나머지 과목에서 중요하다고 생각되는것만

1. 디자인 패턴
    - 생성 패턴
        - Abstract Factory(추상 팩토리): 구체적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공
        - Builder: 복잡한 인스턴스를 조립해서 단계별로 만드는 구조
        - Factory Method: 상위 클래스에서 객체를 생성하는 인터페이스를 정의하고 하위 클래스에서 인스턴스를 생성하는 패턴
        - Prototype(프로토타입): 기존 객체를 복제해서 새 객체를 생성할 수 있도록 하는 패턴
        - Singleton(싱글턴): 전역변수 사용x, 객체 하나만 생성하도록 하며, 생성된 객체를 어디서든지 참조할수 있음.

    - 구조 패턴
        - Adapter(어댑터): 기존에 생성된 클래스를 재사용할 수 있도록 중간에서 맞춰주는 역할을 하는 인터페이스를 만드는 패턴
        - Bridge(브리지): 추상과 구현을 분리해서 결합도를 낮춘 패턴
        - Composite(복합체): 개별 객체와 복합 객체를 클라이언트에서 동일하게 사용하도록 하는 패턴
        - Decorator: 소스를 변경하지 않고 기능을 확장하는 패턴
        - Facade: 하나의 인터페이스를 통해 느슨한 결합을 제공하는 패턴
        - Flyweight(플라이웨이트): 대량의 작은 객체들을 공유하는 패턴

- Proxy: 대리인이 대신 그 일을 처리하는 패턴

- 행위 패턴
  - Command(커맨드): 요구사항을 객체로 캡슐화해 파라미터로 넘기는 패턴
  - Observer: 상태가 변할때 의존자들에게 알리고 자동으로 업데이트하는 패턴
  - Templete method: 상위 클래스에서는 추상적, 하위 클래스에서는 구체적인 내용을 결정하는 디자인 패턴
  - Mediator(중재자): 객체의 수가 너무 많아져 통신이 복잡해지면중간에서 이를 통제하고 지시 역할
  - Interpreter(통역사): 다양한 언어 구문을 해석할 수 있게 만드는 패턴
  - Iterator(반복자): 객체 간 상호작용을 캡슐화한 패턴
  - State(상태): 객체 상태를 캡슐화해서 클래스화 함 그것을 참조하게 하는 방식
  - Visitor(방문자): 처리구조를 분리하여 별도의 클래스를 만들어 놓고 해당 클래스의 메서드가 각 클래스를 돌아다니며 특정 작업을 수행하도록 만드는 패턴
  - Strategy(전략): 알고리즘 군을 정의하고 같은 알고리즘을 각각 하나의 클래스로 캡슐화한 후, 필요할때 교환해서 사용할 수 있게 하는 패턴
  - Memento: 클래스 설계 관점에서 객체의 정보를 저장할 필요가 있을 때 적용하는 디자인 패턴
  - Chain of Responsibility(책임 연쇄): 정적으로 어떤 기능에 대한 처리의 연결이 하드코딩 되어있어있을때 연결 변경이 불가능, 이를 동적으로 연결한 경우 다르게 처리할 수 있도록 하는 디자인 패턴

책임 연쇄

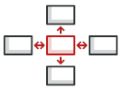
일련의 핸들러들의 사슬을 따라 요청을 전달할 수 있게 해주는 행동 디자인 패턴입니다. 각 핸들러는 요청을 받으면 요청을 처리할지 아니면 체인의 다음 핸들러로 전달할지를 결정합니다.

커맨드

요청을 요청에 대한 모든 정보가 포함된 독립 실행형 객체로 변환합니다. 이 변환은 다양한 요청들이 있는 메서드들을 인수화할 수 있도록 하며, 요청의 실행을 지연 또는 대기열에 넣을 수 있도록 하고, 또 실행 취소할 수 있는 작업을 지원할 수 있도록 합니다.

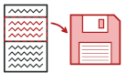
반복자

컬렉션의 요소들의 기본 표현(리스트, 스택, 트리 등)을 노출하지 않고 그들을 하나씩 순회할 수 있도록 합니다.



중재자

객체 간의 혼란스러운 의존 관계들을 줄일 수 있습니다. 이 패턴은 객체 간의 직접 통신을 제한하고 중재자 객체를 통해서만 협력하도록 합니다.

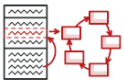


메멘토

객체의 구현 세부 사항을 공개하지 않으면서 해당 객체의 이전 상태를 저장하고 복원할 수 있게 해줍니다.

옵서버

여러 객체에 자신이 관찰 중인 객체에 발생하는 모든 이벤트에 대하여 알리는 구독 메커니즘을 정의할 수 있도록 합니다.



상태

객체의 내부 상태가 변경될 때 해당 객체가 그의 행동을 변경할 수 있도록 합니다. 객체가 행동을 변경할 때 객체가 클래스를 변경한 것처럼 보일 수 있습니다.

전략

알고리즘들의 패밀리를 정의하고, 각 패밀리를 별도의 클래스들에 넣은 후 그들의 객체들을 상호교환할 수 있도록 합니다.

템플릿 메서드

부모 클래스에서 알고리즘의 골격을 정의하지만, 해당 알고리즘의 구조를 변경하지 않고 자식 클래스들이 알고리즘의 특정 단계들을 오버라이드(재정의)할 수 있도록 합니다.

비지터

알고리즘들을 그들이 작동하는 객체들로부터 분리할 수 있습니다.

2. 응집도

강한순서	
기능적 응집도	모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우의 응집도
순차적 응집도	모듈 내 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로 사용할 경우의 응집도
교환적 응집도	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성 요소들이 모였을 경우의 응집도
절차적 응집도	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우의 응집도
시간적 응집도	특정 시간에 츠리되는 몇 개의 기능을 모아 하나의 모듈로 작성할 경우의 응집도
논리적 응집도	유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우의 응집도
우연적 응집도	모듈 내부의 각 구성 요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도

3. 결합도

강한 순서		
내용 결합도 (Content Coupling)	한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정(public 속성)할 때의 결합도	
공유 결합도 (Common Coupling)	공유되는 공통 데이터 영역(전역변수)을 여러 모듈이 사용할 때의 결합도	
외부 결합도 (External Coupling)	어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도	
제어 결합도 (Control Coupling)	어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하기 위해 제어 신로를 통신하거나 제어 요소(Flag)를 전달하는 결합도	
스탬프 결합도 (Stamp Coupling)	모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달될 때의 결합도	
자료 결합도 (Data Coupling)	모듈 간의 인터페이스가 자료 요소로만 구성될 때의 결합도	