

CSC10003 – Phương pháp lập trình hướng đối tượng

Tuần 08: Design Pattern

I. Singleton Design Pattern

- ✚ Singleton là mẫu thiết kế trong lập trình hướng đối tượng thuộc vào nhóm khởi tạo (Creational).
- ✚ Mục tiêu của mẫu thiết kế này:
 - **Mỗi lớp đảm bảo chỉ có một thể hiện duy nhất** trong suốt chương trình chạy.
 - Cung cấp cách thức **truy xuất toàn cục**
- ✚ Vì sao cần đảm bảo chỉ có một thể hiện duy nhất của lớp này?
 - Để cần chỉ có một đối tượng chính xác điều phối trên toàn hệ thống
 - Ví dụ: trong bài toán về quản lý nhân sự của trường đại học, đảm bảo chỉ có duy nhất một đối tượng quản lý danh sách nhân sự chung cho toàn trường.
 - Ví dụ: chỉ có thể mở được một Task Manager trên Windows để tránh việc tranh giành tài nguyên
- ✚ Cách tiếp cận cho việc cài đặt sử dụng hướng đối tượng:
 - Để đảm bảo chỉ có duy nhất 1 thể hiện của lớp trong suốt chương trình chạy, cần phải chặn việc tạo mới đối tượng (không cho phép new trực tiếp)
 - Ý tưởng: để chặn việc tạo mới đối tượng (new), **cần đặt phương thức tạo lập vào private**
 - Cung cấp phương thức public để lấy thể hiện của lớp này. Nếu chưa có thì tạo mới 1 lần, nếu có rồi thì trả về thể hiện duy nhất này.
 - Thuộc tính của lớp chính là biến con trỏ trỏ đến thể hiện duy nhất của lớp.
 - Để cung cấp cách thức truy xuất toàn cục để người dùng chỉ được phép sử dụng đối tượng sẵn có này, cần phải khai báo thuộc tính và phương thức mang tính chất toàn cục.
 - Nếu xài biến toàn cục trong chương trình, vậy sẽ mất đi tính chất đóng gói (encapsulation) của hướng đối tượng
 - Ý tưởng: **đặt thuộc tính và phương thức của lớp này là thành phần tĩnh (static)**.
 - Thành phần tĩnh (static) là thành phần chung cho mọi lớp đối tượng tương ứng, do chỉ có duy nhất 1 lớp đối tượng, việc sử dụng thành phần tĩnh rất phù hợp và vẫn giữ được tính chất của hướng đối tượng

```

//Lớp Singleton chỉ cho phép tạo 1 đối tượng duy nhất
class Singleton
{
private:
    //Đối tượng duy nhất của lớp này, là thuộc tính kiểu con trỏ, tĩnh (static)
    static Singleton* instance;
    //Các thuộc tính khác ...
    //Đặc điểm: phương thức tạo lập phải để trong private
    Singleton()
    {

    }

public:
    //Public phương thức lấy đối tượng duy nhất này
    //Nếu chưa có thì new 1 lần, nếu có rồi thì trả lại đối tượng
    static Singleton* GetInstance()
    {
        if (instance == NULL)
        {
            instance = new Singleton();
        }
        return instance;
    }

    //Public phương thức để xóa đối tượng duy nhất tránh memory leak
    //Nếu chưa có thì không làm gì hết, có rồi thì delete và trả về null
    static void DeleteInstance()
    {
        if (instance)
        {
            delete instance;
            instance = NULL;
        }
    }
    //Nếu delete trong destructor dẫn đến infinite loop do tự xóa chính nó (static)
    ~Singleton()
    {
        //Xóa các thuộc tính khác ngoài thuộc tính static
    }

    //Các phương thức khác
};

```



Thành phần tĩnh (Static) :

- Thành phần xài chung của tất cả đối tượng có chung lớp

- Không thuộc vào một đối tượng cụ thể
- Nếu sử dụng trong lớp, yêu cầu cần phải khởi tạo trước 1 lần khi sử dụng
- ✚ Khi khai báo thuộc tính kiểu con trỏ của thành phần tĩnh, không được phép xóa trong phương thức hủy, vì tính chất xài chung cho tất cả đối tượng, khi gọi destructor sẽ gặp tình trạng tự bản thân thành phần này gọi xóa chính nó, dẫn đến việc lặp vô hạn !!!
 - Cách xử lý để tránh gặp rò rỉ bộ nhớ : tạo một phương thức tĩnh riêng để xử lý việc xóa tương ứng
- ✚ Sau khi cài lớp Singleton, ví dụ sau đây cho thấy việc sử dụng lớp này :

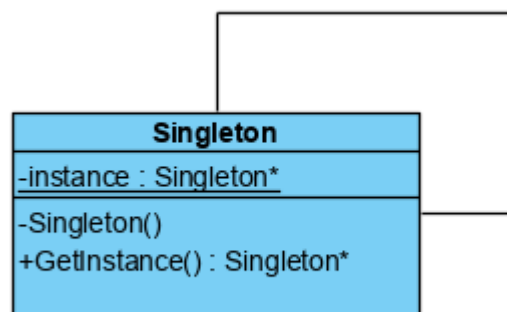
```
//Gán lần đầu khi mới vào chương trình là NULL (do biến static)
Singleton* Singleton::instance = NULL;

void main()
{
    //Gọi tạo đối tượng 1 lần duy nhất
    Singleton* object = Singleton::GetInstance();

    //Các thao tác trên đối tượng duy nhất này...

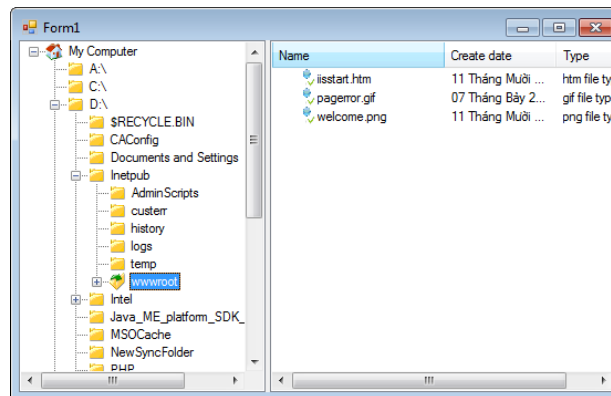
    //Thực hiện gọi xóa đối tượng duy nhất này
    //Xóa thành phần tĩnh KHÔNG cài qua destructor
    Singleton::DeleteInstance();
}
```

- ✚ Mô hình UML cho mẫu thiết kế này :
 - Thành phần static ký hiệu gạch chân



II. Composite Design Pattern

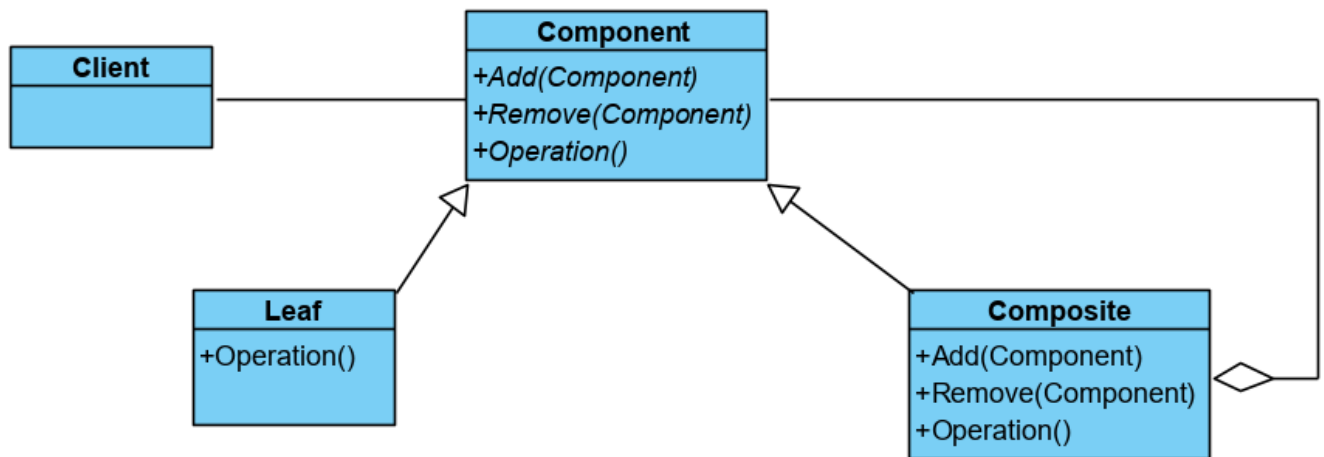
- ✚ Composite là mẫu thiết kế trong lập trình hướng đối tượng thuộc vào nhóm cấu trúc (Structural).
- ✚ Mẫu thiết kế Composite dùng để tạo ra các đối tượng mang tính chất phức hợp (Composite). Một đối tượng phức hợp Composite được tạo thành từ một hay nhiều đối tượng có chức năng tương tự nhau
 - Thao tác trên một nhóm đối tượng theo cách thao tác trên một đối tượng
- ✚ Nhận biết khi nào cần xài mẫu Composite:
 - Khi đối tượng cho phép chứa những loại đối tượng khác, bao gồm cả chính nó
 - Các thao tác của những đối tượng này có chức năng tương tự nhau
 - Mang tính chất đệ quy
- ✚ Mẫu Composite rất thích hợp cho việc xây dựng cấu trúc dạng phân lớp hay cấu trúc cây:
 - Ví dụ: trong hệ thống Windows Explorer, trong ổ đĩa sẽ chứa tập tin và folder, trong folder sẽ chứa tập tin và folder khác
 - Tập tin và folder đều có tên, dung lượng, có cách thức tính dung lượng tương tự nhau. (Nguồn ảnh từ internet)



- Ví dụ: trong bảng điện sẽ có mạch điện song song và mạch điện nối tiếp, trong mạch điện song song có thể chứa mạch điện nối tiếp và mạch điện song song khác
- Ví dụ: cấu trúc lưu trữ trang web dưới dạng file .html lưu bằng cách thẻ tag. Trong thẻ tag chứa thông tin của thẻ tag và có thể chứa các thẻ tag khác
- ✚ Ý tưởng thực hiện:
 - Cần phải xác định rõ các thành phần trong mẫu thiết kế Composite: gồm có 4 thành phần như sau:
 - Composite: thành phần phức hợp: chứa các loại thành phần đơn và bao gồm cả thành phần phức hợp (có tính đệ quy)
 - Ví dụ: bài cây thư mục file, folder, trong folder có thể chứa file và chứa folder khác, vậy folder chính là thành phần phức hợp
 - Ví dụ: bài mạch điện, trong đó mạch điện song song có thể chứa mạch điện nối tiếp và mạch điện song song khác

- Leaf: thành phần đơn: chỉ đối tượng đơn,
 - Ví dụ: tập tin (file) là thành phần đơn
- Component : thành phần mang tính tổng quát, là đối tượng cha (thường là lớp trừu tượng) mang tính chất chung của các thành phần đơn và phức hợp
 - Ví dụ : file và folder đều có tên, dung lượng, vậy ta có thể có 1 component là DriveComponent chỉ chung hết tất cả các loại thành phần có thể có
- Client : là nơi thực thi hoặc sử dụng các thành phần trên (hàm main,...)
 - Cài đặt trong lớp composite gồm 1 mảng chứa tất cả các thành phần chung (component), xử lý trên lớp composite bằng cách duyệt mảng

📁 Sơ đồ UML của mẫu thiết kế Composite :



```

class Component
{
protected:
    string name;
    float size;
public:
    virtual string GetName() = 0;
    virtual float GetSize() = 0;
    virtual void AddComponent(Component* ifile)
    {
        //Không xử lý gì tại chỗ này, để cho lớp Composite xử lý
        //Khai báo đa hình để gọi đệ quy trong Composite
    }
    virtual void RemoveComponent(Component* ifile)
    {
        //Không xử lý gì tại chỗ này, để cho lớp Composite xử lý
        //Khai báo đa hình để gọi đệ quy trong Composite
    }
    virtual ~Component()
    {
        //Nhớ đa hình phương thức hủy (hủy ảo)
    }
};

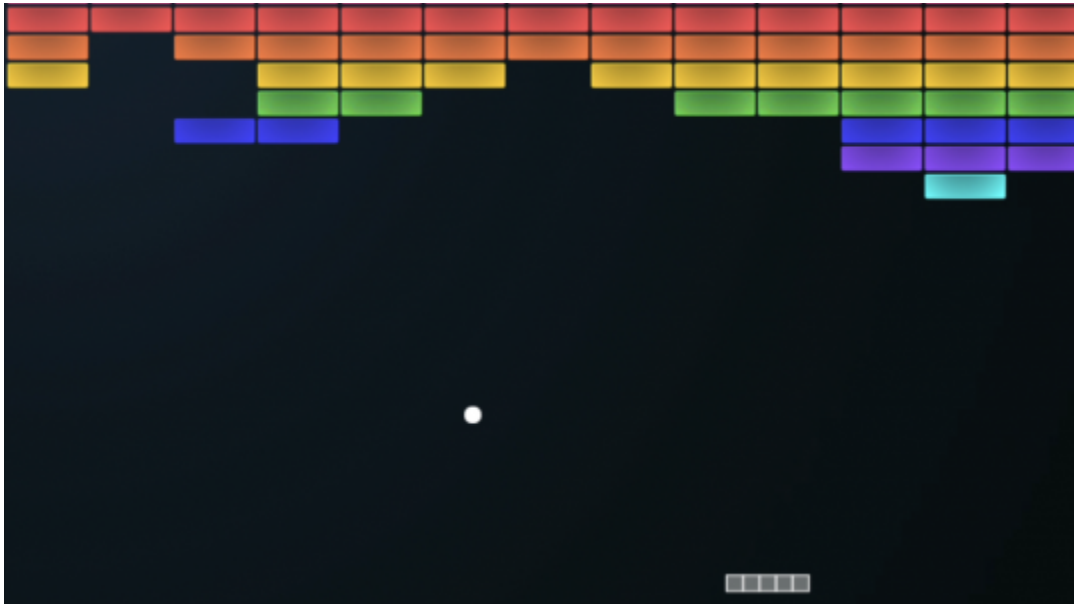
class Composite : public Component
{
    //Mảng chứa tất cả các component (bao gồm cả leaf và composite)
    vector<Component*> components;
public:
    string GetName()
    {
        return this->name;
    }
    //Cài đặt lại hàm GetSize, đệ quy trong Composite
    float GetSize()
    {
        float totalSize = 0;
        for (int i = 0; i < components.size(); i++)
            totalSize += components[i]->GetSize(); //đệ quy ngay tại vị trí này
    }
    void AddComponent(Component* iComp)
    {
        //Xử lý thêm 1 component vào vector<Component*>
    }
    void RemoveComponent(Component* iComp)
    {
        //Xử lý xóa 1 component ra vector<Component*>
    }
    ~Composite() //Destructor xóa hết tất cả file và folder có trong folder
    {
        //Xử lý xóa hết toàn bộ Composite (đệ quy)
    }
};

```

```
class Leaf : public Component
{
public:
    string GetName()
    {
        return this->name;
    }
    float GetSize()
    {
        return this->size;
    }
    ~Leaf() {}
};
```

III. Bài tập thực hành

Bài tập 1:



Giả sử có 2 lập trình viên A và B cùng nhau viết mã nguồn cho game Brick Breakout.
A cài đặt class Game.
B cài đặt lớp Ball, Bar, Brick.

Theo logic, mỗi game chỉ có 1 quả bóng được sinh ra. Vì thế, B muốn ngăn cản A tạo quá nhiều đối tượng Ball trong game.
Bạn hãy giúp B làm công việc này.

```

// Implemented by devB
class Bar
{
}

// Implemented by devB
class Ball
{
}

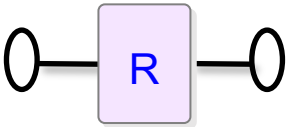
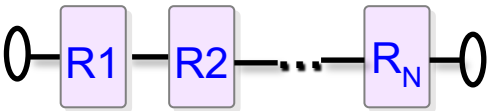
// Implemented by devB
class Brick
{
}

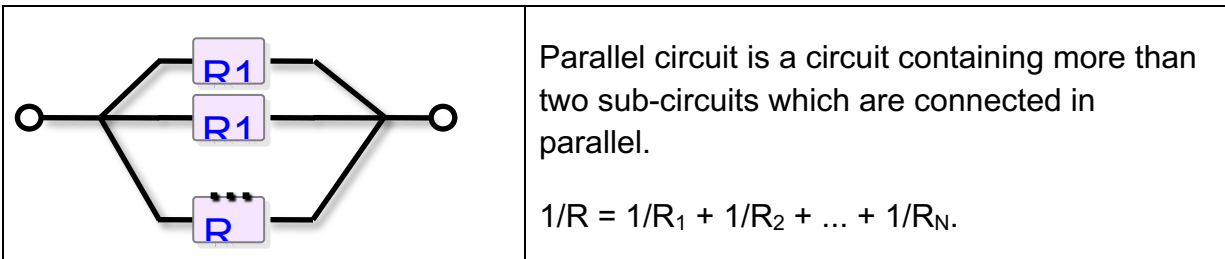
// Implemented by devB
class Game
{
    vector<Brick*> bricks;
    Bar* bar;
    Ball* ball;
}

```

Bài tập 2: Đề thi 19CLC - 2020

There are three types of basic electrical circuits:

	<p>Single circuit is a circuit containing only one resistor.</p> <p>$R \text{ (Resistance)} = U \text{ (Voltage)} / I \text{ (Current)}$.</p>
	<p>Series circuit is a circuit containing more than two sub-circuits which are connected in series.</p> <p>$R = R_1 + R_2 + \dots + R_N$.</p>



The sub-circuit in series or parallel circuit can be either a single circuit, another series circuit, or another parallel circuit.

You are asked to do the followings by applying encapsulation, inheritance, and polymorphism:

- a) Draw a class diagram for a program to calculate circuit resistance. The design should include necessary variables and functions to:
 - Construct a circuit of one type.
 - Add a sub-circuit to a Series or Parallel circuit.
 - Calculate resistance of a circuit.
- b) Write C++ code to implement the design.

Yêu cầu khác:

- Các Class cần được tách ra 2 phần:
 - o Phần định nghĩa thuộc tính và phương thức ở file .h
 - o Phần cài đặt nằm ở file .cpp
- Hàm main nằm ở file Main.cpp, chứa các luồng demo các yêu cầu của từng bài tập như mẫu bên dưới

```
//File Main.cpp nằm Source File. Chứa hàm main

void main()
{
    //Bài 1
    ...

    //Bài 2
    ...

    //Bài 3
    ...
}
```

Các trường hợp không tuân thủ sẽ bị trừ từ 5%-20% số điểm tùy theo mức độ.

IV. Nộp bài

Trên lớp: Hoàn thành ở lớp: ít nhất 1 bài tập.

Về nhà: Hoàn thành tất cả bài tập.

Tổ chức thư mục theo cấu trúc sau:

- ☐ **Source code:** Thư mục chứa mã nguồn. Cần xóa các tập tin trung gian của quá trình biên dịch cho nhẹ bớt (Build > Clean solution và xóa đi thư mục ẩn .vs)
- ☐ **Release:** Thư mục chứa tập tin thực thi .exe biên dịch ra từ mã nguồn.
- ☐ **Other:** Thư mục chứa các tài liệu nộp thêm ví dụ ảnh sơ đồ lớp.

Nén tất cả ở dạng <MSSV>_Week<X>_<Y>.zip/rar.

- ☐ **MSSV:** mã số sinh viên
- ☐ **X:** mã tuần
- ☐ **Y:** số bài tập đã làm
- ☐ Ví dụ: **21120001_Week08_02.zip**