Auteurs : Frolin Balthazar – Wallyn Valentin

Compte-Rendu TP IFA-3-POO1-2

Application « Voyage Voyage »

Table des matières

I. Contexte de l'application

Ce compte-rendu détaille la réalisation du TP POO1-2 en classe de 3IFA INSA de Lyon. Ce TP s'inscrit dans l'initiation des notions abordées en cours : gestion de la mémoire et héritage en C++.

Dans ce cadre, l'application se propose de construire un catalogue de trajets et de proposer des parcours pour un voyage défini par une ville de départ et une ville d'arrivée.

Elle doit notamment respecter le cahier des charges suivant :

- Construction du catalogue : ajout de trajets au catalogue courant ; pour réaliser cette tâche, il faut être capable de saisir de nouveaux trajets qui pourront être simples ou composés.
- Affichage du catalogue courant : à tout instant, il faut être en mesure d'afficher le catalogue courant (affichage d'objets hétérogènes – trajets simples ou composés).
- La recherche de parcours dans le catalogue courant : pour un voyage donné, défini par une ville de départ et d'arrivée, il faut retrouver dans le catalogue courant tous les parcours qui peuvent répondre à la demande. Cette recherche de parcours s'effectue de 2 manières.
    1. Recherche simple : recherche uniquement les parcours constitués d'un seul trajet simple ou composé.
    2. Recherche avancée : recherche les parcours constitués d'un ou plusieurs trajets, simples ou composés, par composition des trajets disponibles dans le catalogue.

II. Description détaillée des classes

Une vision globale de l'application peut se faire à travers le diagramme de classes suivant. (Note : tous les membres publics y sont renseignés. En revanche, le diagramme ne présente pas de manière exhaustive les membres privés/protégés par soucis de concision.)

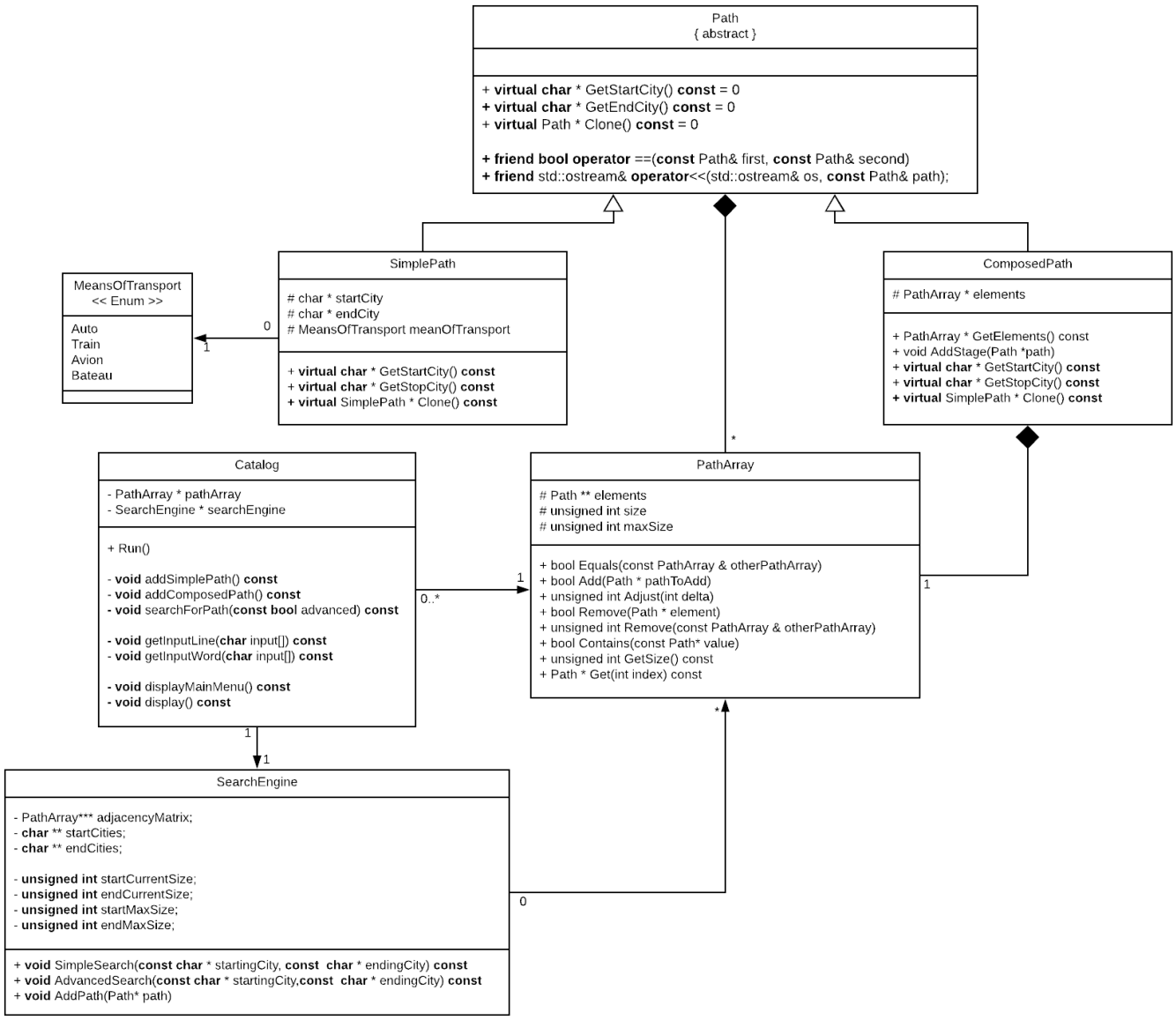

Figure 1 – Diagramme de classes de l'application VoyageVoyage

Le point d'entrée de l'application se fait au niveau de la méthode « Run() » de la classe Catalog. Cette classe gère l'interface utilisateur basée sur un menu console. En ce sens, elle possède un ensemble de méthodes permettant de gérer et de formater les entrées utilisateurs ainsi qu'un ensemble de méthodes liées à l'affichage du menu de l'application. Elle encapsule un « PathArray » qui stocke tous les trajets renseignés et un « SearchEngine » permettant d'effectuer des recherches de parcours. Les différents choix utilisateurs provoqueront les différentes manipulations possibles du catalogue courant : ajout d'un trajet, affichage du catalogue, recherche de parcours.

La gestion des trajets est inspirée d'un pattern Composite. En effet, on peut remarquer qu'un trajet composé peut lui-même être composé d'un trajet composé. De plus, la manipulation dans le PathArray devant être transparente entre trajet simple et composé, cette architecture basée sur une classe abstraite Path et deux classes concrètes SimplePath et ComposedPath représentant respectivement un trajet simple et un trajet composé s'est naturellement imposée. Les attributs « startCity » et « endCity » sont placés dans la classe SimplePath plutôt que la classe mère Path. Dans le cas contraire, comme ces informations peuvent être retrouvées dans les différents trajets du trajet composé, cela aurait causé une redondance de ces informations pour les instances de ComposedPath. De manière similaire, l'attribut représentant le moyen de transport n'a de sens que dans un trajet simple, les trajets composés pouvant être définis par plusieurs trajets aux moyens de transport différents.

La classe SearchEngine s'occupe de la recherche, notamment en implémentant deux algorithmes pour la recherche « simple » et la recherche « avancée ». Son rôle étant clairement défini et l'algorithme de recherche relativement lourd, c'est une entité propre détachée du catalogue (à l'inverse par exemple, de l'affichage).

La classe PathArray représente une collection de trajets manipulée en interne sous la forme d'un tableau dynamique. Les détails de l'implémentation peuvent se trouver dans la partie suivante.

III. Description de la structure de données employée

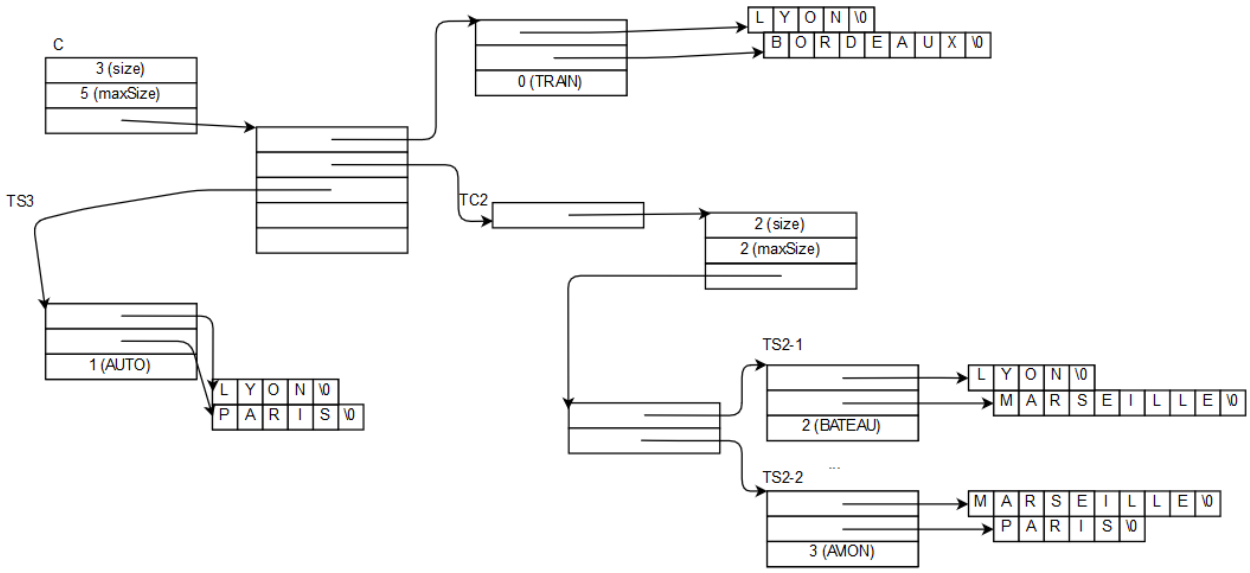

Figure 2 – Dessin d'abstraction de la mémoire lors de l'utilisation de PathArray avec le jeu d'essai du sujet

Le PathArray s'inspire de la classe « Ensemble » élaborée lors du TP-3IFA-POO1-1. Il se base donc sur un tableau dynamique de pointeur sur trajet (Path *). Ce tableau s'agrandit dynamique en doublant sa capacité maximale lorsqu'il est plein. L'ensemble des manipulations s'effectue à l'aide des variables size et maxSize représentant respectivement la taille courante et la taille maximale du tableau.

Il ne s'occupe pas de la création des objets ie. la collection se remplit de pointeurs sur trajet existants. En revanche, à sa destruction le PathArray s'occupe de la destruction des trajets pointés. Cela s'explique dans le cadre de l'application car les trajets sont créés et ajoutés « à la volée », mais est discutable dans le cadre d'une classe se voulant plus générique. Ce point est repris de manière plus générale dans les améliorations.

IV. Listing des classes

IV.1. Catalog

IV.1.a. Fichier d'en-tête (Catalog.h)

```
1.  /*************************************************************************
2.              Catalog  -  Affichage & Manipulation des Trajets
3.                              -------------------
4.      début                : 27/09/2018
5.      copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
6.      e-mail               : ...@insa-lyon.fr
7.  *************************************************************************/
8.
9.  //---------- Interface de la classe <Catalog> (fichier Catalog.h) ----------------
10. #if ! defined ( CATALOG_H )
11. #define CATALOG_H
12.
13. //--------------------------------------------------- Interfaces utilisées
14.
15. #include <iostream>
16.
17. #include "../PathArray/PathArray.h"
18. #include "../SearchEngine/SearchEngine.h"
19. //------------------------------------------------------------------ Constantes
20.
21. //------------------------------------------------------------------- Types
22.
23. //-----------------------------------------------------------------------------
24. // Rôle de la classe <Catalog>
25. //
26. // La classe Catalog permet d'ajouter, d'afficher et de chercher des trajets
27. // via la console.
28. //
29. //-----------------------------------------------------------------------------
30. class Catalog
31. {
32. //------------------------------------------------------------------ PUBLIC
33.
34. public:
35. //----------------------------------------------------- Méthodes publiques
36.
37.     void Run();
38.     // Mode d'emploi :
39.     // Affiche des différentes commandes disponible, attend les entrées utilisa-
    teurs et appelle les fonctions associées.
40.     //
41.     // Contrat :
42.     //
43.
44. //------------------------------------------------ Constructeur & Destructeur
45.
46.     Catalog();
47.     // Mode d'emploi :
48.     // Simple constructeur qui alloue une pathArray & un searchEngine
49.     //
50.     // Contrat :
51.     //
52.
53.     virtual ~Catalog ( );
54.     // Mode d'emploi :
55.     // Simple destructeur qui désalloue pathArray & searchEngine
56.     //
57.     // Contrat :
58.     //
59.
```

```
1.  //------------------------------------------------------------------ PRIVE
2.
3.  private:
4.  //----------------------------------------------------- Méthodes protégées
5.
6.      void addSimplePath() const;
7.      // Mode d'emploi :
8.      // Demande des informations et ajoute un SimplePath à la collection
9.      //
10.     // Contrat :
11.     //
12.     void addComposedPath() const;
13.     // Mode d'emploi :
14.     // Demande des informations et ajoute un ComposedPath à la collection
15.     //
16.     // Contrat :
17.     //
18.     void addPathAndNotifyUser(Path * path) const;
19.     // Mode d'emploi :
20.     // Essaie d'ajouter un Path à la collection et informe l'utilisateur
21.     // Soit le Path est ajouté soit il existe déjà.
22.     //
23.     // Contrat :
24.     //
25.
26.     void searchForPath(const bool advanced) const;
27.     // Mode d'emploi :
28.     // Demande la ville de départ et d'arrivée et lance une recherche de trajet
29.     //
30.     //      > advanced : true pour effectuer une recherche avancée, false pour
31.     // une recherche simple
32.     //
33.     // Contrat :
34.     //
35.
36.     void askForStartingCity(char *startingCity) const;
37.     // Mode d'emploi :
38.     // Demande une chaine de caractères à l'utilisateur pour le champ startCity
39.     //
40.     // Contrat :
41.     //
42.     void askForEndingCity(char *endingCity) const;
43.     // Mode d'emploi :
44.     // Demande une chaine de caractères à l'utilisateur pour le champ endCity
45.     //
46.     // Contrat :
47.     //
48.     unsigned int askForStageQty() const;
49.     // Mode d'emploi :
50.     // Demande un entier positif pour créer un trajet composé de n trajets simples.
51.     //
52.     // Contrat :
53.     //
54.
55.     MeansOfTransport displayAndAskForMeansOfTransport() const;
56.     // Mode d'emploi :
57.     // Affiche les differents moyens de transport et attend que l'utilisateur en se-
    lectionne un.
58.     //
59.     // Contrat :
```

```
1.
        bool getInputLine(char input[]) const;
2.      // Mode d'emploi :
3.      // Récupère une entrée sur le flux standard jusqu'à rencontrer un retour à la ligne
4.      //
5.      // Contrat :
6.      //
7.      bool getInputWord(char input[]) const;
8.      // Mode d'emploi :
9.      // Récupère une entrée sur le flux standard jusqu'a rencontrer un espace ou un re-
    tour à la ligne
10.     //
11.     // Contrat :
12.     //
13.
14.     void cleanInputStream() const;
15.     // Mode d'emploi :
16.     // Efface les données actuelles de l'entrée standard
17.     //
18.     // Contrat :
19.     //
20.     void capitalizeFirstWordsLetter(char *input) const;
21.     // Mode d'emploi :
22.     // Met en majuscule toutes les premières lettres de mots
23.     //
24.     // Contrat :
25.     //
26.     void inputError() const;
27.     // Mode d'emploi :
28.     // Affiche un message d'erreur et quitte
29.     //
30.     // Contrat :
31.     //
32.
33.     void displayMainMenu() const;
34.     // Mode d'emploi :
35.     // Affiche le menu principal.
36.     //
37.     // Contrat :
38.     //
39.     void display() const;
40.     // Mode d'emploi :
41.     // Affiche le catalogue.
42.     //
43.     // Contrat :
44.     //
45.     void displayMeansOfTransport() const;
46.     // Mode d'emploi :
47.     // Affiche les différents moyens de transport.
48.     //
49.     // Contrat :
50.     //
51.
52. //------------------------------------------------------ Attributs protégés
53.
54.     PathArray * pathArray;
55.     SearchEngine * searchEngine;
56.
57. };
58.
```

IV.1.b. Réalisation (Catalog.cpp)

```
/*************************************************************************
                Catalog  -  Affichage & Manipulation des Trajets
                             -------------------
    début                : 27/09/2018
    copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
    e-mail               : ...@insa-lyon.fr
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
#include <iostream>
#include <cstring>

//------------------------------------------------------ Include personnel
#include "Catalog.h"
#include "../SimplePath/SimplePath.h"
#include "../ComposedPath/ComposedPath.h"

using std::cout;
using std::endl;
//------------------------------------------------------------------ Constantes
const int INPUT_MAX_SIZE = 100;
const char SEPARATOR[] = "================================================\r\n";
//------------------------------------------------------------------ PUBLIC

//----------------------------------------------------- Méthodes publiques

void Catalog::Run()
{
    cout << "Bienvenue sur VoyageVoyage, L'app qui vous fera voyager !" << endl;
    displayMainMenu();

    char input[INPUT_MAX_SIZE];
    getInputWord(input);

    while (strcmp(input,"9") != 0 && strcmp(input,"Quit") != 0 && strcmp(input,"Quitter") != 0 && strcmp(input,"Q") != 0)
    {
        if (strcmp(input, "0") == 0)
        {
            displayMainMenu();
        }
        else if (strcmp(input,"1") == 0)
        {
            display();
        }
        else if (strcmp(input, "2") == 0)
        {
            addSimplePath();
        }
        else if (strcmp(input, "3") == 0)
        {
            addComposedPath();
```

```
}
        else if (strcmp(input, "4") == 0)
        {
            //recherche simple
            searchForPath(false);
        }
        else if (strcmp(input, "5") == 0)
        {
            //recherche avancée
            searchForPath(true);
        }
        else
        {
            displayMainMenu();
        }
        cout << "Entrez une nouvelle commande (0 pour afficher le menu) : " << endl;
        getInputWord(input);
    }

    cout << "Merci d'avoir utilisé VoyageVoyage ! à Bientot !" << endl;
}

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

Catalog::Catalog ( )
{
    pathArray = new PathArray();
    searchEngine = new SearchEngine();

#ifdef MAP
    cout << "Appel au constructeur de <Catalog>" << endl;
#endif
} //----- Fin de Catalog


Catalog::~Catalog ( )
{
    delete pathArray;
    delete searchEngine;

#ifdef MAP
    cout << "Appel au destructeur de <Catalog>" << endl;
#endif
} //----- Fin de ~Catalog

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

//------------------------------------------------- Méthodes protégées

/* Edition Methods */

void Catalog::addSimplePath() const
{
    char startingCity[INPUT_MAX_SIZE];
    char endingCity[INPUT_MAX_SIZE];
    MeansOfTransport meansOfTransport;

    cout << SEPARATOR;
    cout << "Ajout d'un Trajet Simple..." << endl;
```

```
1.  askForStartingCity(startingCity);
2.      askForEndingCity(endingCity);
3.      meansOfTransport = displayAndAskForMeansOfTransport();
4.
5.      addPathAndNotifyUser(new SimplePath(startingCity, endingCity, meansOfTransport));
6.  }
7.  void Catalog::addComposedPath() const
8.  {
9.      ComposedPath *composedPath = new ComposedPath();
10.     char startingCity[INPUT_MAX_SIZE];
11.     char endingCity[INPUT_MAX_SIZE];
12.     MeansOfTransport meansOfTransport;
13.
14.     cout << SEPARATOR;
15.     cout << "Ajout d'un Trajet Composé..." << endl;
16.
17.     int stageQty = askForStageQty();
18.
19.     for (int i = 1; i <= stageQty; i++)
20.     {
21.         cout << "Etape " << i << endl;
22.
23.         if (i > 1)
24.         {
25.             cout << "\tVille de départ : ";
26.             strcpy(startingCity, endingCity);
27.             cout << startingCity << endl;
28.         }
29.         else
30.         {
31.             askForStartingCity(startingCity);
32.         }
33.
34.         askForEndingCity(endingCity);
35.         meansOfTransport = displayAndAskForMeansOfTransport();
36.
37.         composedPath->AddStage(new SimplePath(startingCity, end-
    ingCity, meansOfTransport));
38.     }
39.
40.     addPathAndNotifyUser(composedPath);
41. }
42.
43. void Catalog::addPathAndNotifyUser(Path * path) const
44. {
45.     if (pathArray->Add(path))
46.     {
47.         searchEngine->AddPath(path);
48.         cout << "Votre Trajet à bien été ajouté !" << endl;
49.     }
50.     else
51.     {
52.         cout << "Ce Trajet existe déjà !" << endl;
53.     }
54.
55.     cout << SEPARATOR;
56. }
57.
```

```
1.  /* Search Methods */
2.  void Catalog::searchForPath(const bool advanced) const
3.  {
4.      char startingCity[INPUT_MAX_SIZE];
5.      char endingCity[INPUT_MAX_SIZE];
6.
7.      cout << SEPARATOR;
8.      if (advanced)
9.      {
10.         cout << "Recherche d'un trajet (Version avancée)..." << endl;
11.     }
12.     else
13.     {
14.         cout << "Recherche d'un trajet (Version simple)..." << endl;
15.     }
16.
17.     askForStartingCity(startingCity);
18.
19.     do
20.     {
21.         cout << "\tVille d'arrivée : ";
22.     }
23.     while (!getInputLine(endingCity));
24.
25.
26.     cout << endl << "Trajet(s) trouvé(s) :" << endl << endl;
27.     if (advanced)
28.     {
29.         searchEngine->AdvancedSearch(startingCity, endingCity);
30.     }
31.     else
32.     {
33.         searchEngine->SimpleSearch(startingCity, endingCity);
34.     }
35.
36.     cout << SEPARATOR;
37. }
38.
39. /* Input Methods */
40.
41.
42. void Catalog::askForStartingCity(char *startingCity) const
43. {
44.     do
45.     {
46.         cout << "\tVille de départ : ";
47.     }
48.     while (!getInputLine(startingCity));
49. }
50. void Catalog::askForEndingCity(char *endingCity) const
51. {
52.     do
53.     {
54.         cout << "\tVille d'arrivé : ";
55.     }
56.     while (!getInputLine(endingCity));
57. }
```

```
1.  unsigned int Catalog::askForStageQty() const
2.  {
3.      unsigned int stageQty;
4.
5.      do
6.      {
7.          cout << "Nombre d'étapes du trajet (entre 2 et 10) : ";
8.          scanf("%ul", &stageQty);
9.          cleanInputStream();
10.     }
11.     while (stageQty <= 1 || stageQty > 10);
12.
13.     return stageQty;
14. }
15.
16. MeansOfTransport Catalog::displayAndAskForMeansOfTransport() const
17. {
18.     char transport[MEAN_OF_TRANSPORT_STRING_MAX_SIZE];
19.
20.     for ( ; ; )
21.     {
22.         do
23.         {
24.             displayMeansOfTransport();
25.         }
26.         while (!getInputLine(transport));
27.
28.         for (int i = 0; i < MEAN_OF_TRANSPORT_QTY; i++)
29.         {
30.             if (strcmp(transport, MEAN_OF_TRANSPORT_STRINGS[i]) == 0)
31.                 return (MeansOfTransport) i;
32.         }
33.
34.         cout << "Moyen de transport non valide...\n";
35.     }
36. }
37.
38. bool Catalog::getInputLine(char *input) const
39. {
40.     if (fscanf(stdin, "%99[^\n]", input) != 1)
41.     {
42.         cleanInputStream();
43.         return false;
44.     }
45.
46.     cleanInputStream();
47.     capitalizeFirstWordsLetter(input);
48.
49.     return true;
50. }
51. bool Catalog::getInputWord(char *input) const
52. {
53.     if (fscanf(stdin, "%99s", input) != 1)
54.     {
55.         inputError();
56.     }
57.
58.     cleanInputStream();
59.     capitalizeFirstWordsLetter(input);
60.   }
```

```
1.  void Catalog::cleanInputStream() const
2.  {
3.      int c = 0;
4.      while ((c = getchar()) != '\n' && c != EOF);
5.  }
6.  void Catalog::capitalizeFirstWordsLetter(char *input) const
7.  // Algorithme :
8.  // Change la première lettre et toutes les lettres après un espace par des majus-
    cule (si ce sont des lettres).
9.  // Change les autres lettres en majuscule en lettre minuscule.
10. {
11.     for (int i = 0; input[i] != '\0'; i++)
12.     {
13.         if (i == 0) // First Letter
14.         {
15.             if (input[i] >= 'a' && input[i] <= 'z')
16.                 input[i] -= 32;
17.
18.             continue;
19.         }
20.         if(input[i]==' ') // Check Space
21.         {
22.             i++;
23.
24.             if(input[i] >= 'a' && input[i] <= 'z')
25.             {
26.                 input[i] -= 32;
27.                 continue;
28.             }
29.         }
30.         else
31.         {
32.             if(input[i] >= 'A' && input[i] <= 'Z')
33.                 input[i] += 32;
34.         }
35.     }
36. }
37.
38. void Catalog::inputError() const
39. {
40.     cout << "input Error\n";
41.     exit(1);
42. }
43.
44. /* Output Methods */
45.
46. void Catalog::displayMainMenu() const
47. {
48.     cout    <<                                                  endl
49.             << SEPARATOR
50.             << "Taper 0 pour Consulter ce menu."                << endl
51.             << "Taper 1 pour Consulter tous les trajets."       << endl
52.             <<                                                  endl
53.             << "Taper 2 pour Ajouter un trajet simple."         << endl
54.             << "Taper 3 pour Ajouter un trajet composé."        << endl
55.             <<                                                  endl
```

```
   << "Taper 4 pour chercher un trajet (version simple)."      << endl
              << "Taper 5 pour chercher un trajet (version avancée)."     << endl
              <<                                                             endl
              << "Taper 9 pour quitter l'application \"Voyage Voyage\""   << endl
              << SEPARATOR
              <<                                                             endl;
}

void Catalog::display() const
{
    cout << SEPARATOR;
    cout << "Liste des Voyages Disponibles..."                      << endl;
    cout                                                            << endl;

    for (unsigned int i = 0; i < pathArray->GetSize(); i++)
    {
        cout << "#" << i + 1 << " " << *pathArray->Get(i) << endl;
    }

    cout << SEPARATOR;
}

void Catalog::displayMeansOfTransport() const
{
    cout << "\tMoyen de transport ( ";

    for (int i = 0; i < MEAN_OF_TRANSPORT_QTY; i++) {
        cout << MEAN_OF_TRANSPORT_STRINGS[i] << " ";
    }

    cout << ") : ";
}
```

IV.2. Path

IV.2.a. Fichier d'en-tête (Path.h)

```
1.  /*************************************************************************
2.                              Path  -  Trajet (abstrait)
3.                                -------------------
4.      début                : 27/09/2018
5.      copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
6.      e-mail               : ...@insa-lyon.fr
7.  *************************************************************************/
8.
9.  //---------- Interface de la classe <Path> (fichier Path.h) ----------------
10. #if ! defined ( PATH_H )
11. #define PATH_H
12.
13. //------------------------------------------------- Interfaces utilisées
14.
15. #include <iostream>
16.
17. //------------------------------------------------------------------ Types
18. // Note : Nous avons inversé les catégories types et constantes pour pou-
    voir faire les déclarations
19.
20. enum MeansOfTransport { TRAIN, AUTO, BATEAU, AVION, END_DELIMITER };
21.
22. //------------------------------------------------------------ Constantes
23.
24. const int MEAN_OF_TRANSPORT_STRING_MAX_SIZE = 15;
25. const int MEAN_OF_TRANSPORT_QTY = END_DELIMITER;
26.
27. const char MEAN_OF_TRANSPORT_STRINGS[MEAN_OF_TRANSPORT_QTY][MEAN_OF_TRANSPORT_STRING_MAX_
    SIZE] = { "Train", "Auto", "Bateau", "Avion"};
28.
29. //------------------------------------------------------------------------
30. // Rôle de la classe <Path>
31. //
32. // Classe abstraite représentant un trajet entre 2 villes par un certain
33. // moyen de transport. Permet de manipuler les trajets sans distinction de
34. // types
35. //------------------------------------------------------------------------
36.
37. class Path
38. {
39. //----------------------------------------------------------------- PUBLIC
40.
41. public:
42. //---------------------------------------------------- Méthodes publiques
43.
44.     virtual char * GetStartCity() const = 0;
45.     // Mode d'emploi :
46.     // Récupère le nom de la ville de départ
47.     // Voir les classes dérivées pour la réalisation concrète
48.
49.     virtual char * GetEndCity() const = 0;
50.     // Mode d'emploi :
51.     // Récupère le nom de la ville d'arrivée
52.     // Voir les classes dérivées pour la réalisation concrète
53.
54.     virtual Path * Clone() const = 0;
55.     // Mode d'emploi :
56.     // Dublique le trajet
57.     // Voir les classes dérivées pour la réalisation concrète
58.
```

```cpp
//---------------------------------------------- Surcharge d'opérateurs

    friend bool operator ==(const Path& first, const Path& second);
    // implémentation tirée de :
    // https://stackoverflow.com/questions/1691007/whats-the-right-way-to-overload-operator-for-a-class-hierarchy
    // https://stackoverflow.com/questions/9287704/is-there-an-idiomatic-approach-in-c-for-comparing-polymorphic-types-for-object?noredirect=1&lq=1

    friend std::ostream& operator<<(std::ostream& os, const Path& path);
    // Mode d'emploi :
    // Affiche les données formatées en utilisant la fonction print

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

protected:
    Path();
    // Mode d'emploi :
    // Constructeur par défaut (vide)

public:
    virtual ~Path();
    // Mode d'emploi :
    // Destructeur par défaut (vide)

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

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

    virtual std::ostream& print(std::ostream& os) const = 0;
    // Mode d'emploi :
    // Méthode permettant l'affichage de la classe Path sur un flux (os).
    // Voir les classes dérivées pour la réalisation concrète

//------------------------------------------------------- Attributs protégés

private:
//-------------------------------------------------------- Méthodes privées
    virtual bool equals(const Path& other) const = 0;

};

//-------------------------------- Autres définitions dépendantes de <Path>
```

IV.2.b. Réalisation (Path.cpp)

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

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

//-------------------------------------------------------- Include système

#include <iostream>

using std::cout;
using std::endl;

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

#include "Path.h"

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

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

//----------------------------------------------------- Méthodes publiques

//------------------------------------------------ Surcharge d'opérateurs

bool operator==(const Path& first, const Path& second)
{
    // RTTI check
    if (typeid(first) != typeid(second))
        return false;

    // Invoke is_equal on derived types
    return first.equals(second);
} //----- Fin de ==

std::ostream & operator<<(std::ostream & os, const Path & path)
{
    return path.print(os);
} //----- Fin de <<

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

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

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

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

//----------------------------------------------------- Méthodes protégées
```

IV.3. SimplePath

IV.3.a. Fichier d'en-tête (SimplePath.h)

```
1.  /*************************************************************************
2.               SimplePath  -  Trajet Simple entre deux villes
3.                                 -------------------
4.      début                : 27/09/2018
5.      copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
6.      e-mail               : ...@insa-lyon.fr
7.  *************************************************************************/
8.
9.  //---------- Interface de la classe <SimplePath> (fichier SimplePath.h) ----------------
10. #if ! defined ( SIMPLEPATH_H )
11. #define SIMPLEPATH_H
12.
13. //--------------------------------------------------- Interfaces utilisées
14.
15. #include "../Path/Path.h"
16.
17. //------------------------------------------------------------- Constantes
18.
19. //------------------------------------------------------------------ Types
20.
21. //-------------------------------------------------------------------------
22. // Rôle de la classe <SimplePath>
23. // Représente un trajet simple entre une ville de départ et une ville
24. // d'arrivée par un unique moyen de transport.
25. //-------------------------------------------------------------------------
26.
27. class SimplePath : public Path
28. {
29. //----------------------------------------------------------------- PUBLIC
30.
31. public:
32. //---------------------------------------------------- Méthodes publiques
33.
34.     virtual char * GetStartCity() const;
35.     // Mode d'emploi :
36.     //      Getter de "startCity"
37.     //
38.     // Contrat :
39.     //
40.
41.     virtual char * GetEndCity() const;
42.     // Mode d'emploi :
43.     //      Getter de "endCity"
44.     //
45.     // Contrat :
46.     //
47.
48.     virtual SimplePath * Clone() const;
49.     // Mode d'emploi :
50.     //      Duplique le trajet
51.     //
52.     // Contrat :
53.     //
54.
```

```
1.  //------------------------------------------------ Surcharge d'opérateurs
2.
3.      SimplePath & operator=(SimplePath other);
4.
5.  //-------------------------------------------- Constructeurs - destructeur
6.      SimplePath(const SimplePath & other);
7.      // Mode d'emploi (constructeur de copie) :
8.      // Constructeur simple qui copie chaque attributs de "other" dans un nouveau objet
9.      //
10.     // Contrat :
11.     //
12.
13.     SimplePath(char * startingCity, char * endingCity, MeansOfTransport vehicle);
14.     // Mode d'emploi
15.     // Constructeur basique avec un parametre pour chaque attributs
16.     //
17.     // Contrat :
18.     //
19.
20.     virtual ~SimplePath();
21.     // Mode d'emploi
22.     // Destructeur basique qui désalloue startCity & endCity
23.     //
24.     // Contrat :
25.     //
26.
27. //---------------------------------------------------------------- PRIVE
28.
29. protected:
30. //------------------------------------------------------ Méthodes protégées
31.
32.     virtual std::ostream & print(std::ostream & os) const;
33.     // Mode d'emploi :
34.     //      Ecrit une représentation de l'objet en chaîne de caractères
35.     // sur un flux standard
36.     //      > os : flux standard sur lequel on écrit
37.     //
38.     // Contrat :
39.     //
40.
41.     virtual bool equals(const Path& other) const;
42.     // Mode d'emploi :
43.     //      Compare si le ComposedPath est égal à un autre trajet.
44.     // Renvoie true si les trajets sont égaux, faux sinon.
45.     //      > other : trajet à comparer avec l'instance actuelle
46.     // Contrat :
47.     //
48.
49.     friend void swap(SimplePath & first, SimplePath & second);
50.     // Mode d'emploi :
51.     //      Echange les valeurs des attributs entre 2 objets SimplePath
52.     //      > first : Premier objet de l'échange
53.     //      > second : Second objet de l'échange
54.     //
55.     // Contrat :
56.     //
57.
```

```
//------------------------------------------------- Attributs protégés

    char * startCity;
    char * endCity;

    MeansOfTransport meanOfTransport;

};

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

#endif // SIMPLEPATH_H
```

IV.3.b. Réalisation (SimplePath.cpp)

```
/*************************************************************************
             SimplePath  -  Trajet Simple entre deux villes
                             -------------------
    début                : 27/09/2018
    copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
    e-mail               : ...@insa-lyon.fr
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
#include <iostream>
#include <cstring>

using std::cout;
using std::endl;

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

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

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

//----------------------------------------------------- Méthodes publiques

char * SimplePath::GetStartCity() const
{
    return startCity;
} //----- Fin de StartFrom

char * SimplePath::GetEndCity() const
{
    return endCity;
} //----- Fin de StopAt
```

```
1.  SimplePath* SimplePath::Clone() const
2.  {
3.      return new SimplePath(*this);
4.  } //----- Fin de Clone
5.
6.  //------------------------------------------------- Surcharge d'opérateurs
7.
8.  SimplePath & SimplePath::operator=(SimplePath other)
9.  {
10.     swap(*this, other);
11.
12.     return *this;
13. } //----- Fin de =
14.
15. //-------------------------------------------- Constructeurs - destructeur
16.
17. SimplePath::SimplePath(const SimplePath & other)
18. {
19.     startCity = new char[strlen(other.startCity)+1];
20.     strcpy(startCity, other.startCity);
21.
22.     endCity = new char[strlen(other.endCity)+1];
23.     strcpy(endCity, other.endCity);
24.
25.     meanOfTransport = other.meanOfTransport;
26.
27. #ifdef MAP
28.     cout << "Appel au constructeur de copie de <SimplePath>" << endl;
29. #endif
30. } //----- Fin de SimplePath (constructeur de copie)
31.
32.
33. SimplePath::SimplePath(char * startingCity, char * endingCity, MeansOfTransport vehi-
    cle) : meanOfTransport(vehicle)
34. {
35.     startCity = new char[strlen(startingCity)+1];
36.     strcpy(startCity, startingCity);
37.
38.     endCity = new char[strlen(endingCity)+1];
39.     strcpy(endCity, endingCity);
40.
41. #ifdef MAP
42.     cout << "Appel au constructeur de <SimplePath>" << endl;
43. #endif
44. } //----- Fin de SimplePath
45.
46.
47. SimplePath::~SimplePath()
48. {
49.     delete[] startCity;
50.     delete[] endCity;
51.
52. #ifdef MAP
53.     cout << "Appel au destructeur de <SimplePath>" << endl;
54. #endif
55. } //----- Fin de ~SimplePath
```

```cpp
//------------------------------------------------------------------ PRIVE

//------------------------------------------------------ Méthodes protégées

bool SimplePath::equals(const Path& other) const
{
    const SimplePath& other_derived = dynamic_cast<const SimplePath&>(other);

    return (strcmp(startCity, other_derived.GetStartCity()) == 0)
        && (strcmp(endCity, other_derived.GetEndCity()) == 0)
        && meanOfTransport == other_derived.meanOfTransport;
} //----- Fin de equals

std::ostream& SimplePath::print(std::ostream& os) const
{
    os  << "Trajet Simple :"                                          << endl
        << "\tDépart    : " << startCity                              << endl
        << "\tArrivée   : " << endCity                                << endl
        << "\tTransport : " << MEAN_OF_TRANSPORT_STRINGS[meanOfTransport]   << endl;

    return os;
} //----- Fin de print

void swap(SimplePath& first, SimplePath& second)
{
    char * tmp = new char[strlen(first.GetStartCity()) + 1];
    strcpy(tmp, first.GetStartCity());

    delete [] first.GetStartCity();
    first.startCity = new char[strlen(second.GetStartCity()) + 1];
    strcpy(first.startCity, second.GetStartCity());

    delete [] second.GetStartCity();
    second.startCity = tmp;

    char * tmpE = new char[strlen(first.GetEndCity()) + 1];
    strcpy(tmpE, first.GetEndCity());

    delete [] first.GetEndCity();
    first.endCity = new char[strlen(second.GetEndCity()) + 1];
    strcpy(first.endCity, second.GetEndCity());

    delete [] second.GetEndCity();
    second.endCity = tmpE;

    MeansOfTransport tmpMot = second.meanOfTransport;
    second.meanOfTransport = first.meanOfTransport;
    first.meanOfTransport = tmpMot;
} //----- Fin de swap
```

IV.4. ComposedPath

IV.4.a. Fichier d'en-tête (ComposedPath.h)

```
1.  /*************************************************************************
2.                             ComposedPath  -  description
3.                               -------------------
4.      début                : 27/09/2018
5.      copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
6.      e-mail               : ...@insa-lyon.fr
7.  *************************************************************************/
8.
9.  //---------- Interface de la classe <ComposedPath> (fichier ComposedPath.h) ------------
    ---
10. #if ! defined ( COMPOSEDPATH_H )
11. #define COMPOSEDPATH_H
12.
13. //------------------------------------------------- Interfaces utilisées
14. #include "../Path/Path.h"
15. #include "../PathArray/PathArray.h"
16. //------------------------------------------------------------ Constantes
17.
18. //------------------------------------------------------------------ Types
19.
20. //--------------------------------------------------------------------------
21. // Rôle de la classe <ComposedPath>
22. // Représente un trajet composé de plusieurs sous-trajets (simples ou eux-
23. // mêmes composés)
24. //--------------------------------------------------------------------------
25.
26. class ComposedPath : public Path
27. {
28. //----------------------------------------------------------------- PUBLIC
29.
30. public:
31. //---------------------------------------------------- Méthodes publiques
32.     PathArray* GetElements ( ) const;
33.     // Mode d'emploi :
34.     //      Retourne la collection de Path composant le trajet
35.     // Contrat :
36.     //
37.
38.     void AddStage(Path *path) const;
39.     // Mode d'emploi :
40.     //      Ajoute une étape (trajet) au trajet composé
41.     //      > path : pointeur sur le trajet à ajouter
42.     // Contrat :
43.     //      path est un pointeur sur trajet valide
44.
45.     virtual char * GetStartCity() const;
46.     // Mode d'emploi :
47.     //      Getter de "startCity"
48.     // Contrat :
49.     //
50.
51.     virtual char * GetEndCity() const;
52.     // Mode d'emploi :
53.     //      Getter de "endCity"
54.     // Contrat :
55.     //
56.
57.     virtual ComposedPath* Clone() const;
58.     // Mode d'emploi :
59.     //      Duplique le trajet
```

```
1.  //------------------------------------------------ Surcharge d'opérateurs
2.      ComposedPath& operator=(ComposedPath other);
3.
4.
5.  //-------------------------------------------- Constructeurs - destructeur
6.      ComposedPath ( const ComposedPath & other );
7.      // Mode d'emploi (constructeur de copie) :
8.      //
9.      // Contrat :
10.     //
11.
12.     ComposedPath ( unsigned int maxSize = MAX_SIZE );
13.     // Mode d'emploi :
14.     //      > maxSize : taille maximum initiale de la collection de trajets,
15.     // par defaut égale à MAX_SIZE
16.     // Contrat :
17.     //
18.
19.     virtual ~ComposedPath ( );
20.     // Mode d'emploi :
21.     //
22.     // Contrat :
23.     //
24.
25. //-------------------------------------------------------------------- PRIVE
26.
27. protected:
28. //----------------------------------------------------- Méthodes protégées
29.     virtual std::ostream& print(std::ostream& os) const;
30.     // Mode d'emploi :
31.     //      Ecrit une représentation de l'objet en chaîne de caractères sur un
32.     // flux standard
33.     //      > os : flux standard sur lequel on écrit
34.     // Contrat :
35.     //
36.     virtual bool equals(const Path& other) const;
37.     // Mode d'emploi :
38.     //      Compare si le ComposedPath est égal à un autre trajet.
39.     // Renvoie true si les trajets sont égaux, faux sinon.
40.     // Contrat :
41.     //
42.     friend void swap(ComposedPath& first, ComposedPath& second);
43.     // Mode d'emploi :
44.     //      Echange les valeurs des attributs entre 2 objets ComposedPath
45.     //      > first : Premier objet de l'échange
46.     //      > second : Second objet de l'échange
47.     // Contrat :
48.     //
49. //------------------------------------------------------ Attributs protégés
50.
51.      PathArray* elements;
52.
53. };
54.
55. //------------------------------- Autres définitions dépendantes de <ComposedPath>
56.
57. #endif // COMPOSEDPATH_H
```

IV.4.b. Réalisation (ComposedPath.cpp)

```cpp
/*************************************************************************
                      ComposedPath  -  trajet composé
                             -------------------
    début                : 27/09/2018
    copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
    e-mail               : ...@insa-lyon.fr
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
#include <iostream>

using std::cout;
using std::endl;
//-------------------------------------------------------- Include personnel
#include "ComposedPath.h"
//------------------------------------------------------------- Constantes
//------------------------------------------------------------------ PUBLIC

//-------------------------------------------------------- Méthodes publiques
PathArray* ComposedPath::GetElements ( ) const
// Algorithme :
{
    return elements;
} //----- Fin de GetElements

void ComposedPath::AddStage(Path *path) const
// Algorithme :
{
    elements->Add(path);
} //----- Fin de AddStage

char * ComposedPath::GetStartCity() const
{
    return elements->Get(0)->GetStartCity();
} //----- Fin de StartFrom

char * ComposedPath::GetEndCity() const
{
    return elements->Get(elements->GetSize() - 1)->GetEndCity();
} //----- Fin de StopAt

ComposedPath* ComposedPath::Clone() const
{
    return new ComposedPath(*this);
} //----- Fin de Clone

//------------------------------------------------- Surcharge d'opérateurs
ComposedPath& ComposedPath::operator=(ComposedPath other)
{
    swap(*this, other);

    return *this;
} //----- Fin de =
```

```
1.  //----------------------------------------- Constructeurs - destructeur
2.  ComposedPath::ComposedPath ( const ComposedPath & other )
3.  // Algorithme :
4.  //
5.  {
6.  #ifdef MAP
7.      cout << "Appel au constructeur de copie de <ComposedPath>" << endl;
8.  #endif
9.      elements = new PathArray();
10.     *elements = *other.elements;
11. } //----- Fin de ComposedPath (constructeur de copie)
12.
13.
14. ComposedPath::ComposedPath ( unsigned int maxSize)
15. // Algorithme :
16. //
17. {
18. #ifdef MAP
19.     cout << "Appel au constructeur de <ComposedPath>" << endl;
20. #endif
21.     elements = new PathArray(maxSize);
22. } //----- Fin de ComposedPath
23.
24.
25. ComposedPath::~ComposedPath ( )
26. // Algorithme :
27. //
28. {
29. #ifdef MAP
30.     cout << "Appel au destructeur de <ComposedPath>" << endl;
31. #endif
32.     delete elements;
33. } //----- Fin de ~ComposedPath
34.
35.
36. //------------------------------------------------------------------ PRIVE
37.
38. //------------------------------------------------------ Méthodes protégées
39. bool ComposedPath::equals(const Path& other) const
40. {
41.     const ComposedPath& other_derived = dynamic_cast<const ComposedPath&>(other);
42.     return elements->Equals(*other_derived.elements);
43. } //----- Fin de equals
44.
45. std::ostream& ComposedPath::print(std::ostream& os) const
46. {
47.     os << "Trajet Composé :" << endl;
48.
49.     return elements->Print(os, false, true);
50. } //----- Fin de print
51.
52. void swap(ComposedPath& first, ComposedPath& second)
53. {
54.     PathArray tmp = *(first.elements);
55.     *(first.elements) = *(second.elements);
56.     *(second.elements) = tmp;
57. } //----- Fin de swap
```

IV.5. PathArray

IV.5.a. Fichier d'en-tête (PathArray.h)

```
1.  /*************************************************************************
2.                              PathArray  -  description
3.                                -------------------
4.      début                : 27/09/2018
5.      copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
6.      e-mail               : ...@insa-lyon.fr
7.  *************************************************************************/
8.
9.  //---------- Interface de la classe <PathArray> (fichier PathArray.h) ----------------
10. #if ! defined ( PATHARRAY_H )
11. #define PATHARRAY_H
12.
13. //------------------------------------------------- Interfaces utilisées
14. #include <iostream>
15. #include "../Path/Path.h"
16.
17. //------------------------------------------------------------------ Constantes
18. const int MAX_SIZE = 10;
19.
20. //------------------------------------------------------------------------ Types
21.
22. //-------------------------------------------------------------------------
23. // Rôle de la classe <PathArray>
24. // Collection permettant la gestion dynamique de trajets (Path)
25. //-------------------------------------------------------------------------
26.
27. class PathArray
28. {
29. //----------------------------------------------------------------- PUBLIC
30.
31. public:
32. //---------------------------------------------------- Méthodes publiques
33.     bool Equals(const PathArray & otherPathArray) const;
34.     // Mode d'emploi :
35.     //      Compare le contenu de la collection avec une autre collection PathArray.
36.     // Renvoie true si les collections sont égales, false sinon.
37.     //      > otherPathArray : collection à comparer
38.     // Contrat :
39.     //
40.     bool Add(Path* pathToAdd);
41.     // Mode d'emploi :
42.     //      Ajoute un trajet à la collection
43.     // Renvoie false si un trajet similaire est déjà présent dans la collection,
44.     // true sinon.
45.     //      > pathToAdd : trajet à ajouter
46.     // Contrat :
47.     //      pathToAdd pointe vers un objet Path valide
48.     unsigned int Adjust(int delta);
49.     // Mode d'emploi :
50.     //      Ajuste la taille de la collection.
51.     //      > delta : offset de la taille de la collec-
    tion. Si delta > 0, la taille de la
52.     // collection s'agrandit de delta. Sinon, la taille de la collection se ré-
    duit de la valeur
53.     // de delta, dans la limite du nombre d'éléments déjà présent.
54.     // Contrat :
55.     //
```

```
1.  bool Remove(Path* element);
2.      // Mode d'emploi :
3.      //      Retire un trajet de la collection
4.      // Renvoie true si l'objet a pu être supprimé, false sinon
5.      //      > element : trajet à supprimer
6.      // Contrat :
7.      //      element pointe vers un objet Path valide
8.      unsigned int Remove(const PathArray & otherPathArray);
9.      // Mode d'emploi :
10.     //      Retire un ensemble de trajets de la collection
11.     // Renvoie le nombre d'éléments effectivement supprimés.
12.     //      > otherPathArray : ensemble de trajets à supprimer
13.     // Contrat :
14.     //
15.     bool Contains(const Path* value) const;
16.     // Mode d'emploi :
17.     //      Teste si un trajet est présent dans la collection
18.     // Renvoie true si le trajet est présent dans la collection, false sinon
19.     //      > value : Trajet à tester
20.     // Contrat :
21.     //      value pointe vers un objet Path valide
22.     unsigned int GetSize(void) const;
23.     // Mode d'emploi :
24.     //      Retourne la taille courante de la collection
25.     // Contrat :
26.     //
27.     unsigned int GetMaxSize(void) const;
28.     // Mode d'emploi :
29.     //      Retourne la taille maximale actuelle de la collection
30.     // Contrat :
31.     //
32.     Path* Get(int index) const;
33.     // Mode d'emploi :
34.     //      Retourne un élément de la collection
35.     //      > index : indice de l'élément à retourner
36.     // Contrat :
37.     //      0 <= index < size
38.
39.     std::ostream& Print(std::ostream& os, bool advanced = false, bool com-
    posed = false) const;
40.     // Mode d'emploi :
41.     //      Ecrit une représentation en chaîne de caractères de la collection
42.     // sur un flux standard.
43.     //      > os : flux standard sur lequel écrire
44.     // Contrat :
45.     //
46.
47.
48. //------------------------------------------------ Surcharge d'opérateurs
49.     PathArray& operator=(const PathArray & other);
50.
51.
52. //-------------------------------------------- Constructeurs - destructeur
53.     PathArray(const PathArray & otherPathArray);
54.     // Mode d'emploi (constructeur de copie) :
55.     // Constructeur simple qui copie chaque attributs de "other" dans un nouveau objet
56.     //
57.     // Contrat :
58.     //
```

```
1.  PathArray(const unsigned int maxSize = MAX_SIZE);
2.      // Mode d'emploi :
3.      // Constructeur basique qui alloue un tableau de taille "cardMax"
4.      //
5.      // maxSize :
6.      // Taille maximal de la collection sans réallocation (par défaut : MAX_SIZE)
7.      //
8.      // Contrat :
9.      //
10.
11.     virtual ~PathArray();
12.     // Mode d'emploi :
13.     // Destructeur basique qui désalloue le tableau elements
14.     //
15.     // Contrat :
16.     //
17.
18. //------------------------------------------------------------------ PRIVE
19.
20. protected:
21. //---------------------------------------------------- Méthodes protégées
22.
23. //---------------------------------------------------- Attributs protégés
24.
25.     Path ** elements;
26.     unsigned int maxSize;
27.     unsigned int size;
28. };
29.
30. //-------------------------------- Autres définitions dépendantes de <PathArray>
31.
32. #endif // PATHARRAY_H
```

IV.5.b. Réalisation (PathArray.cpp)

```
1.  /*************************************************************************
2.                              PathArray  -  description
3.                                -------------------
4.      début                : 27/09/2018
5.      copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
6.      e-mail               : ...@insa-lyon.fr
7.  *************************************************************************/
8.
9.  //---------- Réalisation de la classe <PathArray> (fichier PathArray.cpp) ------------
10.
11. //------------------------------------------------------------------ INCLUDE
12.
13. //-------------------------------------------------------- Include système
14. #include <iostream>
15. using std::cout;
16. using std::endl;
17. //------------------------------------------------------ Include personnel
18. #include "PathArray.h"
19. //------------------------------------------------------------------ Constantes
```

```
1.  //------------------------------------------------------------------ PUBLIC
2.
3.  //----------------------------------------------------- Méthodes publiques
4.  bool PathArray::Contains(const Path* path) const
5.  // Algorithme :
6.  //
7.  {
8.      unsigned int j;
9.      for (j = 0; j < size; j++)
10.     {
11.         if (*elements[j] == *path)
12.         {
13.             return true;
14.         }
15.     }
16.     return false;
17. } //----- fin de Contains
18.
19. unsigned int PathArray::GetSize(void) const
20. // Algorithme :
21. //
22. {
23.     return size;
24. } //----- fin de GetSize
25.
26. unsigned int PathArray::GetMaxSize(void) const
27. // Algorithme :
28. //
29. {
30.     return maxSize;
31. } //----- fin de GetMaxSize
32.
33. Path* PathArray::Get(int index) const
34. // Algorithme :
35. //
36. {
37.     return elements[index];
38. } //----- fin de GetMaxSize
39.
40. std::ostream& PathArray::Print(std::ostream& os, bool advanced, bool composed) const
41. // Algorithme :
42. //
43. {
44.     unsigned int j;
45.     if (advanced)
46.     {
47.         if (size > 0)
48.         {
49.             for (j = 0; j < size - 1; j++)
50.             {
51.                 os << *elements[j] << std::endl;
52.                 os << "ou" << endl << endl;
53.             }
54.             os << *elements[size-1];
55.         }
56.     }
```

```
1.  else
2.      {
3.          for (j = 0; j < size; j++)
4.          {
5.              os << ((composed) ? "    Etape " : "Trajet ") << j + 1 << " - " << *ele-
    ments[j] << endl;
6.          }
7.      }
8.
9.
10.     return os;
11. } //----- fin de Print
12.
13. bool PathArray::Equals(const PathArray & anotherPathArray) const
14. // Algorithme :
15. //
16. {
17.     if (anotherPathArray.GetSize() != size)
18.     {
19.         return false;
20.     }
21.     else
22.     {
23.         unsigned int j;
24.         for (j = 0; j < size; j++)
25.         {
26.             if (!anotherPathArray.Contains(elements[j]))
27.                 return false;
28.         }
29.     }
30.     return true;
31. } //----- fin de Equals
32.
33. bool PathArray::Add(Path* pathToAdd)
34. // Algorithme :
35. //
36. {
37.     unsigned int j;
38.     for (j = 0; j < size; j++)
39.     {
40.
41.         if (*elements[j] == *pathToAdd)
42.         {
43.             return false;
44.         }
45.     }
46.     if (size == maxSize)
47.     {
48.         Adjust(size);
49.     }
50.     elements[size] = pathToAdd;
51.     size++;
52.
53.     return true;
54. } //----- fin de Add
```

```
1.  unsigned int PathArray::Adjust(int delta)
2.  // Algorithme :
3.  //
4.  {
5.      int temp = maxSize - size;
6.      if (delta >= 0 || -delta <= temp)
7.      {
8.          maxSize += delta;
9.      }
10.     else
11.     {
12.         maxSize = size;
13.     }
14.
15.     Path** newElements = new Path*[maxSize];
16.     unsigned int j;
17.     for (j = 0; j < size; j++)
18.     {
19.         newElements[j] = elements[j];
20.     }
21.     delete [] elements;
22.     elements = newElements;
23.     return maxSize;
24. } //----- fin de Adjust
25.
26. bool PathArray::Remove(Path* element)
27. // Algorithme :
28. //
29. {
30.     if (Contains(element))
31.     {
32.         Path** newElements = new Path*[size-1];
33.         unsigned int j;
34.         unsigned int i = 0;
35.         for (j = 0; j < size; j++)
36.         {
37.
38.             if (!(*elements[j] == *element)) //TODO we could define != operator
39.             {
40.                 newElements[i] = elements[j];
41.                 i++;
42.             }
43.         }
44.         delete [] elements;
45.         elements = newElements;
46.         size--;
47.         maxSize = size;
48.         return true;
49.     }
50.     maxSize = size;
51.     return false;
52. } //----- fin de Remove
53.
```

```
1.  unsigned int PathArray::Remove(const PathArray & anotherPathArray)
2.  {
3.      unsigned int oldmaxSize = maxSize;
4.      unsigned int j;
5.      int count = 0;
6.      if (Equals(anotherPathArray))
7.      {
8.          count = size;
9.          size = 0;
10.     }
11.     else
12.     {
13.         for (j = 0; j < anotherPathArray.GetSize(); j++)
14.         {
15.             if (Remove(anotherPathArray.Get(j)))
16.             {
17.                 count++;
18.             }
19.         }
20.         maxSize = oldmaxSize;
21.     }
22.
23.     return count;
24. } //----- fin de Remove
25.
26. //------------------------------------------------- Surcharge d'opérateurs
27. PathArray& PathArray::operator=(const PathArray& other)
28. {
29.     if (this != &other)
30.     {
31.         for (unsigned int j = 0; j < size; j++)
32.         {
33.             delete elements[j];
34.         }
35.         delete [] elements;
36.
37.         size = other.GetSize();
38.         maxSize = other.GetMaxSize();
39.
40.         elements = new Path*[maxSize];
41.         for (unsigned int j = 0; j < size; j++)
42.         {
43.             elements[j] = other.Get(j)->Clone();
44.         }
45.     }
46.     return *this;
47. #ifdef MAP
48.     cout << "Appel à la surcharge du = de <PathArray>" << endl;
49. #endif
50. }
51.
```

```cpp
//----------------------------------------- Constructeurs - destructeur
PathArray::PathArray(const PathArray & anotherPathArray)
{
    maxSize = anotherPathArray.maxSize;
    size = anotherPathArray.size;

    elements = new Path*[maxSize];

    for (unsigned int j = 0; j < size; j++)
    {
        elements[j] = anotherPathArray.elements[j]->Clone();
    }

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

PathArray::PathArray(const unsigned int maxSize)
{
    elements = new Path*[maxSize];
    this->maxSize = maxSize;

    size = 0;

#ifdef MAP
    cout << "Appel au constructeur de <PathArray>" << endl;
#endif
} //----- Fin de PathArray

PathArray::~PathArray()
{
    for (unsigned int j = 0; j < size; j++)
    {
        delete elements[j];
    }
    delete [] elements;

#ifdef MAP
    cout << "Appel au destructeur de <PathArray>" << endl;
#endif
} //----- Fin de ~PathArray

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

//----------------------------------------------------- Méthodes protégées
```

IV.6. SearchEngine

IV.6.a. Fichier d'en-tête (SearchEngine.h)

```
1.  /*************************************************************************
2.                             SearchEngine  -  description
3.                               -------------------
4.      début                : 04/10/2018
5.      copyright            : (C) 2018 par WALLYN Valentin - FROLIN Balthazar
6.      e-mail               : $EMAIL$
7.  *************************************************************************/
8.
9.  //---------- Interface de la classe <SearchEngine> (fichier SearchEngine.h) ------------
    ---
10. #if ! defined ( SEARCHENGINE_H )
11. #define SEARCHENGINE_H
12.
13. //------------------------------------------------- Interfaces utilisées
14. #include "../Path/Path.h"
15. #include "../PathArray/PathArray.h"
16. //------------------------------------------------------------ Constantes
17. const int DEFAULT_MAX_SIZE = 10;
18. //------------------------------------------------------------------ Types
19. struct node
20. {
21.     unsigned int startIndex;
22.     unsigned int endIndex;
23.     node* previous;
24.     node* next;
25. };
26.
27. //------------------------------------------------------------------------
28. // Rôle de la classe <SearchEngine>
29. //
30. // Le search engine permet de chercher des trajets entre 2 villes dans tout
31. // le catalogue avec ou sans composition.
32. //
33. //------------------------------------------------------------------------
34.
35. class SearchEngine
36. {
37. //----------------------------------------------------------------- PUBLIC
38.
39. public:
40. //-------------------------------------------------- Méthodes publiques
41.
42.
43.     void SimpleSearch(const char * startingCity, const  char * endingCity) const;
44.     // Mode d'emploi :
45.     //      Cherche et affiche le(s) trajet(s) directs correspondant(s) du cata-
    logue pour
46.     // se rendre de la ville "startingCity" à la ville "endingCity"
47.     //      > startingCity : ville de départ
48.     //      > endingCity : ville d'arrivée
49.     // Contrat :
50.     //
51.
52.     void AdvancedSearch(const char * startingCity,const  char * endingCity) const;
53.     // Mode d'emploi :
54.     //      Cherche et affiche le(s) trajet(s) pour se rendre de la
55.     // ville "startingCity" à la ville "endingCity" en effectuant une
56.     // composition de plusieurs sous-trajets du catalogue
57.     //      > startingCity : ville de départ
58.     //      > endingCity : ville d'arrivée
59.     // Contrat :
60.     //
```

```
1.  void AddPath(Path* path);
2.      // Mode d'emploi :
3.      //      Ajoute un trajet au moteur de recherche
4.      //      > path : pointeur sur le trajet à ajouter
5.      // Contrat :
6.      //      path est un pointeur sur trajet valide
7.
8.
9.
10. //------------------------------------------------ Surcharge d'opérateurs
11.     SearchEngine & operator = ( const SearchEngine & other );
12.     // Mode d'emploi :
13.     //
14.     // Contrat :
15.     //
16.
17.
18. //-------------------------------------------- Constructeurs - destructeur
19.     SearchEngine ( const SearchEngine & other );
20.     // Mode d'emploi (constructeur de copie) :
21.     //
22.     // Contrat :
23.     //
24.
25.     SearchEngine ( const unsigned int size = DEFAULT_MAX_SIZE);
26.     // Mode d'emploi :
27.     //
28.     // Contrat :
29.     //
30.
31.     virtual ~SearchEngine ( );
32.     // Mode d'emploi :
33.     //
34.     // Contrat :
35.     //
36.
37. //------------------------------------------------------------------ PRIVE
38.
39. private:
40. //------------------------------------------------------ Méthodes protégées
41.
42.     unsigned int getStartCityIndex(const char * city) const;
43.     unsigned int getEndCityIndex(const char * city) const;
44.     bool recursiveSearch(node* node, unsigned int endIndex, bool * doneIndex) const;
45.
46. //------------------------------------------------------ Attributs protégés
47.
48.     PathArray*** adjacencyMatrix;
49.     char ** startCities;
50.     char ** endCities;
51.
52.     unsigned int startCurrentSize;
53.     unsigned int endCurrentSize;
54.     unsigned int startMaxSize;
55.     unsigned int endMaxSize;
56.
57. };
58.
59. //-------------------------------- Autres définitions dépendantes de <SearchEngine>
60.
61. #endif // SEARCHENGINE_H
```

```
1.  /*************************************************************************
2.                              SearchEngine  -  description
3.                                -------------------
4.      début                : 04/10/2018
5.      copyright            : (C) 2018 par WALLYN Valentin - FROLIN Balthazar
6.      e-mail               : $EMAIL$
7.  *************************************************************************/
8.
9.  //---------- Réalisation de la classe <SearchEngine> (fichier SearchEngine.cpp) ---------
    ---
10.
11. //------------------------------------------------------------------ INCLUDE
12.
13. //-------------------------------------------------------- Include système
14. #include <iostream>
15. #include <cstring>
16.
17. using namespace std;
18.
19. //------------------------------------------------------ Include personnel
20. #include "SearchEngine.h"
21.
22. //------------------------------------------------------------------ Constantes
23.
24. //-------------------------------------------------------------------- PUBLIC
25.
26. //----------------------------------------------------- Méthodes publiques
27.
28. void SearchEngine::SimpleSearch(const char * startingCity, const char * end-
    ingCity) const
29. // Algorithme :
30. //      Recherche dans la matrice de proximité si un trajet direct existe entre starting-
    City
31. // et endingCity. Si oui, affiche le trajet, sinon affiche qu'aucun tra-
    jet n'a été trouvé.
32. {
33.     unsigned int startIndex = getStartCityIndex(startingCity);
34.     unsigned int endIndex = getEndCityIndex(endingCity);
35.
36.     if (startIndex != startCurrentSize && endIndex != endCurrentSize && adjacencyMa-
    trix[startIndex][endIndex]->GetSize() > 0)
37.     {
38.         adjacencyMatrix[startIndex][endIndex]->Print(cout);
39.     }
40.     else
41.     {
42.         cout << "\tAucun trajet correspondant n'a été trouvé." << endl;
43.     }
44.
45. } //----- Fin de simpleSearch
46.
47.
48. void SearchEngine::AdvancedSearch(const char * startingCity, const char * end-
    ingCity) const
49. // Algorithme :
50. //      Instancie le noeud initial de la recherche récursive puis apelle la fonc-
    tion de recherche
51. // récursive. Voir la fonction "recursiveSearch" pour l'algorithme de recherche.
52. {
53.     unsigned int startIndex = getStartCityIndex(startingCity);
54.     unsigned int endIndex = getEndCityIndex(endingCity);
```

```
1.      if (startIndex != startCurrentSize && endIndex != endCurrentSize)
2.      {
3.          SearchEngine::node node;
4.          node.startIndex = startIndex;
5.          node.previous = NULL;
6.
7.          bool* doneIndex = new bool[startCurrentSize];
8.          for (unsigned int j = 0; j < startCurrentSize; j++)
9.          {
10.             doneIndex[j] = false;
11.         }
12.         recursiveSearch(&node, endIndex, doneIndex);
13.         delete [] doneIndex;
14.     }
15.     else
16.     {
17.         cout << "\tAucun trajet correspondant n'a été trouvé." << endl;
18.     }
19. } //----- Fin de advancedSearch
20.
21.
22. void SearchEngine::AddPath(Path* path)
23. // Algorithme :
24. //      Ajoute un trajet à la matrice de proximité en effectuant les réallocations
25. // de mémoire nécessaires lorsque les tableaux sont pleins. Stocke les villes de départ
26. // et d'arrivée de manière unique
27. {
28.     unsigned int startCityIndex = getStartCityIndex(path->GetStartCity());
29.     unsigned int endCityIndex = getEndCityIndex(path->GetEndCity());
30.
31.     if (startCityIndex == startCurrentSize)
32.     {
33.
34.         startCities[startCityIndex] = new char[strlen(path->GetStartCity()) + 1];
35.         strcpy(startCities[startCityIndex], path->GetStartCity());
36.         startCurrentSize++;
37.         if (startCurrentSize == startMaxSize)
38.         {
39.             char ** tmpCities = new char*[startMaxSize * 2];
40.             for (unsigned int j = 0; j < startCurrentSize; j++)
41.             {
42.                 tmpCities[j] = new char[strlen(startCities[j]) + 1];
43.                 strcpy(tmpCities[j], startCities[j]);
44.             }
45.             for (unsigned int j = 0; j < startCurrentSize; j++)
46.             {
47.                 delete [] startCities[j];
48.             }
49.
50.             delete [] startCities;
51.             startCities = tmpCities;
52.
53.
```

```
1.
                PathArray*** tmp = new PathArray**[startMaxSize * 2];
2.              for (unsigned int j = 0; j < startMaxSize * 2; j++)
3.              {
4.                  tmp[j] = new PathArray*[endMaxSize];
5.                  for (unsigned int i = 0; i < endMaxSize; i++)
6.                  {
7.                      tmp[j][i] = new PathArray();
8.                      if (i < endCurrentSize && j < startCurrentSize - 1)
9.                      {
10.                         *tmp[j][i] = *adjacencyMatrix[j][i];
11.                     }
12.                 }
13.             }
14.             for (unsigned int j = 0; j < startMaxSize; j++)
15.             {
16.                 for (unsigned int i = 0; i < endMaxSize; i++)
17.                 {
18.                     delete adjacencyMatrix[j][i];
19.                 }
20.                 delete [] adjacencyMatrix[j];
21.             }
22.             startMaxSize = startMaxSize * 2;
23.             delete [] adjacencyMatrix;
24.             adjacencyMatrix = tmp;
25.         }
26.     }
27.
28.     if (endCityIndex == endCurrentSize)
29.     {
30.         endCities[endCityIndex] = new char[strlen(path->GetEndCity()) + 1];
31.         strcpy(endCities[endCityIndex], path->GetEndCity());
32.         endCurrentSize++;
33.         if (endCurrentSize == endMaxSize)
34.         {
35.             char ** tmpCities = new char*[endMaxSize * 2];
36.             for (unsigned int j = 0; j < endCurrentSize; j++)
37.             {
38.                 tmpCities[j] = new char[strlen(endCities[j]) + 1];
39.                 strcpy(tmpCities[j], endCities[j]);
40.             }
41.             for (unsigned int j = 0; j < endCurrentSize; j++)
42.             {
43.                 delete [] endCities[j];
44.             }
45.
46.             delete [] endCities;
47.             endCities = tmpCities;
48.
49.
50.
51.             PathArray*** tmp = new PathArray**[startMaxSize];
```

```
1.  for (unsigned int j = 0; j < startMaxSize; j++)
2.              {
3.                  tmp[j] = new PathArray*[endMaxSize * 2];
4.                  for (unsigned int i = 0; i < endMaxSize * 2; i++)
5.                  {
6.                      tmp[j][i] = new PathArray();
7.                      if (i < endCurrentSize - 1 && j < startCurrentSize)
8.                      {
9.                          *tmp[j][i] = *adjacencyMatrix[j][i];
10.                     }
11.                 }
12.             }
13.             for (unsigned int j = 0; j < startMaxSize; j++)
14.             {
15.                 for (unsigned int i = 0; i < endMaxSize; i++)
16.                 {
17.                     delete adjacencyMatrix[j][i];
18.                 }
19.                 delete [] adjacencyMatrix[j];
20.             }
21.             endMaxSize = endMaxSize * 2;
22.             delete [] adjacencyMatrix;
23.             adjacencyMatrix = tmp;
24.         }
25.
26.     }
27.
28.     adjacencyMatrix[startCityIndex][endCityIndex]->Add(path->Clone());
29.
30. } //----- Fin de addPath
31.
32. //------------------------------------------------- Surcharge d'opérateurs
33. SearchEngine & SearchEngine::operator = ( const SearchEngine & other )
34. // Algorithme :
35. //
36. {
37.     return *this;
38. } //----- Fin de operator =
39.
40.
41. //-------------------------------------------- Constructeurs - destructeur
42. SearchEngine::SearchEngine ( const SearchEngine & other )
43. // Algorithme :
44. //
45. {
46. #ifdef MAP
47.     cout << "Appel au constructeur de copie de <SearchEngine>" << endl;
48. #endif
49. } //----- Fin de SearchEngine (constructeur de copie)
50.
```

```
1.  SearchEngine::SearchEngine( const unsigned int size )
2.  // Algorithme :
3.  //
4.  {
5.  #ifdef MAP
6.      cout << "Appel au constructeur de <SearchEngine>" << endl;
7.  #endif
8.      adjacencyMatrix = new PathArray**[size];
9.      for (unsigned int j = 0; j < size; j++)
10.     {
11.         adjacencyMatrix[j] = new PathArray*[size];
12.         for (unsigned int i = 0; i < size; i++)
13.         {
14.             adjacencyMatrix[j][i] = new PathArray();
15.         }
16.     }
17.     startCities = new char*[size];
18.     endCities = new char*[size];
19.     startCurrentSize = 0;
20.     startMaxSize = size;
21.     endCurrentSize = 0;
22.     endMaxSize = size;
23.
24. } //----- Fin de SearchEngine
25.
26.
27. SearchEngine::~SearchEngine ( )
28. // Algorithme :
29. //
30. {
31. #ifdef MAP
32.     cout << "Appel au destructeur de <SearchEngine>" << endl;
33. #endif
34.     for (unsigned int j = 0; j < startMaxSize; j++)
35.     {
36.         for (unsigned int i = 0; i < endMaxSize; i++)
37.         {
38.             delete adjacencyMatrix[j][i];
39.         }
40.         delete [] adjacencyMatrix[j];
41.     }
42.     delete [] adjacencyMatrix;
43.
44.     for (unsigned int j = 0; j < startCurrentSize; j++)
45.     {
46.         delete [] startCities[j];
47.     }
48.
49.     for (unsigned int j = 0; j < endCurrentSize; j++)
50.     {
51.         delete [] endCities[j];
52.     }
53.     delete [] startCities;
54.     delete [] endCities;
55. } //----- Fin de ~SearchEngine
56.
57.
```

```
1.  //------------------------------------------------------------------ PRIVE
2.
3.  //---------------------------------------------------- Méthodes protégées
4.  unsigned int SearchEngine::getStartCityIndex(const char * city) const
5.  // Algorithme :
6.  //      Parcoure la liste des villes de départ enregistrées et récupère l'index
7.  // de la ville "city"
8.  {
9.      for (unsigned int  j = 0; j < startCurrentSize; j++)
10.     {
11.         if (strcmp(startCities[j], city) == 0)
12.         {
13.             return j;
14.         }
15.     }
16.     return startCurrentSize;
17. } //----- Fin de getStartCityIndex
18.
19. unsigned int SearchEngine::getEndCityIndex(const char * city) const
20. // Algorithme :
21. //      Parcoure la liste des villes d'arrivée enregistrées et récupère l'index
22. // de la ville "city"
23. {
24.     for (unsigned int j = 0; j < endCurrentSize; j++)
25.     {
26.         if (strcmp(endCities[j], city) == 0)
27.         {
28.             return j;
29.         }
30.     }
31.     return endCurrentSize;
32. } //----- Fin de getEndCityIndex
33.
34. bool SearchEngine::recursiveSearch(node* node, unsigned int endIndex, bool * doneIn-
    dex) const
35. // Algorithme :
36. //      Explore les noeuds reliés au noeud initial "node" dans la matrice de proxi-
    mité récursivement
37. // jusqu'à l'arrivée sur un noeud possèdant comme ville d'arrivée la destination objec-
    tif (endIndex)
38. // ou que tous les noeuds de départs aient été explorés (stockés dans doneIndex)
39. // Si un trajet a été trouvé, affiche la composition de celui-ci en parcou-
    rant les noeuds solution
40. {
41.     for (unsigned int j = 0; j < endCurrentSize; j++)
42.     {
43.         node->endIndex = j;
44.
45.         if (adjacencyMatrix[node->startIndex][j]->GetSize() > 0)
46.         {
47.             if (j == endIndex)
48.             {
49.                 node->next = NULL;
50.                 while (node->previous != NULL)
51.                 {
52.                     node = node->previous;
53.                 }
54.                 cout << "Début trajet :" << endl;
```

```
1.   while (node->next != NULL)
2.                   {
3.                       adjacencyMatrix[node->startIndex][node->endIndex]-
    >Print(cout, true);
4.                       cout << "puis" << endl;
5.                       node = node->next;
6.                   }
7.                   adjacencyMatrix[node->startIndex][node->endIndex]->Print(cout, true);
8.                   cout << "Fin trajet" << endl << endl;
9.                   return true;
10.              }
11.              unsigned int startIndex = getStartCityIndex(endCities[j]);
12.              if (startIndex != startCurrentSize)
13.              {
14.                  if (!doneIndex[startIndex])
15.                  {
16.                      doneIndex[node->startIndex] = true;
17.                      SearchEngine::node next;
18.                      next.previous = node;
19.                      next.startIndex = startIndex;
20.
21.                      node->next = &next;
22.                      recursiveSearch(&next, endIndex, doneIndex);
23.                  }
24.              }
25.
26.          }
27.      }
28.      return false;
29. } //----- Fin de recursiveSearch
```

IV.7. Module main

```
1.  //---------- Réalisation du module <TCatalog> (fichier TCatalog.cpp) ---------------
2.
3.  ////////////////////////////////////////////////////////////////  INCLUDE
4.  //-------------------------------------------------------- Include système
5.
6.  #include <iostream>
7.
8.  //------------------------------------------------------ Include personnel
9.
10. #include "Catalog/Catalog.h"
11.
12. using namespace std;
13.
14. ////////////////////////////////////////////////////////////////  PUBLIC
15. //----------------------------------------------------- Fonctions publiques
16.
17. int main()
18. {
19.     Catalog catalog = Catalog();
20.     catalog.Run();
21.
22.     return 0;
23. }
```

Code Source :

Le code source de VoyageVoyage se trouve dans le dossier « Project » .
Il est également accessible ici : https://github.com/Balthov60/TP2-CPP-Maranzana

Architecture du projet :

```
Project
    |
    | ---------- src
    |              | ---------- <Module>
    |              |                | ---------- <Module>.h
    |              |                | ---------- <Module>.cpp
    |              | ---------- Main.cpp
    |
    | ---------- test
    |              | ---------- <Module>
    |              |                | ---------- T<Module>.h
    |              |                | ---------- T<Module>.cpp
    |
    | ---------- .Build
    |              | ---------- <Module>.o
    |              | ---------- T<Module>                    << Test Executable
    |              | ---------- VoyageVoyage                 << Main Executable
    |
    | ---------- Makefile
```

Instructions d'utilisation

Les instructions d'utilisations sont également disponibles dans le fichier README.md

1. **Compilation**
   - Se placer dans le dossier « Project »
   - Exécuter « make init »
   - Compiler un exécutable :
     - Version de production : Exécuter « make »
     - Version de debug : Exécuter « make debug »
     - Version de test : Exécuter « make test T<Module> »
   - Nettoyer les fichiers de build :
     - Fichiers .o : Exécuter « make clean »
     - Fichiers .o et exécutable : Exécuter « make clean-all »
2. **Exécution**
   - Version de production/debug : Exécuter « .Build/VoyageVoyage »
   - Versions de test : Exécuter « .Build/T<Module> »

V. Conclusion

V.1. Problèmes rencontrés

V.1.a. Makefile

Lors de la création du Makefile, nous avons rencontré un problème de portée des variables. En effet, il nous a été impossible d'éditer des variables du Makefile depuis le code bash appelé dans les règles.

La variable était considérée comme « une commande » et nous n'avons trouvé aucune solution à ce problème. D'une manière générale, la création du Makefile générique a nécessité une part de travail importante.

V.1.b. Gestion de la mémoire

La manipulation intensive des pointeurs et des objets a posé des problèmes de gestion de la mémoire. La rigueur nécessaire à cet exercice est nouvelle et bouscule les habitudes de programmation. A force d'essais/erreurs, nous sommes finalement arrivés à une solution satisfaisante (selon Valgrind). Cependant, avec le recul certains choix de gestion ne sont pas nécessairement très judicieux. Cela est discuté dans la partie suivante.

V.2. Améliorations possibles

Cette application est un exercice académique – il pourrait donc bien entendu être beaucoup plus développé dans ses fonctionnalités. Toutefois, nous avons repéré des points d'amélioration par rapport aux notions abordées qui auraient été intéressants à mettre en place si du temps supplémentaire était accordé à ce TP.

V.2.a. Makefile

Deux améliorations pourraient être apportés au Makefile :

- Récupérer le nom des modules de manière dynamique en récupérant la liste des dossiers afin que le Makefile n'ai pas besoin d'être « configuré ».
- Compiler uniquement les fichiers nécessaires aux exécutables de test sans avoir à les spécifier dans les constantes du Makefile.

V.2.b. Gestion de la mémoire

La solution finale ne possède pas de fuites mémoires, cependant la gestion de la mémoire peut être amélioré.

- Le PathArray pourrait effectuer une copie du trajet pointé lors de l'ajout afin de le rendre indépendant de la source du trajet. Cela serait primordial si cette classe était destinée à être utilisée en externe.
- Le SearchEngine pourrait ne pas dupliquer les trajets entiers et travailler directement sur les pointeurs. Il ne serait ainsi pas en charge de la destruction des trajets sur lesquels il travaille. Dans le cadre de l'apprentissage de la gestion de la mémoire, il était plus simple qu'il ne soit pas dépendant d'une autre classe pour la gestion de sa mémoire afin de simplifier les tests isolés et le repérage des fuites mémoires.

V.2.c. Pattern strategy

Dans le cas où les moyens de transports auraient une influence plus importante qu'une simple modification d'affichage, il serait intéressant de les implémenter sous la forme d'un pattern strategy plutôt qu'une énumération.

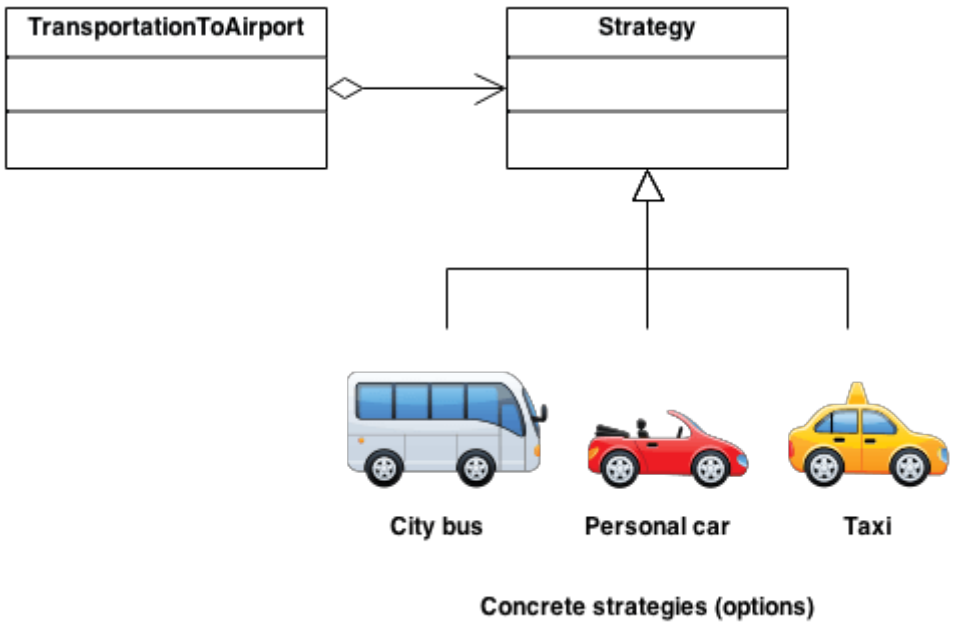

Figure 3 – Illustration du pattern strategy (src : https://sourcemaking.com/design_patterns/strategy)