

TP Héritage - Polymorphisme

Catalogue de voyages: une application en C++

I. Introduction

```
Welcome to Trip Scanner
What would you like to do?
===== MENU =====
1: Enter 1 to add an itinerary.
2: Enter 2 to display current catalogue.
3: Enter 3 to look for an itinerary.
4: Enter 4 to make an advanced search.
5: Enter 5 to exit.
```

L'objectif du TP est de mettre en place une application en C++ qui implémente un catalogue de trajets afin de pouvoir proposer des parcours pour un voyage défini par sa ville de départ et sa destination.

L'application est basée sur de la programmation orientée objet avec une utilisation conséquente de l'héritage en exploitant le polymorphisme ainsi que des principes d'allocation dynamique de mémoire.

Ce compte rendu fait office de documentation pour l'application. Il explique son fonctionnement, détaille ses classes et fournit une représentation graphique de la structure de données adoptée.

II. Description détaillée de toutes les classes de l'application

L'application est constituée de cinq classes dont une classe main qui implémente également l'interface graphique avec laquelle l'utilisateur de l'application interagit pour éditer et interroger un catalogue de voyages. Cette interface propose quatre fonctionnalités: ajout d'un trajet, affichage du catalogue courant, recherche simple d'un trajet et enfin une option de recherche avancée qui permet dans le cas où c'est possible de combiner différents trajets pour proposer une solution composée si l'itinéraire demandé n'existe pas directement dans le catalogue.

Pour ce faire, nous avons créé une classe mère 'Trajet' qui sert ensuite pour manipuler ses classes héritières. De cette dernière héritent deux classes filles, une classe 'TrajetSimple' où se rajoute un attribut 'transport , et une classe 'TrajetComplexe' avec comme attributs supplémentaires un tableau d'instances de TrajetSimple ainsi que sa taille définissant le nombre de trajets simples constituant le trajet composé. La dernière classe 'Catalogue' implémente, comme son nom l'indique une collection d'objets hétérogènes (trajets simples ou composés) que l'on peut gérer sans aucune distinction de leurs natures.

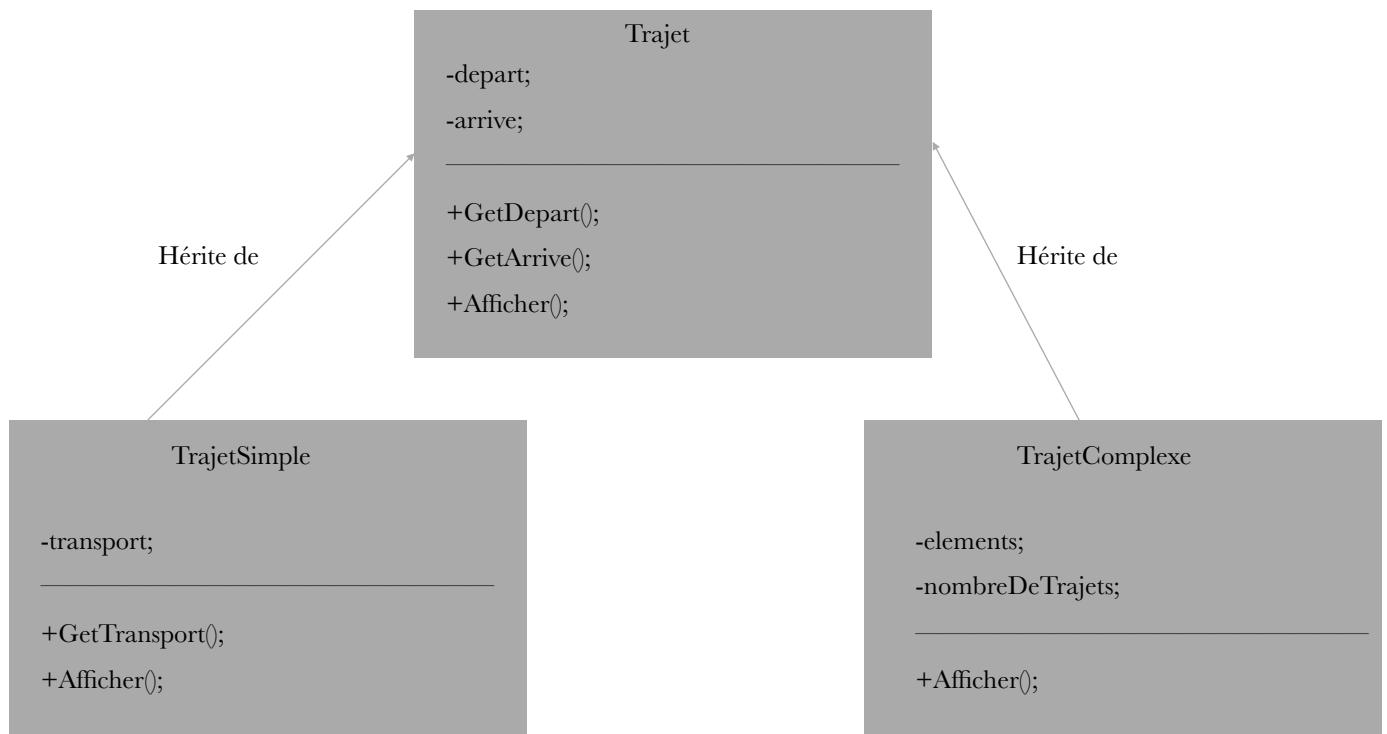


Figure 1: Graphe d'héritage.

La figure ci-dessus représente l'héritage entre les classes implémentant des trajets. L'affichage n'étant pas le même selon la nature du trajet, chacune des classes héritières possède une méthode d'affichage propre à elle, avec une première déclaration (virtual) sans définition dans le fichier interface de la classe mère.

Ci-après est le diagramme UML récapitulant les classes et la structure de l'application. Pour des raisons de lisibilité, ce diagramme ne contient pas les constructeurs et destructeurs des classes. Pour ces derniers, veuillez vous référer au listing des classes (chapitre IV).

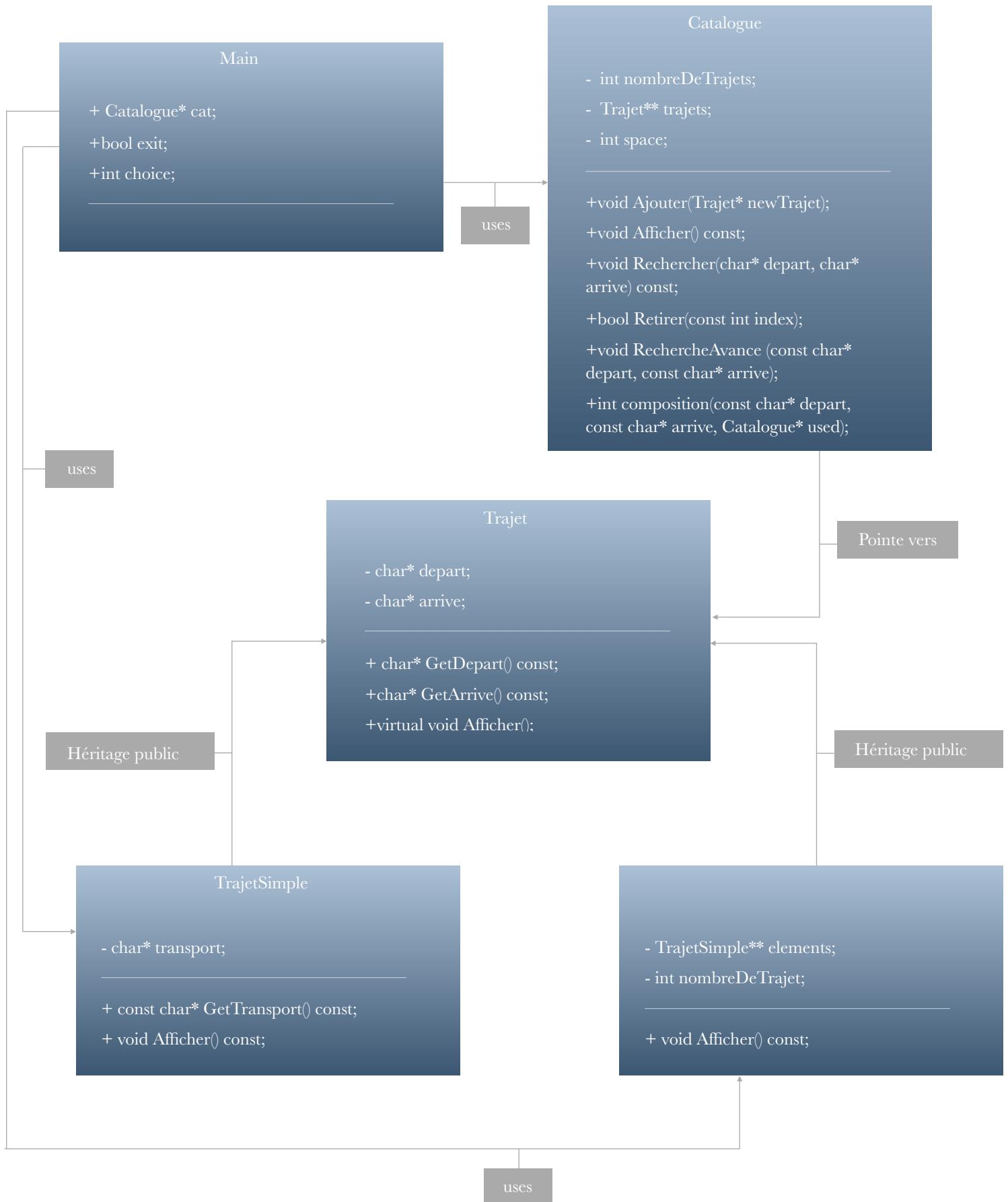


Figure 2: Diagramme UML.

III. Description de la structure de données

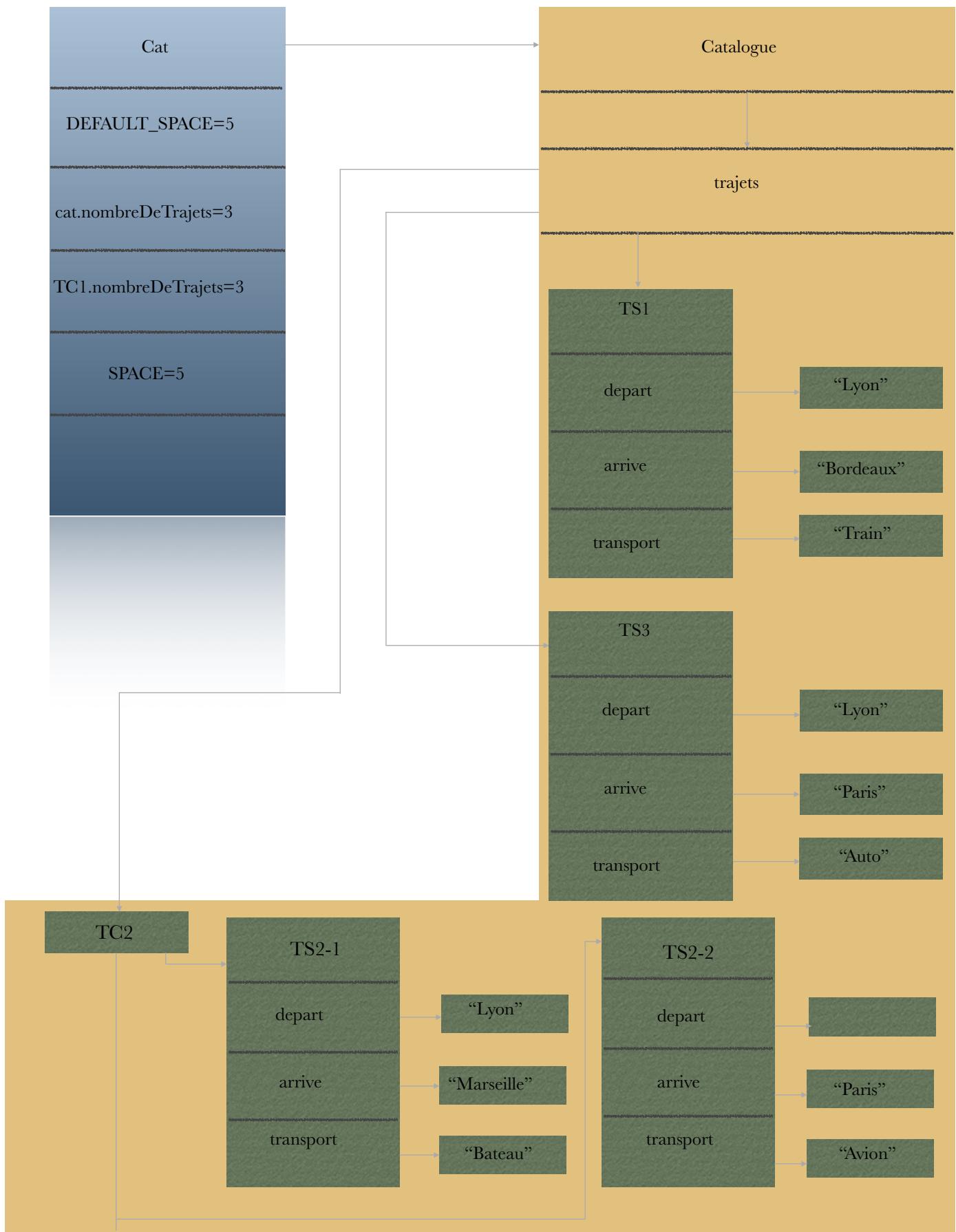


Figure 3: État de la mémoire pour le jeu d'essai de l'énoncé.

IV. Listing des classes

IV.1. Trajet

IV.1.a Interface

```

/***** Trajet - description *****/
-----  

début      : $20/11/2019$  

copyright   : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,  

              Zihao Hua$  

e-mail      : $sophie.crowley@insa-lyon.fr$  

              $zakaria.nassreddine@insa-lyon.fr$  

              $zihao.hua@insa-lyon.fr$  

***** /  

//----- Interface de la classe <Trajet> (fichier Trajet.h) -----  

#ifndef ! defined ( TRAJET_H )  

#define TRAJET_H  

//-----  

// Rôle de la classe <Trajet>  

// Cette classe déclare un trajet de base (la classe mère).  

//-----  

class Trajet  

{  

//----- PUBLIC  

public:  

//----- Méthodes publiques  

    char* GetDepart() const;  

    // Mode d'emploi : Cette méthode renvoie la ville de départ du trajet.

```

```
char* GetArrive() const;
// Mode d'emploi : Cette méthode renvoie la ville d'arrivée du trajet.

virtual void Afficher() const {};
// Mode d'emploi : Cette méthode affiche la ville de départ du trajet.
// Type: ne renvoie rien.
// Contrat: définie dans les classes héritières.

//----- Constructeurs - destructeur
Trajet ( const Trajet & unTrajet );
// Mode d'emploi (constructeur de copie) :

Trajet ( );
// Mode d'emploi (constructeur 1) :
Trajet (char* depart, char* arrive);
// Mode d'emploi : (constructeur 2)

virtual ~Trajet ( );
// Mode d'emploi : (destructeur)

//----- PRIVE
protected:
//----- Méthodes protégées

//----- Attributs protégés
char *depart;
char *arrive;
};

//----- Autres définitions dépendantes de <Trajet>

#endif // TRAJET_H
```

IV.1.b Réalisation

```

/***** Trajet - description *****/
-----  

début      : $20/11/2019$  

copyright   : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,  

              Zihao Hua$  

e-mail      : $sophie.crowley@insa-lyon.fr$  

              $zakaria.nassreddine@insa-lyon.fr$  

              $zihao.hua@insa-lyon.fr  

*****/  

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

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

//----- Include système -----  

#include <iostream>  

#include <cstring>  

using namespace std;  

//----- Include personnel -----  

#include "Trajet.h"  

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

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

//----- Méthodes publiques -----  

char* Trajet::GetDepart() const {  

    return depart;  

}  

char* Trajet::GetArrive() const {  

    return arrive;  

}  

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

```

Trajet::Trajet ( const Trajet & unTrajet )
{
#ifdef MAP
    cout << "Appel au constructeur de copie de <Trajet>" << endl;
#endif
} //---- Fin de Trajet (constructeur de copie)

Trajet::Trajet ( )
// Algorithme :
//
{
#ifdef MAP
    cout << "Appel au constructeur de <Trajet>" << endl;
#endif
} //---- Fin de Trajet

Trajet::Trajet (char* departInput, char* arriveInput)
{
#ifdef MAP
    cout << "Appel au constructeur 2 de <Trajet>" << endl;
#endif
    arrive = new char[strlen(arriveInput)+1];
    strcpy(arrive, arriveInput);
    arrive[strlen(arriveInput)]='\0';
    depart = new char[strlen(departInput)+1];
    strcpy(depart, departInput);
    depart[strlen(departInput)]='\0';
} //---- Fin de Trajet

Trajet::~Trajet ( )
{
#ifdef MAP
    cout << "Appel au destructeur de <Trajet>" << endl;
#endif
} //---- Fin de ~Trajet

```

IV.2 Trajet Simple

IV.2.a Interface

```

/***** TrajetSimple - description *****/
TrajetSimple - description

-----
début      : $20/11/2019$
copyright  : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,
              Zihao Hua$
e-mail     : $sophie.crowley@insa-lyon.fr$,
              $zakaria.nassreddine@insa-lyon.fr$,
              $zihao.hua@insa-lyon.fr$

***** /


//----- Interface de la classe <TrajetSimple> (fichier TrajetSimple.h)
-----

#ifndef ! defined ( TRAJETSIMPLE_H )
#define TRAJETSIMPLE_H

#include "Trajet.h"

//----- Interfaces utilisées

//----- Constantes

//----- Types

//----- Rôle de la classe <TrajetSimple>
// Cette classe déclare un trajet défini par sa ville de départ, sa ville
// sa ville d'arrivée et le moyen de transport utilisé.

//----- PUBLIC

public:
//----- Méthodes publiques

```

```
void Afficher() const;
// Mode d'emploi :
// Cette fonction affiche les caractéristiques du trajet.

const char* GetTransport() const;
// Mode d'emploi :
// Cette fonction retourne le moyen de transport utilisé.

//----- Constructeurs
- destructeur
TrajetSimple ( const TrajetSimple & unTrajetSimple );
// Mode d'emploi (constructeur de copie) :

TrajetSimple ( );
// Mode d'emploi (constructeur 1) :

TrajetSimple ( char* depart, char* arrive, char*
transport );
// Mode d'emploi (constructeur 2) :

virtual ~TrajetSimple ( );
// Mode d'emploi (destructeur) :

//----- PRIVE

protected:
//----- Méthodes protégées

//----- Attributs protégés
char *transport;
};

//----- Autres définitions
dépendantes de <TrajetSimple>

#endif // TRAJETSIMPLE_H
```

IV.2.b Réalisation

```

/*
***** TrajetSimple - description *****

début          : $20/11/2019$
copyright      : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,
                  Zihao Hua$

e-mail         : $sophie.crowley@insa-lyon.fr$,
                  $zakaria.nassreddine@insa-lyon.fr$,
                  $zihao.hua@insa-lyon.fr$

***** */

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


//----- INCLUDE

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

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

//----- Constantes

//----- PUBLIC

//----- Méthodes publiques

void TrajetSimple::Afficher() const {
    cout << "Departing City:" << depart << "\n";
    cout << "Arrival City:" << arrive << "\n";
    cout << "Type of Trajet :" << "Simple" << "\n";
    cout << "Transportation :" << transport << "\n";
    cout << "\n";
}

```

```

//----- Constructeurs – destructeur

TrajetSimple::TrajetSimple ( const TrajetSimple & unTrajetSimple )
{
#ifdef MAP
    cout << "Appel au constructeur de copie de <TrajetSimple>" << endl;
#endif
} //---- Fin de TrajetSimple (constructeur de copie)

TrajetSimple::TrajetSimple ( )
{
#ifdef MAP
    cout << "Appel au constructeur de <TrajetSimple>" << endl;
#endif
} //---- Fin de TrajetSimple

TrajetSimple::TrajetSimple (char* departInput, char* arriveInput, char*
transportInput)
: Trajet(departInput, arriveInput)
{
#ifdef MAP
    cout << "Appel au constructeur 2 de <TrajetSimple>" << endl;
#endif
    transport = new char[strlen(arriveInput)+1];
    strcpy(transport, transportInput);
    transport[strlen(transportInput)]='\0';
} //---- Fin de TrajetSimple

TrajetSimple::~TrajetSimple ( )
// Algorithme :
//
{
#ifdef MAP
    cout << "Appel au destructeur de <TrajetSimple>" << endl;
#endif
    delete[] transport;
    delete[] depart;
    delete[] arrive;
} //---- Fin de ~TrajetSimple

```

IV.3 Trajet Complexe

IV.3.a Interface

```

/********************* TrajetComplexe - description
-----  

début          : $20/11/2019$  

copyright      : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,  

                  Zihao Hua$  

e-mail         : $sophie.crowley@insa-lyon.fr$  

                  $zakaria.nassreddine@insa-lyon.fr$  

                  $zihao.hua@insa-lyon.fr$  

*****  

//----- Interface de la classe <TrajetComplexe> (fichier TrajetComplexe.h)  

-----  

#if ! defined ( TRAJETCOMPLEXE_H )  

#define TRAJETCOMPLEXE_H  

#include "Trajet.h"  

#include "TrajetSimple.h"  

//----- Interfaces utilisées  

//----- Constantes  

//----- Types  

//-----  

// Rôle de la classe <TrajetComplexe>  

// Cette classe déclare un trajet composé de plusieurs trajets simples  

// allant chacun d'une ville de départ à une ville d'arrivée avec un moyen  

// de transport défini.  

//-----  

class TrajetComplexe : public Trajet  

{  

//----- PUBLIC  

public:  


```

```
//-----  
Méthodes publiques  
void Afficher() const;  
// Cette méthode affiche les caractéristique du trajet  
  
//----- Constructeurs  
- destructeur  
TrajetComplexe ( const TrajetComplexe &  
unTrajetComplexe );  
// Mode d'emploi (constructeur de copie) :  
  
TrajetComplexe ( );  
// Mode d'emploi (constructeur 1) :  
  
TrajetComplexe (TrajetSimple** elements, int nombre);  
// Mode d'emploi (constructeur 2) :  
  
virtual ~TrajetComplexe ( );  
// Mode d'emploi (destructeur) :  
  
//-----  
----- PRIVE  
  
protected:  
//-----  
Méthodes protégées  
  
//-----  
Attributs protégés  
TrajetSimple **elements;  
int nombreDeTrajets;  
};  
  
//----- Autres définitions  
dépendantes de <TrajetComplexe>  
  
#endif // TRAJETCOMPLEXE_H
```

IV.3.b Réalisation

```

***** *****
        TrajetComplexe - description
-----
début          : $20/11/2019$
copyright      : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,
                  Zihao Hua$
e-mail         : $sophie.crowley@insa-lyon.fr$
                  $zakaria.nassreddine@insa-lyon.fr$
                  $zihao.hua@insa-lyon.fr$

*****
//----- Réalisation de la classe <TrajetComplexe> (fichier
TrajetComplexe.cpp) -----
//----- INCLUDE
//----- Include système
#include <iostream>
#include <cstring>
using namespace std;
//----- Include personnel
#include "TrajetComplexe.h"
//----- Constantes
//----- PUBLIC
//----- Méthodes publiques
void TrajetComplexe::Afficher() const {
    cout << "Departing City:" << depart << "\n";
    cout << "Arrival City:" << arrive << "\n";
    cout << "Type of Trajet :" << "Complexe" << "\n";
    cout << "== List of Simple Trajets Inside :" << "\n";
    cout << endl;
    for(int i=0; i < nombreDeTrajets; i++) {
        elements[i]->Afficher();
    }
    cout << "== End of List" << "\n";
    cout << "\n";
}
//----- Constructeurs - destructeur
TrajetComplexe::TrajetComplexe ( const TrajetComplexe & unTrajetComplexe )
// Algorithme :

```

```

{
#ifdef MAP
    cout << "Appel au constructeur de copie de <TrajetComplexe>" << endl;
#endif
} //---- Fin de TrajetComplexe (constructeur de copie)
TrajetComplexe::TrajetComplexe ( )
// Algorithme :
{
#ifdef MAP
    cout << "Appel au constructeur de <TrajetComplexe>" << endl;
#endif
} //---- Fin de TrajetComplexe
TrajetComplexe::TrajetComplexe (TrajetSimple **elementInput, int nombre)
// Algorithme :
{
#ifdef MAP
    cout << "Appel au constructeur 2 de <TrajetComplexe>" << endl;
#endif
nombreDeTrajets = nombre;
elements= new TrajetSimple*[nombreDeTrajets];

for (int i=0; i<nombreDeTrajets; i++){
    elements[i]=elementInput[i];
}
depart = elements[0]->GetDepart();
arrive = elements[nombreDeTrajets-1]->GetArrive();
} //---- Fin de TrajetComplexe
TrajetComplexe::~TrajetComplexe ( )
// Algorithme :
{
#ifdef MAP
    cout << "Appel au destructeur de <TrajetComplexe>" << endl;
#endif
    for(int i=0;i<nombreDeTrajets;i++){
        delete elements[i];
    }
    delete[] elements;
} //---- Fin de ~TrajetComplexe

```

IV.4 Catalogue

IV.4.a Interface

```

/*
***** Catalogue - description *****

début          : $20/11/2019$
copyright      : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,
                  Zihao Hua$
e-mail         : $sophie.crowley@insa-lyon.fr$,
                  $zakaria.nassreddine@insa-lyon.fr$,
                  $zihao.hua@insa-lyon.fr$

*/
----- Interface de la classe <Catalogue> (fichier Catalogue.h)

----- Interfaces utilisées

----- Constantes

----- Types

// ----- Rôle de la classe <Catalogue>
// Cette classe a pour rôle de contenir un ensemble de trajets tout en
// offrant la possibilité d'en rajouter d'autres, de chercher un trajet
// souhaité et d'afficher le contenu à tout moment
// -----
// ----- PUBLIC
class Catalogue
{

```

```

public:
//----- Méthodes publiques
void Ajouter(Trajet* newTrajet);
// Mode d'emploi : Cette méthode permet d'ajouter un trajet au
// catalogue
void Afficher() const;
// Mode d'emploi : Permet d'afficher, à tout moment, le contenu du
// catalogue courant
void Rechercher(char* depart, char* arrive) const;
// Mode d'emploi : Cette méthode permet de chercher un trajet dont
// les villes de départ et d'arrivée sont rentrées en paramètres
// et d'ensuite l'afficher sur la sortie standard de la console.
bool Retirer(const int index);
// Mode d'emploi: cette méthode retire un trajet du catalogue dont
// l'indice est rentré en paramètre. On en a eu besoin pour le
// parcours récursif de la recherche avancée.
void RechercheAvance (const char* depart, const char* arrive);
int composition(const char* depart, const char* arrive, Catalogue* used);

//----- Constructeurs – destructeur
Catalogue ( const Catalogue & unCatalogue );
// Mode d'emploi (constructeur de copie) :
Catalogue();
// Mode d'emploi : (constructeur par défaut)
virtual ~Catalogue ( );
// Mode d'emploi : (destructeur de Catalogue)
Catalogue(Trajet** content, int nombre);
// Mode d'emploi : (destructeur de Catalogue)

//----- PRIVE
protected:
//----- Méthodes protégées
//----- Attributs protégés
int nombreDeTrajets;
Trajet **trajets;
int space;
};

//----- Autres définitions dépendantes de <Catalogue>

#endif // CATALOGUE_H

```

IV.4.b Réalisation

```

/*****
 * Catalogue - description
 -----
 début : $20/11/2019$
 copyright : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,
 Zihao Hua$

 e-mail : $sophie.crowley@insa-lyon.fr$
 $zakaria.nassreddine@insa-lyon.fr$
 $zihao.hua@insa-lyon.fr$

 ****/
//----- Réalisation de la classe <Catalogue> (fichier Catalogue.cpp)
//----- INCLUDE
//----- Include système
#include <iostream>
#include <cstring> //for more cross-platform compatibility
using namespace std;
//----- Include personnel
#include "Catalogue.h"
#include "TrajetSimple.h"
#include "TrajetComplexe.h"
//----- Constantes
const int DEFAULT_SPACE = 5;
//----- PUBLIC
//----- Méthodes publiques
// Cette méthode ajoute un trajet au catalogue.
void Catalogue::Ajouter(Trajet* newTrajet) {
    if(nombreDeTrajets==space){
        space+=DEFAULT_SPACE;
        Trajet** temp=new Trajet*[space];
        for (int i=0; i<nombreDeTrajets ; i++){
            temp[i]=trajets[i];
        }
        for (int i=0; i<nombreDeTrajets; i++){
            delete trajets[i];
        }
        delete[] trajets;
        trajets=temp;
    }
}

```

```

        trajets[nombreDeTrajets]=newTrajet;
        nombreDeTrajets++;
    }

/* Cette méthode permet d'afficher, à tout moment, le contenu du
catalogue courant */

void Catalogue::Afficher() const {
    if(nombreDeTrajets==0)
        cout << "Pas de trajet enregistre" << endl;
    for(int i=0; i < nombreDeTrajets;i++) {
        Trajet* curr = trajets[i];
        curr->Afficher();
    }
} //---- Fin de Méthode

// Cette méthode permet de rechercher un trajet dans le catalogue.

void Catalogue::Rechercher(char* depart, char* arrive) const {
    if(nombreDeTrajets==0)
        cout << "Pas de trajet enregistre" << endl;
    int count = 1;
    for(int i=0; i < nombreDeTrajets;i++) {
        if(strcmp(trajets[i]->GetDepart(),depart)==0
        && strcmp(trajets[i]->GetArrive(),arrive)==0) {
            cout << "Found Trajet " << count << "--\n";
            trajets[i]->Afficher();
            count++;
        }
    }
} //---- Fin de Méthode

bool Catalogue::Retirer(const int index) {
    if (index < 0 || index >= nombreDeTrajets){
        cout << "Pas de trajet enregistre, retrait impossible" << endl;
        return false;
    }
    for (int i=index; i<nombreDeTrajets-1; i++){
        trajets[i]=trajets[i+1];
    }
    nombreDeTrajets--;
    return true;
}

```

```

void Catalogue::RechercheAvance(const char* depart, const char* arrive) {
    cout << endl;
    Catalogue* used = new Catalogue();
    int nbPropositions = composition(depart, arrive, used);
    cout << "-- A total of: " << nbPropositions << " option(s) for you to choose
from " << endl;
    delete used;
}

int Catalogue::composition(const char* depart, const char* arrive, Catalogue*
used){
    int composable=0;
    if (strcmp(depart, arrive)==0){
        cout << "Composition(s) of trajets that you might want to consider:" <<
endl;
        used->Afficher();
        return 1;
    }

    for (int i=0; i<nombreDeTrajets; i++){
        Trajet* t=trajets[i];
        if (strcmp(t->GetDepart(), depart)==0){
            int valid=1;
            for (int j=0; j<used->nombreDeTrajets; j++){
                Trajet* tUsed=used->trajets[j];
                if (strcmp(t->GetArrive(), tUsed->GetDepart())==0){
                    valid=0;
                }
            }
            if(valid){
                used->Ajouter(t);
                composable += composition(t->GetArrive(), arrive, used);
                used->Retirer(used->nombreDeTrajets-1);
            }
        }
    }
    return composable;
}

```

```

//----- Constructeurs – destructeur

Catalogue::Catalogue ( const Catalogue & unCatalogue )
{
#ifdef MAP
    cout << "Appel au constructeur de copie de <Catalogue>" << endl;
#endif
} //---- Fin de Catalogue (constructeur de copie)

Catalogue::Catalogue()
{
#ifdef MAP
    cout << "Appel au constructeur de <Catalogue>" << endl;
#endif
    nombreDeTrajets = 0;
    space = DEFAULT_SPACE;
    trajets = new Trajet*[space];
} //---- Fin de Catalogue (Constructeur 1)

Catalogue::Catalogue(Trajet** content, int nombre) {
    space = nombre;
    nombreDeTrajets = nombre;
    trajets= new Trajet*[nombreDeTrajets];
    for(int i=0; i < nombreDeTrajets; i++) {
        trajets[i] = content[i];
    }
} //---- Fin de Catalogue (Constructeur 2)

Catalogue::~Catalogue ( )
{
#ifdef MAP
    cout << "Appel au destructeur de <Catalogue>" << endl;
#endif
    for (int i=0; i<nombreDeTrajets;i++){
        delete trajets[i];
    }
    delete[] trajets;
} //---- Fin de ~Catalogue
//----- PRIVE

//----- Méthodes protégées

```

IV.5 Classe main (interface) - Réalisation

```
*****
                main - description
-----
début          : $20/11/2019$
copyright      : (C) $2019$ par $Sophie Crowley, Zakaria Nassreddine,
                  Zihao Hua$
e-mail         : $sophie.crowley@insa-lyon.fr$,
                  $zakaria.nassreddine@insa-lyon.fr$,
                  $zihao.hua@insa-lyon.fr$

*****
#include <iostream>
#include <cstring>
using namespace std;

#include "Catalogue.h"
#include "Trajet.h"
#include "TrajetSimple.h"
#include "TrajetComplexe.h"

int main() {
    Catalogue *cat=new Catalogue;
    bool exit = false;
    int choice;
    while(!exit) {
        cout << "      Welcome to Trip Scanner" << endl << "What would you like
to do?" << endl;
        cout << "===== MENU =====" << endl;
        cout << "1: Enter 1 to add an itinerary." << endl;
        cout << "2: Enter 2 to display current catalogue. " << endl;
        cout << "3: Enter 3 to look for an itinerary." << endl;
        cout << "4: Enter 4 to make an advanced search." << endl;
        cout << "5: Enter 5 to exit." << endl;
        if(cin >> choice) {
            switch(choice) {
                case 1: {
                    char complexity;
```

```

char complexity;

printf("Enter 's' for simple or 'c' for complex trajet type:\n");
cin >> complexity;
if(complexity=='s') {
    char depart[30];
    char arrive[30];
    char transport[20];
    printf("Enter city of departure (replace spaces with '_'):\n");
    cin >> depart;
    printf("Enter city of arrival (replace spaces with '_'):\n");
    cin >> arrive;
    printf("Enter mode of transport:\n");
    cin >> transport;
    TrajetSimple* curr = new TrajetSimple(depart, arrive, transport);
    cat->Ajouter(curr);
    //delete curr;
}
else if(complexity=='c') {
    int nombre;
    printf("Enter the no. of Trajets in this Complex Trajet:\n");
    if(cin >> nombre) {
        char depart[30];
        char arrive[30];
        char transport[20];
        TrajetSimple** components = new TrajetSimple*[nombre];
        char stopover[30];
        for(int i=0; i < nombre; i++) {
            printf("Receiving info for trajet %d:\n", i+1);
            printf("Enter city of departure (replace spaces with '_'):\n");
            cin >> depart;
            if (i>0){
                while(strcmp(depart,stopover)!=0){
                    printf("Incoherent stopover city. Please enter
valid departure:\n");
                    cin >> depart;
                }
            }
            // checking that trajet i+1 starts where trajet i ended
    }
}

```

```

printf("Enter city of arrival (replace spaces
with '_'):\n");

        cin >> arrive;
        strcpy(stopover,arrive);
        printf("Enter mode of transport:\n");
        cin >> transport;
        TrajetSimple* curr = new TrajetSimple(depart,
arrive, transport);
        components[i] = curr;
    }

    TrajetComplexe* complexCurr = new
TrajetComplexe(components, nombre);
    cat->Ajouter(complexCurr);
    delete[] components;
}

else {
    cout << "Invalid input" << endl;
    cin.clear();
    cin.ignore(80, '\n');
}
}

else {
    cin.clear();
    printf("Invalid complexity type\n");
}
break;
}

case 2: {
    cat->Afficher();
    break;
}

case 3: {
    char depart[30];
    char arrive[30];
    printf("Enter city of departure (replace spaces with '_'):
\n");
    cin >> depart;
    printf("Enter city of arrival (replace spaces with '_')\n");
}

```

```
        cin >> arrive;
        cat->Rechercher(depart, arrive);
        break;
    }
    case 4: {
        char depart [30];
        char arrive [30];
        printf("Enter city of departure (replace
spaces with '_':\n");
        cin >> depart;
        printf("Enter city of arrival (replace
spaces with '_')\n");
        cin >> arrive;
        cat->RechercheAvance(depart, arrive);
        break;
    }
    case 5: {
        exit = true;
        break;
    }
    default: {
        printf("Invalid input\n");
        break;
    }
}
else {
    cout << "Invalid input" << endl;
    cin.clear();
    cin.ignore(80, '\n');
}
delete cat;
int debug;
cin >> debug; // pour suivre les fuites après sortie de
l'application par la commande leaks sur MacOS
return 0;
}
```

V. Problèmes et améliorations

V.1 Problèmes

V.1.a Compilation:

Nous avons rencontré des problèmes lors de la compilation car on ne maîtrisait pas tout à fait le principe de polymorphisme. La solution était bien évidemment d'appeler la classe Trajet en compilant TrajetSimple et TrajetComplexe . Erreur de débutant mais qui nous a pris de nombreuses heures à déboguer.

V.1.b Répartition des attributs des classes héritières

La répartition des caractéristiques des objets ne nous a pas été des plus évidentes. Par exemple, nous avons eu du mal à décider si l'attribut ‘transport’ devait être dans la classe mère Trajet ou dans TrajetSimple. Pour des raisons d’efficacité et afin d’avoir une classe mère la plus générique possible, nous l'avons mis comme attribut de la classe TrajetSimple.

V.1.c Méthodes virtuelles

Nous avons eu du mal à appeler les fonctions virtuelles des classes TrajetSimple et TrajetComplexe vu que nous étions légèrement confus par rapport à la vraie définition de cette notion. Le problème était dû au fait que la saisie de l’utilisateur était ajoutée directement comme Trajet au catalogue, ce qui faisait appel à la méthode Afficher de la classe Trajet qui, elle, ne fait rien. Pour y remédier, on a passé tous les trajets dans la classe main comme pointeurs ce qui leur permettait de garder leurs natures propres aux objets hétérogènes.

V.1.d Accès aux attributs

Nous ne pouvions pas accéder aux attributs départ, arrive vu qu’ils étaient protégés dans la classe Trajet, de même pour l’attribut transports de la classe TrajetSimple. On a donc dû implémenter des Getters.

V.1.e Affichage des trajets composés

On a hésité entre l'affichage de tous les trajets simples constituant un trajet composé et la combinaison de tous leurs moyens de transport en n'affichant que leurs villes de départ et d'arrivée ce qui aurait nécessité un double pointeur **transport. Au final, on a opté pour la solution qui nous semble la plus simple et qui permet de factoriser le code, celle d'appeler la méthode Afficher de TrajetSimple pour chaque trajet élémentaire de TrajetComplexe.

V.1.f Constructeur de TrajetComplexe

On s'est rendu compte qu'on ne pouvait pas juste affecter le pointeur tableau d'éléments saisis par l'utilisateur à la liste de trajets composant un trajet composé. Cela engendrait des fuites de mémoires et des problèmes d'allocation. La solution a donc été d'ajouter chaque trajet élémentaire individuellement via une boucle for.

V.1.g Libération de la mémoire

La partie la plus frustrante a certainement été la vérification des fuites de mémoires. Avec des résultats sur nos deux Macs avec la commande leaks d’OSX qui étaient complètement différents de ce qu’obtenait le troisième membre du binôme sous Linux avec Valgrind, on a eu beaucoup de mal à suivre et réparer ce qui n’allait pas. Quand ce n’était pas des fuites, on avait des erreurs de ‘Segmentation fault’. Il s’est avéré que c’était l’appel au destructeur de TrajetSimple qui était à l’origine de beaucoup d’erreurs.

V.1.h Création d'un catalogue

On a dû ajouter des variables imposant une taille pour le catalogue à l'instantiation (space et DEFAULT_SPACE) parce que sinon à l'appel du constructeur par défaut, la taille n'est jamais affectée ce qui provoquait des problèmes lors de l'ajout de trajets.

V.1.i Erreurs au niveau de l'interface, de la saisie

Plein de petites erreurs à la saisie sur l’entrée standard quand les données entrées au clavier n’étaient pas parfaitement formatées. On n’y a pas consacré énormément de temps et on a juste préconisé des formats de saisie à l'affichage vu que le TP porte essentiellement sur l'héritage et que l'interface n'est pas primordiale.

V2 Améliorations

V2.a Contraintes sur les ajouts et les saisies

Il aurait été judicieux de contrôler l'ajout de trajets, notamment en éliminant les doublons. C'est une perspective que l'on a abandonnée pour plus de simplicité, notamment au niveau de la gestion dynamique de la mémoire, d'autant plus que ce n'est pas précisé au niveau du cahier des charges.

Nous aurions pu également prendre garde à ce que la saisie soit mieux gérée via des contraintes dans le code source de l'application au lieu de se contenter de faire confiance au respect des consignes de saisie par l'utilisateur qui cause, dans de nombreux cas, des erreurs de segmentations.

V2.b Amélioration de l'interface graphique

Il demeure évident que l'interface de notre application n'est pas des plus soignées. L'expérience utilisateur n'est donc très clairement pas idéale. Il reste énormément de marge d'amélioration à ce niveau. Mais, une fois encore, comme ce n'est pas le but principal de l'exercice, on a fait en sorte que ça marche sans trop d'attention aux détails.

V2.c Structures de données

On est également conscient que notre choix de structures de données n'est pas forcément le meilleur qui se prête à ce cas de figure. En ayant discuté avec nos camarades ayant opté pour d'autres types de structures, le choix d'une liste chaînée pour stocker les trajets aurait peut-être été plus efficace notamment au niveau de l'ajout et de la suppression de trajets, quoique moins optimal pour la recherche, surtout en augmentation le nombre de trajets.