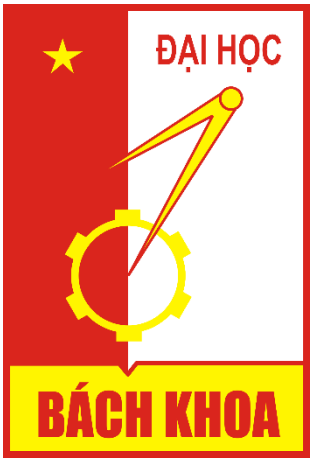
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO PROJECT II

ĐỀ TÀI: DESIGN PATTERN

Giảng viên hướng dẫn: TS. Nguyễn Bá Ngọc

Sinh viên: Nguyễn Duy Hoài Lâm

MSSV: 20173225

Giới Thiệu

Design Pattern là một kỹ thuật trong lập trình hướng đối tượng, nó khá quan trọng và mọi lập trình viên muốn giỏi đều phải biết. Được sử dụng thường xuyên trong các ngôn ngữ OOP. Nó sẽ cung cấp cho bạn các "mẫu thiết kế", giải pháp để giải quyết các vấn đề chung, thường gặp trong lập trình. Các vấn đề mà bạn gặp phải có thể bạn sẽ tự nghĩ ra cách giải quyết nhưng có thể nó chưa phải là tối ưu. Design Pattern giúp bạn giải quyết vấn đề một cách tối ưu nhất, cung cấp cho bạn các giải pháp trong lập trình OOP.

Design Patterns không phải là ngôn ngữ cụ thể nào cả. Nó có thể thực hiện được ở phần lớn các ngôn ngữ lập trình, chẳng hạn như Java, C#, C++ thậm chí là Javascript hay bất kỳ ngôn ngữ lập trình nào khác.

Việc dử dụng Design Patterns giúp thiết kế của chúng ta linh hoạt, dễ dàng thay đổi và bảo trì hơn.

Vì thế trong đề tài này dưới sự hướng dẫn của Ngọc em nghiên cứu và tìm hiểu về các loại Design Patterns cơ bản trong C++.

Về cơ bản Design Patterns chia làm 3 nhóm với các mẫu tương ứng mà em sẽ nghiên cứu như sau.

- Nhóm 1: Creational Patterns (Nhóm khởi tạo: giúp chúng ta khởi tạo đối tượng)
  + Singleton
  + Builder
  + Prototype
- Nhóm 2: Structural Patterns (Nhóm cấu trúc: giúp ta thiết lập, định nghĩa quan hệ giữa các đối tượng)
  + Adapter
  + Bridge
  + Composite
- Nhóm 3: Behavioral Patterns (Nhóm hành vi: tập trung thực hiện các hành vi của đối tượng)
  + Command
  + Observer
  + State
  +Visitor

I. Singleton

Singleton là mẫu thuộc nhóm Creational Patterns đảm bảo chỉ duy nhất một thể hiện được tạo ra và nó sẽ cung cấp cho bạn một method để có thể truy xuất được thể hiện duy nhất đó mọi lúc mọi nơi trong chương trình.

Sử dụng Singleton khi chúng ta muốn:

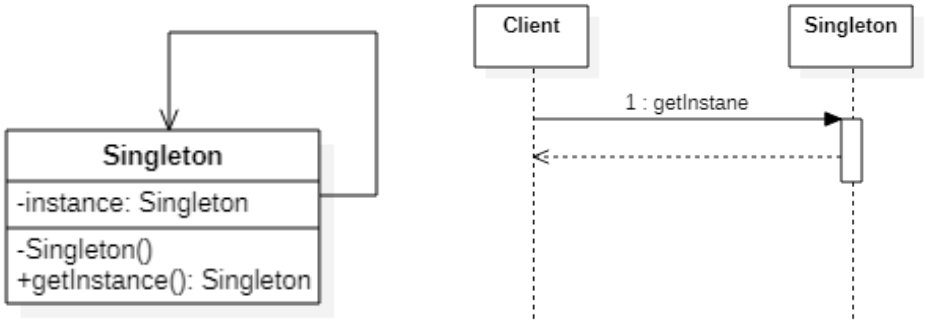
- Đảm bảo rằng chỉ có một thể hiện của lớp.
- Việc quản lý việc truy cập tốt hơn vì chỉ có một thể hiện duy nhất.
- Có thể quản lý số lượng thể hiện của một lớp trong giới hạn chỉ định.

Khi implement singleton pattern phải đảm bảo rằng chỉ có một thể hiện duy nhất được tạo ra và thể hiện tại có thể dùng mọi lúc mọi nơi. Nói cách khác, khi xây dựng lớp cần một kỹ thuật để có thể truy xuất được vào các thành viên public của lớp mà không tạo ra một thể hiện nào (thông qua việc khởi tạo bên ngoài lớp).

Để chắc chắn rằng, không có bất kỳ thể hiện nào bên ngoài lớp được tạo ra, constructor của lớp đó sẽ có phạm vi truy cập là private. Điều đó cũng có nghĩa là thể hiện duy nhất được tạo ra thông qua chính lớp mà chúng ta xây dựng. Vì vậy, ta sử dụng thành phần static để thực hiện điều này. Ta tạo một thể hiện của lớp kiểu private static và một refractor trả về đối tượng thuộc chính lớp đó.

Trong C++, singleton định nghĩa một thuộc tính static là một con trỏ trỏ tới thể hiện duy nhất của lớp đó.

Mô hình:



Mã nguồn:

Khai báo lớp:

```
#include <iostream>
using namespace std;
class Singleton
{
   private:
         static Singleton* m_instance;
         Singleton();
   public:
         static Singleton* getInstance();
         void method();
};
```

Định nghĩa lớp:

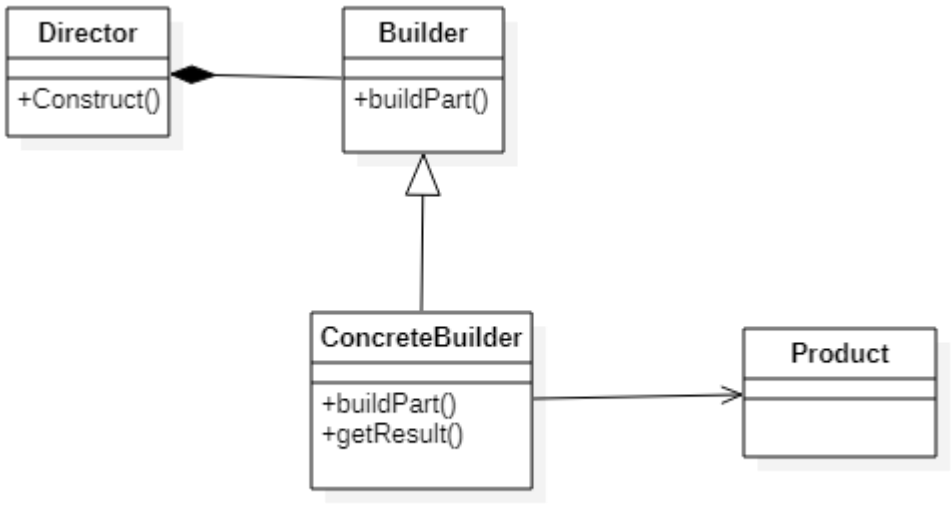
```
Singleton* Singleton::m_instance = NULL;
Singleton* Singleton::getInstance()
{
   if (m_instance == NULL)
   {
         m_instance = new Singleton();
   }
   return m_instance;
}
void Singleton::method()
   {
   cout << "This is singleton parttern";
   }
```

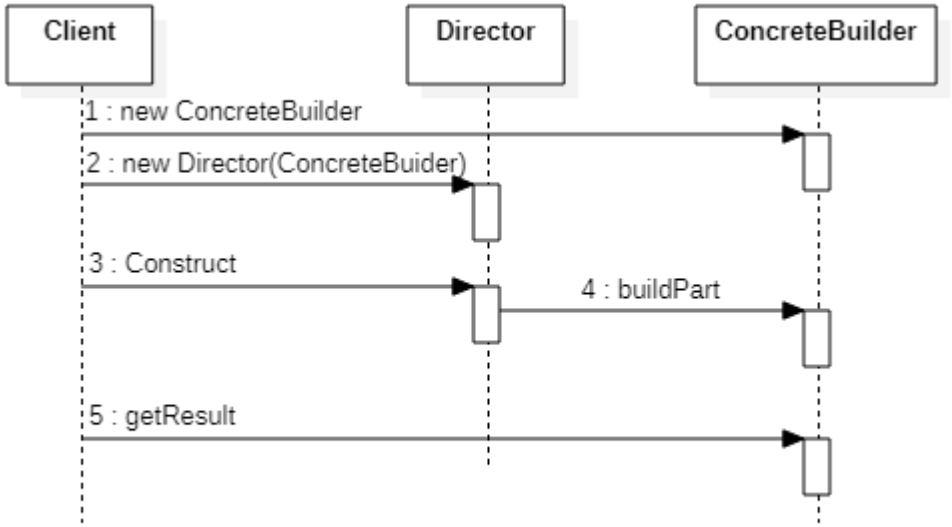
Như vậy, thông qua các truy cập Singleton::getInstance() ta luôn có một đối tượng duy nhất, đồng thời có thể sử dụng nó để truy cập mọi thành phần public trong lớp. Ví dụ để gọi phương thức method trong lớp ta dùng Singleton::getInstance()->method().

II. Builder

Builder pattern là loại design pattern thuộc loại creational pattern, mô hình này cung cấp một trong những cách tốt nhất để tạo ra một đối tượng.Builder pattern là mẫu thiết kế đối tượng được tạo ra để xây dựng một đôi tượng phức tạp bằng cách sử dụng các đối tượng đơn giản và sử dụng tiếp cận từng bước, việc xây dựng các đối tượng đôc lập với các đối tương khác.

Mô hình:





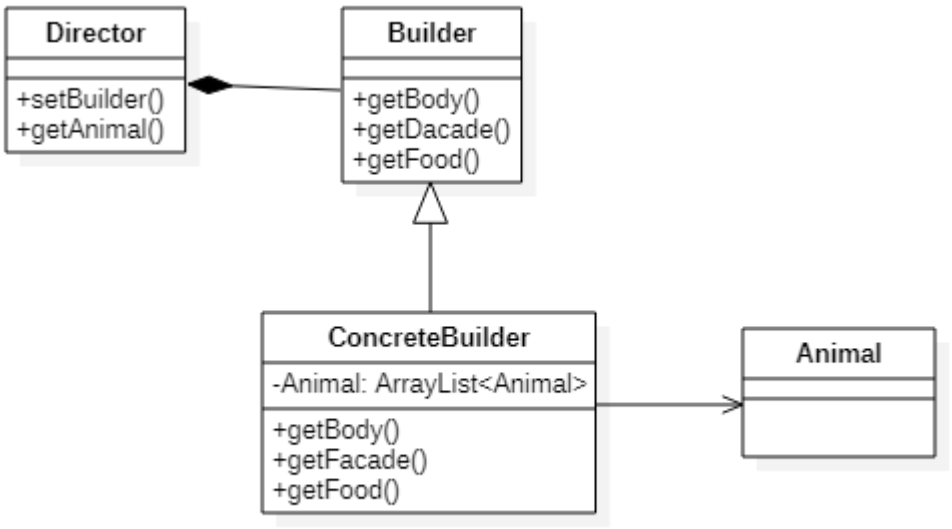
Builder: là thành phần định nghĩa lớp trừu tượng (abstract class) để tạo ra một hoặc nhiều thành phần của đối tượng Product.

ConcreteBuilder: là thành phần triển khai, cụ thể hoá các lớp trừu tượng để tạo ra các thành phần và tập hợp các thành phần đó với nhau, xác định và nắm giữ các thành phần mà nó tạo ra.

Product: thành phần này đại diện cho đối tượng phức tạp được tạo ra.

Director: thông báo với Builder khi nào một thành phần cần được tạo ra.

Ví dụ:



Mã nguồn:

Animal:

```
class Animal
{
   public:
   Body* body[4];
   Facade* facade;
   Food* food;

   void specifications()
   {
      cout << "Animal with" << "body ," << body[0]->weight << "facade" <<
      facade->color << "food " << food->plant << endl;
   }
};
```

Builder:

```
class Builder
{
      public:
      virtual Body* getBody() = 0;
      virtual Facade* getFacade() = 0;
      virtual Food* getFood() = 0;
};
```

Director:

```
class Director
{
   Builder* builder;

   public:
   void setBuilder(Builder* newBuilder)
   {
         builder = newBuilder;
   }

   Animal* getAnimal()
   {
         Animal* animal = new Animal();

         animal->facade = builder->getFacade();

         animal->food = builder->getFood();

         animal->body[0] = builder->getBody();
         animal->body[1] = builder->getBody();
         animal->body[2] = builder->getBody();
         animal->body[3] = builder->getBody();

         return animal;
   }
};
```