8. 의존성 관리하기

1) 의존성 이해하기

- 변경과 의존성
  - 객체가 협력을 위해 다른 객체를 필요로 할 때 의존성이 생긴다
    - 실행시점 : 객체가 정상적으로 동작하기 위해 실행 시 의존 대상 객체가 존재해야 한다
    - 구현시점 : 객체가 변경될 경우 의존하는 객체도 변경된다.
  - 의존성은 방향성을 가지며 단방향이다
  - 의존되는 요소가 변경되면 의존하는 요소도 함께 변경될 수 있다
- 의존성 전이
  - 의존성은 변경될 수 있는 가능성
    - 의존성 전이 여부는 변경의 방향과 캡슐화 정도에 따라 달라짐
  - 직접 의존성
    - 다른 요소에 직접 의존, 코드에 명시적으로 드러난다
  - 간접 의존성
    - 의존성 전이에 의해 영향이 전파, 코드에 명시적으로 드러나지 안흔다
- 런타임 의존성과 컴파일 타임 의존성
  - 런타임 - 객체, 컴파일 타임 - 클래스
    - 런타임 의존성과 컴파일 타임 의존성은 다를 수 있다
      - 유연한 설계를 위해 두 의존성을 다르게 만들어야 한다
- 컨텍스트 독립성
  - 구체 클래스에 의존하는 것은 어떤 컨텍스트에서 사용될 것 인지 명시하는 것
    - 구체적인 문맥은 컴파일타임 의존성을 어떤 런타임 의존성으로 대체하느냐에 따라 달라진다
- 의존성 해결하기
  - 컴파일타임 의존성을 실행 컨텍스트에 맞는 적절한 런타임 의존성으로 교체하는 것
    - 객체를 생성하는 시점에 생성자를 통해 의존성 해결
    - 객체 생성 후 setter 메서드를 통해 의존성 해결
    - 메서드 실행 시 인자를 이용해 의존성 해결

2) 유연한 설계

- 의존성과 결합도
  - 의존성의 존재가 아니라 의존성의 정도가 문제다
    - 바람직한 의존성은 재사용성과 관련이 있다
      - 의존성의 정도를 결합도 라고 부른다
  - 컨텍스트에 독립적인 의존성은 바람직하고 특정 컨텍스트에 강하게 결합된 의존성은 바람직 하지 않다
- 지식이 결합을 낳는다
  - 결합도의 정도는 자신이 의존하고 있는 다른 요소에 대해 알고 있는 정보의 양
    - 결합도를 느슨하게 하기 위해서 협력하는 대상에 대해 필요한 정보외 최대한 감춰야 한다
      - 추상화
- 추상화에 의존하라
  - 추상화에 의존할 수록 결합도는 느슨해진다
    - 구체 클래스 의존성
    - 추상 클래스 의존성
    - 인터페이스 의존성
- 명시적인 의존성
  - 결합도를 느슨하게 만들기 위해서 클래스 안의 구체 클래스에 대한 모든 의존성을 제거해야 한다
    - 인스턴스 변수의 타입은 추상 클래스나 인터페이스로 정의
      - 생성자, setter, 메서드 인자는 구체 클래스를 전달
  - 명시적인 의존성
    - 의존성이 퍼블릭 인터페이스에 표현되는 경우
  - 숨겨진 의존성
    - 의존성이 퍼블릭 인터페이스에 표현되지 않는 경우
      - 클래스를 다른 컨텍스트에서 재사용하기 위해 내부 구현을 직접 변경해야 하는 문제가 있다
- new 는 해롭다
  - 클래스 사이의 결합도가 극단적으로 높아진다
    - 구체 클래스의 이름을 직접 기술해야 해서 결합도가 높아진다
    - 생성자에 어떤 인자를 이용하는지 알아야 해서 결합도가 높아진다
  - 사용과 생성의 책임 분리
    - 외부에서 생성된 인스턴스를 전달 받아야 한다
- 컨텍스트 확장하기
  - 의존하는 추상화 클래스에 자식 클래스를 추가해 컨텍스트를 확장한다
- 조합 가능한 행동
  - 유연하고 재사용 가능한 설계는 '어떻게' 하는지 나열하지 않고 객체들의 조합을 통해 '무엇'을 하는지 표현한다
    - 작은 객체들을 조합해 새로운 행동을 이끌어 낼 수 있는 설계
      - 의존성을 관리해야 가능하다