3주차 과제: 클래스

- ## 클래스란?

먼저 클래스가 무엇인지 우리는 어렴풋이 알고 있을 것이다.

객체 지향에 대해 공부할 때 class가 **붕어빵 틀**이면 instance가 그 틀로 **찍어낸 붕어빵**들 이라는 표현을 가장 많이본 기억이 나고,
앞서 1, 2주차에서 공부한 내용에서는 JVM 이 .class 파일로 컴파일 할 때, 올라가는 단위가 class 였고, 변수의 종류 중 전역변수가 **클래스 변수**와 **인스턴스 변수**로 세분화 되었던 것이 기억이 난다.

우선, 위키백과에서 클래스에 대한 정의는 다음과 같다.

> **클래스**(class, 어원: **classification**)는 객체 지향 프로그래밍(OOP)에서 특정 객체를 생성하기 위해 변수와 메소드를 정의하는 일종의 틀(template)이다.

그럼, 객체 지향 프로그래밍(OOP) 란 무엇이냐
물론 너무 많이 들었겠지만, 면접 질문으로 나왔다고 하니까 한 번 짚고 넘어가보자.

객체지향 프로그래밍 (OOP) 이란 무엇인가?

참고하면 좋은 링크입니다. https://jeong-pro.tistory.com/95

객체지향 프로그래밍이란 컴퓨터 프로그래밍 방식중 하나로, 프로그래밍에서 필요한 데이터를 추상화 시켜서
상태와 행위[변수와 메소드]를 가진 객체를 만들고 그 객체들간의 유기적인 상호작용을 통해 로직을 구성하는 프로그래밍 방법 입니다.

출처: https://coinco.tistory.com/157

여기서 주목해볼 키워드가 추상화 와 상태와 행위를 가진 객체 정도가 될 것 같다. 이 두 키워드를 기억하고 클래스가 나오게 된 배경을 살펴보자.

- ## 클래스가 나오게 된 배경

우선 지금까지도 사용되는 C언어는 클래스라는 개념이 없다! 그리고 파이썬 프로그램을 보면 클래스를 사용하지 않고도 작성된 코드들이 꽤 많다. 즉, 클래스가 필수는 아니란 말인데, C 언어

다음 C 언어에 객체 지향을 도입한 C++ 이 생겼다는 말은 클래스를 사용함으로써 분명 이득이 있기 때문이다.

클래스는 어떻게 생기게 되었을까?

클래스는 1966 Alan Kay 라는 개발자가 클래스, 객체지향 프로그래밍(OOP)라는 개념 탄생시켰다.

위 링크는 좀 더 객체 지향에 중점을 두고 설명을 한 포스팅이다. 예시로 Swift 언어 등으로 설명해준다. 읽어보면 재밌다.

여기서는 클래스에 대한 설명에 중점을 두고 Python으로 예시를 들어보겠다!

class 설명할 때 맨날 나오는 그 붕어빵 틀 Car 을 클래스 없이 살펴보자.
먼저 자동차에 대한 데이터들을 저장할 수 있는 **자료 구조**가 필요하다. 일단 딕셔너리로 구현해보자.

```
car = {
    "maker"               : "BENZ", # 차량 브랜드
    "year"                : 2020, # 출시 년도
    "model"               : "E-Class", # 모델명
    "AI Auto Drive Mode" : True # AI 자동 주행 가능 유무
}
```

car 에 필요한 데이터를 만들었으니, 이제 car 가 실행할 수 있는 **기능**, 즉 동작들을 **함수**로 표현해보자!
drive 라는 함수를 구현해볼건데, 요즘은 세상이 좋아졌으니 AI 자동 주행 옵션 유무에 따라 운전 방식이 달라질 것이다.

```
# car.py

def drive(car):
    if car[`AI Auto Drive Mode`]:
        print("인간 시대의 끝이 도래했다")
    else:
        print("알파고 그냥 망치로 깨부시면 끝 아니냐")
```

이제 이 데이터와 함수(기능)을 합쳐서 실행해보면

```
from car import drive

car = {
```

```
    "maker"                : "BENZ",
    "year"                 : 2020,
    "model"                : "E-Class",
    "AI Auto Drive Mode" : True
}
drive(car) #인간 시대의 끝이 도래했다
```

뭐야 클래스 없어도 잘만 돌아가는데?
싶지만,

하지만 클래스를 괜히 만들진 않았을거다. 분명 문제가 있으니까 그걸 해결하기 위해 클래스를 도입했겠지..?

```
from car import drive

car = {
    "maker" : "BENZ",
    "year"  : 2020,
    "model" : "E-Class"
}

drive(car) # KeyError!
```

위 코드에서 drive 함수에 매개변수로 넘겨지는 딕셔너리에 요구되는 필드인 `AI Auto Drive Mode` 가 없는 경우 **KeyError** 가 난다.

즉, 우리는 drive 함수를 실행할 때 drive 함수에 대한 이해가 필요하다.

1. 매개 변수로 넘겨지는 car 에 정확히 어떤 데이터가 있어야하는지
2. car 이 어떤 자료 구조인지

등 말이다.

문제는 drive 함수를 이해하기 위해서는 해당 코드를 직접 뜯어보거나, 해당 함수의 API 문서를 읽어야하는데 요구되는 데이터의 구조와 정확한 필드명까지 모두 이해해야하는 건 너무 불편하고 오류가 나기 쉬운 구조이다.

그럼 이 문제를 해결하기 위해서는?
*데이터와 함수를 분리해서 정의하지 않고 합쳐서 정의*하면 된다!
즉, 이 둘을 하나로 묶어서 사용하기 위해 만든 개념이 바로 **Class** 이다.

```
# car.py
```

```
class Car:
    def __init__(self, maker, year, model, ai_auto_drive_mode):
        self.maker              = maker
        self.year               = year
        self.model              = model
        self.ai_auto_drive_mode = ai_auto_drive_mode

    def drive(car):
        if car[`AI Auto Drive Mode`]:
            print("인간 시대의 끝이 도래했다")
        else:
            print("알파고 그냥 망치로 깨부시면 끝 아니냐")
```

```
from car import Car

car = Car(
    maker              = "BENZ",
    year               = 2020,
    model              = "E-Class",
    ai_auto_drive_mode = True
)

car.drive() #인간 시대의 끝이 도래했다
```

클래스를 사용함으로써 서로 관계된 데이터와 함수를 하나로 묶었기 때문에 일단 Car라는 클래스의 객체가 생성되면 아래와 같은 장점이 생긴다!

1. 앞서 필요했던 drive 함수에 대한 이해가 없이도 사용 가능.
2. Car 에 속하는 함수/데이터인지 아닌지 명확.

또한, 만약 차 레이싱 게임 프로그램이 있다고 가정하자.
Car 클래스가 없다면 8대의 레이싱 카가 서로 경쟁한다고 할 때, 차마다 현재 위치를 보여주는 변수와 차의 속도만큼을 매개변수로 받아 차를 이동시키는 함수가 필요할 것이다. 하지만 클래스를 사용한다면 재사용성이 높아지고, 코드가 훨씬 단순해진다. 아마 이 점이 클래스 및 객체 지향 프로그램의 장점을 꼽을 때 가장 먼저 나오는 게 아닌가 싶다.

그럼 이제 앞에서 객체 지향의 키워드 중 `상태와 행위를 가진 객체` 부분은 이해가 됐다. 그럼 `추상화`는?

먼저, 추상이란 단어의 뜻과 객체 지향에서 사용하는 추상화의 뜻을 먼저 살펴보자.

추상
여러가지 사물이나 개념에 공통되는 특성이나 속성을 추출해 파악하는 행위

추상화
클래스간의 공통점을 찾아내서 공통의 부모를 설계하는 작업

여기서는 추상에서의 공통되는 특성이나 속성을 추출해 파악하는 행위를 클래스간의 공통점을 찾아낸다고 생각하였지만, 좀 더 확장해서 생각해보면 아까 Car 라는 클래스를 만들기 위해 Car 가 가진 '브랜드명', '출시년도', '모델명', 'AI 주행 옵션 유무', '운전하기 기능' 을 설정하는 것도 사람들이 생각하는 '차'를 클래스로 정의하기 위해 각각의 차들이 가진 특성과 속성을 찾아냈다고 생각할 수도 있을 것 같다.

추상화에서는 클래스의 부모라는 개념이 등장하는데 이는 상속 파트에서 자세히 알아보자!

이렇게 클래스에 대한 정의 및 탄생 배경, 필요한 이유에 대해 알아보았는데 클래스를 더 깊게 이해하고 설명하기 위해서는 꼭 알아야 하는 3가지 개념이 있다.
encapsulation (캡슐화) , inheritance(상속), 그리고 polymorphism(다형성) 이다.

inheritance와 polymorphism은 클래스가 처음 생겼을때 부터 적용된 개념은 아닌데 객체 지향 프로그래밍 언어들이 발전하면서 나중에 도입된 개념이다.
상속은 상속 공부하는 주가 따로 있고, 다형성은 인터페이스와 밀접한 관련이 있기 때문에 여기서는 캡슐화만 다루고 넘어가겠다.

상속과 다형성이 나중에 도입된 개념이란 뜻은 캡슐화는 클래스가 생길 때부터 있었다는건데, 이는 클래스의 구조와 함께 살펴보자.

위 코드를 살펴보면 클래스를 구성하는 모든 요소들에 **접근제어자**가 있는 것을 확인할 수 있는데, 이 접근제어자가 **캡슐화**와 관련이 있다.

캡슐은 안의 내용물은 드러나지 않는다. 마찬가지로 encapsulation은 클래스의 정보에 대한 접근 권한을 제어함으로써 안의 정보가 드러나지 않는 것을 말한다. Hiding infromation 혹은 hiding data라고 한다.

그럼 이 구조를 바탕으로 클래스를 구성하는 요소를 하나씩 살펴 보자!

1. 클래스 정의하는 방법

정의

```
접근제어자 class 클래스이름 {
    접근제어자 필드1의타입 필드1의이름;
    접근제어자 필드2의타입 필드2의이름;
    ...
    접근제어자 메소드1의 원형
    접근제어자 메소드2의 원형
    ...
};
```

2. 자바의 클래스 멤버 변수

+ **면접 질문!**

Q. 자바의 클래스 멤버 변수 초기화 순서에 대해 설명하세요

1. 기본값
2. 명시적 초기화
3. 초기화 블럭
4. 생성자(인스턴스 변수만 해당)

먼저 지난 시간 변수에 대해 공부할 때, 우리는 변수가 클래스 변수와 인스턴스 변수로 나누어지고 클래스 변수는 클래스가 처음 로딩될 때 딱 한 번 초기화가 진행되고, 인스턴스 변수는 인스턴스가 생성될 때마다 인스턴스별로 초기화가 이루어짐을 알 수 있었다.

```
class Car{
    int instanceVariable = 1; // 인스턴스 변수
    static int staticVariable =1 ; // 클래스 변수(static, 공유)

     // 인스턴스 초기화 블럭
    {        instanceVariable = 2;          }
    // 클래스 초기화 블럭, 앞에 static이 붙음.
    static{ staticVariable = 2; }

    // 생성자, 인스턴스 변수 초기화 과정이 들어있음.
    Car(){
        instanceVariable = 3;
    }

}
```

위 코드가 실행된다고 할 때, 클래스 변수의 경우

1. 해당 클래스가 처음 로딩될 때, staticVariable 은 0으로 기본값을 갖는다.
2. 명시적 초기화 적용으로 staticVariable 은 0에서 1로 값이 바뀐다.
3. 클래스 초기화 블럭에서 staticVariable 은 1에서 2로 값이 바뀌게 된다.

똑같은 원리로 인스턴스 변수의 경우,

1. 인스턴스가 생성된다.
2. 기본값으로 instanceVariable 은 0의 값을 갖는다.
3. 명시적 초기화로 instanceVariable은 0에서 1로 값이 바뀐다.
4. 인스턴스 초기화 블럭에서 instanceVariable은 1에서 2로 값이 바뀐다.
5. 생성자에서 instanceVariable 의 값은 2에서 3이 된다.

즉, 변수의 초기화 순서는 아래와 같다.

기본값 -> 명시적 초기화 -> 초기화 블럭 -> 생성자(인스턴스에만 해당)

+ 복습겸 함께 보면 좋을 것 같은 자료 : java 클래스 변수 : 어떻게 메모리에 올라갈까?

3. 메소드 정의하는 방법

메소드 정의

클래스에서 메소드를 정의하는 방법은 일반 함수를 정의하는 방법과 크게 다르지 않습니다.

자바에서 메소드를 정의하는 방법은 다음과 같습니다.

문법

```
접근제어자 반환타입 메소드이름(매개변수목록) { // 선언부
    // 구현부
}
```

1. 접근 제어자 : 해당 메소드에 접근할 수 있는 범위를 명시합니다.
2. 반환 타입(return type) : 메소드가 모든 작업을 마치고 반환하는 데이터의 타입을 명시합니다.
3. 메소드 이름 : 메소드를 호출하기 위한 이름을 명시합니다.
4. 매개변수 목록(parameters) : 메소드 호출 시에 전달되는 인수의 값을 저장할 변수들을 명시합니다.
5. 구현부 : 메소드의 고유 기능을 수행하는 명령문의 집합입니다.

출처: http://www.tcpschool.com/java/java_methodConstructor_method

4. 생성자 정의하는 방법

생성자의 선언

자바에서 클래스 생성자를 선언하는 문법은 다음과 같습니다.

문법

```
1. 클래스이름() { ... }                    // 매개변수가 없는 생성자 선언
2. 클래스이름(인수1, 인수2, ...) { ... } // 매개변수가 있는 생성자 선언
```

자동으로 생성되지만, 커스터마이징 하기 위해 사용자 정의 생성자를 생성하기도 한다.

생성자가 있다는 뜻은 소멸자도 있다는 뜻인데 Java 에서 소멸자는 대부분 사용하지 않는다. 이유가 궁금해서 찾아본 결과 아래와 같다.

자바에서는 사용되지 않는 객체를 GC가 자동으로 메모리에서 해제시키는데, 객체가 GC에 의해 해지가 되면 finalize() 메소드, 즉 소멸자가 호출되며 제거되는 방식이다. 이 때, 강제로 GC가 메모리 해제를 시키기를 원한다면 사용하는 것이 GC를 호출하는 system.gc() 혹은 Runtime.getRuntime().gc() 인데 어째서인지 Effective Java라는 책에서는 이 둘의 사용을 최대한 피하라고 한다. 이유는 아래와 같다.

1. finalize 는 언제 수행되는지도 알 수 없으며, 수행을 반드시 보장하지 않는다.

   system.gc 를 실행한다고해서 GC가 바로 작동하는 것이 아니다. GC는 JVM 구현마다 다르기 때문에 finalize가 언제 수행되는지 알 수 없다. 심지어, finalize 의 수행 자체가 반드시 보장되는 것도 아니다..(..) 원래는 반드시 finalize 를 수행하는 메서드들이 존재했으나, 심각한 결함이 있어 폐기되었다.

2. finalize 에서 발생하는 예외는 무시된다.
3. finalize 를 재정의할 경우 성능 저하가 발생한다.
4. Finalizer 공격이라는 보안 이슈에도 사용될 수 있다(..)

이런 이유로 finalize 를 사용하지 말라고 하는데... 그럼 왜 만든 걸까..? 왜 존재하는거지? 라고 생각하는데 Java 9부터 deprecated API로 지정되었다고 한다(..)

그 대안으로 cleaner 를 사용한다고 하는데 cleaner 역시 즉시 수행된다는 보장이 없어 제 때 실행되어야하는 작업(파일 닫기 등)은 불가능하고, finalize 보다는 덜 위험하지만 예측할 수 없고, 느리고, 일반적으로 불필요하다고 한다. 그럼 그냥 cleaner 를 굳이 대안으로 만들 필요가 있나? 아예 없애버리면 안되나? 싶지만 사용처가 있긴 했다.

1. 명시적 종료 메서드에서 호출되지 않을 것을 대비하는 방어 역할

   명시적 종료 메서드 = 자원을 다 사용하고 나서 메모리 해제를 명시적으로 하도록 만든 메서드. FileInputStream, FileOutputStream, Timer, Connection 등이 있다. 본인이 만든 걸 본인만 사용하면 모르겠지만, 다른 사람들이 사용할 때 해당 API를 올바르게 사용하지 않을 수도 있음을 고려해서, 대비책으로 finalize 메서드에 메모리 해제를 하도록 작성한다.

2. 네이티브 피어 리소스를 해제할 때

   네이티브 = 자바 외 다른 언어로 작성된 프로그램.

   이런 프로그램을 자바에서 다루기 위해 만들어놓은 객체를 네이티브 피어라고 하는데, 안타깝게도 이는 일반 객체가 아니므로 GC 관리 대상에서 제외된다. 그렇기 때문에 해당 네이티브 피어를 사용하는 일반적인 클래스의 finalize 를 실행함으로써 네이티브 객체의 리소스를 해제할 수 있다.

5. this 키워드 이해하기

this 참조 변수는 인스턴스가 바로 자기 자신을 참조하는 데 사용하는 변수이다.

이러한 this 참조 변수는 해당 인스턴스의 주소를 가리키고 있다! 주로 매개 변수와 객체 자신이 갖고 있는 변수의 이름이 같을 때 이를 구분하기 위해 사용한다.

갑자기 든 의문점이 파이썬에서 class 를 정의할 때는 self 를 사용했는데, java와 C++, C#은 this 를 사용하는 이유가 뭘까? self 와 this 는 기능상으로 차이가 있는걸까? 라는 의문이 들었다.

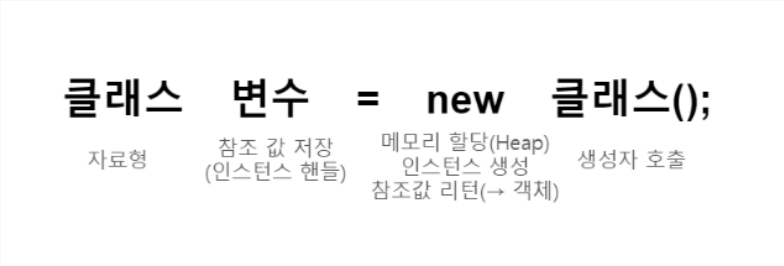
답이 생각보다 별 거 없었다... 그냥 토막 상식(?) 정도로 알고 있으면 좋을 것 같다.

> 관례적으로는 self 를 사용한다. 객체 지향의 시초인 smalltalk 에서도 self를 사용한다.
>
> 처음 스몰토크가 객체 지향이라는 개념을 갖고 오면서, 다른 객체 지향을 적용한 언어들에서 self 란 키워드를 사용하게 되었는데, C는 원래 객체 지향이 없으니까 개발자들이 스스로 self 라는 키워드를 사용해 객체 지향을 사용했다.
> 그런 C에 객체 지향을 도입한 C++ 이 C에서의 self 와 명칭이 겹치는 것을 피하기 위해 다른 키워드인 this 를 사용했고, 그대로 스몰토크 스타일을 따른 언어들은 self를, C++ 문법을 따르는 언어들은 this 를 사용하게 되었다고 한다!

혹시 관심이 있다면 자바스크립트의 경우 self와 this 를 모두 사용하는데 둘 차이점에 대해 설명한 링크도 있다! [자바스크립트] self와 this의 차이점

이렇게 정의된 클래스를 사용하기 위해서는 객체, instance 를 생성해주어야한다.

- **객체 만드는 방법 (new 키워드 이해하기)**



출처: [JAVA] 클래스란? (클래스, 객체, new, 메소드, 생성자, this...)

java에서는 new 연산자를 통해 객체를 생성할 수 있다. 2주차에서 공부 했듯이, 객체는 선언만 할 땐 저장 공간을 할당받지 못한다. new 연산자를 사용하여 객체가 생성되면, 메모리의 Heap 영역에 데이터를 저장할 공간을 할당받고, 그 공간의 참조값을 객체에게 반환해 준다!

일반 클래스 vs 추상 클래스

추가로! 같이 보면 좋을 것 같은 링크!

- 창시자 앨런 케이가 말하는, 객체 지향 프로그래밍의 본질
- 파이썬과 객체지향 프로그래밍: 5가지 클래스 설계의 원칙 (S.O.L.I.D) - 잔재미코딩

이 부분은 뒤에서 상속 다루고, 인터페이스 다루면서 함께 보면 좋을 것 같은데 일단 미리 가져와봤다.

- 면접 질문: 인터페이스와 추상 클래스의 차이는 무엇인가?

추상 클래스와 인터페이스 모두 상속 받는 클래스 혹은 구현하는 인터페이스 안에 있는 추상 메소드를 구현하도록 강제한다.

둘의 차이는,
추상클래스 = 추상 클래스를 상속 받아, 기능을 이용 및 확장.
인터페이스 = 함수의 구현을 강제하기 위해 함수의 껍데기만 두는 것.

자바는 다중 상속을 지원하지 않는다.

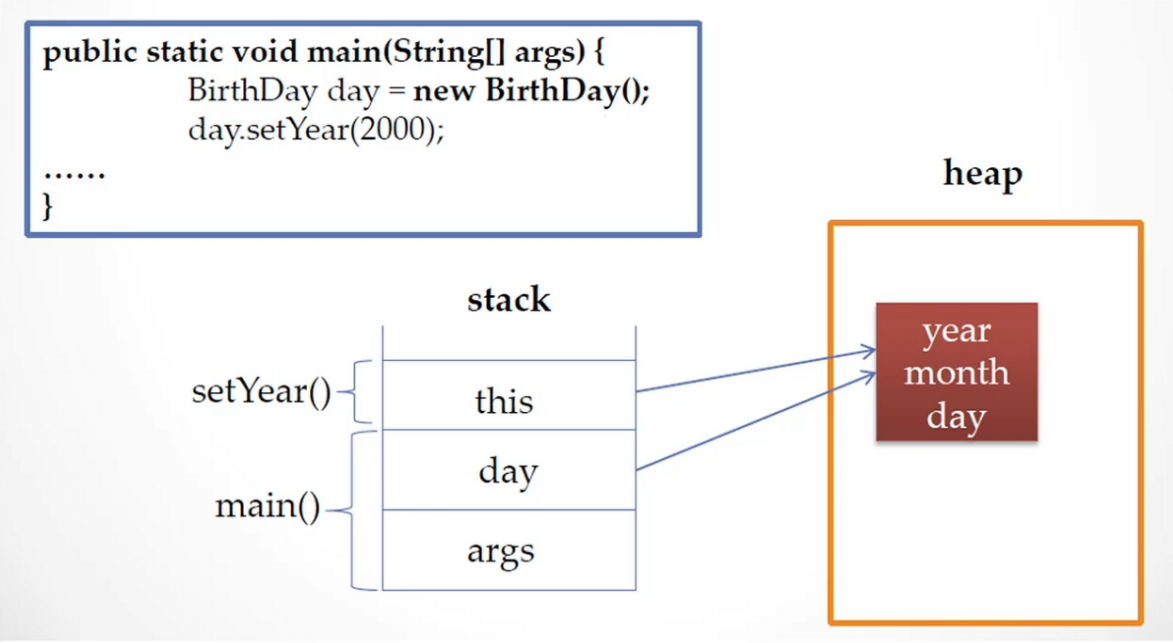
출처

- https://reakwon.tistory.com/26
- https://wikidocs.net/214
- http://www.tcpschool.com/java/java_class_intro
- 개발자 면접 질문: https://gem1n1.tistory.com/27
- https://byul91oh.tistory.com/228
- 추상 클래스 vs 인터페이스 차이 및 존재 이유
- https://ji-gwang.tistory.com/122?category=1041303
- https://rampart81.github.io/post/understanding-encapsulation/
- https://negabaro.github.io/archive/oop-abstract_class
- https://blog.naver.com/da91love/220942226757
- https://rampart81.github.io/post/understanding-class/
- http://www.tcpschool.com/java/java_class_declaration
- Java 클래스 멤버 변수들의 초기화 순서
- https://www.zehye.kr/java/2019/08/22/11java_constructor/
- https://coinco.tistory.com/157
- https://interconnection.tistory.com/129
- https://right1203.github.io/study/2018/09/17/java-2/
- 왜 파이썬은 self를 사용하고 C++, java, c# 은 this는 사용하는가?
- http://openlook.org/wp/why-self-in-python-is-attractive/
- https://markim94.tistory.com/138
- Java 클래스 멤버 변수 초기화 순서
- https://cano721.tistory.com/130
- https://camel-context.tistory.com/43

- this 는 언제 생성되고, 어디에 저장되는지

this 는 class 내에 사용자가 생성자를 선언하지 않으면 기본 생성자를 자동으로 생성해주듯, 컴파일 단계에서 컴파일러에 의해 생성됩니다.

다른 참조 변수들과 동일하게 stack 에 저장되어, Heap 에 저장되어 있는 객체를 가리키는 역할을 합니다.



이미지출처: [Java] 클래스와 객체 - this에 대하여