Compte-Rendu TP IFA-3-POO1-2

Application « Voyage Voyage »



Table des matières

[I. Contexte de l’application 4](#_Toc526867092)

[II. Description détaillée des classes 5](#_Toc526867093)

[III. Description de la structure de données employée 7](#_Toc526867094)

[IV. Listing des classes 7](#_Toc526867095)

[IV.1. Catalog 7](#_Toc526867096)

[IV.1.a. Fichier d’en-tête (Catalog.h) 7](#_Toc526867097)

[8](#_Toc526867098)

[10](#_Toc526867099)

[IV.1.b. Réalisation (Catalog.cpp) 11](#_Toc526867100)

[12](#_Toc526867101)

[13](#_Toc526867102)

[14](#_Toc526867103)

[15](#_Toc526867104)

[16](#_Toc526867105)

[IV.2. Path 17](#_Toc526867106)

[IV.2.a. Fichier d’en-tête (Path.h) 17](#_Toc526867107)

[18](#_Toc526867108)

[IV.2.b. Réalisation (Path.cpp) 19](#_Toc526867109)

[20](#_Toc526867110)

[IV.3. SimplePath 21](#_Toc526867111)

[IV.3.a. Fichier d’en-tête (SimplePath.h) 21](#_Toc526867112)

[22](#_Toc526867113)

[IV.3.b. Réalisation (SimplePath.cpp) 23](#_Toc526867114)

[23](#_Toc526867115)

[24](#_Toc526867116)

[IV.4. ComposedPath 25](#_Toc526867117)

[IV.4.a. Fichier d’en-tête (ComposedPath.h) 25](#_Toc526867118)

[IV.4.b. Réalisation (ComposedPath.cpp) 27](#_Toc526867119)

[IV.5. PathArray 29](#_Toc526867120)

[IV.5.a. Fichier d’en-tête (PathArray.h) 30](#_Toc526867121)

[31](#_Toc526867122)

[IV.5.b. Réalisation (PathArray.cpp) 32](#_Toc526867123)

[IV.6. SearchEngine 37](#_Toc526867124)

[IV.6.a. Fichier d’en-tête (SearchEngine.h) 37](#_Toc526867125)

[IV.7. Module main 46](#_Toc526867126)

[V. Conclusion 48](#_Toc526867127)

[V.1. Problèmes rencontrés 48](#_Toc526867128)

[V.1.a. Makefile 48](#_Toc526867129)

[V.1.b. Gestion de la mémoire 48](#_Toc526867130)

[V.2. Améliorations possibles 48](#_Toc526867131)

[V.2.a. Makefile 48](#_Toc526867132)

[V.2.b. Gestion de la mémoire 48](#_Toc526867133)

[V.2.c. Pattern strategy 49](#_Toc526867134)

1. 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.

1. 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.)

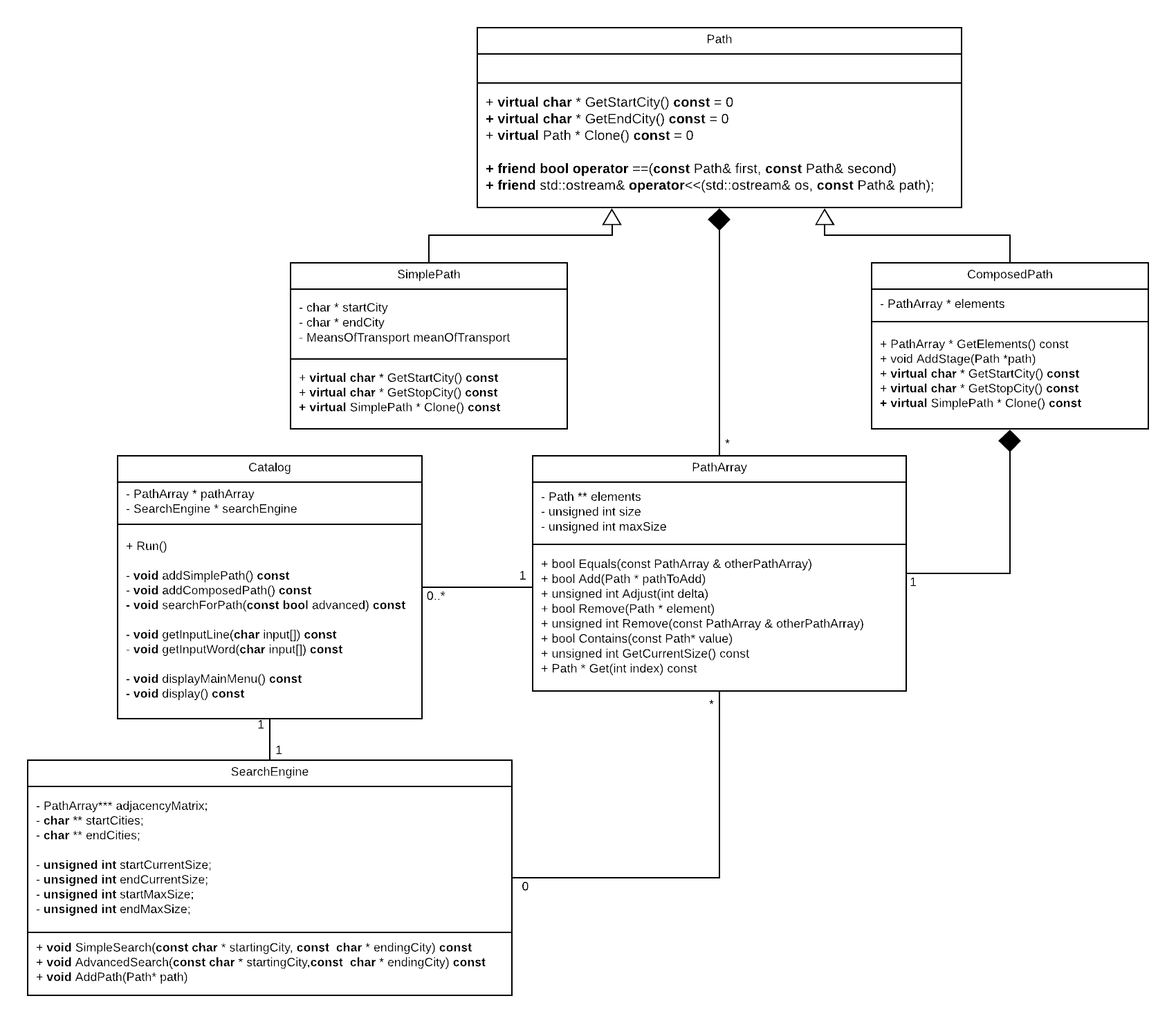


Figure 1 – Diagramme de classes de l’application VoyageVoyage

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 inspiré 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.

1. Description de la structure de données employée

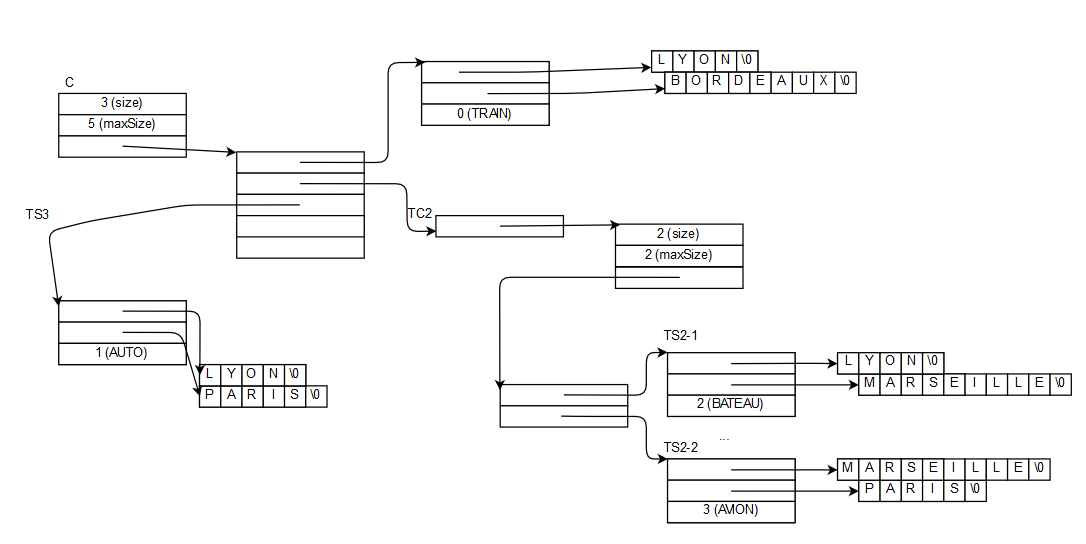


Figure 2 – Dessin d’abstraction de la mémoire lors de l’utilisation de PathArray avec le jeu d’essai du sujet

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.

1. Listing des classes
   1. Catalog
      1. 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. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
9. //---------- Interface de la classe <Catalog> (fichier Catalog.h) ----------------
10. #if ! defined ( CATALOG\_H )
11. #define CATALOG\_H
13. //--------------------------------------------------- Interfaces utilisées
15. #include <iostream>
17. #include "../PathArray/PathArray.h"
18. #include "../SearchEngine/SearchEngine.h"
19. //------------------------------------------------------------- Constantes
21. //------------------------------------------------------------------ Types
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
34. **public**:
35. //----------------------------------------------------- Méthodes publiques
37. **void** Run();
38. // Mode d'emploi :
39. // Affiche des différentes commandes disponible, attend les entrées utilisateurs et appelle les fonctions associées.
40. //
41. // Contrat :
42. //
44. //--------------------------------------------- Constructeur & Destructeur
46. Catalog();
47. // Mode d'emploi :
48. // Simple constructeur qui alloue une pathArray & un searchEngine
49. //
50. // Contrat :
51. //
53. **virtual** ~Catalog ( );
54. // Mode d'emploi :
55. // Simple destructeur qui désalloue pathArray & searchEngine
56. //
57. // Contrat :
58. //
60. //------------------------------------------------------------------ PRIVE
62. **private**:
63. //----------------------------------------------------- Méthodes protégées
65. **void** addSimplePath() **const**;
66. // Mode d'emploi :
67. // Demande des informations et ajoute un SimplePath à la collection
68. //
69. // Contrat :
70. //
71. **void** addComposedPath() **const**;
72. // Mode d'emploi :
73. // Demande des informations et ajoute un ComposedPath à la collection
74. //
75. // Contrat :
76. //
77. **void** addPathAndNotifyUser(Path \* path) **const**;
78. // Mode d'emploi :
79. // Essaie d'ajouter un Path à la collection et informe l'utilisateur
80. // Soit le Path est ajouté soit il existe déjà.
81. //
82. // Contrat :
83. //
85. **void** searchForPath(**const** **bool** advanced) **const**;
86. // Mode d'emploi :
87. // Demande la ville de départ et d'arrivée et lance une recherche de trajet
88. //
89. //      > advanced : true pour effectuer une recherche avancée, false pour
90. // une recherche simple
91. //
92. // Contrat :
93. //
95. **void** askForStartingCity(**char** \*startingCity) **const**;
96. // Mode d'emploi :
97. // Demande une chaine de caractères à l'utilisateur pour le champ startCity
98. //
99. // Contrat :
100. //
101. **void** askForEndingCity(**char** \*endingCity) **const**;
102. // Mode d'emploi :
103. // Demande une chaine de caractères à l'utilisateur pour le champ endCity
104. //
105. // Contrat :
106. //
107. unsigned **int** askForStageQty() **const**;
108. // Mode d'emploi :
109. // Demande un entier positif pour créer un trajet composé de n trajets simples.
110. //
111. // Contrat :
112. //
114. MeansOfTransport displayAndAskForMeansOfTransport() **const**;
115. // Mode d'emploi :
116. // Affiche les differents moyens de transport et attend que l'utilisateur en selectionne un.
117. //
118. // Contrat :
119. //

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 retour à la ligne
10. //
11. // Contrat :
12. //
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. //
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. //
52. //----------------------------------------------------- Attributs protégés
54. PathArray \* pathArray;
55. SearchEngine \* searchEngine;
57. };
59. //-------------------------------- Autres définitions dépendantes de <Catalog>
61. #endif // CATALOG\_H
    * 1. Réalisation (Catalog.cpp)
62. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
63. Catalog  -  Affichage & Manipulation des Trajets
64. -------------------
65. début                : 27/09/2018
66. copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
67. e-mail               : ...@insa-lyon.fr
68. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
70. //---------- Réalisation de la classe <Catalog> (fichier Catalog.cpp) ------------
72. //---------------------------------------------------------------- INCLUDE
74. //-------------------------------------------------------- Include système
75. #include <iostream>
76. #include <cstring>
78. //------------------------------------------------------ Include personnel
79. #include "Catalog.h"
80. #include "../SimplePath/SimplePath.h"
81. #include "../ComposedPath/ComposedPath.h"
83. **using** std::cout;
84. **using** std::endl;
85. //------------------------------------------------------------- Constantes
86. **const** **int** INPUT\_MAX\_SIZE = 100;
87. **const** **char** SEPARATOR[] = "=================================================\r\n";
88. //----------------------------------------------------------------- PUBLIC
90. //----------------------------------------------------- Méthodes publiques
92. **void** Catalog::Run()
93. {
94. cout << "Bienvenue sur VoyageVoyage, L'app qui vous fera voyager !" << endl;
95. displayMainMenu();
97. **char** input[INPUT\_MAX\_SIZE];
98. getInputWord(input);
100. **while** (strcmp(input,"9") != 0 && strcmp(input,"Quit") != 0 && strcmp(input,"Quitter") != 0 && strcmp(input,"Q") != 0)
101. {
102. **if** (strcmp(input, "0") == 0)
103. {
104. displayMainMenu();
105. }
106. **else** **if** (strcmp(input,"1") == 0)
107. {
108. display();
109. }
110. **else** **if** (strcmp(input, "2") == 0)
111. {
112. addSimplePath();
113. }
114. **else** **if** (strcmp(input, "3") == 0)
115. {
116. addComposedPath();

1. }
2. **else** **if** (strcmp(input, "4") == 0)
3. {
4. //recherche simple
5. searchForPath(**false**);
6. }
7. **else** **if** (strcmp(input, "5") == 0)
8. {
9. //recherche avancée
10. searchForPath(**true**);
11. }
12. **else**
13. {
14. displayMainMenu();
15. }
16. cout << "Entrez une nouvelle commande (0 pour afficher le menu) : " << endl;
17. getInputWord(input);
18. }
20. cout << "Merci d'avoir utilisé VoyageVoyage ! à Bientot !" << endl;
21. }
23. //-------------------------------------------- Constructeurs - destructeur
25. Catalog::Catalog ( )
26. {
27. pathArray = **new** PathArray();
28. searchEngine = **new** SearchEngine();
30. #ifdef MAP
31. cout << "Appel au constructeur de <Catalog>" << endl;
32. #endif
33. } //----- Fin de Catalog

36. Catalog::~Catalog ( )
37. {
38. **delete** pathArray;
39. **delete** searchEngine;
41. #ifdef MAP
42. cout << "Appel au destructeur de <Catalog>" << endl;
43. #endif
44. } //----- Fin de ~Catalog
46. //------------------------------------------------------------------ PRIVE
48. //----------------------------------------------------- Méthodes protégées
50. /\* Edition Methods \*/
52. **void** Catalog::addSimplePath() **const**
53. {
54. **char** startingCity[INPUT\_MAX\_SIZE];
55. **char** endingCity[INPUT\_MAX\_SIZE];
56. MeansOfTransport meansOfTransport;
58. cout << SEPARATOR;
59. cout << "Ajout d'un Trajet Simple..." << endl;

1. askForStartingCity(startingCity);
2. askForEndingCity(endingCity);
3. meansOfTransport = displayAndAskForMeansOfTransport();
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;
14. cout << SEPARATOR;
15. cout << "Ajout d'un Trajet Composé..." << endl;
17. **int** stageQty = askForStageQty();
19. **for** (**int** i = 1; i <= stageQty; i++)
20. {
21. cout << "Etape " << i << endl;
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. }
34. askForEndingCity(endingCity);
35. meansOfTransport = displayAndAskForMeansOfTransport();
37. composedPath->AddStage(**new** SimplePath(startingCity, endingCity, meansOfTransport));
38. }
40. addPathAndNotifyUser(composedPath);
41. }
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. }
55. cout << SEPARATOR;
56. }

1. /\* Search Methods \*/
2. **void** Catalog::searchForPath(**const** **bool** advanced) **const**
3. {
4. **char** startingCity[INPUT\_MAX\_SIZE];
5. **char** endingCity[INPUT\_MAX\_SIZE];
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. }
17. askForStartingCity(startingCity);
19. **do**
20. {
21. cout << "\tVille d'arrivée : ";
22. }
23. **while** (!getInputLine(endingCity));

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. }
36. cout << SEPARATOR;
37. }
39. /\* Input Methods \*/

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;
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);
13. **return** stageQty;
14. }
16. MeansOfTransport Catalog::displayAndAskForMeansOfTransport() **const**
17. {
18. **char** transport[MEAN\_OF\_TRANSPORT\_STRING\_MAX\_SIZE];
20. **for** ( ; ; )
21. {
22. **do**
23. {
24. displayMeansOfTransport();
25. }
26. **while** (!getInputLine(transport));
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. }
34. cout << "Moyen de transport non valide...\n";
35. }
36. }
38. **bool** Catalog::getInputLine(**char** \*input) **const**
39. {
40. **if** (fscanf(stdin, "%99[^\n]", input) != 1)
41. {
42. cleanInputStream();
43. **return** **false**;
44. }
46. cleanInputStream();
47. capitalizeFirstWordsLetter(input);
49. **return** **true**;
50. }
51. **bool** Catalog::getInputWord(**char** \*input) **const**
52. {
53. **if** (fscanf(stdin, "%99s", input) != 1)
54. {
55. inputError();
56. }
58. cleanInputStream();
59. capitalizeFirstWordsLetter(input);
60. }
61. **return** **true**;
62. }

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 majuscule (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;
18. **continue**;
19. }
20. **if**(input[i]==' ') // Check Space
21. {
22. i++;
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. }
38. **void** Catalog::inputError() **const**
39. {
40. cout << "input Error\n";
41. exit(1);
42. }
44. /\* Output Methods \*/
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
    1. Path
56. << "Taper 4 pour chercher un trajet (version simple)."      << endl
57. << "Taper 5 pour chercher un trajet (version avancée)."     << endl
58. <<                                                             endl
59. << "Taper 9 pour quitter l'application \"Voyage Voyage\""   << endl
60. << SEPARATOR
61. <<                                                             endl;
62. }
64. **void** Catalog::display() **const**
65. {
66. cout << SEPARATOR;
67. cout << "Liste des Voyages Disponibles..."                      << endl;
68. cout                                                            << endl;
70. **for** (unsigned **int** i = 0; i < pathArray->GetSize(); i++)
71. {
72. cout << "#" << i + 1 << " " << \*pathArray->Get(i) << endl;
73. }
75. cout << SEPARATOR;
76. }
78. **void** Catalog::displayMeansOfTransport() **const**
79. {
80. cout << "\tMoyen de transport ( ";
82. **for** (**int** i = 0; i < MEAN\_OF\_TRANSPORT\_QTY; i++) {
83. cout << MEAN\_OF\_TRANSPORT\_STRINGS[i] << " ";
84. }
86. cout << ") : ";
87. }
    * 1. 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. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
9. //---------- Interface de la classe <Path> (fichier Path.h) ----------------
10. #if ! defined ( PATH\_H )
11. #define PATH\_H
13. //--------------------------------------------------- Interfaces utilisées
15. #include <iostream>
17. //------------------------------------------------------------------ Types
18. // Note : Nous avons inversé les catégories types et constantes pour pouvoir faire les déclarations
20. **enum** MeansOfTransport { TRAIN, AUTO, BATEAU, AVION, END\_DELIMITER };
22. //------------------------------------------------------------- Constantes
24. **const** **int** MEAN\_OF\_TRANSPORT\_STRING\_MAX\_SIZE = 15;
25. **const** **int** MEAN\_OF\_TRANSPORT\_QTY = END\_DELIMITER;
27. **const** **char** MEAN\_OF\_TRANSPORT\_STRINGS[MEAN\_OF\_TRANSPORT\_QTY][MEAN\_OF\_TRANSPORT\_STRING\_MAX\_SIZE] = { "Train", "Auto", "Bateau", "Avion"};
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. //------------------------------------------------------------------------
37. **class** Path
38. {
39. //----------------------------------------------------------------- PUBLIC
41. **public**:
42. //----------------------------------------------------- Méthodes publiques
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
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
54. **virtual** Path \* Clone() **const** = 0;
55. // Mode d'emploi :
56. // Dublique le trajet
57. // Voir les classes dérivées pour la réalisation concrète
58. * 1. Réalisation (Path.cpp)
59. //------------------------------------------------- Surcharge d'opérateurs
61. **friend** **bool** operator ==(**const** Path& first, **const** Path& second);
62. // implémentation tirée de :
63. // https://stackoverflow.com/questions/1691007/whats-the-right-way-to-overload-operator-for-a-class-hierarchy
64. // https://stackoverflow.com/questions/9287704/is-there-an-idiomatic-approach-in-c-for-comparing-polymorphic-types-for-object?noredirect=1&lq=1
66. **friend** std::ostream& operator<<(std::ostream& os, **const** Path& path);
67. // Mode d'emploi :
68. // Affiche les données formatées en utilisant la fonction print
70. //-------------------------------------------- Constructeurs - destructeur
72. **protected**:
73. Path();
74. // Mode d'emploi :
75. // Constructeur par défaut (vide)
77. **public**:
78. **virtual** ~Path();
79. // Mode d'emploi :
80. // Destructeur par défaut (vide)
82. //------------------------------------------------------------------ PRIVE
84. **protected**:
85. //----------------------------------------------------- Méthodes protégées
87. **virtual** std::ostream& print(std::ostream& os) **const** = 0;
88. // Mode d'emploi :
89. // Méthode permettant l'affichage de la classe Path sur un flux (os).
90. // Voir les classes dérivées pour la réalisation concrète
92. //----------------------------------------------------- Attributs protégés
94. **private**:
95. //----------------------------------------------------- Méthodes privées
96. **virtual** **bool** equals(**const** Path& other) **const** = 0;
98. };
100. //-------------------------------- Autres définitions dépendantes de <Path>

1. //---------- Réalisation de la classe <Path> (fichier Path.cpp) ------------
3. //---------------------------------------------------------------- INCLUDE
5. //-------------------------------------------------------- Include système
7. #include <iostream>
9. **using** std::cout;
10. **using** std::endl;
12. //------------------------------------------------------ Include personnel
14. #include "Path.h"
16. //------------------------------------------------------------- Constantes
18. //----------------------------------------------------------------- PUBLIC
20. //----------------------------------------------------- Méthodes publiques
22. //------------------------------------------------- Surcharge d'opérateurs
24. **bool** operator==(**const** Path& first, **const** Path& second)
25. {
26. // RTTI check
27. **if** (**typeid**(first) != **typeid**(second))
28. **return** **false**;
30. // Invoke is\_equal on derived types
31. **return** first.equals(second);
32. } //----- Fin de ==
34. std::ostream & operator<<(std::ostream & os, **const** Path & path)
35. {
36. **return** path.print(os);
37. } //----- Fin de <<
39. //-------------------------------------------- Constructeurs - destructeur
41. Path::Path()
42. {
43. #ifdef MAP
44. cout << "Appel au constructeur de <Path>" << endl;
45. #endif
46. } //----- Fin de Path
48. Path::~Path()
49. {
50. #ifdef MAP
51. cout << "Appel au destructeur de <Path>" << endl;
52. #endif
53. } //----- Fin de ~Path
55. //------------------------------------------------------------------ PRIVE
57. //----------------------------------------------------- Méthodes protégées
    1. SimplePath
       1. Fichier d’en-tête (SimplePath.h)
58. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
59. SimplePath  -  Trajet Simple entre deux villes
60. -------------------
61. début                : 27/09/2018
62. copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
63. e-mail               : ...@insa-lyon.fr
64. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
66. //---------- Interface de la classe <SimplePath> (fichier SimplePath.h) ----------------
67. #if ! defined ( SIMPLEPATH\_H )
68. #define SIMPLEPATH\_H
70. //--------------------------------------------------- Interfaces utilisées
72. #include "../Path/Path.h"
74. //------------------------------------------------------------- Constantes
76. //------------------------------------------------------------------ Types
78. //------------------------------------------------------------------------
79. // Rôle de la classe <SimplePath>
80. // Représente un trajet simple entre une ville de départ et une ville
81. // d'arrivée par un unique moyen de transport.
82. //------------------------------------------------------------------------
84. **class** SimplePath : **public** Path
85. {
86. //----------------------------------------------------------------- PUBLIC
88. **public**:
89. //----------------------------------------------------- Méthodes publiques
91. **virtual** **char** \* GetStartCity() **const**;
92. // Mode d'emploi :
93. //      Getter de "startCity"
94. //
95. // Contrat :
96. //
98. **virtual** **char** \* GetEndCity() **const**;
99. // Mode d'emploi :
100. //      Getter de "endCity"
101. //
102. // Contrat :
103. //
105. **virtual** SimplePath \* Clone() **const**;
106. // Mode d'emploi :
107. //      Duplique le trajet
108. //
109. // Contrat :
110. //

1. //------------------------------------------------- Surcharge d'opérateurs
3. SimplePath & operator=(SimplePath other);
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. //
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. //
20. **virtual** ~SimplePath();
21. // Mode d'emploi
22. // Destructeur basique qui désalloue startCity & endCity
23. //
24. // Contrat :
25. //
27. //------------------------------------------------------------------ PRIVE
29. **protected**:
30. //----------------------------------------------------- Méthodes protégées
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. //
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. //
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. * 1. Réalisation (SimplePath.cpp)
58. //----------------------------------------------------- Attributs protégés
60. **char** \* startCity;
61. **char** \* endCity;
63. MeansOfTransport meanOfTransport;
65. };
67. //-------------------------------- Autres définitions dépendantes de <SimplePath>
69. #endif // 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. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
9. //---------- Réalisation de la classe <SimplePath> (fichier SimplePath.cpp) ------------
11. //---------------------------------------------------------------- INCLUDE
13. //-------------------------------------------------------- Include système
14. #include <iostream>
15. #include <cstring>
17. **using** std::cout;
18. **using** std::endl;
20. //------------------------------------------------------ Include personnel
21. #include "SimplePath.h"
23. //------------------------------------------------------------- Constantes
25. //----------------------------------------------------------------- PUBLIC
27. //----------------------------------------------------- Méthodes publiques
29. **char** \* SimplePath::GetStartCity() **const**
30. {
31. **return** startCity;
32. } //----- Fin de StartFrom
34. **char** \* SimplePath::GetEndCity() **const**
35. {
36. **return** endCity;
37. } //----- Fin de StopAt

1. SimplePath\* SimplePath::Clone() **const**
2. {
3. **return** **new** SimplePath(\***this**);
4. } //----- Fin de Clone
6. //------------------------------------------------- Surcharge d'opérateurs
8. SimplePath & SimplePath::operator=(SimplePath other)
9. {
10. swap(\***this**, other);
12. **return** \***this**;
13. } //----- Fin de =
15. //-------------------------------------------- Constructeurs - destructeur
17. SimplePath::SimplePath(**const** SimplePath & other)
18. {
19. startCity = **new** **char**[strlen(other.startCity)+1];
20. strcpy(startCity, other.startCity);
22. endCity = **new** **char**[strlen(other.endCity)+1];
23. strcpy(endCity, other.endCity);
25. meanOfTransport = other.meanOfTransport;
27. #ifdef MAP
28. cout << "Appel au constructeur de copie de <SimplePath>" << endl;
29. #endif
30. } //----- Fin de SimplePath (constructeur de copie)

33. SimplePath::SimplePath(**char** \* startingCity, **char** \* endingCity, MeansOfTransport vehicle) : meanOfTransport(vehicle)
34. {
35. startCity = **new** **char**[strlen(startingCity)+1];
36. strcpy(startCity, startingCity);
38. endCity = **new** **char**[strlen(endingCity)+1];
39. strcpy(endCity, endingCity);
41. #ifdef MAP
42. cout << "Appel au constructeur de <SimplePath>" << endl;
43. #endif
44. } //----- Fin de SimplePath

47. SimplePath::~SimplePath()
48. {
49. **delete**[] startCity;
50. **delete**[] endCity;
52. #ifdef MAP
53. cout << "Appel au destructeur de <SimplePath>" << endl;
54. #endif
55. } //----- Fin de ~SimplePath
    1. ComposedPath
56. //------------------------------------------------------------------ PRIVE
58. //----------------------------------------------------- Méthodes protégées
60. **bool** SimplePath::equals(**const** Path& other) **const**
61. {
62. **const** SimplePath& other\_derived = **dynamic\_cast**<**const** SimplePath&>(other);
64. **return** (strcmp(startCity, other\_derived.GetStartCity()) == 0)
65. && (strcmp(endCity, other\_derived.GetEndCity()) == 0)
66. && meanOfTransport == other\_derived.meanOfTransport;
67. } //----- Fin de equals
69. std::ostream& SimplePath::print(std::ostream& os) **const**
70. {
71. os  << "Trajet Simple :"                                              << endl
72. << "\tDépart    : " << startCity                                    << endl
73. << "\tArrivée   : " << endCity                                      << endl
74. << "\tTransport : " << MEAN\_OF\_TRANSPORT\_STRINGS[meanOfTransport]   << endl;
76. **return** os;
77. } //----- Fin de print
79. **void** swap(SimplePath& first, SimplePath& second)
80. {
81. **char** \* tmp = **new** **char**[strlen(first.GetStartCity()) + 1];
82. strcpy(tmp, first.GetStartCity());
84. **delete** [] first.GetStartCity();
85. first.startCity = **new** **char**[strlen(second.GetStartCity()) + 1];
86. strcpy(first.startCity, second.GetStartCity());
88. **delete** [] second.GetStartCity();
89. second.startCity = tmp;
91. **char** \* tmpE = **new** **char**[strlen(first.GetEndCity()) + 1];
92. strcpy(tmpE, first.GetEndCity());
94. **delete** [] first.GetEndCity();
95. first.endCity = **new** **char**[strlen(second.GetEndCity()) + 1];
96. strcpy(first.endCity, second.GetEndCity());
98. **delete** [] second.GetEndCity();
99. second.endCity = tmpE;
101. MeansOfTransport tmpMot = second.meanOfTransport;
102. second.meanOfTransport = first.meanOfTransport;
103. first.meanOfTransport = tmpMot;
104. } //----- Fin de swap
     * 1. Fichier d’en-tête (ComposedPath.h)
105. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
106. ComposedPath  -  description
107. -------------------
108. début                : 27/09/2018
109. copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
110. e-mail               : ...@insa-lyon.fr
111. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
113. //---------- Interface de la classe <ComposedPath> (fichier ComposedPath.h) ----------------
114. #if ! defined ( COMPOSEDPATH\_H )
115. #define COMPOSEDPATH\_H
117. //--------------------------------------------------- Interfaces utilisées
118. #include "../Path/Path.h"
119. #include "../PathArray/PathArray.h"
120. //------------------------------------------------------------- Constantes
122. //------------------------------------------------------------------ Types
124. //------------------------------------------------------------------------
125. // Rôle de la classe <ComposedPath>
126. // Représente un trajet composé de plusieurs sous-trajets (simples ou eux-
127. // mêmes composés)
128. //------------------------------------------------------------------------
130. **class** ComposedPath : **public** Path
131. {
132. //----------------------------------------------------------------- PUBLIC
134. **public**:
135. //----------------------------------------------------- Méthodes publiques
136. PathArray\* GetElements ( ) **const**;
137. // Mode d'emploi :
138. //      Retourne la collection de Path composant le trajet
139. // Contrat :
140. //
142. **void** AddStage(Path \*path) **const**;
143. // Mode d'emploi :
144. //      Ajoute une étape (trajet) au trajet composé
145. //      > path : pointeur sur le trajet à ajouter
146. // Contrat :
147. //      path est un pointeur sur trajet valide
149. **virtual** **char** \* GetStartCity() **const**;
150. // Mode d'emploi :
151. //      Getter de "startCity"
152. // Contrat :
153. //
155. **virtual** **char** \* GetEndCity() **const**;
156. // Mode d'emploi :
157. //      Getter de "endCity"
158. // Contrat :
159. //
161. **virtual** ComposedPath\* Clone() **const**;
162. // Mode d'emploi :
163. //      Duplique le trajet
164. // Contrat :
165. //
     * 1. Réalisation (ComposedPath.cpp)
166. //------------------------------------------------- Surcharge d'opérateurs
167. ComposedPath& operator=(ComposedPath other);

170. //-------------------------------------------- Constructeurs - destructeur
171. ComposedPath ( **const** ComposedPath & other );
172. // Mode d'emploi (constructeur de copie) :
173. //
174. // Contrat :
175. //
177. ComposedPath ( unsigned **int** maxSize = MAX\_SIZE );
178. // Mode d'emploi :
179. //      > maxSize : taille maximum initiale de la collection de trajets,
180. // par defaut égale à MAX\_SIZE
181. // Contrat :
182. //
184. **virtual** ~ComposedPath ( );
185. // Mode d'emploi :
186. //
187. // Contrat :
188. //
190. //------------------------------------------------------------------ PRIVE
192. **protected**:
193. //----------------------------------------------------- Méthodes protégées
194. **virtual** std::ostream& print(std::ostream& os) **const**;
195. // Mode d'emploi :
196. //      Ecrit une représentation de l'objet en chaîne de caractères sur un
197. // flux standard
198. //      > os : flux standard sur lequel on écrit
199. // Contrat :
200. //
201. **virtual** **bool** equals(**const** Path& other) **const**;
202. // Mode d'emploi :
203. //      Compare si le ComposedPath est égal à un autre trajet.
204. // Renvoie true si les trajets sont égaux, faux sinon.
205. // Contrat :
206. //
207. **friend** **void** swap(ComposedPath& first, ComposedPath& second);
208. // Mode d'emploi :
209. //      Echange les valeurs des attributs entre 2 objets ComposedPath
210. //      > first : Premier objet de l'échange
211. //      > second : Second objet de l'échange
212. // Contrat :
213. //
214. //----------------------------------------------------- Attributs protégés
216. PathArray\* elements;
218. };
220. //-------------------------------- Autres définitions dépendantes de <ComposedPath>
222. #endif // COMPOSEDPATH\_H
223. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
224. ComposedPath  -  trajet composé
225. -------------------
226. début                : 27/09/2018
227. copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
228. e-mail               : ...@insa-lyon.fr
229. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
231. //---------- Réalisation de la classe <ComposedPath> (fichier ComposedPath.cpp) ------------
233. //---------------------------------------------------------------- INCLUDE
235. //-------------------------------------------------------- Include système
236. #include <iostream>
238. **using** std::cout;
239. **using** std::endl;
240. //------------------------------------------------------ Include personnel
241. #include "ComposedPath.h"
242. //------------------------------------------------------------- Constantes
243. //----------------------------------------------------------------- PUBLIC
245. //----------------------------------------------------- Méthodes publiques
246. PathArray\* ComposedPath::GetElements ( ) **const**
247. // Algorithme :
248. {
249. **return** elements;
250. } //----- Fin de GetElements
252. **void** ComposedPath::AddStage(Path \*path) **const**
253. // Algorithme :
254. {
255. elements->Add(path);
256. } //----- Fin de AddStage
258. **char** \* ComposedPath::GetStartCity() **const**
259. {
260. **return** elements->Get(0)->GetStartCity();
261. } //----- Fin de StartFrom
263. **char** \* ComposedPath::GetEndCity() **const**
264. {
265. **return** elements->Get(elements->GetSize() - 1)->GetEndCity();
266. } //----- Fin de StopAt
268. ComposedPath\* ComposedPath::Clone() **const**
269. {
270. **return** **new** ComposedPath(\***this**);
271. } //----- Fin de Clone
273. //------------------------------------------------- Surcharge d'opérateurs
274. ComposedPath& ComposedPath::operator=(ComposedPath other)
275. {
276. swap(\***this**, other);
278. **return** \***this**;
279. } //----- Fin de =
     1. PathArray
280. //-------------------------------------------- Constructeurs - destructeur
281. ComposedPath::ComposedPath ( **const** ComposedPath & other )
282. // Algorithme :
283. //
284. {
285. #ifdef MAP
286. cout << "Appel au constructeur de copie de <ComposedPath>" << endl;
287. #endif
288. elements = **new** PathArray();
289. \*elements = \*other.elements;
290. } //----- Fin de ComposedPath (constructeur de copie)

293. ComposedPath::ComposedPath ( unsigned **int** maxSize)
294. // Algorithme :
295. //
296. {
297. #ifdef MAP
298. cout << "Appel au constructeur de <ComposedPath>" << endl;
299. #endif
300. elements = **new** PathArray(maxSize);
301. } //----- Fin de ComposedPath

304. ComposedPath::~ComposedPath ( )
305. // Algorithme :
306. //
307. {
308. #ifdef MAP
309. cout << "Appel au destructeur de <ComposedPath>" << endl;
310. #endif
311. **delete** elements;
312. } //----- Fin de ~ComposedPath

315. //------------------------------------------------------------------ PRIVE
317. //----------------------------------------------------- Méthodes protégées
318. **bool** ComposedPath::equals(**const** Path& other) **const**
319. {
320. **const** ComposedPath& other\_derived = **dynamic\_cast**<**const** ComposedPath&>(other);
321. **return** elements->Equals(\*other\_derived.elements);
322. } //----- Fin de equals
324. std::ostream& ComposedPath::print(std::ostream& os) **const**
325. {
326. os << "Trajet Composé :" << endl;
328. **return** elements->Print(os, **false**, **true**);
329. } //----- Fin de print
331. **void** swap(ComposedPath& first, ComposedPath& second)
332. {
333. PathArray tmp = \*(first.elements);
334. \*(first.elements) = \*(second.elements);
335. \*(second.elements) = tmp;
336. } //----- Fin de swap
     * 1. Fichier d’en-tête (PathArray.h)
337. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
338. PathArray  -  description
339. -------------------
340. début                : 27/09/2018
341. copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
342. e-mail               : ...@insa-lyon.fr
343. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
345. //---------- Interface de la classe <PathArray> (fichier PathArray.h) ----------------
346. #if ! defined ( PATHARRAY\_H )
347. #define PATHARRAY\_H
349. //--------------------------------------------------- Interfaces utilisées
350. #include <iostream>
351. #include "../Path/Path.h"
353. //------------------------------------------------------------- Constantes
354. **const** **int** MAX\_SIZE = 10;
356. //------------------------------------------------------------------ Types
358. //------------------------------------------------------------------------
359. // Rôle de la classe <PathArray>
360. // Collection permettant la gestion dynamique de trajets (Path)
361. //------------------------------------------------------------------------
363. **class** PathArray
364. {
365. //----------------------------------------------------------------- PUBLIC
367. **public**:
368. //----------------------------------------------------- Méthodes publiques
369. **bool** Equals(**const** PathArray & otherPathArray) **const**;
370. // Mode d'emploi :
371. //      Compare le contenu de la collection avec une autre collection PathArray.
372. // Renvoie true si les collections sont égales, false sinon.
373. //      > otherPathArray : collection à comparer
374. // Contrat :
375. //
376. **bool** Add(Path\* pathToAdd);
377. // Mode d'emploi :
378. //      Ajoute un trajet à la collection
379. // Renvoie false si un trajet similaire est déjà présent dans la collection,
380. // true sinon.
381. //      > pathToAdd : trajet à ajouter
382. // Contrat :
383. //      pathToAdd pointe vers un objet Path valide
384. unsigned **int** Adjust(**int** delta);
385. // Mode d'emploi :
386. //      Ajuste la taille de la collection.
387. //      > delta : offset de la taille de la collection. Si delta > 0, la taille de la
388. // collection s'agrandit de delta. Sinon, la taille de la collection se réduit de la valeur
389. // de delta, dans la limite du nombre d'éléments déjà présent.
390. // Contrat :
391. //

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
39. std::ostream& Print(std::ostream& os, **bool** advanced = **false**, **bool** composed = **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. //

48. //------------------------------------------------- Surcharge d'opérateurs
49. PathArray& operator=(**const** PathArray & other);

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. Réalisation (PathArray.cpp)
59. PathArray(**const** unsigned **int** maxSize = MAX\_SIZE);
60. // Mode d'emploi :
61. // Constructeur basique qui alloue un tableau de taille "cardMax"
62. //
63. // maxSize :
64. // Taille maximal de la collection sans réallocation (par défaut : MAX\_SIZE)
65. //
66. // Contrat :
67. //
69. **virtual** ~PathArray();
70. // Mode d'emploi :
71. // Destructeur basique qui désalloue le tableau elements
72. //
73. // Contrat :
74. //
76. //------------------------------------------------------------------ PRIVE
78. **protected**:
79. //----------------------------------------------------- Méthodes protégées
81. //----------------------------------------------------- Attributs protégés
83. Path \*\* elements;
84. unsigned **int** maxSize;
85. unsigned **int** size;
86. };
88. //-------------------------------- Autres définitions dépendantes de <PathArray>
90. #endif // PATHARRAY\_H
91. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
92. PathArray  -  description
93. -------------------
94. début                : 27/09/2018
95. copyright            : (C) 2018 par Valentin Wallyn et Balthazar Frolin
96. e-mail               : ...@insa-lyon.fr
97. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
99. //---------- Réalisation de la classe <PathArray> (fichier PathArray.cpp) ------------
101. //---------------------------------------------------------------- INCLUDE
103. //-------------------------------------------------------- Include système
104. #include <iostream>
105. **using** std::cout;
106. **using** std::endl;
107. //------------------------------------------------------ Include personnel
108. #include "PathArray.h"
109. //------------------------------------------------------------- Constantes
110. //----------------------------------------------------------------- PUBLIC
112. //----------------------------------------------------- Méthodes publiques
113. **bool** PathArray::Contains(**const** Path\* path) **const**
114. // Algorithme :
115. //
116. {
117. unsigned **int** j;
118. **for** (j = 0; j < size; j++)
119. {
120. **if** (\*elements[j] == \*path)
121. {
122. **return** **true**;
123. }
124. }
125. **return** **false**;
126. } //----- fin de Contains
128. unsigned **int** PathArray::GetSize(**void**) **const**
129. // Algorithme :
130. //
131. {
132. **return** size;
133. } //----- fin de GetSize
135. unsigned **int** PathArray::GetMaxSize(**void**) **const**
136. // Algorithme :
137. //
138. {
139. **return** maxSize;
140. } //----- fin de GetMaxSize
142. Path\* PathArray::Get(**int** index) **const**
143. // Algorithme :
144. //
145. {
146. **return** elements[index];
147. } //----- fin de GetMaxSize
149. std::ostream& PathArray::Print(std::ostream& os, **bool** advanced, **bool** composed) **const**
150. // Algorithme :
151. //
152. {
153. unsigned **int** j;
154. **if** (advanced)
155. {
156. **if** (size > 0)
157. {
158. **for** (j = 0; j < size - 1; j++)
159. {
160. os << \*elements[j] << std::endl;
161. os << "ou" << endl << endl;
162. }
163. os << \*elements[size-1];
164. }
165. }
166. **else**
167. {
168. **for** (j = 0; j < size; j++)
169. {
170. os << ((composed) ? "    Etape " : "Trajet ") << j + 1 << " - " << \*elements[j] << endl;
171. }
172. }

175. **return** os;
176. } //----- fin de Print
178. **bool** PathArray::Equals(**const** PathArray & anotherPathArray) **const**
179. // Algorithme :
180. //
181. {
182. **if** (anotherPathArray.GetSize() != size)
183. {
184. **return** **false**;
185. }
186. **else**
187. {
188. unsigned **int** j;
189. **for** (j = 0; j < size; j++)
190. {
191. **if** (!anotherPathArray.Contains(elements[j]))
192. **return** **false**;
193. }
194. }
195. **return** **true**;
196. } //----- fin de Equals
198. **bool** PathArray::Add(Path\* pathToAdd)
199. // Algorithme :
200. //
201. {
202. unsigned **int** j;
203. **for** (j = 0; j < size; j++)
204. {
206. **if** (\*elements[j] == \*pathToAdd)
207. {
208. **return** **false**;
209. }
210. }
211. **if** (size == maxSize)
212. {
213. Adjust(size);
214. }
215. elements[size] = pathToAdd;
216. size++;
218. **return** **true**;
219. } //----- fin de Add
220. unsigned **int** PathArray::Adjust(**int** delta)
221. // Algorithme :
222. //
223. {
224. **int** temp = maxSize - size;
225. **if** (delta >= 0 || -delta <= temp)
226. {
227. maxSize += delta;
228. }
229. **else**
230. {
231. maxSize = size;
232. }
234. Path\*\* newElements = **new** Path\*[maxSize];
235. unsigned **int** j;
236. **for** (j = 0; j < size; j++)
237. {
238. newElements[j] = elements[j];
239. }
240. **delete** [] elements;
241. elements = newElements;
242. **return** maxSize;
243. } //----- fin de Adjust
245. **bool** PathArray::Remove(Path\* element)
246. // Algorithme :
247. //
248. {
249. **if** (Contains(element))
250. {
251. Path\*\* newElements = **new** Path\*[size-1];
252. unsigned **int** j;
253. unsigned **int** i = 0;
254. **for** (j = 0; j < size; j++)
255. {
257. **if** (!(\*elements[j] == \*element)) //TODO we could define != operator
258. {
259. newElements[i] = elements[j];
260. i++;
261. }
262. }
263. **delete** [] elements;
264. elements = newElements;
265. size--;
266. maxSize = size;
267. **return** **true**;
268. }
269. maxSize = size;
270. **return** **false**;
271. } //----- fin de Remove
273. unsigned **int** PathArray::Remove(**const** PathArray & anotherPathArray)
274. {
275. unsigned **int** oldmaxSize = maxSize;
276. unsigned **int** j;
277. **int** count = 0;
278. **if** (Equals(anotherPathArray))
279. {
280. count = size;
281. size = 0;
282. }
283. **else**
284. {
285. **for** (j = 0; j < anotherPathArray.GetSize(); j++)
286. {
287. **if** (Remove(anotherPathArray.Get(j)))
288. {
289. count++;
290. }
291. }
292. maxSize = oldmaxSize;
293. }
295. **return** count;
296. } //----- fin de Remove
298. //------------------------------------------------- Surcharge d'opérateurs
299. PathArray& PathArray::operator=(**const** PathArray& other)
300. {
301. **if** (**this** != &other)
302. {
303. **for** (unsigned **int** j = 0; j < size; j++)
304. {
305. **delete** elements[j];
306. }
307. **delete** [] elements;
309. size = other.GetSize();
310. maxSize = other.GetMaxSize();
312. elements = **new** Path\*[maxSize];
313. **for** (unsigned **int** j = 0; j < size; j++)
314. {
315. elements[j] = other.Get(j)->Clone();
316. }
317. }
318. **return** \***this**;
319. #ifdef MAP
320. cout << "Appel à la surcharge du = de <PathArray>" << endl;
321. #endif
322. }
323. 1. SearchEngine
324. //-------------------------------------------- Constructeurs - destructeur
325. PathArray::PathArray(**const** PathArray & anotherPathArray)
326. {
327. maxSize = anotherPathArray.maxSize;
328. size = anotherPathArray.size;
330. elements = **new** Path\*[maxSize];
332. **for** (unsigned **int** j = 0; j < size; j++)
333. {
334. elements[j] = anotherPathArray.elements[j]->Clone();
335. }
337. #ifdef MAP
338. cout << "Appel au constructeur de copie de <PathArray>" << endl;
339. #endif
340. } //----- Fin de PathArray (constructeur de copie)
342. PathArray::PathArray(**const** unsigned **int** maxSize)
343. {
344. elements = **new** Path\*[maxSize];
345. **this**->maxSize = maxSize;
347. size = 0;
349. #ifdef MAP
350. cout << "Appel au constructeur de <PathArray>" << endl;
351. #endif
352. } //----- Fin de PathArray
354. PathArray::~PathArray()
355. {
356. **for** (unsigned **int** j = 0; j < size; j++)
357. {
358. **delete** elements[j];
359. }
360. **delete** [] elements;
362. #ifdef MAP
363. cout << "Appel au destructeur de <PathArray>" << endl;
364. #endif
365. } //----- Fin de ~PathArray
367. //------------------------------------------------------------------ PRIVE
369. //----------------------------------------------------- Méthodes protégées
     * 1. Fichier d’en-tête (SearchEngine.h)
370. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
371. SearchEngine  -  description
372. -------------------
373. début                : 04/10/2018
374. copyright            : (C) 2018 par WALLYN Valentin - FROLIN Balthazar
375. e-mail               : $EMAIL$
376. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
378. //---------- Interface de la classe <SearchEngine> (fichier SearchEngine.h) ----------------
379. #if ! defined ( SEARCHENGINE\_H )
380. #define SEARCHENGINE\_H
382. //--------------------------------------------------- Interfaces utilisées
383. #include "../Path/Path.h"
384. #include "../PathArray/PathArray.h"
385. //------------------------------------------------------------- Constantes
386. **const** **int** DEFAULT\_MAX\_SIZE = 10;
387. //------------------------------------------------------------------ Types
388. **struct** node
389. {
390. unsigned **int** startIndex;
391. unsigned **int** endIndex;
392. node\* previous;
393. node\* next;
394. };
396. //------------------------------------------------------------------------
397. // Rôle de la classe <SearchEngine>
398. //
399. // Le search engine permet de chercher des trajets entre 2 villes dans tout
400. // le catalogue avec ou sans composition.
401. //
402. //------------------------------------------------------------------------
404. **class** SearchEngine
405. {
406. //----------------------------------------------------------------- PUBLIC
408. **public**:
409. //----------------------------------------------------- Méthodes publiques

412. **void** SimpleSearch(**const** **char** \* startingCity, **const**  **char** \* endingCity) **const**;
413. // Mode d'emploi :
414. //      Cherche et affiche le(s) trajet(s) directs correspondant(s) du catalogue pour
415. // se rendre de la ville "startingCity" à la ville "endingCity"
416. //      > startingCity : ville de départ
417. //      > endingCity : ville d'arrivée
418. // Contrat :
419. //
421. **void** AdvancedSearch(**const** **char** \* startingCity,**const**  **char** \* endingCity) **const**;
422. // Mode d'emploi :
423. //      Cherche et affiche le(s) trajet(s) pour se rendre de la
424. // ville "startingCity" à la ville "endingCity" en effectuant une
425. // composition de plusieurs sous-trajets du catalogue
426. //      > startingCity : ville de départ
427. //      > endingCity : ville d'arrivée
428. // Contrat :
429. //
430. **void** AddPath(Path\* path);
431. // Mode d'emploi :
432. //      Ajoute un trajet au moteur de recherche
433. //      > path : pointeur sur le trajet à ajouter
434. // Contrat :
435. //      path est un pointeur sur trajet valide


439. //------------------------------------------------- Surcharge d'opérateurs
440. SearchEngine & operator = ( **const** SearchEngine & other );
441. // Mode d'emploi :
442. //
443. // Contrat :
444. //

447. //-------------------------------------------- Constructeurs - destructeur
448. SearchEngine ( **const** SearchEngine & other );
449. // Mode d'emploi (constructeur de copie) :
450. //
451. // Contrat :
452. //
454. SearchEngine ( **const** unsigned **int** size = DEFAULT\_MAX\_SIZE);
455. // Mode d'emploi :
456. //
457. // Contrat :
458. //
460. **virtual** ~SearchEngine ( );
461. // Mode d'emploi :
462. //
463. // Contrat :
464. //
466. //------------------------------------------------------------------ PRIVE
468. **private**:
469. //----------------------------------------------------- Méthodes protégées
471. unsigned **int** getStartCityIndex(**const** **char** \* city) **const**;
472. unsigned **int** getEndCityIndex(**const** **char** \* city) **const**;
473. **bool** recursiveSearch(node\* node, unsigned **int** endIndex, **bool** \* doneIndex) **const**;
475. //----------------------------------------------------- Attributs protégés
477. PathArray\*\*\* adjacencyMatrix;
478. **char** \*\* startCities;
479. **char** \*\* endCities;
481. unsigned **int** startCurrentSize;
482. unsigned **int** endCurrentSize;
483. unsigned **int** startMaxSize;
484. unsigned **int** endMaxSize;
486. };
488. //-------------------------------- Autres définitions dépendantes de <SearchEngine>
490. #endif // SEARCHENGINE\_H
491. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
492. SearchEngine  -  description
493. -------------------
494. début                : 04/10/2018
495. copyright            : (C) 2018 par WALLYN Valentin - FROLIN Balthazar
496. e-mail               : $EMAIL$
497. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
499. //---------- Réalisation de la classe <SearchEngine> (fichier SearchEngine.cpp) ------------
501. //---------------------------------------------------------------- INCLUDE
503. //-------------------------------------------------------- Include système
504. #include <iostream>
505. #include <cstring>
507. **using** **namespace** std;
509. //------------------------------------------------------ Include personnel
510. #include "SearchEngine.h"
512. //------------------------------------------------------------- Constantes
514. //----------------------------------------------------------------- PUBLIC
516. //----------------------------------------------------- Méthodes publiques
518. **void** SearchEngine::SimpleSearch(**const** **char** \* startingCity, **const** **char** \* endingCity) **const**
519. // Algorithme :
520. //      Recherche dans la matrice de proximité si un trajet direct existe entre startingCity
521. // et endingCity. Si oui, affiche le trajet, sinon affiche qu'aucun trajet n'a été trouvé.
522. {
523. unsigned **int** startIndex = getStartCityIndex(startingCity);
524. unsigned **int** endIndex = getEndCityIndex(endingCity);
526. **if** (startIndex != startCurrentSize && endIndex != endCurrentSize && adjacencyMatrix[startIndex][endIndex]->GetSize() > 0)
527. {
528. adjacencyMatrix[startIndex][endIndex]->Print(cout);
529. }
530. **else**
531. {
532. cout << "\tAucun trajet correspondant n'a été trouvé." << endl;
533. }
535. } //----- Fin de simpleSearch

538. **void** SearchEngine::AdvancedSearch(**const** **char** \* startingCity, **const** **char** \* endingCity) **const**
539. // Algorithme :
540. //      Instancie le noeud initial de la recherche récursive puis apelle la fonction de recherche
541. // récursive. Voir la fonction "recursiveSearch" pour l'algorithme de recherche.
542. {
543. unsigned **int** startIndex = getStartCityIndex(startingCity);
544. unsigned **int** endIndex = getEndCityIndex(endingCity);
545. **if** (startIndex != startCurrentSize && endIndex != endCurrentSize)
546. {
547. SearchEngine::node node;
548. node.startIndex = startIndex;
549. node.previous = NULL;
551. **bool**\* doneIndex = **new** **bool**[startCurrentSize];
552. **for** (unsigned **int** j = 0; j < startCurrentSize; j++)
553. {
554. doneIndex[j] = **false**;
555. }
556. recursiveSearch(&node, endIndex, doneIndex);
557. **delete** [] doneIndex;
558. }
559. **else**
560. {
561. cout << "\tAucun trajet correspondant n'a été trouvé." << endl;
562. }
563. } //----- Fin de advancedSearch

566. **void** SearchEngine::AddPath(Path\* path)
567. // Algorithme :
568. //      Ajoute un trajet à la matrice de proximité en effectuant les réallocations
569. // de mémoire nécessaires lorsque les tableaux sont pleins. Stocke les villes de départ
570. // et d'arrivée de manière unique
571. {
572. unsigned **int** startCityIndex = getStartCityIndex(path->GetStartCity());
573. unsigned **int** endCityIndex = getEndCityIndex(path->GetEndCity());
575. **if** (startCityIndex == startCurrentSize)
576. {
578. startCities[startCityIndex] = **new** **char**[strlen(path->GetStartCity()) + 1];
579. strcpy(startCities[startCityIndex], path->GetStartCity());
580. startCurrentSize++;
581. **if** (startCurrentSize == startMaxSize)
582. {
583. **char** \*\* tmpCities = **new** **char**\*[startMaxSize \* 2];
584. **for** (unsigned **int** j = 0; j < startCurrentSize; j++)
585. {
586. tmpCities[j] = **new** **char**[strlen(startCities[j]) + 1];
587. strcpy(tmpCities[j], startCities[j]);
588. }
589. **for** (unsigned **int** j = 0; j < startCurrentSize; j++)
590. {
591. **delete** [] startCities[j];
592. }
594. **delete** [] startCities;
595. startCities = tmpCities;

598. PathArray\*\*\* tmp = **new** PathArray\*\*[startMaxSize \* 2];
599. **for** (unsigned **int** j = 0; j < startMaxSize \* 2; j++)
600. {
601. tmp[j] = **new** PathArray\*[endMaxSize];
602. **for** (unsigned **int** i = 0; i < endMaxSize; i++)
603. {
604. tmp[j][i] = **new** PathArray();
605. **if** (i < endCurrentSize && j < startCurrentSize - 1)
606. {
607. \*tmp[j][i] = \*adjacencyMatrix[j][i];
608. }
609. }
610. }
611. **for** (unsigned **int** j = 0; j < startMaxSize; j++)
612. {
613. **for** (unsigned **int** i = 0; i < endMaxSize; i++)
614. {
615. **delete** adjacencyMatrix[j][i];
616. }
617. **delete** [] adjacencyMatrix[j];
618. }
619. startMaxSize = startMaxSize \* 2;
620. **delete** [] adjacencyMatrix;
621. adjacencyMatrix = tmp;
622. }
623. }
625. **if** (endCityIndex == endCurrentSize)
626. {
627. endCities[endCityIndex] = **new** **char**[strlen(path->GetEndCity()) + 1];
628. strcpy(endCities[endCityIndex], path->GetEndCity());
629. endCurrentSize++;
630. **if** (endCurrentSize == endMaxSize)
631. {
632. **char** \*\* tmpCities = **new** **char**\*[endMaxSize \* 2];
633. **for** (unsigned **int** j = 0; j < endCurrentSize; j++)
634. {
635. tmpCities[j] = **new** **char**[strlen(endCities[j]) + 1];
636. strcpy(tmpCities[j], endCities[j]);
637. }
638. **for** (unsigned **int** j = 0; j < endCurrentSize; j++)
639. {
640. **delete** [] endCities[j];
641. }
643. **delete** [] endCities;
644. endCities = tmpCities;


648. PathArray\*\*\* tmp = **new** PathArray\*\*[startMaxSize];
649. **for** (unsigned **int** j = 0; j < startMaxSize; j++)
650. {
651. tmp[j] = **new** PathArray\*[endMaxSize \* 2];
652. **for** (unsigned **int** i = 0; i < endMaxSize \* 2; i++)
653. {
654. tmp[j][i] = **new** PathArray();
655. **if** (i < endCurrentSize - 1 && j < startCurrentSize)
656. {
657. \*tmp[j][i] = \*adjacencyMatrix[j][i];
658. }
659. }
660. }
661. **for** (unsigned **int** j = 0; j < startMaxSize; j++)
662. {
663. **for** (unsigned **int** i = 0; i < endMaxSize; i++)
664. {
665. **delete** adjacencyMatrix[j][i];
666. }
667. **delete** [] adjacencyMatrix[j];
668. }
669. endMaxSize = endMaxSize \* 2;
670. **delete** [] adjacencyMatrix;
671. adjacencyMatrix = tmp;
672. }
674. }
676. adjacencyMatrix[startCityIndex][endCityIndex]->Add(path->Clone());
678. } //----- Fin de addPath
680. //------------------------------------------------- Surcharge d'opérateurs
681. SearchEngine & SearchEngine::operator = ( **const** SearchEngine & other )
682. // Algorithme :
683. //
684. {
685. **return** \***this**;
686. } //----- Fin de operator =

689. //-------------------------------------------- Constructeurs - destructeur
690. SearchEngine::SearchEngine ( **const** SearchEngine & other )
691. // Algorithme :
692. //
693. {
694. #ifdef MAP
695. cout << "Appel au constructeur de copie de <SearchEngine>" << endl;
696. #endif
697. } //----- Fin de SearchEngine (constructeur de copie)
699. SearchEngine::SearchEngine( **const** unsigned **int** size )
700. // Algorithme :
701. //
702. {
703. #ifdef MAP
704. cout << "Appel au constructeur de <SearchEngine>" << endl;
705. #endif
706. adjacencyMatrix = **new** PathArray\*\*[size];
707. **for** (unsigned **int** j = 0; j < size; j++)
708. {
709. adjacencyMatrix[j] = **new** PathArray\*[size];
710. **for** (unsigned **int** i = 0; i < size; i++)
711. {
712. adjacencyMatrix[j][i] = **new** PathArray();
713. }
714. }
715. startCities = **new** **char**\*[size];
716. endCities = **new** **char**\*[size];
717. startCurrentSize = 0;
718. startMaxSize = size;
719. endCurrentSize = 0;
720. endMaxSize = size;
722. } //----- Fin de SearchEngine

725. SearchEngine::~SearchEngine ( )
726. // Algorithme :
727. //
728. {
729. #ifdef MAP
730. cout << "Appel au destructeur de <SearchEngine>" << endl;
731. #endif
732. **for** (unsigned **int** j = 0; j < startMaxSize; j++)
733. {
734. **for** (unsigned **int** i = 0; i < endMaxSize; i++)
735. {
736. **delete** adjacencyMatrix[j][i];
737. }
738. **delete** [] adjacencyMatrix[j];
739. }
740. **delete** [] adjacencyMatrix;
742. **for** (unsigned **int** j = 0; j < startCurrentSize; j++)
743. {
744. **delete** [] startCities[j];
745. }
747. **for** (unsigned **int** j = 0; j < endCurrentSize; j++)
748. {
749. **delete** [] endCities[j];
750. }
751. **delete** [] startCities;
752. **delete** [] endCities;
753. } //----- Fin de ~SearchEngine

756. //------------------------------------------------------------------ PRIVE
758. //----------------------------------------------------- Méthodes protégées
759. unsigned **int** SearchEngine::getStartCityIndex(**const** **char** \* city) **const**
760. // Algorithme :
761. //      Parcoure la liste des villes de départ enregistrées et récupère l'index
762. // de la ville "city"
763. {
764. **for** (unsigned **int**  j = 0; j < startCurrentSize; j++)
765. {
766. **if** (strcmp(startCities[j], city) == 0)
767. {
768. **return** j;
769. }
770. }
771. **return** startCurrentSize;
772. } //----- Fin de getStartCityIndex
774. unsigned **int** SearchEngine::getEndCityIndex(**const** **char** \* city) **const**
775. // Algorithme :
776. //      Parcoure la liste des villes d'arrivée enregistrées et récupère l'index
777. // de la ville "city"
778. {
779. **for** (unsigned **int** j = 0; j < endCurrentSize; j++)
780. {
781. **if** (strcmp(endCities[j], city) == 0)
782. {
783. **return** j;
784. }
785. }
786. **return** endCurrentSize;
787. } //----- Fin de getEndCityIndex
789. **bool** SearchEngine::recursiveSearch(node\* node, unsigned **int** endIndex, **bool** \* doneIndex) **const**
790. // Algorithme :
791. //      Explore les noeuds reliés au noeud initial "node" dans la matrice de proximité récursivement
792. // jusqu'à l'arrivée sur un noeud possèdant comme ville d'arrivée la destination objectif (endIndex)
793. // ou que tous les noeuds de départs aient été explorés (stockés dans doneIndex)
794. // Si un trajet a été trouvé, affiche la composition de celui-ci en parcourant les noeuds solution
795. {
796. **for** (unsigned **int** j = 0; j < endCurrentSize; j++)
797. {
798. node->endIndex = j;
800. **if** (adjacencyMatrix[node->startIndex][j]->GetSize() > 0)
801. {
802. **if** (j == endIndex)
803. {
804. node->next = NULL;
805. **while** (node->previous != NULL)
806. {
807. node = node->previous;
808. }
809. cout << "Début trajet :" << endl;
     1. Module main
810. //---------- Réalisation du module <TCatalog> (fichier TCatalog.cpp) ---------------
812. /////////////////////////////////////////////////////////////////  INCLUDE
813. //-------------------------------------------------------- Include système
815. #include <iostream>
817. //------------------------------------------------------ Include personnel
819. #include "Catalog/Catalog.h"
821. **using** **namespace** std;
823. //////////////////////////////////////////////////////////////////  PUBLIC
824. //---------------------------------------------------- Fonctions publiques
826. **int** main()
827. {
828. Catalog catalog = Catalog();
829. catalog.Run();
831. **return** 0;
832. }
833. **while** (node->next != NULL)
834. {
835. adjacencyMatrix[node->startIndex][node->endIndex]->Print(cout, **true**);
836. cout << "puis" << endl;
837. node = node->next;
838. }
839. adjacencyMatrix[node->startIndex][node->endIndex]->Print(cout, **true**);
840. cout << "Fin trajet" << endl << endl;
841. **return** **true**;
842. }
843. unsigned **int** startIndex = getStartCityIndex(endCities[j]);
844. **if** (startIndex != startCurrentSize)
845. {
846. **if** (!doneIndex[startIndex])
847. {
848. doneIndex[node->startIndex] = **true**;
849. SearchEngine::node next;
850. next.previous = node;
851. next.startIndex = startIndex;
853. node->next = &next;
854. recursiveSearch(&next, endIndex, doneIndex);
855. }
856. }
858. }
859. }
860. **return** **false**;
861. } //----- Fin de recursiveSearch

**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.txt

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 »

1. **Exécution**

* Version de production/debug : Exécuter « .Build/VoyageVoyage »

Version de test : Exécuter « .Build/T<Module> »

1. Conclusion
   1. Problèmes rencontrés
      1. 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.

* + 1. 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 cette 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.

* 1. 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.

* + 1. 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 exécutable de test sans avoir à les spécifier dans les constantes du Makefile.
  + 1. 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.
  + 1. 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.

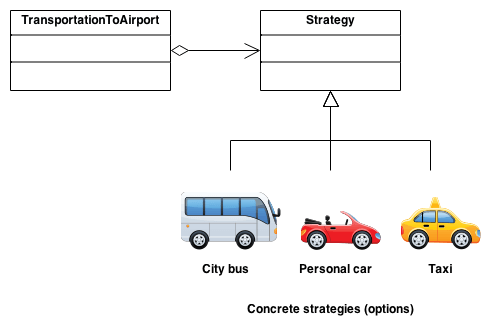


Figure – Illustration du pattern strategy (src : https://sourcemaking.com/design\_patterns/strategy)