

# 파이썬

## 22강. 객체지향 기법

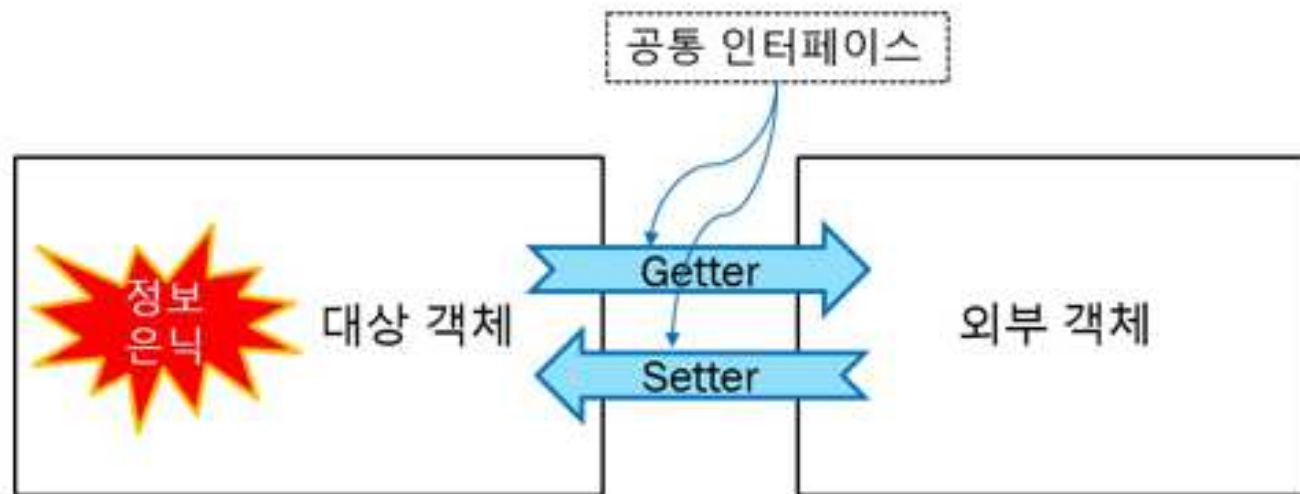
## 1. 객체지향 기법

- 클래스와 객체는 객체지향프로그래밍에서 나오는 용어이다. 따라서 대표적인 객체지향 기법으로 알려진 캡슐화, 상속, 재정의 그리고 다형성 등의 개념을 살펴보고, 파이썬의 클래스에서는 이들이 어떤 방식으로 설계되고 객체화되는지를 알아본다.

## 2. 캡슐화

- 자료와 알고리즘이 구현된 함수를 하나로 묶고 공용 인터페이스만 으로 접근을 제한하여 객체의 세부 내용을 외부로 부터 감추는 기법을 캡슐화(encapsulation)라고 한다. 클래스에서 은닉정보는 변수 앞 부분에 \_기호를 연속하여 두 개 넣으면 외부에서 접근이 불가능한 은닉(private) 변수가 된다. 그
- 리고 은닉 변수를 외부에서 접근할 수 있는 공용 인터페이스는 획득자(getter)와 지정자(setter)로 분류한다.
- 획득자는 외부에서 은닉된 값을 꺼내오는 메서드이고, 지정자는 외부에서 값을 수정하는 메서드이다.
- 클래스의 정보은닉과 공용 인터페이스의 관계를 나타내고 있다.

## 2. 캡슐화



## 2. 캡슐화

chapter06.lecture.step04\_encapsulation.py ↵

Python Console ↵

```
class Account:↵  
    # (1) 은닉 멤버변수↵  
    __balance = 0 # 잔액↵  
    __accName = None # 예금주↵  
    __accNo = None # 계좌번호↵  
  
    # (2) 생성자 : 멤버변수 초기화↵  
    def __init__(self, bal, name, no):  
        self.__balance = bal # 잔액 초기화↵
```

## 2. 캡슐화

```
        self.__accName = name # 예금주↵
        self.__accNo = no # 계좌번호↵

# (3) 계좌정보 확인 : Getter
def getBalance(self):↵
    return self.__balance, self.__accName, self.__accNo↵

# (4) 입금하기 : Setter↵
def deposit(self, money):
    if money < 0:↵
        print('금액 확인')
        return # 종료(exit)↵
    self.__balance += money↵

# (5) 출금하기 : Setter↵
def withdraw(self, money):
    if self.balance < money:↵
        print('잔액 부족')
        return # 종료(exit)↵
    self.__balance -= money↵
```

## 2. 캡슐화

```
# (6) object 생성 ↵  
acc = Account(1000, '홍길동', '125-152-4125-41'  
) # 생성자 ↵
```

```
# (7) Getter 호출 ↵  
acc.__balance # 오류(Error) ↵  
bal = acc.getBalance()  
print('계좌정보 : ', bal) ↵
```

```
# (8) Setter 호출 ↵  
acc.deposit(10000) # 10,000원 입금 ↵  
bal = acc.getBalance()  
print('계좌정보 : ', bal) # 입금 확인 ↵
```

계좌정보 : (1000, '홍길동', '125-152-4125-41') ↵

계좌정보 : (21000, '홍길동', '125-152-4125-41') ↵

## 2. 캡슐화

- 캡슐화 예
- # (1) 은닉 멤버변수
- 잔액(balance), 예금주(accName), 계좌번호(accNo) 멤버변수는 모두 '변수명' 형태로 선언되었기 때문에 외부에서 직접 접근이 불가능한 은닉 멤버변수이다.
- # (2) 생성자 : 멤버변수 초기화
- 생성자는 객체 생성 시점에서 매개변수로 넘겨받은 실인수를 은닉 멤버변수에 초기화는 가능하다.
- # (3) 계좌정보 확인 : Getter
- 은닉 멤버변수의 값을 외부에서 받을 수 있는 획득자 메서드이다.
- # (4) 입금하기 : Setter
- 은닉 멤버변수 중에서 잔액( balance) 변수에 입금액(money)을 더해서 잔액을 수정하는 메서드이다. 외부에서 은닉 멤버변수의 값을 수정하는 역할을 하므로 deposit은 지정자 메서드가 된다. 입금액이 음수이면 메서드는 실행을 종료한다.



## 2. 캡슐화

- # (5) 출금하기 : Setter
- 은닉 멤버변수 중에서 잔액( balance) 변수에 입금액(money)을 빼서 잔액을 수정하는 메서드이다. 외부에서 은닉 멤버변수의 값을 수정하는 역할을 하므로 withdraw는 지정자 메서드가 된다. 출금액이 잔액보다 더 많으면 메서드는 실행을 종료한다.
- # (6) object 생성
- Account 클래스의 생성자와 잔액, 예금주, 계좌번호의 실인수를 이용하여 '홍길동'의 계좌를 생성한다.
- # (7) Getter 호출
- 객체의 참조변수를 이용하여 계좌정보를 확인하는 getBalance() 메서드를 호출하여 은닉 멤버변수의 값을 확인한다. 은닉 멤버변수는 '참조변수.멤버변수' 형태로 접근은 불가능하고, 획득자나 지정자 메서드에 의해서만 접근할 수 있다.
- # (8) Setter 호출
- 계좌에 입금하는 deposit() 메서드를 호출하여 10,000원을 입금하고, getBalance() 메서드를 호출하여 입금된 잔액을 확인한다.

### 3. 상속

- 클래스의 상속(inheritance)은 캡슐화와 더불어서 객체지향프로그래밍의 중요한 특징 중의 하나이다. 클래스 간의 계층적 관계를 구성하여 높은 수준의 코드 재사용성과 다형성의 문법적 토대를 마련하였다.
- 다음 부모클래스를 상속받아서 2개의 자식클래스를 상속받는 클래스 다이어그램이 다. 일반적으로 부모클래스는 자식클래스에서 공통으로 사용할 수 있는 멤버를 선언하여 클래스를 정의한다. 예를 들면 사원클래스는 정규직이든 임시직이든 모두 사원명과 부서명이 있고, 또한 급여를 받는다.
- 물론 각 변수에 들어갈 값은 서로 다르다. 이렇게 잘 설계된 부모클래스를 정의해 놓고 필요한 경우 상속을 통해서 새로운 자식클래스를 만들 수 있기 때문에 사용자에게 높은 수준의 코드 재활용성을 제공한다.

### 3. 상속



### 3. 클래스의 상속

chapter06.lecture.step05\_inheritance.py ↵

Python Console ↵

```
# (1) 부모 클래스 ↵
class Super : ↵
    # 생성자 : 동적멤버 생성 ↵
    def __init__(self, name, age):
        self.name = name ↵
        self.age = age ↵

    # 메서드 ↵
    def display(self): ↵
        print('name : %s, age : %d'%(self.name, se
lf.age)) ↵
```

### 3. 클래스의 상속

```
sup = Super('부모', 55)
sup.display() # 부모 멤버 호출

# (2) 자식 클래스
class Sub(Super) : # 클래스 상속
    gender = None # 자식 멤버

# (3) 생성자
def __init__(self, name, age, gender):
    self.name = name
    self.age = age
    self.gender = gender

# (4) 메서드 확장
def display(self):
    print('name : %s, age : %d, gender : %s'
          %(self.name, self.age, self.gende
r))

sub = Sub('자식', 25, '여자')
sub.display() # 자식 멤버 호출
```

---

### 3. 상속

- 해설 상속 예
- # (1) 부모 클래스
- Super 클래스는 생성자에 의해서 동적 멤버변수 name과 age가 만들어진다. 그리고 display()
- 메서드는 동적 멤버변수를 출력하는 역할을 한다.
- # (2) 자식 클래스
- Sub 클래스는 Super 클래스를 상속받아서 만들어졌다. 따라서 부모 클래스의 멤버변수(name, age) 와 메서드(display)가 자식 클래스로 상속된다. 자식 클래스에서는 필요하면 멤버변수와 메서드를 새롭게 추가할 수 있다. gender는 자식 클래스에서 추가한 멤버변수이다.
- # (3) 생성자
- 결국 자식 클래스의 멤버변수는 부모 클래스의 멤버변수 2개와 자식 클래스에서 추가한 멤버변수 1개를 더해서 전체 3개의 멤버변수가 된다. 생성자에서는 이들 멤버변수를 초기화하여 객체가 생성되도록 정의되어 있다.

### 3. 상속

- # (4) 메서드 확장
- 부모 클래스에서 상속받은 메서드는 필요에 따라서 자식 클래스에서 내용을 수정하거나 확장시킬 수 있는데, 이러한 기법을 메서드 재정의 (method override)라고 한다. 원래 부모에서 상속받은 display는 name 과 age 변수만 출력되도록 되어 있다. 따라서 자식 클래스에서 새롭게 추가한 gender 변수를 출력할 수 없기 때문에 상속받은 display() 메서드를 확장하여 3개의 멤버변수가 모두 출력되도록 재정의 하였다.

## 4. super 클래스

- 자식 클래스에서 부모 클래스의 생성자를 호출하기 위해서 super 클래스를 제공한다. super 클래스의 생성자('super()')를 이용해서 부모 클래스의 생성자를 호출하는 형식은 다음과 같다.

```
super().__init() # 부모 클래스 생성자 호출 ↵
```

- 클래스의 상속 과정에서 부모 클래스의 생성자는 상속 대상이 아니라고 하였다. 따라서 자식 클래스에서 부모 클래스의 생성자가 필요할 경우에는 super().init() 형식의 명령문으로 호출할 수 있다.



## 4. super 클래스

chapter06.lecture.step05\_inheritance.py ↵

Python Console ↵

```
# (1) 부모 클래스 ↵
class Parent : ↵

    # 생성자 : 객체 + 초기화 ↵
    def __init__(self, name, job):
        self.name = name ↵
        self.job = job ↵

    # 멤버 함수 (method)
    def display(self): ↵
        print('name : {}, job : {}'.format(self.
name, self.job)) ↵

# 부모 클래스 객체 생성 ↵
p = Parent('홍길동', '회사원') ↵
p.display() ↵

# (2) 자식 클래스 ↵
class Children(Parent): # Parent 클래스 상속 ↵
    gender = None # 자식 클래스 멤버변수 추가 ↵

    # (3) 자식 클래스 생성자 ↵
    def __init__(self, name, job, gender):
        # 부모 클래스 생성자 호출 ↵
        super().__init__(name, job) # name, job ↵
        self.gender = gender ↵
```

↵  
↵  
↵  
name : 홍길동, job : 회사  
원 ↵

## 4. super 클래스

chapter06.lecture.step05\_inheritance.py ↵

Python Console ↵

```
# (1) 부모 클래스 ↵  
class Parent : ↵
```

```
    # 생성자 : 객체 + 초기화 ↵
```

```
    def __init__(self, name, job):  
        self.name = name ↵  
        self.job = job ↵
```

```
    # 멤버 함수 (method)
```

```
    def display(self): ↵  
        print('name : {}, job : {}'.format(self.  
name, self.job)) ↵
```

```
# 부모 클래스 객체 생성 ↵
```

```
p = Parent('홍길동', '회사원') ↵  
p.display() ↵
```

```
↵  
↵  
↵
```

```
name : 홍길동, job : 회사  
원 ↵
```

```
# (2) 자식 클래스 ↵
```

```
class Children(Parent): # Parent 클래스 상속 ↵  
    gender = None # 자식 클래스 멤버변수 추가 ↵
```

```
    # (3) 자식 클래스 생성자 ↵
```

```
    def __init__(self, name, job, gender):  
        # 부모 클래스 생성자 호출 ↵  
        super().__init__(name, job) # name, job ↵
```

## 4. super 클래스

- 해설 super 예
- # (1) 부모 클래스
- Parent 클래스는 생성자에 의해서 동적 멤버변수 name과 job이 만들어진다. 매개변수에 의해서 값이 초기화 된다. 그리고 display() 메서드는 동적 멤버변수를 출력하는 역할을 한다.
- # (2) 자식 클래스
- Children 클래스는 Parent 클래스를 상속받아서 만들어졌다. 따라서 부모 클래스의 멤버변수 (name, job)와 메서드(display)가 자식 클래스로 상속된다. 또한 자식 클래스에서는 멤버변수 gender를 추가하였다.
- # (3) 자식 클래스 생성자
- 자식 클래스는 전체 3개의 멤버변수를 가지고 있는데, name과 job은 부모 클래스로부터 상속받은 멤버변수이다. 따라서 다음과 같은 명령문으로 부모 클래스의 생성자를 호출해서 상속받은 멤버변수를 초기화할 수 있다.
- super(). init (name, job)
- 물론 gender는 자식 클래스의 멤버변수이기 때문에 매개변수 gender를 이용해서 직접 초기화해야한다.

## 5. 메서드 재정의

- 부모 클래스의 멤버변수가 자식 클래스로 상속되는 것처럼 부모 클래스가 가지고 있는 메서드도 자식 클래스로 상속된다. 또한 상속받은 메서드는 자식 클래스에서 다시 작성해서 사용할 수 있다. 이러한 기법을 메서드 재정의(Method Overriding)라고 한다



## 5. 메서드 재정의

chapter06.lecture.step06\_override.py ↗

Python Console ↗

```
# (1) 부모클래스 ↗
class Employee:
    name = None
    pay = 0 ↗

    def __init__(self, name):
        self.name = name ↗

    def pay_calc(self):
        pass ↗

# (2) 자식클래스 : 정규직 ↗
class Permanent(Employee):
    def __init__(self, name): ↗
        super().__init__(name) # 부모 생성자 호출 ↗

    def pay_calc(self, base, bonus): ↗ ↗
        self.pay = base + bonus # 급여=기본급+상여금 ↗
        print('총 수령액 : ', format(self.pay, '3d'), '원', end='') ↗
        print(' 총 수령액 : 3,200,000 원' ↗
d'), '원') ↗
```

## 5. 메서드 재정의

```
# (3) 자식클래스 : 임시직 ↵
class Temporary(Employee):
    def __init__(self, name): ↵
        super().__init__(name) # 부모 생성자 호출 ↵

    def pay_calc(self, tpay, time): ↵
        self.pay = tpay * time # 급여=작업시간*시급 ↵
        print('총 수령액 : ', format(self.pay, '3,000,000'), '원 ↵
d'), '원') ↵

# (4) 객체 생성 ↵
p = Permanent("이순신") ↵
p.pay_calc(3000000, 200000) ↵

t = Temporary("홍길동")
t.pay_calc(15000, 80) ↵
```

---

## 5. 메서드 재정의

- # (1) 부모 클래스
- Employee 클래스는 두 개의 멤버변수(name, pay)와 하나의 메서드 (pay\_calc)를 갖는다. 그리고 생성자를 이용하여 사원의 이름을 초기화 하고 있다.
- # (2) 자식클래스 : 정규직
- Permanent 클래스는 Employee 클래스를 상속받아서 만들어졌다. 따라서 부모 클래스의 멤버변수 (name, pay)와 메서드(pay\_calc)를 상속받는다. 그리고 상속 받은 급여계산 메서드의 내용을 정규 직 급여 계산 방식으로 수정하여 메서드를 재정의 하였다.
- # (3) 자식클래스 : 임시직
- Temporary 클래스는 Employee 클래스를 상속받아서 만들어졌다. 따라서 부모 클래스의 멤버변수 (name, pay)와 메서드(pay\_calc)를 상속받는다. 그리고 상속 받은 급여계산 메서드의 내용을 임시 직 급여 계산 방식으로 수정하여 메서드를 재정의 하였다.

## 5. 메서드 재정의

- # (4) 객체 생성
- Permanent와 Temporary 클래스의 객체를 각각 생성한 후 객체의 참조 변수를 이용하여 pay\_calc()
- 메서드를 호출한다. 메서드 호출 결과는 총 수령액이다.



THANK YOU