7 & 부록. 추상화 기법

7장. 함께 모으기
    💭 직접 설계, 구현해보기
추상화 기법
분류와 인스턴스화
    개념과 범주
    타입
    외연과 집합
    클래스
일반화와 특수화
    범주의 계층
    서브타입
    상속
    💭
집합과 분해
    계층적인 복잡성
    합성 관계
    패키지
💭

7장. 함께 모으기

> **“코드와 모델을 밀접하게 연관시키는 것은 코드에 의미를 부여하고 모델을 적절하게 한다.”**

객체지향 설계 안에는 세 가지 상호 연관된 관점이 존재한다.

- **개념 관점(Conceptual Perspective)**
    - 설계는 도메인 안에 존재하는 개념과 개념 간의 관계를 표현한다.
    - 소프트웨어는 도메인에 존재하는 문제를 해결하기 위해 개발된다.
    - 따라서 **실제 도메인의 규칙과 제약을 최대한 유사하게 반영하는 것이 핵심**이다.
- **명세 관점(Specification Perspective)**
    - 도메인이 아니라 객체들의 책임, 즉 인터페이스에 초점을 맞춘다.

    - 객체가 협력을 위해 **'무엇'**을 할 수 있는가에 초점을 맞춘다.
- **구현 관점(Implementation Perspective)**
    - 객체들이 책임을 **'어떻게'** 수행할 것인가에 초점을 맞추며 인터페이스를 구현하는 데 필요한 속성과 메서드를 클래스에 추가한다.

클래스는 세 가지 관점을 모두 수용할 수 있도록 개념, 인터페이스, 구현을 함께 드러내는 동시에 깔끔하게 분리해야 한다.

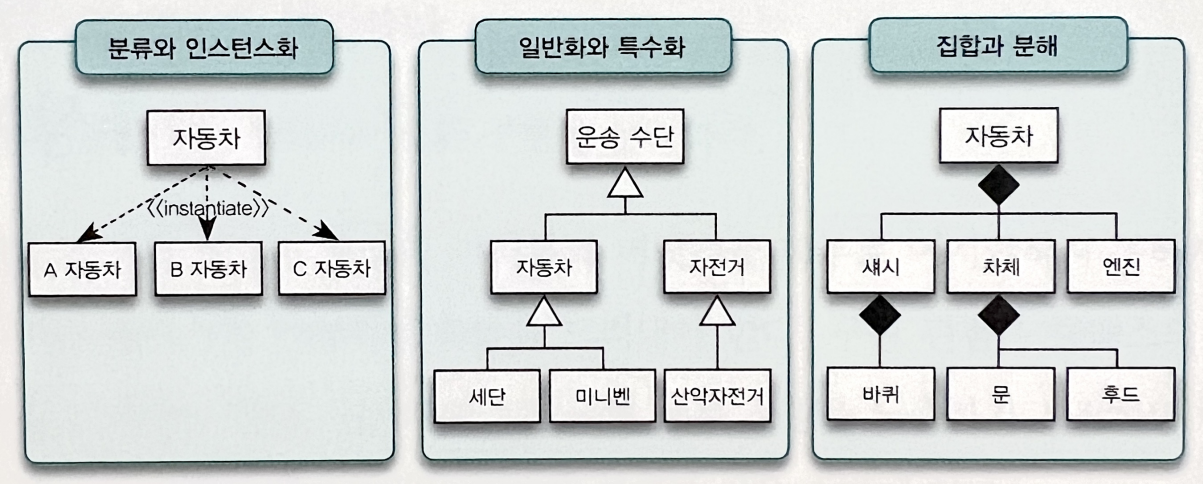
💭 직접 설계, 구현해보기

https://github.com/Java-Entrance/OOP-Study/tree/main/재모/7장

- 왜 Coffee도 price를 들고 있는거지????? 신경쓰지 않아도 되는 부분인가? 모르겠다.
- 처음에는 바리스타에게 손님이 준 돈과 메뉴 항목의 가격을 계산해야 한다는 책임이 있도록 설계했다. 책에서 캐셔는 없다고 가정했기 때문이다.
  그런데.. 현실 세계 커피숍에 캐셔가 없다고 해서 객체지향 세계의 커피숍에도 없다고 하는 것은 내가 또 다시 창조가 아닌 모방을 해버린게 아닐까..?
  책을 읽고도 창조가 아닌 모방을 해서 부끄럽지만, 내가 한 설계가 모방이라는 것임을 깨달았다는 것이 조금은 성장했다는 증거라고 생각하려고 한다.

추상화 기법

사람들이 세계를 이해하는 데 사용하는 추상화 기법의 종류



- **분류와 인스턴스화**

분류는 객체의 구체적인 특성을 숨기고 공통적인 특성을 기반으로 범주화하는 과정이다.
인스턴스화는 특정 범주에서 구체적인 객체를 생성하는 과정이다. 즉 분류의 역이라고 보면 된다.

- **일반화와 특수화**

  일반화는 범주 사이의 차이를 숨기고 공통적인 특성을 강조한다.
  특수화는 일반화의 역이다.

- **집합과 분해**

  집합은 부분과 관련된 세부 사항을 숨기고 부분을 사용해서 전체를 형성하는 과정이다.
  분해는 전체를 부분으로 분리하며, 집합의 반대 과정이다.

분류와 인스턴스화

개념과 범주

- 객체를 분류하고 **범주**로 묶는 것은 객체들의 특정 집합에 공통의 **개념**을 적용하는 것을 의미한다.
- 세상에 존재하는 객체에 개념을 적용하는 과정을 **분류**라고 한다.
- 사람들은 분류를 통해 개별 현상을 하나의 개념으로 다룬다.
  - '수많은 개별적인 현상들' - **객체**
  - '하나의 개념' - **타입**
- 분류는 객체와 타입 간의 관계를 나타낸 것이다. 어떤 객체가 타입의 정의에 부합할 경우 그 객체는 해당 타입으로 분류되며 자동으로 타입의 인스턴스가 된다.

타입

어떤 객체의 타입이 자동차라고 말할 수 있으려면 자동차가 무엇인지에 대한 명확한 정의가 필요하다.
즉, 타입을 객체의 분류로서 적용할 수 있으려면 다음 세 가지 관점에서의 정의가 필요하다.

- **심볼**: 타입을 가리키는 간략한 이름이나 명칭
  - ex) 자동차
- **내연**: 타입의 완전한 정의. 내연의 의미를 이용해 객체가 타입에 속하는지 여부를 확인할 수 있다.
  - ex) 원동기를 동력원으로 해서 주행하는 사람이나 화물을 운반하는 기계

- **외연**: 타입에 속하는 모든 객체들의 집합
    - ex) 버스, 택시, 승용차, …

외연과 집합

집합은 외연을 가리키는 또 다른 명칭이다.

- 집합은 많은 객체를 포함하고, 객체는 하나 이상의 집합에 포함될 수 있다.

> ✏️ **분류의 종류**
>
> - **단일 분류(single classification)**: 한 객체가 한 시점에 하나의 타입에만 속하는 경우
> - **다중 분류(multiple classification)**: 한 객체가 한 시점에 여러 타입에 속하는 경우
> - **동적 분류(dynamic classification)**: 객체가 한 집합에서 다른 집합의 원소로 자신이 속하는 타입을 변경할 수 있는 경우
> - **정적 분류(static classification)**: 객체가 자신의 타입을 변경할 수 없는 경우

- 개념적인 관점에서 다중 분류와 동적 분류를 함께 적용하는 것이 실세계의 복잡성을 모델링하는 데 유용하지만, 대부분의 언어는 정적 분류만 허용하며 동적 분류를 구현할 수 있는 방법을 제공하지 않는다.
- 다중 분류와 동적 분류 관점에서 도메인 모델의 초안을 만든 후 실제 구현에 적합하도록 단인 분류와 정적 분류 방식으로 객체들의 범주를 재조정하는 편이 분석과 구현 간의 차이를 메울 수 있는 가장 현실적인 방법이다.

클래스

- `클래스 ≠ 타입`
    - 클래스는 타입을 구현하는 용도 외에도 코드를 재사용하는 용도로 사용되기도 한다.
    - 클래스 외에도 인스턴스를 생성할 수 없는 추상 클래스나 인터페이스를 이용해 타입을 구현할 수도 있다.
- 대부분의 객체지향 언어는 본질적인 속성은 표현할 수 있지만 우연적인 속성은 표현할 수 없기 때문에 동일한 범주에 속하는 객체는 모두 동일한 속성을 가져야만 한다.

✏️ **본질적인 속성과 우연적인 속성**
본질이란 한 사물의 가장 핵심적이고 필수불가결한 속성이다. 본질적이지 않은 속성을 우연적인 속성이라고 한다.
예를 들어, 어떤 사람이 취업해서 회사원이 됐다고 해도 그 사람은 여전히 그 사람일 뿐이다. 회사원이라는 역할이 그 사람의 본질을 바꾸지는 못한다.

일반화와 특수화

범주의 계층

카를로스 린네가 제시한 계층 구조는 좀 더 세부적인 범주가 계층의 하위에 위치하고, 좀 더 일반적인 범주가 계층의 상위에 위치한다.

이때 계층의 상위에 위치한 범주를 계층의 하위에 위치한 범주의 **일반화**라고 하고, 그 반대를 **특수화**라고 한다.

서브타입

- 좀 더 일반적인 타입을 이용해 더 세부적인 타입을 정의함으로써 타입 간의 계층 구조를 구축할 수 있다.
- **슈퍼타입**: 어떤 타입보다 일반적인 타입
- **서브타입**: 어떤 타입보다 특수한 타입
- 슈퍼타입은 서브타입의 일반화이고, 서브타입은 슈퍼타입의 특수화다.
- 계층 구조에서 서브타입은 슈퍼타입이 가진 본질적인 속성과 함께 자신만의 추가적인 속성을 가진다.
  → 계층에 속하는 모든 서브타입들이 슈퍼타입의 속성을 공유함
  → 부분적인 사실을 통해 복잡한 사실에 대한 논리적인 추론이 가능해짐
- 어떤 타입이 다른 타입의 서브타입이 되기 위해서는 '100%의 규칙'과 'Is-a 규칙'을 준수해야 한다.
  - **100% 규칙**: 슈퍼타입의 정의가 100% 서브타입에 적용돼야만 한다. 서브타입은 속성과 연관관계 면에서 슈퍼타입과 100% 일치해야 한다.
  - **Is-a 규칙**: 서브타입의 모든 인스턴스는 슈퍼타입 집합에 포함돼야 한다. 서브타입 is a 슈퍼타입을 만족해야 한다.

상속

- **일반화의 원칙**: 한 타입이 다른 타입의 서브타입이 되기 위해서는 슈퍼타입에 순응해야 한다.
- 순응에는 구조적인 순응과 행위적인 순응이 있으며, 두 가지 모두 특정 기대 집합에 대해 서브타입의 슈퍼타입에 대한 **대체 가능성**을 의미한다.
    - **구조적인 순응(structural conformance)**

      타입의 내연과 관련된 100% 규칙을 의미하므로, 서브타입이 슈퍼타입을 대체하더라도 구조에 관한 동일한 기대 집합을 만족시킬 수 있다.
      ex) Person이 name 속성을 가진다면, Person의 서브타입인 Employee 역시 name 속성을 가질 것이라고 기대할 수 있다.
    - **행위적인 순응(behavioral conformance)**

      서브타입은 슈퍼타입을 행위적으로 대체 가능해야 한다. 행위적인 순응을 흔히 **리스코프 치환 원칙(LSP)**이라고 한다.

      ex) Person이 getAge()에 대한 응답으로 나이를 반환한다면, Employee 역시 getAge()에 대한 응답으로 나이를 반환해야 한다.
- 상속은 서브타이핑과 서브클래싱의 두 가지 용도로 사용될 수 있다.
    - **서브타이핑(subtyping)**
        - 서브클래스가 슈퍼클래스를 대체할 수 있는 경우
        - 설계의 유연성이 목적이다.
        - **인터페이스 상속(interface inheritance)**이라고 한다.
    - **서브클래싱(subclassing)**
        - 서브클래스가 슈퍼클래스를 대체할 수 없는 경우
        - 코드의 중복 제거와 재사용이 목적이다.
        - **구현 상속(implementation inheritance)**이라고 한다.
- 여러 클래스로 구성된 상속 계층에서 수신된 메시지를 이해하는 기본적인 방법은 클래스 간의 **위임(delegation)**을 사용하는 것이다.
    - ex) 어떤 객체의 클래스가 수신된 메시지를 이해할 수 없다면 부모 클래스로 위임한다.

💭

항상 인터페이스와 추상클래스가 각각 언제 어떻게 왜 사용되는지 구분하기 힘들었는데,, 이제야 좀 알겠다.

집합과 분해

계층적인 복잡성

- 복잡성은 '계층'의 형태를 띤다.
- 단순한 형태로부터 복잡한 형태로 진화하는 데 걸리는 시간은 그 사이에 존재하는 '안정적인 형태'의 수와 분포에 의존한다.
- **집합**: 안정적인 형태의 부분으로부터 전체를 구축하는 행위
- **분해**: 집합과 반대로 전체를 부분으로 분할하는 행위
- 집합은 전체의 내부로 불필요한 세부 사항을 감춰주므로 추상화 메커니즘인 동시에 캡슐화 메커니즘이다.

합성 관계

- 객체와 객체 사이의 전체-부분 관계를 구현하기 위해서는 **합성 관계**를 사용한다.
- 객체와 객체 사이에 단순한 물리적 통로가 존재하는 경우에는 **연관 관계**를 사용한다.
- 합성 관계는 생명주기 측면에서 연관 관계보다 더 강하게 객체들을 결합한다.

패키지

- 소프트웨어에는 물리적인 형체라는 것이 존재하지 않으므로 구조를 단순화하기 위해서는 서로 관련성이 높은 클래스 집합을 논리적인 단위로 통합해야 한다. → **패키지(package)** 또는 **모듈(module)**
- 패키지를 이용하면
    - 시스템의 전체적인 구조를 이해하기 위해 한 번에 고려해야 하는 요소의 수를 줄일 수 있다.
    - 개별 클래스가 아닌 클래스의 집합을 캡슐화함으로써 전체적인 복잡도를 낮출 수 있다.

세 가지의 관점이 뭐가 다른건지 알듯말듯하다.

분류와 인스턴스화는 공통적인 특성으로 묶은거,

일반화와 특수화는 계층 구조,

집합과 분해는 포함하는지?