



CES1017 Programming Fundamentals

# Inheritance and Polymorphism

Instructor: Eunil Park (pa1324@hanyang.ac.kr)



**HANYANG UNIVERSITY**



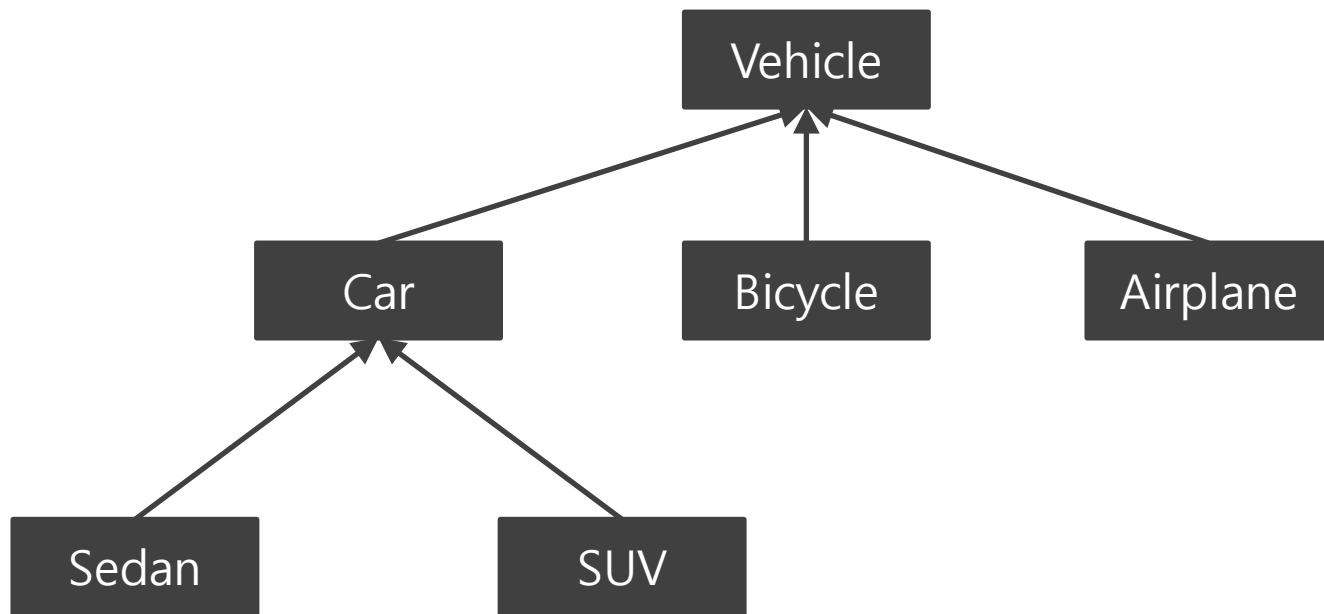
## 학습 목표

1. 객체지향 프로그래밍 언어의 상속 이해하기
2. 객체지향 프로그래밍 언어의 다형성 이해하기
3. 파이썬의 상속과 다형성
4. 메소드 오버라이딩과 오버로딩의 차이 이해하기

# 01. 상속 (Inheritance)

## 상속이란?

- 상속
  - 기존의 클래스(부모 클래스 혹은 상위 클래스)를 확장하여 새로운 클래스(자식 클래스 혹은 하위 클래스)를 정의하는 효율적인 메카니즘



# 01. 상속 (Inheritance)

## 상속 선언

- 파이썬에서 상속 선언하기

**class** 자식클래스(부모클래스):

생성자

메소드



# 01. 상속 (Inheritance)

## inheritance-ex1.py

```
class Vehicle:
    def __init__(self, name, color):
        self.__name = name
        self.__color = color
    def getColor(self):          # getColor() 가 Car에 상속
        return self.__color
    def setColor(self, color):   # setColor가 Car에 상속
        self.__color = color
    def getName(self):           # getName() 가 Car에 상속
        return self.__name

class Car(Vehicle):
    def __init__(self, name, color, model):
        # name과 color를 초기화하기 위해 부모 생성자 호출
        super().__init__(name, color)
        self.__model = model
    def getDescription(self):
        return self.getName()+" "+self.__model+" "+self.getColor()+" color"

c = Car("Ford Mustang", "red", "GT350")
print(c.getDescription())
print(c.getName())
```



# 01. 상속 (Inheritance)

## 상속 예제

- 프로그램 분석
  - Vehicle 클래스
    - private 변수로 \_\_name과 \_\_color 선언하고 생성자로 그 값을 초기화함
    - 파이썬에서는 클래스 변수나 메소드 앞에 \_\_ 를 붙어 private 변수나 메소드로 선언 가능
    - private 변수나 메소드는 클래스 안에서만 접근이 가능한 변수나 메소드로 클래스 외부에서 접근이 불가
    - 생성자 이외에 getColor(), setColor(), getName() 메소드를 정의
    - 메소드들은 자식 클래스인 Car에 상속됨

# 01. 상속 (Inheritance)

## 상속 예제

- 프로그램 분석
  - Car 클래스
    - 생성자에서 name과 color는 부모 클래스의 생성자를 호출하여 초기화하고 model은 Car 클래스 자체에서 초기화함
      - `super()`는 부모클래스를 가리킴
      - Car 클래스는 `__name`, `__color` 이외에 `__model` 속성을 더 갖음
    - Car 클래스는 `getColor()`, `setColor()`, `getName()` 메소드를 부모 클래스로부터 상속 받음
      - 마치 자신이 가지고 있는 메소드인 것처럼 사용할 수 있음
      - 마지막 줄, `print(c.getName())`을 보면 실제로 Car 클래스엔 `getName()` 메소드가 없지만 호출이 가능함
    - Car 클래스는 `getDescription()` 메소드도 추가적으로 정의

# 01. 상속 (Inheritance)

## 상속 예제

- 프로그램 실행

```
>>>
RESTART:
C:/Users/clairlsy/AppData/Local/Programs/Python/Python36-
32/inheritance-ex1.py
Ford MustangGT350 in red color
Ford Mustang
>>>
```

## 02. 다중 상속(multiple inheritance)



### 다중 상속

- 파이썬에서 다중상속 선언하기

**class** 자식클래스(부모클래스1, 부모클래스2, ...):

생성자

메소드

## 02. 다중 상속(multiple inheritance)



HANYANG  
UNIVERSITY

### 다중 상속 예시

#### inheritance-ex2.py

```
class MySuperClass1():

    def method_super1(self):
        print("method_super1 method called")

class MySuperClass2():

    def method_super2(self):
        print("method_super2 method called")

class ChildClass(MySuperClass1, MySuperClass2):

    def child_method(self):
        print("child method")

c = ChildClass()
c.method_super1()
c.method_super2()
```

# 02. 다중 상속(multiple inheritance)

## 다중 상속

- 프로그램 분석
  - MySuperClass1 클래스
    - 간단한 메시지("method\_super1 method called")를 출력하는 method\_super1() 메소드 정의
  - MySuperClass2 클래스
    - 간단한 메시지("method\_super2 method called")를 출력하는 method\_super2() 메소드 정의
  - ChildClass 클래스
    - 부모 클래스로 MySuperClass1과 MySuperClass2가 선언됨
    - 따라서 양쪽 부모 클래스가 가지고 있는 모든 메소드를 그대로 상속 받음
    - 상속받는 메소드 이외에 자체 메소드로 child\_method()를 정의함

## 02. 다중 상속(multiple inheritance)



### 다중 상속

```
>>>
RESTART:
C:/Users/clairlsy/AppData/Local/Programs/Python/Python36-
32/inheritance-ex2.py
method_super1 method called
method_super2 method called
```

다중 상속의 경우에는 동일한 이름의 메소드가 부모 클래스에 모두 존재할 경우 어떤 부모 클래스의 메소드를 상속 받을까?

# 02. 다중 상속(multiple inheritance)



## 다중 상속

- 아래 코드를 실행해 보고 분석해 보시오.

```
>>> class ParentOne:  
    def func(self):  
        print("ParentOne의 함수 호출!")  
  
>>> class ParentTwo:  
    def func(self):  
        print("ParentTwo의 함수 호출!")  
  
>>> class Child(ParentOne, ParentTwo):  
    def childFunc(self):  
        ParentOne.func(self)  
        ParentTwo.func(self)  
  
>>> objectChild = Child()  
>>> objectChild.childFunc()  
ParentOne의 함수 호출!  
ParentTwo의 함수 호출!  
>>> objectChild.func()  
ParentOne의 함수 호출!
```

## 02. 다중 상속(multiple inheritance)



```
>>> class A:  
    def __init__(self):  
        print("A 클래스의 생성자 호출!")  
  
>>> class B(A):  
    def __init__(self):  
        A.__init__(self)  
        print("B 클래스의 생성자 호출!")  
  
>>> class C(A):  
    def __init__(self):  
        A.__init__(self)  
        print("C 클래스의 생성자 호출!")  
  
>>> class D(B, C):  
    def __init__(self):  
        B.__init__(self)  
        C.__init__(self)  
        print("D 클래스의 생성자 호출!")  
  
>>> objectD = D()  
A 클래스의 생성자 호출!  
B 클래스의 생성자 호출!  
A 클래스의 생성자 호출!  
C 클래스의 생성자 호출!  
D 클래스의 생성자 호출!
```

# 03. 다형성 (polymorphism)

## 오버로딩이란?

- 동일한 연산자(혹은 메소드)가 서로 다른 종류의 자료형을 처리하는 경우

```
>>> 3 + 4
7
>>> 3.0 + 4.0
7.0
>>> "abc" + "def"
'abcdef'
>>>
```

- 정수, 실수, 문자열에 대해서 + 연산자가 동작
- 결국, 동일한 연산자가 자료형에 따라 다른 형태로 사용될 수 있다  
→ 다형성
- 오버로딩은 **동일한 클래스나 하위 클래스** 안에서 동일한 이름의 연산자(혹은 메소드)를 사용할 수 있다.

# 03. 다형성 (polymorphism)

## 오버라이딩이란?

- 부모 클래스에서 정의된 연산자(혹은 메소드)를 하위 클래스에서 동일한 이름으로 재정의해 사용하는 경우
- 일반적으로 부모 클래스에서 정의된 메소드를 상속 받아서 동일한 메소드 이름으로 기능을 확장하여 사용함  
→ 메소드 향상(method improvement)
- 오버라이딩은 부모·자식 클래스 간에 사용됨  
(오버로딩은 동일 클래스 안에서도 같은 이름 사용 가능)



# 03. 다형성 (polymorphism)

## 오버라이딩

### overriding-ex.py

```
class Student:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def speak(self):  
        print("I am a student !")  
  
class CS_Student(Student):  
    def __init__(self, name, age, major):  
        super().__init__(name, age)  
        self.major = major  
  
    def speak(self): #부모 클래스의 speak() 메소드를 상속 받아 재정의  
        print("I am a ", self.major, "student !")  
  
s = CS_Student("Alex", 20, "CS")  
s.speak()
```



# 03. 다형성 (polymorphism)

## 오버라이딩

- 프로그램 분석
  - CS\_Student 클래스는 Student 클래스의 자식 클래스
  - Student 클래스와 CS\_Student 클래스에 모두 동일한 이름의 메소드 speak()가 정의되어 있음
  - CS\_Student 클래스에 있는 speak()는 Student 클래스에 존재하는 동일한 이름의 메소드를 재정의한 것임 → 메소드 오버라이딩
- 실행 결과

```
>>>
RESTART:
C:/Users/clairlsy/AppData/Local/Programs/Python/Python36-
32/overriding-ex.py
I am a  CS student !
>>>
```

# Thanks

Instructor: Eunil Park(pa1324@hanyang.ac.kr)