파이썬 프로그래밍

상속과 다형성

한국기술교육대학교
온라인평생교육원

▣ 클래스 상속

1. 클래스 상속과 이름 공간의 관계

- 상속의 이유
  - 코드의 재사용
  - 상속받은 자식 클래스는 상속을 해준 부모 클래스의 모든 기능을 그대로 사용
  - 자식 클래스는 필요한 기능만을 정의하거나 기존의 기능을 변경할 수 있음

```
class Person:
    def __init__(self, name, phone=None):
        self.name = name
        self.phone = phone
    def __str__(self):
        return '<Person %s %s>' % (self.name, self.phone)
```

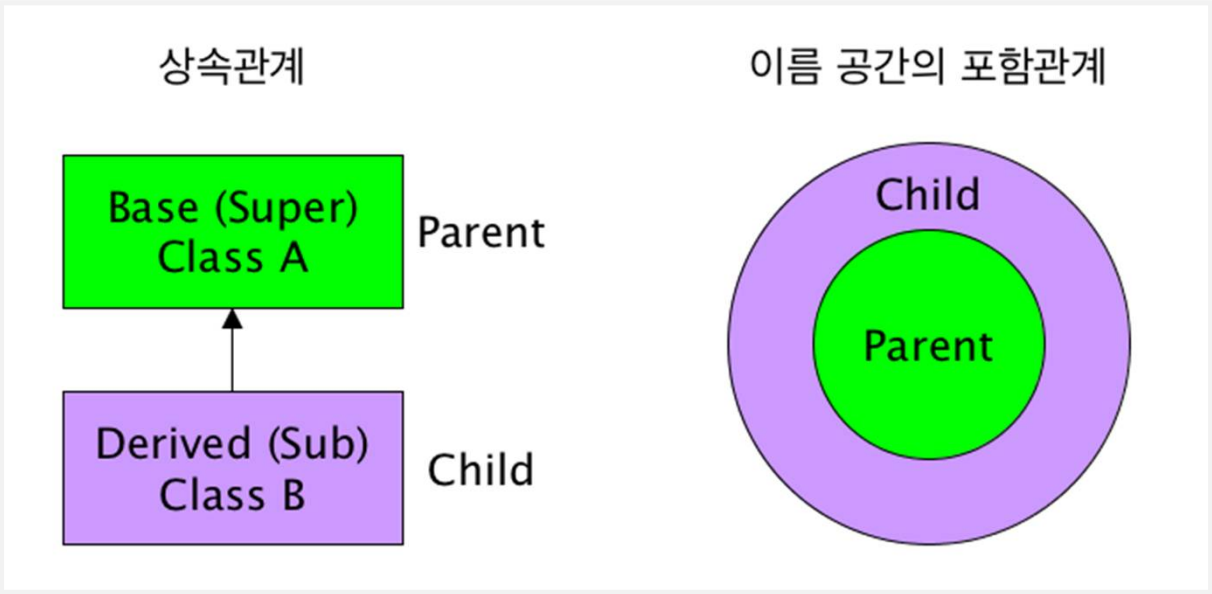
```
class Employee(Person):                    # 괄호 안에 쓰여진 클래스는 슈퍼클래스를 의미한다.
    def __init__(self, name, phone, position, salary):
        Person.__init__(self, name, phone) # Person클래스의 생성자 호출
        self.position = position
        self.salary = salary
```

- person의 내용이 그대로 Employee에서 활용 가능
- self → 인스턴스 메소드가 기본적으로 가져가는 인자
- %s %s → 문자열 포맷팅
- class 이름 (상속 받으려는 class 이름)
- __init__를 사용하여 생성자 정의 → 필요한 기능만을 새로 정의

▣ 클래스 상속

1. 클래스 상속과 이름 공간의 관계

- 이름 공간의 포함관계
  - 자식 클래스 > 부모 클래스



- Person = 부모 클래스
- Employee = 자식 클래스
- Derived : 유도가 되어진, 상속되어진
- 자식 클래스 > 부모 클래스
- 자식 클래스가 부모 클래스를 가지며 더 많은 식별자를 가짐

▣ 클래스 상속

1. 클래스 상속과 이름 공간의 관계

```
p1 = Person('홍길동', 1498)
print p1.name
print p1

print

m1 = Employee('손창희', 5564, '대리', 200)
m2 = Employee('김기동', 8546, '과장', 300)
print m1.name, m1.position  # 슈퍼클래스와 서브클래스의 멤버를 하나씩 출력한다.
print m1
print m2.name, m2.position
print m2
```

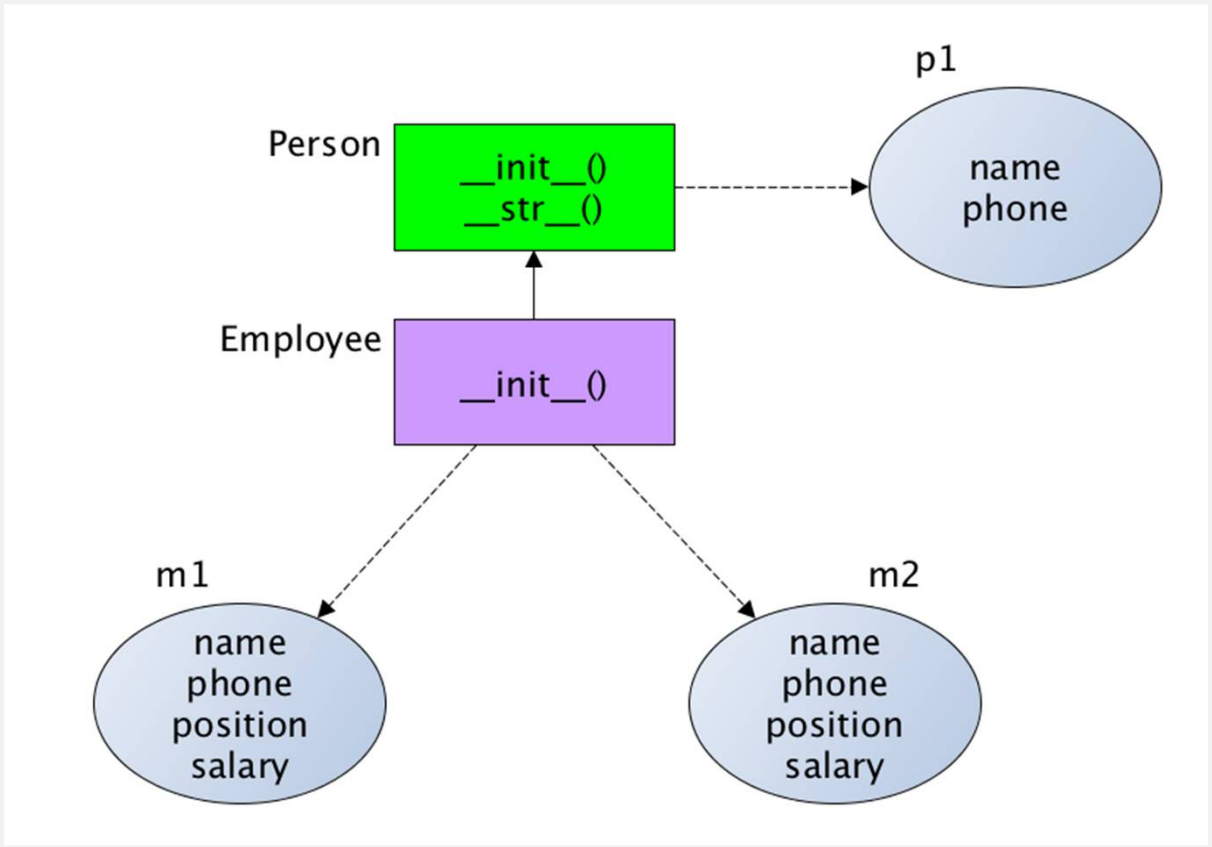
```
홍길동
<Person 홍길동 1498>

손창희 대리
<Person 손창희 5564>
김기동 과장
<Person 김기동 8546>
```

- p1 → __str__ 메소드 호출
- m1 은 employee 객체이기 때문에 그 안에서 __str__ 찾음
- 없으면 자연스럽게 부모 클래스에 가서 __str__찾음

▣ 클래스 상속

1. 클래스 상속과 이름 공간의 관계



- 자식 클래스에 있는 init이 부모 클래스의 init을 overiding 함
- override: 재정의

▣ 클래스 상속

2. 생성자 호출

- 서브 클래스의 생성자는 슈퍼 클래스의 생성자를 자동으로 호출하지 않는다.

```
class Super:
    def __init__(self):
        print 'Super init called'

class Sub(Super):
    def __init__(self):
        print 'Sub init called'

s = Sub()
```

```
Sub init called
```

- 자식 클래스에 있는 init이 부모 클래스의 init을 overiding 함

▣ 클래스 상속

2. 생성자 호출

- 서브 클래스의 생성자에서 슈퍼 클래스의 생성자를 명시적으로 호출해야 한다.

```
class Super:
    def __init__(self):
        print 'Super init called'

class Sub(Super):
    def __init__(self):
        Super.__init__(self)   # 명시적으로 슈퍼클래스의 생성자를 호출한다.
        print 'Sub init called'

s = Sub()
```

```
Super init called
Sub init called
```

- sub 클래스의 생성자가 호출되자마자 super 클래스의 init 호출
- 명시적으로 super 클래스의 생성자 호출

▣ 클래스 상속

2. 생성자 호출

- 서브 클래스에 생성자가 정의되어 있지 않은 경우에는 슈퍼 클래스의 생성자가 호출된다.

```
class Super:
    def __init__(self):
        print 'Super init called'

class Sub(Super):
    pass

s = Sub()
```

```
Super init called
```

- 내용이 없다면 무조건 pass 써야 함
- sub 클래스의 init이 없으면 super 클래스의 init 만 호출

▣ 클래스 상속

3. 메소드의 대치(메소드 오버라이드 - Override)

- 서브 클래스에서 슈퍼 클래스에 정의된 메소드를 재정의하여 대치하는 기능

```
class Person:
    def __init__(self, name, phone=None):
        self.name = name
        self.phone = phone
    def __str__(self):
        return '<Person %s %s>' % (self.name, self.phone)

class Employee(Person):
    def __init__(self, name, phone, position, salary):
        Person.__init__(self, name, phone)
        self.position = position
        self.salary = salary

p1 = Person('gslee', 5284)
m1 = Employee('kslee', 5224, 'President', 500)

print p1
print m1
```

```
<Person gslee 5284>
<Person kslee 5224>
```

- m1도 str을 호출하나 Employee 클래스 안에 없어 부모 클래스 활용
- Employee도 str을 가지게 되면 부모 클래스의 str 위로 올라탐
- 부모 클래스의 str은 무시, 자식 클래스의 str을 호출

▣ 클래스 상속

3. 메소드의 대치(메소드 오버라이드 - Override)

```
class Employee(Person):
    def __init__(self, name, phone, position, salary):
        Person.__init__(self, name, phone)
        self.position = position
        self.salary = salary
    def __str__(self):
        return '<Employee %s %s %s %s>' % (self.name, self.phone, self.position,
                    self.salary)

p1 = Person('gslee', 5284)
m1 = Employee('kslee', 5224, 'President', 500)

print p1
print m1
```

```
<Person gslee 5284>
<Employee kslee 5224 President 500>
```

- m1도 str을 호출하나 Employee 클래스 안에 없어 부모 클래스 활용
- Employee도 str을 가지게 되면 부모 클래스의 str 위로 올라탐
- 부모 클래스의 str은 무시, 자식 클래스의 str을 호출

▣ 클래스 상속

4. 다형성(Polymorphism)

• 상속 관계 내의 다른 클래스들의 인스턴스들이 같은 멤버 함수 호출에 대해 각각 다르게 반응하도록 하는 기능
- 연산자 오버로딩도 다형성을 지원하는 중요한 기술
* 예를 들어, a와 b의 객체 형에 따라 a + b의 + 연산자 행동 방식이 변경되는 것

• 다형성의 장점
- 적은 코딩으로 다양한 객체들에게 유사한 작업을 수행시킬 수 있음
- 프로그램 작성 코드 량이 줄어든다.
- 코드의 가독성을 높혀준다.

• 파이썬에서 다형성의 장점
- 형 선언이 없다는 점에서 파이썬에서는 다형성을 적용하기가 더욱 용이하다.
- 실시간으로 객체의 형이 결정되므로 단 하나의 메소드에 의해 처리될 수 있는 객체의 종류에 제한이 없다.
* 즉, 다른 언어보다 코드의 양이 더욱 줄어든다.

- a 에 들어오는 형태에 따라서 + 연산자가 다르게 행동
- a에 클래스가 들어오면 이에 대응되는 __add__ 메소드가 호출
- 파이썬은 디폴트로 다형성이 잘 제공되고 있음

▣ 클래스 상속

4. 다형성(Polymorphism)

```
class Animal:
    def cry(self):
        print '...'

class Dog(Animal):
    def cry(self):
        print '멍멍'

class Duck(Animal):
    def cry(self):
        print '꽥꽥'

class Fish(Animal):
    pass

for each in (Dog(), Duck(), Fish()):
    each.cry()
```

```
멍멍
꽥꽥
...
```

- Animal에 있는 cry를 Dog의 cry가 override 함
- Fish는 cry가 없으므로 Animal의 내용을 그대로 사용
- in 뒤에는 튜플 안에 객체 3개 존재
- each의 객체 형이 동적으로 결정되므로 그때마다의 cry가 다르게 사용