SWIFT
Abstract Factory

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Abstract Factory

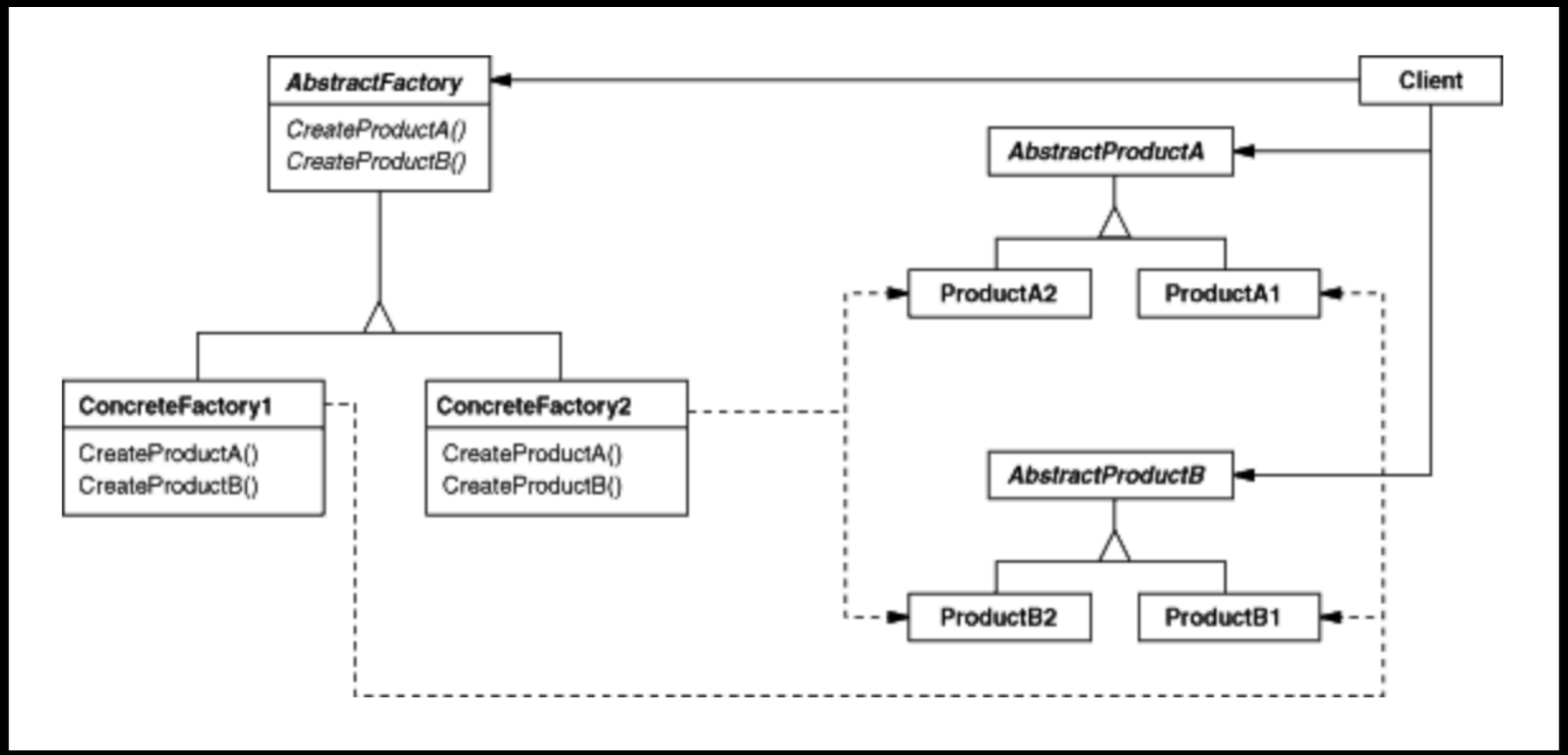
Abstract Factory(추상 팩토리) 디자인 패턴은 연관된 혹은 의존성이 있는 객체의 그룹을 구체적인 클래스를 지정하지 않고 생성하기 위해서 사용하는 패턴입니다.

결국 객체 생성을 추상화하여 이를 사용하는 모듈과 독립적인 인터페이스를 정의할 수 있습니다.

인스턴스를 생성하기 위한 프레임워크와 실제로 인스턴스를 생성하는 클래스를 분리하여 생각할 수 있다.

Abstract Factory

추상 팩토리 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Abstract Factory : Abstract Product에 대한 객체를 생성하는 오퍼레이션으로 인터페이스를 정의하는 추상 클래스 객체

Concreate Factory : 구체적인 제품에 대한 객체를 생성하는 오퍼레이션을 구현하는 객체

Abstract Product : 제품에 대한 인터페이스를 추상적으로 정의하는 객체

Concreate Product : 구체적으로 Factory가 생성할 객체를 정의하고 Abstract Product가 정의하고 있는 인터페이스를 구현하는 객체

Client : Abstract Factory 와 Abstract Product 클래스로 선언된 인터페이스를 사용하는 클라이언트 객체

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
protocol AbstractFactory {
    func createProduct1() -> AbstractProduct
     func createProduct2() -> AbstractProduct
}

class ConcreteFactory1 : AbstractFactory{
    func createProduct1() -> AbstractProduct {
        return ProductA1()
    }

    func createProduct2() -> AbstractProduct {
        return ProductA2()
    }
}

class ConcreteFactory2 : AbstractFactory{
    func createProduct1() -> AbstractProduct {
        return ProductB1()
    }

    func createProduct2() -> AbstractProduct {
        return ProductB2()
    }
}
```

Implementation

```
protocol AbstractProduct {
    func useProduct()
}


class AbstractProductA : AbstractProduct {
    func useProduct() { }
}
```

```
class AbstractProductB : AbstractProduct {
    func useProduct() { }
}
```

Implementation

```
class ProductA1 : AbstractProductA {
    override init() {
        print("ProductA1 init")
    }

    override func useProduct() {
        print("ProductA1 useProduct")
    }
}
```

```
class ProductA2 : AbstractProductA {
    override init() {
        print("ProductA2 init")
    }

    override func useProduct() {
        print("ProductA2 useProduct")
    }
}
```

Implementation

```
class ProductB1 : AbstractProductB {
    override init() {
        print("ProductB1 init")
    }

    override func useProduct() {
        print("ProductB1 useProduct")
    }
}
```

```
class ProductB2 : AbstractProductB {
    override init() {
        print("ProductB2 init")
    }

    override func useProduct() {
        print("ProductB2 useProduct")
    }
}
```

Implementation

```
let factory1 = ConcreteFactory1()
let productA1 = factory1.createProduct1() // ProductA1 init
let productA2 = factory1.createProduct2() // ProductA2 init

let factory2 = ConcreteFactory2()
let productB1 = factory2.createProduct1() // ProductB1 init
let productB2 = factory2.createProduct2() // ProductB2 init

productA1.useProduct() // ProductA1 useProduct
productA2.useProduct() // ProductA1 useProduct
productB1.useProduct() // ProductB1 useProduct
productB2.useProduct() // ProductB2 useProduct
```

References

[1] 추상 팩토리 패턴 (Abstract Factory Pattern in Swift) : https://jerome.kr/entry/abstract-factory-pattern

[2] [Design Pattern] 추상 팩토리(Abstract factory) 패턴 : https://palpit.tistory.com/190

[3] 추상 팩토리 패턴 (Abstract Factory Pattern) : https://victorydntmd.tistory.com/300

[4] [자바 디자인패턴(Java Design Pattern) #5] 추상 팩토리 패턴 (Abstract Factory Pattern) - 제품군을 형성해 보자. : http://oniondev.egloos.com/9663271

[5] 추상 팩토리 패턴(Abstract Factory Pattern) - 자바 디자인 패턴과 JDK 예제 : https://blog.naver.com/PostView.nhn?blogId=2feelus&logNo=220643071966&parentCategoryNo=&categoryNo=28&viewDate=&isShowPopularPosts=false&from=postView

References

[6] 추상 팩토리(Abstract Factory) : https://babamba-playground.tistory.com/37

[7] Objective-c로 구현한 Factory Pattern (Protocol을 이용한 추상 팩토리) : https://thdev.net/321

[8] #2 Abstract Factory Pattern in C++ (추상 팩토리 패턴 C+) : https://vallista.tistory.com/entry/2-Abstract-Factory-Pattern-in-C-추상-팩토리-패턴-C

[9] 디자인 패턴 : 추상 팩토리 vs 팩토리 메소드 : https://www.it-swarm.dev/ko/design-patterns/디자인-패턴-추상-팩토리-vs-팩토리-메소드/970371311/

[10] Design Pattern - Abstract Factory Pattern : https://a292run.tistory.com/entry/003-Design-Pattern-Abstract-Factory-Pattern

Thank you!