Abstract Factory 패턴

같은 클래스를 상속받는 여러 파생 클래스 중 하나의 클래스에 대한 객체를 만드는 유틸리티 클래스. Abstract Factory 패턴의 경우에는 Factory 클래스에서 만들 수 있는 Product가 여러 가지가 있다.

타입

Creational Pattern

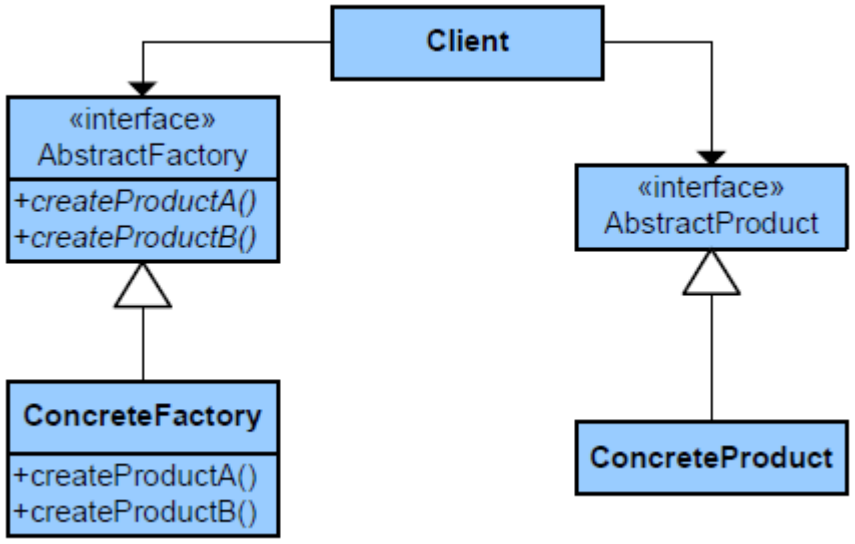
문제

런타임에 성질, 파라메터가 결정되는 객체를 만들고 싶다. 컴파일 타임 이전, 즉 코드 작성 시에는 어떤 클래스의 객체가 초기화될지 알 수 없다.

해결

객체를 만드는 인터페이스를 정의하고 어떤 클래스를 인스턴스화할지 결정하는 서브클래스를 정의한다.

클래스 다이어그램



예제

abstract_factory1.cpp

```
class Computer
```

```
{
public:
    virtual void Run() = 0;
    virtual void Stop() = 0;
};
class Laptop: public Computer
{
public:
    virtual void Run(){mHibernating = false;}
    virtual void Stop(){mHibernating = true;}
private:
    bool mHibernating; // Whether or not the machine is hibernating
};
class Desktop: public Computer
{
public:
    virtual void Run(){mOn = true;}
    virtual void Stop(){mOn = false;}
private:
    bool mOn; // Whether or not the machine has been turned on
};

class ComputerFactory
{
public:
    static Computer *NewComputer(const std::string &description)
    {
        if(description == "laptop")
            return new Laptop;
        if(description == "desktop")
            return new Desktop;
        return NULL;
    }
};
```

abstract_factory2.cpp

```
#include <stdexcept>
#include <iostream>
#include <memory>

class Pizza {
public:
    virtual int getPrice() const = 0;
};

class HamAndMushroomPizza : public Pizza {
public:
```

```
    virtual int getPrice() const { return 850; }
};

class DeluxePizza : public Pizza {
public:
    virtual int getPrice() const { return 1050; }
};

class HawaiianPizza : public Pizza {
public:
    virtual int getPrice() const { return 1150; }
};

class PizzaFactory {
public:
    enum PizzaType {
         HamMushroom,
         Deluxe,
         Hawaiian
    };

    static Pizza* createPizza(PizzaType pizzaType) {
        switch (pizzaType) {
            case HamMushroom:
                return new HamAndMushroomPizza();
            case Deluxe:
                return new DeluxePizza();
            case Hawaiian:
                return new HawaiianPizza();
        }
        throw "invalid pizza type.";
    }
};

/*
 * Create all available pizzas and print their prices
 */
void pizza_information( PizzaFactory::PizzaType pizzatype )
{
        Pizza* pizza = PizzaFactory::createPizza(pizzatype);
        std::cout << "Price of " << pizzatype << " is " <<
pizza->getPrice() << std::endl;
        delete pizza;
}

int main ()
{
        pizza_information( PizzaFactory::HamMushroom );
        pizza_information( PizzaFactory::Deluxe );
        pizza_information( PizzaFactory::Hawaiian );
```

```
}
```

참고

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

From:
http://www.obg.co.kr/doku/ - **OBG WiKi**

Permanent link:
http://www.obg.co.kr/doku/doku.php?id=programming:design_pattern:abstract_factory

Last update: **2020/11/29 14:09**