역할, 책임, 협력

객체지향의 사실과 오해

협력

- 누군가의 요청으로 협력은 시작된다.
- 동일한 목적을 달성하기 위해 객체들은 협력하고 이런 객체들이 공동체가 객체지향의 세계이다.
- 하나의 객체가 요청을 하면 응답 받은 객체는 필요한 지식과 행동을 가지고 응답을 한다. 따라서 객체들은 이것으로 협력한다.

책임

- 어떤 객체가 어떤 요청에 대해 대답해 줄 수 있거나, 적절한 행동을 할 의무가 있는 경우 해당 객체가 책임을 가진다고 말한다.
- 객체의 책임은 '하는 것' 과 '아는 것'으로 나눠진다.

1. 하는 것 (외부에 제공해 줄 수 있는 정보)

: 객체를 생성하거나 계산을 하는등의 스스로 하는것, 다른 객체의 행동을 시작시키는 것, 다른 객체의 행동을 제어하고 조절하는 것.

2. 아는 것(외부에 제공해 줄 수 있는 서비스)

: 개인적인 정보에 관해 아는 것, 관련된 객체에 관해 아는 것, 자신이 유도하거나 계산할 수 있는 것에 관해 아는 것.

- 책임은 객체의 공용 인터페이스를 구성한다.
- 객체가 다른 객체에게 주어진 책임을 수행하도록 요청을 보내는 것을 메시지 전송이라고 한다.
- 책임과 메시지의 수준은 다르다.

역할

- 역할을 사용하면 여러가지의 협력을 하나의 협력으로 추상화가 가능하다. 추상화를 통해 단순화 할 수 있으며 이것은 애플리케이션의 설계를 이해하고 기억하기 쉬워진다.
- 협력 내에서 다른 객체로 대체할 수 있음을 나타내는 일종의 표식이다. 대체가능성은 행위 호환성을 의미하고, 행위 호환성은 동일한 책임의 수행을 의미하는 것이다.
- 역할을 대체하기 위해서는 각 역할이 수신할 수 있는 메시지를 동일한 방식으

로 이해해야 한다.

객체의 모양을 결정하는 협력

- 협력이라는 실행 문맥안에서 책임을 분배해야 한다.
- 객체지향 세계에서는 충분히 자율적인 동시에 충분히 협력적인 객체를 창조하는 것이 가장 중요한다.

객체지향 설계방법

1. 책임 - 주도 설계
- 협력에 필요한 책임들을 식별하고 적합한 객체에게 책임을 할당하는 방식으로 애플리케이션을 설계하는 것이다.

2. 디자인 패턴
- 전문가들이 반복적으로 사용하는 해결 방법을 정의해 놓은 설계 템플릿 모음이다.

3. 테스트-주도 개발
- 테스트를 먼저 작성하고 테스트를 통과하는 구제적인 코드를 추가하면서 애플리케이션을 완성해 가는 방식이다.
- 테스트가 아니가 설계를 위한 기법이다.

책임과 메시지

객체지향의 사실과 오해

자율적인 책임

- 스스로의 의지와 판단에 따라 각자 맡은 책임을 수행하는 것이다.
- 책임을 다하기 위해 객체는 구체적인 방법이나 절차를 스스로 결정하여 수행한다.
- '어떻게'가 아니라 '무엇을'이 중요하다.

메시지와 메서드

- 메시지-전송 메커니즘은 객체가 다른 객체에 접근할 수 있는 유일한 방법이다.
- 메시지를 처리할 수 있다는 것는 객체가 해당 메시지에 해당하는 행동을 수행해야 할 책임이 있다는 것을 의미한다.
- 외부의 객체는 메시지에 관해서만 볼 수 있고, 객체 내부는 볼 수 없기 때문에 메시지로 객체의 외부와 내부를 분리 할 수 있다.
- 메시지를 처리하기 위해 내부적으로 선택하는 것을 메서드라고 한다,
- 메시지를 수신한 객체가 실행 시간에 메서드를 선택할 수 있다는 사실은 객체지향의 특징 중 하나이다.
- 서로 다른 객체는 동일한 메시지에 서로다르게 반응한다.(다형성)
- 다형성은 수신자의 종류를 캡슐화 한다.
- 수신자의 캡슐화가 설계 품질에 영한을 미치는 이유

1. 협력이 유연해짐
2.협력이 수행되는 방식을 확장할 수 있음
3. 협력이 수행되는 방식을 재사용 할 수 있음

- 다형성을 지탱하는 메시지는 송신자와 수신자를 약하게 연결하기 때문이다.

메시지를 따라라

- 객체지향의 핵심은 메시지이다. 클래스가 아니다.
- 메시지가 수신자의 책임을 결정한다.
- 책임-주도 설계 방법에서는 what/who 사이클에 따라 협력에 참여하 객체를 결정하기 전에 협력에 필요한 메시지를 먼저 결정한다.

- 묻지말고 시켜라스타일, 데메테르 법칙은 애플리케이션이 자율적인 객체들의 공동체라는 사실을 강조한다.
- 메시지 중심 설계는 메세지의 수신자 캡슐화를 증진시킨다.
- 메시지를 믿으면 자율적인 책임은 저절로 따라온다.

객체 인터페이스

- 인터페이스 특징

1. 사용법을 익히면 내부 구조나 동작방식을 몰라도 쉽게 대상을 조작하거나 의사 전달이 가능하다.
2. 단순히 내부 구성이나 작동방식만을 변경하는 것은 인터페이스 사용자에게 어떤 영향도 미치지 않는다.
3. 대상이 변경되더라도 동일한 인터페이스를 제공하기만 하면 아무런 문제 없이 상호작용할 수 있다.

- 객체가 어떤 메시지를 수신 할 수 있는지가 객체가 제공하는 인터페이스의 모양을 만든다.
- 외부에 공개된 인터페이스를 공용인터페이스라고 한다.

인터페이스와 구현의 분리

- 맷 와이스펠드의 세가지 원칙

1. 좀 더 추상적인 인터페이스
2. 최소 인터페이스
3. 인터페이스와 구현 간에 차이가 있다는 점을 인식

- 객체지향 세계에서 내부 구조와 작동방식을 가리키는 고유의 용어가 구현 이다. 구현에는 메서드도 포함됨.
- 객체를 설계할 때 객체 외부에 노출되는 인터페이스(위험지대)와 객체의 내부에 숨겨지는 구현(안전지대)을 명확하게 분리해서 고려해야한다.
- 소프트웨어는 항상 변경되기 때문에 인터베이스과 구현의 분리는 중요하다.
- 캡슐화

1. 상태와 행위의 캡슐화
2. 사적인 비밀의 캡슐화

책임의 자율성이 협력의 품질을 결정한다.

- 자율적인 책임은 협력을 단순하게 만든다.
- 자율적인 책임은 내부와 외부를 명확하게 만든다.
- 책임이 자율적일 경우 책임을 수행하는 내부적인 방법을 변경하더라도 외부에 영향을 미치지 않는다.
- 자율적인 책임은 협력의 대상을 다양하게 선택할 수 있는 유연성을 제공한다.
- 객체가 수행하는 책임들이 자율적일수록 객체의 역할을 이해하기 쉬워진다.

객체 지도

객체지향의 사실과 오해

기능 설계 대 구조 설계

- 기능 설계는 제품이 사용자를 위해 무엇을 할 수 있는지에 초점을 맞춘다.
- 구조 설계는 제춤의 형태가 어떠해야 하는 지에 초점을 맞춘다.
- 미래를 예측할 수 없기 때문에 언제든지 변경하기에 용이한 구조 설계가 효율적이다.

두 가지 재료: 기능과 구조

- 기능을 수집하고 표현하기 위한 기법을 유스케이스 모델링이라고 한다.
- 구조를 수집하고 표현하기 위한 기법을 도메인 모델링이라고 한다.

안정적인 재료: 구조

- 사용자가 프로그램을 사용하는 대상 영역에 관한 지식을 선택적으로 단순화하고 의식적으로 구조화한 형태이다
- 이해관계자들이 바라보는 멘탈 모델이다.
- 객체지향을 이용하면 사용자들이 이해하고 있는 도메인의 구조와 최대한 유사하게 코드를 구조화 할 수 있다.
- 소프트웨어 객체와 현실 객체 사이의 의미적 거리는 존재한다.(표현적차이) 하지만 은유를 통해 차이를 최대한 줄여나가야 한다. 따라서 소프트웨어 객체를 창조하기 위해 우리가 은유해야하는 대상은 바로 도메인 모델이다.
- 도메인 모델이 제공하는 구조는 상당히 안정적이다.

불안정한 재료: 기능

- 사용자의 목표를 달성하기 위해 사용자와 시스템 간에 이뤄지는 상호작용의 흐름을 텍스트로 정리한 것을 유스케이스라고 한다.
- 유스케이스 특성

1. 사용자와 시스템 간의 상호작용을 보여주는 '텍스트'이다.
2. 하나의 시나리오가 아니라 여러 시나리오들의 집합이다.
3. 단순한 피처목록이 아닌 상호작용 흐름 속에서 피처들을 포함하는 이야기를 제공한다.

4. 사용자 인터페이스와 관련된 세부 정보를 포함하지 말아야 한다.
5. 내부설계와 관련된 정보를 포함하지 않는다.
- 유스케이스는 설계기법도 객체지향 기법도 아닌 사용자가 바라보는 외부 관점만 표현하는 것이다.

재료합치기: 기능과 구조의 통합
- 책임-주도 설계방법은 시스템의 기능을 역할 과 책임을 수행하는 객체들의 협력 관계로 바라보게 함으써 유스케이스와 도메인 모델을 통합한다.
- 안정적인 도메인 모델을 기반으로 시스템의 기능을 구현한느 것이 설계를 유지보수 하기 쉽고 유연한 시스템을 만들 수 있다.