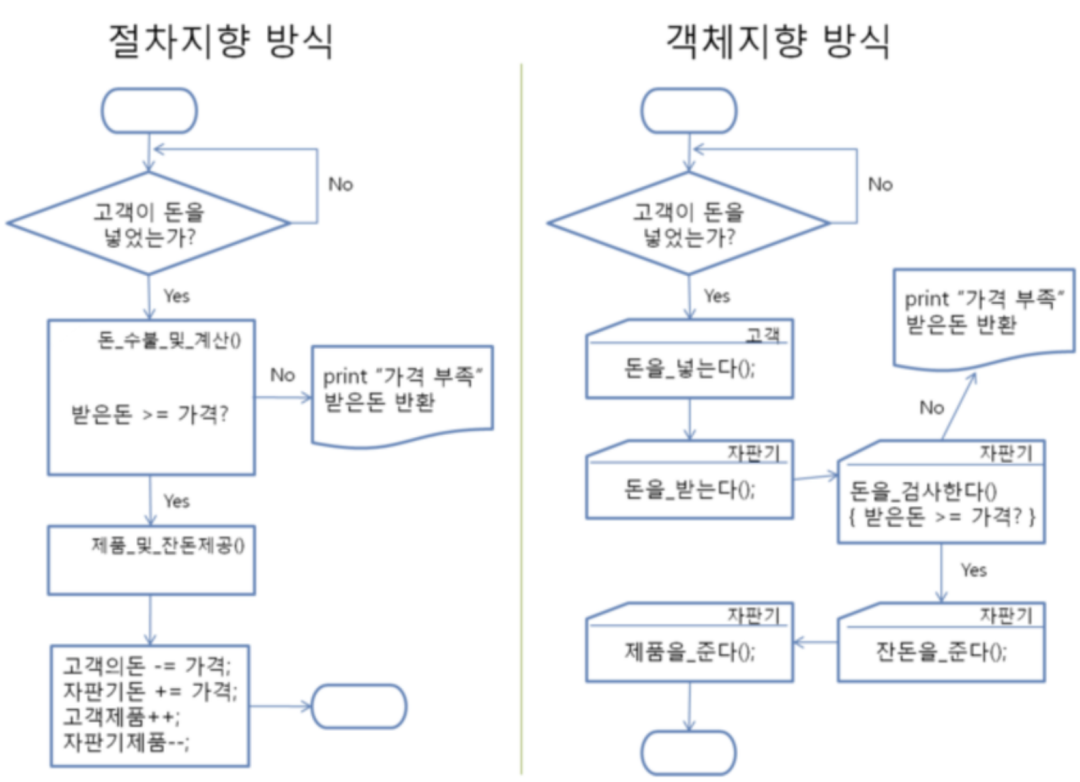
# OOP

| | |
|---|---|
| 과목 | 소프트웨어공학 |
| 날짜 | @2023년 3월 13일 |
| 발표자 | 윤선영 |

## 절차지향 VS 객체지향

| | 절차지향 | 객체지향 |
|---|---|---|
| 접근 방식 | Top-down | Bottom-up |
| 구현 관점 | 전체적인 기능 동작 고려 ⇒ 각 단계별로 기능 구현 | 필요한 속성의 객체를 설계 ⇒ 각 객체의 상호작용을 설계 |
| 구성 요소 | 함수 | 객체 |
| 접근 제어 | 없음 (모두 Public) | Public, Private, Protected |
| 오버로딩, 다형성 | 불가능 | 함수, 생성자, 연산자 등을 오버로딩 가능 |
| 상속 | 불가능 | 가능 |
| 보안성 | 낮음 | 높음 |
| 데이터 공유 | 모든 함수가 가능 | 객체 간 멤버 함수로 가능 |
| 예시 언어 | C | C++, Python, Java, Javascript |




⇒ 절차 지향은 데이터를 중심으로 함수를 구현하고, 객체지향은 기능을 중심으로 메서드를 구현한다.

### 설계 방식의 차이

절차지향 프로그래밍은 프로그램의 순서와 흐름을 먼저 세우고, 필요한 자료구조와 함수들을 설계하는 방식이다.

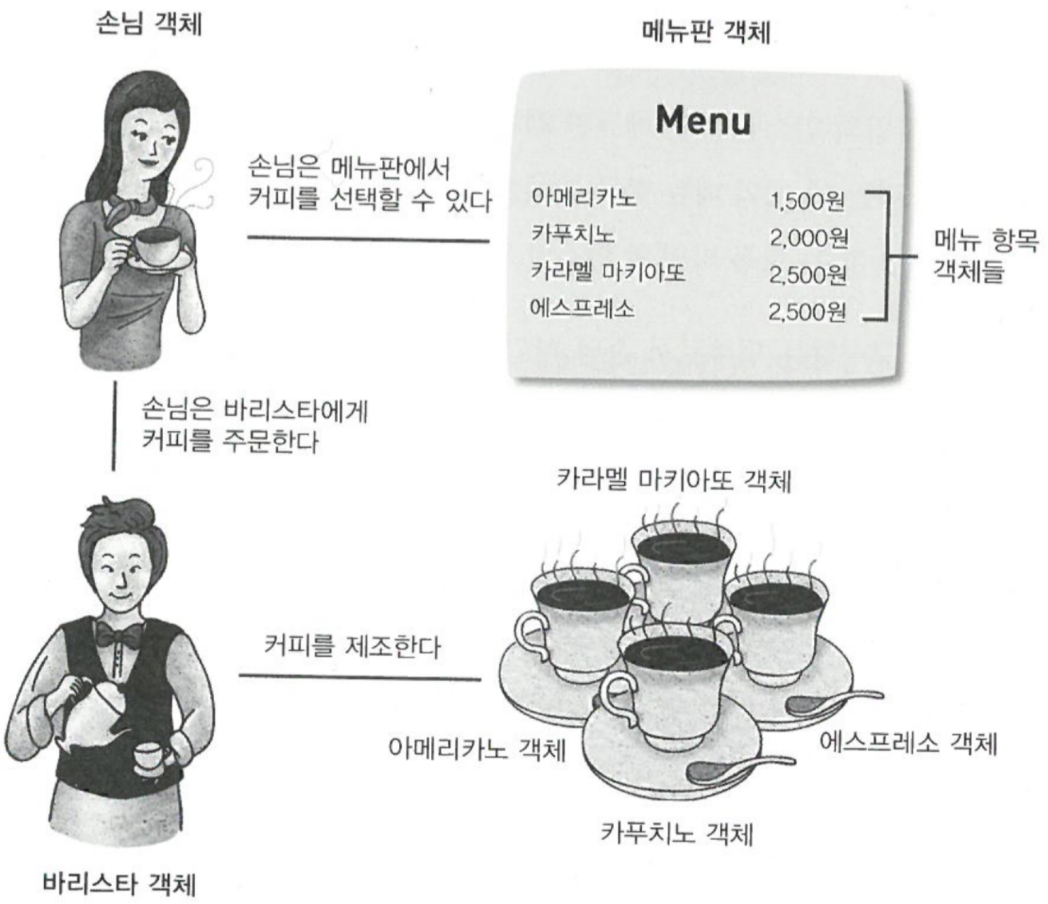
객체지향 프로그래밍은 자료구조와 이를 중심으로 한 모듈들을 먼저 설계하고, 이들의 실행 순서와 흐름을 조합하는 방식이다.

### 목적

절차지향 프로그래밍은 실행순서, 즉 절차가 중심이 된다.

객체지향 프로그래밍은 필요한 객체들의 종류와 속성들이 더 중심이 된다.

## 카페를 객체지향적으로 생각하기




[출처] 우아한형제들 기술 블로그, 책 '객체지향의 사실과 오해'

## 각 객체 간 관계를 파악하기

- 메뉴판 ↔ 손님 : 손님은 메뉴판에서 메뉴를 선택한다.
- 바리스타 ↔ 손님 : 손님은 바리스타에게 커피를 주문한다.
- 바리스타 ↔ 커피 : 바리스타는 커피를 만든다.

## 객체들을 분류하기

- 손님 객체는 '손님 타입'의 인스턴스
- 바리스타 객체는 '바리스타 타입'의 인스턴스
- 4가지 커피 모두 '커피 타입'의 인스턴스
- 메뉴판 객체는 '메뉴판 타입'의 인스턴스
- 4가지 메뉴 항목 객체들 모두 동일한 '메뉴 항목 타입'의 인스턴스

- 메뉴판 객체는 4개의 메뉴 항목 객체를 포함

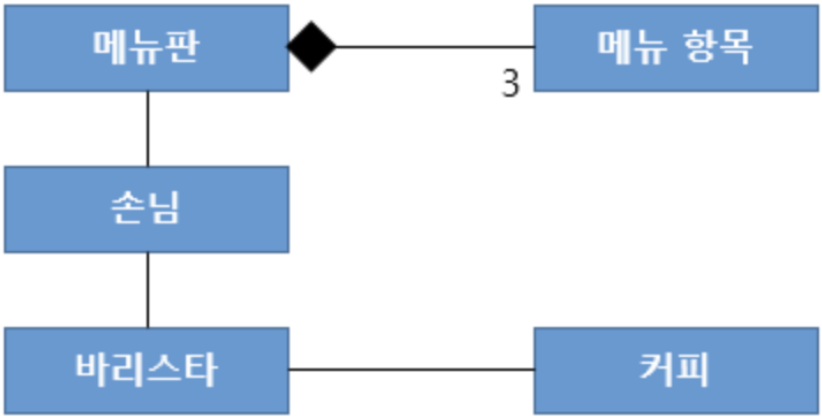
## 도메인 모델

도메인이란 소프트웨어로 해결하고자 하는 문제 영역을 의미한다. 여기서 도메인은 '카페'가 될 것이다.

'카페'라는 도메인은 '주문', '제조', '서빙' 등의 하위 도메인으로 쪼개질 수 있다.

도메인 모델은 특정 도메인은 개념적으로 표현한 것이다. 도메인을 모델링하는 방법은 여러 가지가 있는데,

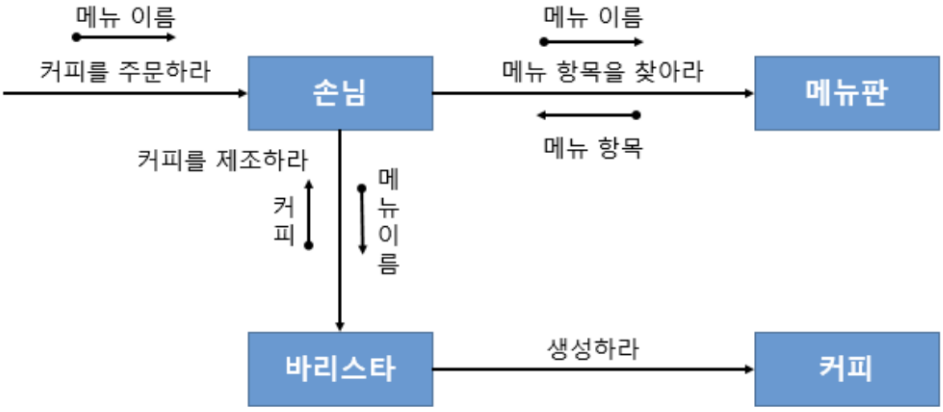
아래는 카페와 관련된 객체들을 타입과 관계를 이용해 추상화한 도메인 모델이다.




메뉴 항목은 메뉴판에 포함(합성)되는 관계이다. 이 때, 마름모 기호로 표현하며, 1:N 관계임을 알 수 있다.

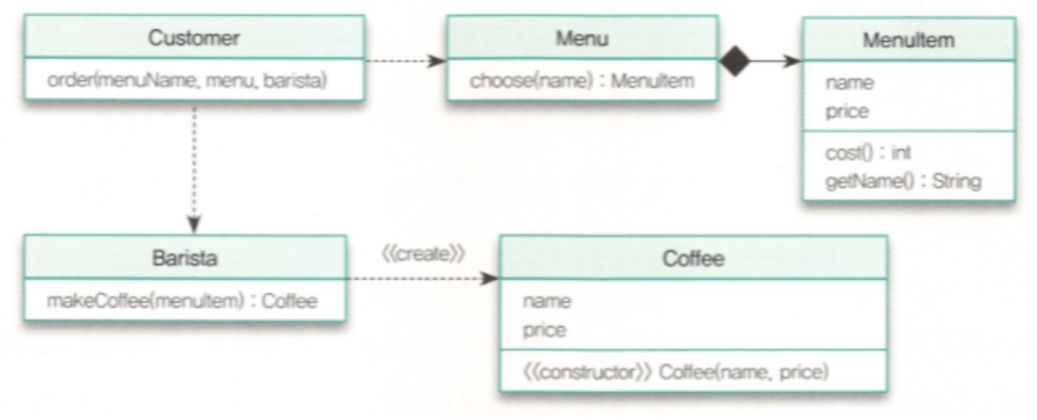
손님과 메뉴판은 포함(합성) 관계는 아니지만, 서로 알고 있어야 할 연관 관계이다.

## 객체지향적으로 설계하기




- 훌륭한 객체를 설계하기보다, 훌륭한 협력 관계를 설계하라.
- 객체가 메시지를 선택하는 것이 아닌, 메시지가 객체를 선택하도록 해라.
- 메시지를 선택하고, 이를 수신하기에 적합한 객체를 선택해라.
- 예를 들어, `커피를 주문하라` 라는 메시지는 손님이 수신하여야 한다.
- 손님 객체는 `커피를 주문하라` 라는 책임을 할당 받았다. 손님이 할당된 책임을 수행하는 도중에 스스로 할 수 없는 것들은 다른 객체에게 도움을 요청해야 한다.
- 손님은 주문을 하기 위해 `메뉴 항목을 찾아라` 라는 요청 또한 해야한다. 이는 메뉴판 객체가 해줄 수 있다. 손님은 메뉴판을 통해 메뉴 항목을 얻었고, `커피를 제조` 해 달라고 요청한다.
- 이 메시지는 바리스타만이 할 수 있다. 손님은 메뉴 항목을 메시지의 인자로 바리스타에게 전달하고 반환값으로 제조된 커피를 받아야 한다.
- 바리스타가 커피를 만들면 커피 주문의 협력은 끝난다.

## 클래스 다이어그램



## 오퍼레이션 순서

1. `손님`은 `메뉴판`에게 메뉴 항목 이름에 해당하는 `메뉴 항목`을 요청
2. `메뉴 항목`을 받아 `바리스타`에게 원하는 `커피`를 제조하도록 요청

## 메뉴 항목

```
public class MenuItem{
  private String name;
  private int price

  public MenuItem(String name, int price){
    this.name = name;
    this.price = price;
  }

  public int cost() {
    return price;
  }

  public String getName() {
    return name;
  }
}
```

## 커피

```
class Coffee {
  private String name;
  private int price;

  public Coffee(MenuItem menuItem){
    this.name = menuItem.getName();
    this.price = menuItem.cost();
  }
}
```

## 메뉴판

```
class Menu {
  private List<MenuItem> items;

  public Menu(List<MenuItem> items){
```

```
    this.items = items;
  }

  public MenuItem choose(String menuName){
    for(MenuItem each : items) {
      if(each.getName().equals(menuName)){
        return each;
      }
    }
    return null;
  }
}
```

### 바리스타

```
class Barista {
  public Coffee makeCoffee(MenuItem menuItem){
    Coffee coffee = new Coffee(menuItem);
    return coffee;
  }
}
```

### 손님

```
class Customer {
  public void order(String menuName, Menu menu, Barista barista) {
    MenuItem menuItem = menu.choose(menuName);
    Coffee coffee = barista.makeCoffee(menuItem);
    ...
  }
}
```

## UML (Unified Modeling Language)

- 시스템 분석, 설계, 구현 등 시스템 개발 과정에서 개발자와 고객 또는 개발자 상호 간의 의사소통이 원활하게 이루어지도록 표준화한 대표적인 객체지향 모델링 언어
- 시스템 구조를 표현하는 6개의 구조 다이어그램과 시스템의 동작을 표현하는 7개의 행위 다이어그램이 있음
- 구성 요소로는 사물, 관계, 다이어그램이 있음

## UML 관계

- 사물과 사물 사이의 연관성을 표현하는 것
- 연관관계, 집합관계, 포함관계, 일반화관계, 의존관계, 실체화 관계 등

### 1. 연관관계

• 2개 이상의 사물이 서로 관련되어 있음을 의미

• 사물 사이를 실선으로 연결하여 표현

### 2. 집합관계

• 하나의 사물이 다른 사물에 포함되어 있는 관계를 표현

• 포함되는 쪽에서 포함하는 쪽으로 마름모를 연결하여 표현

## UML 다이어그램

- 사물과 관계를 도형으로 표현한 것

### 클래스 다이어그램

• 클래스와 클래스가 가지는 속성, 클래스 사이의 관계를 표현

◦ 시스템의 구조를 파악하고 구조상의 문제점을 도출할 수 있음

### 객체 다이어그램

• 클래스에 속한 사물(객체)들, 즉 인스턴스를 특정 시점의 객체와 객체 사이의 관계로 표현

### 다음 시간

다이어그램 더 자세하게, 구성 요소는 무엇인지, 그리는 방법 등