객체지향의 사실과 오해

5장. 책임과 메시지

우리는 모두 출근할 ‘책임‘ 이 있다....

만약 출근을 하는데 자가용으로 출근시에만 출근이 인정된다면?

자율적인 책임

객체지향 공동체를 구성하는 기본 단위는 ‘자율적인‘ 객체이다.

객체가 어떠한 행동을 하는 이유는 다른 객체로부터 **‘메시지’**를 수신 받았기 때문이고,
이 메시지를 받아 처리하기 위해 객체가 수행하는 행동을 **‘책임’**이라고 한다.

적절한 책임 -> 자율적인 객체 -> 유연하고 단순한 협력

협력에 참여하는 객체가 얼마나 자율적인지가 전체 애플리케이션의 품질을 결정
(객체지향의 목표는 ‘변경’에 유연하고 ‘이해하기’ 단순한 구조를 만드는 것이기 때문!)

자율적인 책임

1.증언하라

2.목격했던 장면을 떠올리고 떠오르는 기억을 시간 순서대로 재구성한 후 말로 간결하게 표현하라

1번의 경우 증인은 자신의 방법대로 자율적으로 증언을 하면 됨
2번의 경우 판사가 요구하는 ‘어떻게’에 집중하기 때문에 증언 그 자체에 집중하지 못할 수 있음

자율적인 책임의 특징은 **어떻게(how)**가 아니라 **무엇(what)을** 해야 하는가
"어떻게"가 추가될수록 객체가 수행할 수 있는 방법은 제한된다

메시지와 메서드

```
모자장수.증언하라(어제, 왕국);
```

```
public void 증언하라(DateTime date, long mapId)
in class 모자장수
```

메시지 -> 증언하라(어제, 왕국)
메시지이름(증언하라)와 인자(어제, 왕국)의 조합

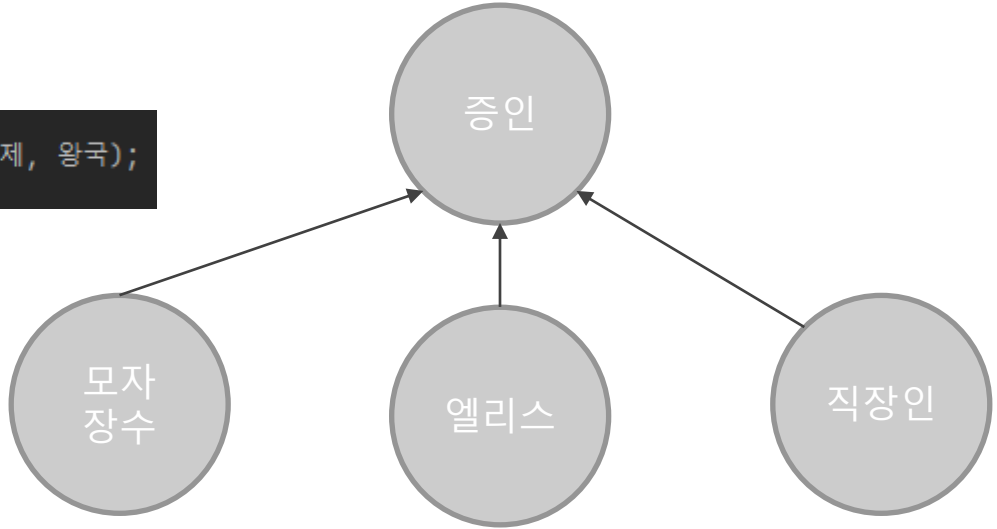
메시지전송 -> 모자장수.증언하라(어제, 왕국)
수신자와 메시지의 조합

메서드 -> 메시지를 처리하기 위해 선택할 수 있는 **방법**
어떤 메서드를 선택할 것인지는 전적으로 수신자의 결정에 좌우됨

메시지 자체를 변경하지만 않는다면 책임을 수행하는 방법을 변경하더라도 판사는 그 사실을 알 수 없음
-> **객체의 외부와 내부가 메시지를 기준으로 분리**

메시지와 메서드

```
증인.증언하라(어제, 왕국);
```



런타임에 메서드를 선택할 수 있다는 사실은 다른 프로그래밍 언어와 **객체지향** 프로그래밍 언어를 구분 짓는 핵심적인 특징

메시지와 메서드

메시지와 메서드의 차이와 관계를 이해하고 나면 객체지향의 핵심 개념인 **다형성**을 쉽게 이해할 수 있음

송신자의 관점에서 다형적인 수신자들을 구별할 필요가 없음

다형성은 동일한 역할을 수행할 수 있는 객체들은 대체가 가능함

-> 객체들의 대체 가능성을 이용해 설계를 유연하고 재사용 가능하게

다형성은 수신자의 종류를 **캡슐화**
-> 송신자와 수신자 간의 대한 결합도를 낮춤
-> 유연하고 확장 가능하고 재사용성이 높다!

메시지와 메서드

유연하고 확장 가능하고 재사용성이 높은 협력의 의미

1. 유연함
송신자는 수신자가 메시지를 이해한다면 누구라도 상관하지 않는다.

2. 확장가능성
송신자에게 아무런 영향을 미치지 않고서도 수신자를 교체할 수 있기 때문에 협력의 세부적인 수행 방식을 쉽게 수정할 수 있다.

3. 재사용
협력에 영향을 미치지 않고서도 다양한 객체들이 수신자의 자리를 대체할 수 있기 때문에 다양한 문맥에서 협력을 재사용할 수 있다.

```
증인.증언하라(어제, 왕국);
```

메시지와 메서드

객체지향 애플리케이션은 클래스를 이용해
만들어지지만 메시지를 통해 정의된다.

책임 주도 설계 다시보기

책임을 완수하기 위해 협력하는 객체들을 이용해 시스템을 설계하는 방법

책임 주도 설계 다시보기

What/Who 사이클

어떤 **행위(What)**를 수행할 것인지를 결정한 후 **누가(Who)**그 행위를 수행할 것인지를 결정
- 이 때, 행위란 바로 ‘메시지‘

수신가능한 메시지가 모여 객체의 **인터페이스**를 구성

What/Who 사이클은 역할을 수행할 객체의 **인터페이스를 발견하기 위해 메시지를 이용**하는 책임-주도 설계의 핵심 아이디어를 명확하게 표현

메시지를 통한 '인터페이스 발견'은 테스트-주도 설계 방법을 이용해 객체를 설계할 때 핵심이 되는 아이디어이기도 하다.

책임 주도 설계 다시보기

묻지 말고 시켜라

메시지를 먼저 결정하고 객체가 메시지를 따르게 하는 설계 방식
-> 묻지 말고 시켜라(Tell, Don't Ask) 스타일, 데메티르 법칙(Law of Demeter)

송신자는 수신자가 어떤 객체인지 모르기 때문에 객체에 관해 꼬치꼬치 캐물을 수 없다

이 스타일은 객체지향 애플리케이션이 자율적인 객체들의 공동체라는 사실을 강조한다

모든 객체는 자신의 상태를 기반으로 스스로 결정을 내려야 한다.

객체 인터페이스

인터페이스
두 사물이 마주치는 경계 지점에서 서로 상호작용할 수 있게 이어주는 방법이나 장치를 의미

1. 내부 구조나 동작 방식을 몰라도 쉽게 대상을 조작가능
- 자동차의 엔진구조를 몰라도 운전하는데 큰 문제가 없음

2. 단순히 내부 구성이나 작동방식만을 변경하는 것은 사용자에게 어떤 영향도 미치지 않음
- 자동차 엔진을 교체한다고 운전 방식이 달라지지 않음

3. 대상이 변경되더라도 동일한 인터페이스를 제공하기만 하면 아무런 문제 없이 상호작용 가능
- 하나의 자동차를 운전하기 위한 인터페이스에 익숙해지면, 다른 자동차도 운전할 수 있음

인터페이스는 객체가 다른 객체와 협력하기 위한 접점

객체 인터페이스

객체지향적인 사고 방식을 이해하기 위한 세가지 원칙

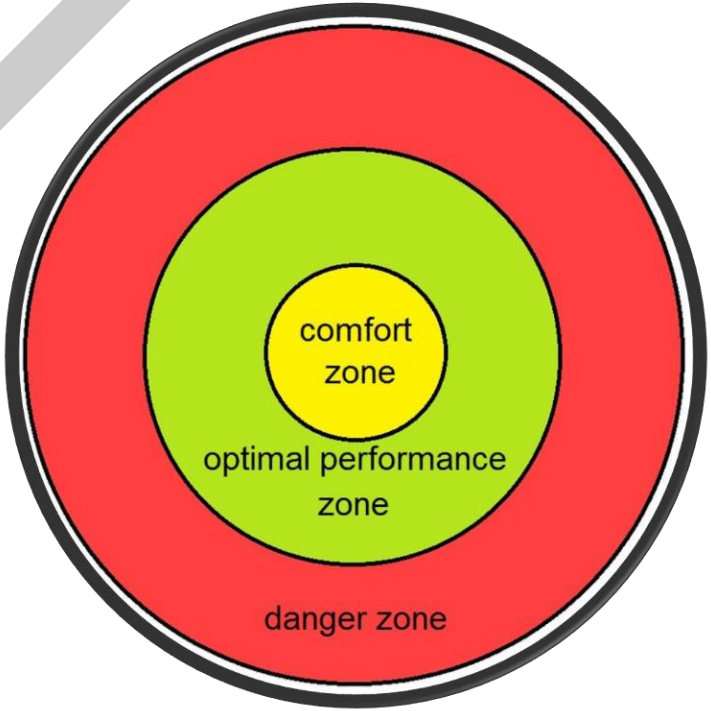
- 좀 더 추상적인 인터페이스
- 최소 인터페이스
- 인터페이스와 구현 간에 차이가 있다는 점을 인식

구현(Implementation)
객체지향의 세계에서 내부 구조와 작동 방식을 가리키는 고유의 용어

객체를 구성하지만 공용 인터페이스에 포함되지 않는 모든 것이 구현에 포함된다.

객체의 외부와 내부를 분리하라는 것은 결국 객체의 공용 인터페이스와 구현을 명확하게 분리하라는 말과 동일하다(인터페이스와 구현의 분리 원칙)

객체 인터페이스



인간의 두뇌의 한계를 인정하고 변경이라는 강력한 적과의 전쟁에서 승리하기 위해 인간이 취할 수 있는 생존 전략
-> 변경해도 무방한 안전 지대와 변경했을 경우 외부에 영향을 미치는 위험 지대를 구분하는 것

안전 지대는 **객체 내부의 구현**
위험 지대는 **객체 외부의 공용 인터페이스**

적절한 구현을 선택하고 이를 인터페이스 뒤로 감추는 것은 객체의 자율성을 향상시킬 수 있는 가장 기본적인 방법이다.

-> 송신자와 수신자가 구체적인 구현 부분이 아니라 느슨한 인터페이스에 대해서만 결합되도록 만드는 것(캡슐화)

자율성

1. 자율적인 책임은 협력을 단순하게 만든다.
-> '증언하다' 라는 책임은 세부 사항을 불필요하게 만든다. (책임의 추상화)

2. 자율적인 책임은 객체의 외부와 내부를 명확하게 분리한다.
-> 협력에 참여하기 위해 외부에 노출하는 부분과 책임을 수행하기 위해 내부적으로 선택하는 방법을 명확하게 나눈다.

3. 책임이 자율적일 경우 책임을 수행하는 내부적인 방법을 변경하더라도 외부에 영향을 미치지 않는다.
-> 책임이 자율적일수록 변경에 의해 수정돼야 하는 범위가 좁아지고 명확해진다.

4. 자율적인 책임은 협력의 대상을 다양하게 선택할 수 있는 유연성을 제공한다.
-> '증언하라' 라는 책임을 수행할 수 있다면 어떠한 객체가 그 책임을 수행하더라도 문제가 없다.

5. 객체가 수행하는 책임들이 자율적일수록 객체의 역할을 이해하기 쉬워진다.
-> 책임이 자율적일수록 특정 역할을 한다는 것을 명확하게 표현하기가 쉬워진다.