출처

- https://refactoring.guru/design-patterns/factory-method

Factory Method란?

- 생성 디자인 패턴 (creational design pattern)
- super class에서 object를 생성할 수 있는 interface를 제공하고, super class를 구현하는 파생 클래스에서는 object의 타입을 바꿀 수 있도록 하는 패턴입니다.

상황

- 트럭을 이용한 기존 물류 앱이 있습니다.
- 이 사업이 번창하여 앱에 배(Ship)을 추가하여 해상 물류 서비스도 지원하려고 합니다.
- 하지만 Truck이 전체 시스템과 강하게 **Coupling**되어 있어 Ship을 추가하기 어렵게 됩니다.

해결책

문제의 포인트는

1. 시스템이 truck이라는 구체적인 클래스에 의존하고 있습니다.
    - **생성** 및 **사용**함으로써 의존합니다.

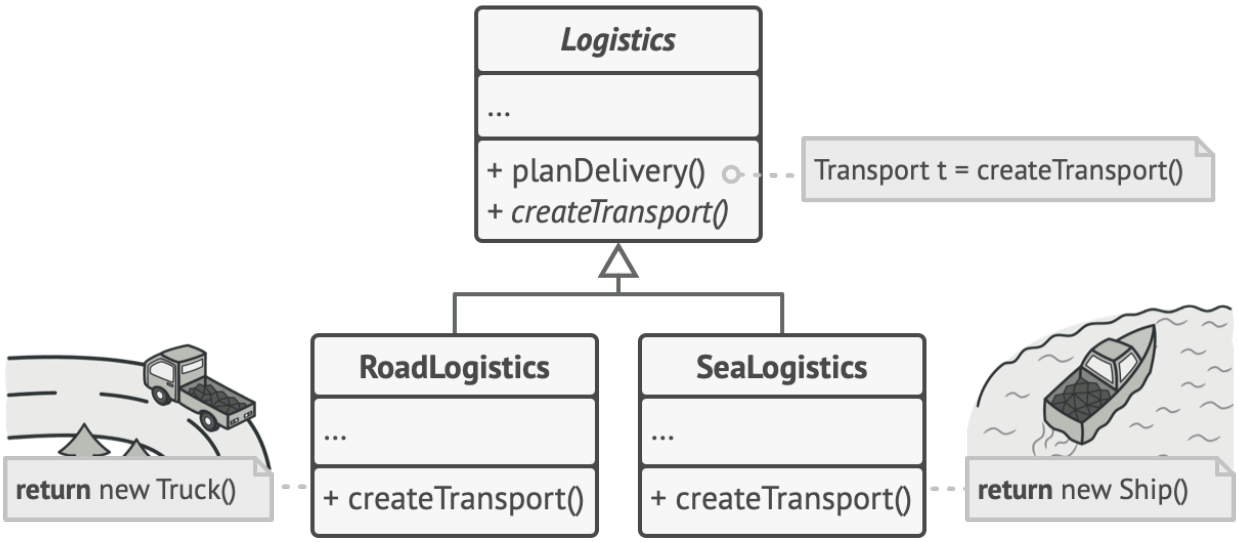
라 말할 수 있습니다. 따라서 이 문제의 해결책으로,

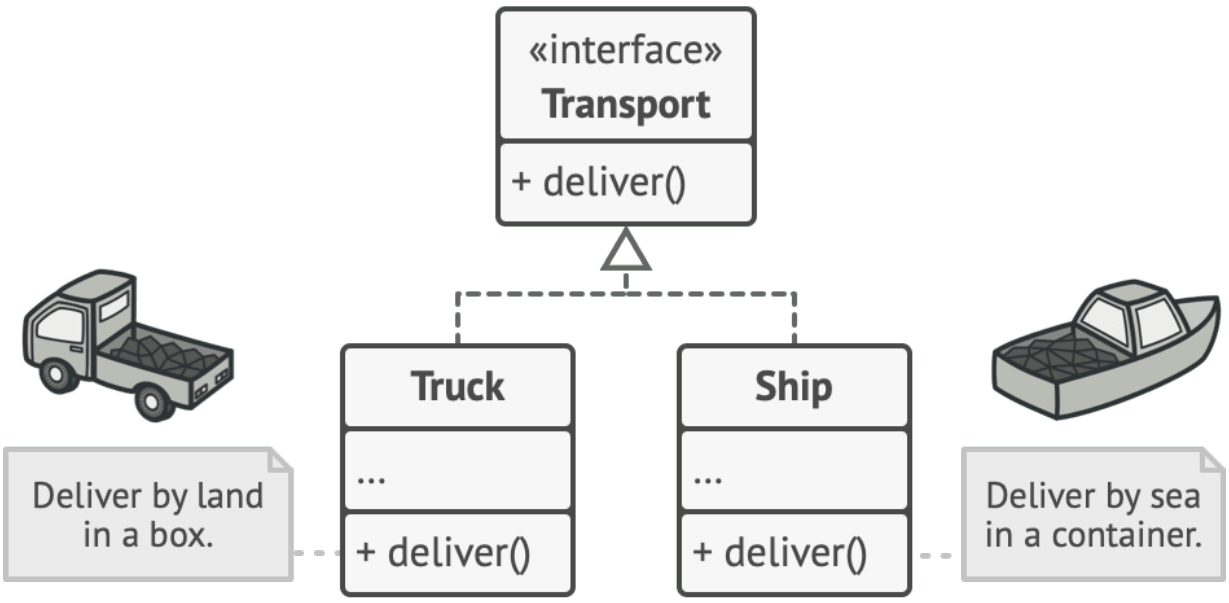
1. client에서 직접적으로 truck을 생성하는 call(new Truck)을 factory method로 대체합니다.

2. 그리고 Truck과 Ship은 추상화시켜 Transport라는 인터페이스를 구현하도록 합니다.

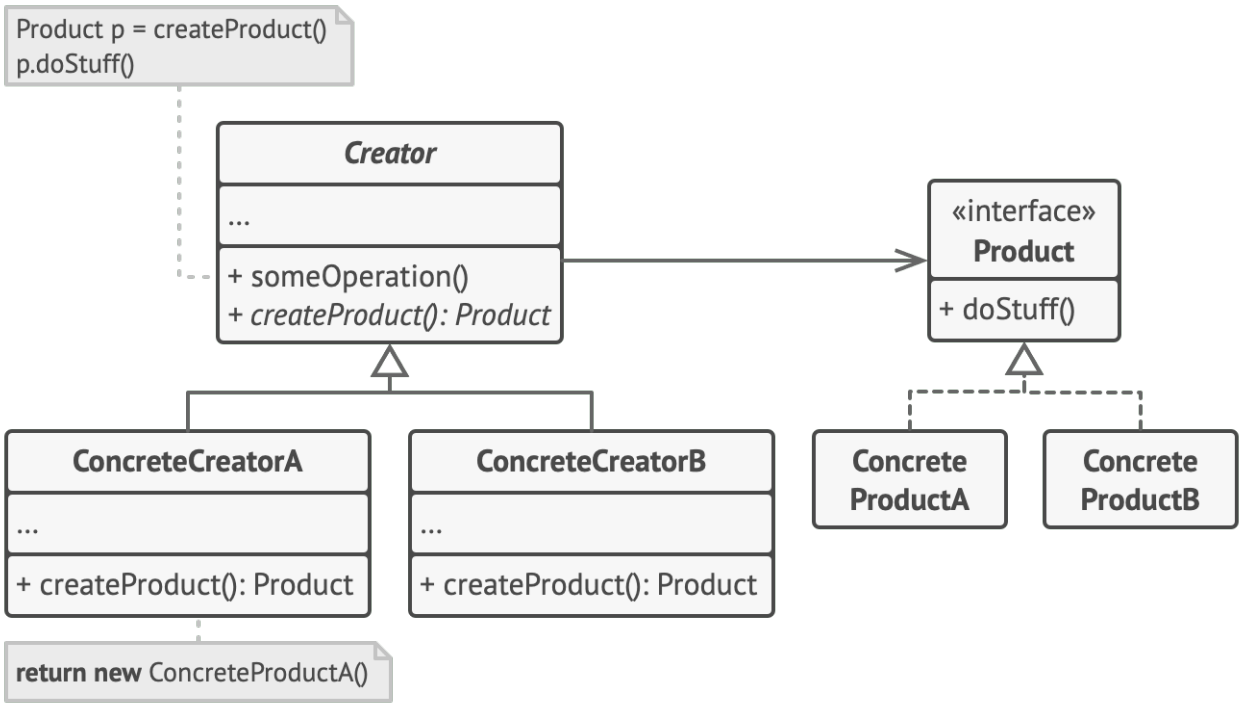
truck을 생성할 지, ship을 생성할 지는 factory method를 오버라이딩 함으로써 구현할 수 있습니다.

> **[용어 정리]** object를 생성해주는 method를 **factory method**라 부르고, 이 method로 인해 생성되는 object를 **product**(truck,ship)라 합니다.





구조



Product

- interface로 선언 합니다.

Concrete Products

- product에 대해 서로 다르게 구현합니다.

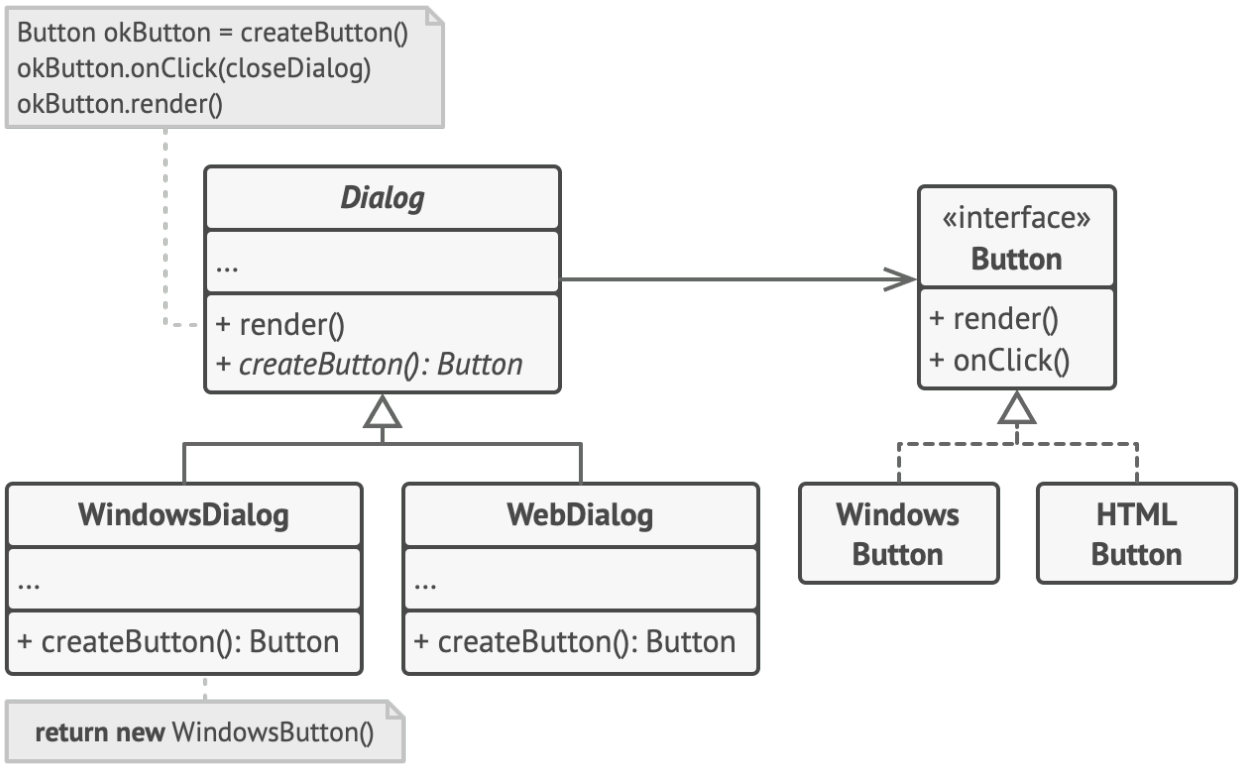
Creator

- factory method를 생성합니다.
- facory method의 **Return type**을 지정하는 것이 중요합니다. 다시 말해 Products의 추상화된 type을 지정하는 것이 중요한 것이죠.
- 이름과 다르게, 이 클래스의 주요 책임은 객체를 생성하는 것만 아닌, products와 관련된 business logic 또한 가지고 있습니다. factory method가 생성 로직과 비즈니스 로직이 분리되는 것을 돕는 것이죠.

Concrete Creators

- 서로 다른 타입의 product를 반환할 수 있도록 factory method를 오버라이드합니다.
- **반드시 새로운 instance를 반환하는 것이 아닙니다**. 이미 생성된 instance를 반환할 수 있습니다.

또 다른 예



- render()

Button을 이용하여 대화 창을 rendering하는 **business logic**을 나타냅니다.

- createButton()

  Button Interface를 구현하는 클래스의 인스턴스를 생성하는 **factory method**

WindowsDialog, WebDialog일 경우에 비즈니스 로직은 똑같이 작동합니다.

그렇기 때문에 WindowsDialog를 구현할 경우, facotry method가 생성하는 Product는 WindowsButton을 생성해주도록하고, WebDialog를 구현할 경우엔 HTMLButton을 생성해주도록만 한다면, 수정없이 새로 추가가 가능해집니다.

어떤 상황에 적용할까?

1. Client code가 products의 Type을 정확히 모르고 작동해야할 때
2. 유저에게 라이브러나 프레임워크의 내부 component를 확장하기 위한 방법으로 사용할 때
   - ex:) UIButton을 제공하는 프레임워크라면, 사용자가 RoundButton을 사용하고 싶을 때, 단지 Button interface를 RoundButton으로 구현하고, factory mthod에서 이를 오버라이딩하여 RoundButton을 return하도록 만들면 됩니다.
3. 기존의 instance를 재사용하고 싶을 때
   - factory method creation에서 해당 로직을 적용해주면 됩니다.

장, 단점

장점

1. 생성하는 코드를 프로그램의 다른 부분으로 옮김으로써 SRP원칙 따릅니다.
2. 새로운 타입의 products를 생성할 때마다 기존 코드를 수정할 필요가 없으므로 OCP따릅니다.

단점

1. 새로운 object type이 나타날 때마다 subclass를 만들어야 하므로 복잡해집니다.