객체지향 용어 /Is - a / Has - a

객체 (Object)

- 실제로 존재하는 모든것
- 함수와 변수(데이터)를 함께 묶는 방법
- 속성(attribute) : 객체의 특징 또는 객체에 관해 알고 있는 사항, 숫자, 문자열 등과 같은 정보로 구성되어 있음. 즉 속성은 객체 안에 포함된 변수(데이터)임
- 매서드(methog) : 파이썬 행동 또는 객체에 대해, 객체를 통해 할 수 있는 것. 어떤 일을 하기 위해 호출(call)할 수 있는 코드 덩어리 .
- 객체 = 속성(변수) + 메서드(함수) : 그러므로 객체는 어떤 것에 대한 속성과 메서드를 함께 모으는 한가지 방법 , 여기서 속성은 정보고, 메서드는 행동

—

<기본 구성 요소 >

- 클래스(Class) : 같은 종류의 집단에 속하는 속성과 행위를 정의한 것으로 객체지향 프로그램의 기본적인 사용자 정의 데이터형이라고 할 수 있음 . 클래스는 다른 클래스 또는 외부 요소와 독립적으로 디자인해야만함 ,
- 객체(Object) : 클래스의 인스턴스(실제로 메모리상에 할당된 것). 객체는 자신 고유의 속성을 가지며 클래스에서 행위를 수행가능
- 매서드(Method) , 메시지(Message) : 클래스로부터 생성된 객체를 사용하는 방법으로서 객체에 명령을 내리는 메시지라 할 수 있음 , 메서드는 한 객체의 서브루틴 형태로 객체의 속성을 조작하는 데 사용됨, 또 객체 간의 통신은 메시지를 통해 이루어짐

- 객체만 지칭할 때는 객체라고 말하지만, 클래스와 연관지어서 말할 때는 인스턴스라고 부름

```
# Step 1 : class에 이름 저장, 보통 대문자로 시작 ( 클래스 이름 )
# Step 2 : def로 메서드 작성, 메서드의 첫 번째 매개변수는 보통 self를 지정 ( def 메서드)
```

```
class Dog :
  def bark(self):
    print("멍")

---

# 클래스를 만들었으면 클래스를 이용해 실제 객체를 만들어야 함, 이때 객체를 해당 클래스의 인스턴스라고 함
# 인스턴스 = 클래스()

class Dog :
  def bark(self):
    print ("bark!")

#인스턴스
  # 클래스 Dog 에 myDog 라는 이름을 할당하였는데 이 myDog 이 Dog의 인스턴스가 됨
myDog = Dog()
```

- 객체의 속성과 메서드 사용하기

```
class Dog:
  def bark(self):
    print("shit")

myDog = Dog()

#속성 추가하기
myDog.name = "hell_hound"
myDog.color = "white"
myDog.size = "tiny"

# 각 개체의 속성 출력
print("My Dog is" , myDog.name)
print("He is", myDog.color)
print("He is" , myDog.size)
```

Is- a

- 생성된 클래스 및 객체는 사속 관계에서 둘은 밀접하게 결합되므로 부모 또는 기저 클래스의 명세에 변경이 발생하면 코드가 손상될 위험이 존재

- 이러한 밀접한 관계는 클래스 계층 구조에서 좀 더 안정적인 기반을 마련한다는 의미이기도 함
- 상위 클래스의 기능을 하위 클래스가 물려받아 사용할 수 있다는 장점도 존재

Has - a

- 느슨하게 결합되므로 상속에 비해서 명세에 변경이 발생하더라도 구성 요소를 쉽게 변경할 수 있다는 의미
  (코드의 손상이 적거나 없다는 의미)
- 이러한점에서 유연성을 제공함

상속의 경우 is관계가 확실할때 사용할것

- 사람은 인간이다
- 고양이는 동물이다
- 게임 캐릭터는 엔티티이다. (?)

구성(Composition) 의 경우, 하나의 객체가 다른 객체를 (부분으로써) 갖거나, 하는 경우에 사용할 수 있습니다.

- 자동차는 배터리를 가지고 있다.
- 사람은 심장을 가지고 있다.
- 전투기는 HUD를 가지고 있다 .

Is-a & Has-a 예시

```
# IS  - A
class B:
  def __init__ (self, b_property):
    self.b_property = b_property

class A(B):
  def __init__(self, a_property , b_property):
    super().__init__(b_property)
    self.a_property = a_property


----------

# HAS - A

class B:
  def __init__ (self, b_property):
    self.b_property = b_property

class A :
  def __init__ (self, a_property , b):
    self.a_property = a_property
    self.b = B(b_property)
```

자 예시를 들어보자

```
# IS - A
class Animal :
  def __init__(self, name):
    self.name = name

  def eat(self):
    print(f"{self.name} is eating.")

class Dog(Animal):
  def __init__(self , name , breed):
    super().__init__(name)   # 부모 메소드 사용하는거  ( Animal)
    self.breed = breed

  def bark(self):
    print(f"{self.name} is barking")

my_dong = Dog("Buddy" , "hell hound")

print(my_dog.name) # Buddy 출력
print(my_dog.breed) # 헬하운드 출력
```

```
my_dog.eat() # Buddy is eating 출력
my_dog.bark() # Buddy is barking 출력
```