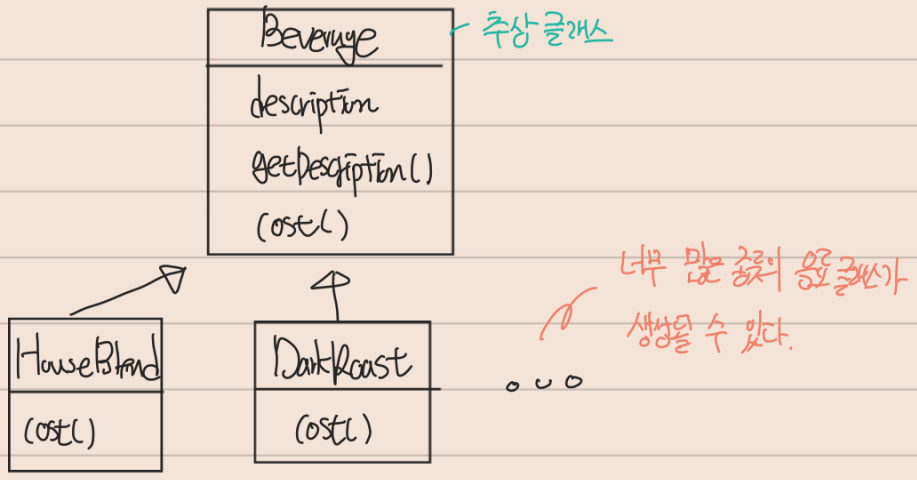
문제점



① 상속대신 상속사용

② 바뀌는 부분을 캡슐화하지 않음

=) 참깨물로 음료를 "장식"할 것! (wrapper로생각!)

DarkRoast 객체 가져와서 → Mocha 객체로 장식 → Whip 객체로 장식 →...

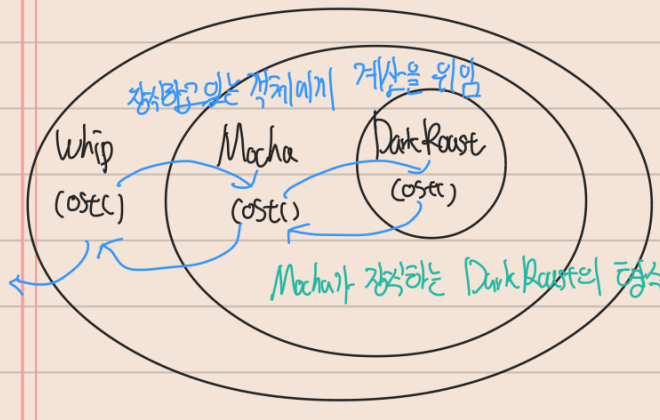
- 서브클래스를 만드는 방식으로 행동을 상속받으면 그 행동은 컴파일 타임에 완전히 결정됨. + 모든 서브 클래스에서 똑같이 상속
  - 구성을 이용하면 실행중에 동적으로 행동 설정 가능
- 구성을 이용하면 기존 코드를 고치는 대신 새로운 코드를 만들어서 새로운 기능 추가 가능.

OCP

디자인 원칙

클래스는 확장에 대해서는 열려 있어야 하지만 코드 변경에 대해서는 닫혀 있어야 한다.

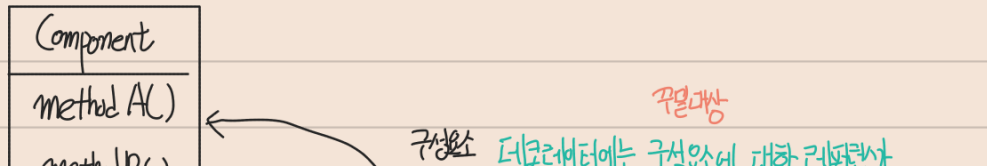
ex) 옵저버 패턴에서 Subject 코드를 수정하지 않아도 언제든 observer을 추가해 확장할 수 있었다.

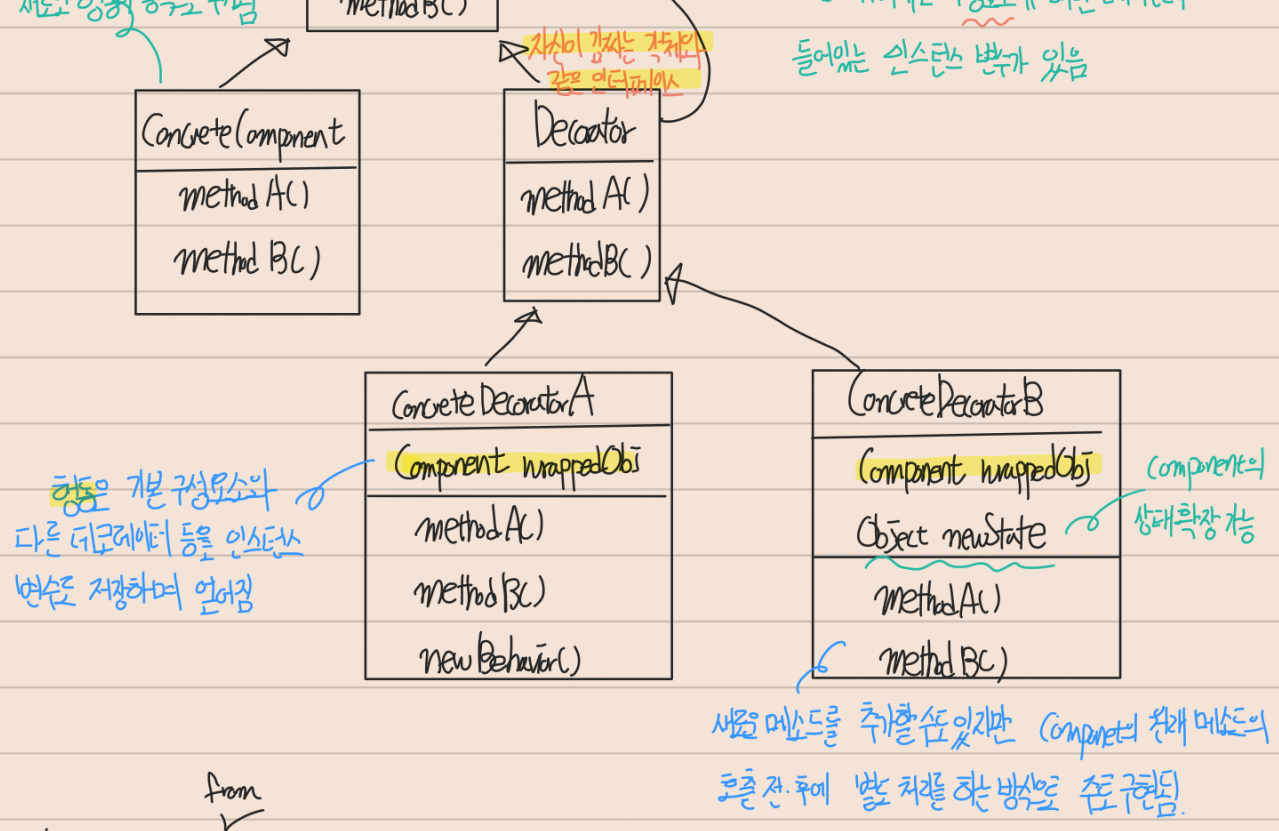


- 데코레이터의 슈퍼 클래스는 자신이 장식하고 있는 객체의 슈퍼클래스와 같음 ← 서로자리바꿔도 상관X
- 한 객체를 여러 개의 데코레이터로 감쌀 수 있음
- 데코레이터는 행동 위임 외에 추가작업 수행도 가능
- 실행중에도 마음대로 원하는 데코레이터 적용가능

데코레이터 패턴

객체에 추가적인 요건을 동적으로 첨가한다. 데코레이터는 서브클래스를 만드는 것을 통해서 기능을 유연하게 확장할 수 있는 방법을 제공한다.





- 새로운 행동을 얻는 게 from 슈퍼클래스의 확장 객체들을 구성하는 방법
- 상속을 쓰면 행동이 컴파일시에 정적으로 결정됨. ↔ 구성을 사용하면 실행중에 데코레이터를 마음대로 조합해서 사용할 수 있음.

- 구상 구성요소의 형식을 알아내서 그 결과를 바탕으로 어떤 작업을 처리하는 코드에는 적용불가
- 데코레이터는 일반적으로 팩토리나 빌더를 써서 만들고 사용하게됨
- 데코레이터는 그 데코레이터가 감싸고 있는 객체에 행동을 추가하기 위해서 만들어진 것.
  ↳ 여러단계의 데코레이터를 파고들어가 어떤 작업을 수행하는것은 원래 의도와 다름
- ex) Java.io의 InputStream 계층.

단점

- 데코레이터를 이용하다보면 잡다한 클래스가 많아짐
- 구성요소를 초기화하는데 필요한 코드가 복잡해짐 ← 팩토리/빌더 사용

- 구성요소의 클라이언트 입장에서는 데코레이터의 존재를 알 수 없음