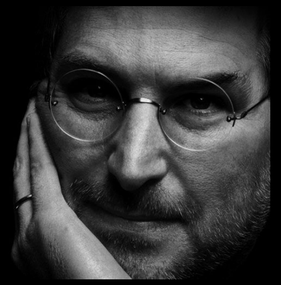
#tasty code

Design Pattern

Creation, Structure, Behavior

Sajae, SeSAC AI를 활용한 iOS 앱 개발
Harry, SeSAC AI를 활용한 iOS 앱 개발

이유1: 디자인 패턴은 개발자의 어휘력이다



: URL을 생성하는 부분이 복잡하네요

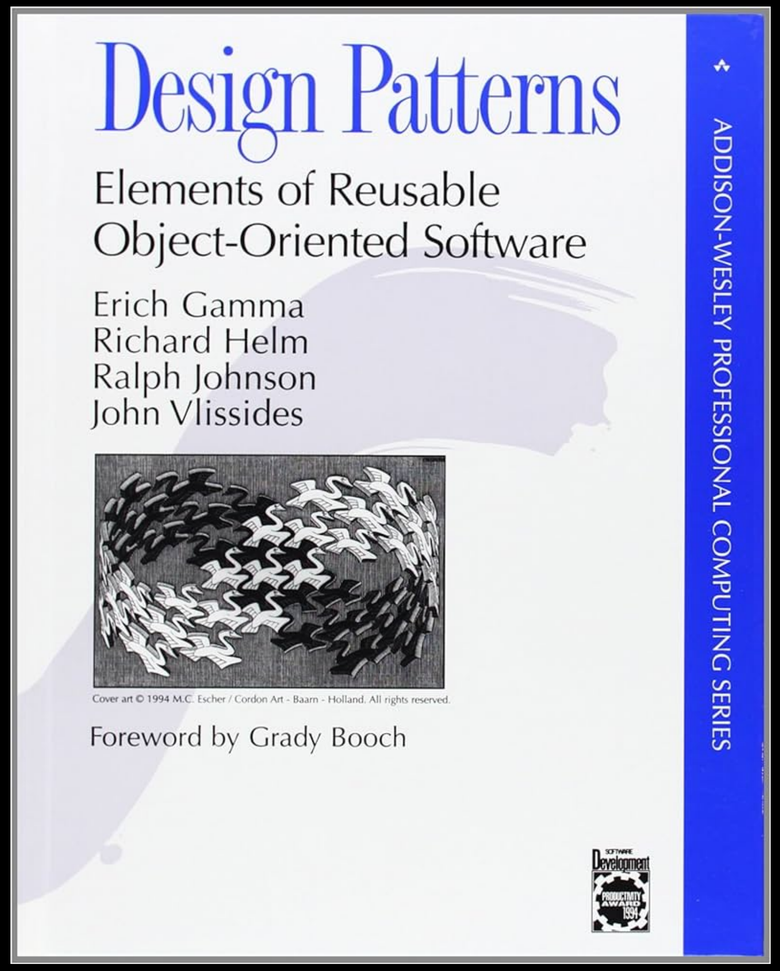
1. URL 생성을 빌더 패턴으로 구현하는 것은 어떨까요?
2. URL 생성을 팩토리 메서드 패턴으로 구현하는 것은 어떨까요?
3. URL 관리를 싱글톤 패턴으로 구현하는 것은 어떨까요?

이유2: 디자인 패턴은 문제 해결 템플릿이다

Design Pattern		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	• Factory Method	• Adapter	• Interperter
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Momento • Observer • State • Strategy • Visitor

개발자가 겪는 문제는 객체의 생성, 확대, 관계짓기로 분류된다

이유2: 디자인 패턴은 문제 해결 템플릿이다



Factory Method

Abstract Factory

Builder

Mediator

Design Pattern		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	• Factory Method	• Adapter	• Interperter
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Momento • Observer • State • Strategy • Visitor

Factory Method

Abstract Factory

Builder

Mediator

Factory Method 는 무슨 목적으로 사용하는 디자인 패턴인가요?

1. 생성

2. 구조

3. 행위

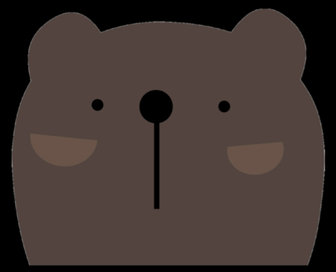
Factory Method 는 무슨 목적으로 사용하는 디자인 패턴인가요?

1. 생성

2. 구조

3. 행위

Factory Method - Why?



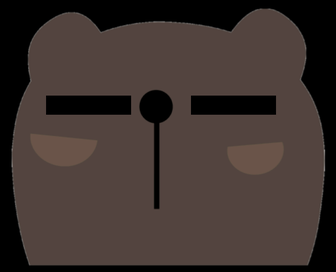
Client

iOS 개발자

Android 개발자

클라우드 개발자

Factory Method - Why?



Client

swift
dart

xcode

iOS 개발자

kotlin
dart

flutter

Android 개발자

????

cloud

amazon

클라우드 개발자

Factory Method - How?

Client

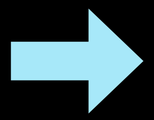
Factory Method - How?

공장 설계도
func make() -> 개발자

Client

Factory Method - How?

Client



공장 설계도
func make() -> 개발자

Factory
Yagom Academy

Factory Method - How?

Client

factory.make()

공장 설계도
func make() -> 개발자

Factory
Yagom Academy

Factory Method - How?

Client

factory.make()

공장 설계도
func make() -> 개발자

Factory
Yagom Academy

iOS 개발자

Android 개발자

클라우드 개발자

개발자

Factory Method - What?

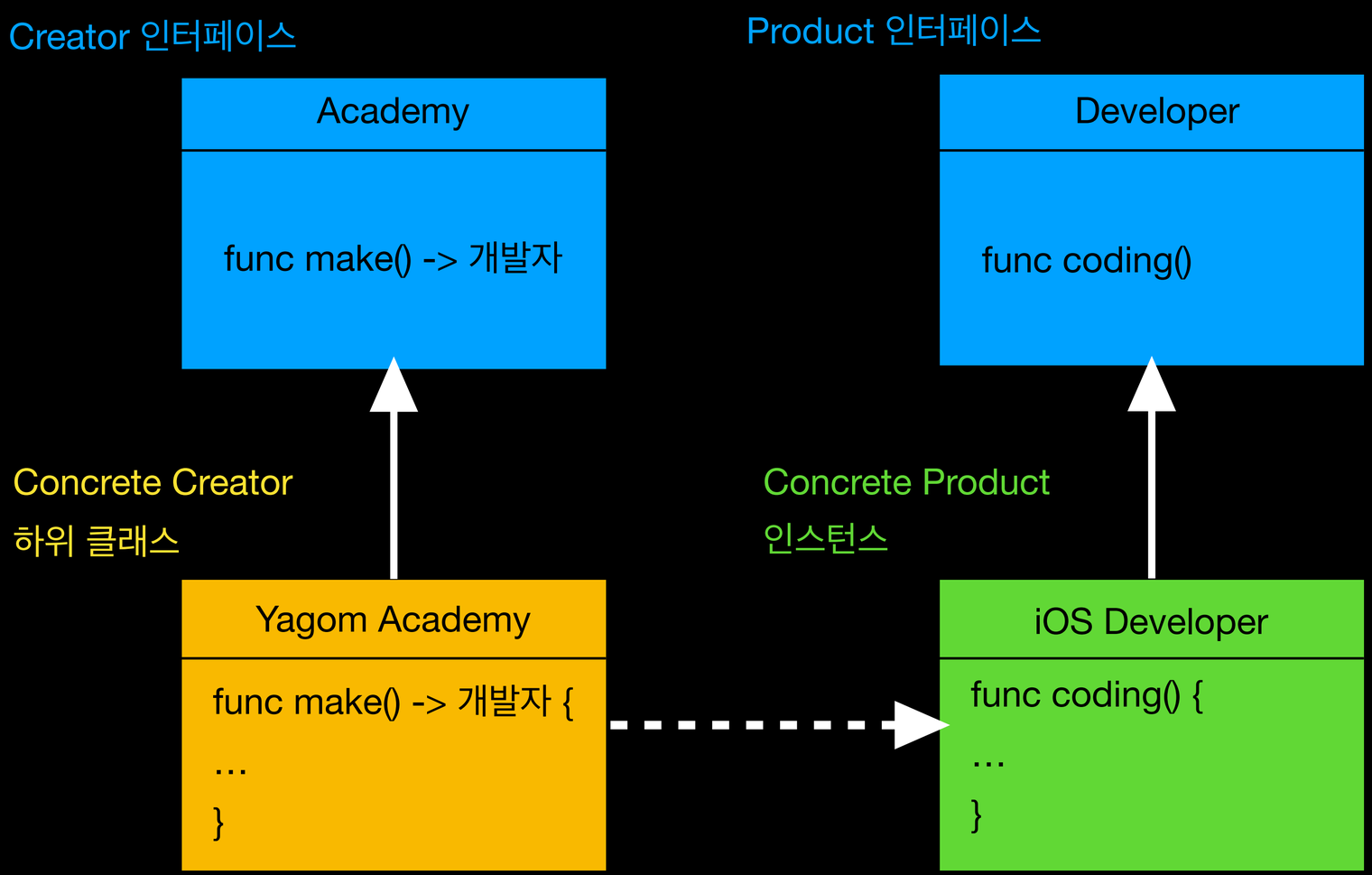
Definition

The Factory Method Pattern defines an interface for creating an object(Instance). But, let subsclasses decide which class to instantiate. Factory method lets the class defer instantiation to subclass

정의

객체를 생성하는 인터페이스는 미리 정의하되, 인스턴스를 만들 클래스의 결정은 서브클래스 쪽에서 내리는 패턴입니다. 팩토리 메서드 패턴에서는 클래스의 인스턴스를 만드는 시점을 서브클래스로 미룹니다.

Factory Method - What?



예제

Factory Method - What?

캡슐화

다형성

의존성 역전

Factory Method

```swift
// 추상 제품(Product)의 인터페이스
protocol Product {
    func use()
}

// 구체적인 제품 A
class ConcreteProductA: Product {
    func use() {
        print("Using Concrete Product A")
    }
}

// 구체적인 제품 B
class ConcreteProductB: Product {
    func use() {
        print("Using Concrete Product B")
    }
}

// 추상 팩토리(Creator)의 인터페이스
protocol Creator {
    func createProduct() -> Product
}
```

```swift
// 구체적인 팩토리 A
class ConcreteCreatorA: Creator {
    func createProduct() -> Product {
        return ConcreteProductA()
    }
}

// 구체적인 팩토리 B
class ConcreteCreatorB: Creator {
    func createProduct() -> Product {
        return ConcreteProductB()
    }
}

// 클라이언트 코드
let creatorA: Creator = ConcreteCreatorA()
let productA: Product = creatorA.createProduct()
productA.use()

let creatorB: Creator = ConcreteCreatorB()
let productB: Product = creatorB.createProduct()
productB.use()
```

Factory Method

Abstract Factory

Builder

Mediator

Abstract Factory 는 무슨 목적으로 사용하는 디자인 패턴인가요?

1. 생성

2. 구조

3. 행위

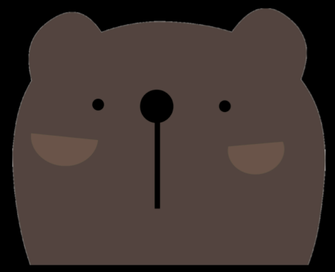
Abstract Factory 는 무슨 목적으로 사용하는 디자인 패턴인가요?

1. 생성

2. 구조

3. 행위

Abstract Factory - Why?



Client

Xcode + 

(네이티브)
iOS 개발자

Flutter + 

(하이브리드)
iOS 개발자

Xcode + 

(하이브리드)
iOS 개발자

Abstract Factory - How?

공장 설계도
func make() -> 개발자
func use() -> 툴

Client

Abstract Factory - How?

공장 설계도
func make() -> 개발자
func use() -> 툴

Client

factory.make()

Native

Hybrid

(네이티브)
iOS 개발자
Xcode

(하이브리드)
iOS 개발자
Flutter

Abstract Factory - How?

	네이티브	하이브리드
Xcode	네이티브 아카데미 네이티브 + Xcode	독학 아카데미1 하이브리드 + Xcode
Flutter	독학 아카데미2 네이티브 + Flutter	하이브리드 아카데미 하이브리드 + Flutter

Abstract Factory - What?

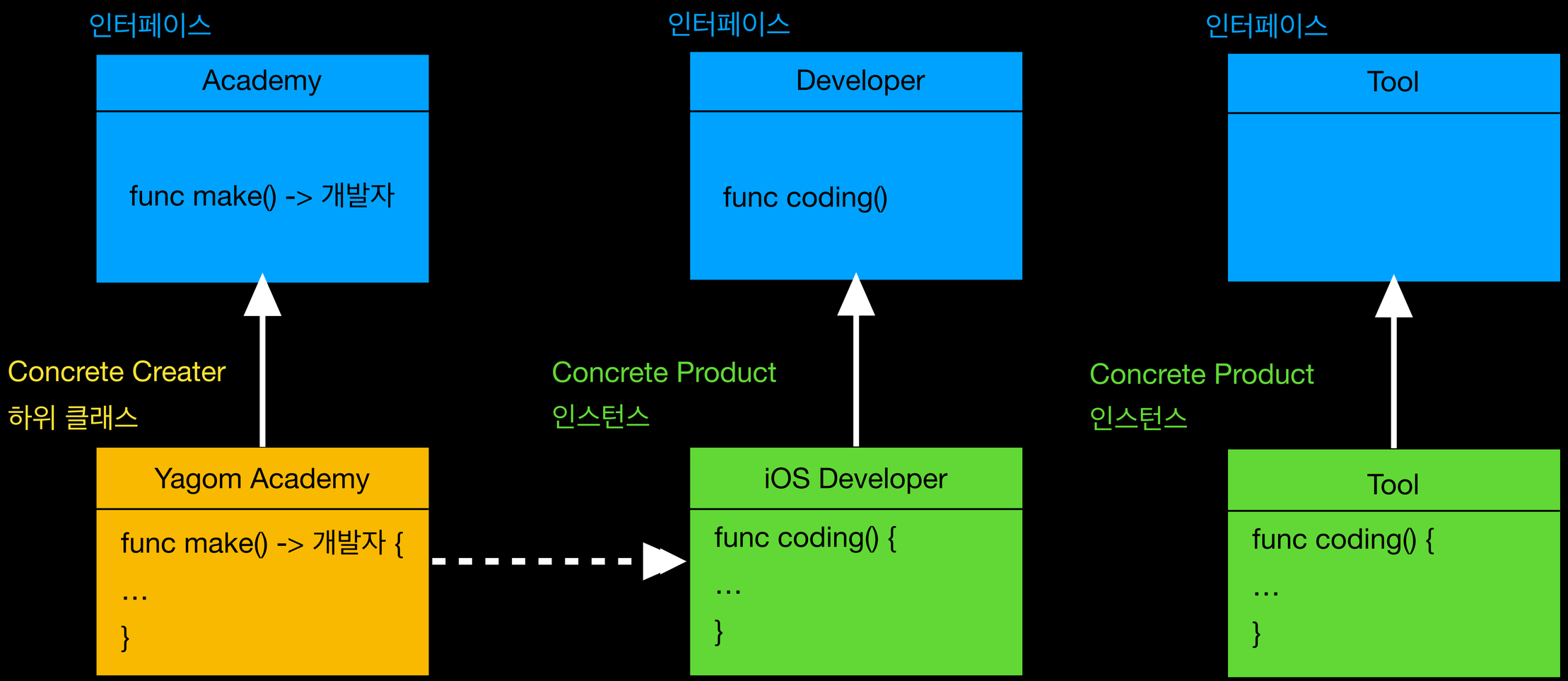
Definition

The abstract factory pattern provides an interface of creating families of related or dependent objects without specifying their concrete classes.

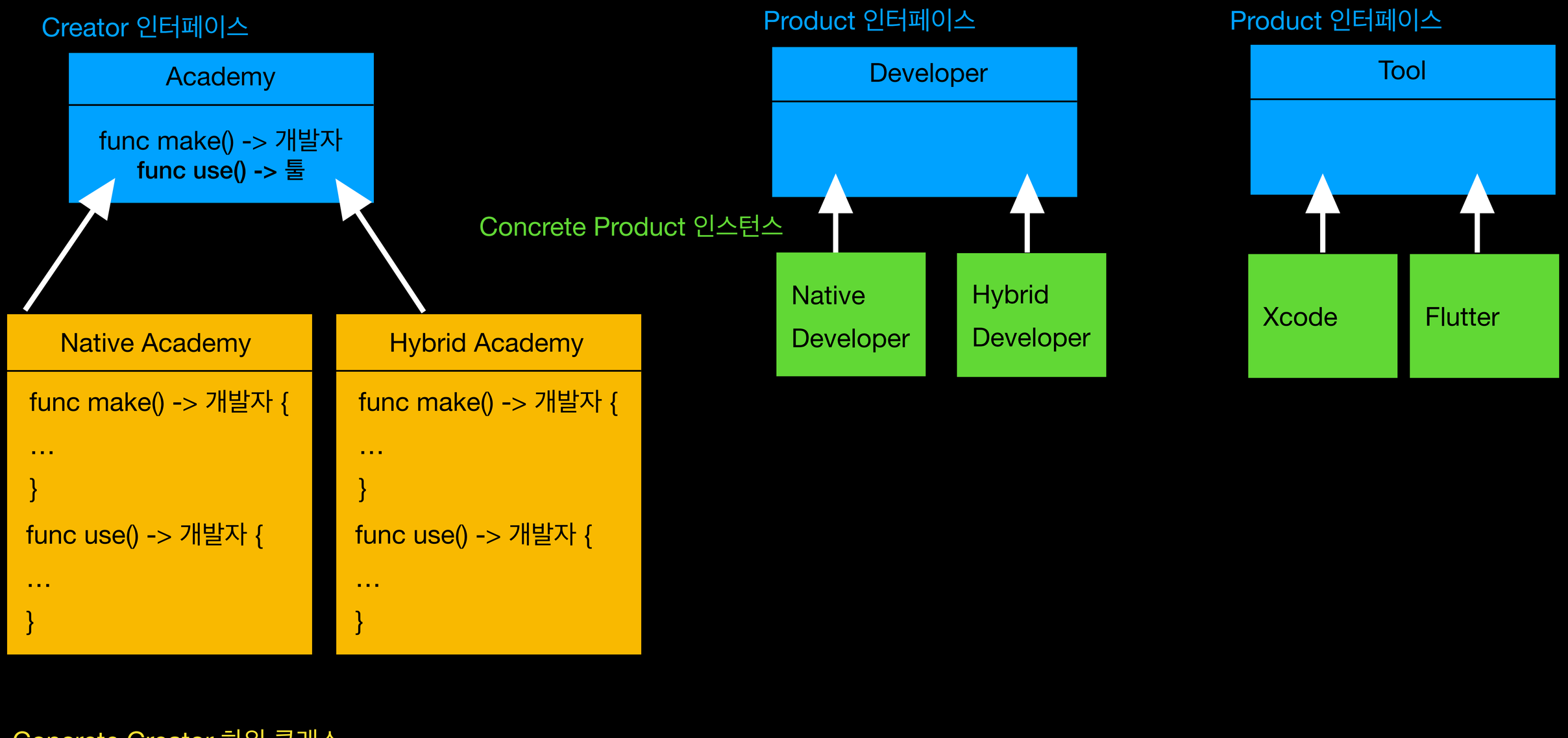
정의

구체적인 클래스를 지정하지 않고 관련성을 갖는 객체들의 집합을 생성하거나 서로 독립적인 객체들의 집합을 생성할 수 있는 인터페이스를 제공하는 패턴입니다.

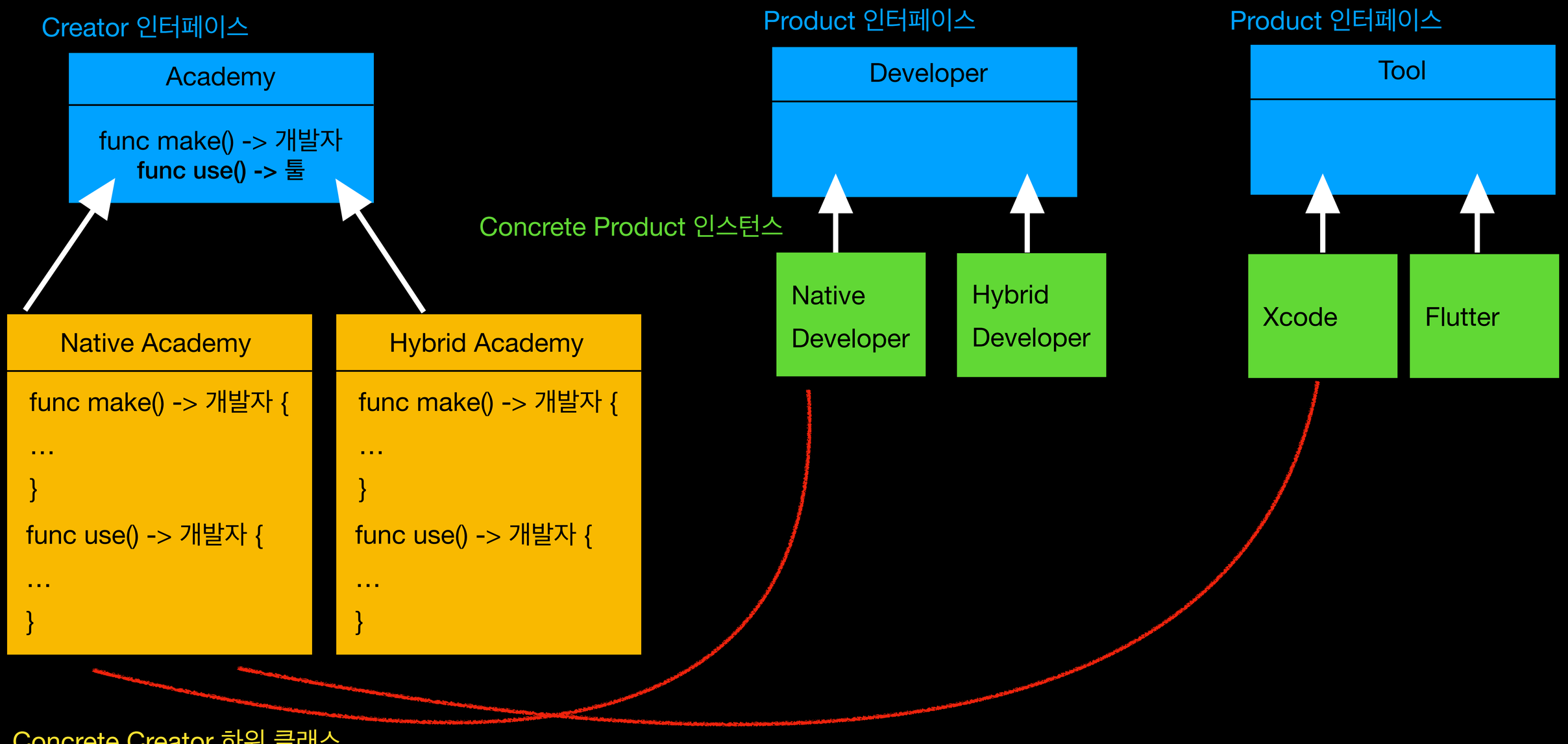
Abstract Factory - What?



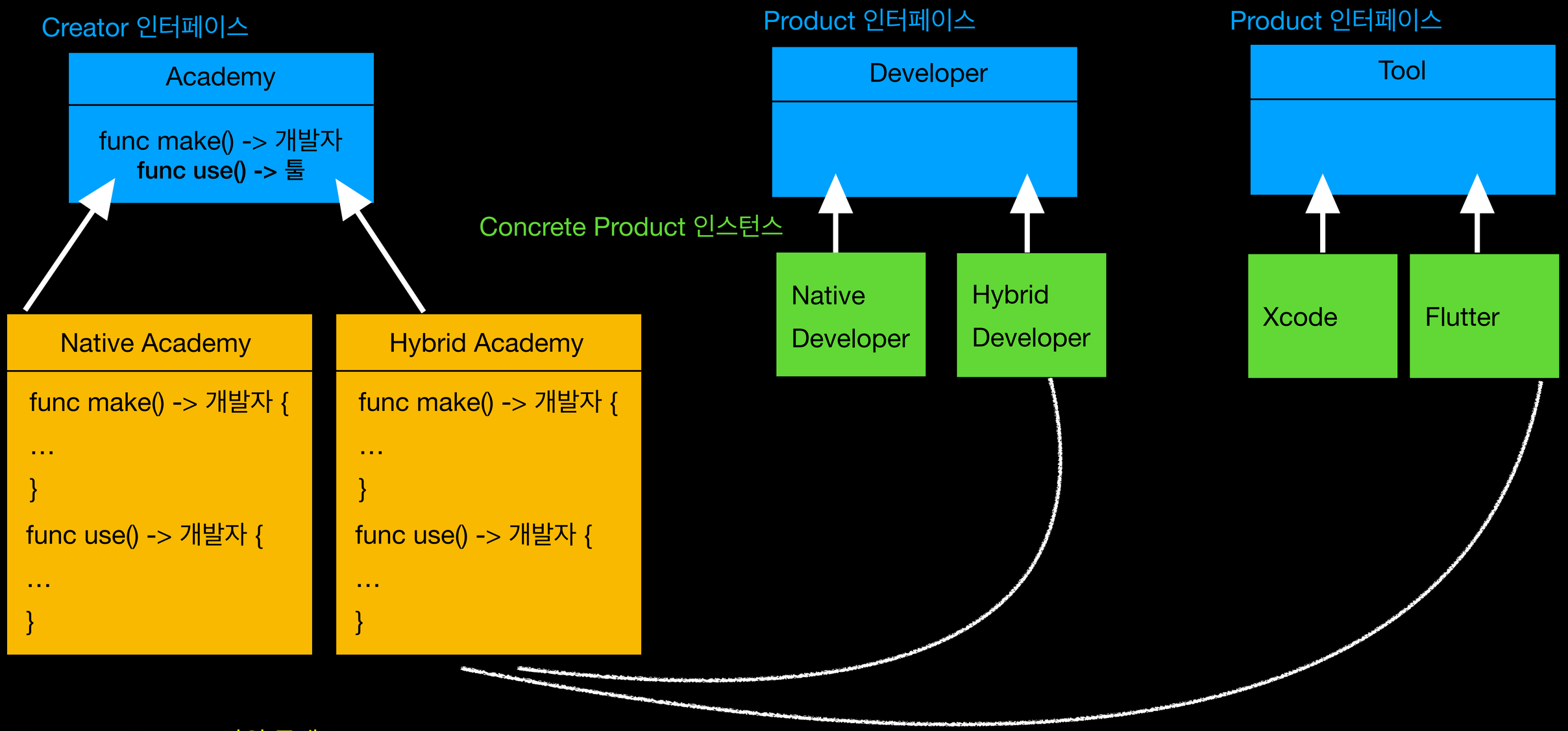
Abstract Factory - What?



Abstract Factory - What?



Abstract Factory - What?



예제

Abstract Factory - What?

캡슐화

다형성

의존성 역전

Abstract Factory - What?

라이트 모드

다크 모드

Abstract Factory

```swift
// 추상 팩토리(Creator)의 인터페이스
protocol AbstractFactory {
    func createProductA() -> AbstractProductA
    func createProductB() -> AbstractProductB
}

// 추상 제품 A의 인터페이스
protocol AbstractProductA {
    func use()
}

// 추상 제품 B의 인터페이스
protocol AbstractProductB {
    func consume()
}

// 구체적인 제품 A1
class ConcreteProductA1: AbstractProductA {
    func use() {
        print("Using Concrete Product A1")
    }
}

// 구체적인 제품 B1
class ConcreteProductB1: AbstractProductB {
    func consume() {
        print("Consuming Concrete Product B1")
    }
}

// 구체적인 제품 A2
class ConcreteProductA2: AbstractProductA {
    func use() {
        print("Using Concrete Product A2")
    }
}
```

```swift
// 구체적인 제품 B2
class ConcreteProductB2: AbstractProductB {
    func consume() {
        print("Consuming Concrete Product B2")
    }
}

// 구체적인 추상 팩토리
class ConcreteFactory1: AbstractFactory {
    func createProductA() -> AbstractProductA {
        return ConcreteProductA1()
    }

    func createProductB() -> AbstractProductB {
        return ConcreteProductB1()
    }
}

class ConcreteFactory2: AbstractFactory {
    func createProductA() -> AbstractProductA {
        return ConcreteProductA2()
    }

    func createProductB() -> AbstractProductB {
        return ConcreteProductB2()
    }
}
```

```swift
// 클라이언트 코드
let factory1: AbstractFactory = ConcreteFactory1()
let productA1: AbstractProductA = factory1.createProductA()
let productB1: AbstractProductB = factory1.createProductB()

productA1.use()
productB1.consume()

let factory2: AbstractFactory = ConcreteFactory2()
let productA2: AbstractProductA = factory2.createProductA()
let productB2: AbstractProductB = factory2.createProductB()

productA2.use()
productB2.consume()
```

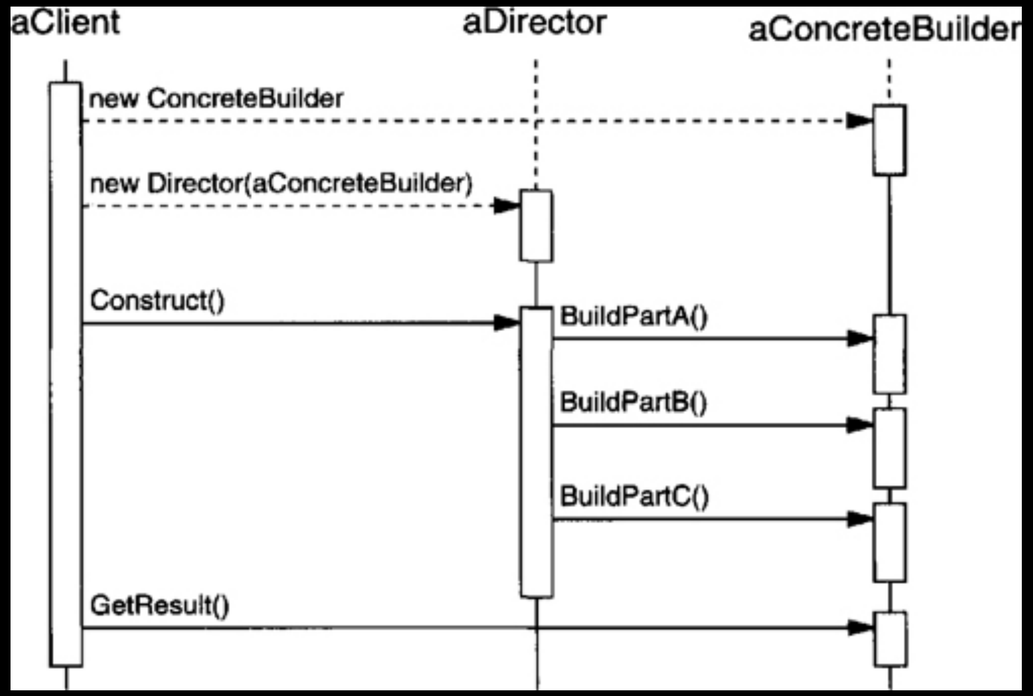
Factory Method

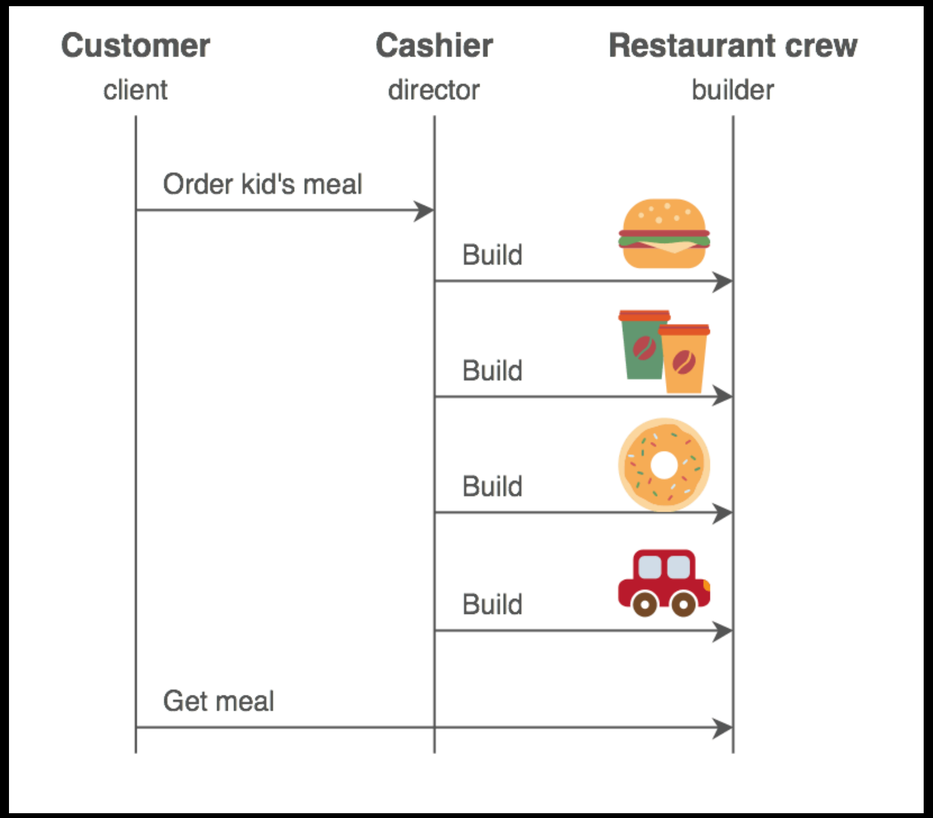
Abstract Factory

Builder

Mediator

Builder Pattern





Builder Pattern

```
public func foregroundColor(_ color: Color?) -> Text
```

```
public func font(_ font: Font?) -> Text
```

```
3
4   import SwiftUI
5
6   struct Example: DisclosureGroupStyle {
7       func makeBody(configuration: Configuration) -> some View {
8           Text("Lorem ipsum")
9               .foregroundColor(Color.red)
10              .font(.title)
11              .animation(nil, value: configuration.isExpanded)
12              .lineSpacing(50)
13      }
14  }
15
```

빌더 패턴은 내부구현에 자율성, 객체 생성 과정에
주도권을 주고 생성과 재현을 분리시킨다.

(아주) 간단한

예제 라이브 코딩

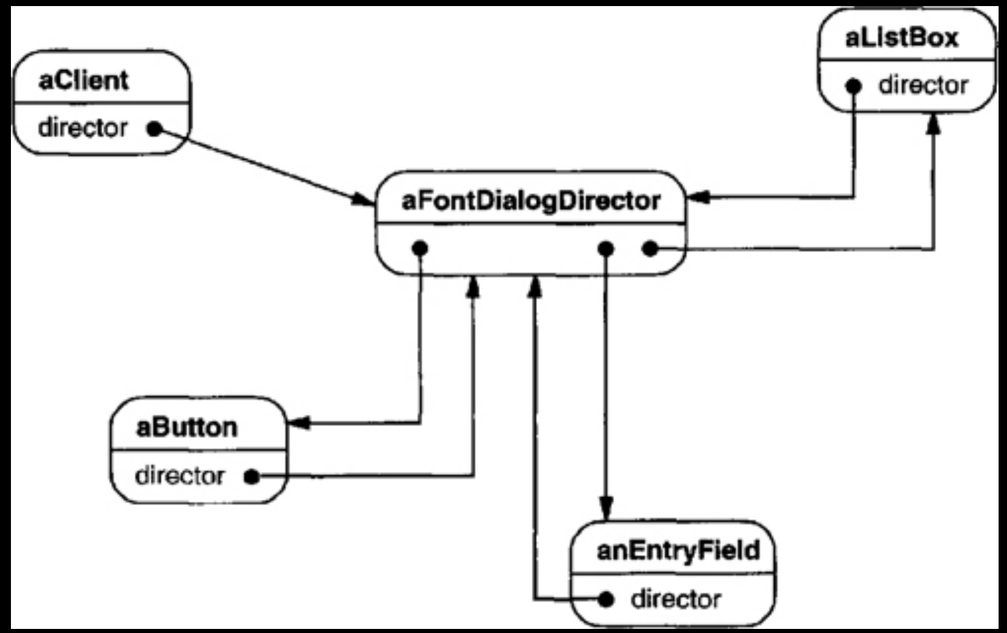
Factory Method

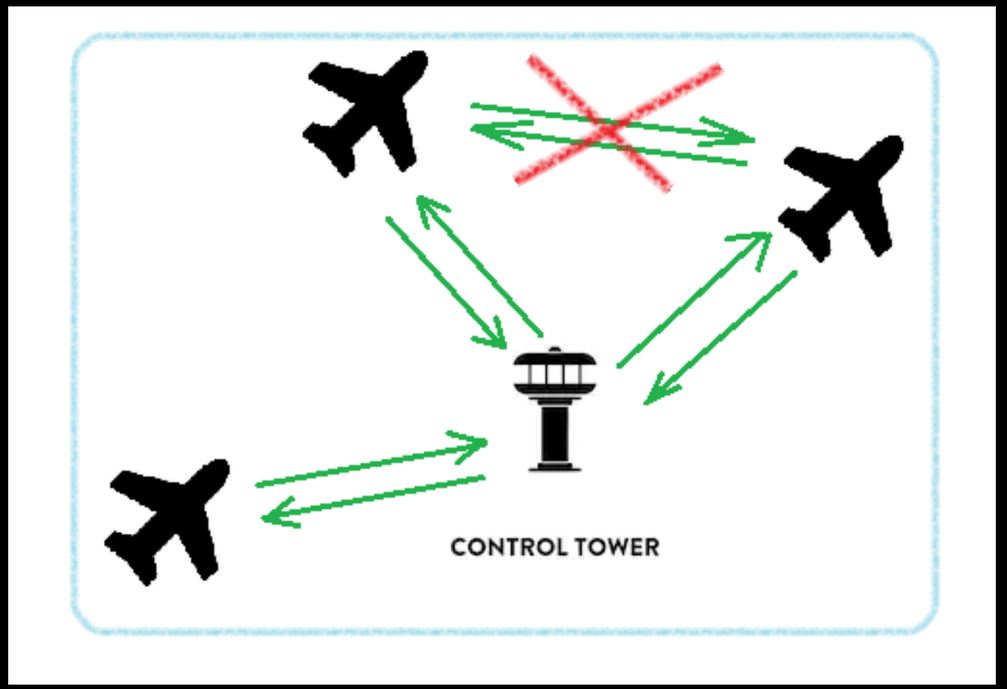
Abstract Factory

Builder

Mediator

Mediator Pattern





Mediator Pattern

Figure 4-7 Cocoa version of MVC as a compound design pattern

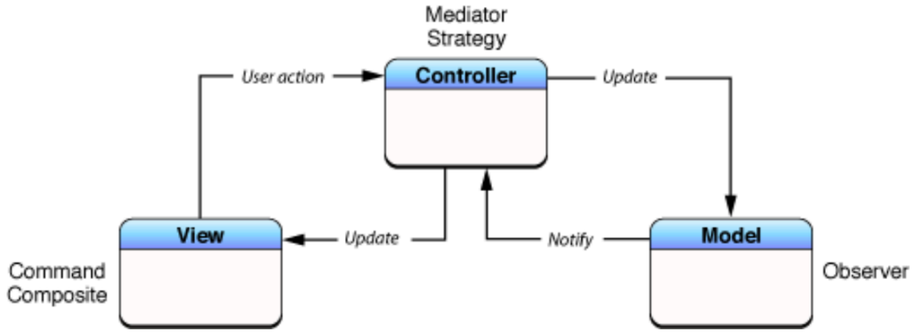
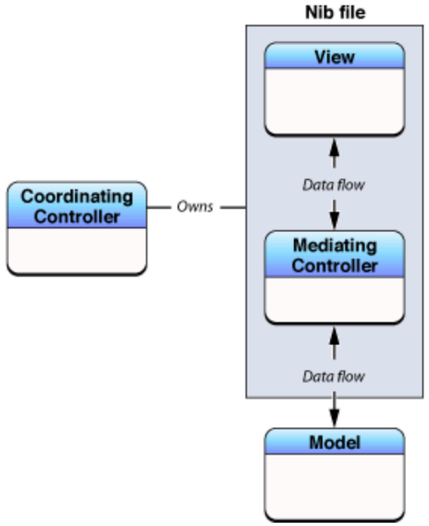


Figure 4-8 Coordinating controller as the owner of a nib file



중재자 패턴은 같은 그룹으로 분류되는 객체들이

중재자 객체만 소유함으로써 서로 소통함에 있어서

의존성을 최소화한다

출처: Cocoa Fundamentals Guide

(아주) 간단한

예제 라이브 코딩

More Information

https://github.com/jeonyeohun/Design-Patterns-In-Swift

https://github.com/ochococo/Design-Patterns-In-Swift?tab=readme-ov-file

https://velog.io/@suojae0516/posts?tag=디자인패턴

GoF의 디자인 패턴 - 에릭 감마, 리처드 헬름, 랄프 존슨, 존 블리시디스

SeSAC24

tasty code