Java II - 객체지향 프로그래밍 패러다임 I

Programming Paradigm

- 프로그래밍을 보는 관점
- 프로그램의 일처리를 위해서 어떻게 코드들을 구성할까?

객체지향 프로그래밍 (Object - Oriented Programming, OOP)

- 좀 더 나은 프로그램을 만들기 위한 프로그래밍 패러다임으로
  로직을 상태(state)와 행위(behave)로 이루어진 객체로 만드는 것
  이 객체들을 마치 레고 블럭처럼 조립해서 하나의 프로그램을 만드는 것

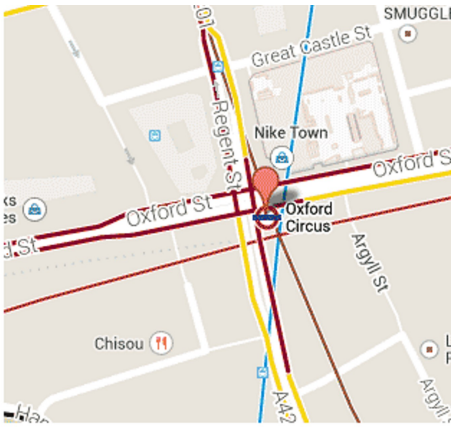
- **프로그램을 상호작용하는 객체들의 집합으로 표현, 설계가 주가 되는 프로그래밍**
  - 상태(State) : Variable
  - 행위(Behave) : Method

문법과 설계

문법 객체지향을 편하게 할 수 있도록 언어가 제공하는 문법을 익히는 것이다

설계

- 현실 → S/W
- 좋은 설계는 현실을 잘 반영해야 한다. 현실은 복잡하다. 하지만 그 복잡함 전체가 필요한 것은 아니다.



Abstract(추상화) : 위성사진처럼 **복잡한 현실을** 지하철노선도 처럼 우리의 관심사에 맞게 **단순화**시킨다.

부품화 (모듈화)

- **객체를 어디서든 재활용 가능하도록 부품화 시킨다.**
- **객체지향의 정점은 부품화**

i. **초창기의 컴퓨터**

ii. **컴퓨터를 기능별로 추상화하여 부품화**

기능들을 부품화 시킨 덕분에 소비자들은 더 좋은 키보드나 저렴한 모니터를 선택할 수 있게 되었다.
또 문제가 생겼을 때 그 문제가 어디에서 발생한 것인지 파악하고 해결하기가 훨씬 쉬워진다.

객체지향의 등장이유

i. **메소드는 중복되는 로직들을 그룹핑한 것**이므로, 부품화의 예이다.
메소드를 사용하면 코드의 양을 극적으로 줄일 수 있고, 메소드 별로 기능이 분류되어 있기 때문에 필요한 코드를 찾기도 쉽고 문제의 진단도 빨라진다.

ii. **그런데 프로그램이 커지면서 엄청나게 많은 메소드들이 생겨나게 된다.** 메소드와 변수를 관리하는 것은 점점 어려워진다.
급기야는 메소드가 없을 때와 같은 상황을 맞이하게 된다.

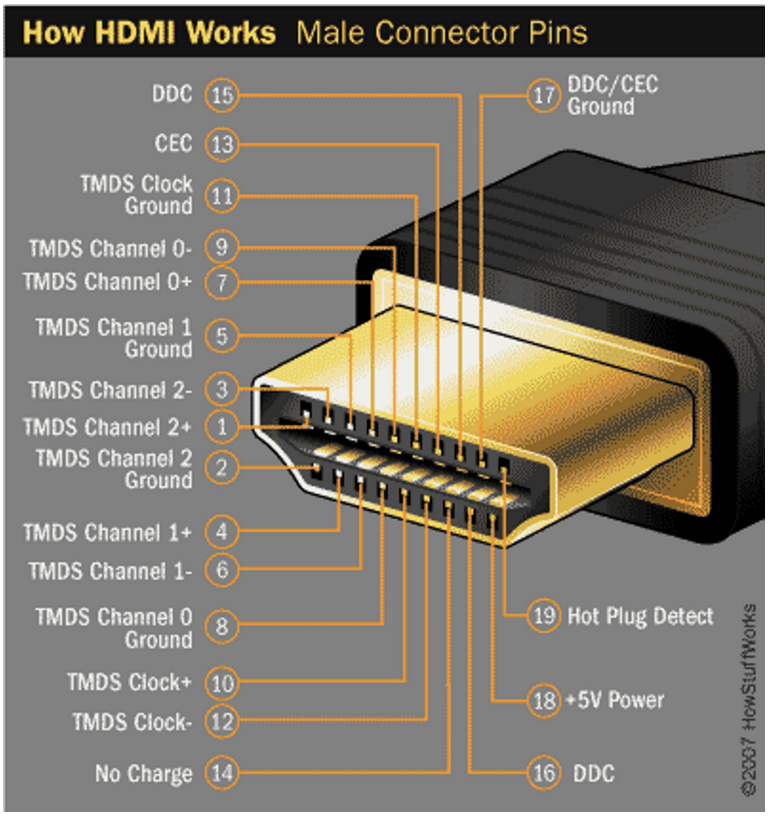
iii. 이 문제에서의 도약 중의 하나가 객체 지향 프로그래밍이다. 이것의 핵심은
연관된 메소드와 그 메소드가 사용하는 변수들을 분류하고 그룹핑하는 것이다.
바로 그렇게 그룹핑 한 대상이 객체(Object)다.

은닉화, 캡슐화

- **은닉화, 캡슐화**
제대로된 부품이라면 그것이 어떻게 만들어졌는지 모르는 사람도 그 부품을 사용하는 방법만 알면 쓸 수 있어야 한다.
즉 내부의 동작 방법을 단단한 케이스(객체) 안으로 숨기고 사용자에게는 그 부품의 사용방법(Method)만을 노출시키는 것

인터페이스

잘 만들어진 부품이라면 부품과 부품을 서로 교환 할 수 있어야 한다.



컴퓨터와 모니터를 만드는 업체들은 위와 같은 케이블의 규격을 공유한다. 각각의 부품은 미리 정해진 약속에 따라서 신호를 입, 출력하고 연결점의 모양을 표준에 맞게 만들면 된다. 이러한 **연결점을 인터페이스(interface)**라고 한다.

실생활에서의 Object And Class

- **Object (객체)**
  - 모든 사물이나 대상
  - 의식, 특징이나 행위를 가지는 형체

- **Class**
  - 객체가 가질 수 있는 속성(상태)과 행위를 글로써 정의하는 틀(설계도)

Car Class (자동차 설계도)

상태(Variable)	색상	연료형식

	전진	후진
기능(Method)	Gear D	Gear R
기능(Method)	Accelerator On	Accelerator On

객체생성

↓

Car Object (자동차)

BMW	속성 : 흰색, 휘발유	기능: 전진, 후진
AUDI	속성 : 검은색, 경유	기능: 전진, 후진
Chevrolet	속성 : 베이지색, 휘발유	기능: 전진, 후진

- 프로그래밍 에서는 객체를 생성하는 것은 실체를 생성하는 것이다.
  (메모리에서 실제 동작하게 만드는 것)

- 자동차 객체 a, b, c를 생성하면 메모리 상에서 서로 상호작용한다. (서로 치고박는다.)
- 객체마다 필요한 메소드만 호출할 수 있다.