11장. DIP: 의존성 역전 원칙

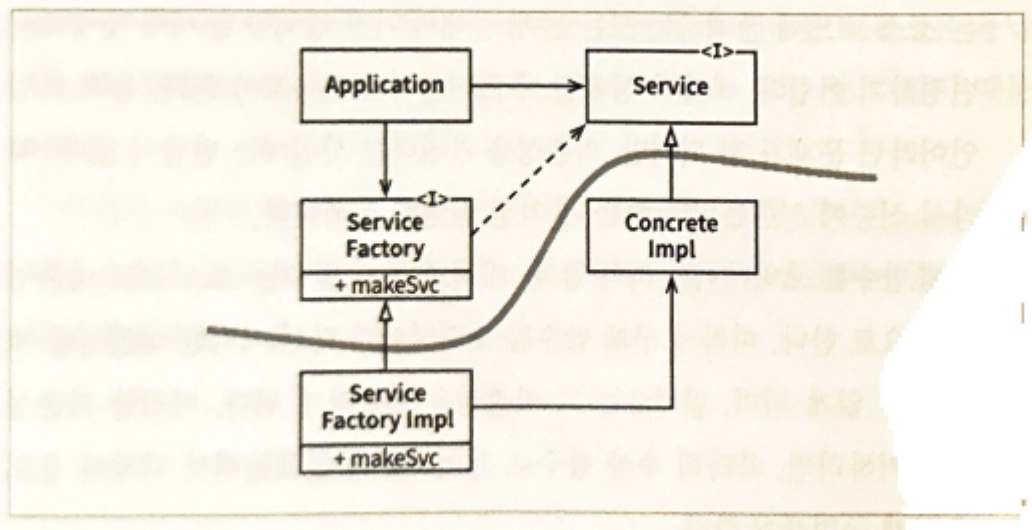
- 유연성이 극대화 된 시스템이란 소스코드 의존성이 추상에 의존하며 구체에는 의존하지 않는 시스템이다.
- 자바와 같은 정적 타입 언어에서 use, import, include 구문은 오직 인터페이스나 추상 클래스같은 선언만을 참조해야한다는 뜻이다. 동적타입 언어도 같다.
- DIP를 논할 때 운영체제나 플랫폼 같이 안정성이 보장된 환경에 대해서는 무시하는 편이다.

안정된 추상화

- 추상 인터페이스에 변경이 생기면 이를 구체화한 구현체들도 따라서 수정해야한다. 반대로 구현체들이 변해도 대부분 인터페이스는 변경될 필요는 없다.
- 안정화 된 소프트웨어 아키텍처란 변동성이 큰 구현체에 의존하는 일은 지양하고, 안정된 추상 인터페이스를 선호하는 아키텍처라는 뜻이다.
- 구체적인 코딩 실천법
  - 변동성이 큰 구체 클래스를 참조하지 말라. 대신 추상 인터페이스를 참조하라.
  - 변동성이 큰 구체 클래스로부터 파생하지 말라.
  - 구체 함수를 오버라이드 하지 말라.
  - 구체적이며 변동성이 크다면 절대로 그 이름을 언급하지 말라.

팩토리

- 위 규칙들을 준수하려면 변동성이 큰 구체적인 객체는 특별히 주의해서 생성해야한다. 사실상 모든 언어에서 객체를 생성하려면 해당 객체를 구체적으로 정의한 코드에 대해 소스 코드 의존성이 발생하기 때문
- 객체지향 언어에서 이처럼 바람직하지 못한 의존성을 처리할 때 추상 팩토리를 사용한다.



구체 컴포넌트

- 위 그림에서 구체적인 의존성이 하나 있고, 따라서 DIP에 위배된다. DIP위배를 모두 없앨 수는 없다. 하지만 DIP를 위배하는 클래스들은 적은 수의 구체 컴포넌트 내부로 모을 수 있고, 이를 통해 시스템의 나머지 부분과는 분리 가능하다.

결론

- DIP는 굉장히 중요하다.
- 의존성은 위 사진의 곡선을 경계로 더 추상적인 엔티티가 있는쪽으로만 향하도록 할 것이다. 이 규칙은 의존성 규칙이라고 부른다.