Builder 패턴

타입

Creational Pattern

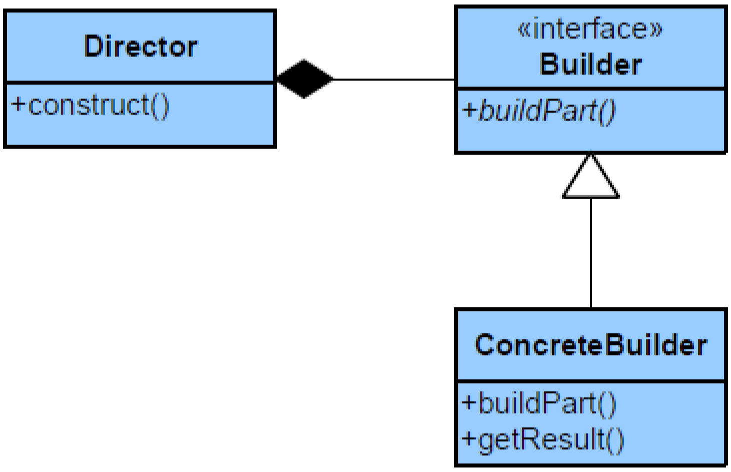
문제

복잡한 객체를 생성하고 싶다. 그러나 인수가 많은 생성자를 쓰고 싶지는 않다.

해결

원하는 객체만 정의하는 멤버 함수를 가지는 중간 클래스를 정의한다.
그리고 중간 클래스 여러개를 조합하여 클라이언트가 원하는 객체를 만드는 최종 클래스를 정의한다.

클래스 다이어그램



예제

builder.cpp

```
#include <string>
#include <iostream>

using namespace std;

// "Product"
```

```
class Pizza
{
        public:
                void setDough(const string& dough)
                {
                        m_dough = dough;
                }
                void setSauce(const string& sauce)
                {
                        m_sauce = sauce;
                }
                void setTopping(const string& topping)
                {
                        m_topping = topping;
                }
                void open() const
                {
                        cout << "Pizza with " << m_dough << " dough, " << m_sauce << " sauce and "
                        << m_topping << " topping. Mmm." << endl;
                }
        private:
                string m_dough;
                string m_sauce;
                string m_topping;
};

// "Abstract Builder"
class PizzaBuilder
{
        public:
                Pizza* getPizza()
                {
                        return m_pizza;
                }
                void createNewPizzaProduct()
                {
                        m_pizza = new Pizza;
                }
                virtual void buildDough() = 0;
                virtual void buildSauce() = 0;
                virtual void buildTopping() = 0;
        protected:
                Pizza* m_pizza;
};

//----------------------------------------------------------------

class HawaiianPizzaBuilder : public PizzaBuilder
{
        public:
                virtual void buildDough()
                {
                        m_pizza->setDough("cross");
                }
                virtual void buildSauce()
                {
                        m_pizza->setSauce("mild");
                }
                virtual void buildTopping()
                {
                        m_pizza->setTopping("ham+pineapple");
                }
};

class SpicyPizzaBuilder : public PizzaBuilder
{
        public:
                virtual void buildDough()
                {
                        m_pizza->setDough("pan baked");
                }
                virtual void buildSauce()
                {
                        m_pizza->setSauce("hot");
                }
                virtual void buildTopping()
                {
                        m_pizza->setTopping("pepperoni+salami");
                }
};

//-------------------------------------------------------------------

class Cook
{
        public:
                void setPizzaBuilder(PizzaBuilder* pb)
                {
                        m_pizzaBuilder = pb;
                }
                Pizza* getPizza()
                {
                        return m_pizzaBuilder->getPizza();
                }
                void constructPizza()
                {
                        m_pizzaBuilder->createNewPizzaProduct();
                        m_pizzaBuilder->buildDough();
                        m_pizzaBuilder->buildSauce();
                        m_pizzaBuilder->buildTopping();
                }
        private:
                PizzaBuilder* m_pizzaBuilder;
};

int main()
{
        Cook cook;
        PizzaBuilder* hawaiianPizzaBuilder = new HawaiianPizzaBuilder;
        PizzaBuilder* spicyPizzaBuilder   = new SpicyPizzaBuilder;

        cook.setPizzaBuilder(hawaiianPizzaBuilder);
        cook.constructPizza();

        Pizza* hawaiian = cook.getPizza();
        hawaiian->open();

        cook.setPizzaBuilder(spicyPizzaBuilder);
        cook.constructPizza();

        Pizza* spicy = cook.getPizza();
        spicy->open();

        delete hawaiianPizzaBuilder;
        delete spicyPizzaBuilder;
        delete hawaiian;
        delete spicy;

        return 0;
}
```

참고

http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns#Builder