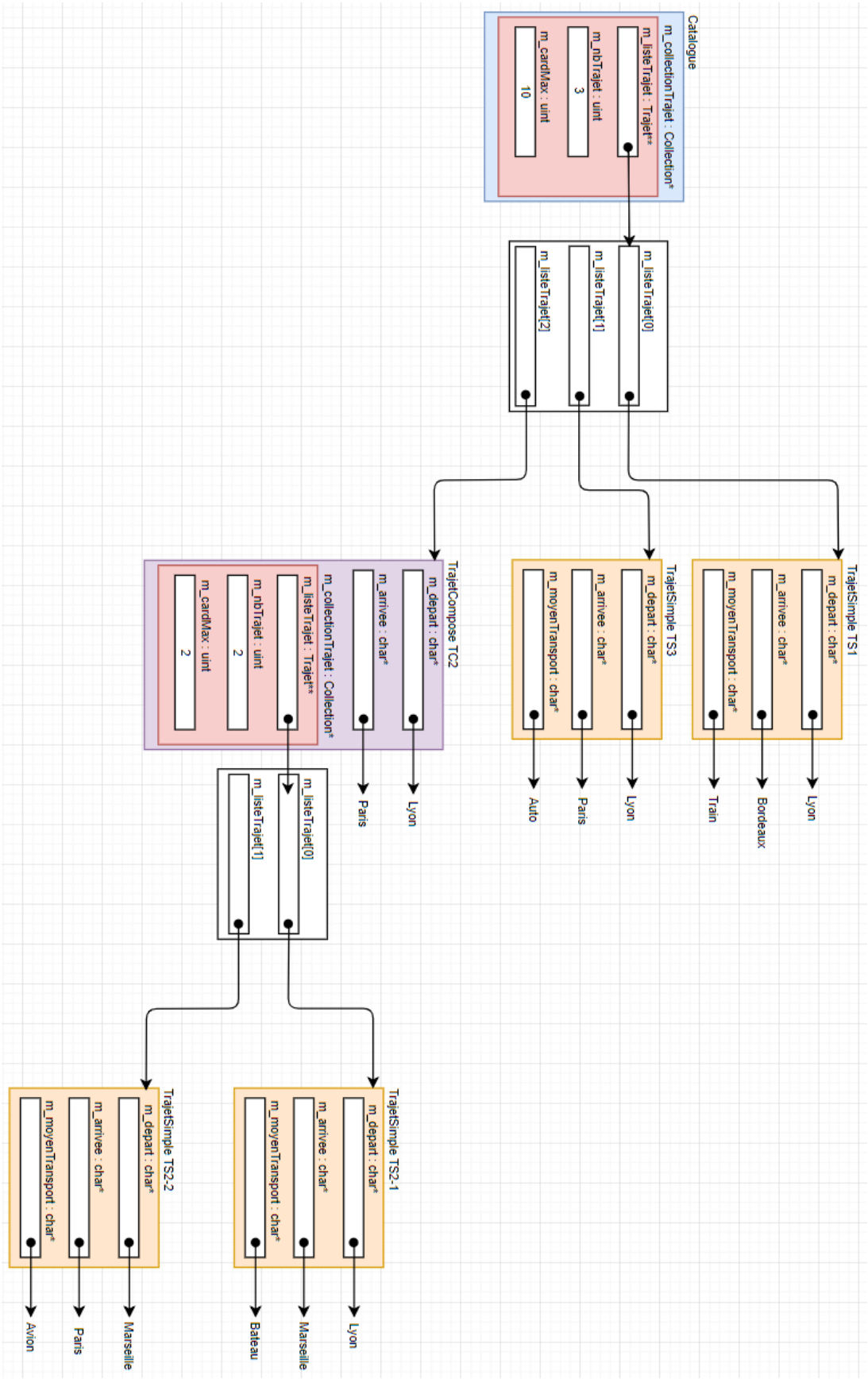
Rôle de la classe Catalogue :
Le rôle de la classe catalogue est de réaliser une UI et de gérer les trajets. Elle propose trois types de services à l'utilisateur :
- l'affichage du catalogue courant : affiche tous les trajets enregistrés dans le catalogue
- ajout de trajets : permet à l'utilisateur de créer de nouveaux trajets simples ou composes
- la recherche de trajets : l'utilisateur peut chercher la liste des parcours menant d'une ville de départ à une ville d'arrivée saisies en entrée

Rôle de la classe Trajet :
La classe trajet réprésente un trajet ayant une ville de départ et une ville d'arrivée.
Il instancie un objet de cette classe uniquement via le constructeur de ses spécialisations, TrajetSimple et TrajetCompose.

Rôle de la classe TrajetSimple :
La classe TrajetSimple est une spécification de la classe Trajet. Elle a comme attribut, en plus de ceux de Trajet, m_moyenTransport qui précise quel moyen de transport est utilisé pour un trajet.

Rôle de la classe TrajetCompose :
La classe TrajetCompose est une spécification de la classe Trajet. Elle a comme attribut, en plus de ceux de Trajet, une instance de la classe collection.

2 Description de la Structure de Données



3 Listing des Classes Définies

3.1 Main

main.cpp

```cpp
#include <stdio.h>
#include <iostream>
#include "catalogue.h"
#include "trajet.h"
#include "trajetsimple.h"
#include "trajetcompose.h"

using namespace std;

int main()
{
    Catalogue* catalogue = new Catalogue();
    catalogue->Interface();

    delete catalogue;

    return 0;
}
```

3.2 Collection

3.2.1 Interface

Collection.h

```
/*************************************************************************
Collection  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe <Collection> (fichier Collection.h) ----------------
#if ! defined ( Collection_H )
#define Collection_H

//--------------------------------------------------- Interfaces utilisees
#include "Trajet.h"
#include <cstring>
//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ Types
typedef unsigned int uint;

//-----------------------------------------------------------------------------
// Role de la classe <Collection>
// Structure de donnees permettant de stocker des Trajets
// L'ajout dynamique est possible et est gere
// Propose plusieurs services sur les Trajets
//-----------------------------------------------------------------------------

class Collection
{
    //------------------------------------------------------------------- PUBLIC

public:
    //-------------------------------------------------------- Methodes publiques

    void Afficher(const char prefixe = '\0') const;
    // Mode d'emploi :
    // Parcours tous les Trajets de la collection et appelle leur fonction Afficher()
    // -> prefixe peut servir a ajouter un character avant l'affichage de chaque Trajet , comme par exemple un \t

    void AjouterTrajet(Trajet*);
    // Mode d'emploi :
    // Ajoute un Trajet* a la Collection courante
    // Si la capacite maximale de la Collection est atteinte alors une nouvelle Collection de taille maximale egale au double
    // de la taille maximale de la premiere collection est creee. Dedans on copie la premiere Collection. Cette nouvelle Collection remplace la premiere.
```

```
45
46      void DepilerTrajet();
47      // Mode d'emploi :
48      // Supprime le dernier trajet ajoute dans la Collection
49      // Decremente le nombre de Trajets presents dans la Collection ( m_nbTrajet )
50      // Contrat :
51      // On suppose qu'on ne depile jamais une Collection vide (sans Trajets)
52
53      bool isCityIn(const char * cityName) const;
54      // Mode d'emploi :
55      // Retourne True si la ville cityName est presente en tant que ville d'arrivee ou
        ↪  de depart d'un Trajet de la Collection
56      // Sinon retourne False
57
58      Collection* GetTrajetsFrom(const char * villeDepart) const;
59      // Mode d'emploi :
60      // Renvoie la liste des Trajets presents dans la Collection dont la ville de
        ↪  depart correspond a villeDepart
61
62
63      Collection* RechercheSimple(const char * villeDepart, const char * villeArrivee)
        ↪  const;
64      // Mode d'emploi :
65      // Parcours la liste des Trajets de Collection, renvoie la liste des Trajets dont
66      // -> la ville de depart correspond a villeDepart
67      // -> la ville d'arrivee correspond a villeArrivee
68      // La combinaison de Trajets n'est PAS permise
69
70      uint GetNbTrajet() const;
71      // Mode d'emploi :
72      // Retourne la valeur de m_nbTrajet
73
74      Trajet** GetListeTrajet();
75      // Mode d'emploi :
76      // Retourne m_listeTrajet
77
78      //------------------------------------------- Constructeurs - destructeur
79
80      Collection(uint taille = 10);
81      //Cree une Collection de taille initiale 'taille'
82
83      Collection(Collection const& model);
84      //Constructeur par copie
85
86      virtual ~Collection();
87
88      //----------------------------------------------------------------- PRIVE
89
90  protected:
91      //------------------------------------------------------ Methodes protegees
92
```

```
93      //------------------------------------------------------ Attributs proteges
94      Trajet** m_listeTrajet; // Liste de pointeurs de Trajets
95      uint m_nbTrajet; // Nombre de Trajets dans la Collection
96      uint m_cardMax; // Nombre de Trajets maximal pouvant etre dans la Collection
97
98  };
99
100 #endif // Collection_H
```

3.2.2 Réalisation

Collection.cpp

```cpp
/*************************************************************************
Collection  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Realisation de la classe <Collection> (fichier Collection.cpp) ------------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include systeme
#include <iostream>
using namespace std;

//------------------------------------------------------ Include personnel
#include "Collection.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//-------------------------------------------------------- Methodes publiques


void Collection::Afficher(const char prefixe) const
{
    for (uint i = 0; i < m_nbTrajet; ++i)
    {
        cout << prefixe ;
        cout << i+1 << ") ";
        m_listeTrajet[i]->Afficher();
    }
}// ---- fin de Afficher

void Collection::AjouterTrajet(Trajet * trajet)
{

    if (m_nbTrajet == m_cardMax)
    {
        //Creation d'une nouvelle Collection
        m_cardMax *= 2;
        Trajet ** temp = new Trajet*[m_cardMax];
        //Copie des elements de la collection actuelle dans la nouvelle
        for (uint i = 0; i < m_nbTrajet; ++i)
        {
            temp[i] = m_listeTrajet[i];
```

```
        }
        m_listeTrajet = temp;
        delete[] temp;
    }
    //Ajout du trajet et incrementation du nombre de trajets
    m_listeTrajet[m_nbTrajet++] = trajet;
}// ---- fin de AjouterTrajet

void Collection::DepilerTrajet()
{
    --m_nbTrajet;
    delete m_listeTrajet[m_nbTrajet];
}// ---- fin de DepilerTrajet

bool Collection::isCityIn(const char * cityName) const
{
    for (uint i = 0; i < m_nbTrajet; ++i)
    {
        if (!strcmp(m_listeTrajet[i]->GetArrivee(), cityName) ||
        ↪  !strcmp(m_listeTrajet[i]->GetDepart(), cityName))return true;
    }
    return false;
}// ---- fin de isCityIn

Collection* Collection::GetTrajetsFrom(const char * depart) const
{
    Collection* itineraires = new Collection();
    for (uint i = 0; i < m_nbTrajet; ++i) {
        if (!strcmp(m_listeTrajet[i]->GetDepart(), depart)) {
            itineraires->AjouterTrajet(m_listeTrajet[i]->Copier());
        }
    }
    return itineraires;
}// ---- fin de GetTrajetsFrom

Collection* Collection::RechercheSimple(const char * depart, const char * arrivee)
↪  const
{
    Collection* itineraires = new Collection();
    for (uint i = 0; i < m_nbTrajet; ++i)
    {
        if (!strcmp(m_listeTrajet[i]->GetDepart(), depart) &&
        ↪  !strcmp(m_listeTrajet[i]->GetArrivee(), arrivee))
        {
            itineraires->AjouterTrajet(m_listeTrajet[i]);
        }
    }
    return itineraires;
}// ---- fin de RechercheSimple

uint Collection::GetNbTrajet() const
```

```
97  {
98      return m_nbTrajet;
99  }// ---- fin de GetNbTrajet
100
101 Trajet ** Collection::GetListeTrajet()
102 {
103     return m_listeTrajet;
104 }// ---- fin de GetListeTrajet
105
106 //------------------------------------------- Constructeurs - destructeur
107
108 Collection::Collection(uint taille)
109
110 {
111     m_nbTrajet = 0;
112     m_cardMax = taille;
113     m_listeTrajet = new Trajet*[m_cardMax];
114
115 #ifdef MAP
116     cout << "Appel au constructeur de <Collection>" << endl;
117 #endif
118 }// ---- fin de Collection
119
120 Collection::Collection(Collection const& model)
121 {
122     m_cardMax = model.m_cardMax;
123     m_nbTrajet = model.m_nbTrajet;
124     m_listeTrajet = new Trajet*[m_cardMax];
125
126     for (uint i = 0; i < m_nbTrajet; ++i)m_listeTrajet[i] =
      ↪  model.m_listeTrajet[i]->Copier();
127
128 #ifdef MAP
129     cout << "Appel au constructeur par copie de <Collection>" << endl;
130 #endif
131 }// ---- fin de Collection
132
133 Collection::~Collection()
134 {
135     for (uint i = 0; i < m_nbTrajet; i++)
136     {
137         delete m_listeTrajet[i];
138     }
139     delete[] m_listeTrajet;
140 #ifdef MAP
141     cout << "Appel au destructeur de <Collection>" << endl;
142 #endif
143 }// ---- fin de ~Collection
144
145 //------------------------------------------------------------------ PRIVE
146
```

```
147 //------------------------------------------------- Methodes protegees
```

3.3 Catalogue

3.3.1 Interface

Catalogue.h

```
1  /*************************************************************************
2  Catalogue  -  description
3  -------------------
4  debut                : 14 decembre 2018
5  copyright            : (C) 2018 par CERBA, RAUDRANT
6  e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
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 "TrajetSimple.h"
16 #include "TrajetCompose.h"
17 #include "Collection.h"
18
19 //------------------------------------------------------------ Constantes
20
21 //---------------------------------------------------------------- Types
22
23 //-------------------------------------------------------------------------
24 // Role de la classe <Catalogue>
25 //Le role de la classe catalogue est de realiser une UI.
26 //Elle propose trois types de services a l'utilisateur :
27 //- l'affichage du catalogue courant : affiche tous les trajets enregistres dans le
   ↪  catalogue
28 //- ajout de trajets : permet de hum... euh... a l'utilisateur de creer de nouveaux
   ↪  trajets simples ou composes
29 //- la recherche de trajets : l'utilisateur peut chercher la liste des parcours
   ↪  menant d'une ville de depart a une ville d'arrivee saisies en entree
30 //-------------------------------------------------------------------------
31
32 class Catalogue
33 {
34     //------------------------------------------------------------- PUBLIC
35
36 public:
37     //----------------------------------------------- Methodes publiques
38
39     void Interface();
40     // Mode d'emploi :
41     // Interface avec l'utilisateur , boucle jusqu'a l'ecriture de "bye"
```

```
42      //--numero de la commande a realiser
43      //1. Ajouter un trajet simple
44      //2. Ajouter un trajet compose
45      //3. Chercher un trajet
46      //4. Afficher tout le catalogue
47
48      //------------------------------------------ Constructeurs - destructeur
49
50      Catalogue(uint tailleDeBase = 10);
51      // Mode d'emploi :
52      // Constructeur d'un catalogue en donnant une taille initiale 'tailleDeBase' a la
        ↪  Collection 'm_collectionTrajet' du Catalogue
53
54
55      virtual ~Catalogue();
56      // Mode d'emploi : Fonction recursive appelant les destructeurs des objets qu'il
        ↪  contient
57
58      //---------------------------------------------------------------------- PRIVE
59
60      //------------------------------------------------------ Methodes privees
61  private:
62      void Chemin(const char * depart, const char * arrivee, Collection & wayToHere)
        ↪  const;
63      // Mode d'emploi : Fonction recursive
64      // Fonction appelee lors de la RechercheComplexe
65      // Retourne toutes les combinaisons des trajets du catalogue permettant de relier
        ↪  la ville de depart a celle d'arrivee
66      // wayToHere permet d'enregistrer le chemin parcouru par l'algorithme , pour le
        ↪  premier appel , lui donner une collection vide
67
68  protected:
69      //------------------------------------------------------ Methodes protegees
70
71      void Afficher() const;
72      // Mode d'emploi :
73      // Affiche l'ensemble des trajets contenus dans le catalogue
74      // Un trajet composants un trajet compose sont precedes d'une tabulation pour les
        ↪  differencier des autres trajets du catalogue
75      // Contrat :
76      // Il est suppose qu'un TrajetCompose n'est compose QUE de trajets simples
77
78      void AjouterTrajet(Trajet*);
79      // Mode d'emploi :
80      // Permet d'ajouter un Trajet au Catalogue
81      // -> dans le cas d'un TrajetCompose , une verification de validite sera
        ↪  appliquee a chaque trajets simples le composant
82      // afin de s'assurer que le TrajetCompose soit valide dans son integralite (ville
        ↪  de depart d'un trajet = ville d'arrivee du trajet le precedent)
83
84
```

```
85      void RechercheSimple(const char* depart, const char* arrivee) const;
86      // Mode d'emploi :une ville de depart et d'arrivee sont donnees en entree
87      // la fonction affiche la liste des trajets (simples ou composes) qui vont de la
        ↪  ville de depart a celle d'arrivee
88      // la composition de trajets n'est PAS prise en compte dans cette fonction
89      // Cette fonction gere l'affichage des resultats
90
91      void RechercheComplexe(const char*, const char*) const;
92      // Mode d'emploi :
93      //une ville de depart et d'arrivee sont donnees en entree
94      // la fonction affiche la liste des trajets (simples ou composes) qui vont de la
        ↪  ville de depart a celle d'arrivee
95      // la composition de trajets EST prise en compte dans cette fonction
96      // Contrat :
97      // -> les cycles ne sont pas consideres comme de protentielles solutions
98      // -> un trajet (simple ou compose) ne peut etre emprunte qu'une seule fois dans
        ↪  une meme solution
99
100     void AjouterTrajet(int selection);
101     // Mode d'emploi :
102     // Ajoute un Trajet au Catalogue en precisant via selection de quel type de
        ↪  trajet il s'agit
103     // Les verifications de coherence des trajets seront effectuees dans les classes
        ↪  des Trajets
104     // -> selection = 1 : TrajetSimple
105     // -> selection = 2 : TrajetCompose
106
107     TrajetSimple* ScanTrajetSimple() const;
108     // Mode d'emploi :
109     // Affiche les phrases d'interactions avec l'utilisateur
110     // Cree et renseigne un nouveau TrajetSimple et le retourne en sortie
111
112     TrajetCompose* ScanTrajetCompose() const;
113     // Mode d'emploi :
114     // Affiche les phrases d'interactions avec l'utilisateur
115     // Cree et renseigne un nouveau TrajetCompose et le retourne en sortie
116
117     void ChercherTrajet(int selection) const;
118     // Mode d'emploi :
119     // Demande a l'utilisateur de rentrer le nom de la ville de depart et d'arrivee
        ↪  de l'itineraire qu'il recherche
120     // Appelle ensuite une fonction de recherche
121
122     char* ScanString(const char*) const;
123     // Mode d'emploi :
124     //Demande a l'utilisateur le nom des villes et les moyens de transports des
        ↪  trajets qu'il insere dans le catalogue
125
126     //------------------------------------------------------ Attributs proteges
127     Collection* m_collectionTrajet;
128
```

```
};

//------------------------------- Autres definitions dependantes de <Catalogue>

#endif // Catalogue_H
```

3.3.2 Réalisation

Catalogue.cpp

```
/*************************************************************************
Catalogue  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Realisation de la classe <Catalogue> (fichier Catalogue.cpp)
 ↪  ------------

//------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include systeme
#include <iostream>
#include <cstring>
using namespace std;

//------------------------------------------------------ Include personnel
#include "Catalogue.h"

//-------------------------------------------------------------- Constantes

//------------------------------------------------------------------ PUBLIC

//----------------------------------------------------- Methodes publiques
// type Catalogue::Methode ( liste des parametres )
// Algorithme :
//
//{
//} //----- Fin de Methode


//-------------------------------------------- Constructeurs - destructeur


void Catalogue::Afficher() const
{
    m_collectionTrajet->Afficher();
}// ---- fin de Afficher


```

```
42  void Catalogue::AjouterTrajet(Trajet * trajet)
43  {
44      m_collectionTrajet->AjouterTrajet(trajet);
45  }// ---- fin de AjouterTrajet
46
47  void Catalogue::RechercheSimple(const char * depart, const char * arrivee) const
48  //Algorithme :
49  //Regarde tous les trajets du catalogue , si les villes de depart et d'arrivee
    ↪  correspondent
50  //L'ajoute a la collection rendue en retour
51  {
52      Collection* itineraires = new Collection();
53      itineraires = m_collectionTrajet->RechercheSimple(depart, arrivee);
54
55      if (itineraires->GetNbTrajet() != 0)
56          cout << "Voici la liste de(s) " << itineraires->GetNbTrajet() << " trajet(s)
            ↪  correspondant(s) a votre recherche : " << endl;
57      else
58          cout << "Aucun trajet ne correspond a votre recherche" << endl;
59
60      itineraires->Afficher();
61  }// ---- fin de RechercheSimple
62
63
64  void Catalogue::RechercheComplexe(const char * depart, const char * arrivee) const
65  {
66      Collection* chemins = new Collection();
67      Chemin(depart, arrivee, *chemins);
68      delete chemins;
69  }// ---- fin de RechercheComplexe
70
71
72  void Catalogue::Chemin(const char * depart, const char * arrivee, Collection &
    ↪  wayToHere) const
73  //Algorithme :
74  //Algorithme recursif de parcours en profondeur // backtracking
75  // Creation d'une Collection contenant l'ensemble des trajets partants de depart
76  // Pour chacun de ces trajets :
77  //          si la ville d'arrivee est la ville de destination souhaitee : on met a
    ↪  jour wayToHere , on l'affiche puis depile le dernier trajet ajoute
78  //          sinon on relance la recherche en mettant comme nouveau depart la ville
    ↪  d'arrivee du trajet , et on met a jour wayToHere que l'on envoie par reference
79  // On libere cette collection
80  {
81      Collection* departs = m_collectionTrajet->GetTrajetsFrom(depart);
82      for (uint i = 0; i < departs->GetNbTrajet(); ++i)
83      {
84          if (!strcmp(departs->GetListeTrajet()[i]->GetArrivee(), arrivee))
85          {
86              wayToHere.AjouterTrajet(departs->GetListeTrajet()[i]->Copier());
87              wayToHere.Afficher();
```

```
88              wayToHere.DepilerTrajet();
89          }
90          else if(!wayToHere.isCityIn(departs->GetListeTrajet()[i]->GetArrivee()))
91          {
92              wayToHere.AjouterTrajet(departs->GetListeTrajet()[i]->Copier());
93
94              Chemin(departs->GetListeTrajet()[i]->GetArrivee(), arrivee, wayToHere);
95              wayToHere.DepilerTrajet();
96          }
97      }
98      delete departs;
99  }// ---- fin de Chemin
100
101 Catalogue::Catalogue(uint taille)
102 {
103     m_collectionTrajet = new Collection(taille);
104 #ifdef MAP
105     cout << "Appel au constructeur de <Catalogue>" << endl;
106 #endif
107 }// ---- fin de Catalogue
108
109 Catalogue::~Catalogue()
110 {
111     delete m_collectionTrajet;
112 #ifdef MAP
113     cout << "Appel au destructeur de <Catalogue>" << endl;
114 #endif
115 }// ---- fin de ~Catalogue
116
117 void Catalogue::AjouterTrajet(int selection)
118 {
119     if (selection == 1)
120     {
121         AjouterTrajet(ScanTrajetSimple());
122     }
123     else
124     {
125         AjouterTrajet(ScanTrajetCompose());
126     }
127 }// ---- fin de AjouterTrajet
128
129 TrajetSimple* Catalogue::ScanTrajetSimple() const
130 //Entrees :
131 //--ville de depart
132 //--ville d'arrivee
133 //--moyen de transport
134 //du nouveau TrajetSimple cree
135 {
136     const char* depart = ScanString("Choisissez un depart : ");
137     const char* arrivee = ScanString("\nChoisissez une arrivee : ");
138     const char* transport = ScanString("\nChoisissez un moyen de transport : ");
```

```
139
140     TrajetSimple* unTrajetSimple = new TrajetSimple(depart, arrivee, transport);
141
142     delete[] depart;
143     delete[] arrivee;
144     delete[] transport;
145
146     return unTrajetSimple;
147 }// ---- fin de ScanTrajetSimple
148
149 TrajetCompose * Catalogue::ScanTrajetCompose() const
150 //Entrees :
151 //--nombre de trajets composant le trajet composee
152 //Appel a ScanTrajetSimple()
153 {
154     uint nbTrajets;
155     cout << "Combien de trajets simples composent ce nouveau trajet ? ";
156     cin >> nbTrajets;
157     if (nbTrajets != 0)
158     {
159         TrajetCompose* trajetC = new TrajetCompose();
160         for (uint i = 0; i < nbTrajets; ++i)
161         {
162             TrajetSimple* ts = ScanTrajetSimple();
163             while (!trajetC->AjouterTrajet(ts))
164             {
165                 delete ts;
166                 ts = ScanTrajetSimple();
167             }
168         }
169         return trajetC;
170     }
171     return nullptr;
172 }// ---- fin de ScanTrajetCompose
173
174 void Catalogue::ChercherTrajet(int selection) const
175 //Entrees :
176 //--nom de la ville de depart
177 //--nom de la ville d'arrivee
178 // de l'itineraire recherche dans le catalogue
179 {
180     const char* depart = ScanString("Choisissez un depart : ");
181     const char* arrivee = ScanString("\nChoisissez une arrivee : ");
182
183     if (selection == 3)
184     {
185         RechercheComplexe(depart, arrivee);
186     }
187     else
188     {
189         RechercheSimple(depart, arrivee);
```

```
190     }
191
192     delete[] depart;
193     delete[] arrivee;
194 } // ---- fin de ChercherTrajet
195
196
197 char * Catalogue::ScanString(const char * message) const
198 //Entrees :
199 //--Chaine de char a retourner
200 //utilisee pour demander a l'utilisateur le nom des villes et les moyens de
    ↪ transports
201 {
202     char * temp = new char[20];
203     cout << message << endl;
204     cin >> temp;
205     return temp;
206 }// ---- fin de ScanString
207
208 void Catalogue::Interface()
209 //Entrees :
210 //--numero de la commande a realiser
211 //1. Ajouter un trajet simple
212 //2. Ajouter un trajet compose
213 //3. Chercher un trajet
214 //4. Afficher tout le catalogue
215 {
216     char lecture[100];
217     cout << "Bienvenue !" << endl;
218     cout << "**************************" << endl;
219     cout << "1. Ajouter un trajet simple" << endl;
220     cout << "2. Ajouter un trajet compose" << endl;
221     cout << "3. Chercher un Trajet Complexe" << endl;
222     cout << "4. Chercher un Trajet Simple" << endl;
223     cout << "5. Afficher tout le catalogue" << endl;
224     cout << "**************************" << endl;
225
226     fscanf(stdin, "%99s", lecture);
227     while (strcmp(lecture, "bye") != 0)
228     {
229         if (strcmp(lecture, "1") == 0 || strcmp(lecture, "2") == 0)
230         {
231             AjouterTrajet(atoi(lecture));
232         }
233
234         if ((strcmp(lecture, "3") == 0) || strcmp(lecture, "4") == 0)
235         {
236             ChercherTrajet(atoi(lecture));
237         }
238         if (strcmp(lecture, "5") == 0)
239         {
```

```
            Afficher();
        }

        fscanf(stdin, "%99s", lecture);
    }
}// ---- fin de Interface

//-------------------------------------------------------------------- PRIVE

//------------------------------------------------------ Methodes protegees
```

3.4 Trajet

3.4.1 Interface

Trajet.h

```
/*************************************************************************
Trajet  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe <Trajet> (fichier Trajet.h) ----------------
#if ! defined ( Trajet_H )
#define Trajet_H

//------------------------------------------------------ Interfaces utilisees

//------------------------------------------------------------ Constantes

//---------------------------------------------------------------- Types

//------------------------------------------------------------------------
// Role de la classe <Trajet>
// La classe trajet represente un trajet ayant une ville de depart et une
// ville d'arrivee.
// Il instancie un objet de cette classe uniquement via le constructeur de
// ses specialisations, TrajetSimple et TrajetCompose.
//
//------------------------------------------------------------------------



class Trajet
{
    //-------------------------------------------------------------- PUBLIC

public:
    //----------------------------------------------------- Methodes publiques

```

```
37      virtual void Afficher() const;
38      // Mode d'emploi :
39      // Affiche sur la console une description du trajet, c'est a dire son depart
40      // et son arrivee.
41      // Contrat :
42      // Il faut que cette methode soit redefinie dans ses specialisations
43
44      virtual Trajet* Copier();
45      // Mode d'emploi :
46      // Permet d'appeler le constructeur par copie de ses specialisations
47      // Contrat :
48      // Il faut que cette methode soit redefinie dans ses specialisations
49
50      char* GetDepart() const;
51      // Mode d'emploi :
52      // Retourne la ville de depart d'une instance de la classe trajet, en tant
53      // que pointeur sur une chaine de caracteres.
54
55      char* GetArrivee() const;
56      // Mode d'emploi :
57      // Retourne la ville d'arrivee d'une instance de la classe trajet, en tant
58      // que pointeur sur une chaine de caracteres.
59
60      //------------------------------------------ Constructeurs - destructeur
61
62      virtual ~Trajet();
63      // Mode d'emploi (destructeur) :
64      // Destructeur d'une instance de la classe Trajet.
65      // Contrat :
66      // Il doit bien supprimer tous les attributs de l'instance
67
68      //----------------------------------------------------------------- PRIVE
69
70  protected:
71      //---------------------------------------------------- Constructeurs proteges
72
73      Trajet(const char* depart, const char* arrivee);
74      // Mode d'emploi (constructeur) :
75      // Construction d'une instance de Trajet a partir des parametres depart et
76      // arrivee.
77      // Contrat :
78      // Doit etre appele depuis une des specialisations de Trajet.
79
80      Trajet(Trajet const&);
81      // Mode d'emploi (constructeur par copie) :
82      // Permet de creer un nouveau trajet en copiant l'instance de trajet qui lui
83      // est passe par reference.
84      // Contrat :
85      // Doit etre appele depuis une des specialisations de Trajet.
86
87      //------------------------------------------------------ Methodes protegees
```

```
88
89      void SetDepart(const char *);
90      // Mode d'emploi :
91      // Fonction qui permet de changer l'attribut m_depart d'une instance de la
92      // classe Trajet qui prend en parametre le nouveau depart.
93
94      void SetArrivee(const char *);
95      // Mode d'emploi :
96      // Fonction qui permet de changer l'attribut m_arrivee de Trajet qui prend en
97      // parametre le nouveau depart.
98
99      //------------------------------------------------------ Attributs proteges
100     char* m_depart;
101     char* m_arrivee;
102
103 };
104
105 //-------------------------------- Autres definitions dependantes de <Trajet>
106
107 #endif // TRAJET_H
```

3.4.2 Réalisation

Trajet.cpp

```cpp
/*************************************************************************
Trajet  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Realisation de la classe <Trajet> (fichier Trajet.cpp) ------------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include systeme
#include <iostream>
#include <cstring>
#include <stdio.h>
using namespace std;

//------------------------------------------------------ Include personnel
#include "trajet.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//-------------------------------------------------------- Methodes publiques

void Trajet::Afficher() const
// Algorithme :
// Affiche les attributs de Trajet
{
    cout << "Depart : " << m_depart << ", Arrivee : " << m_arrivee;
} //----- Fin de Afficher

Trajet * Trajet::Copier()
// Algorithme :
// Methode virtuelle donc va chercher la fonction Copier de la specialisation
// correspondante.
{
    return nullptr;
} //----- Fin de Copier

char* Trajet::GetDepart() const
// Algorithme :
// Retourne le depart d'une instance de TrajetSimple.
{
    return m_depart;
} //----- Fin de GetDepart

```

```cpp
char* Trajet::GetArrivee() const
// Algorithme :
// Retourne l'arrivee d'une instance de TrajetSimple.
{
    return m_arrivee;
} //----- Fin de GetArrivee

//------------------------------------------- Destructeur

Trajet::~Trajet()
// Algorithme :
// Destructeur d'une instance de Trajet.
{
    delete[] m_depart;
    delete[] m_arrivee;
#ifdef MAP
    cout << "Appel au destructeur de <Trajet>" << endl;
#endif
} //----- Fin de ~TrajetSimple

//------------------------------------------------------------------ PRIVE

//------------------------------------------------------ Methodes protegees

void Trajet::SetDepart(const char * depart)
// Algorithme :
// Modifie l'attribut m_depart par le parametre formel.
{
    int lg = strlen(depart);
    if (m_depart != nullptr) delete[] m_depart;
    m_depart = new char[lg + 1];
    m_depart = strcpy(m_depart, depart);
} //----- Fin de SetDepart

void Trajet::SetArrivee(const char * arrivee)
// Algorithme :
// Modifie l'attribut m_arrivee par le parametre formel.
{
    int lg = strlen(arrivee);
    if (m_arrivee != nullptr)delete[] m_arrivee;
    m_arrivee = new char[lg + 1];
    m_arrivee = strcpy(m_arrivee, arrivee);
} //----- Fin de SetArrivee

//-------------------------------------------- Constructeurs

Trajet::Trajet(const char* depart, const char* arrivee)
// Algorithme :
// Constructeur qui prend comme argument un depart, une arrivee.
{
    int lg = strlen(depart);
```

```
101     m_depart = new char[lg + 1];
102     m_depart = strcpy(m_depart, depart);
103
104
105     lg = strlen(arrivee);
106     m_arrivee = new char[lg + 1];
107     m_arrivee = strcpy(m_arrivee, arrivee);
108
109 #ifdef MAP
110     cout << "Appel au constructeur de <Trajet>" << endl;
111 #endif
112 } //----- Fin de Trajet (constructeur)
113
114 Trajet::Trajet(Trajet const& model)
115 // Algorithme :
116 // Constructeur par copie qui... copie une reference d'un Trajet donne
117 // en parametre.
118 {
119     m_arrivee = model.m_arrivee;
120     m_depart = model.m_depart;
121
122 #ifdef MAP
123     cout << "Appel au constructeur par copie de <Trajet>" << endl;
124 #endif
125 } //----- Fin de Trajet (constructeur de copie)
```

3.5 TrajetSimple

3.5.1 Interface

TrajetSimple.h

```
/*************************************************************************
TrajetSimple  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe <TrajetSimple> (fichier TrajetSimple.h) ----------------
#if ! defined ( TrajetSimple_H )
#define TrajetSimple_H

//--------------------------------------------------- Interfaces utilisees
#include "trajet.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ Types

//-----------------------------------------------------------------------------
// Role de la classe <TrajetSimple>
// La classe TrajetSimple est une specification de la classe Trajet.
// Elle a comme attribut, en plus de ceux de Trajet, m_moyenTransport qui
// precise quel moyen de transport est utilise pour un trajet.
//-----------------------------------------------------------------------------

class TrajetSimple : public Trajet
{
    //------------------------------------------------------------------- PUBLIC

public:
    //----------------------------------------------------- Methodes publiques

    void Afficher() const;
    // Mode d'emploi :
    // Affiche sur la console une description du trajet simple. Pour cela, cette
    // methode appelle aussi la methode Afficher sa generalisation.

    TrajetSimple* Copier();
    // Mode d'emploi :
    // Methode appelee par la generalisation de cette classe (ie Trajet). Elle
    // appelle le constructeur par copie de la classe TrajetSimple, en lui donnant
    // en parametre l'objet appelant cette methode.

    char* GetMoyenTransport() const;
    // Mode d'emploi :
    // Retourne le moyen de transport d'une instance de la classe trajet simple,
```

```
48      // en tant que pointeur sur une cha de caracteres.
49
50      //------------------------------------------ Constructeurs - destructeur
51
52      TrajetSimple(const char* depart, const char* arrivee, const char*
        ↪  moyenTransport);
53      // Mode d'emploi (constructeur):
54      // Construit une nouvelle instance de TrajetSimple a partir d'un depart, d'une
55      // arrivee et d'un moyen de transport.
56
57      TrajetSimple(TrajetSimple const& model);
58      // Mode d'emploi (constructeur par copie) :
59      // Construit une nouvelle instance de la classe TrajetSimple en copiant
        ↪  l'instance
60      // de TrajetSimple qui lui est passe par reference.
61
62      virtual ~TrajetSimple();
63      // Mode d'emploi (destructeur) :
64      // Destructeur d'une instance de la classe TrajetSimple.
65      // Contrat :
66      // Il doit bien supprimer tous les attributs de l'instance
67
68
69      //----------------------------------------------------------------- PRIVE
70
71  protected:
72      //---------------------------------------------------- Methodes protegees
73
74      //------------------------------------------------------ Attributs proteges
75      char* m_moyenTransport;
76  };
77
78  //-------------------------------- Autres definitions dependantes de <Xxx>
79
80  #endif // TRAJETSIMPLE_H
```

3.5.2 Réalisation

TrajetSimple.cpp

```cpp
/*************************************************************************
TrajetSimple  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Realisation de la classe <trajetsimple> (fichier TrajetSimple.cpp) ----------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include systeme
#include <iostream>
#include <cstring>
using namespace std;

//-------------------------------------------------------- Include personnel
#include "TrajetSimple.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//-------------------------------------------------------- Methodes publiques
void TrajetSimple::Afficher() const
// Algorithme :
// Appelle la methode Afficher de Trajet (pour afficher le depart et l'arrivee).
// Puis afficher dans la console le moyen de transport utilise.
{
    Trajet::Afficher();
    cout << ", Moyen de Transport : " << m_moyenTransport << endl;
} //----- Fin de Afficher

TrajetSimple* TrajetSimple::Copier()
// Algorithme :
// Appelle le constructeur par copie de TrajetSimple en lui passant en
// parametre l'objet appelant.
{
    return new TrajetSimple(*this);
} //----- Fin de Copier

char* TrajetSimple::GetMoyenTransport() const
// Algorithme :
// Retourne le moyen de transport utilise dans une instance de TrajetSimple.
{
    return m_moyenTransport;
} //----- Fin de GetMoyenTransport
```

```
49
50
51  //---------------------------------------- Constructeurs - destructeur
52
53  TrajetSimple::TrajetSimple(const char* depart, const char* arrivee, const char*
    ↪  moyenTransport) : Trajet(depart, arrivee)
54  // Algorithme :
55  // Constructeur qui prend comme argument un depart, une arrivee et un moyen
56  // de transport.
57  {
58      int lg = strlen(moyenTransport);
59      m_moyenTransport = new char[lg + 1];
60      m_moyenTransport = strcpy(m_moyenTransport, moyenTransport);
61
62  #ifdef MAP
63      cout << "Appel au constructeur de <TrajetSimple>" << endl;
64  #endif
65  } //----- Fin de TrajetSimple (constructeur)
66
67  TrajetSimple::TrajetSimple(TrajetSimple const& model) : Trajet(model.m_depart,
    ↪  model.m_arrivee)
68  // Algorithme :
69  // Constructeur par copie qui... copie une reference d'un TrajetSimple donne
70  // en parametre.
71  {
72      m_moyenTransport = new char[strlen(model.m_moyenTransport)+1];
73      strcpy(m_moyenTransport,model.m_moyenTransport);
74  #ifdef MAP
75      cout << "Appel au constructeur par copie de <TrajetSimple>" << endl;
76  #endif
77  } //----- Fin de TrajetSimple (constructeur de copie)
78
79
80  TrajetSimple::~TrajetSimple()
81  // Algorithme :
82  // Destructeur d'une instance de TrajetSimple. Appelle automatiquement le
83  // destructeur de sa generalisation.
84  {
85      delete[] m_moyenTransport;
86  #ifdef MAP
87      cout << "Appel au destructeur de <TrajetSimple>" << endl;
88  #endif
89  } //----- Fin de ~TrajetSimple
90
91  //------------------------------------------------------------------ PRIVE
92
93  //---------------------------------------------------- Methodes protegees
```

3.6 TrajetCompose

3.6.1 Interface

TrajetCompose.h

```
/*************************************************************************
TrajetCompose  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Interface de la classe <TrajetCompose> (fichier TrajetCompose.h) ----------
#if ! defined ( TrajetCompose_H )
#define TrajetCompose_H

//--------------------------------------------------- Interfaces utilisees
#include "trajet.h"
#include "collection.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ Types
typedef unsigned int uint;

//-----------------------------------------------------------------------------
// Role de la classe <TrajetCompose>
// La classe TrajetCompose est une specification de la classe Trajet.
// Elle a comme attribut, en plus de ceux de Trajet, une instance de la
// classe collection.
//-----------------------------------------------------------------------------

class TrajetCompose : public Trajet
{
    //------------------------------------------------------------------- PUBLIC

public:
    //----------------------------------------------------- Methodes publiques

    void Afficher() const;
    // Mode d'emploi :
    // Affiche sur la console une description du trajet compose. Pour cela, cette
    // methode appelle aussi la methode Afficher sa generalisation.

    bool AjouterTrajet(Trajet *);
    // Mode d'emploi :
    // Methode qui ajoute un trajet simple dans la collection de l'instance du
    // trajet compose.

    TrajetCompose* Copier();
    // Mode d'emploi :
```

```
48      // Methode appelee par la generalisation de cette classe (ie Trajet). Elle
49      // appelle le constructeur par copie de la classe TrajetCompose, en lui donnant
50      // en parametre l'objet appelant cette methode.
51
52      //------------------------------------------ Constructeurs - destructeur
53
54      TrajetCompose(uint taille = 10);
55      // Mode d'emploi (constructeur) :
56      // Construit une nouvelle instance de TrajetCompose a partir d'une taille,
57      // mise par defaut a 10 si elle n'est pas renseignee.
58
59      TrajetCompose(TrajetCompose const&);
60      // Mode d'emploi (constructeur par copie) :
61      // Construit une nouvelle instance de la classe TrajetCompose en copiant
62      // l'instance de TrajetCompose qui lui est passe par reference.
63
64      virtual ~TrajetCompose();
65      // Mode d'emploi (destructeur) :
66      // Destructeur d'une instance de la classe TrajetCompose.
67      // Contrat :
68      // Il doit bien supprimer tous les attributs de l'instance
69
70      //----------------------------------------------------------------- PRIVE
71
72  protected:
73      //------------------------------------------------------ Methodes protegees
74
75      //------------------------------------------------------ Attributs proteges
76      Collection *m_collectionTrajet;
77  };
78
79  //-------------------------------- Autres definitions dependantes de <TrajetCompose>
80
81  #endif // TRAJETCOMPOSE_H
```

3.6.2 Réalisation

TrajetCompose.cpp

```cpp
/*************************************************************************
TrajetCompose  -  description
-------------------
debut                : 14 decembre 2018
copyright            : (C) 2018 par CERBA, RAUDRANT
e-mail               : guilhem.cerba@insa-lyon.fr, sophie.raudrant@insa-lyon.fr
*************************************************************************/

//---------- Realisation de la classe <TrajetCompose> (fichier TrajetCompose.cpp) ----------

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include systeme
#include <iostream>
using namespace std;

//------------------------------------------------------ Include personnel
#include "trajetcompose.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//-------------------------------------------------------- Methodes publiques

void TrajetCompose::Afficher() const
// Algorithme :
// Appelle la methode Afficher de sa collection.
{
    cout << "Trajet compose : " << endl;
    m_collectionTrajet->Afficher('\t');
} //----- Fin de Afficher

bool TrajetCompose::AjouterTrajet(Trajet * trajet)
// Algorithme :
// Ajoute un trajet simple a la collection du trajet compose precedemment cree.
// Si le trajet compose ne possede encore aucun trajet simple, on modifie
// le depart et l'arrivee de ce trajet compose
// Si le trajet compose possede deja un/des trajet(s) simple(s), on verifie
// que le trajet est coherent. Si c'est le cas, on modifie uniquement l'arrivee.
{
    if (m_collectionTrajet->GetNbTrajet() == 0)
    {
        this->SetArrivee(trajet->GetArrivee());
        this->SetDepart(trajet->GetDepart());
        m_collectionTrajet->AjouterTrajet(trajet);
    }
    else
```

```
49      {
50          if (!strcmp(trajet->GetDepart(),
            ↪  m_collectionTrajet->GetListeTrajet()[m_collectionTrajet->GetNbTrajet()
            ↪  -1]->GetArrivee()))
51          {
52              this->SetArrivee(trajet->GetArrivee());
53              m_collectionTrajet->AjouterTrajet(trajet);
54          }
55          else {
56              cout << "Trajet incoherent" << endl;
57              return false;
58          }
59
60      }
61      return true;
62  } //----- Fin de AjouterTrajet
63
64  TrajetCompose * TrajetCompose::Copier()
65  // Algorithme :
66  // Appelle le constructeur par copie de TrajetSimple en lui passant en
67  // parametre l'objet appelant.
68  {
69      return new TrajetCompose(*this);
70  } //----- Fin de Copier
71
72  //------------------------------------------- Constructeurs - destructeur
73
74  TrajetCompose::TrajetCompose(uint taille) : Trajet("", "")
75  // Algorithme :
76  // Constructeur qui initialise la collecion d'une instance de trajet
77  // compose a la taille passee en parametre.
78  {
79      m_collectionTrajet = new Collection(taille);
80
81  #ifdef MAP
82      cout << "Appel au constructeur de <TrajetCompose>" << endl;
83  #endif
84  } //----- Fin de TrajetCompose (constructeur [par defaut])
85
86  TrajetCompose::TrajetCompose(TrajetCompose const& model) : Trajet(model.m_depart,
    ↪  model.m_arrivee)
87  // Algorithme :
88  // Constructeur par copie qui... copie une reference d'un TrajetCompose donne
89  // en parametre.
90  {
91      m_collectionTrajet = new Collection(*model.m_collectionTrajet);
92
93  #ifdef MAP
94      cout << "Appel au constructeur par copie de <TrajetCompose>" << endl;
95  #endif
96  } //----- Fin de TrajetCompose (constructeur par copie)
```

```
97
98  TrajetCompose::~TrajetCompose()
99  // Algorithme :
100 // Destructeur d'une instance de TrajetCompose. Appelle automatiquement le
101 // destructeur de sa generalisation.
102 {
103     delete m_collectionTrajet;
104 #ifdef MAP
105     cout << "Appel au destructeur de <TrajetCompose>" << endl;
106 #endif
107 } //----- Fin de ~TrajetCompose
```

3.7 Makefile

makefile

```
COMP = g++
COMPFLAGS = -ansi -pedantic -Wall -std=c++11
VALFLAGS = -g -O0
HEADER=$(wildcard *.h)
SOURCE=$(wildcard *.cpp)
OBJET=$(SOURCE:.cpp=.o)

ifdef MAP
    COMPFLAGS += -DMAP
endif

%.o : %.cpp %.h
    $(COMP) $(COMPFLAGS) $(DEFINES) -c $<

main : $(OBJET)
    $(COMP) -o main $(VALFLAGS) $(OBJET)

valgrind :
    valgrind --leak-check=yes ./main
```

4 Conclusion

4.1 Problèmes Rencontrés

Le majeur problème que nous avons rencontré lors de la création de l'application fût l'implémentation de la recherche complexe. Nous avons d'abord essayé de résoudre le problème qui se posait à nous en recherchant une solution dynamique, mais cela s'est soldé avec un échec.
Nous avons donc mis en place une solution avec une fonction récursive qui permet de parcourir toutes les solutions possibles et qui s'arrête dès qu'il trouve une solution de chemin ou dès que la recherche ne peut pas continuer (c'est-à-dire qu'il n'y a plus de trajet qui a pour départ l'arrivée du trajet actuel de recherche).

Nous avons rencontré un autre problème en début d'implémentation lorsqu'il a fallu passer des const char* au constructeur de TrajetSimple. Mais nous l'avons simplement résolu en passant par une variable temporaire.

4.2 Axes d'Evolution et d'Amélioration

L'un des axes d'évolution qui peut sembler moindre mais qui est important pour l'utilisation, est de permettre à l'utilisateur de rentrer des noms de ville avec ou sans majuscules sans que ça dérange pour la recherche. En effet, actuellement l'algorithme de recherche est sensible aux majuscules (case-sensitive).

L'ajout d'attributs 'heure de départ' et 'heure d'arrivée' dans un Trajet peut être envisagée. Celà permettrait des recherches plus pertinentes pour l'utilisateur et faciliterait les recherches.

Ajouter des distances entre les villes ou se servir des attributs mentionnés dans le paragraphe ci-dessus pour implémenter des algorithmes de recherche d'un trajet optimal (comme A* par exemple) qui peuvent représenter un réel intérêt pour l'utilisateur.

Ajouter du multi-threading lors de la recherche complexe aurait pu être une bonne amélioration, dans les cas où il y a beaucoup de trajets dans un catalogue, sinon ce n'est pas assez rentable de diviser la tâche. Il est possible de multi-threader notre programme pour chaque trajet ayant le même départ que le trajet recherché (GetTrajetsFrom()).

Quand on demande le nombre de trajets qu'il y aura dans un trajet composé, il faut imposer à l'utilisateur de rentrer un nombre.