Factory Pattern

결론 및 정리

Factory Pattern 이란? : 객체 생성에 관련된 디자인 패턴 중 하나. 객체 생성을 하나의 Factory Class 에 위임하는 Pattern.
일반적으로 객체 생성 로직을 서브 클래스에 위임하는 방식을 띔.

Simple Factory

정의

엄밀하게 말하여 Design Pattern 중 하나로 간주할 수 없을 정도로 간단함.
여러 종류의 객체를 생성하여 주는 "Factory" Class 를 만들어, 해당 객체에서 객체의 생성을 담당.
특정 객체를 생성해야하는 다른 Class 들은 해당 Factory Class 에만 접근하여, 해당 객체를 가지고 옴.

구현

1. 여러 객체를 생성하는 Factory Class 를 제작한다.
2. 해당 객체를 생성하여야 하는 Class 에서 Factory Class 에 접근하여 해당 객체를 반환받는다.

Context & Problem & Merit

- 중앙 집중화 : 객체 생성 로직이 한 곳에 집중됨. 해당 위치에서만 수정하면 됨.
- 사용의 간결성 : 클라이언트는 단순히 팩토리에게 원하는 객체를 요청하기만 하면 됨.

Factory Method

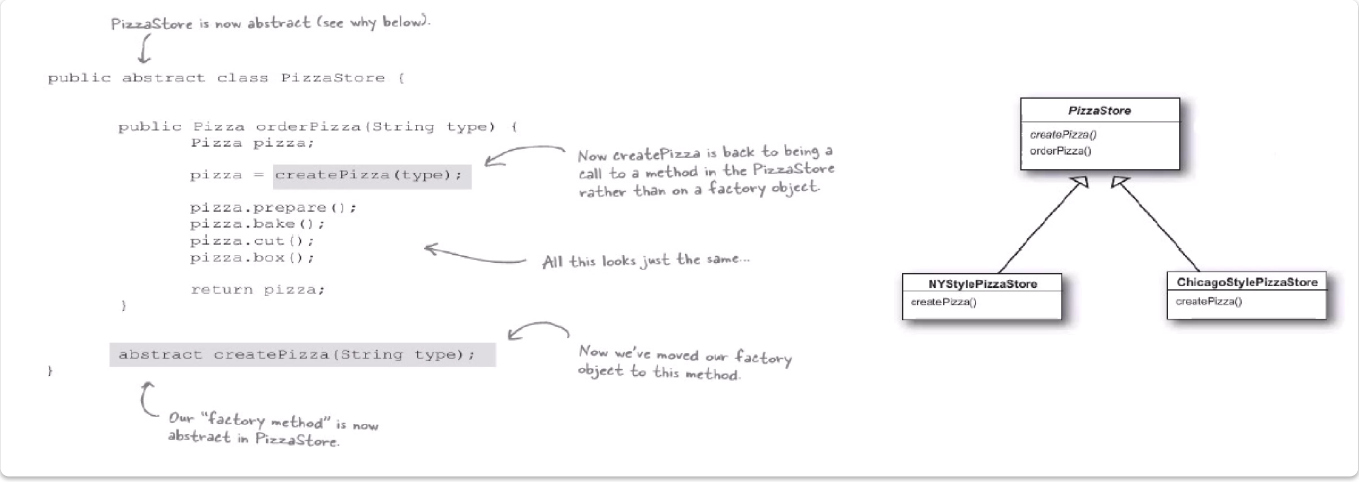
정의

Factory Pattern의 한 변형. 객체 생성에 대한 Method 가 존재하며, 해당 객체 생성 Method 를 서브클래스가 정의하도록 함으로써 생성을 다양화하는 패턴.

- Product : 생성될 객체 (= Product) 들이 구현 / 상속 받을 Interace / Abstract Class.
- ConcreteProduct : Factory Method 가 생성할 객체.
- Creator : 객체 생성에 관한 abstract method (= factoryMethod()) 를 포함하는 Abstract Class. factoryMethod() 를 사용하는 Method 또한 포함하는 것이 일반적이다.

- ConcreteCreator : Creator 를 상속받아, abstract method를 정의하는 Class. 객체 생성에 관한 Method 가 어떻게 구현되었는지를 위임받은 Method 이다.

구조에 대해 좀 더 자세히 포함된 도식



구현

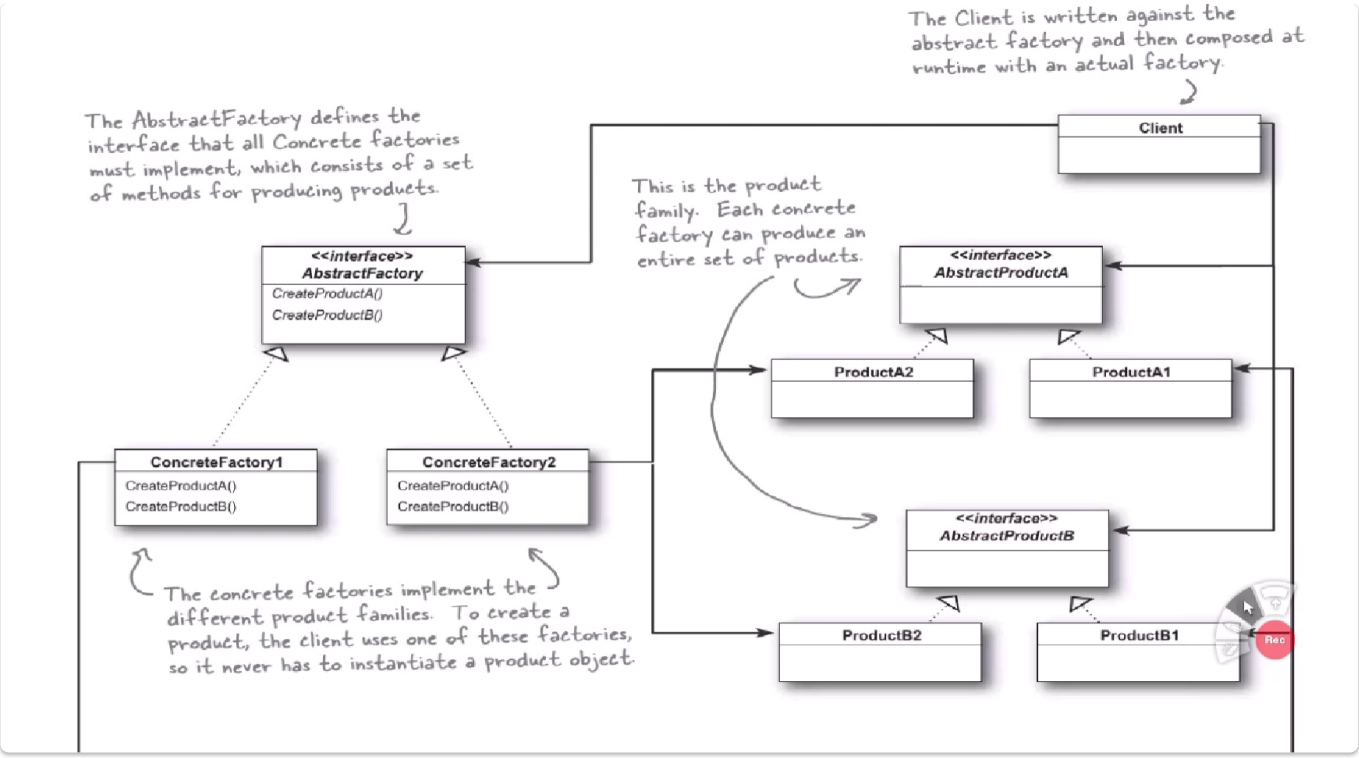
1. Product 타입을 반환하는 abstract method - factory method 가 포함된 creator interface / abstract class 를 만든다.
    1. abstract method 를 사용하는 method 가 creator class 에 포함될 때는 abstract class 로 제작해야 한다.
2. creator 를 구현 / 상속 하며 서로 다른 factory method 를 정의하는 concrete creator 를 만든다.

Context & Problem & Merit

기본적인 Factory Pattern 에서 얻을 수 있는 장점과 동일한 양상을 띈다.

1. **유연성**: 객체 생성 로직을 서브클래스에게 위임함으로써, 슈퍼클래스는 생성되는 객체의 구체적인 타입을 알 필요가 없습니다.
2. **확장성**: 새로운 객체 타입을 추가하려면 해당 객체와 관련된 ConcreteCreator만 추가하면 됩니다.
3. **의존성 분리**: 클라이언트와 객체 생성 로직 사이의 직접적인 의존성을 제거합니다.

Abstract Factory



정의

관련된 객체의 집합을 생성하기 위한 패턴.
위의 사진을 보는 것이 더 정확하게 이해하는 것이 가능하다.

Abstract Product 의 상속으로 묶여진 Product 가 있고. 이 묶음이 A / B / C 가 있다 하자.
(Abstract Product / Product)

Abstract Factory 는 해당 각 종류의 Abstract Product 를 반환하는 Method 가 존재하는 Interface / Abstract Class 이다. (Abstract Factory) 즉, Abstract Factory 는 다양한 종류의 Product 를 생산할 수 있는 Factory인 것이다.

이 Abstract Factory 를 구현 / 상속 하는 Concrete Factory 는 실제 각 Product 묶음(종류) 에 속한 Product 를 생산해내는 것을 정의한다. 이에 따라 실제 각 Concrete Factory 가 무얼 생산하는지 결정된다. (Concrete Factory)

구현

0. 종류를 구분하는 Abstract Product Interface 와 그 세부적으로 존재하는 Product Class 는 이미 존재한다고 생각하자.
1. 원하는 종류의 Abstract Product 를 생산하는 Abstract Factory Interface / Abstract Class 를 만든다.
2. Abstract Factory 를 구현 / 상속하는 Concrete Factory Class 를 제작한다.
   이때, Factory 는 각 abstract product 를 담당하는 method 를 정의하여야 한다. 정의할 때, 각

종류 내부의 어떤 concrete product 를 생산할지를 정의함으로써, concrete factory 를 완성한다.

Context & Problem & Merit

기본적인 Factory Pattern 에서 얻을 수 있는 장점과 동일한 양상을 띈다.

단, 더해진 점은 하나의 Factory 가 다른 여러 종류의 product 를 처리할 수 있다는 점이다.

Design Pattern Principles

강의 들으며

Simple Factory / Factory Method / Abstract Factory 세 가지로 나눠짐.
Creational Pattern 중 하나임.

Dependency Injection 처럼, Dependency 등을 없애기 위한 여러 것들을 이미 알아보았었다.
(Flexible 하게 할 수 없기 때문에. 직접적인 연결관계의 경우, 왜 권장되지 않았더라? => 의존성이 적을 수록 시스템의 한 부분을 변경할 때 더 쉬워지기 때문에. 또한 모듈성으로서 생각했을 때도 최대한 재사용하기 쉽게 만드는 것이 중요하며, 확장성 또한 향상됨.) 일단 나오게 되는 Simple Factory 는 이와 비슷하다.

Simple Factory 는 우리가 Autowiring 을 바탕으로 이미 어느 정도 친숙했던 내용임.

Object 를 만들어서 정리해주는 Class 를 하나 만들어주고. 그 오브젝트를 통해서 다른 Class 를 생성해서 반환해줄 수 있도록 하는 것.

이게 Simple Factory 임.

다음은 바로 factory Method 로 넘어가는데, 시간 없음. 문서에다가 조금 적어놨었으니까, 이걸 한번 보고 바로 넘어가기.

다른 종류의 피자를 만들어 내는 두 개의 Factory 가 있다고 하자. 이때는 Factory Class 가 두 개 있는 것이고.
하지만, 이건 좋은 방법은 아니기 때문에, Factory Method 를 한번 이용해볼 것이다.

Factory Method는 이런 거다.

객체 생성에 관한 Method 가 있고, 그 객체 생성 Method를 사용하는 Method 도 Parent Class 에 존재한다. 하지만, "객체 생성에 관한 Method" 는 abstract method 로 처리되어 있어, subclass 에서 추가적으로 정의하게 된다.

즉, 객체 생성을 서브클래스에게 위임하는 패턴인 것이다.

Factory Method 는 어떤 것을 생성할 지를 subclass 에서 정하도록 하는 Pattern 이다.

Product 에 대한 Interface 도 따로 존재하여, 공통적으로 담아둘 수 있도록 만들어 두는 것이일반적이다. 그 경우, Product Class 를 Sub Class 에서 생성하거나 할 때 담아두기 쉬우니까.

Design Principle 7 :
Depend upon abstractions. Do not depend upon concrete classes.
PizzaStore 를 예로 들어서, PizzaStore 가 모든 종류의 Pizza 에 대해 Dependency 를 가지기 보다, PizzaStore 는 모든 종류의 Pizza 를 다 포함할 수 있는 Pizza 하나만 Dependency 를 가지고, 이 Pizza 에 대해 다른 pizza 들이 상속받아, 이 pizza 에 들어갈 수 있도로 ㄱ하라는.

이 Design Principle 을 이루기 위해 뭘 할 수 있을까?

- new 를 쓰지 말고, factory pattern 을 써라.
  - concrete class 를 type 으로 가지는 변수가 없도록 해라.
- concrete class 로부터 상속을 받는 일은 없도록 하라.
- 이미 구현되어 있는 method 를 다시 override 하는 일이 없도록 하라.

Abstract Factory
Multiple Factory 가 있고, 이 각각의 Factory 들 마다 아마 생성하는 것들의 종류가 다른 것이겠지.

생성해야 하는 것들의 종류가 정해져있고 (각각마다 다른 종류. 다른 Implements 할 Proejct 를 가직 있을 듯), 각 Factory의 종류마다 이 생성해야 하는 Product 의 내용이 조금 달라질 때.

Families of related or dependent objects without specifying concrete classes.

- 서로 관련이 있는 객체들을 통째로 묶어서 팩토리 클래스로 만들고, 이들 팩토리를 조건에 따라 생성하도록 다시 팩토리를 만들어서 객체를 생성하는 패턴.

여러 종류의 Product 가 있고, Product 가 종류별로 이를 대표하는 AbstractProduct 가 있다 하자.
이때, AbstractFactory 는 여러 종류의 Product 를 생산하는 Factory 를 대표하는 Interface Class 다.
ConcreteFactory 는 실제 Product에 접근해 실제 제품을 생산한다.

이건 그림으로서 생각하는 것 이 딱 맞다.