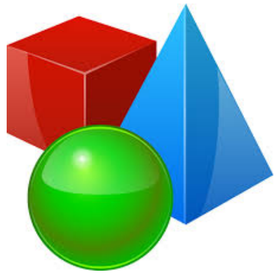
CHAPTER 12

클래스와 객체

Classes and Objects



박 진 수 교수
서울대학교•경영대학
jinsoo@snu.ac.kr

학습 목차

- ☑ 클래스와 객체 만들기
  - 클래스와 객체
  - 클래스 속성
  - 인스턴스 속성과 초기화 메소드
  - 메소드
- ☑ 클래스 상속
- ☑ 설명문자열

클래스와 객체 만들기

Constructing Classes and Objects

클래스와 객체

Classes and Objects

Classes and Objects

사용자 정의 클래스

사용자 정의 클래스(custom class)란? 사용자가 직접 만든 클래스

작성 방법

```
class 클래스이름:
    클래스-명령문...
```

- 클래스를 정의할 때 **클래스이름** 다음에 바로 '쌍점(:)'이 온다
- 다음 두 가지 방법은 파이썬 2에서 클래스를 정의할 때 사용하는 방법이기 때문에 권장하지 않는다
  - 방법 1 : `class 클래스이름(object):`
  - 방법 2 : `class 클래스이름( ):`이며 최상위 클래스인 `object`를 상속할 때는 이를 생략하고, 빈 괄호를 사용해서 정의
  - 다른 클래스를 상속하려면 괄호 안에 다른 클래스의 이름을 적어 넣으면 된다
  - 최상위 클래스인 `object`를 상속할 경우 괄호를 생략해도 되기 때문에 요즘은 방법1과 방법 2를 사용하지 않는다
- **클래스-명령문**에는 클래스의 속성과 메소드들이 온다
  - 속성의 종류로는 **클래스 속성**과 **인스턴스 속성**이 있으며, 각각 **클래스 변수**와 **인스턴스 변수**라고도 부른다
  - 메소드는 파이썬이 제공하는 메소드와 사용자가 정의한 메소드로 나뉜다
    - 파이썬이 제공하는 특별한 메소드들은 주로 메소드 이름이 밑줄 두 개로 시작해서 밑줄 두 개로 끝난다
    - 예) `__init__()` 등

파이썬에서 클래스와 객체 만들기

클래스의 자료형을 확인해보자

```
class MyClass: pass
type(MyClass)
```

```
type(int)
```

```
type(list)
```

객체의 자료형을 확인해보자

```
c = MyClass()
type(c)
```

```
x = int()
type(x)
```

```
y = list()
type(y)
```

파이썬에서 클래스와 객체 만들기

서울대학교 느티나무 아르바이트생(PartTimer) 클래스를 정의하고 자료형을 확인해보자

```
class PartTimer:
    pass

type(PartTimer)
```

```
kim = PartTimer()
lee = PartTimer()

type(kim)
type(lee)
```

- PartTimer 클래스의 특성을 가진(물론 아직은 아무것도 없지만) kim 과 lee 객체를 생성했다
- 두 객체 모두 PartTimer 클래스라는 '틀'을 이용해 만들었기 때문에 같은 특징을 가진다
- 이처럼 클래스를 한 번 정의해 놓으면 이후에 다시 정의하지 않고도 같은 특성을 가진 객체를 반복적으로 생성할 수 있다

클래스 속성

Class Attributes

클래스 속성 vs. 인스턴스 속성

- 클래스 속성(class attributes)
  - 클래스에 속한 변수를 말하며, 클래스 내 모든 인스턴스에 동일하게 적용하는 속성
  - 해당 클래스의 모든 인스턴스가 공유하는 데이터를 저장(참조)하기 때문에 '클래스 변수(class variable)'라고도 한다
  - 모든 인스턴스에 적용하는 공통적인 데이터를 클래스 속성으로 만들어 사용하면 편리하다
    - 예를 들어, 계좌 클래스의 이자율을 모든 계좌 인스턴스에 동일하게 적용할 경우 이자율을 클래스 속성으로 모델링하는 것이 좋다
- 인스턴스 속성(instance attributes)
  - 개별 인스턴스가 각자의 고유한 값을 가져야 할 때 사용하는 속성
    - '객체 속성'과 같은 의미
  - 각 인스턴스의 고유한 데이터를 저장(참조)하기 때문에 '인스턴스 변수(instance variable)'라고도 한다
  - 개별 객체의 고유한 값은 인스턴스 속성을 사용해서 저장
    - 예를 들어, '학생' 클래스를 모델링 할 때, 학생의 '이름'이나 '성적'은 개별 학생에 속한 고유한 값이기 때문에 인스턴스 속성으로 지정하는 것이 좋다
- 클래스 속성과 인스턴스 속성의 가장 큰 차이
  - 속성 값이 적용되는 범위
    - 클래스 속성의 값은 모든 인스턴스가 공유하기 때문에 이 값이 바뀌면 전체 인스턴스에도 영향을 준다
    - 인스턴스 속성의 값은 개별 객체에 속하기 때문에 특정 객체의 값이 바뀌어도 다른 객체에는 영향을 주지 않는다

그럼 먼저 느티나무 카페 아르바이트생인 **`PartTimer`** 클래스에 필요한 속성이 무엇인지 생각해보자

??

클래스 속성

- 시급
  - 느티나무 카페 아르바이트생의 시급은 시간 당 9,500원이라 가정
  - 모든 아르바이트생의 시급을 기본적으로 9,500원으로 한다면, 클래스를 정의할 때 시급의 기본값으로 9,500원을 지정할 수가 있다

앞의 예에서는 클래스의 이름만 정의했지만 클래스의 속성과 메소드를 정의하기 위해서는
함수를 정의할 때와 마찬가지로 들여쓰기(indentation)를 해서 내용을 작성해야 한다

```
class PartTimer:
    hour_rate = 9500
```

PartTimer 클래스 속성 정의하기

- PartTimer는 느티나무 아르바이트생 클래스
- PartTimer 클래스의 속성으로 hour_rate 변수를 정의
- hour_rate의 기본값을 '9500'으로 설정
- hour_rate 클래스를 통해 만들어지는 모든 객체들은 이 변수를 속성(attribute)으로 가진다

클래스 속성에 접근하기

클래스이름.클래스속성이름

클래스를 통해 클래스 속성에 접근

```
PartTimer.hour_rate    # 클래스를 통해 클래스 속성에 접근
```

객체이름.클래스속성이름

인스턴스를 통해 클래스 속성에 접근

```
kim = PartTimer()
kim.hour_rate          # 인스턴스를 통해 클래스 속성에 접근
```

```
lee = PartTimer()
lee.hour_rate          # 인스턴스를 통해 클래스 속성에 접근
```

인스턴스에서 속성 값 수정하기

객체이름.속성이름 = 속성값

수정 또는 추가가 가능

```
lee.hour_rate = 7000    # 무경험자의 시급을 7,000원으로 변경
lee.hour_rate           # lee의 시급은?
kim.hour_rate           # kim의 시급은?
PartTimer.hour_rate     # 클래스의 시급은?
```

클래스와 인스턴스의 속성

커피 가게에 손님이 많이 찾아와 아르바이트생 한 명을 더 고용했다
변수 이름이 park 인 객체를 하나 더 생성해보자

```
park = PartTimer()
park.hour_rate                  # park의 시급은?
PartTimer.hour_rate = 10000     # 클래스 속성을 수정하려면 '클래스이름.클래스속성이름'으로 접근
park.hour_rate                  # park의 시급은?
kim.hour_rate                   # kim의 시급은?
lee.hour_rate                   # lee의 시급은?
```

클래스 속성 추가하기

```
# 새로운 클래스 속성을 만들어 PartTimer 클래스에 추가
PartTimer.cafe_name = '느티나무 카페'
```

```
kim.cafe_name
park.cafe_name
lee.cafe_name
```

```
kim.hour_rate
park.hour_rate
lee.hour_rate
```

정리 : 클래스 속성

클래스에 속한 변수를 말하며, 클래스 내 모든 인스턴스에 동일하게 적용하는 속성

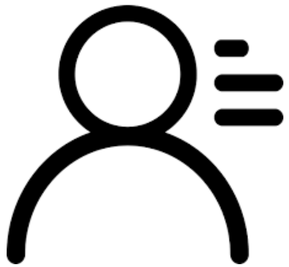
해당 클래스의 모든 인스턴스가 공유하는 데이터를 저장(참조)하기 때문에 '클래스 변수(class variable)'라고도 한다

모든 인스턴스에 적용하는 공통적인 데이터를 클래스 속성으로 만들어 사용하면 편리하다

예를 들어, 계좌 클래스의 이자율을 모든 계좌 인스턴스에 동일하게 적용할 경우 이자율을 클래스 속성으로 모델링하는 것이 좋다

인스턴스 속성과 초기화 메소드

Instance Attributes & Initialization Method



개별 아르바이트생의 근무 시간은 어떻게 기록해야 할까?

그렇다면 객체마다 자신의 값을 가질 수 있도록 하는 방법은 없을까?

초기화 메소드

작성 방법

```
__init__(self[, 매개변수, ...]):
    초기화-명령문...
```

- 클래스가 인스턴스를 생성할 때 초기화 할 수 있도록 하는 특별한 메소드
- 어떤 객체가 클래스로부터 만들어질 때 저절로 호출되며, 그 객체가 갖게 될 여러 가지 기본적인 초깃값을 설정해주는 역할을 한다
- 첫 번째 매개변수로 `self`가 온다

`self`란?

- 메소드 정의는 일반 함수 정의와 비슷하지만, 첫 번째 매개변수로 `self`가 오는 것이 다르다
- 메소드를 호출할 때 자동으로 전달하는 인스턴스 자신에 대한 참조를 위해서 메소드를 정의할 때 첫 번째 매개변수로 `self`를 지정해야만 인스턴스의 메소드로 사용이 가능
- 따라서 이 매개변수는 반드시 제공해야 하며, 관례에 따라 이 매개변수의 이름을 `self`라고 명한다

아르바이트생의 근무 시간을 기록할 **whours**란 인스턴스 속성을 초기화 메소드 안에 정의해보자

그리고 아르바이트생의 수가 늘어날 것을 대비해 아르바이생마다 닉네임(**nickname**)을 만들어 주도록 하자

클래스에 인스턴스 속성 추가하기

인스턴스 속성은 주로 `__init__` 메소드 안에 정의한다

`self.인스턴스속성이름` 형식으로 정의하면 된다

```
class PartTimer:
    cafe_name = '느티나무 카페'
    hour_rate = 9500
    def __init__(self, name):
        self.nickname = name
        self.whours = 0
```

`PartTimer` 인스턴스 속성 정의 및 초기화하기

`PartTimer` 인스턴스 속성 정의 및 초기화하기

속성 한정자

- 클래스 속성과 인스턴스 속성을 결정하는 것은...
  - 앞서 클래스 속성은 클래스 이름 아래 선언하였고 인스턴스 속성은 초기화 메소드 안에서 선언
  - 클래스 속성과 인스턴스 속성의 선언 위치를 구분하는 것이 의미론적으로 좋지만 속성의 선언 위치가 속성의 종류를 반드시 결정하는 것은 아니다
- 한정자(qualifier)!!!
  - 속성의 한정자가 클래스 이름이면 클래스 속성의 자격이 주어진다
  - 속성의 한정자가 `self`면 인스턴스 속성의 자격이 주어진다

```
class PartTimer:
    def __init__(self, name):
        PartTimer.cafe_name = '느티나무 카페'
        PartTimer.hour_rate = 9500
        self.nickname = name
        self.whours = 0
```

클래스 속성
클래스 속성
인스턴스 속성
인스턴스 속성

초기화 메소드의 역할

이 코드를 실행해보자

```
lee = PartTimer()
```

왜 오류가 날까?

오류가 나지 않으려면?

```
lee.cafe_name    # 클래스 속성

lee.hour_rate    # 클래스 속성

lee.nickname     # 인스턴스 속성

lee.whours       # 인스턴스 속성
```

객체에 인스턴스 속성 추가하기

조금 전에 생성한 닉네임이 '네오'인 객체 `lee`에 새로운 속성인 `email` 을 추가해보자

```
lee.email = 'neo@kfriends.kr'
```

```
lee.email
```

```
PartTimer.email
```

이유는?

정리 : 인스턴스 속성

개별 인스턴스가 각자의 고유한 값을 가져야 할 때 사용하는 속성

각 인스턴스의 고유한 데이터를 저장(참조)하기 때문에 '인스턴스 변수(instance variable)'라고도 한다

개별 객체의 고유한 값은 인스턴스 속성을 사용해서 저장

예를 들어, '학생' 클래스를 모델링 할 경우,
학생의 ‘이름'이나 ‘성적'은 개별 학생에 속한 고유한 값이기 때문에 인스턴스 속성으로 지정하는 것이 좋다

클래스 속성과 인스턴스 속성의 차이

클래스 속성

```
class MyClass:
    counter = 0
    def __init__(self, name):
        self.nickname = name
        MyClass.counter += 1
```

```
x = MyClass('네오')
x.nickname
```

```
x.counter
```

```
y = MyClass('프로도')
y.nickname
```

```
y.counter
```

```
z = MyClass('라이언')
z.nickname
```

```
z.counter
```

```
x.counter
```

```
y.counter
```

```
MyClass.counter
```

클래스 속성과 인스턴스 속성의 차이

인스턴스 속성

```
class MyClass:
    counter = 0
    def __init__(self, name):
        self.nickname = name
        self.counter += 1
```

```
x = MyClass('네오')
x.nickname
```

```
x.counter
```

```
y = MyClass('프로도')
y.nickname
```

```
y.counter
```

```
z = MyClass('라이언')
z.nickname
```

```
z.counter
```

```
x.counter
```

```
y.counter
```

```
MyClass.counter
```

정리 : 클래스 속성과 인스턴스 속성의 가장 큰 차이

속성 값이 적용되는 범위

- 클래스 속성의 값은
  모든 인스턴스가 공유하기 때문에 이 값이 바뀌면 전체 인스턴스에도 영향을 준다
- 인스턴스 속성의 값은
  개별 객체에 속하기 때문에 특정 객체의 값이 바뀌어도 다른 객체에는 영향을 주지 않는다

캡슐화와 속내용 감추기

- 캡슐화(encapsulation)란?
  - 외부 세계로부터 객체 안의 세부적인 내용을 숨기는 것(data hiding)을 말한다
  - 캡슐화가 잘된 클래스는 데이터 손상을 방지한다
    - 클래스에서 정의한 객체의 속성은 객체 내부에 존재하며, 다른 객체가 접근하거나 변경하기 쉽지 않도록 설계하기 때문
    - 데이터의 손상을 방지하기 위해서 주로 속성을 사적(private) 속성으로 정의해서 외부로부터 데이터가 변경되거나 손상되는 것을 방지
    - 이를 속내용 감추기 또는 데이터 숨기기(data hiding)라 한다

- 사적 속성(private attribute)이란?
  - 해당 속성을 생성한 객체를 외부의 다른 객체가 접근할 수 없도록 만든 속성
  - 즉, 객체 자신만이 접근 가능한 속성

그렇다면 외부 세계에서 객체의 속성(데이터)에 어떻게 접근할까?

문제는...

순수한 사적 속성이 파이썬에는 존재하지 않는다는 것이다

그렇다면 파이썬에서는 어떻게 데이터를 외부 세계로부터 보호할 수 있을까?

네임 맹글링

네임 맹글링(name mangling)이란?

- 속성 이름을 쉽게 알아보지 못하게 함으로써 외부에서 의도적으로 속성에 직접 접근하는 것을 어렵게 하는 기법
- 속성 이름 앞에 밑줄 두 개(__)를 붙여서 사용하면 된다

```
class MyClass:
    def __init__(self, attr):
        self.__attr = attr  # 인스턴스 속성 __attr를 초기화한다

c = MyClass('속성')
c.__attr                    # 인스턴스 속성 __attr에 접근한다
```

📖 네밍 맹글링 관련해서는 교재의 '부록 6 : 클래스와 객체 완전 정복'에서 상세히 설명

네임 맹글링

그렇다고 속성에 접근하는 것이 불가능한 것은 아니다

다음과 같은 형식으로 접근이 가능

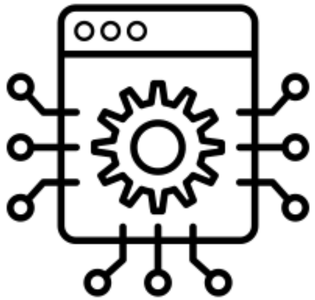
객체이름._클래스이름__속성이름

```
c._MyClass__attr
```

앞서 정의한 인스턴스 속성인 `nickname`과 `whours`를 네임 맹글링해보자

메소드

Methods



메소드

메소드(method)란? 인스턴스의 행동(behavior)을 정의하기 위해 클래스에 구현한 일종의 함수

메소드는 클래스가 외부 세계에 제공하는 서비스며 특정 정보를 전달하거나 어떤 행동을 요청하는 등 서로 다른 객체 사이의 상호 작용을 위한 의사소통 수단으로 사용

메소드 호출 방법

```
객체이름.메소드이름([...])
```

- 클래스가 생성한 인스턴스를 통해 메소드를 호출할 때 사용
- 괄호 안의 [...]는 전달인자가 있을 때 사용

```
클래스이름.메소드이름(인스턴스[, ...])
```

- 직접 클래스를 통해 메소드를 호출할 때 사용
- 메소드를 호출하는 인스턴스를 첫 번째 전달인자로 지정해야 한다
- 괄호 안의 [...]는 전달인자가 있을 때 사용

메소드 정의

PartTimer 초기화 메소드 정의하기

PartTimer 메소드 정의하기

PartTimer 메소드 정의하기

PartTimer 메소드 정의하기

```
class PartTimer:
    cafe_name = '느티나무 카페'
    hour_rate = 9500
    def __init__(self, name):
        """초기화 메소드다."""
        self.__nickname = name
        self.__whours = 0
    def nickname(self):
        """닉네임을 불러오는 메소드다."""
        return self.__nickname
    def whours(self, hours_worked):
        """근무 시간을 설정하는 메소드다.
        기존 근무시간에 현 근무 시간을 추가해서 합한다.
        """
        self.__whours += hours_worked
    def total_wage(self):
        """급여를 계산하는 메소드다."""
        return self.__whours * PartTimer.hour_rate
```

인스턴스를 통해 메소드 호출하기

객체이름.메소드이름([...])

```
kim = PartTimer('프로도')
lee = PartTimer('네오')
```

```
kim.nickname()
```

```
lee.nickname()
```

```
kim.whours(2)      # '프로도'가 2시간 근무했다
lee.whours(3)      # '네오'가 3시간 근무했다
lee.whours(5)      # '네오'가 5시간을 근무했다(총 8시간)
kim.whours(3)      # '프로도'가 3시간 근무했다(총 5시간)
```

```
kim.total_wage()  # 이번 달 '프로도'의 급여는 9500 x 5시간
```

```
lee.total_wage()  # 이번 달 '네오'의 월급은 9500 x 8시간
```

클래스를 통해 메소드에 접근하기

```
클래스이름.메소드이름(인스턴스[, ...])
```

```
PartTimer.whours(7)
```

```
PartTimer.whours(kim, 3)    # '프로도'가 3시간 추가 근무했다(총 8시간)
PartTimer.total_wage(kim)
```

정리 : 클래스와 객체

- 클래스(class)
  - 같은 종류의 집단에 속하는 모든 객체(object)들의 속성(attribute)과 행동(behavior)을 기술하여 추상화(abstraction)시킨 사용자 정의 자료형(custom (or user defined) data type)
  - 인스턴스화(instantiation) 할 수 있는 틀(template)을 정의한 것

- 객체(object)
  - 클래스의 인스턴스(instance)

- 클래스 멤버(class members)
  - 속성(attributes) 즉, 변수 또는 필드는 클래스의 특정 인스턴스의 데이터를 저장(참조)한다
    - 속성은 미리 선언할 필요가 없다(로컬 변수를 생성할 때와 같은 원리)
      - 속성은 값이 처음 할당될 때 생성된다
  - 메소드(method)는 클래스와 관련된 행동(behavior)을 구현한다
    - 메소드는 '특정 클래스와 연관된 함수'이기 때문에 클래스 내부에 정의하고 메소드 호출(method invocation; method call)로 접근할 수 있다

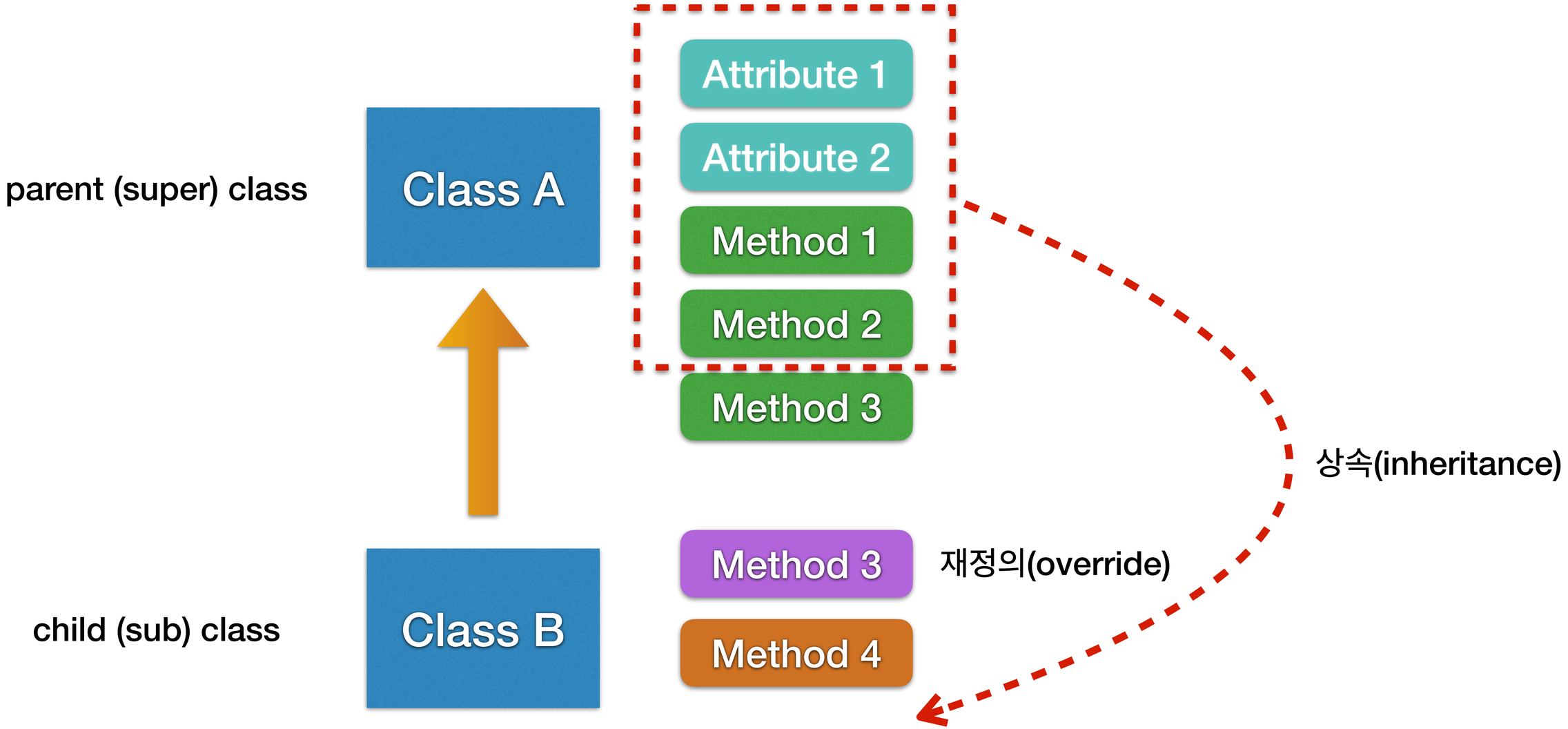
정리 : 클래스와 객체

- 클래스 멤버 호출(class members)
  - 속성이나 메소드를 호출하는 방법은 도트 연산자('.')를 사용
    - `클래스이름.속성이름`, `클래스이름.메소드이름(객체이름)`
    - `객체이름.속성이름`, `객체이름.메소드이름()`
  - 클래스 멤버는 보통 공용(public)으로 사용
- 상속(inheritance)
  - 다중상속(multiple inheritance)을 지원
- 캡슐화(encapsulation)
  - 파이썬에서 데이터 숨기기(data hiding)가 가능은 하지만 완벽하지는 않다
- 다형성(polymorphism)
  - 특정 클래스의 객체를 그 클래스의 상위 클래스(또는 베이스 클래스(base class)라고 부른다) 객체로도 사용 할 수 있다
  - 하위 클래스는 상위 클래스에서 상속받은 메소드를 재정의(override)해서 다른 방식으로 동작(행동)하도록 할 수 있다

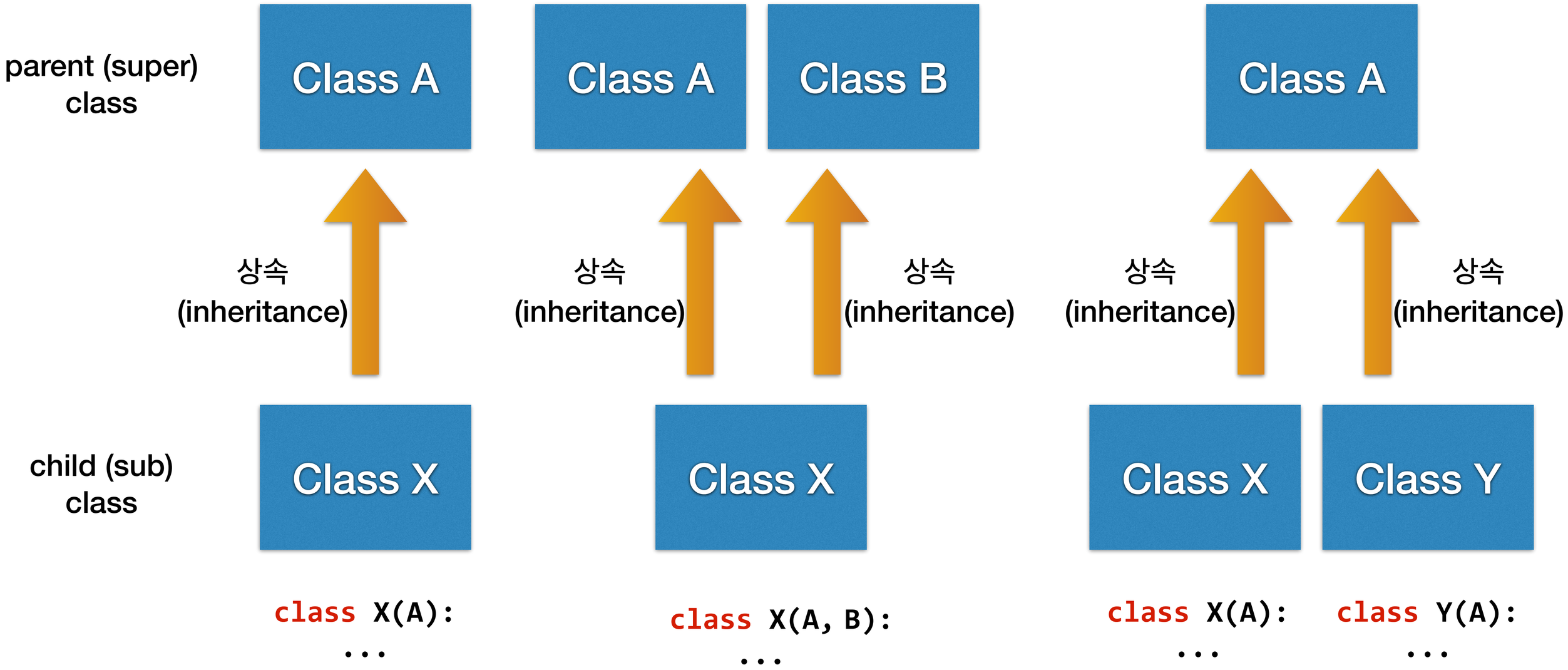
클래스 상속

Class Inheritance

클래스 상속



클래스 상속


parent (super) class
Class A
Class A
Class B
Class A
상속 (inheritance)
상속 (inheritance)
상속 (inheritance)
상속 (inheritance)
상속 (inheritance)
child (sub) class
Class X
Class X
Class X
Class Y
class X(A):
...
class X(A, B):
...
class X(A):
...
class Y(A):
...

클래스 상속

클래스 상속 작성 방법

```
class 클래스이름(상위-클래스-이름):
    [클래스 속성 정의 ...]
    def __init__(self[, 매개변수-1, 매개변수-2, ...]):
        super().__init__([...])
        [인스턴스 속성 정의 ...]
        명령문...
    [기타 메소드 정의 ...]
```

super 함수

- 하위 클래스에서 상위 클래스의 메소드를 호출할 때 사용
- `super` 함수를 통해 상위 클래스의 초기화 함수인 `__init__`를 호출하여 상속받는 속성들을 초기화

하위 클래스 만들기

PartTimer의 하위 클래스인 ChiefPartTimer 클래스를 정의

```
class ChiefPartTimer(PartTimer):

    def __init__(self, name):
        super().__init__(name)  # super 함수로 상속받는 모든 속성들을 초기화
        self.__workers = []     # 인스턴스 속성 : 근무자 리스트를 초기화

    def add_worker(self, other):
        if isinstance(other, PartTimer):              # PartTimer의 인스턴스면
            self.__workers.append(other.nickname())   # __workers에 닉네임을 추가
        else:                                         # PartTimer의 인스턴스가 아니면
            print(f'{PartTimer.__name__}의 인스턴스가 아닙니다.')  # 오류 메시지를 출력

    def show_workers(self):
        print(self.__workers)   # 자신이 관리하는 아르바이트생 모두의 닉네임을 출력
```

- PartTimer의 모든 속성(클래스와 인스턴스 속성)을 상속 받는다
- ChiefPartTimer의 속성으로 __workers를 추가
  - 자신이 관리하는 아르바이트생 정보를 담고 있는 리스트
- ChiefPartTimer의 메소드로 add_worker와 show_workers를 추가
  - add_worker 메소드는 전달인자가 PartTimer 또는 PartTimer 하위 클래스의 인스턴스면 __workers에 추가
  - show_workers 메소드는 아르이트생의 닉네임을 출력

예시 : 하위 클래스 사용하기

```
park = ChiefPartTimer('라이언')
park.nickname()
```

```
park.add_worker(kim)
```

```
park.add_worker(lee)
```

```
park.add_worker('제이지')
```

```
park.show_workers()
```

```
park.whours(5)
park.total_wage()      # 9500 x 5시간
```

상속을 처리하는 내장 함수

`isinstance(x, y)`

- *x* 가 *y* 클래스 혹은 *y* 의 하위 클래스 인스턴스면 '참(`True`)'을, 그렇지 않으면 '거짓(`False`)'을 반환
- *y* 에는 클래스 이름, 튜플, **객체이름.`__class__`** 가 올 수 있다
  - *y* 가 튜플이고 튜플에 포함한 여러 클래스(자료형) 중 한 클래스의 인스턴스면 '참(`True`)'을 반환
  - *y* 가 클래스가 아니면 `TypeError` 오류가 난다
  - 참고로 **객체이름.`__class__`** 를 호출하면 해당 객체를 생성한 클래스를 확인할 수 있다

`issubclass(x, y)`

- *x* 클래스가 *y* 클래스의 하위 클래스(subclass)면 '참(`True`)'을 그렇지 않으면 '거짓(`False`)'을 반환
- 모든 클래스는 자기 자신을 자신의 하위 클래스로 간주한다
- *y* 가 튜플이면 튜플이 포함한 모든 클래스를 확인
- 만약 *x* 나 *y* 가 클래스가 아니면 `TypeError` 오류가 난다

예시 : isinstance(x, y)

```
isinstance(-5, int)
```

```
isinstance(-5, str)
```

```
isinstance(0.0, float)
```

```
isinstance('', str)
```

```
isinstance([1], list)
```

```
class Person:
    pass

p = Person()
```

```
isinstance(p, Person)
```

```
type(p)
```

```
p.__class__
```

```
p.__class__.__name__
```

예시 : issubclass(x, y)

```
class Person: pass
class Student(Person): pass
class CollegeStudent(Student): pass
class Teacher(Person): pass
class Professor(Teacher): pass
```

```
issubclass(Person, CollegeStudent)
```

```
issubclass(CollegeStudent, Person)
```

```
issubclass(CollegeStudent, Student)
```

```
issubclass(CollegeStudent, CollegeStudent)
```

```
issubclass(Teacher, Person)
```

```
issubclass(Teacher, Teacher)
```

```
issubclass(Teacher, Student)
```

```
issubclass(Teacher, Professor)
```

```
issubclass(Professor, Student)
```

```
issubclass(Professor, Person)
```

```
issubclass(Student, Professor)
```

다중속성

다중상속(multiple inheritance)**이란?** 클래스가 한 개 이상의 상위 클래스를 상속할 수 있도록 하는 상속 메커니즘

작성 방법

```
class 클래스이름(상위-클래스-1, 상위-클래스-2, ..., 상위-클래스-N):
    명령문...
```

- ☑ 장점 : 여러 클래스의 속성과 메소드를 상속받아 재사용할 수 있다
- ☑ 단점 : 명칭 오류(name conflict) 가능성이 있다

상위-클래스에서 상속받는 클래스 또는 객체 멤버(즉, 데이터 속성과 메소드)의 검색 순서 : 깊이 우선, 좌에서 우로 검색(depth-first, left-to-right)

- 클래스이름을 먼저 검색 —> 검색이 안되면…
- 다음으로 상위-클래스-1을 검색 —> 검색이 안되면 반복(재귀)적으로 상위-클래스-1의 상위 클래스들을 순차적으로 검색 —> 그래도 검색이 안되면…
- 다음으로 상위-클래스-2을 검색 —> 검색이 안되면 반복(재귀)적으로 상위-클래스-2의 상위 클래스들을 순차적으로 검색 —> 그래도 검색이 안되면…
- 해당 클래스가 검색되거나 최종 상위 클래스에 도달할 때까지 같은 방식으로 검색을 계속
- 그래도 찾지 못하면, 예외가 발생

설명문자열

docstrings

설명문자열

```python
class PartTimer:
    """느티나무 카페에서 일하는 아르바이트생 클래스다.

    Attributes:
        Class Attributes:
            cafe_name (str): 커피 가게 이름
            hour_rate (int): 시급
        Instance Attributes:
            __nickname (str): 아르바이트생 닉네임
            __whours (int): 근무한 시간

    Methods:
        __init__(self, name: str) -> None:
        nickname(self) -> str:
        whours(self, hours_worked: int) -> None:
        total_wage(self) -> int:
    """
    # --- 클래스 속성 ---------------------------------------------------------- #
    cafe_name = '느티나무 카페'  # 클래스 속성 : 커피 가게 이름
    hour_rate = 9500           # 클래스 속성 : 시급

    # --- 초기화 메소드 -------------------------------------------------------- #
    def __init__(self, name):
        """인스턴스를 생성할 때 아르바이트생의 닉네임으로 초기화한다.

        Args:
            name (str): 아르바이트생의 닉네임
        """
        self.__nickname = name  # 인스턴스 속성 : 아르바이트생 닉네임
        self.__whours = 0       # 인스턴스 속성 : 근무한 시간
```

```python
    # --- 접근자 메소드 -------------------------------------------------------- #
    def nickname(self):
        """아르바이트생의 닉네임을 반환한다.

        Returns:
            str: 아르바이트생의 닉네임
        """
        return self.__nickname

    def whours(self, hours_worked):
        """아르바이트생의 근무한 시간을 기록한다.

        Args:
            hours_worked (int): 근무한 시간
        """
        self.__whours += hours_worked  # 기존 근무시간에 현 근무 시간을 추가해서 합한다.

    # --- 일반 메소드 ---------------------------------------------------------- #
    def total_wage(self):
        """지금까지 근무한 시간을 기준으로 계산한 급여를 반환한다.

        Returns:
            int: 현재까지 일한 급여
        """
        return self.__whours * PartTimer.hour_rate   # 총 근무시간 * 시급
```

설명문자열 사용

```
print(PartTimer.__doc__)
```

```
느티나무 카페에서 일하는 아르바이트생 클래스다.

    Attributes:
        Class Attributes:
            cafe_name (str): 커피 가게 이름
            hour_rate (int): 시급
        Instance Attributes:
            __nickname (str): 아르바이트생 닉네임
            __whours (int): 근무한 시간

    Methods:
        __init__(self, name: str) -> None:
        nickname(self) -> str:
        whours(self, hours_worked: int) -> None:
        total_wage(self) -> None:
```

설명문자열 사용

```
help(PartTimer)
```

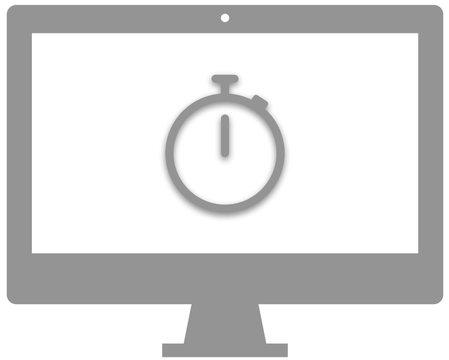
```
Help on class PartTimer in module __main__:

class PartTimer(builtins.object)
 |  PartTimer(name)
 |
 |  느티나무 카페에서 일하는 아르바이트생 클래스다.
 |
 |  Attributes:
 |      Class Attributes:
 |          cafe_name (str): 커피 가게 이름
 |          hour_rate (int): 시급
 |      Instance Attributes:
 |          __nickname (str): 아르바이트생 닉네임
 |          __whours (int): 근무한 시간
 |
 |  Methods:
 |      __init__(self, name: str) -> None:
 |      nickname(self) -> str:
 |      whours(self, hours_worked: int) -> None:
 |      total_wage(self) -> None:
 |
 |  Methods defined here:
 |
 |  __init__(self, name)
 |      인스턴스를 생성할 때 아르바이트생의 닉네임으로 초기화한다.
 |
 |      Args:
 |          name (str): 아르바이트생의 닉네임
 |
 |  nickname(self)
 |      아르바이트생의 닉네임을 반환한다.
 |
 |      Returns:
 |          str: 아르바이트생의 닉네임
 |
 |  total_wage(self)
 |      지금까지 근무한 시간을 기준으로 계산한 급여를 반환한다.
 |
 |      Returns:
 |          int: 현재까지 일한 급여
 |
 |  whours(self, hours_worked)
 |      아르바이트생의 근무한 시간을 기록한다.
 |
 |      Args:
 |          hours_worked (int): 근무한 시간
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  cafe_name = '느티나무 카페'
 |
 |  hour_rate = 9500
```

CHAPTER 12 실습

클래스와 객체

Lab Exercises

클래스와 객체 만들기



Lab : 아르바이트생 클래스 완성

- 아르바이트생 클래스 완성
  - 앞서 다룬 아르바이트생 클래스(`PartTimer`)를 일부 변경해서 구현한 후 '`parttimer.py`'로 저장
  - 클래스 속성
    - `hour_rate` : 시급(기본 시급은 `9,500`)
    - `total_parttimers` : 아르바이트 전체 수(인스턴스를 하나 생성할 때 마다 `1`씩 증가)
  - 인스턴스 속성 : 인스턴스를 생성할 때 닉네임과 근무지를 전달인자로 사용
    - `nickname` : 개별 아르바이트생의 닉네임
    - `workplace` : 근무지(기본값은 `'113동'`)
    - `whours` : 근무시간
  - 메소드
    - `__init__()` : 초기화 메소드
    - `nickname()` : 닉네임을 반환
    - `workplace()` : 근무지를 반환
    - `whours()` : 근무한 시간을 기록
    - `total_wage()` : 급여(시급 × 지금까지 근무한 시간)를 계산하여 반환
  - 이렇게 구현한 `PartTimer` 클래스를 `import`문 또는 `from...import`문으로 불러와 실행

Lab : 아르바이트생 클래스 완성

- 실행 결과 예시

```
>>> from parttimer import PartTimer
>>> park = PartTimer('라이언')
>>> lee = PartTimer('네오', '137-1동')
>>> park.nickname()
라이언
>>> lee.nickname()
네오
>>> park.workplace()
113동
>>> lee.workplace()
137-1동
>>> park.whours(4)
>>> lee.whours(3)
>>> park.whours(4)
>>> park.whours(3)
>>> lee.whours(3)
>>> park.total_wage()
104500
>>> lee.total_wage()
57000
>>> PartTimer.total_parttimers
2
```