SWIFT Builder

Bill Kim(김정훈) | ibillkim@gmail.com

목차

Builder

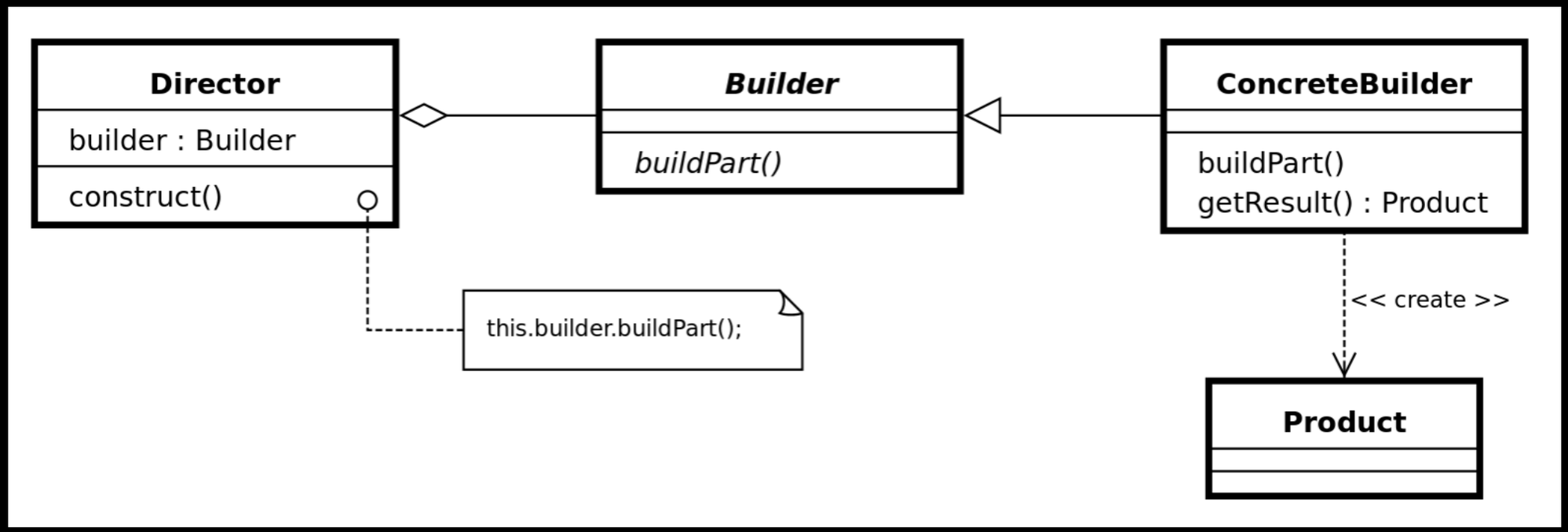
Builder(빌더) 디자인 패턴은 복합 객체의 생성 과정과 표현 방법을 분리하여 동일한 생성 절차를 통하여 서로 다른 결과를 만들 수 있게 해주는 패턴입니다.

즉 생성 절차는 항상 동일하되 결과는 다르게 만들어주는 디자인 패턴입니다.

또한 객체 내의 여러 속성들에 대해서 체이닝 형식으로 생성할 수도 있습니다.

Builder

빌더 패턴을 UML로 도식화하면 아래와 같습니다.



Structure

Director : 객체 생성 방식에 대한 책임을 가진 객체

Builder : 객체를 생성하는 추상 인터페이스 객체

ConCreateBuilder : 제품에 대한 인터페이스를 추상적으로 정의하는 객체

Product : Builder를 이용해서 Director가 만들어 낸 최종 객체

Implementation

구체적인 구현에 대해서 소스 코드를 통하여 살펴봅니다.

```
class Product {
    var value1: Int = 0
    var value2: Int = 0
    var value3: Int = 0
}
```

```
protocol Builder {
    var product:Product { get set }

    func build() -> Builder
    func getProduct() -> Product

    func setValue1(value: Int) -> Builder
    func setValue2(value: Int) -> Builder
    func setValue3(value: Int) -> Builder
}
```

Implementation

```
class ConCreateBuilderA : Builder {
    var product = Product()

    func getProduct() -> Product {
        return product
    }

    func build() -> Builder
    {
        setValue1(value: 1)
        setValue2(value: 2)

        return self
    }

    func setValue1(value: Int) -> Builder {
       product.value1 = value
       return self
    }

    func setValue2(value: Int) -> Builder {
       product.value2 = value
       return self
    }

    func setValue3(value: Int) -> Builder {
       product.value3 = value
       return self
    }
}
```

Implementation

```
class ConCreateBuilderB : Builder {
    var product = Product()

    func getProduct() -> Product {
        return product
    }

    func build() -> Builder
    {
        setValue1(value: 10)
        setValue2(value: 20)
        setValue2(value: 30)
         return self
    }

    func setValue1(value: Int) -> Builder {
       product.value1 = value
       return self
    }

    func setValue2(value: Int) -> Builder {
       product.value2 = value
       return self
    }

    func setValue3(value: Int) -> Builder {
       product.value3 = value
       return self
    }
}
```

Implementation

```
class Director {
    var builder: Builder

    init(builder: Builder) {
        self.builder = builder
    }

    // Construct
    func build() -> Builder {
        return builder.build()
    }
}
```

Implementation

```swift
let product1 = Director(builder:ConCreateBuilderA()).build()
print("product1.value1 : \(product1.getProduct().value1)")
// product1.value1 : 1

let product2 = Director(builder:ConCreateBuilderB()).build()
print("product2.value1 : \(product2.getProduct().value1)")
// product2.value1 : 10

let product3 = Director(builder:ConCreateBuilderB()).build()
// 체이닝 방식으로 각 속성 재설정 가능
product3.setValue1(value: 100).setValue2(value: 200).setValue3(value: 300)
print("product3.value1 : \(product3.getProduct().value1)")
// product3.value1 : 100
```

References

[1] 스위프트에서 빌더 패턴 구현 : https://dev3m.tistory.com/entry/스위프트에서-빌더-패턴-구현

[2] Builder pattern(swift) : https://linsaeng.tistory.com/7

[3] [DP] 2. 빌더 패턴(Builder Pattern) : https://asfirstalways.tistory.com/350

[4] Top 5 스위프트 디자인 패턴 (번역) : https://leejigun.github.io/Top_5_Design_Patterns

[5] 빌더 패턴 Builder Pattern : https://dev-momo.tistory.com/entry/빌더-패턴-Builder-Pattern

References

[6] iOS와 디자인 패턴 : https://10apps.tistory.com/153

[7] 디자인 패턴[1] : 빌더 패턴 Builder pattern : https://altongmon.tistory.com/508

[8] [Design Pattern] 빌더(Builder) 패턴 - 디자인 패턴 : https://palpit.tistory.com/191

[9] The Builder Pattern : https://medium.com/jeremy-codes/the-builder-pattern-eef3351bcae9

[10] 빌더 패턴(Builder Pattern) : https://johngrib.github.io/wiki/builder-pattern/

Thank you!