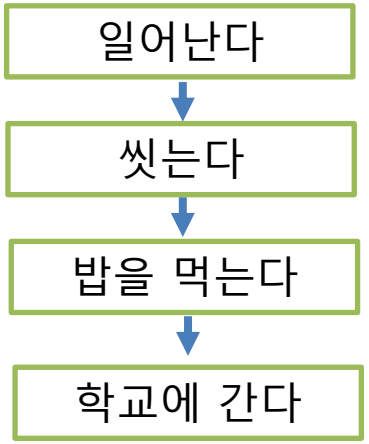
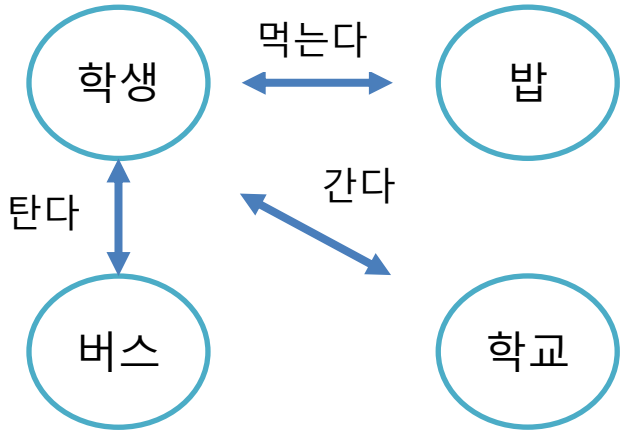
5장. 클래스와 상속

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>

객체 지향 언어의 특성

- **캡슐화(EnCapsulation)**

  데이터와 메서드를 하나의 단위(클래스)로 묶고, 외부에서 직접 접근을 제한하여 데이터를 보호합니다

- **상속(Inheritance)**

  기존 클래스의 속성과 메서드를 새로운 클래스가 물려받아 코드 재사용성을 높입니다.

- **다형성(Polymorphism)**

  같은 이름의 메서드가 다른 클래스에서 다르게 동작할 수 있습니다

클래스(class)

- **클래스란?**
  - 객체 지향 프로그래밍(OOP)의 기본 개념 중 하나로, 관련된 속성과 메서드를 하나의 단위로 묶는 틀입니다.
  - 클래스를 사용하면 코드의 재사용성과 유지보수성을 높일 수 있습니다.
  - 클래스는 설계도와 같으며, 이를 기반으로 여러 객체(인스턴스)를 생성할 수 있습니다.
- **클래스의 속성과 기능**
  - 객체의 특성(property), 속성(attribute) -> **멤버 변수**
  - 객체가 하는 기능 -> **멤버 함수**

클래스(class)

- 클래스 정의 및 사용

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

    def 함수이름(self):
        return
```

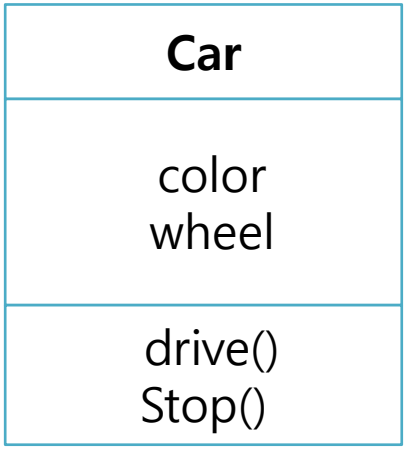
객체(인스턴스) = 클래스( )

객체.멤버변수
객체.멤버 메서드

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

클래스(class)

- **Car 클래스 – 다이어 그램**



속성(Attributes)

메서드(Methord)

클래스(class) 정의

- Car 클래스 정의 및 사용

```
class Car:
    # 속성(Attributes)
    color = "red"
    wheels = 4

    # 메서드(Methods)
    def drive(self):
        print("자동차가 달리고 있습니다.")

    def stop(self):
        print("자동차가 멈췄습니다.")
```

```
if __name__ == "__main__":
    my_car = Car()
    print(f"자동차 색상: {my_car.color}")
    print(f"자동차 바퀴 수: {my_car.wheels}")
    my_car.drive()
    my_car.stop()
```

클래스(class) 정의

- **생성자(constructor)**
  - 객체가 생성될때 자동으로 호출되는 특별한 메서드
  - 생성자는 **__init__(매개변수)**의 형태로 작성하고, 리턴값이 없다
  - 클래스 내의 모든 함수(메서드)의 매개변수에 **self**를 넣어줌

속성(Attributes)

메서드(Methord)

클래스(class) 정의

- **Bike 클래스 정의 및 사용**

```python
# 클래스의 생성자(Constructor)
class Bike:
    # 생성자 메서드 - 객체가 생성될 때 자동으로 호출되는 특별한 메서드
    def __init__(self, color, gears):
        self.color = color  # 자전거 색상
        self.gears = gears  # 자전거 기어 수

    def ride(self):
        print(f"{self.color} 자전거가 {self.gears}단 기어로 달린다.")

    def get_info(self):
        return f"자전거 색상: {self.color}, 기어 수: {self.gears}"

# 클래스 사용 예시
if __name__ == "__main__":
    my_bike = Bike("blue", 21)
    print(my_bike.get_info())
    my_bike.ride()
```

__str__(self) : 객체 정보 함수

- **__str(self)__ 사용하기**

  문자열을 return하는 함수이다. 객체의 정보를 담고 있다.

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

    def learn(self):
        print(f"{self.name} 학생이 공부하고 있습니다.")

    def __str__(self):
        return f"학생 이름: {self.name}, 학생 ID: {self.student_id}"

# 클래스 사용 예시
if __name__ == "__main__":
    student = Student("홍길동", "S12345")
    print(student)  # __str__ 메서드가 호출되어 문자열 출력
    student.learn()
```

객체 리스트

- 학생 리스트 만들기

```
# students 리스트에 여러 학생 추가
    students = [
        Student("김선화", "S10001"),
        Student("고담덕", "S10002"),
        Student("이순신", "S10003")
    ]

    for student in students:
        print(student)
        student.learn()
```

```
학생 이름: 김선화, 학생 ID: S10001
김선화 학생이 공부하고 있습니다.
학생 이름: 고담덕, 학생 ID: S10002
고담덕 학생이 공부하고 있습니다.
학생 이름: 이순신, 학생 ID: S10003
이순신 학생이 공부하고 있습니다.
```

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 구현

```python
# 장바구니 관리 클래스 - cart.py
class Cart:
    # 초기화 메서드
    def __init__(self, user_id):
        self.user_id = user_id
        self.items = [] # 장바구니 아이템 리스트

    # 아이템 추가 메서드
    def add_item(self, item):
        self.items.append(item)
        return f"아이템 '{item}'이(가) 장바구니에 추가되었습니다."

    # 아이템 제거 메서드
    def remove_item(self, item):
        if item in self.items:
            self.items.remove(item)
            return f"아이템 '{item}'이(가) 장바구니에서 제거되었습니다."
        else:
            return f"아이템 '{item}'이(가) 장바구니에 없습니다."
```

쇼핑몰 장바구니 클래스

- 쇼핑몰 장바구니 구현

```
    # 장바구니 조회 메서드
    def view_cart(self):
        if not self.items:
            return "장바구니가 비어 있습니다."
        return f"장바구니 아이템: {', '.join(self.items)}"

# 장바구니 사용 예시
if __name__ == "__main__":
    cart = Cart(user_id="user123")
    print(cart.add_item("노트북"))
    print(cart.add_item("마우스"))
    print(cart.view_cart())
    print(cart.remove_item("마우스"))
    print(cart.view_cart())
```

쇼핑몰 장바구니 클래스

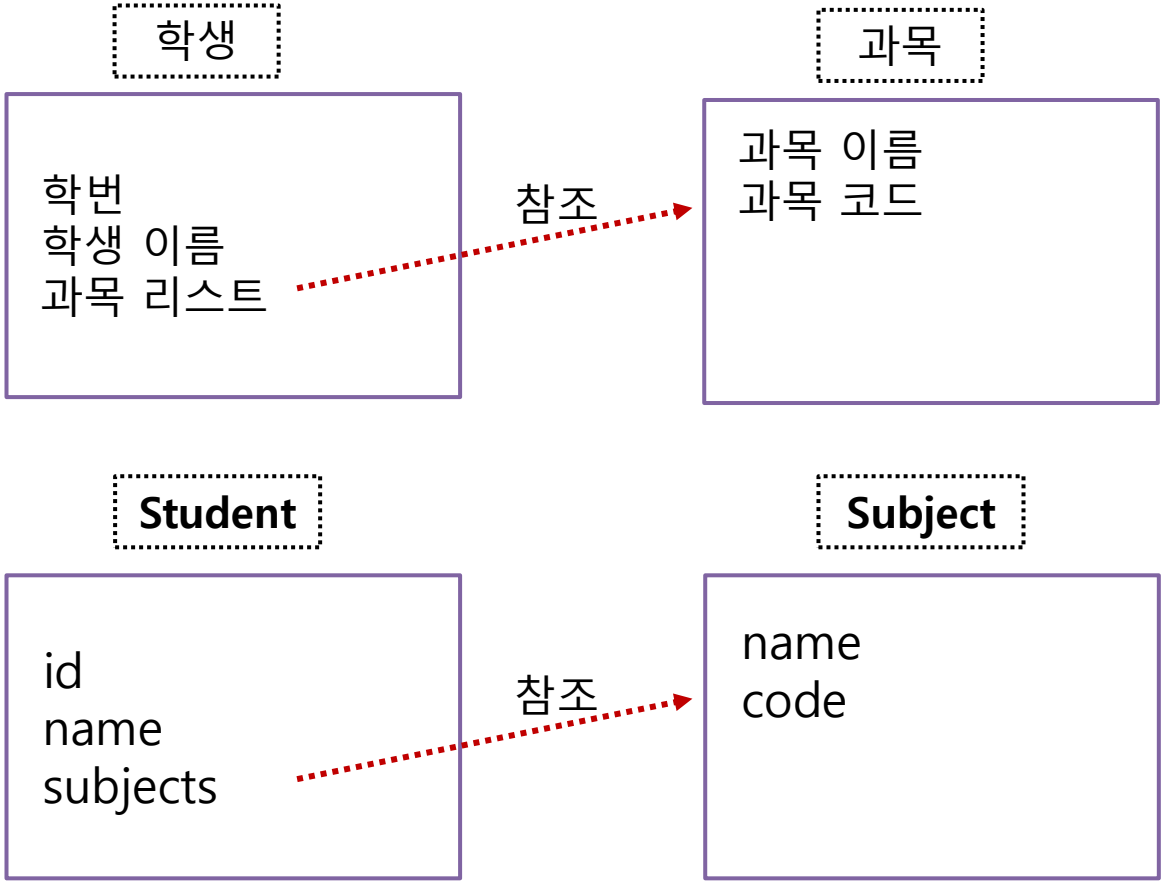
- 챗GPT 활용

상품 추가 메서드를 상품을 복수(여러 개)로 추가하는 메서드로 변경하기

```
# 장바구니 사용 예시
if __name__ == "__main__":
    cart = Cart2(user_id="user456")
    print(cart.add_item("키보드", "모니터", "헤드셋"))
    print(cart.view_cart())
    print(cart.remove_item("모니터"))
    print(cart.view_cart())
```

클래스간 참조

- **클래스 참조(Reference)**



클래스간 참조

- **Subject 클래스**

```python
class Subject:
    # Subject 클래스의 생성자
    def __init__(self, name, code):
        self.name = name  # 과목 이름
        self.code = code  # 과목 코드

    # 과목 정보 출력 메서드
    def get_info(self):
        return f"과목명: {self.name}, 과목코드: {self.code}"

# Subject 클래스 사용 예시
if __name__ == "__main__":
    math = Subject("수학과", "MATH101")
    print(math.get_info())

    physics = Subject("컴퓨터학과", "CS101")
    print(physics.get_info())
```

클래스간 참조

- **Student 클래스**

```python
from subject import Subject

class Student:
    # Student 클래스의 생성자
    def __init__(self, id, name):
        self.id = id            # 학생의 ID
        self.name = name        # 학생의 이름
        self.subjects = []      # 학생이 수강하는 과목들의 리스트

    # 과목을 추가하는 메서드
    def add_subject(self, subject):
        self.subjects.append(subject)

    # 학생의 정보를 출력하는 메서드
    def display_info(self):
        print(f"Student ID: {self.id}, Name: {self.name}")
        print("수강 과목:")
        for subject in self.subjects:
            print(f"- {subject.name} (Code: {subject.code})")
```

클래스간 참조

- **Student 클래스**

```
if __name__ == "__main__":
    math = Subject("MATH101", "수학과")
    computer = Subject("COM101", "컴퓨터학과")

    student = Student(1, "박봄")
    student.add_subject(math)
    student.add_subject(computer)

    student.display_info()

    # 리스트로 과목 추가 예시
    subjects = [
        Subject("ENG101", "영어과"),
        Subject("HIST101", "역사과")
    ]
    for subj in subjects:
        student.add_subject(subj)

    student.display_info()
```

```
Student ID: 1, Name: 박봄
수강 과목:
- MATH101 (Code: 수학과)
- COM101 (Code: 컴퓨터학과)
- ENG101 (Code: 영어과)
- HIST101 (Code: 역사과)
```

클래스 변수

◆ 클래스 변수

- 해당 클래스를 사용하는 모두에게 공용으로 사용되는 변수.
- 생성자 def __init__() 위쪽에 위치
- 객체(인스턴스)를 만들지 않고 클래스 이름으로 직접 속성 및 메서드를 실행함

```
class Dog:
    kind = '말티즈' # 클래스 변수

    def __init__(self, name):
        self.name = name  # 인스턴스 변수
```

클래스 변수

◆ 인스턴스 변수 예제

```
class Dog:
    def __init__(self, name, kind):
        self.name = name  # 인스턴스 변수
        self.kind = kind  # 인스턴스 변수

    def bark(self):
        print(f"{self.name}가 짖습니다: 멍멍!")

if __name__ == "__main__":
    dog1 = Dog("초코", "말티즈")
    dog2 = Dog("콩이", "푸들")

    # 인스턴스 변수 출력
    print(f"Dog1 이름: {dog1.name}, 종류: {dog1.kind}")
    print(f"Dog2 이름: {dog2.name}, 종류: {dog2.kind}")

    dog1.bark()
    dog2.bark()
```

클래스 변수

◆ 클래스 변수 예제

```
class Dog:
    kind = '말티즈' # 클래스 변수

    def __init__(self, name):
        self.name = name  # 인스턴스 변수

    def bark(self):
        print(f"{self.name}가 짖습니다: 멍멍!")

if __name__ == "__main__":
    dog1 = Dog("초코")
    dog2 = Dog("콩이")

    # 클래스 변수와 인스턴스 변수 출력
    print(f"Dog1 이름: {dog1.name}, 종류: {Dog.kind}")
    print(f"Dog2 이름: {dog2.name}, 종류: {Dog.kind}")

    dog1.bark()
    dog2.bark()
```

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
```

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)
```

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.__account_number = None # 은닉된 속성
        self.__account_holder = None  # 은닉된 속성
        self.__balance = 0  # 은닉된 속성

account1 = BankAccount()
# 직접 접근 시도 (에러 발생)
print(account1.__account_number)  # AttributeError 발생
```

정보 은닉(Information Hiding)

- 은행 계좌 클래스

```
# 접근자 메서드(getter)와 설정자 메서드(setter)를 통해 접근
class BankAccount:
    def __init__(self):
        self.__account_number = None
        self.__account_holder = None
        self.__balance = 0

    # 접근자 메서드 (getter)
    def get_account_number(self):
        return self.__account_number

    def get_account_holder(self):
        return self.__account_holder

    def get_balance(self):
        return self.__balance
```

정보 은닉(Information Hiding)

- 은행 계좌 클래스

```
    # 설정자 메서드 (setter)
    def set_account_number(self, account_number):
        self.__account_number = account_number

    def set_account_holder(self, account_holder):
        self.__account_holder = account_holder

account1 = BankAccount()
# 설정자 메서드를 통해 값 설정
account1.set_account_number("123-456-789")
account1.set_account_holder("홍길동")
# 접근자 메서드를 통해 값 출력
print(f"계좌 번호: {account1.get_account_number()}")
print(f"계좌 주: {account1.get_account_holder()}")
print(f"잔액: {account1.get_balance()}")
```

BankAccount 클래스

- **은행 업무 시스템 프로젝트**

BankAccount
balance(잔고) transaction_history(거래내역)
deposit() - 입금 withdraw() - 출금 get_balance() – 잔고 조회 get_transaction_history() - 거래내역 조회

Banking

- **은행 업무 시스템 출력 결과**

```
=== 은행 계좌 관리 시스템 ===
1. 계좌 정보 조회
2. 입금
3. 출금
4. 잔액 조회
5. 거래 내역 조회
6. 종료
옵션을 선택하세요 (1-6): 1
계좌 번호: 123456789, 계좌주: 우영우, 잔액: 1000원

=== 은행 계좌 관리 시스템 ===
1. 계좌 정보 조회
2. 입금
3. 출금
4. 잔액 조회
5. 거래 내역 조회
6. 종료
옵션을 선택하세요 (1-6): 2
입금할 금액을 입력하세요: 5000
입금 완료: 5000원. 현재 잔액: 6000원
```

```
=== 은행 계좌 관리 시스템 ===
1. 계좌 정보 조회
2. 입금
3. 출금
4. 잔액 조회
5. 거래 내역 조회
6. 종료
옵션을 선택하세요 (1-6): 3
출금할 금액을 입력하세요: 5000
출금 완료: 5000원. 현재 잔액: 1000원

=== 은행 계좌 관리 시스템 ===
1. 계좌 정보 조회
2. 입금
3. 출금
4. 잔액 조회
5. 거래 내역 조회
6. 종료
옵션을 선택하세요 (1-6): 4
현재 잔액: 1000원
```

BankAccount 클래스

- 은행 업무 시스템

```
class BankAccount:
    # 초기화 메서드
    def __init__(self, account_number, account_holder, balance=0):
        self.__account_number = account_number
        self.__account_holder = account_holder
        self.__balance = balance
        self.__transactions = []  # 거래 내역 리스트

    # 입금 메서드
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            self.__transactions.append(("입금", amount)) # 입금 기록 추가
            return f"입금 완료: {amount}원. 현재 잔액: {self.__balance}원"
        else:
            return "입금 금액은 양수여야 합니다."
```

BankAccount 클래스

- 은행 업무 시스템

```
    # 출금 메서드
    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.__balance:
                self.__balance -= amount
                self.__transactions.append(("출금", amount)) # 출금 기록 추가
                return f"출금 완료: {amount}원. 현재 잔액: {self.__balance}원"
            else:
                return "잔액이 부족합니다."
        else:
            return "출금 금액은 양수여야 합니다."

    # 잔액 조회 메서드
    def get_balance(self):
        return f"현재 잔액: {self.__balance}원"

    # 계좌 정보 출력 메서드
    def get_account_info(self):
        return f"계좌 번호: {self.__account_number}, \
계좌주: {self.__account_holder}, 잔액: {self.__balance}원"
```

BankAccount 클래스

- 은행 업무 시스템

```
    # 거래 내역 조회 메서드
    def get_transactions(self):
        return self.__transactions

def main():
    account = BankAccount("123456789", "우영우", 1000)

    while True:
        print("\n=== 은행 계좌 관리 시스템 ===")
        print("1. 계좌 정보 조회")
        print("2. 입금")
        print("3. 출금")
        print("4. 잔액 조회")
        print("5. 거래 내역 조회")
        print("6. 종료")

        choice = input("메뉴를 선택하세요 (1-6): ")

        if choice == '1':
            print(account.get_account_info())
```

BankAccount 클래스

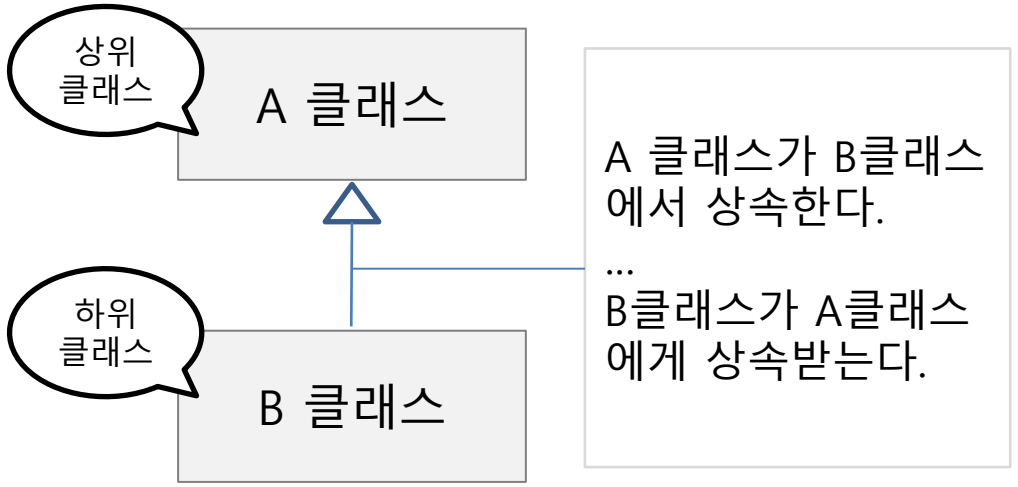
- 은행 업무 시스템

```
        elif choice == '2':
            amount = int(input("입금할 금액을 입력하세요: "))
            print(account.deposit(amount))
        elif choice == '3':
            amount = int(input("출금할 금액을 입력하세요: "))
            print(account.withdraw(amount))
        elif choice == '4':
            print(account.get_balance())
        elif choice == '5':
            for transaction in account.get_transactions():
                print(f"{transaction[0]}: {transaction[1]}원")
        elif choice == '6':
            print("프로그램을 종료합니다.")
            break
        else:
            print("잘못된 선택입니다. 다시 시도하세요.")

# 예시 사용법
if __name__ == "__main__":
    main()
```

상속(Inheritance)

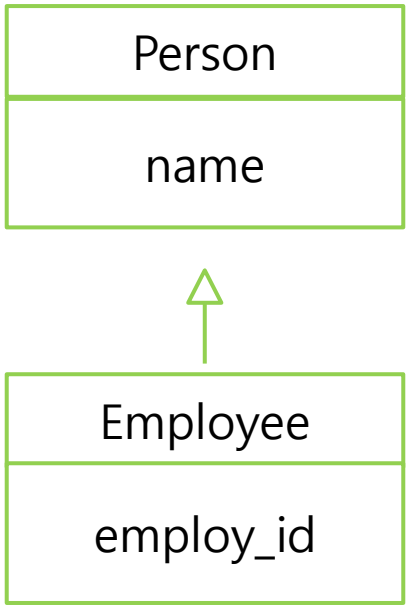
- **상속이란?**
  - 기존 클래스(부모 클래스)의 속성과 메서드를 새로운 클래스(자식 클래스)에서 물려받아 사용하는 것을 의미합니다.
  - 클래스 상속 문법

```
class 클래스 이름(상속할 클래스 이름)
```



상속(inheritance)

- 상속 다이어그램



부모 클래스(사람)
고유 속성 – name

자식 클래스(사원)
고유 속성 – employ_id

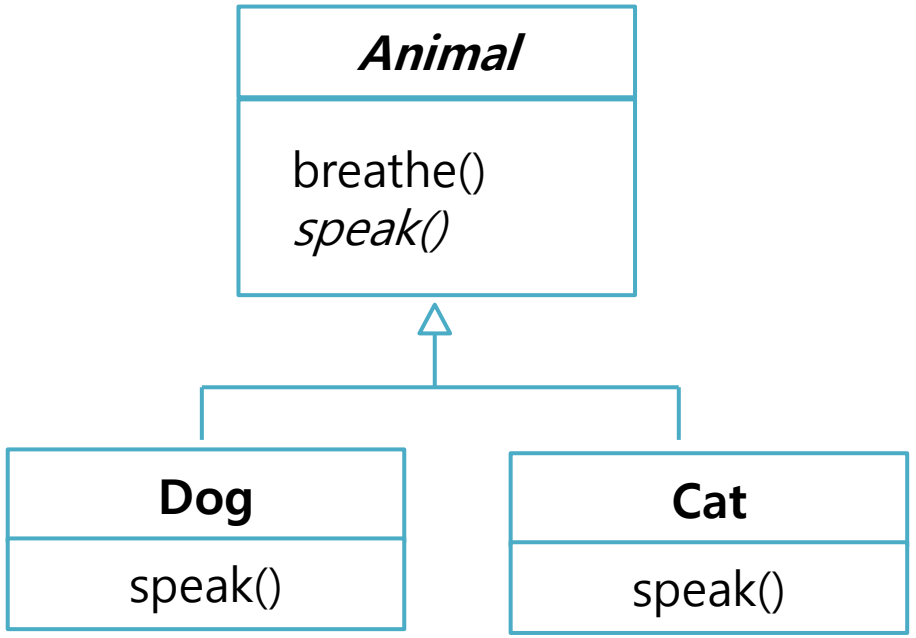
상속(inheritance)

- **Person 클래스 -> Employee 클래스**

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

    def introduce(self):
        return f"안녕하세요, 저는 {self.name}입니다."

class Employee(Person):  # Person 클래스를 상속받음
    def __init__(self, name, employee_id):
        super().__init__(name)  # 부모 클래스의 생성자 호출
        self.employee_id = employee_id

    def introduce(self):
        parent_intro = super().introduce()  # 부모 클래스의 메서드 호출
        return f"{parent_intro} 제 사번은 {self.employee_id}입니다."

if __name__ == "__main__":
    employee = Employee("김상희", "E2026001")
    print(employee.introduce())
```

상속(inheritance)

- **추상 클래스의 상속**
  - 추상클래스란 추상 메서드를 포함하고 있는 클래스입니다.
  - 추상 메서드는 선언만 하고 구현은 상속받는 클래스에서 반드시 구현해야 한다.(예외처리 구문 사용)



추상 클래스

- 추상 클래스의 상속

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

    def speak(self):
        raise NotImplementedError("서브클래스에서 구현해야 합니다.")

class Dog(Animal):
    def speak(self):
        return "멍멍!"

class Cat(Animal):
    '''
    def speak(self):
        return "야옹!"
    '''
```

추상 클래스

- 추상 클래스의 상속

```
if __name__ == "__main__":
    try:
        dog = Dog()
        cat = Cat()

        print(dog.speak())
        print(cat.speak())  # NotImplementedError 발생
    except NotImplementedError as e:
        print(e)
```

```
멍멍!
서브클래스에서 구현해야 합니다.
```