DS 부트 캠프 2024

한상곤(Sangkon Han)
부산대학교 정보컴퓨터공학부
2024/02/08

목차

클래스와 객체 지향(1/2)

객체 지향 프로그래밍(Object Oriented Programming, OOP)

"객체 지향 프로그래밍(영어: Object-Oriented Programming, OOP)은 컴퓨터 프로그래밍의 패러다임 중 하나이다. 객체 지향 프로그래밍은 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것이다. 각각의 객체는 메시지를 주고받고, 데이터를 처리할 수 있다.

객체 지향 프로그래밍은 프로그램을 유연하고 변경이 쉽게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용된다. 또한 프로그래밍을 더 배우기 쉽게 하고 소프트웨어 개발과 보수를 간편하게 하며, 보다 직관적인 코드 분석을 가능하게 하는 장점이 있다. 그러나 지나친 프로그램의 객체화 경향은 실제 세계의 모습을 그대로 반영하지 못한다는 비판을 받기도 한다." (출처: 위키백과)

객체 지향 프로그래밍 구성 요소

- 클래스(Class) - 속성(attribute)과 행위(behavior)를 정의한 것으로 객체 지향 프로그램의 기본적인 사용자 정의 데이터형(user defined data type)이라고 할 수 있음, 클래스는 다른 클래스 또는 외부 요소와 독립적으로 디자인
- 객체(Object) - 클래스의 인스턴스(실제로 메모리상에 할당된 것), 객체는 자신 고유의 속성(attribute)을 가지며 클래스에서 정의한 행위(behavior)를 수행, 객체의 행위는 클래스에 정의된 행위에 대한 정의를 공유함으로써 메모리를 경제적으로 사용
- 메서드(Method), 메시지(Message) - 객체에 명령을 내리는 메시지라 할 수 있음, 메서드는 한 객체의 서브루틴(subroutine) 형태로 객체의 속성을 조작하는 데 사용, 또 객체 간의 통신은 메시지를 통해 이루어짐

객체 지향 프로그래밍 구성 요소

```
# 전체 SoccerPlayer 코드
class SoccerPlayer(object):
    def __init__(self, name, position, back_number):
        self.name = name
        self.position = position
        self.back_number = back_number

     def change_back_number(self, new_number):
        print("선수의 등번호를 변경한다: From %d to %d" % (self.back_number, new_number))
        self.back_number = new_number

   def __str__(self):
        return "Hello, My name is %s. I play in %s in center." % (self.name, self.position)

 # SoccerPlayer를 사용하는 instance 코드
hong = SoccerPlayer("홍길동", "MF", 10)

print("현재 선수의 등번호는:", hong.back_number)
hong.change_back_number(5)
print("현재 선수의 등번호는:", hong.back_number)
```

클래스와 객체 지향(2/2)

상속

```python
class Person:
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender
    def about_me(self):
        print("저의 이름은", self.name, "이고요, 제 나이는", str(self.age), "살입니다.")

 class Employee(Person):
    def __init__(self, name, age, gender, salary, hire_date):
        super().__init__(name, age, gender) # 부모 객체 사용
        self.salary = salary
        self.hire_date = hire_date
    def do_work(self):
        print("열심히 일을 한다.")
    def about_me(self):
        super().about_me()
        print("제 급여는", self.salary, "원이고, 제 입사일은", self.hire_date, "입니다.")
```

다형성

```
class Animal:
    def __init__(self, name):
        self.name = name
    def talk(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Cat(Animal):
    def talk(self):
        return 'Meow!'

class Dog(Animal):
    def talk(self):
        return 'Woof! Woof!'
animals = [Cat('Missy'), Cat('Mr. Mistoffelees'), Dog('Lassie')]

for animal in animals:
    print(animal.name + ': ' + animal.talk())
```