디자인 패턴

디자인패턴

- 코드 문제 해결을 위해 사용한 기법들을 모아 만들어 졌다

- 싱글톤(Singleton), 옵저버(Observer), 컴포지트(Composite), 팩토리(Factory)가 많이 사용되는 패턴이며 그 외에 여러 디자인 패턴이 존재한다

싱글톤 패턴(Singleton Pattern)

- 가장 많이 사용되는 디자인 패턴
- 하나의 프로그램 내에 하나의 인스턴스만을 생성해야 할 경우 사용
- 가장 문제가 될 소지가 많은 디자인 패턴

싱글톤 패턴(Singleton Pattern)

```
class Singleton
{
private:
    static Singleton * m_pInstance;

public:
    Singleton() { }
    virtual ~Singleton() { }

    static Singleton* get_instance()
    {
          if (NULL == m_pInstance) m_pInstance = new Singleton;
          return m_pInstance;
    }
};
```

Singleton :: get_instance()->…

옵저버 패턴(Observer Pattern)

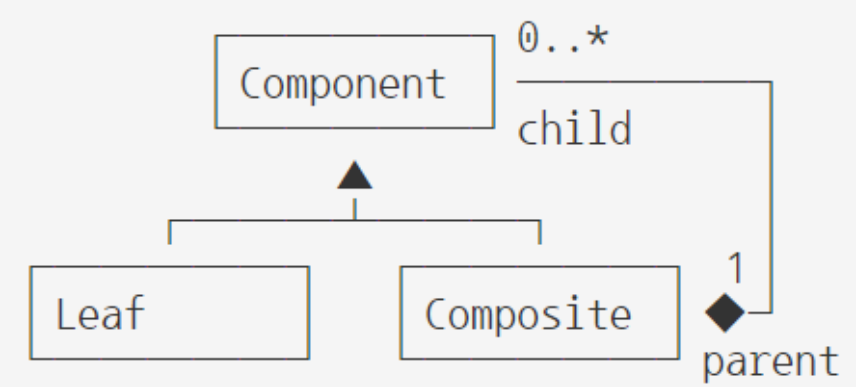
- **일 대 다의 관계**를 가진다
- 옵저버가 **관찰하던 대상**의 상태가 **갱신**되면 **모든 옵저버들**은 갱신된 내용을 알고 **자신의 상태를 갱신**한다
- **관찰자인 observer**와 **관찰 대상인 subject**로 구성된다
- 데이터 전달 방식
  - **대상에서 옵저버**에게 데이터를 **보내는** 방식**(푸시 방식)**
  - **옵저버에서 대상**의 데이터를 **가져오는** 방식**(풀 방식)**

옵저버 패턴(Observer Pattern)

```
class Observer
{
public:
    Observer() {}
    virtual ~Observer() {}

    virtual void update(…) abstract;
};

class Subject
{
public:
    Subject() {}
    virtual ~Subject() { }

    virtual void add(Observer*) abstract;
    virtual void remove(Observer*) abstract;
    virtual void notify(…) abstract;
};
```

컴포지트 패턴(Composite Pattern)

- 객체들의 집합과 단일 객체를 구별 없이 다루게 해준다
- 일부 또는 그룹의 표현하는 **객체들**을 **트리 구조**로 구성한다



- Component
  - 모든 표현할 요소들의 추상적인 인터페이스
- Leaf
  - Component로 지정된 인터페이스를 구현
- Composite
  - Component를 지니고 관리하며, 관리를 위함 함수를 구현

팩토리 패턴(Factory pattern)

- 팩토리 패턴
  - **객체를 생성**하고, 구체적인 **타입을 감춘다**

- 팩토리 메소드(Factory Method)
  - 생성 후 해야 **할 일의 공통점** 정의
  - 생성해야 할 **객체가 한 종류**

- 추상 팩토리(Abstract Factory)
  - 연관된 또는 독립된 **객체들의 타입 군을 생성할 인터페이스를 지니고 있다**
  - 생성해야 할 **객체 군의 공통점**에 유의
  - 생성해야 할 **객체가 여러 종류**