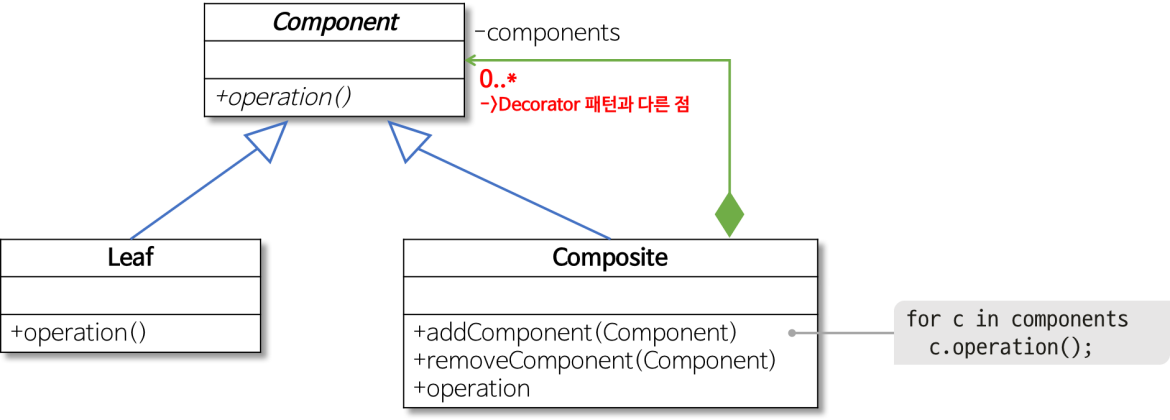
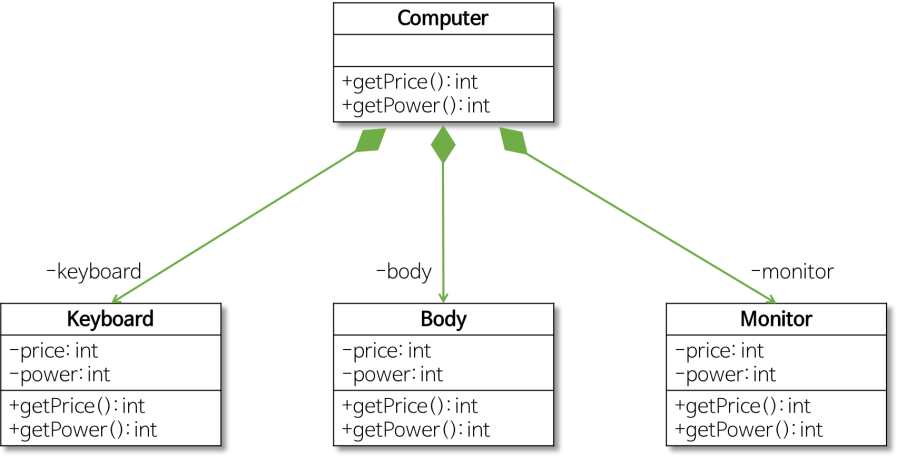
컴퍼지트 패턴

* 여러 개의 객체들로 구성된 복합 객체와 단일 객체를 클라이언트에서 구별 없이 다루게 해주는 패턴
  + 즉 전체- 부분의 관계 (ex. Directory-file)를 갖는 객체들 사이의 관계를 정의할 때 유용하다.
  + 또한 클라이언트는 전체와 부분을 구분하지 않고 동일한 인터페이스를 사용할 수 있다.
  + 구조(Structural)의 하나



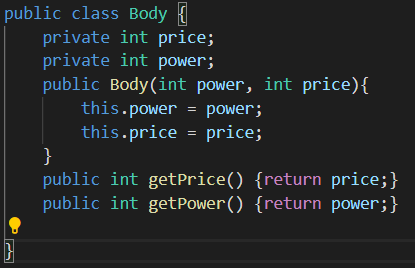
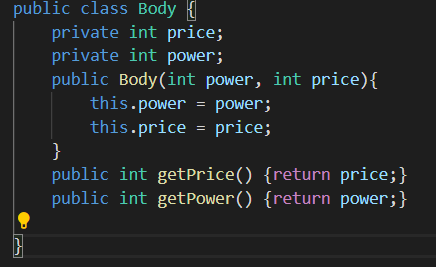
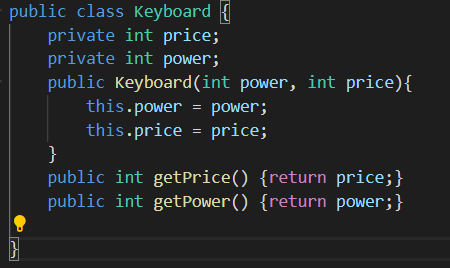
* 역할이 수행하는 작업
  + Component
    - 구체적인 부분
    - 즉 leaf 클래스와 전체에 해당하는 Composite클래스에 공통 인터페이스를 정의
  + Leaf
    - 구체적인 부분 클래스
    - Composite객체의 부품으로 설정
  + Composite
    - 전체 클래스
    - 복수개의 Component를 갖도록 정의
    - 그러므로 복수개의 leaf, 심지어 복수개의 Composite객체를 부분으로 가질 수 있음
* 참고
* 구조 패턴
  + 클래스나 객체를 조합해 더 큰 구조를 만드는 패턴
  + 예를 들어 서로 다른 인터페이스를 지닌 2개의 객체를 묶어 단일 인터페이스를 제공하거나 객체들을 서로 묶어 새로운 기능을 제공하는 패턴
* 예시
* 컴퓨터에 추가 장비 지원하기

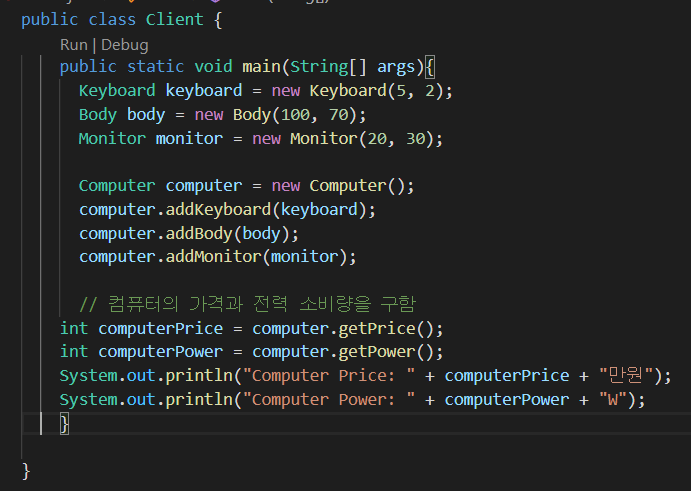
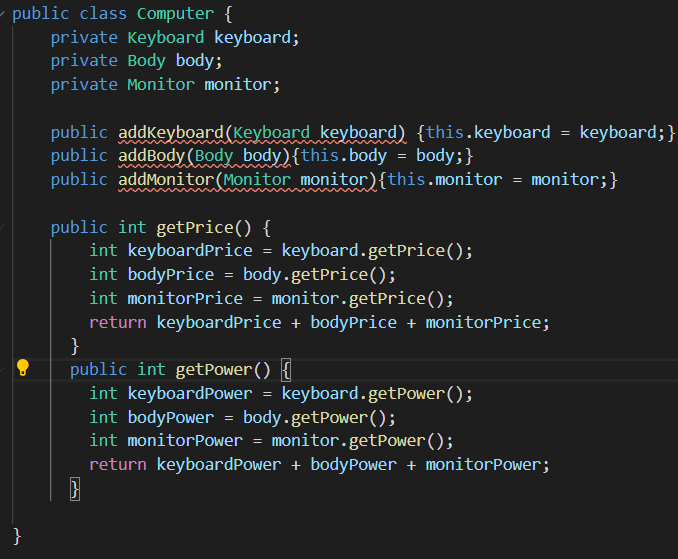


* 컴퓨터(Computer 클래스) 모델링
  + 키보드 본체(keyboard클래스) : 데이터를 입력받는다
  + 본체(Body 클래스) : 데이터를 처리한다.
  + 모니터(Monitor클래스) : 처리 결과를 출력한다.
* Computer클래스 –‘합성관계’-구성장치

참고

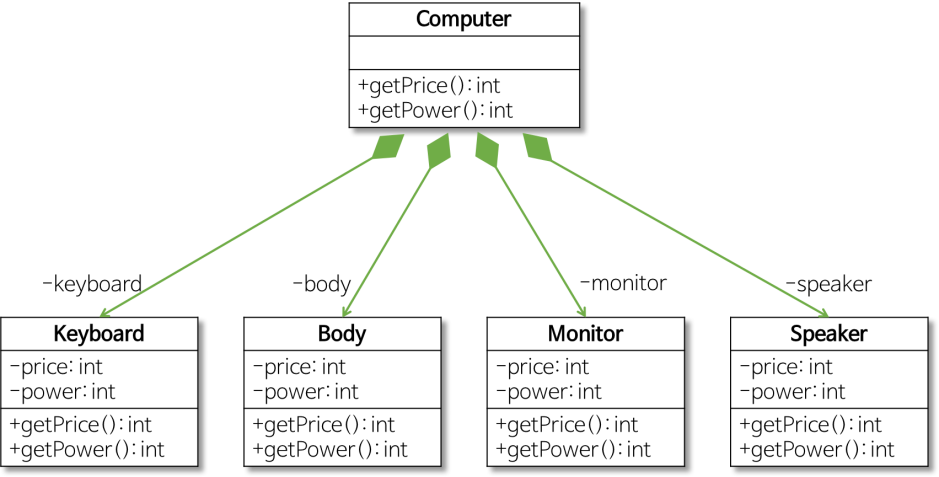
* 합성관계
  + 생성자에서 필드에 대한 객체를 생성하는 경우
  + 전체 객체의 라이프타입과 부분 객체의 라이프타임은 의존적
  + 즉, 전체 객체(마름모가 표시된 클래스) 가 없어지면 부분 객체도 없어진다

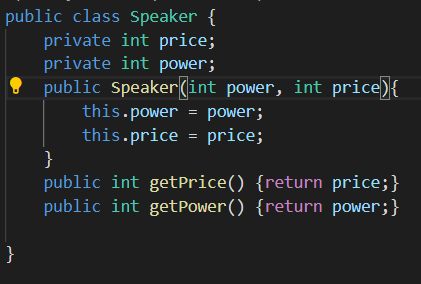


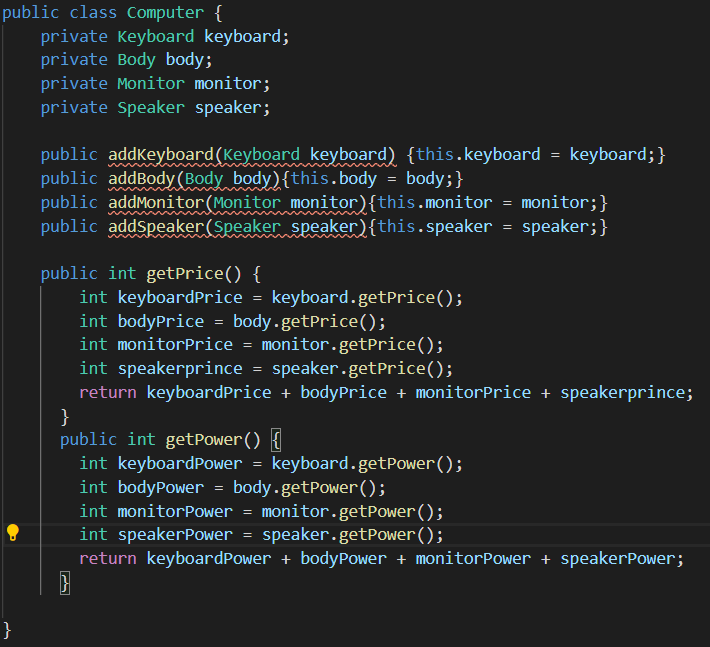


문제점

* 다른 부품이 추가되는 경우
  + Computer 클래스의 부품으로 Speaker 클래스 또는 Mouse 클래스를 추가한다면



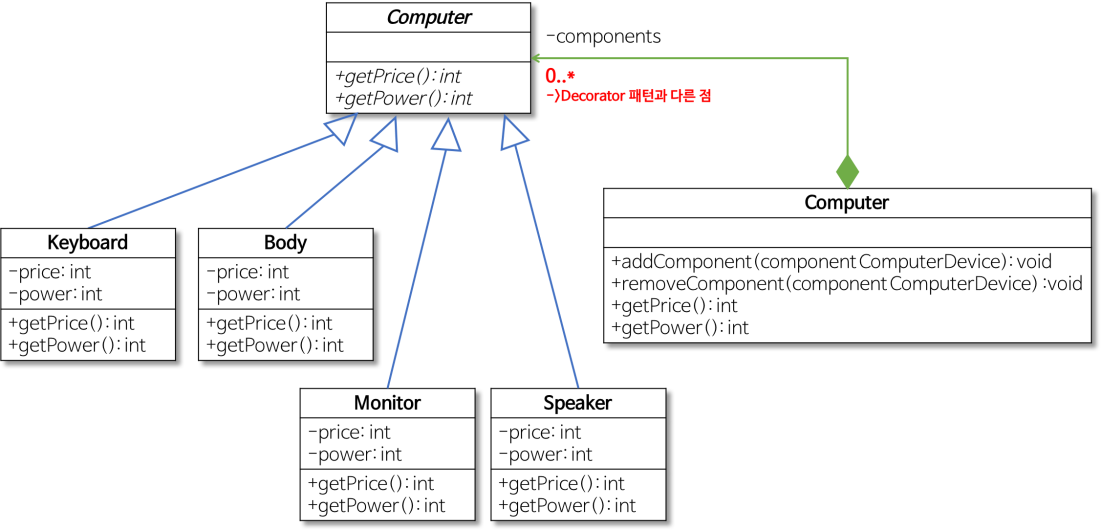




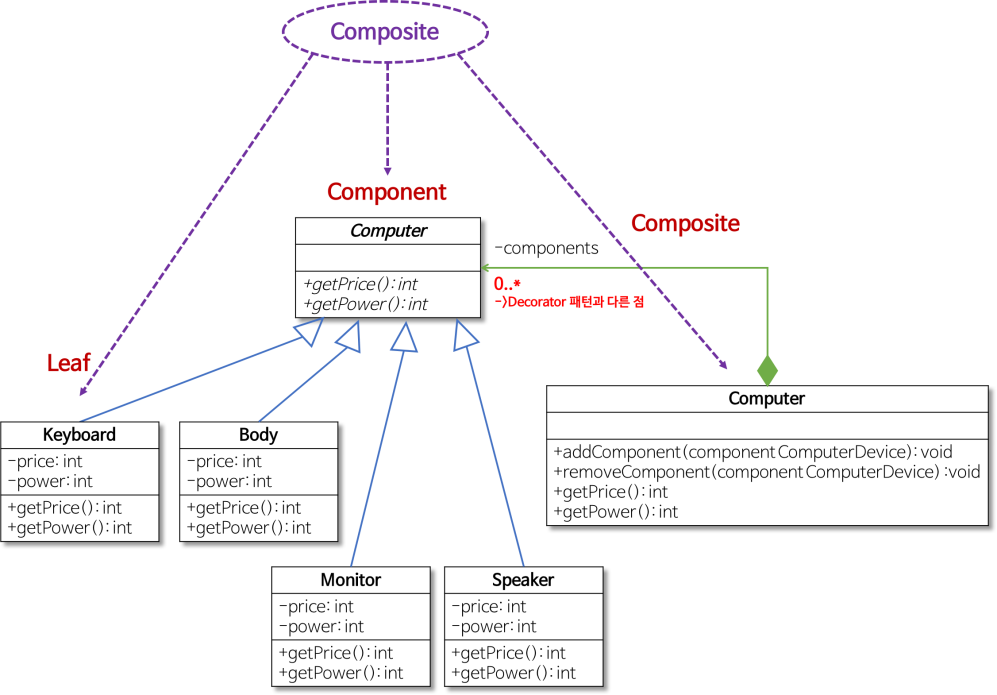
* 위와 같은 방식의 설계는 확장성이 좋지 않다. 즉 OCP를 만족하지 않는다.
* 새로운 부품을 추가할 때 마다 Computer 클래스를 아래와 같이 수정해야 한다
  + 새로운 부품에 대한 참조를 필드로 추가한다
  + 새로운 부품 객체를 설정하는 setter메서드로 addDevice와 같은 메서드를 추가한다.
  + getPrice getPower 등과 같이 컴퓨터의 부품을 이용하는 모든 메서드에서는 새롭게 추가된 부품 객체를 이용할 수 있도록 수정한다.
* 문제점의 핵심은 Computer클래스에 속한 부품의 구체적인 객체를 가리키면 OCP를 위반하게 된다

해결책

* 구체적인 부품들을 일반화한 클래스를 정의하고, 이를 Computer클래스가 가리키도록 설계한다.



* + 구체적인 부품들을 일반화한 ComputerDevice클래스를 정의
    - ComputerDevice 클래스는 구체적인 부품 클래스의 공통 기능만 가지며 실제로 존재하는 구체적인 부품이 될 수 없다.(즉 ComputerDevice 객체를 실제로 생성할 수 없다.)
    - 그러므로 ComputerDevice 클래스는 추상 클래스가 아니다.
  + 구체적인 부품 클래스들 (KeyBoard, Body 등)은 ComputerDevice의 하위 클래스로 정의
  + Computer클래스는 복수 개(0..\*)의 ComputerDevice객체를 갖음
  + Computer클래스도 ComputerDevice클래스의 하위 클래스로 정의
    - 즉 Computer 클래스도 ComputerDevice클래스의 일종
    - ComputerDevice클래스를 이용하면 Client프로그램은 KeyBoard, Body등과 마찬가지로 Computer를 사용할 수 없다.
* Computer클래스
  + ComputerDevice의 하위 클래스면서 복수 개의 ComputerDevice를 갖도록 설계했다
  + addComponent() 메서드를 통해 구체적인 부품인 KeyBoard, body 등을 Computer클래스의 부품으로 설정했다.
* Client
  + addComponent() 메서드를 통해 부품 종류에 관계 없이 동일한 메서드로 부품을 추가할 수 있다.
* 이제 Computer 클래스는 OCP를 준수한다
  + 새로운 부품을 추가한다면 ComputerDevice 클래스의 하위 클래스로 구현하면 된다
* 컴퍼지트 패턴을 이용하면 부분 객체의 추가나 삭제 등이 있어도 전체 객체의 클래스 코드를 변경하지 않아도 된다.
  + 즉, 전체-부분 관계(ex. Directory-File)를 갖는 객체들 사이의 관계를 정의할 때 유용하다.



* Component : ComputerDevice클래스
* Leaf : KeyBoard클래스, Body클래스, Monitor클래스, Speaker클래스
* Composite : Computer클래스