**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**  
  
  
  
  
**BÁO CÁO ĐỒ ÁN**  
**ĐỀ TÀI: MẪU THIẾT KẾ ITERATOR TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

Nhóm tác giả

Kiều Công Hậu – 18127259

Mai Đăng Khánh – 18127118

Huỳnh Nhật Nam – 18127014

**Môn học: Phương pháp lập trình hướng đối tượng**

**Thành phố Hồ Chí Minh – 2019**

LỜI CẢM ƠN

MỤC LỤC

[LỜI CẢM ƠN 1](#_Toc14577761)

[MỤC LỤC 2](#_Toc14577762)

[DANH MỤC HÌNH 3](#_Toc14577763)

[DANH MỤC BẢNG 4](#_Toc14577764)

[DANH MỤC TỪ VIẾT TẮT 5](#_Toc14577765)

[DANH MỤC THUẬT NGỮ 6](#_Toc14577766)

[CHƯƠNG 1. VẤN ĐỀ THỰC TẾ 7](#_Toc14577767)

[1. Dạo đầu 7](#_Toc14577768)

[2. Tình huống thực tế 7](#_Toc14577769)

[3. Vấn đề gặp phải 11](#_Toc14577770)

[4. Giải quyết 12](#_Toc14577771)

[CHƯƠNG 2. MẪU THIẾT KẾ 13](#_Toc14577772)

[CHƯƠNG 3. ITERATOR 14](#_Toc14577773)

[CHƯƠNG 4. KẾT LUẬN 15](#_Toc14577774)

[PHỤ LỤC 16](#_Toc14577775)

[TÀI LIỆU THAM KHẢO 17](#_Toc14577776)

DANH MỤC HÌNH

DANH MỤC BẢNG

DANH MỤC TỪ VIẾT TẮT

DANH MỤC THUẬT NGỮ

CHƯƠNG 1. VẤN ĐỀ THỰC TẾ

1. Dạo đầu

Ở chương này, chúng ta sẽ tạm gác qua “mẫu thiết kế” (design pattern) là gì mà chỉ đi tìm ra lý do tại sao khái niệm “mẫu thiết kế” lại được ra đời.

1. Tình huống thực tế

Kevin, Harry và Ned đang là những sinh viên năm nhất của trường đại học Harvard. Họ đang cùng nhau viết một phần mềm quản lý trường học. Cả ba bắt đầu bàn bạc và chia ra các class cần thiết của dự án này cho từng người đảm nhiệm.

|  |
| --- |
|  |
| *Class diagram* |

Kevin được phân công viết class ***cStudent*** để quản lý thông tin của 1 sinh viên và class ***cStudentCollection*** để quản lý tất các các sinh viên trong trường. Kevin quyết định sử dụng cấu trúc **Linked List** để lưu trữ các ***cStudent*** trong ***cStudentCollection***. Sau đây là đoạn code mà Kevin đã cài đặt:

1. **/\* Định nghĩa class cStudent \*/**
2. **class** cStudent
3. {
4. **public**:
5. cStudent(string name = "N/A", string id = "N/A");
6. **void** input\_info(); //Nhập thông tin của sinh viên từ bàn phím
7. **void** show\_info();   //In thông tin của sinh viên ra màn hình
8. cStudent\*& next(); //Trả về con trỏ next của sinh viên này (linked list)
10. **private**:
11. string Name;
12. string ID;
13. cStudent\* Next;
14. };
15. **/\* Khai báo class cStudent \*/**
16. cStudent::cStudent(string name, string id)
17. {
18. **this**->Name = name;
19. **this**->ID = id;
20. **this**->Next = nullptr;
21. }
23. **void** cStudent::input\_info()
24. {
25. cout << "Input info for this Student" << endl;
26. cout << "  - Name: ";
27. getline(cin, **this**->Name);
28. cout << "  - ID: ";
29. getline(cin, **this**->ID);
30. }
32. **void** cStudent::show\_info()
33. {
34. cout << "  - Type: Student" << endl;
35. cout << "  - Name: " << **this**->Name << endl;
36. cout << "  - ID: " << **this**->ID << endl;
37. }
38. cStudent\*& cStudent::next()
39. {
40. **return** **this**->Next;
41. }
42. **/\* Định nghĩa class cStudentCollection \*/**
43. **class** cStudentCollection
44. {
45. **friend** **class** cMinistry;   //Tạm thời bỏ qua dòng này
46. **public**:
47. cStudentCollection();
48. ~cStudentCollection();
50. **private**:
51. cStudent\* Head; //Con trỏ trỏ tới phần tử đầu tiên của linked list
52. };
53. **/\* Khai báo class cLecturerCollection \*/**
54. cStudentCollection::cStudentCollection()
55. {
56. **this**->Head = nullptr;
57. }
59. cStudentCollection::~cStudentCollection()
60. {
61. cStudent\* temp;
62. **while** (**this**->Head)
63. {
64. temp = **this**->Head;
65. **this**->Head = temp->next();
66. **delete** temp;
67. }
68. }

Harry thì viết class ***cLecturer*** để quản lý thông tin của 1 giảng viên và class ***cLecturerCollection*** để quản lý tất cả các giảng viên trong trường. Harry quyết định sử dụng cấu trúc **Stack** để lưu trữ các ***cLecturer*** trong ***cLecturerCollection***. Sau đây là phần cài đặt của Harry:

1. **/\* Định nghĩa class cLecturer \*/**
2. **class** cLecturer
3. {
4. **public**:
5. cLecturer(string name = "N/A", string id = "N/A");
6. **void** input\_info();
7. **void** show\_info();
9. **private**:
10. string Name;
11. string ID;
12. };
13. **/\* Khai báo class cLecturer \*/**
14. cLecturer::cLecturer(string name, string id)
15. {
16. **this**->Name = name;
17. **this**->ID = id;
18. }
20. **void** cLecturer::input\_info()
21. {
22. cout << "Input info for this Lecturer" << endl;
23. cout << "  - Name: ";
24. getline(cin, **this**->Name);
25. cout << "  - ID: ";
26. getline(cin, **this**->ID);
27. }
29. **void** cLecturer::show\_info()
30. {
31. cout << "  - Type: Lecturer" << endl;
32. cout << "  - Name: " << **this**->Name << endl;
33. cout << "  - ID: " << **this**->ID << endl;
34. }
35. **/\* Định nghĩa class cLecturerCollection \*/**
36. **class** cLecturerCollection
37. {
38. **friend** **class** cMinistry;   //Tạm thời bỏ qua dòng này
39. **public**:
40. cLecturerCollection();
41. ~cLecturerCollection();
43. **private**:
44. **static** **const** **int** MAX = 100;   //Số lượng giảng viên tối đa
45. cLecturer\* Lecturers[MAX];
46. **int** Count;   //Số lượng giảng viện hiện tại
47. };
48. **/\* Khai báo class cLecturerCollection \*/**
49. cLecturerCollection::cLecturerCollection()
50. {
51. **this**->Count = 0;
52. **for** (**int** i = 0; i < **this**->MAX; ++i)
53. **this**->Lecturers[i] = nullptr;
54. }
56. cLecturerCollection::~cLecturerCollection()
57. {
58. **for** (**int** i = 0; i < **this**->Count; ++i)
59. **delete** **this**->Lecturers[i];
60. }

Còn Ned viết class ***cMinistry*** để quản lý tất cả các phương thức của ban giáo vụ đối với tập hợp những sinh viên và giảng viên trong trường. Sau đây là phần cài đặt của Ned:

1. **/\* Định nghĩa class cMinistry \*/**
2. **class** cMinistry
3. {
4. **public**:
5. cMinistry(cLecturerCollection\* lecturerCollection, cStudentCollection\* studentCollection);
6. **void** show\_list\_lecturers();   // In ra danh sách các giảng viên trong trường
7. **void** show\_list\_students();   // In ra danh sách các sinh viên trong trường
8. **void** add\_one\_lecturer(**const** cLecturer& lecturer);  // Thêm 1 giáo viên
9. **void** add\_one\_student(**const** cStudent& student);  // Thêm 1 sinh viên
11. **private**:
12. cLecturerCollection\* LecturerCollection;
13. cStudentCollection\* StudentCollection;
14. };
15. **/\* Khai báo class cMinistry \*/**
16. cMinistry::cMinistry(cLecturerCollection\* lecturerCollection, cStudentCollection\* studentCollection)
17. {
18. **this**->LecturerCollection = lecturerCollection;
19. **this**->StudentCollection = studentCollection;
20. }
22. **void** cMinistry::show\_list\_lecturers()
23. {
24. cout << "~ List of lecturers ~" << endl;
25. **for** (**int** i = 0; i < **this**->LecturerCollection->Count; ++i)
26. {
27. cout << " #" << i + 1 << endl;
28. **this**->LecturerCollection->Lecturers[i]->show\_info();
29. }
30. }
32. **void** cMinistry::show\_list\_students()
33. {
34. cout << "~ List of students ~" << endl;
35. **int** i = 0;
36. **for** (cStudent\* st = **this**->StudentCollection->Head; st != nullptr; st = st->next())
37. {
38. cout << " #" << i + 1 << endl;
39. st->show\_info();
40. i++;
41. }
42. }
44. **void** cMinistry::add\_one\_lecturer(**const** cLecturer& lecturer)
45. {
46. **if** (**this**->LecturerCollection->Count == **this**->LecturerCollection->MAX)
47. **throw**;  // Full
49. \***this**->LecturerCollection->Lecturers[**this**->LecturerCollection->Count++] = lecturer;
50. }
52. **void** cMinistry::add\_one\_student(**const** cStudent& student)
53. {
54. cStudent\* newStudent = **new** cStudent(student);
55. newStudent->next() = **this**->StudentCollection->Head;
56. **this**->StudentCollection->Head = newStudent;
57. }
58. Vấn đề gặp phải

Rắc rối bắt đầu nảy sinh khi Ned bắt tay vào cài đặt class ***cMinistry*** sau khi Kevin và Harry đã hoàn thành phần cài đặt cho các class của mình.

Về căn bản, class cMinistry mà Ned đảm nhiệm sẽ chứa 2 đối tượng tập hợp (aggregate object) là tập hợp các sinh viên (***cStudentCollection***) và tập hợp các giảng viên (***cLecturerCollection***) trong trường. Và nhiệm vụ của class cMinistry là phải cung cấp các phương thức liên quan đến 2 tập hợp trên (Ví dụ: Liệt kê danh sách các sinh viên, giảng viên có trong trường; Thêm / Bớt sinh viên và giảng viên; Sắp xếp danh sách theo thứ tự tăng dần theo tên; Liệt kê các sinh viên được nhận học bổng;…).

Nhưng trở ngại lớn nhất là mỗi tập hợp lại có 1 cấu trúc khác nhau (**Linked List** của ***cStudentCollection*** và **Stack** của ***cLecturerCollection***), dẫn tới cách duyệt các phần tử trong các tập hợp đó là khác nhau. Lúc này, Ned phải đi hỏi Kevin và Harry về cấu trúc bên trong của 2 class tập hợp mà 2 người này đã viết kèm với cách để duyệt từng phần tử đối với mỗi tập hợp ấy (cLecturerCollection thì xài vòng lặp for index, cStudentCollection thì xài con trỏ next). Có vẻ rắc rối nhỉ!!!

Tới đây, Ned đã tốn một ít thời gian để hiểu được cấu trúc của 2 class tập hợp trên và cách để duyệt từng phần tử trong 2 tập hợp ấy và hoàn thành được 2 phương thức là **void** show\_list\_lecturers(); và **void** show\_list\_students();. May mắn là vì class ***cMinistry*** chỉ quản lý có 2 tập hợp là các sinh viên và các giảng viên trong trường, bên cạnh đó 2 class này có cấu trúc cũng khá đơn giản, nên vẫn khá nhẹ nhàng cho Ned. Nhưng sẽ là vấn đề nan giải nếu số tập hợp cần quản lý có tận 5 – 10 loại, mỗi loại có một loại cấu trúc đơn giản và phức tạp khác nhau (Stack, Dynamic Array, Singly Linked List, Doubly Linked List, Skip List, Tree, Hash Table,…). Điều này dẫn tới Ned sẽ phải hiểu cấu trúc và ghi nhớ từng tập hợp duyệt theo từng cách tương ứng nào.

Thậm chí sẽ còn tệ hơn nếu Kevin hoặc Harry thay đổi cấu trúc của 2 class tập hợp của mình sau khi Ned hoàn thành phần cài đặt class ***cMinistry***. Điều này dẫn tới cách duyệt từng phần tử trong tập hợp cũ của tập hợp bị thay đổi không còn hợp lệ nữa và Ned sẽ phải 2 người bạn của mình về cấu trúc mới và cách để duyệt từng phần tử trong loại tập hợp đó, sau đó Ned sẽ phải ngồi sửa lại một đống code đã viết.

“Không đời nào! =.=” – Ned nói.

1. Giải quyết

Để giải quyết được những vấn đề mà Kevin, Harry và Ned đang gặp phải. Ta phải sử dụng **mẫu thiết kế Iterator** (Iterator design pattern) – một trong những mẫu thiết kế rất phổ biến trong lập trình hướng đối tượng. Mẫu thiết kế này như thế nào, cách sử dụng và cài đặt ra sao, lợi ích và tác hại của nó sẽ được trình bày cụ thể ở các chương sau.

CHƯƠNG 2. MẪU THIẾT KẾ

CHƯƠNG 3. ITERATOR

CHƯƠNG 4. KẾT LUẬN

PHỤ LỤC

TÀI LIỆU THAM KHẢO

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”.

[2] Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra, “Head First Design Patterns”.

[3] Dmitri Nesteruk, “Design Patterns in Modern C++: Reusable Approaches for Object-Oriented Software Design”.

[4] <https://refactoring.guru/design-patterns/iterator>