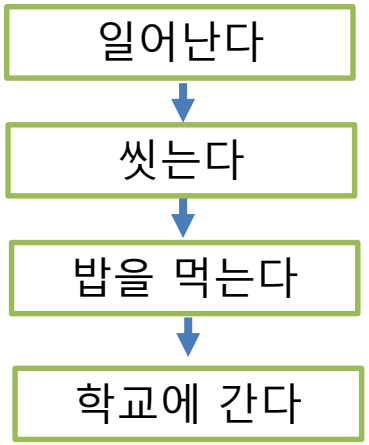
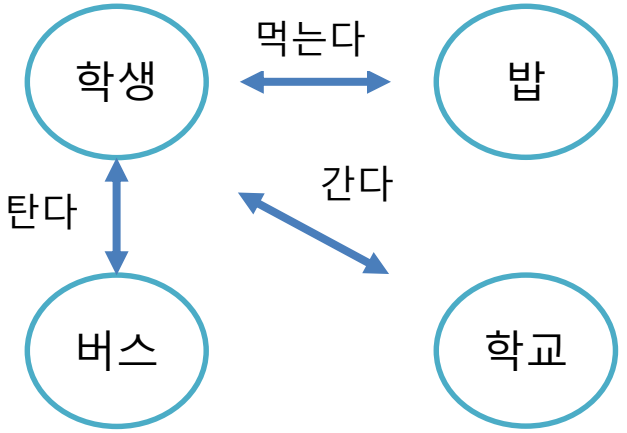
7장. 클래스와 상속

객체 지향 프로그래밍

- **객체(Object)란?**
  - "의사나 행위가 미치는 대상" -> 사전적 의미
  - 구체적, 추상적 데이터 단위 (구체적- 책상, 추상적-회사)
- **객체지향 프로그래밍(Objected Oriented Programming)**
  - 객체를 기반으로 하는 프로그래밍
  - 먼저 객체를 만들고 객체 사이에 일어나는 일을 구현함.



<절차지향 –C언어>



<객체지향 – Python>

클래스(class)

- **클래스란?**
  - ✓ **객체에 대한 속성과 기능을 코드로 구현** 한 것
  - ✓ 객체에 대한 설계도 또는 청사진.
- **객체의 속성과 기능**
  - ✓ 객체의 특성(property), 속성(attribute) -> **멤버 변수**
  - ✓ 객체가 하는 기능 -> **멤버 함수**

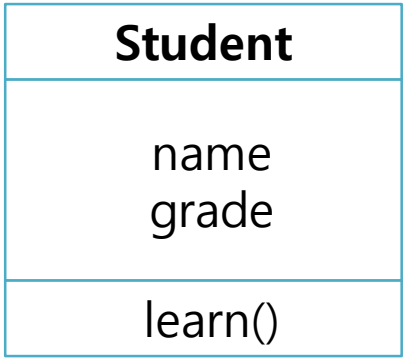
```
class 클래스 이름 :
    def __init__(self):
        멤버변수

    def 함수이름(self):
        return
```

학생 클래스

- 속성(멤버변수) : 학번, 이름, 학년, 사는 곳 등..
- 기능(함수) : 수강신청, 수업듣기, 시험 보기 등..

클래스(class) 정의

- **학생 클래스 정의 및 사용**



객체(인스턴스) = 클래스()

객체.멤버변수
객체.멤버 메서드

객체를 생성한 후 점(.) 연산자를 사용하여 멤버변수나 메서드에 접근함

```
class Student:
    name = "김하나"
    grade = 5

st1 = Student()
print(st1.name, st1.grade)
```

클래스(class) 정의

- **생성자(constructor) – 기본 생성자**
  - 클래스를 생성할 때 호출되는 명령어 집합, 초기자라고도 한다.
  - 생성자는 **__init__()**의 형태로 작성하고, 리턴값이 없다
  - 클래스 내의 모든 함수(메서드)의 매개변수에 **self**를 넣어줌

```
class Student:
    def __init__(self):
        self.name = "콩쥐"
        self.grade = 1
        print("생성자")

    def learn(self):
        print("수업을 듣습니다.")

s = Student() # 객체 생성
print(s)  # 객체 출력
print(type(s)) # 자료형 : 클래스
print(s.name, "학생은", s.grade, "학년입니다.")
s.learn()
```

```
생성자
<__main__.Student object at 0x000001A1D2336A50>
<class '__main__.Student'>
콩쥐 학생은 1학년입니다.
수업을 듣습니다.
```

클래스(class) 정의

- **생성자(constructor) – 매개변수가 있는 생성자**
  - 객체를 생성할 때 매개 변수로 전달하여 멤버변수에 저장함

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def learn(self):
        print("수업을 듣습니다.")

s1 = Student("김하나", 1)
print(f"{s1.name} 학생은 {s1.grade}학년입니다.")
s1.learn()

s2 = Student("이돌", 3)
print(f'{s2.name} 학생은 {s2.grade}학년입니다.')
s2.learn()
```

__str__(self) : 객체 정보 함수

- **__str(self)__ 사용하기**

  문자열을 return하는 함수이다. 객체의 정보를 담고 있다.

```
class Student:
    #생성자
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def learn(self):
        print(f"{self.name} 학생이 수업을 들어요")

    # 학생 정보 출력 메서드
    def __str__(self):  #파이썬 제공 - 문자열 정보
        return f'{self.name} 학생은 {self.grade}학년입니다.'
```

__str__(self) : 객체 정보 함수

- 학생 인스턴스 생성

```
# 인스턴스 생성
st1 = Student("콩쥐", 2)
print(st1)  #객체 출력
st1.learn()

st2 = Student("팥쥐", 1)
print(st2)
st2.learn()
```

```
콩쥐 학생은 2학년입니다.
콩쥐 학생이 수업을 들어요
팥쥐 학생은 1학년입니다.
팥쥐 학생이 수업을 들어요
```

객체 리스트

- 학생 리스트 만들기

```
students = [] #빈 리스트 생성

st1 = Student("김하나", 1)
students.append(st1)

st2 = Student("박열", 3)
students.append(st2)

st3 = Student("이넷", 4)
students.append(st3)

# 2번 인덱스 인스턴스 검색
print(students[2])

print("---------- 학생 명단 ----------")
n = len(students)
for i in range(0, n):
    print(students[i])
    students[i].learn()
```

```
---------- 학생 명단 ----------
김하나 학생은 1학년입니다.
김하나 학생이 수업을 들어요
박열 학생은 3학년입니다.
박열 학생이 수업을 들어요
이넷 학생은 4학년입니다.
이넷 학생이 수업을 들어요
```

자동차 클래스 만들기

- **Car 클래스**

Car
model (모델명) year (연식)
drive() (주행 기능)

자동차 클래스 만들기

- Car 클래스

```
class Car:
    # 생성자
    def __init__(self, model, year):
        self.model = model  #모델명
        self.year = year    #연식

    def drive(self):
        print(f"{self.model}가 달립니다.")

    # 인스턴의 문자열 정보
    def __str__(self):
        return f"모델명: {self.model}, 연식: {self.year}"
```

자동차 클래스 만들기

- Car 클래스

```
# Car의 인스턴스(객체) 생성
c1 = Car("Ionic6", 2024)
print(c1)
c1.drive()

c2 = Car("Sportage", 2021);
print(c2)
c2.drive()
```

```
모델명: Ionic6, 연식: 2024
Ionic6가 달립니다.
모델명: Sportage, 연식: 2021
Sportage가 달립니다.
```

계산기 클래스 만들기

- 계산기 클래스 만들기

Calculator
x, y (2개의 수)
add() (더하기) sub() (빼기) mul() (곱하기) div() (나누기)

계산기 클래스 만들기

- 계산기 클래스 만들기

```
class Calculator:
    def __init__(self):
        self.x = 0

    def add(self, y): # 덧셈
        self.x += y
        return self.x

    def sub(self, y): # 뺄셈
        self.x -= y
        return self.x

    def mul(self, y): # 곱셈
        self.x *= y
        return self.x
```

계산기 클래스 만들기

- 계산기 클래스 만들기

```
    def div(self, y): # 나눗셈
        if y != 0:
            self.x /= y
        else:
            print("Error: 0으로 나눌 수 없습니다.")
        return self.x

cal = Calculator()
print(cal.add(10))   #10
print(cal.sub(4))    #6
print(cal.mul(2))    #12
# print(cal.div(10)) #1.2
# print(cal.div(0))  #12
```

쇼핑몰 장바구니 클래스

- 쇼핑몰 장바구니 구현하기

```
class Cart:
    def __init__(self, user):
        self.user = user
        self.items = []  #장바구니 리스트

    def add(self, *goods):  # 여러 상품 한 번에 추가(가변 매개변수)
        self.items.extend(goods)

    def remove(self, item):
        if item in self.items:
            self.items.remove(item)

    def __str__(self):
        return f"{self.user}'s 장바구니: {self.items}"
```

쇼핑몰 장바구니 클래스

- 쇼핑몰 장바구니 구현하기

```
# 사용
my_cart = Cart("장그래")
my_cart.add("계란", "우유", "라면")
my_cart.remove("우유")
print(my_cart)  # "장그래's 장바구니: ['계란', '라면']"
```

정보 은닉(Information Hiding)

- **정보 은닉**
  - 멤버 변수에 언더스코어(__) 2개를 붙이면 직접 접근할 수 없음
  - **메서드(getter, setter) : get + 변수 이름(), set + 변수이름()**

접근 제어	설 명
public	외부 클래스 어디에서나 접근 할수 있다.
private	같은 클래스 내부 가능, 그 외 접근 불가

정보 은닉(Information Hiding)

- 정보 은닉(접근 제한)

```
class BankAccount:
    def __init__(self):
        self.__ano = ""
        self.__owner = ""
        self.__balance = ""

account1 = BankAccount()
print(account1.__ano)
```

```
Traceback (most recent call last):
  File "d:\korea_IT\pyworks\classes\클래스.py", line 121, in <module>
    print(account1.__ano)
          ^^^^^^^^^^^^^^
AttributeError: 'BankAccount' object has no attribute '__ano'
```

정보 은닉(Information Hiding)

- 은행 계좌 만들기

```
class BankAccount:
    def __init__(self):
        self.__ano = ""
        self.__owner = ""
        self.__balance = ""

    # 계좌 번호
    def set_ano(self, ano):
        self.__ano = ano

    def get_ano(self):
        return self.__ano

    # 계좌주
    def set_owner(self, owner):
        self.__owner = owner

    def get_owner(self):
        return self.__owner
```

```
계좌번호: 12-1234
계좌주: 김기용
잔고: 20000
```

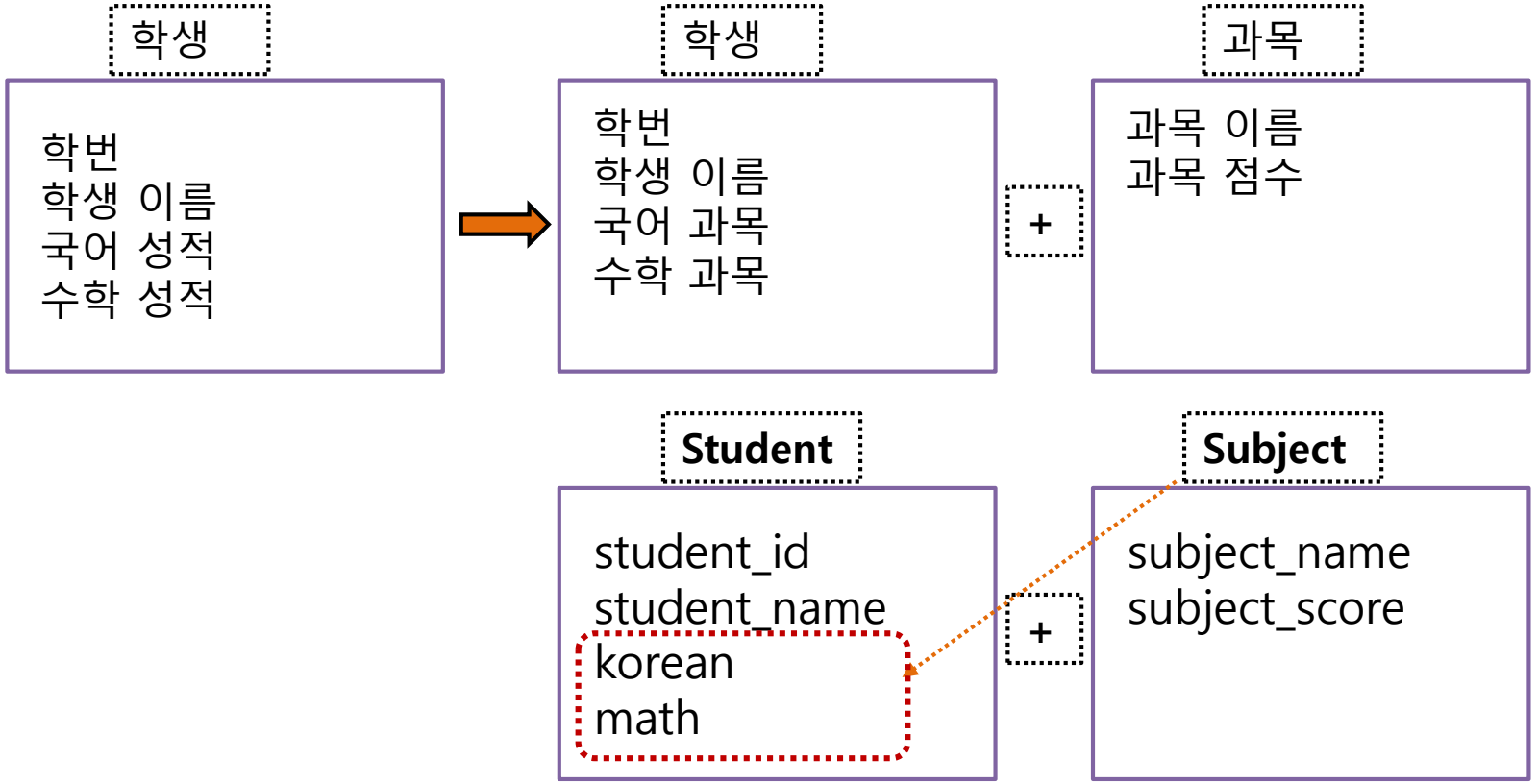
정보 은닉(Information Hiding)

- 은행 계좌 만들기

```
    # 잔고
    def set_balance(self, balance):
        self.__balance = balance

    def get_balance(self):
        return self.__balance

account1 = BankAccount()
# setter
account1.set_ano("12-1234")
account1.set_owner("김기용")
account1.set_balance(20000)

# getter
print("계좌번호:", account1.get_ano())
print("계좌주:", account1.get_owner())
print("잔고:", account1.get_balance())
```

클래스간 참조

- **클래스 참조(Reference)**



클래스간 참조

- **Subject 클래스**

```
class Subject:
    def __init__(self):
        self.subject_name = ""     #과목명
        self.subject_score = 0    #점수

    # 과목명
    def set_subject_name(self, subject_name):
        self.subject_name = subject_name

    def get_subject_name(self):
        return self.subject_name

    # 점수
    def set_subject_score(self, subject_score):
        self.subject_score = subject_score

    def get_subject_score(self):
        return self.subject_score
```

클래스간 참조

- **Student 클래스**

```
# Student 클래스 - Subject 참조
class Student:
  def __init__(self, student_id, student_name):
    self.student_id = student_id       #학번
    self.student_name = student_name   #이름
    self.korean = Subject()            #국어 점수
    self.math = Subject()              #수학 점수

  # 국어 설정
  def set_korean_subject(self, name, score):
    self.korean.set_subject_name(name)
    self.korean.set_subject_score(score)

  # 수학 설정
  def set_math_subject(self, name, score):
    self.math.set_subject_name(name)
    self.math.set_subject_score(score)
```

클래스간 참조

- **Student 클래스**

```
# 평균 점수 계산
def calc_average(self):
    return (self.korean.get_subject_score() + self.math.get_subject_score()) / 2

# 학생 정보 출력
def print_info(self):
    print(f"학번: {self.student_id}")
    print(f"이름: {self.student_name}")
    print(f"{self.korean.get_subject_name()} 점수: {self.korean.get_subject_score()}")
    print(f"{self.math.get_subject_name()} 점수: {self.math.get_subject_score()}")
    print(f"평균 점수: {self.calc_average()}")
    print("----------------------------")
```

클래스간 참조

- 인스턴스 생성

```
# 리스트로 관리
students = []

lee = Student(101, "이정후")
lee.set_korean_subject("국어", 85)
lee.set_math_subject("수학", 88)
students.append(lee)

shin = Student(102, "신유빈")
shin.set_korean_subject("국어", 90)
shin.set_math_subject("수학", 93)
students.append(shin)

print("========== 성 적 표 ==========")
for student in students:
  student.print_info()
```

클래스 변수

◆ 클래스 변수

- 해당 클래스를 사용하는 모두에게 공용으로 사용되는 변수.
- 생성자 def __init__() 위에 위치
- 클래스 이름으로 직접 접근함

```
class Dog:
    kind = "진돗개"  #클래스 변수

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

dog1 = Dog("백구")
dog2 = Dog("밀크")
```

```
print(dog1.name)  # dog1만 유일
print(dog2.name)  # dog2만 유일

# 모든 dog이 공유
# print(dog1.kind)
# print(dog2.kind)

# 클래스 이름으로 직접 접근(올바른 유형)
print(Dog.kind)
```

인스턴스 변수& 클래스 변수

◆ 카운터 만들기

인스턴스 변수

```
class Counter:
    def __init__(self):
        self.x = 0
        self.x += 1

    def get_count(self):
        return self.x

c1 = Counter()
print(c1.get_count())    # 1
c2 = Counter()
print(c2.get_count())    # 1
c3 = Counter()
print(c3.get_count())    # 1
```

클래스 변수

```
class Counter:
    x = 0    # 클래스 변수

    def __init__(self):
        Counter.x += 1
        # 클래스이름으로 직접 접근

    def get_count(self):
        return self.x

c1 = Counter()
print(c1.get_count())    # 1
c2 = Counter()
print(c2.get_count())    # 2
c3 = Counter()
print(c3.get_count())    # 3
```

클래스 변수

◆ 값 교환하기

```
class Cls:
    x, y = 10, 20 # 클래스 변수

    #교환 1
    def change(self):
        temp = self.x
        self.x = self.y
        self.y = self.x

    #교환 2
    def change2(self):
        self.x, self.y = self.y, self.x
```

```
a = Cls()
print(a.x, a.y)

# a.change()
# print(a.x, a.y)

a.change2()
print(a.x, a.y)
```

클래스(class) 모듈 사용

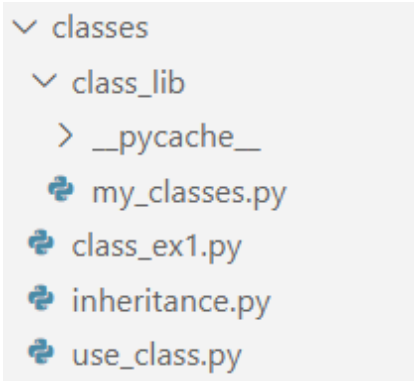
- 클래스를 모듈로 import 하는 방법 – 외부 파일에서 사용

```python
class Student:
    def __init__(self, name, grade):
        self.name = name    #인스턴스 변수
        self.grade = grade

    def learn(self):
        print("수업을 듣습니다.")

    def __str__(self):  #정보 출력 메서드
        return f"{self.name} 학생은 {self.grade}학년입니다."
```

```python
# 외부에서 사용할 때 실행 방지
if __name__ == "__main__":
    # Student 인스턴스 생성
    s1 = Student("김하나", 1)
    print(s1)
    s1.learn()

    s2 = Student("이돌", 3)
    print(s2)
    s2.learn()
```

클래스(class) 모듈 사용

- 모듈 - 클래스 사용하기




```
from class_lib.my_classes import Student

st1 = Student("흥부", 1)
print(st1)
st1.learn()

st2 = Student("놀부", 3)
print(st2)
st2.learn()
```

```
흥부 학생은 1학년입니다.
수업을 듣습니다.
놀부 학생은 3학년입니다.
수업을 듣습니다.
```

실습 문제 - 클래스

아래의 프로그램을 분석하여 결과를 그 실행 결과를 쓰시오

```
class City:
    a = ['Seoul', 'Incheon', 'Daejon', 'Jeju']

str = ''
for i in City.a:
    str += i[0]

print(str)
```

Banking

- **은행 업무**

```
==========================================
1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 종료
==========================================
선택> 1
입금액> 10000
10000원 입금되었습니다. 현재 잔액: 10000
==========================================
1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 종료
==========================================
선택> 2
출금액> 20000
잔액이 부족합니다. 현재 잔액: 10000
==========================================
1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 종료
==========================================
선택> 2
출금액> 5000
5000원 출금되었습니다. 현재 잔액: 5000
```

```
==========================================
1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 종료
==========================================
선택> 3
잔액> 5000
==========================================
1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 종료
==========================================
선택> 4
프로그램을 종료합니다.
```

```
==========================================
1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 종료
==========================================
선택> 1
입금액> addd
숫자로 입력해주세요.
```

오류 처리

Banking

- **은행 업무**

```
def main():
    balance = 0  #전역 변수
    while True:
        print("=============================================")
        print("1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 종료 ")
        print("=============================================")
        choice = input("선택> ")

        try:
            if choice == '1':
                amount = int(input("입금액> "))
                balance += amount  # 잔액 = 잔액 + 입금액
                print(f"{amount}원 입금되었습니다. 현재 잔액: {balance}")
```

Banking

- **은행 업무**

```
            elif choice == '2':
                amount = int(input("출금액> "))
                if amount > balance:
                    print(f"잔액이 부족합니다. 현재 잔액: {balance}")
                else:
                    balance -= amount  # 잔액 = 잔액 + 출금액
                    print(f"{amount}원 출금되었습니다. 현재 잔액: {balance}")
            elif choice == '3':
                print(f"잔액> {balance}")
            elif choice == '4':
                print("프로그램을 종료합니다.")
                break
            else:
                print("지원되지 않는 기능입니다. 다시 선택해 주세요")
        except ValueError:
            print("숫자로 입력해주세요.")

# 실행
main()
```

BankAccount 클래스

- **은행 업무 – 클래스로 구현**

BankAccount
balance(잔고) transaction_history(거래내역)

deposit() - 입금

withdraw() - 출금

get_balance() – 잔고 조회

get_transaction_history() - 거래내역 조회

BankAccount 클래스

- 은행 업무 – 클래스로 구현

```
class BankAccount:
    def __init__(self):
        self.balance = 0 #잔고
        self.transaction_history = [] #거래 내역

    # 입금 기능
    def deposit(self, amount):
        self.balance += amount   #잔고 = 잔고 + 입금액
        self.transaction_history.append(('입금', amount)) #튜플로 저장
        print(f"{amount}원 입금되었습니다. 현재 잔액: {self.balance}")
```

BankAccount 클래스

- 은행 업무 – 클래스로 구현

```
# 출금 기능
def withdraw(self, amount):
    if(amount > self.balance):
        print(f"잔액이 부족합니다. 현재 잔액: {self.balance}")
    else:
        self.balance -= amount   #잔고 = 잔고 + 입금액
        self.transaction_history.append(('출금', amount))
        print(f"{amount}원 출금되었습니다. 현재 잔액: {self.balance}")

# 잔액 조회
def get_balance(self):
    return self.balance

# 거래내역 조회
def get_transaction_history(self):
    return self.transaction_history
```

BankAccount 클래스

- 은행 업무 – 클래스로 구현

```
def main():
    # 은행 계좌 인스턴스 생성
    account = BankAccount()

    while True:
        print("=============================================================")
        print("1. 입금 | 2. 출금 | 3. 잔액조회 | 4. 거래내역 | 5. 종료 ")
        print("=============================================================")
        choice = input("선택> ")

        try:
            if choice == '1':
                amount = int(input("입금액> "))
                account.deposit(amount)  #deposit() 메서드 호출
            elif choice == '2':
                amount = int(input("출금액> "))
                account.withdraw(amount) #withdraw() 메서드 호출
            elif choice == '3':
                print(f"현재 잔액> {account.get_balance()}")
```

BankAccount 클래스

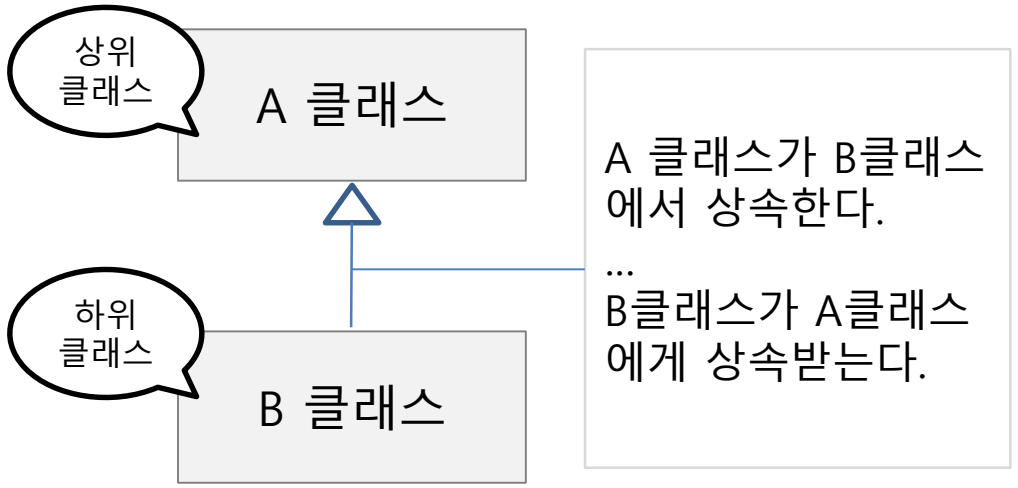
- 은행 업무 – 클래스로 구현

```
            elif choice == '4':
                print("\n[거래 내역]")
                '''
                for type, amount in account.transaction_history:
                    print(f"- {type}: {amount}원")
                '''
                for transaction in account.transaction_history:
                    print(f"- {transaction[0]}: {transaction[1]}원")
            elif choice == '5':
                print("프로그램 종료")
                break
            else:
                print("메뉴를 잘못 선택했습니다. 다시 입력해주세요")
        except ValueError:
            print("숫자를 입력해주세요.")

if __name__ == "__main__":
    main() #실행 함수 호출
```

상속(Inheritance)

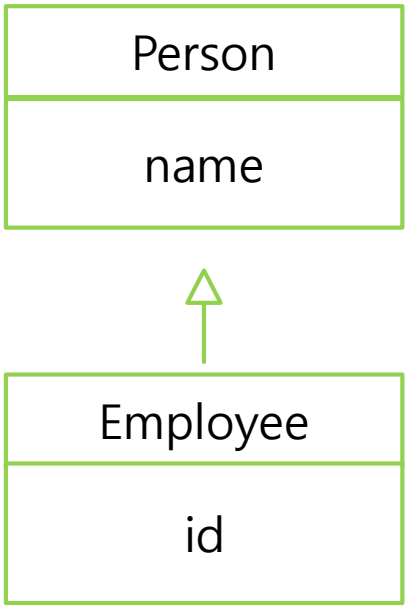
- **상속이란?**
  - 클래스를 정의할때 이미 구현된 클래스를 상속(inheritance) 받아서 속성이나 기능이 확장되는 클래스를 구현함.
  - 클래스 상속 문법

```
class 클래스 이름(상속할 클래스 이름)
```



상속(inheritance)

- 클래스 상속



부모 클래스(사람)
고유 속성 – name

자식 클래스(사원)
고유 속성 – id

상속(inheritance)

- **Person 클래스**

```
class Person:
    def __init__(self, name):
        self.name = name # 부모 클래스 멤버에 값 저장

    def greet(self):
        print(f"안녕하세요. 성명: {self.name}", end="")

    def __str__(self):  #객체 정보 출력
        return f"<Person name: {self.name}>"
```

상속(inheritance)

- **Employee 클래스**

```
class Employee(Person):
    def __init__(self, name, id):
        super().__init__(name) # 부모 클래스의 생성자 호출
        self.id = id # 자식 클래스 멤버에 값 저장

    def greet(self): #메서드 재정의(오버라이드-override)
        super().greet() #부모 클래스의 메서드 호출
        print(f", 사번은 {self.id}입니다.")

    def __str__(self):
        return f"<Employee name: {self.name}, id:{self.id}>"
```

상속(inheritance)

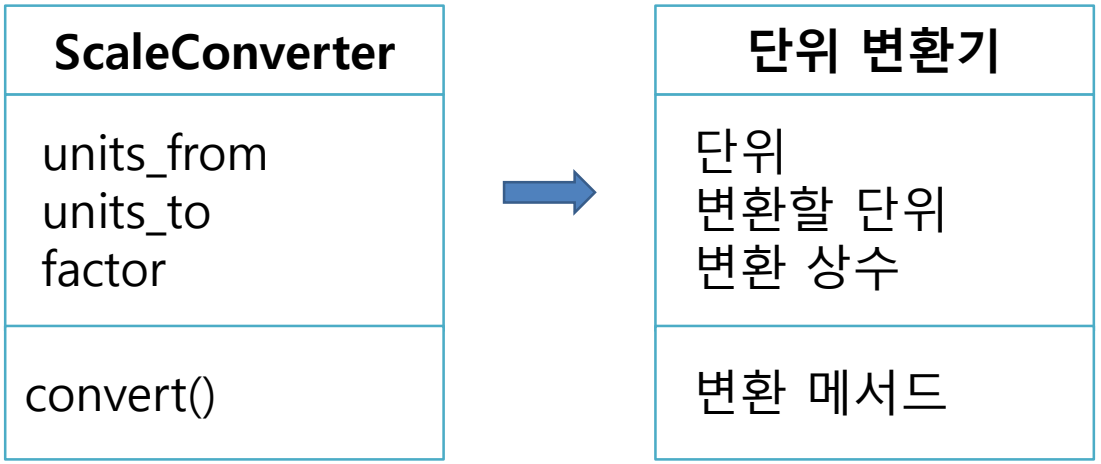
- 클래스 테스트

```
if __name__ == "__main__":
    p1 = Person("김기용")
    e1 = Employee("김기용", "e1234")

    # 객체 정보
    print(p1)
    print(e1)

    # 인사하기
    p1.greet()
    e1.greet()
```

```
<Employee name: 김기용, id:e1234>
안녕하세요. 성명: 김기용, 사번은 e1234입니다.
```

상속(inheritance)

- **단위변환- inch(인치)를 mm로 변환하는 클래스**



1 MB = 1024 KB

1 inch = 25mm

상속(inheritance)

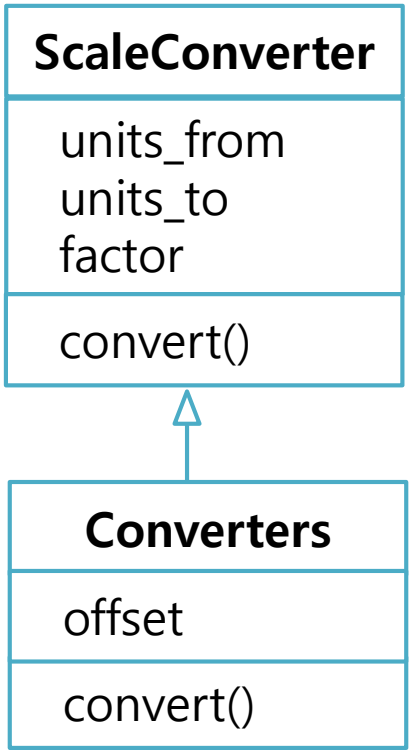
- 단위 변환기 클래스

```
class ScaleConverter:
    def __init__(self, units_from, units_to, factor):
        self.units_from = units_from  #단위
        self.units_to = units_to      #변환할 단위
        self.factor = factor          #변환 상수

    def convert(self, value):
        return value * self.factor   # 입력값 * 변환상수
```

```
sc1 = ScaleConverter("MB", "KB", 1024)
print("10MB를 KB로 변환하기")
print(f"{sc1.convert(10)} {sc1.units_to}")

sc2 = ScaleConverter("inches", "mm", 25)
print("2인치를 mm로 변환하기")
print(f"{sc2.convert(2)} {sc2.units_to}")
```

상속 실습예제

- 단위 변환기 확장 클래스 만들기



화씨온도(F) = 섭씨온도(C) x 1.8 + 32

상속 실습예제

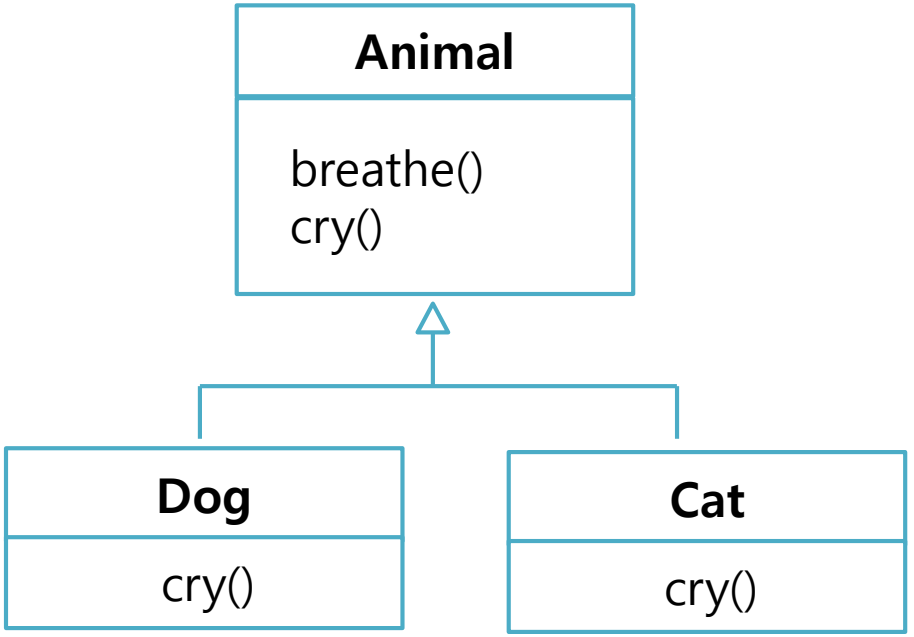
- 단위 변환기 확장 클래스 만들기

```
class Converter(ScaleConverter):
    def __init__(self, units_from, units_to, factor, offset):
        super().__init__(units_from, units_to, factor)
        self.offset = offset #변환 상수2

    def convert(self, value): # 입력값 * 변환상수1 + 변환상수2
        return self.factor * value + self.offset
```

```
con1 = Converter('C', 'F', 1.8, 32)
print("섭씨온도 23C를 화씨온도로 변환하기")
print(str(con1.convert(23)) + con1.units_to)
```

추상 클래스

- **추상 클래스**
  - 추상 메서드를 포함하고 있는 클래스이다.
  - 추상 메서드는 선언만 하고 구현은 상속받는 클래스에서 반드시 구현해야 한다.(예외처리 구문 사용)



추상 클래스

- 추상 클래스

```
class Animal:
    def breathe(self):
        print("동물은 숨을 쉽니다.")

    # raise 예외 처리를 미루면 사용하는 곳에서 try ~ except함
    def cry(self):
        raise NotImplementedError("반드시 메서드를 구현해야 합니다.")

# Animal을 상속받은 Dog 클래스
class Dog(Animal):
    def cry(self):
        print("왈~ 왈~")

class Cat(Animal):
    # def cry(self):
    #     print("야~ 옹!")
    pass
```

추상 클래스

- 추상 클래스

```
# 부모클래스의 인스턴스 생성
animal = Animal()
animal.breathe()

 # 자식클래스의 인스턴스 생성
try:
    dog = Dog()
    dog.breathe() #부모 클래스의 메서드 호출
    dog.cry()

    cat = Cat()
    dog.breathe()
    cat.cry()
except NotImplementedError as e:
    print(f'오류: {e}')
```

```
동물은 숨을 쉽니다.
동물은 숨을 쉽니다.
왈~ 왈~
동물은 숨을 쉽니다.
오류: 반드시 메서드를 구현해야 합니다.
```