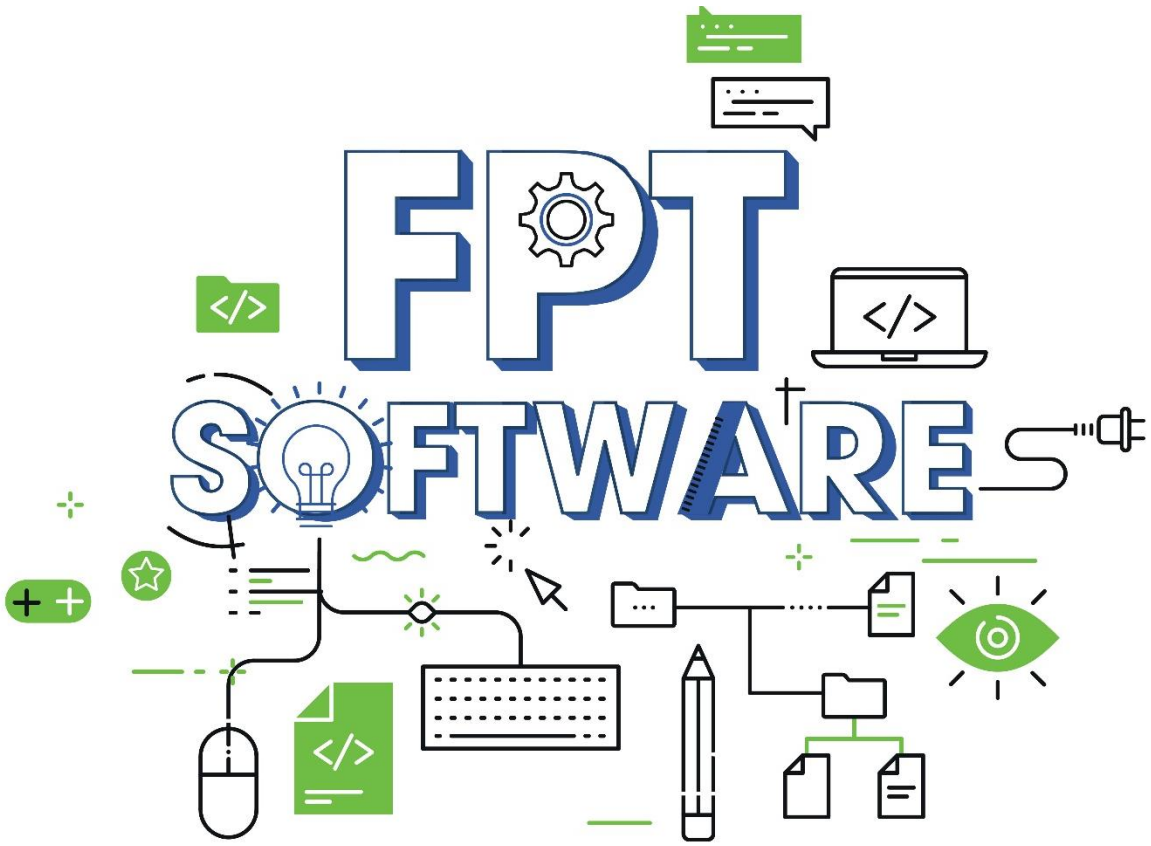
C++ Training Course

Design Pattern

factory



Lesson Objectives

- Understand Design Pattern Principle
- Understand about Singleton Pattern
- Understand about Observer Pattern

Agenda

1. • **Design Pattern Principles**
2. • **Singleton Pattern**
3. • **Observer Pattern**

Section 1

Design Pattern Principles

Design Pattern

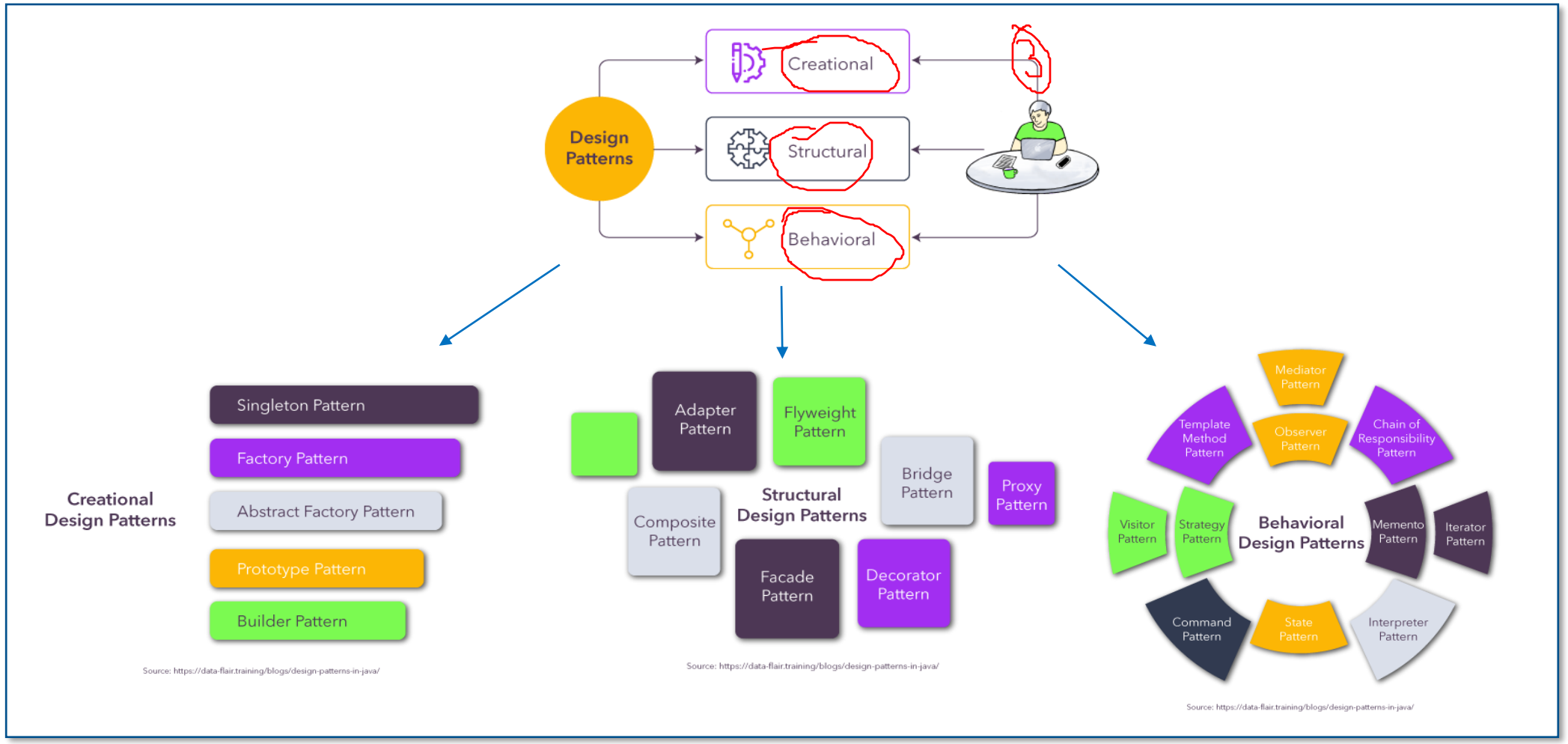
Mẫu thiết kế tái sử dụng giải pháp phổ biến gặp phải

Design Patterns reusable solutions to common problems encountered in software design. Some key key principles underlying design patterns:

nguyên tắc

- **Abstraction:** Design patterns abstract common solutions from specific implementation details.
  trừu tượng Mẫu thiết kế trừu tượng hóa các giải pháp phổ biến từ chi tiết thực hiện cụ thể.
- **Encapsulation**: Hide implementation/data detail.
  Đóng gói: Ẩn chi tiết cài đặt/dữ liệu.
- **Flexibility:** provide flexible solutions that can adapt to changing requirements without significant modification.
  Linh hoạt: cung cấp các giải pháp linh hoạt có thể thích nghi với các yêu cầu thay đổi mà không cần chỉnh sửa đáng kể.
- **Reusability:** promote reusable designs that can be applied in various contexts.
  - Khả năng tái sử dụng: khuyến khích các thiết kế có thể tái sử dụng được áp dụng trong nhiều ngữ cảnh khác nhau.
- **Maintainability**: make it easier to understand and modify code, reducing the likelihood of introducing bugs during maintenance. Dễ bảo trì: làm cho việc hiểu và chỉnh sửa mã nguồn dễ dàng hơn, giảm khả năng gây ra lỗi trong quá trình bảo trì.
- **Scalability:** support scalable designs that can accommodate growth and changes in the system's size and complexity. Khả năng mở rộng: hỗ trợ thiết kế có khả năng mở rộng có thể chứa đựng sự phát triển và sự thay đổi về quy mô và độ phức tạp của hệ thống.
- **Decoupling:** help decouple different components of a system, reducing dependencies and improving modularity Giảm phụ thuộc: giúp tách ra các thành phần khác nhau của hệ thống, giảm sự phụ thuộc và nâng cao tính mô-đun

Design Pattern



1. Nhóm Sáng tạo (Creational Patterns)Các mẫu thiết kế trong nhóm này liên quan đến việc tạo ra các đối tượng, giúp làm cho hệ thống linh hoạt hơn bằng cách quyết định cách thức và thời điểm tạo ra các đối tượng.

Factory Method: Cung cấp một giao diện để tạo đối tượng, nhưng để các lớp con quyết định lớp nào sẽ được khởi tạo.

Abstract Factory: Cung cấp một giao diện để tạo ra các họ đối tượng liên quan hoặc phụ thuộc mà không chỉ rõ lớp cụ thể của chúng.

Singleton: Đảm bảo một lớp chỉ có một thể hiện duy nhất và cung cấp một điểm truy cập toàn cục cho nó.

2. Nhóm Cấu trúc (Structural Patterns)Các mẫu thiết kế trong nhóm này liên quan đến cách tổ chức các lớp và đối tượng để hình thành các cấu trúc lớn hơn.

3. Nhóm Hành vi (Behavioral Patterns)Các mẫu thiết kế trong nhóm này liên quan đến cách mà các lớp và đối tượng tương tác và phân chia trách nhiệm.

Observer: Định nghĩa một sự phụ thuộc một-nhiều giữa các đối tượng để khi một đối tượng thay đổi trạng thái, tất cả các phụ thuộc của nó được thông báo và cập nhật tự động.

Section 2

Singleton Pattern

Singleton Pattern Agenda

1. What is Singleton Pattern
2. Singleton Pattern Implementation

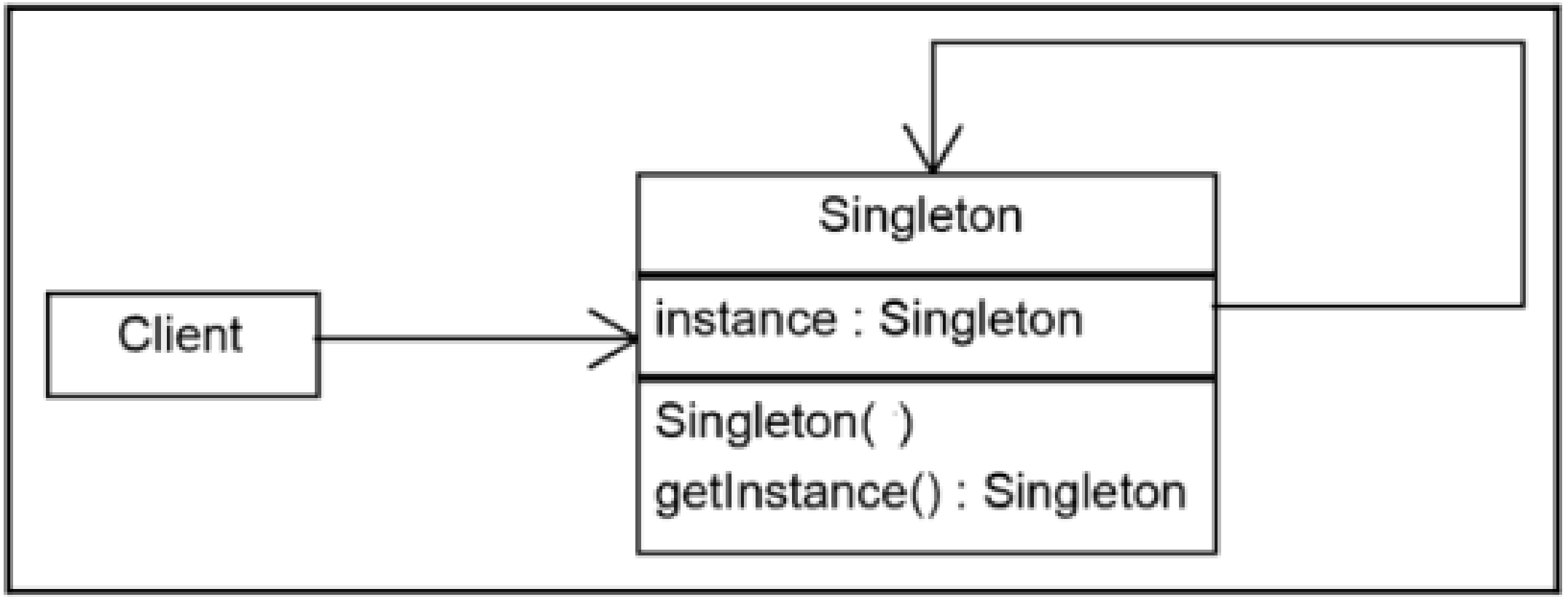
What is Singleton Pattern?

Mẫu thiết kế Singleton đảm bảo rằng một lớp chỉ có duy nhất một đối tượng và cung cấp một điểm truy cập toàn cục cho đối tượng đó (giống như biến toàn cục).

- The Singleton pattern ensures a class has **only one instance** and provides **a global point of access** to it (just like global variable).
- Use cases of Singleton Pattern:
  - ✓ Thread pool, caches
  - ✓ Dialog boxes, objects
  - ✓ Oreferencs and registry setting
  - ✓ Logging

Thread pool: quản lý một nhóm các luồng để tái sử dụng.
Caches: quản lý bộ nhớ đệm để lưu trữ và truy xuất dữ liệu tạm thời.
Dialog boxes: đảm bảo chỉ có một hộp thoại duy nhất mở tại một thời điểm.
Objects: đối tượng cần được chia sẻ trên toàn hệ thống.
Preferences and registry setting: quản lý cài đặt tùy chọn và đăng ký hệ thống.
Logging: ghi nhật ký hoạt động của ứng dụng.

Singleton Pattern Implementation



Singleton Pattern Implementation

Sử dụng các cơ chế đồng bộ hóa như mutex để đảm bảo rằng chỉ có một luồng có thể tạo ra thể hiện duy nhất tại một thời điểm.

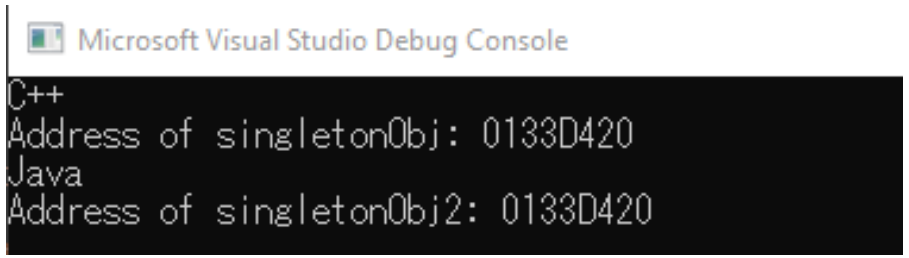
```
#include <iostream>
using namespace std;

class Singleton {
private:
string language;
static Singleton* instancePtr;
// Default constructor
Singleton(){}
public:
Singleton(const Singleton& obj) = delete;

static Singleton* getInstance(){
if (instancePtr == NULL){
instancePtr = new Singleton();
return instancePtr;
}
else{
return instancePtr;
}
}

void setValues(string language){
this->language = language;
}

void print(){
cout << language.c_str() << endl;
}
};
```

tạo đối tượng static ngay khi khởi tọa class

```
class Shape{
public:
        static Shape *GetInstance (int x)
        {
                static Shape obj(x);
                return & obj;
        }
        void print(){
                cout << m_x;
        }
private:
        Shape (){};
        Shave(int x): m_x(x){};
        int m_x;
};

int main(){
        Shape *new = Shape::GetInstance(10);
        new->print();
}
```

mutex singleton Pattern

```
// initializing instancePtr with NULL
Singleton* Singleton::instancePtr = NULL;

int main()
{
Singleton* singletonObj1 = Singleton::getInstance();
singletonObj1->setValues("C++");
singletonObj1->print();
cout << "Address of singletonObj: " << singletonObj1 << endl;

Singleton* singletonObj2 = Singleton::getInstance();
singletonObj2->setValues("Java");
singletonObj2->print();
cout << "Address of singletonObj2: " << singletonObj2 << endl;
return 0;
}
```



Section 3

Observer Pattern

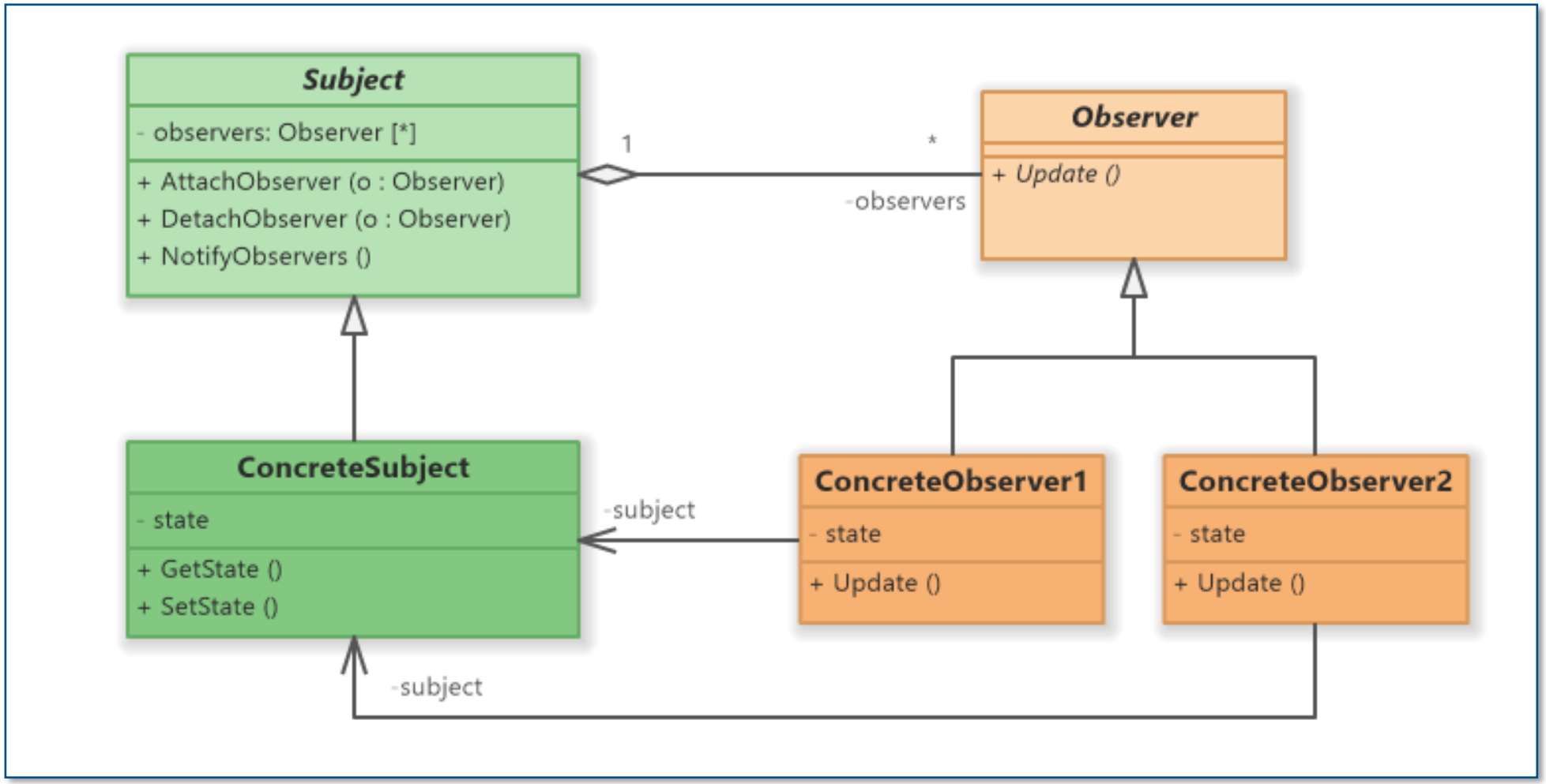
Observer Pattern Agenda

1. What is Observer Pattern
2. Observer Pattern Implementation

What is Observer Pattern

- The observer pattern is pattern design in which object, called the subject, maintain a list of its dependents, called observers, notify them of any changes. Mẫu thiết kế Observer là một mẫu thiết kế trong đó một đối tượng, gọi là Subject, duy trì một danh sách các phụ thuộc của nó, gọi là Observers, và thông báo cho họ về bất kỳ sự thay đổi nào.
  - **Ex:** Company updates all its shareholders for any decision they make. So, Company is Subject and Shareholders are Observers, any changes in policy of company and Company(Subject) notifies all its Shareholder (Observer)

Observer Pattern Implementation



Observer Pattern Implementation

```cpp
#include <iostream>
#include <vector>

// Observer interface
class Observer {
public:
virtual void update(float temperature, float humidity, float pressure) = 0;
};

// Subject (WeatherStation) class
class WeatherStation {
private:
float temperature;
float humidity;
float pressure;
std::vector<Observer*> observers;

public:
void registerObserver(Observer* observer) {
observers.push_back(observer);
}

void removeObserver(Observer* observer) {
// You can implement the removal logic if needed.
}

void notifyObservers() {
for (Observer* observer : observers) {
observer->update(temperature, humidity, pressure);
}
}

void setMeasurements(float temp, float hum, float press) {
temperature = temp;
humidity = hum;
pressure = press;
notifyObservers();
}
};

// Concrete Observer
class Display : public Observer {
public:
void update(float temperature, float humidity, float pressure) {
std::cout << "Display: Temperature = " << temperature
<< "°C, Humidity = " << humidity
<< "%, Pressure = " << pressure << " hPa"
<< std::endl;
}
};
```

```cpp
int main() {
WeatherStation weatherStation;

// Create displays
Display display1;
Display display2;

// Register displays as observers
weatherStation.registerObserver(&display1);
weatherStation.registerObserver(&display2);

// Simulate weather data updates
weatherStation.setMeasurements(25.5, 60, 1013.2);
weatherStation.setMeasurements(24.8, 58, 1014.5);

return 0;
}
```

Microsoft Visual Studio Debug Console

```
Display: Temperature = 25.5° C, Humidity = 60%, Pressure = 1013.2 hPa
Display: Temperature = 25.5° C, Humidity = 60%, Pressure = 1013.2 hPa
Display: Temperature = 24.8° C, Humidity = 58%, Pressure = 1014.5 hPa
Display: Temperature = 24.8° C, Humidity = 58%, Pressure = 1014.5 hPa
```

References

- https://www.geeksforgeeks.org/implementation-of-singleton-class-in-cpp/
- https://www.geeksforgeeks.org/observer-pattern-c-design-patterns/

THANK YOU!