1️⃣

협력하는 객체들의 공동체

객체지향이란 실세계를 직접적이고 직관적으로 모델링할 수 있는 패러다임

객체지향 프로그래밍이란 현실 속에 존재하는 사물을 최대한 유사하게 모방해 소프트웨어 내부로 옮겨오는 작업이기 때문에 그 결과물인 객체지향 소프트웨어는 실세계의 투영이며, 객체란 현실 세계에 존재하는 사물에 대한 추상화라는 것

실세계의 모방

객체지향의 기반을 이루는 철학적인 개념을 설명하는 데에 적합함

유연하고 실용적인 관점에서 객체지향 분석, 설계를 설명하기에는 접합하지 않음

애플리케이션을 개발하면서 객체에 직접적으로 대응되는 실세계의 사물을 발견할 확률이 높지 않음

객체지향의 목표

실세계를 모방하는 것이 아님

오히려 새로운 세계를 창조하는 것

소프트웨어 개발자의 역할은 고객과 사용자를 만족시킬 수 있는 신세계를 창조하는 것

실세계의 모방이라는 개념이 비현실적임에도 사람들이 실게계 객체와 소프트웨어 객체 간의 대응이라는 과거의 유산을 반복적으로 재생산하는 이유는?

⇒ 실세계에 대한 비유가 객체지향의 다양한 측면을 이해하고 학습하는 데 매우 효과적이기 때문

> 객체를 스스로 생각하고 스스로 결정하는 현실 세계의 생병체에 비유하는 것
> = 상태와 행위를 캡슐화하는 소프트웨어 객체의 자율성을 설명하는 데 효과적임

현실 세계의 사람들이 암묵적인 약속과 명시적인 계약을 기반으로 협력하며 목표를
= 메세지를 주고받으며 공동의 목표를 달성하기 위해 협력하는 객체들의관계를 설

실세계의 사물을 기반으로 소프트웨어 객체를 식별하고 구현까지 이어간다
= 객체지향 설계의 핵심 사상인 연결완전성을 설명하는 데 적합

실세계의 모방이라는 객체지향의 개념은 훌륭한 프로그램을 설계하고 구현하는 실무적인 관점에서는 부적합

객체지향이라는 용어에 담긴 기본 사상을 이해하고 합습하는 데는 매우 효과적

객체지향에 관한 기본적인 내용을 설명하기 위해 잠시 동안만 실세계의 모방이라는 과거의 인습에 얽매일 예정

- 소프트웨어 객체란 실세계 사물의 모방이라는 전통적인 관점에서 객체지향의 다양한 개념을 설명
- 전통적인 관점이 문제 해결방법으로서의 실용성 측면에서는 의심스럽더라고 객체지향이라는 세계를 이해하는 데는 도움이 될 것

협력하는 사람들

요청과 응답으로 구성된 협력

사람들은 스스로 해결하지 못하는 문제와 마주치면 문제 해결에 필요한 지식을 알고 있거나 서비스를 제공해줄 수 있는 사람에게 도움을 **요청**함

요청을 받은 사람은 주어진 책임을 다하면서 필요한 지식이나 서비스를 제공하는데 이는 다른 사람의 요청에 **응답**하는 것임

요청과 응답을 통해 다름 사람과 **협력**할 수 있는 능력은 인간으로 하여금 거대하고 복잡한 문제를 해결할 수 있는 공동체를 형성할 수 있게 만듦

```
    커피를 주문          커피를 제조하라
       ->                   ->
손님            캐시어                  바리스타
```

```
        <-                  <-
    커피  완성          커피  완성
```

역할과 책임

사람들은 다른 사람과 협력하는 과정 속에서 특정한 **역할**을 부여받음

어떤 협력에 참여하는 특정한 사람이 협력 안에서 차지하는 책임이나 의미

역할이라는 단어는 의미적으로 **책임**이라는 개념을 내포함

우리가 프로그래머라는 역할을 맡을 수 있는 이유는 훌륭한 프로그램을 개발할 책임을 기꺼이 받아들이기 때문

특정한 역할 = 특정한 책임

협력에 참여하며 특정한 역할을 사람들은 역할에 적합한 책임을 수행하게 됨

역할과 책임은 협력이 원활하게 진행되는 데 필요한 핵심적인 구성 요소

사람들이 협력을 위해 특정한 역할을 맡고 역할에 적합한 책임을 수행한다는 사실은 몇 가지 중요한 개념을 제시함

- 여러 사람이 동일한 역할을 수행할 수 있음
- 역할은 대체 가능성을 의미함
- 책임을 수행하는 방법은 자율적으로 선택할 수 있음
- 한 사람이 동시에 여러 역할을 수행할 수 있음

역할, 책임, 협력

기능을 구현하기 위해 협력하는 객체들

지금까지 설명한 실세계의 커피를 주문하는 과정은 객체지향의 핵심적이고 중요한 개념을 거의 포함하고 있음

사람 = **객체**

에이전트의 요청 = **메시지**

에이전트가 요청을 처리하는 방법 = **메서드**

⇒ 많은 사람들이 객체지향을 설명하기 위해 실세계의 모방이라는 은유를 차용하는 이유

커피 주문이라는 협력 관계를 통해 알아본 역할, 책임, 협렵의 개념을 객체지향이라는 문맥으로 옮겨 보자

이번 장을 모두 읽은 후에는 객체지향의 근본 개념이 실세계에서 사람들이 타인과 관계를 맺으며 협력하는 과정과 유사하다는 사실에 공감하게 될 것

역할과 책임을 수행하며 협력하는 객체들

사람들은 커피 주문과 같은 특정한 목표를 이루기 위해 서로 협력함

협력의 핵심은 특정한 책임을 수행하는 역할들 간의 연쇄적인 요청과 응답을 통해 목표를 달성한다는 것

협력에 참여하는 각 개인은 책임을 수행하기 위해 다른 사람에게 도움을 요청하기도 함

이를 통해 연쇄적인 요청과 응답으로 구성되는 협력 관계가 완성됨

객체의 세계는 인간의 세계와 유사함

객체 공통체 안에 살고 있는 객체 시민은 자신에게 주어진 역할과 책임을 다하는 동시에 시스템의 더 큰 목적을 이루기 위해 다른 객체와도 적극적으로 협력함

사용자가 최종적으로 인식하게 되는 시스템의 기능은 객체들이 열심히 협력해서 일궈낸 결실

객체는 애플리케이션의 기능을 구현하기 위해 협력함

애플리케이션의 기능은 더 작은 책임으로 분할되고 책임은 적절한 역할을 수행할 수 있는 객체에 의해 수행됨

객체는 자심의 책임을 수행하는 도중에 다른 객체에게 도움을 청하기도 함

시스템은 역할과 책임을 수행하는 객체로 분할되고 시스템의 기능은 객체 간의 연쇄적인 요청과 응답의 흐름으로 구성된 협력으로 구현됨

객체지향 설계는 적절한 객체에게 적절한 책임을 할당하는 것으로 시작됨

책임은 객체지향 설계의 품질을 결정하는 가장 중요한 요소

책임이 불분명한 객체는 애플리케이션의 미래 역시 불분명하게 만듦

역할은 관련성 높은 책임의 집합

객체의 역할은 사람의 역할과 유사하게 다음과 같은 책임을 지님

- 여러 사람이 동일한 역할을 수행할 수 있음
- 역할은 대체 가능성을 의미함
- 각 객체는 책임을 수행하는 방법은 자율적으로 선택할 수 있음
- 하나의 객체가 동시에 여러 역할을 수행할 수 있음

객체지향 프로그래밍에 경험이 많은 사람들도 역할의 중요성을 간과하곤 함

역할을 유연하고 재사용 가능한 협력 관계를 구축하는 데 중요한 설계 요소

대체 가능한 역할과 책임은 객체지향 패러다임의 중요한 기반을 제공하는 다형성과도 깊이 연관돼 있음

협력 속에 사는 객체

객체는 애플리케이션의 기능을 구현하기 위한 존재

아주 작은 기능도 다른 객체와의 협력을 통해 기능을 구현하게 됨

객체지향 애플리케이션의 아름다움을 결정하는 협력이 얼마나 조화를 이루어지는지를 결정하는 것은 결국 객체임

협력의 품질을 결정하는 것 = 객체의 품질

협력 공동체의 일원으로서 객체가 갖춰야 할 덕목

1. 객체는 충분히 협력적이어야 함
    - 객체는 다른 객체의 요청에 충실히 귀 기울이고 다른 객체에게 적극적으로 도움을 요청할 정도로 열린 마음을 지녀야 함
    - 외부의 도움을 받지 않고 모든 것을 혼자 해결하는 객체는 내부적인 복잡도에 의해 자멸함

- 충분히 협력적이라는 많이 다른 객체의 명령에 따라 행동하는 수동적인 존재를 의미하는 것은 아님
- 어떤 방식으로 응답할지는 객체 스스로 판단하고 결정함

2. 객체가 충분히 자율적이어야 함

객체지향 설계의 묘미는 다른 객체와 조화롭게 협력할 수 있을 만큼 충분히 개방적인 동시에 협력에 참여하는 방법을 스스로 결정할 수 있을 만큼 자율적인 객체들의 공동체를 설계하는 데 있음

상태와 행동을 함께 지닌 자율적인 객체

객체 = **상태**와 **행동**을 함께 지닌 실체

객체가 협력에 참여하기 위해 어떤 행동을 해야 한다면 그 행동을 하는 데 필요한 상태도 함께 지니고 있어야 함

객체가 협력에 참여하는 과정 속에서 스스로 판단하고 스스로 결정하는 자율적인 존재로 남기 위해서는 필요한 행동과 상태를 함께 지니고 있어야 함

객체의 자율성 = 객체의 내부와 외부를 명확하게 구분하는 것으로부터 나옴

객체의 사적인 부분은 객체 스스로 관리하고 외부에서 일체 간섭할 수 없도록 차단해야 함

객체의 외부에서는 접근이 허락된 수단을 통해서만 객체와 의사소통해야 함

객체는 다른 객체가 무엇을 수행하는지는 알 수 없지만 어떻게 수행하는지는 알 수 있음

객체는 행동을 위해 필요한 상태를 포함하는 동시에 특정한 행동을 수행하는 방법을 스스로 결정할 수 있어야 함

객체는 상태와 행위를 하나의 단위로 묶는 자율적인 존재

객체지향에서는 데이터와 프로세스를 객체라는 하나의 틀 안에 함께 묶어 높음으로써 객체의 자율성을 보장함

이것이 전통적인 개발 방법과 객체지향을 구분 짓는 가장 핵심적인 차이

자율적인 객체로 구성된 공동체는 유지보수가 쉽고 재사용이 용이한 시스템을 구축할 수 있는 가능성을 제시함

협력과 메시지

객체지향의 세계에서는 오직 한 가지 의사소통 수단만이 존재함 = 메시지

한객체가 다른 객체에게 요청하는 것 = 메시지를 전송한다

다른 객체로부터 요청을 받는 것 = 메시지를 수신한다

객체는 협력을 위해 다른 객체에게 메시지를 전송하고 다른 객체로부터 메시지를 수신함

객체지향의 세계에서 협력은 메시지를 전송하는 객체와 메시지를 수신하는 객체 사이의 관계로 구성됨

메시지를 전송하는 객체 = 송신자

메시지를 수신하는 객체 = 수신자

메서드와 자율성

객체는 다른 객체와 협력하기 위해 메시지를 전송함

수신자는 먼저 수신된 메시지를 이해할 수 있는지 여부를 판단한 후 미리 정해진 자신만의 방법에 따라 메시지를 처리함

객체가 수신된 메시지를 처리하는 방법 = **메서드**

객체지향 프로그래밍 언어에서 메서드는 클래스 안에 포함된 함수 또는 프로시저를 통해 구현됨

어떤 객체에게 메시지를 전송하면 결과적으로 메시지에 대응되는 특정 메서드가 실행됨

메시지를 수시한 객체가 실행 시간에 메서드를 선택할 수 있다는 점은 다른 프로그래밍 언어와 객체지향 프로그래밍 언어를 구분 짓는 핵심적인 특징 중 하나

프로시저 호출에 대한 실행 코드를 컴파일 시간에 결정하는 절차적인 언어와 확연히 구분되는 특징

메시지와 메서드의 분리는 객체의 협력에 참여하는 객체들 간의 자율성을 증진시킴

외부의 요청이 무엇인지 표현하는 메시지와 요청을 처리하기 위한 구체적인 방법인 메서드를 분리하는 것

= 객체의 자율성을 높이는 핵심 메커니즘

= **캡슐화**라는 개념과도 깊은 관련이 있음

객체지향의 본질

객체지향의 개념

- 객체지향이란 시스템을 상호작용하는 **자율적인 객체들의 공동체**로 바라보고 객체를 이용해 시스템을 분할하는 방법
- 자율적인 객체란 **상태**와 **행위**를 함께 지니며 스스로 자기 자시늘 책임지는 객체를 의미함
- 객체는 시스템의 행위를 구현하기 위해 다른 객체와 **협력**하고, 각 객체는 협력 내에서 정해진 **역할**을 수행하며 역할은 관련된 **책임**의 집합임
- 객체는 다른 객체와 협력하기 위해 메시지를 전송하고, **메시지**를 수신한 객체는 메시지를 처리하는 데 적합한 **메서드**를 자율적으로 선택함

객체를 지향하라

1960년대에 발표된 프로그래밍 언어인 시뮬라67에서 출발

객체지향의 인기를 주도한 것은 스몰토크와 C++, 자바로 대표되는 클래스 기반 프로그래밍 언어의 유행

초기 객체지향 프로그래밍 언어의 초점은 새로운 개념의 데이터 추상화를 제공하는 클래스라는 빌딩 블록에 맞춰져 있었음

다양한 프로그래밍 언어가 출현하고 관련 서적들이 출간되는 과정 속에서 클래스에 대한 중요성이 과하다 싶을 정도로 강조됨

클래스의 중요성은 프로그래밍 언어라는 다리를 건너면서 조금씩 부풀려졌음

그 결과 사람들은 객체지향의 중심에 있어야 할 객체로부터 조금씩 멀어져 갔음

클래스가 객체지향 프로그래밍 언어의 관점에서 매우 중요한 구성요소인 것은 맞음

하지만 객체지향의 핵심을 이루는 중심 개념이라고 말하기에는 무리가 있음

자바스트립트처럼 프로토타입 기반의 객체지향 언어에서는 클래스가 존재하지 않음

지나치게 클래스를 강조하는 프로그래밍 언어적인 관점은 객체의 캡슐화를 저해하고 클래스를 서로 강하게 결합시키고, 애플리케이션을 협력하는 객체들의 공동체가 아닌 클래스로 구성된 설계도로 보는 관점은 유연하고 확장 가능한 애플리케이션의 구축을 방해함

훌륭한 객체지향 설계자가 되기 위해 거쳐야 할 첫번째 도전

= 코드를 담는 클래스의 관점에서 메시지를 주고받는 객체의 관점으로 사고의 중심을 전환하는 것

= 클래스는 객체들의 협력 관계를 코드로 옮기는 도구에 불과함

객체지향의 핵심

적절한 책임을 수행하는 역할 간의 유연하고 견고한 협력 관계를 구축하는 것

클래스는 협력에 참여하는 캑체를 만드는 데 필요한 구현 메커니즘일 뿐

객체지향의 중심에는 객체가 위치하며, 중요한 것은 클래스들의 정적인 관계가 아니라 메시지를 주고받는 객체들의 동적인 관계