1. 옵저버 패턴(Observer Pattern)

1) 옵저버 패턴

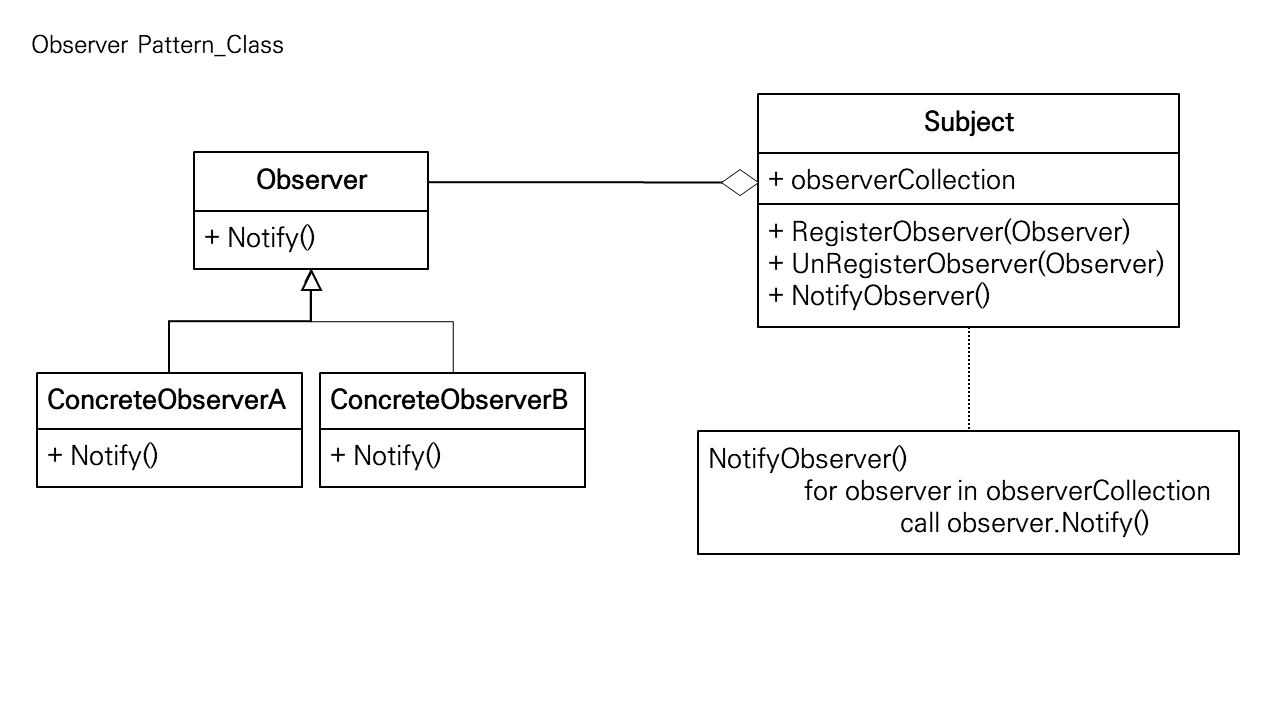
- 객체의 상태 변화를 관찰하는 관찰자들, 즉 옵저버들의 목록을 객체에 등록하여 상태 변화가 있을

때마다 메서드 등을 통해 객체가 직접 목록의 각 옵저버에게 통지하도록 하는 디자인 패턴

- 주로 분산 이벤트 핸들링 시스템을 구현하는데 사용됨

- 발행 / 구독 모델로 알려져 있기도 함

2) 구현



- 패턴 구현의 핵심은 옵저버 또는 리스너(Listener)라 불리는 하나 이상의 객체를 관찰 대상이 되는

객체에 등록시키고, 각각의 옵저버들은 관찰 대상인 객체가 발생시키는 이벤트를 받아 처리함

- 이벤트 발생 시 각 옵저버는 콜백(Callback)을 받음. Notify 함수는 관찰 대상이 발행한 메시지

이외에 옵저버 자신이 생성한 인자 값을 전달할 수 있음

- 주체에는 일반적으로 등록(Register), 제거(Unregister) 메서드가 존재함.

등록 : 새로운 옵저버를 목록에 등록

제거 : 목록에서 옵저버 제거

이외에도 임시로 작동을 멈추거나 재개하는 메서드를 이용해 이벤트가 계속해서 있을 때 홍수같이 발생하는 요청을 제어할 수 있음

- 옵저버 패턴이 많이 쓰인 시스템에서는 순환 실행을 막는 매커니즘 필요

ex) 이벤트 X 발생 시 옵저버 A가 옵저버 B를 갱신한다는 가정. B가 처리를 위해 A를 갱신한다면 A가 이벤트 X를 발생하게 함. 이런 상황을 막기 위해 이벤트 X가 한 번 처리된 후에는 A가 이벤트 X를 다시 발생시키지 않는 방법이 요구됨

3) 대표적인 사례

- 외부에서 발생한 이벤트에 대한 응답(사용자 입력 등), 이벤트 기반 프로그래밍

- 객체의 속성 값 변화에 따른 응답. 종종 콜백은 속성 값 변화를 처리하기 위해 호출될 뿐 아니라

속성 값 또한 바뀜. 때때로 이벤트 연쇄의 원인이 될 수 있음

- 모델 뷰 컨트롤러(Model View Controller) 패러다임과 자주 결합됨. 옵저버 패턴은 MVC에서

모델과 뷰 사이를 느슨히 연결하기 위해 사용됨. 대표적으로 모델에서 일어나는 이벤트를 통보

받는 옵저버는 뷰의 내용을 바꾸는 스위치를 작동시킴

4) 예제

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class AlarmListener

{

public:

virtual void Alarm() = 0;

};

class SensorSystem

{

private:

vector<AlarmListener\*> listeners;

public:

void Attach(AlarmListener \*al)

{

listeners.push\_back(al);

}

void SoundTheAlarm()

{

for (int i = 0; i < listeners.size(); i++)

listeners[i]->Alarm();

}

};

class Lighting : public AlarmListener

{

public:

void Alarm()

{

cout << "lights up" << '\n';

}

};

class Gates : public AlarmListener

{

public:

void Alarm()

{

cout << "gates close" << '\n';

}

};

class CheckList

{

private:

virtual void Localize()

{

cout << " establish a perimeter" << '\n';

}

virtual void Isolate()

{

cout << " isolate the grid" << '\n';

}

virtual void Identify()

{

cout << " identify the source" << '\n';

}

public:

void ByTheNumbers()

{

Localize();

Isolate();

Identify();

}

};

class Surveillance : public CheckList, public AlarmListener

{

private:

void Isolate()

{

cout << " train the cameras" << '\n';

}

public:

void Alarm()

{

cout << "Surveillance - by the numbers:" << '\n';

ByTheNumbers();

}

};

int main()

{

SensorSystem ss;

ss.Attach(&Gates());

ss.Attach(&Lighting());

ss.Attach(&Surveillance());

ss.SoundTheAlarm();

return 0;

}