Abstract Factory / Factory Method Pattern

문관경

Abstract Factory

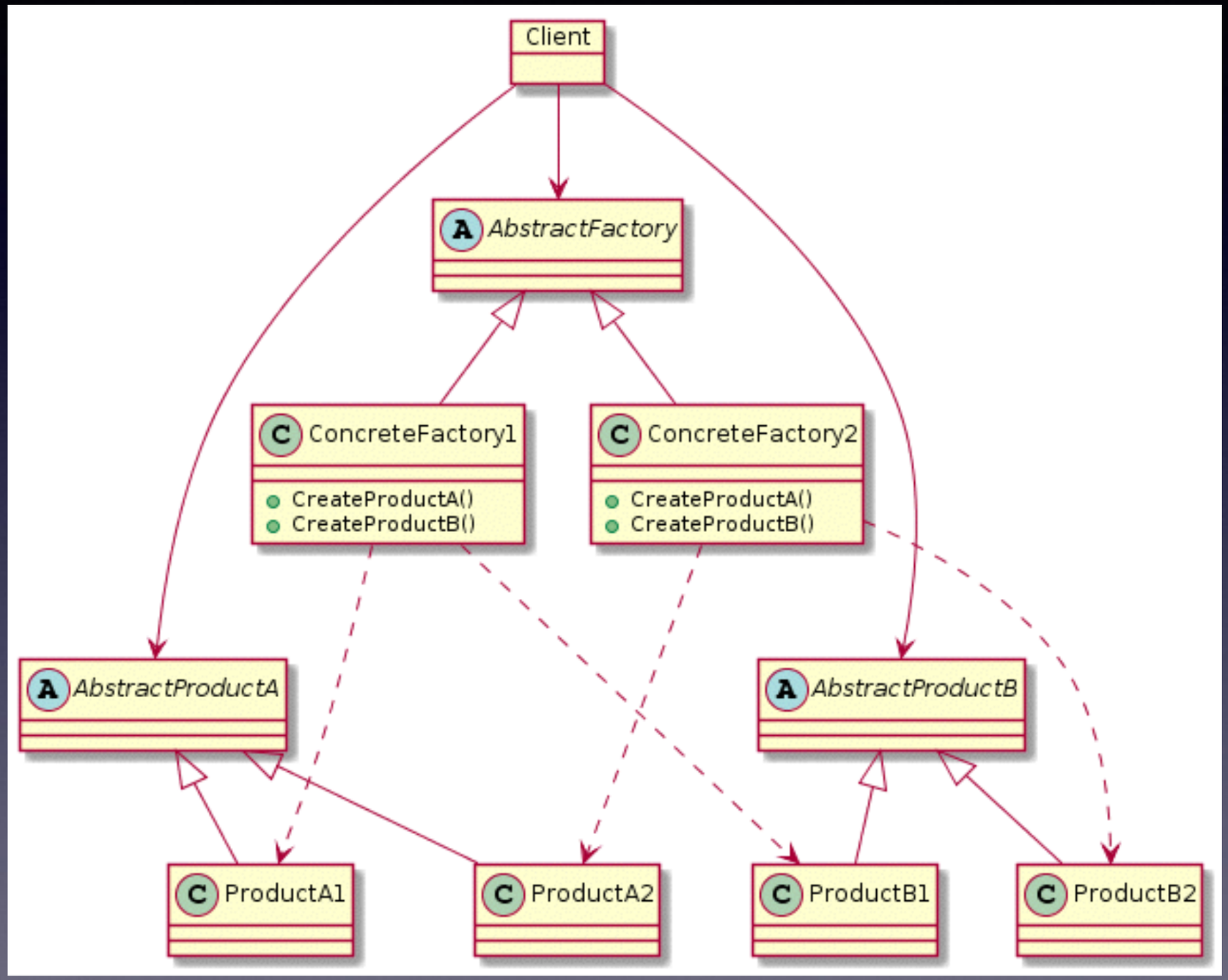
Abstract Factory

- 제품군별 객체 생성 문제
- 목적

- 상세화 된 서브 클래스를 정의하지 않고도 서로 관련성이 있거나 독립적인 여러 객체의 군을 생성하기 위한 인터페이스를 제공

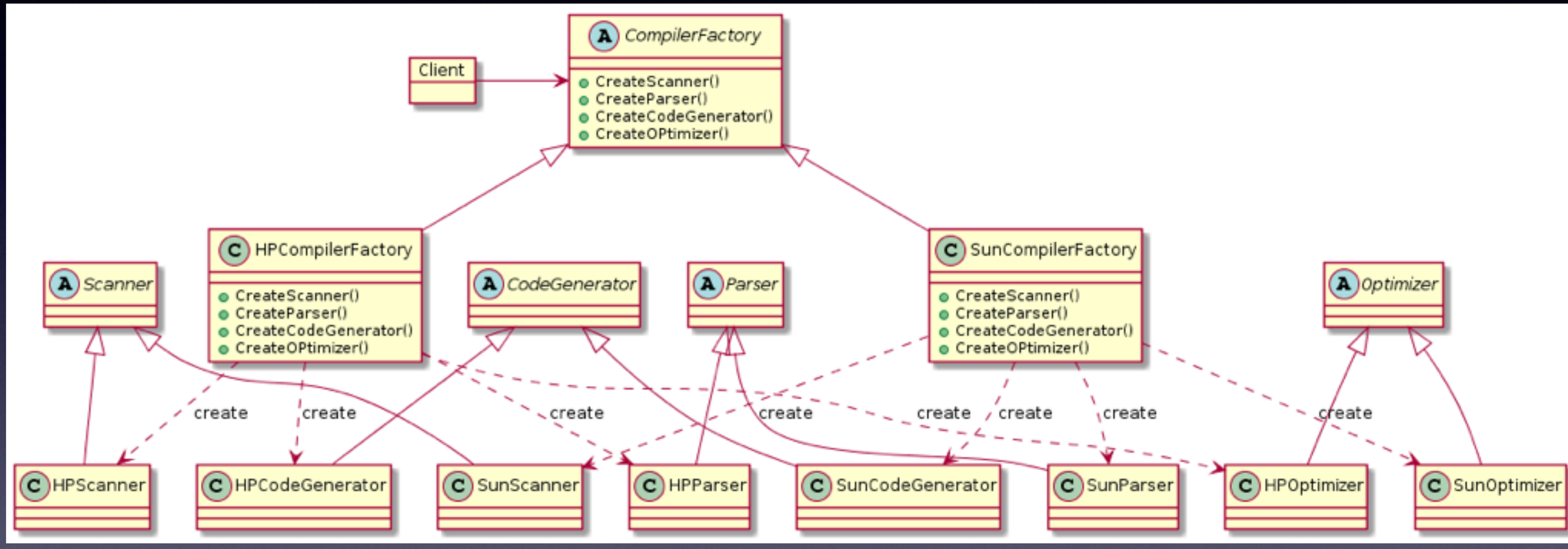
- 인터페이스를 이용하여 서로 연관된 또는 의존하는 객체를 Concrete Class 를 지정하지 않고도 생성할 수 있음

Basic Architecture

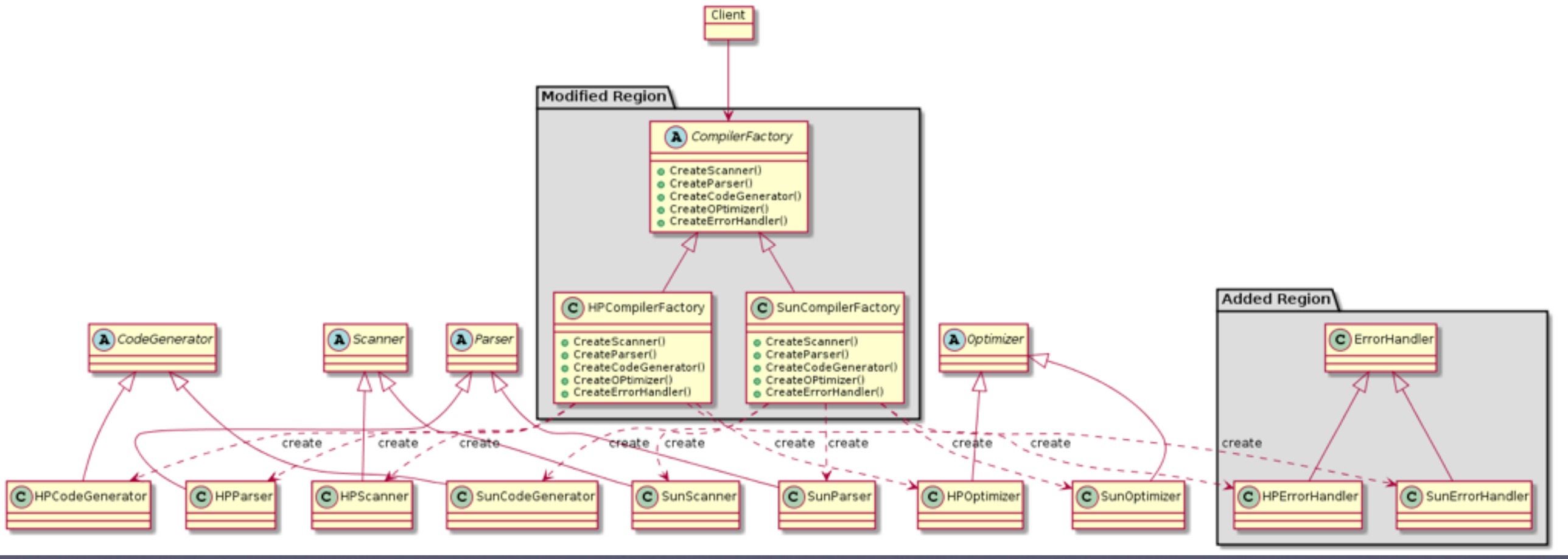


- 제품군을 구성하는 클래스의 객체를 전담 생성하는 클래스를 두되, 새로운 제품군 생성을 추가하는 것이 쉽도록 클래스 상속을 도입하고, 구체적인 제품군별 Factory 클래스 상위에 Abstract Base Class 를 정의한 형태의 설계구조를 말함

Simple Example



Simple Example (cont.)



Characteristic & Coverage

- **Abstract Factory Pattern을 통해서 다양화 가능한 부분**

  제품 객체군

- **생성과 관련된 패턴, 범위는 객체**

  (객체를 생성하는 책임의 일부를 다른 객체에 위임)

- **잘 알려진 사용예**

  다양한 윈도우 시스템을 지원하고자 할 때 사용되는 패턴

Usability

- 객체가 생성되거나 표현되는 방식과 무관하게 시스템을 독립적으로 만들고자 할 때
- 여러제품군중 사용할 제품군을 쉽게 선택할수 있도록 만들고 싶을 때
- 관련된 제품 객체들이 함께 사용 되도록 설계 되었고, 이 부분에 대한 제약이 외부에도 지켜지도록 하고 싶을 때
- 제품에 대한 클래스 라이브러리를 제공하고 그들의 구현이 아닌 인터페이스를 노출 시키고 싶을 때

Cons. and Pros.

- 장점
  - 객체가 생성되는 방식이나 과정 및 책임을 클라이언트가 모르도록 구성하여 객체 생성 구성방법등이 내부적으로 변경 되어도 클라이언트의 변경사항은 최소화됨
  - 제품군간 교체가 쉬움 : 클라이언트의 Concrete Factorty Class의 객체 생성 부분만 변경하면 됨
  - 여러 제품군 들이 실수로 섞여서 사용되는것을 구조적으로 방지
- 단점
  - 제품군 개수가 늘어날수록 ConcreteFactory class 가 늘어나야 됨
  - 새로운 product 추가시 모든 factory class 에 추가되어야 하는 문제.

Factory 객체를 하나만 생성, 유지

- **Abstract Factory + Singletone**

  singleton_abstractfactory.cpp

- **Issues**

  - 일반적인 singletone과는 다르게 하위클래스에서 객체 생성 (일반적인 싱글톤은 abstract class 가 하위 클래스의 CreateInstance를 호출하는 구조)

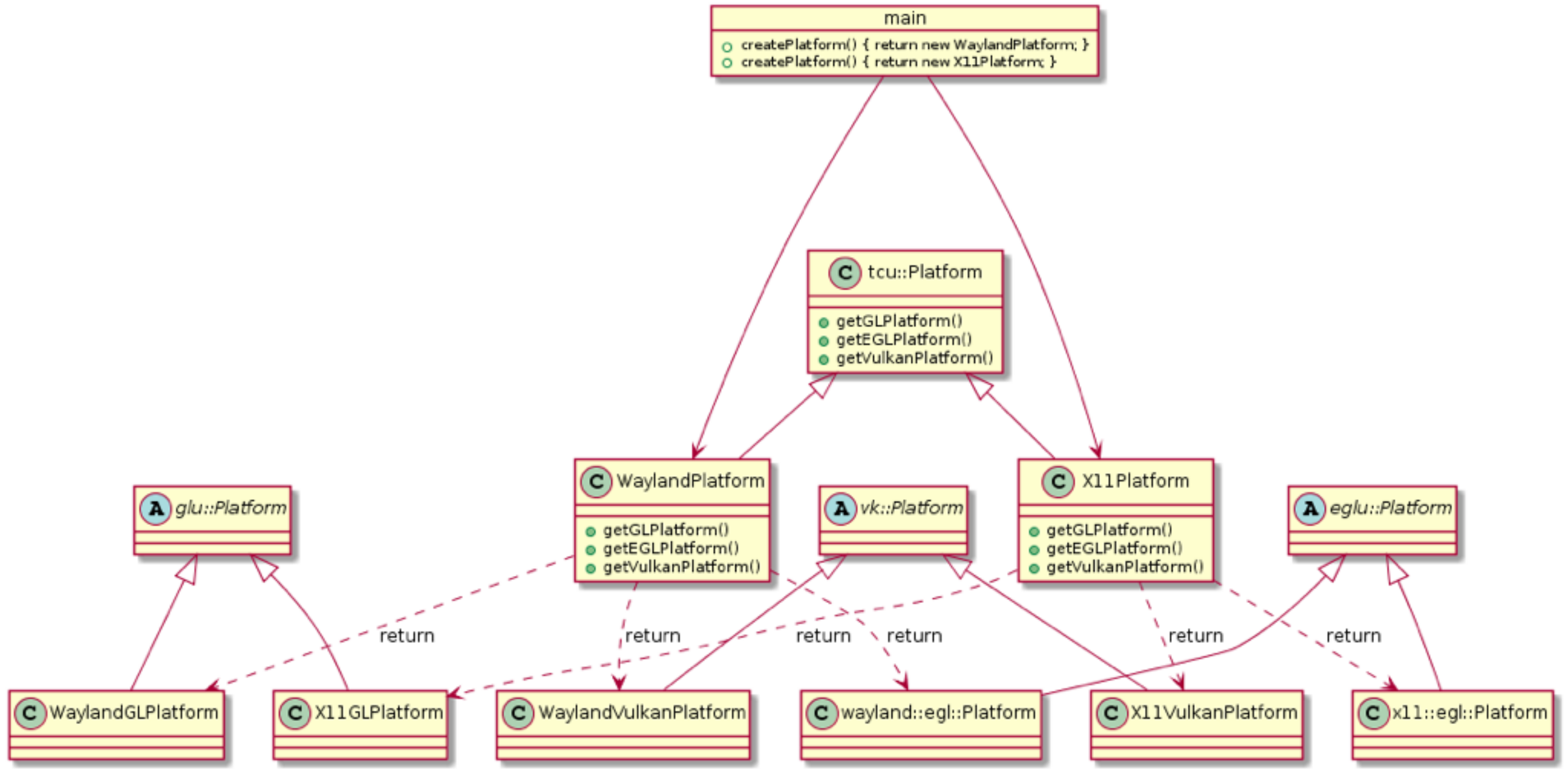
  - 새로운 하위클래스 추가 시 싱글톤 구현 내용도 같이 하위 클래스에 구현해줘야함

Abstract Factory 구현시 prototype 패턴 이용

- **prototype_abstractfactory.cpp**
- **장점**
  - 제품군 추가 시 해당되는 팩토리 클래스 구현이 필요 없음 (product 클래스는 구현해줘야 함)
- **단점**
  - 팩토리 객체에 product 객체 추가 작업이 필요, 객체의 수가 늘어날경우 객체가 같은 팩토리에 포함되는것을 보장하는것에 대한 관리의 어려움이 있음

Use Case

- Vulkan-CTS - Platform Creation

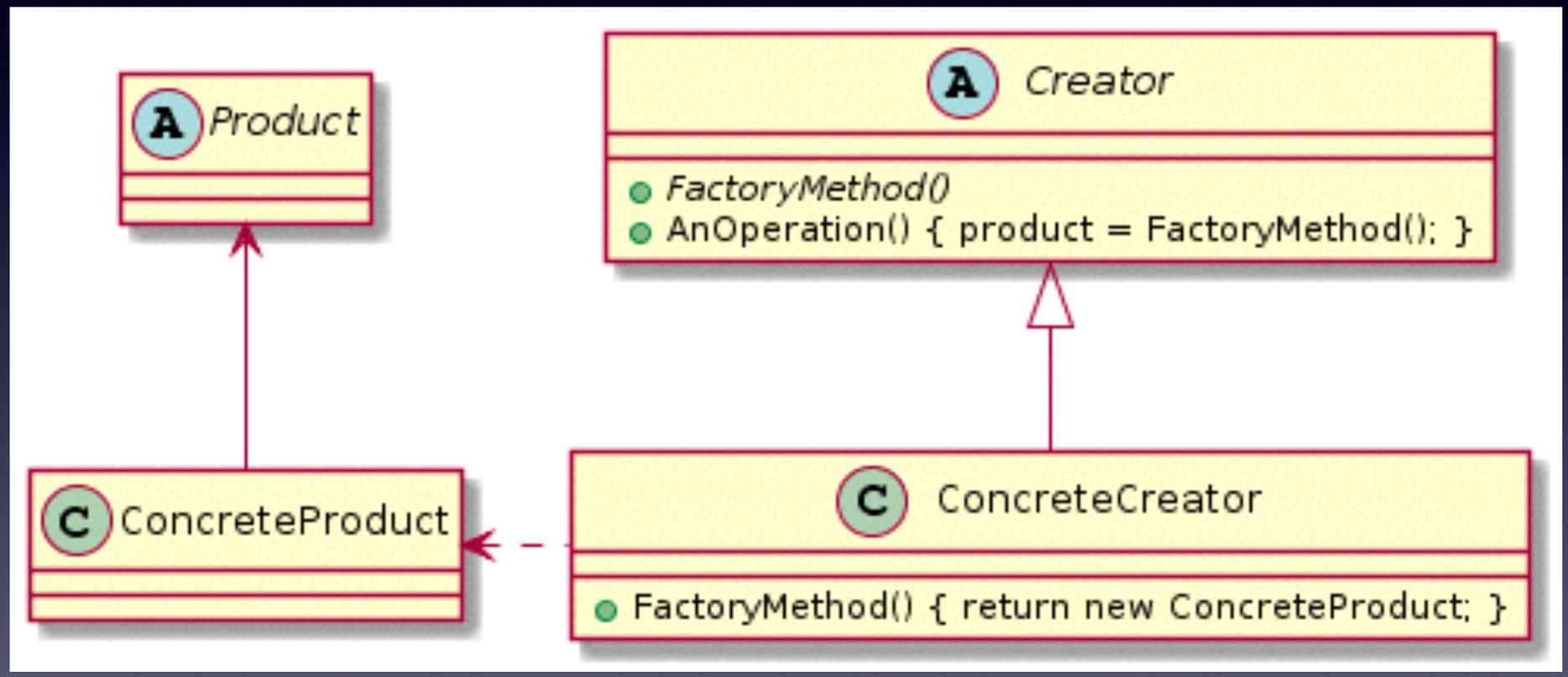


Factory Method

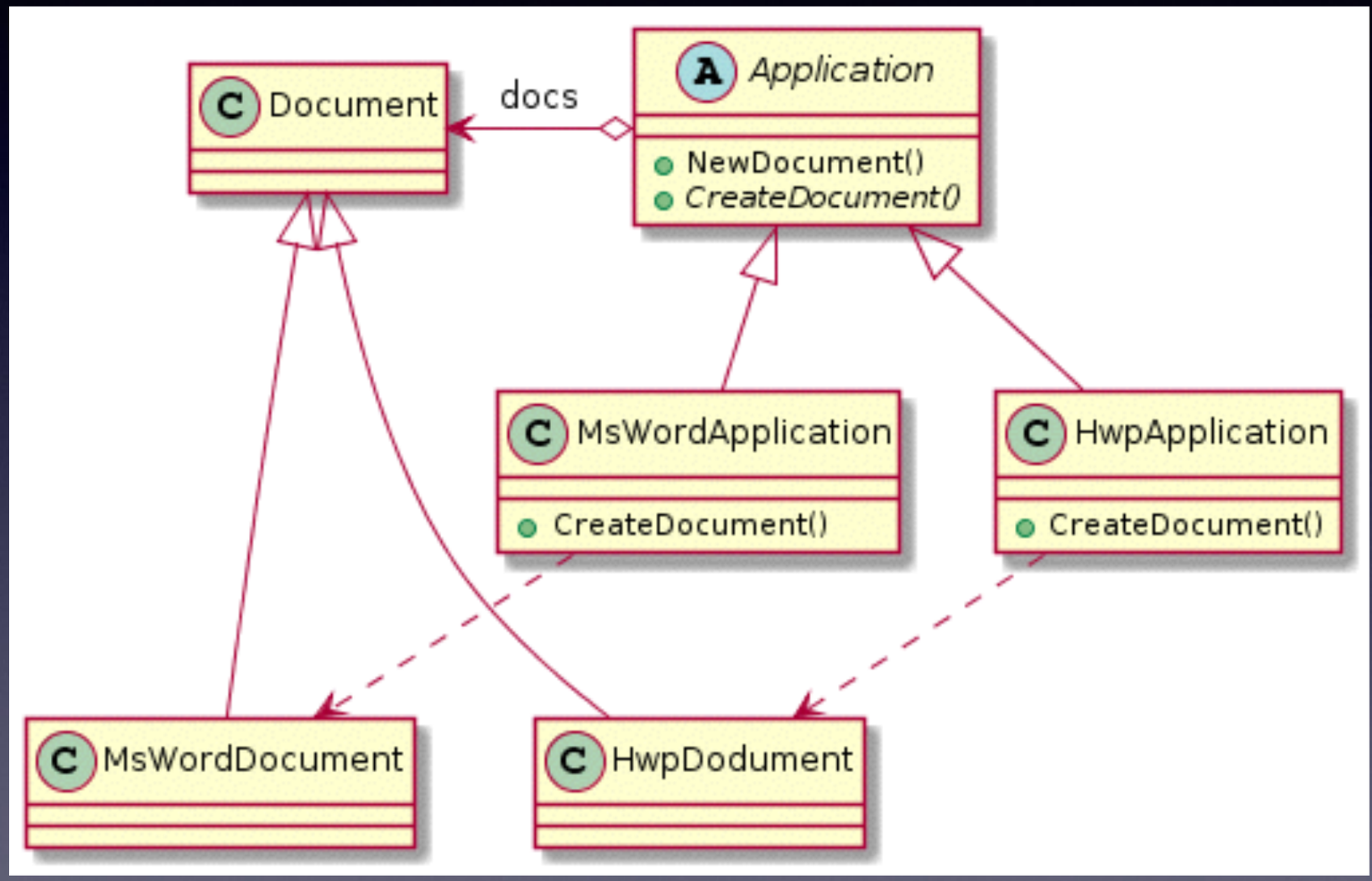
Factory Method

- **대행함수를 통한 객체 생성 문제**
  - 객체를 생성하되 직접 객체 생성자 (constructor) 를 호출해서 객체를 생성하는 것이 아니라 대행 함수를 통해 간접적으로 객체를 생성하는 방식
- **목적**
  - 객체를 생성하기 위해 인터페이스를 정의하지만, 어떤 클래스의 인스턴스 를 생성할 지에 대한 결정은 서브클래스가 내리도록함

Basic Architecture



Simple Example



Factory Method

- 구현
  - 통상적으로 클래스의 상속 구조와 같이 사용, 상위 클래스는 생성하는 객체의 종류와 상관없이 공통적으로 사용할 수 있는 처리 모듈을 포함
  - 일반적인 구현시 하위클래스에서 구현하는 함수는 pure virtual function 을 상속받아 구현, Factory Method 는 Base Class 에서만 구현

Characteristic & Coverage

- **Factory Method Pattern을 통해서 다양화 가능한 부분**
  - 인스턴스화 될 객체의 서브클래스
- **생성과 관련된 패턴, 범위는 클래스**
  - (객체를 생성하는 책임의 일부를 서브클래스가 담당)

Usability

- 구체적으로 어떤 클래스의 객체를 생성해야 할지 미리 알지 못할 경우

- 하위 클래스가 객체를 생성 하기를 원할때 (객체의 생성을 하위클래스에 위임)

- 하위 클래스들에게 개별 객체의 생성 책임을 분산시켜 객체의 종류별로 객체 생성과 관련된 부분을 국지화

Cons. and Pros.

- 장점
  - 어떤 객체를 생성할 것인지 와는 무관하게 동일한 형태로 프로그래밍 가능
  - 객체 생성 구조가 유연하고 확장이 용이
    - 새로운 객체 또는 기존의 코드 확장시 기존 코드 수정이 아닌 새로운 하위 클래스 추가로 구현가능
  - 상속관계에 있는 클래스들의 멤버 함수가 동일한 프로그램 로직으로 내부적으로 생성할 객체만 다룰 경우 편리, 객체 생성 로직만 하위 클래스에서 구현해주면 되기 때문
- 단점
  - 생성할 객체의 종류가 달라질 때 마다 새로운 하위 클래스를 정의 해줘야함 (클래스의 개수가 객체의 종류에 따라서 늘어나는 문제점)

Caution

- 상위 클래스의 생성자에서 Factory Method에 해당하는 멤버함수를 호출하면 안됨
  - 하위 클래스의 객체가 생성되는 중에 상위 클래스의 생성자가 호출되는데 이때는 아직 객체가 완전히 생성된 상태가 아니므로 상위 클래스의 Factory Method에서 하위 클래스의 재정의 한 구현 함수가 불려지지 않기 때문

인자를 통해 생성할 객체의 종류를 정하는 Factory Method 예제

- nonpurevirtual.cpp
- 상속 받는 클래스의 증가 시 복잡성을 늘리지 않는 목적
- 미리 고려되지 않은 객체를 생성 하려면 기존의 소스코드를 수정 또는 별도의 상속받는 클래스를 정의해야함

C++ template를 이용한 하위 클래스 생성

- templatefactory.cpp

- 하위클래스를 위한 클래스 추가를 별도로 하지 않아되는 장점

Use Case

- Vulkan-Demo - Smoke