UNIT 36

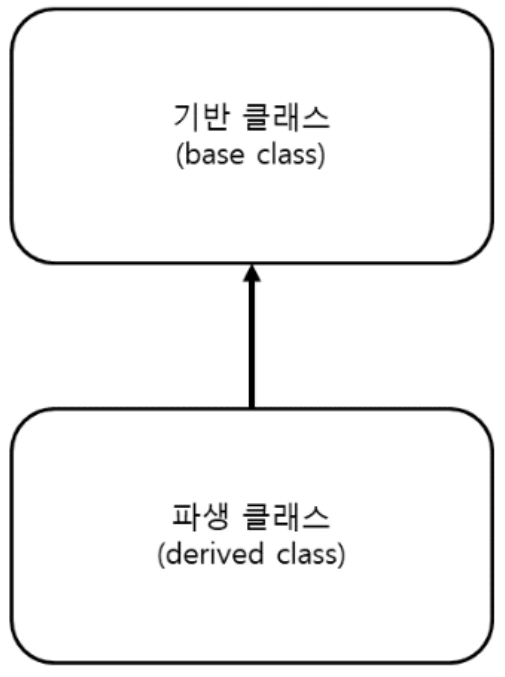
클래스 상속 사용하기

36 클래스 상속 사용하기

클래스 속성 사용하기

- 클래스 상속은 물려받은 기능을 유지한 채로 다른 기능을 추가할 때 사용하는 기능임
- 기능을 물려주는 클래스를 기반 클래스(base class), 상속을 받아 새롭게 만드는 클래스를 파생 클래스(derived class)라고 함

▼ 그림 클래스 상속

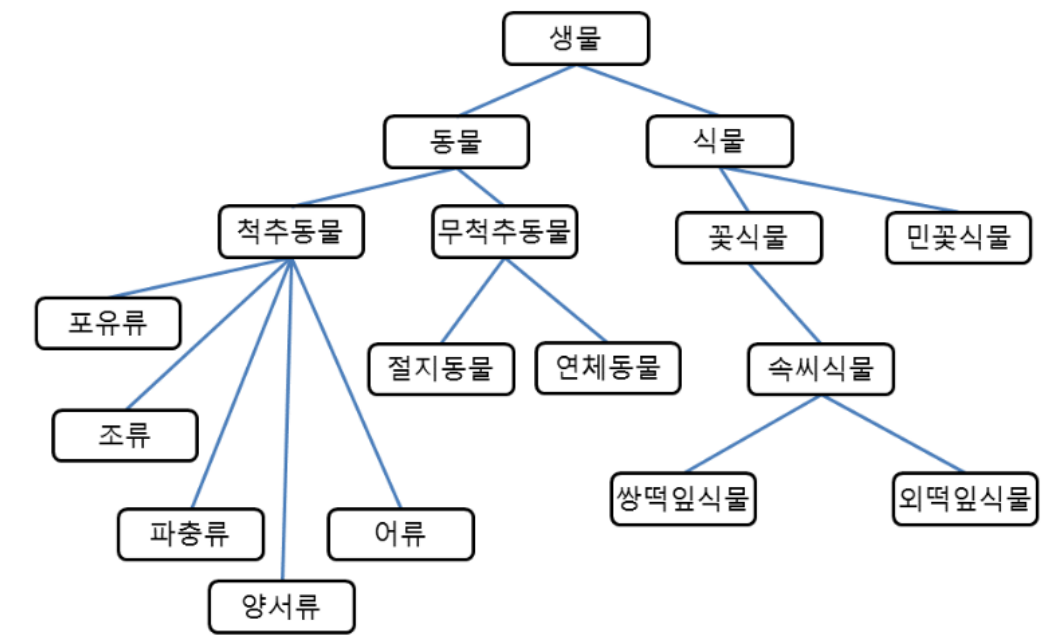


36 클래스 상속 사용하기

클래스 속성 사용하기

- 보통 기반 클래스는 부모 클래스(parent class), 슈퍼 클래스(superclass)라고 부르고, 파생 클래스는 자식 클래스(child class), 서브 클래스(subclass)라고도 부름

▼ 그림 생물 분류



36 클래스 상속 사용하기

클래스 속성 사용하기

- 클래스 상속도 기반 클래스의 능력을 그대로 활용하면서 새로운 클래스를 만들 때 사용함
- 상속은 기존 기능을 재사용할 수 있어서 효율적임

36.1 사람 클래스로 학생 클래스 만들기

사람 클래스로 학생 클래스 만들기

- 클래스 상속은 다음과 같이 클래스를 만들 때 ( )(괄호)를 붙이고 안에 기반 클래스 이름을 넣음

```
class 기반클래스이름:
    코드

class 파생클래스이름(기반클래스이름):
    코드
```

36.1 사람 클래스로 학생 클래스 만들기

» 사람 클래스로 학생 클래스 만들기

- 그럼 간단하게 사람 클래스를 만들고 사람 클래스를 상속받아 학생 클래스를 만들어보자

class_inheritance.py

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def study(self):
        print('공부하기')

james = Student()
james.greeting()    # 안녕하세요.: 기반 클래스 Person의 메서드 호출
james.study()       # 공부하기: 파생 클래스 Student에 추가한 study 메서드
```

실행 결과

```
안녕하세요.
공부하기
```

- Person 클래스의 기능을 물려받은 Student 클래스가 됨

36.1 사람 클래스로 학생 클래스 만들기

사람 클래스로 학생 클래스 만들기

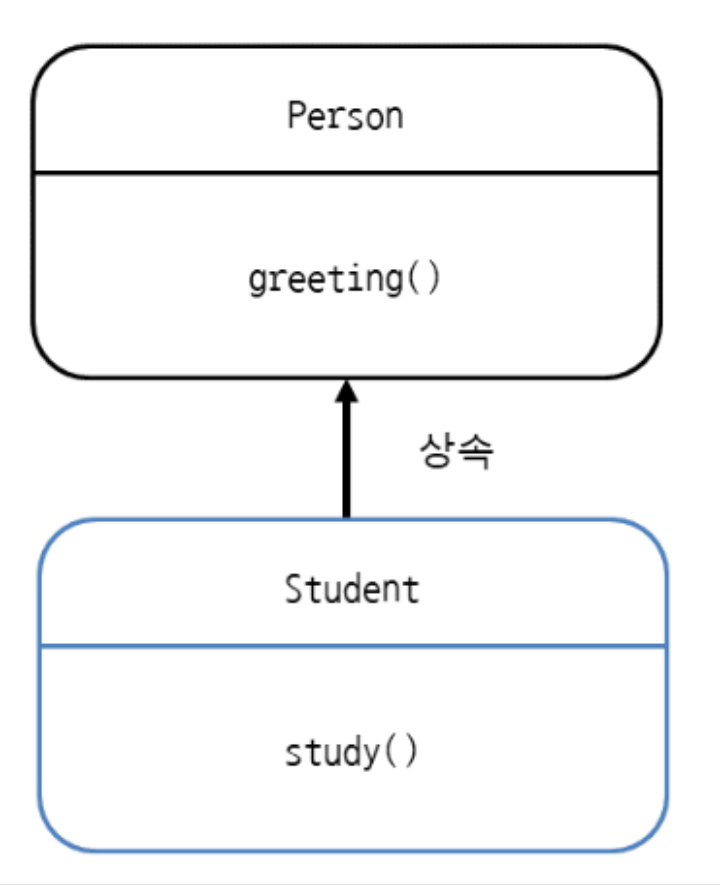
- Student 클래스에는 greeting 메서드가 없지만 Person 클래스를 상속받았으므로 greeting 메서드를 호출할 수 있음

```
james = Student()
james.greeting()    # 안녕하세요.: 기반 클래스 Person의 메서드 호출
```

```
james.study()       # 공부하기: 파생 클래스 Student에 추가한 study 메서드
```

36.1 사람 클래스로 학생 클래스 만들기

▼ 그림 클래스 상속과 메서드 추가



36.1 사람 클래스로 학생 클래스 만들기

사람 클래스로 학생 클래스 만들기

- 클래스 상속은 기반 클래스의 기능을 유지하면서 새로운 기능을 추가할 수 있음
- 클래스 상속은 연관되면서 동등한 기능일 때 사용함
- 학생은 사람이므로 연관된 개념이고, 학생은 사람에서 역할만 확장되었을 뿐 동등한 개념

36.2 상속 관계와 포함 관계 알아보기

상속 관계

class_is_a.py

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def study(self):
        print('공부하기')
```

- 상속은 명확하게 같은 종류이며 동등한 관계일 때 사용함
- "학생은 사람이다."라고 했을 때 말이 되면 동등한 관계임
- 상속 관계를 영어로 is-a 관계라고 부름(Student is a Person)

36.2 상속 관계와 포함 관계 알아보기

포함 관계

- 학생 클래스가 아니라 사람 목록을 관리하는 클래스를 만든다면 다음과 같이 리스트 속성에 Person 인스턴스를 넣어서 관리하면 됨

class_has_a.py

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class PersonList():
    def __init__(self):
        self.person_list = []    # 리스트 속성에 Person 인스턴스를 넣어서 관리

    def append_person(self, person):    # 리스트 속성에 Person 인스턴스를 추가하는 함수
        self.person_list.append(person)
```

- 같은 종류에 동등한 관계일 때는 상속을 사용하고, 그 이외에는 속성에 인스턴스를 넣는 포함 방식을 사용하면 됨

36.3 기반 클래스의 속성 사용하기

» 기반 클래스의 속성 사용하기

class_inheritance_attribute_error.py

```
class Person:
    def __init__(self):
        print('Person __init__')
        self.hello = '안녕하세요.'

class Student(Person):
    def __init__(self):
        print('Student __init__')
        self.school = '파이썬 코딩

james = Student()
print(james.school)
print(james.hello)    # 기반 클래스의 속성을 출력하려고 하면 에러가 발생함
```

실행 결과

```
Student __init__
파이썬 코딩
Traceback (most recent call last):
  File "C:\project\class_inheritance_attribute_error.py", line 14, in <module>
    print(james.hello)
AttributeError: 'Student' object has no attribute 'hello'
```

- 실행을 해보면 에러가 발생하는데 .Person의 __init__ 메서드가 호출되지 않으면 self.hello = '안녕하세요.'도 실행되지 않아서 속성이 만들어지지 않음

36.3 기반 클래스의 속성 사용하기

super()로 기반 클래스 초기화하기

- super()를 사용해서 기반 클래스의 __init__ 메서드를 호출해줌

- super().메서드()

class_inheritance_attribute.py

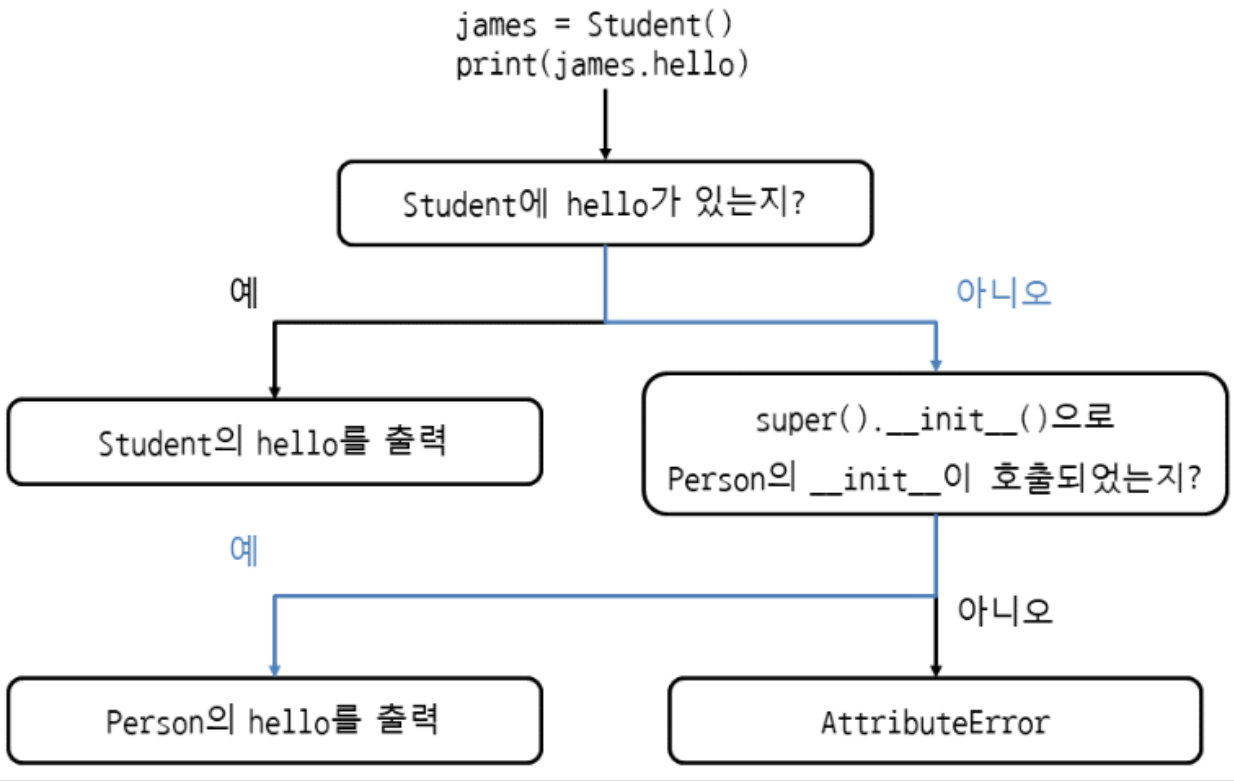
```
class Person:
    def __init__(self):
        print('Person __init__')
        self.hello = '안녕하세요.'

class Student(Person):
    def __init__(self):
        print('Student __init__')
        super().__init__()                  # super()로 기반 클래스의 __init__ 메서드 호출
        self.school = '파이썬 코딩 '

james = Student()
print(james.school)
print(james.hello)
```

실행 결과

```
Student __init__
Person __init__
파이썬 코딩
안녕하세요.
```

36.3 기반 클래스의 속성 사용하기

▼ 그림 기반 클래스의 속성을 찾는 과정



36.3 기반 클래스의 속성 사용하기

기반 클래스를 초기화하지 않아도 되는 경우

- 만약 파생 클래스에서 __init__ 메서드를 생략한다면 기반 클래스의 __init__이 자동으로 호출되므로 super()는 사용하지 않아도 됨

class_inheritance_no_init.py

```
class Person:
    def __init__(self):
        print('Person __init__')
        self.hello = '안녕하세요.'

class Student(Person):
    pass

james = Student()
print(james.hello)
```

실행 결과

```
Person __init__
안녕하세요.
```

- 파생 클래스에 __init__ 메서드가 없다면 기반 클래스의 __init__이 자동으로 호출되므로 기반 클래스의 속성을 사용할 수 있음

36.4 메서드 오버라이딩 사용하기

메서드 오버라이딩 사용하기

- 다음과 같이 Person의 greeting 메서드가 있는 상태에서 Student에도 greeting 메서드를 만듦

class_method_overriding.py

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def greeting(self):
        print('안녕하세요. 저는 파이썬 코딩 학교 학생입니다.')

james = Student()
james.greeting()
```

실행 결과

```
안녕하세요. 저는 파이썬 코딩 학교 학생입니다.
```

36.4 메서드 오버라이딩 사용하기

» 메서드 오버라이딩 사용하기

- 오버라이딩(overriding)은 무시하다, 우선하다라는 뜻을 가지고 있는데 말 그대로 기반 클래스의 메서드를 무시하고 새로운 메서드를 만든다는 뜻임
- 여기서는 Person 클래스의 greeting 메서드를 무시하고 Student 클래스에서 새로운 greeting 메서드를 만들었음
- 메서드 오버라이딩은 보통 프로그램에서 어떤 기능이 같은 메서드 이름으로 계속 사용되어야 할 때 메서드 오버라이딩을 활용함
- Student 클래스에서 인사하는 메서드를 greeting2로 만들어야 한다면 모든 소스 코드에서 메서드 호출 부분을 greeting2로 수정해야함

```
    def greeting(self):
        print('안녕하세요.')
```

```
    def greeting(self):
        print('안녕하세요. 저는 파이썬 코딩 도장 학생입니다.')
```

- 이럴 때는 기반 클래스의 메서드를 재활용하면 중복을 줄일 수 있음

36.4 메서드 오버라이딩 사용하기

» 메서드 오버라이딩 사용하기

- 다음과 같이 오버라이딩된 메서드에서 super()로 기반 클래스의 메서드를 호출해보자

class_method_overring_super.py

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class Student(Person):
    def greeting(self):
        super().greeting()    # 기반 클래스의 메서드 호출하여 중복을 줄임
        print('저는 파이썬 코딩 학교 학생입니다.')

james = Student()
james.greeting()
```

실행 결과

```
안녕하세요.
저는 파이썬 코딩 학교 학생입니다.
```

- 중복되는 기능은 파생 클래스에서 다시 만들지 않고, 기반 클래스의 기능을 사용하면 됨
- 메서드 오버라이딩은 원래 기능을 유지하면서 새로운 기능을 덧붙일 때 사용함

36.5 다중 상속 사용하기

다중 상속 사용하기

- 다중 상속은 여러 기반 클래스로부터 상속을 받아서 파생 클래스를 만드는 방법임

```
class 기반클래스이름1:
    코드

class 기반클래스이름2:
    코드

class 파생클래스이름(기반클래스이름1, 기반클래스이름2):
    코드
```

36.5 다중 상속 사용하기

다중 상속 사용하기

- 사람 클래스와 대학교 클래스를 만든 뒤 다중 상속으로 대학생 클래스를 만들어보자

class_multiple_inheritance.py

```
class Person:
    def greeting(self):
        print('안녕하세요.')

class University:
    def manage_credit(self):
        print('학점 관리')

class Undergraduate(Person, University):
    def study(self):
        print('공부하기')

james = Undergraduate()
james.greeting()        # 안녕하세요.: 기반 클래스 Person의 메서드 호출
james.manage_credit()   # 학점 관리: 기반 클래스 University의 메서드 호출
james.study()           # 공부하기: 파생 클래스 Undergraduate에 추가한 study 메서드
```

실행 결과

```
안녕하세요.
학점 관리
공부하기
```

36.5 다중 상속 사용하기

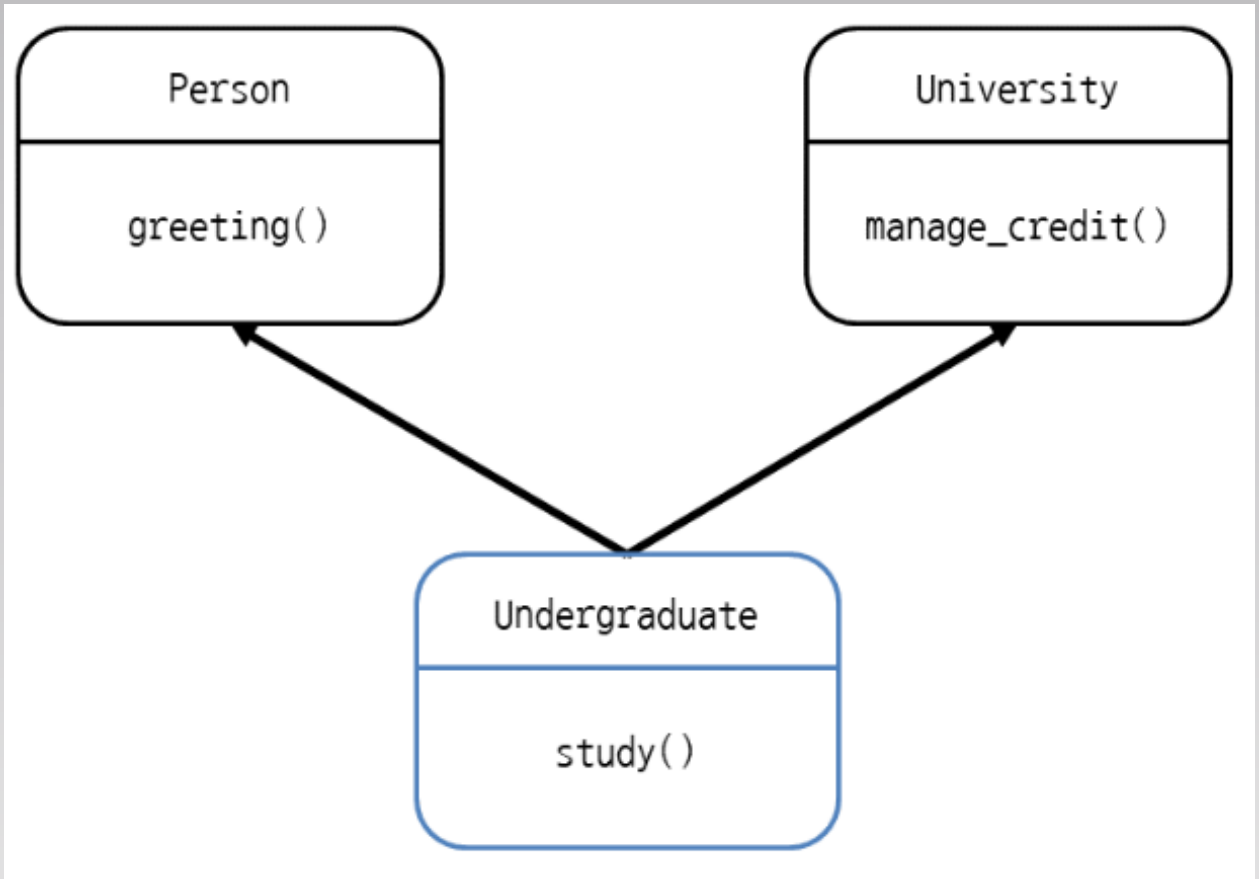
다중 상속 사용하기

- 다음과 같이 Undergraduate 클래스의 인스턴스로 Person의 greeting과 University의 manage_credit을 호출할 수 있음

```
james = Undergraduate()
james.greeting()        # 안녕하세요.: 기반 클래스 Person의 메서드 호출
james.manage_credit()   # 학점 관리: 기반 클래스 University의 메서드 호출
james.study()           # 공부하기: 파생 클래스 Undergraduate에 추가한 study 메서드
```

36.5 다중 상속 사용하기

▼ 그림 다중 상속



36.6 추상 클래스 사용하기

추상 클래스 사용하기

- 추상 클래스는 메서드의 목록만 가진 클래스이며 상속받는 클래스에서 메서드 구현을 강제하기 위해 사용함

```
from abc import *

class 추상클래스이름(metaclass=ABCMeta):
    @abstractmethod
    def 메서드이름(self):
        코드
```

36.6 추상 클래스 사용하기

추상 클래스 사용하기

- 그럼 학생 추상 클래스 StudentBase를 만들고, 이 추상 클래스를 상속받아 학생 클래스 Student를 만들어보자

class_abc_error.py

```
from abc import *

class StudentBase(metaclass=ABCMeta):
    @abstractmethod
    def study(self):
        pass

    @abstractmethod
    def go_to_school(self):
        pass

class Student(StudentBase):
    def study(self):
        print('공부하기')

james = Student()
james.study()
```

실행 결과

```
Traceback (most recent call last):
  File "C:\project\class_abc_error.py", line 16, in <module>
    james = Student()
TypeError: Can't instantiate abstract class Student with abstract methods go_to_school
```

36.6 추상 클래스 사용하기

추상 클래스 사용하기

- StudentBase를 상속받은 Student에서는 study 메서드만 구현하고, go_to_school 메서드는 구현하지 않았으므로 에러가 발생함
- 추상 클래스를 상속받았다면 @abstractmethod가 붙은 추상 메서드를 모두 구현해야 함

36.6 추상 클래스 사용하기

추상 클래스 사용하기

- 다음과 같이 Student에서 go_to_school 메서드도 구현해주자

class_abc.py

```
from abc import *

class StudentBase(metaclass=ABCMeta):
    @abstractmethod
    def study(self):
        pass

    @abstractmethod
    def go_to_school(self):
        pass

class Student(StudentBase):
    def study(self):
        print('공부하기')

    def go_to_school(self):
        print('학교가기')

james = Student()
james.study()
james.go_to_school()
```

실행 결과

```
공부하기
학교가기
```

36.6 추상 클래스 사용하기

추상 클래스 사용하기

- 추상 클래스는 파생 클래스가 반드시 구현해야 하는 메서드를 정해줄 수 있음
- 참고로 추상 클래스의 추상 메서드를 모두 구현했는지 확인하는 시점은 파생 클래스가 인스턴스를 만들 때임
- james = Student()에서 확인(구현하지 않았다면 TypeError 발생)

36.6 추상 클래스 사용하기

추상 메서드를 빈 메서드로 만드는 이유

- 또 한 가지 중요한 점이 있는데 추상 클래스는 인스턴스로 만들 수가 없다는 점
- 다음과 같이 추상 클래스 StudentBase로 인스턴스를 만들면 에러가 발생함

```
>>> james = StudentBase()
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    james = StudentBase()
TypeError: Can't instantiate abstract class StudentBase with abstract methods go_to_school, study
```

```
    @abstractmethod
    def study(self):
        pass    # 추상 메서드는 호출할 일이 없으므로 빈 메서드로 만듦

    @abstractmethod
    def go_to_school(self):
        pass    # 추상 메서드는 호출할 일이 없으므로 빈 메서드로 만듦
```

- 추상 클래스는 인스턴스로 만들 때는 사용하지 않으며 오로지 상속에만 사용함
- 파생 클래스에서 반드시 구현해야 할 메서드를 정해 줄 때 사용함