

스크립트 프로그래밍

07 객체와 클래스

2016 2학기 (02분반)

강승우

학습 목표

- 객체와 클래스를 설명하고 객체를 설계하기위해 클래스를 사용할 수 있다 (§7.2).
- 데이터 필드와 메소드를 포함하여 클래스를 정의할 수 있다 (§7.2.1).
- 데이터 필드를 생성하고 초기화하기위한 초기자를 호출하는 생성자를 사용하여 객체를 구성할 수 있다 (§7.2.2).
- 점 연산자(.)를 사용하여 객체의 멤버에 접근할 수 있다 (§7.2.3).
- self 매개변수를 이용하여 객체 자기자신을 참조할 수 있다 (§7.2.4).
- 클래스와 객체를 설명하기위해 UML 그래픽 표기법을 사용할 수 있다 (§7.3).
- 변경 가능 객체와 변경 불가능 객체를 구분할 수 있다 (§7.4).
- 데이터 오류를 방지하고 클래스 관리의 편의성을 위해 데이터 필드를 은닉시킬 수 있다 (§7.5).
- 소프트웨어 개발에 클래스 추상화와 캡슐화 기법을 적용할 수 있다 (§7.6).
- 절차적 패러다임과 객체지향 패러다임의 차이점에 대하여 설명할 수 있다 (§7.7).

객체지향 프로그래밍의 개념

- 객체지향 프로그래밍(OOP: Object Oriented Programming)
 - 프로그램을 생성하기 위해 객체를 사용하는 것
 - **컴퓨터 프로그램**을 명령어의 목록으로 보는 것이 아닌, **객체들의 모임**으로 보는 것
 - 객체들 사이에 데이터를 주고 받고, 그 데이터를 처리하는 과정으로 프로그램이 구성됨

객체(object)

- 실세계에서 개별적으로 구분되는 개체를 표현
 - 예를 들어 학생, 교수, 책상, 의자, TV, 세탁기, 원, 버튼 그리고 심지어 대출까지도 객체로 볼 수 있음
- 유일한 식별자, 상태, 행동을 포함
- 객체의 상태(객체의 특성 또는 속성)
 - 데이터 필드(data field)라고 하는 변수로 표현됨
 - 인스턴스 변수
- 객체의 행동(행위)
 - 메소드(method)의 집합으로 정의됨
 - (인스턴스) 메소드

클래스

- 객체를 생성하기 위한 일종의 설계도/템플릿(template)/청사진(blueprint)
- 클래스로 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance)라고 함



출처: <http://sstorm.egloos.com/m/5539929>

- 하나의 붕어빵 틀로 여러 개의 붕어빵을 찍어내듯이 하나의 클래스로 여러 개의 객체를 생성하는 것


클래스 정의

class 클래스이름:

```
def 메소드1(self, ...):  
    ...
```

```
def 메소드2(self, ...):  
    ...
```

메소드 정의 (첫번째 매개변수로 self를 가짐)



- 클래스 내부에는 인스턴스 변수와 메소드 정의 → 클래스의 멤버
- 인스턴스 변수 생성
 - self.변수이름
- self
 - 객체 자신을 가리키는 변수
 - 클래스 정의에 포함된 객체의 멤버에 접근할 수 있음
 - 예: self.var / self.method()

클래스 예

- 경기장이나 콘서트에 입장하는 관객 수를 세기 위해 사용하는 계수기를 표현하는 클래스를 만들어보자
- Counter 클래스
 - 카운트 값을 저장하는 변수가 필요
 - 카운트 값을 0으로 초기화 하는 기능 필요
 - 카운트 값을 1씩 증가하는 기능 필요
 - 현재 카운트 값을 알 수 있게 해주는 기능 필요



Counter 클래스 예

- 클래스 정의

class Counter:

```
def reset(self):  
    self.count = 0
```

메소드 정의

```
def increment(self):  
    self.count += 1
```

인스턴스 변수 생성

```
def get(self):  
    return self.count
```


Counter 클래스 객체 생성 및 메소드 호출

- 객체 생성

```
a = Counter()
```

- 메소드 호출

```
a.reset()
```

```
a.increment()
```

```
print("카운터 a의 값은", a.get())
```

Counter 클래스 예제에서는 reset() 메소드를 호출하여야 인스턴스 변수 count가 생성되므로 다른 메소드를 호출하기 전에 reset() 메소드를 먼저 호출해야 함

- 객체는 원하는 만큼 여러 개 생성할 수 있음

```
b = Counter()
```

```
c = Counter()
```

생성자 (Constructor)

- 앞의 Counter 클래스의 문제
 - 인스턴스 변수(count)를 생성하기 위해서 reset() 메소드를 반드시 호출해야 함
 - → 객체가 생성될 때 자동으로 인스턴스 변수를 생성하고 초기화 할 수 있으면 편리할 것임
 - 생성자
 - 객체가 생성될 때 객체를 기본값으로 초기화하는 특수한 메소드
 - `__init__()`
 - 파이썬에서는 인스턴스 변수가 생성되는 곳이기도 함
 - 객체가 생성될 때 생성자는 자동으로 호출됨
 - 객체를 생성할 때 클래스 이름과 동일한 메소드를 호출하고 이때 `__init__()` 메소드가 실행됨
- a = Counter()

생성자

- Counter 클래스에 생성자 추가
class Counter:

```
    def __init__(self):  
        self.count = 0  
    def reset(self):  
        self.count = 0  
    def increment(self):  
        self.count += 1  
    def get(self):  
        return self.count
```

- 파이썬에서는 클래스 당 하나의 생성자만 허용
- 기본 인자값을 사용하는 매개변수를 이용할 수 있음
- 왼쪽 예제에서

```
def __init__(self, initValue = 0):  
    self.count = initValue
```

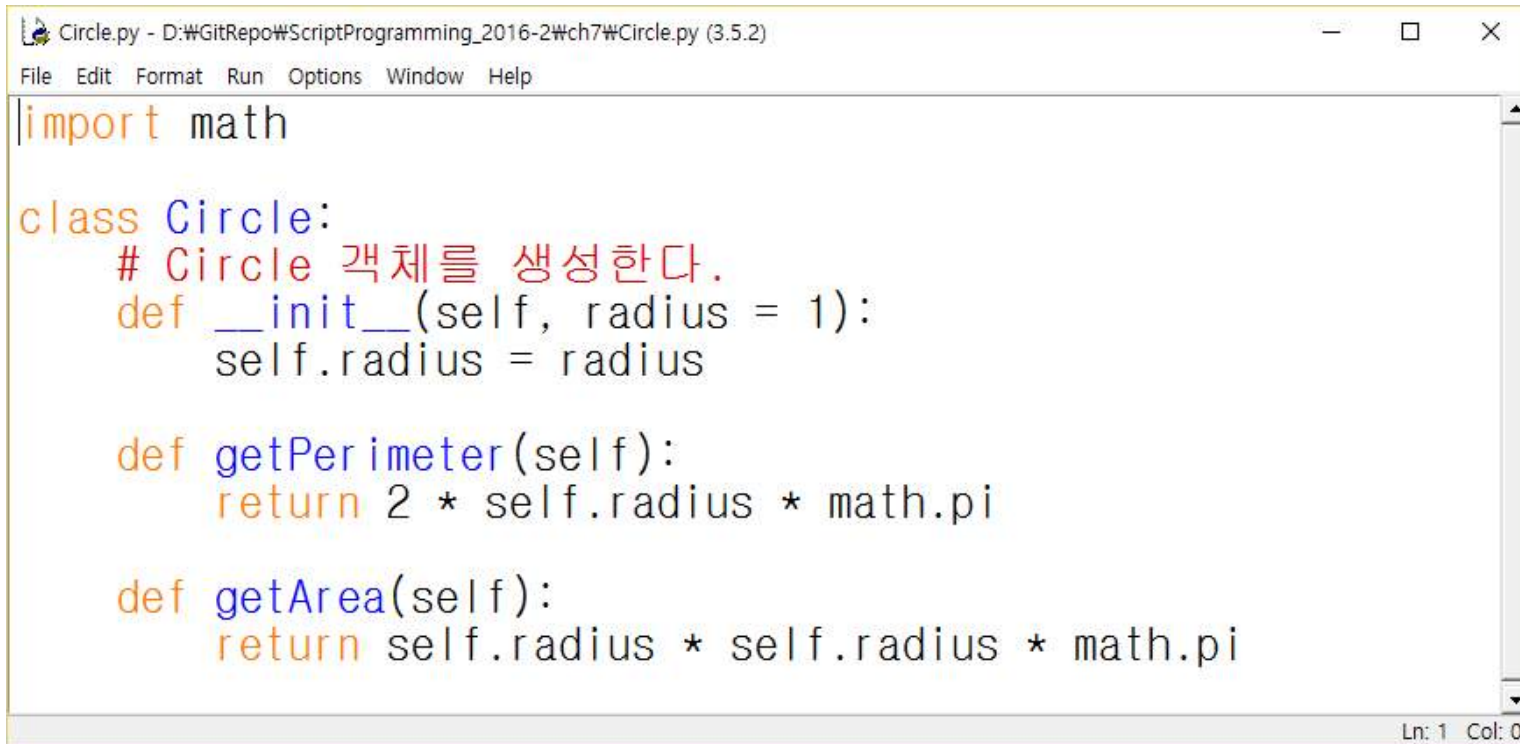
이 경우 아래와 같이 객체 생성 가능

```
a = Counter(100)
```

```
b = Counter()
```

클래스 정의 및 사용 예제

- 예제 코드
 - Circle.py
 - TestCircle.py



```
Circle.py - D:\GitRepo\ScriptProgramming_2016-2\ch7\Circle.py (3.5.2)
File Edit Format Run Options Window Help
import math

class Circle:
    # Circle 객체를 생성한다.
    def __init__(self, radius = 1):
        self.radius = radius

    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

Ln: 1 Col: 0
```

Circle 클래스를 모듈
화 했음

클래스 정의 및 사용 예제

객체 생성

객체 circle2의 인스턴스 변수 radius의 값 변경

```
*TestCircle.py - D:\GitRepo\ScriptProgramming_2016-2\ch7\TestCircle.py (3.5.2)*
File Edit Format Run Options Window Help
from Circle import Circle

def main():
    # Create a circle with radius 1
    circle1 = Circle()
    print("반지름이 ", circle1.radius,
          "인 원의 넓이는 ", circle1.getArea(), "입니다.")

    # Create a circle with radius 25
    circle2 = Circle(25)
    print("반지름이 ", circle2.radius,
          "인 원의 넓이는 ", circle2.getArea(), "입니다.")

    # Create a circle with radius 125
    circle3 = Circle(125)
    print("반지름이 ", circle3.radius,
          "인 원의 넓이는 ", circle3.getArea(), "입니다.")

    # Modify circle radius
    circle2.radius = 100
    print("반지름이 ", circle2.radius,
          "인 원의 넓이는 ", circle2.getArea(), "입니다.")

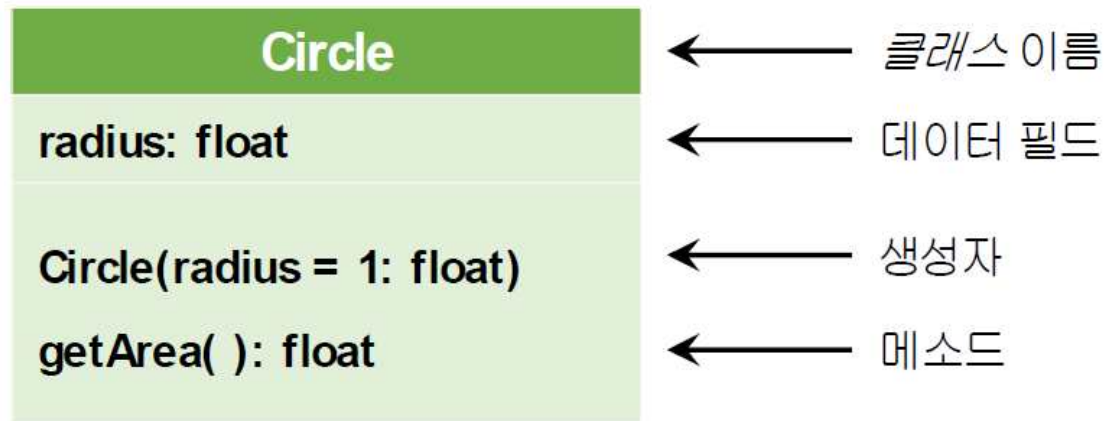
main() # Call the main function
```

Circle 모듈에서 Circle 클래스를
임포트하여 사용

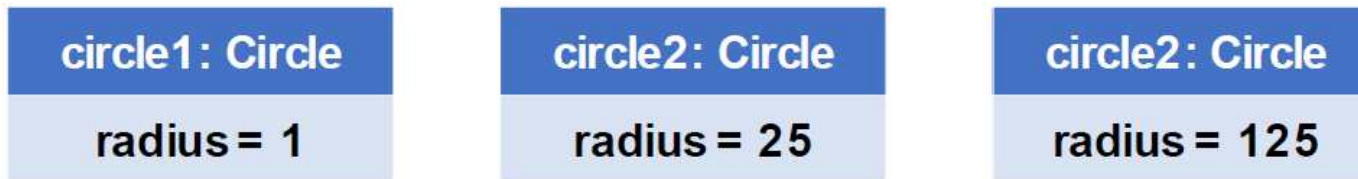
Ln: 8 Col: 0

UML 클래스 다이어그램

UML 클래스 다이어그램



객체 UML 표기법



UML 클래스 다이어그램

- 데이터 필드 (멤버 변수)
 - 데이터 필드 이름: 데이터 필드 타입
- 생성자
 - 클래스 이름(매개변수 이름: 매개변수 타입)
- 메소드
 - 메소드 이름(매개변수 이름: 매개변수 타입): 반환 타입
- 클래스를 이용하는 방법을 알려줌
 - 객체를 어떻게 생성하고
 - 객체의 메소드를 어떻게 호출하는지 설명

TV를 표현하는 클래스 정의

- 먼저 TV 클래스를 정의하기 위해 필요한 TV의 속성과 행위를 생각한다
 - TV는 켜고 끌 수 있다,
 - 채널을 변경할 수 있다,
 - 볼륨을 조절할 수 있다, ...
- 속성
 - 전원 on/off 상태
 - 현재 채널 (1-120)
 - 현재 볼륨 (1-7)
- 행위
 - TV 전원을 켜다/끄다
 - 채널을 하나 높인다/낮춘다, 새로운 채널을 설정한다, 현재 채널을 확인한다
 - 볼륨을 하나 높인다/낮춘다, 현재 볼륨을 확인한다
 - ...

TV 클래스의 UML 다이어그램

TV	
channel: int volumeLevel: int on: bool	현재 채널(1~120) 현재 음량 크기(1~7) TV의 전원 on/off 상태
TV() turnOn(): None turnOff(): None getChannel(): int setChannel(channel: int): None getVolume(): int setVolume(volumeLevel: int): None channelUp(): None channelDown(): None volumeUp(): None volumeDown(): None	기본 TV 객체 생성한다 전원을 켜다 전원을 끈다 현재 채널을 반환한다 새로운 채널을 설정한다 현재 음량 크기를 얻는다 음량 크기를 설정한다 한 채널을 높인다 한 채널을 낮춘다 음량 크기를 1 높인다 음량 크기를 1 낮춘다

TV 클래스의 구현

```
class TV:
    def __init__(self):
        self.channel = 1 # 기본 채널은 1
        self.volumeLevel = 1 # 기본 음량 크기는 1
        self.on = False # 초기에 TV의 전원 off
    def turnOn(self):
        self.on = True
    def turnOff(self):
        self.on = False
    def getChannel(self):
        return self.channel
    def setChannel(self, channel):
        if self.on and 1 <= self.channel <= 120:
            self.channel = channel
    def getVolumeLevel(self):
        return self.volumeLevel
```

```
    def setVolume(self, volumeLevel):
        if self.on and 1 <= self.volumeLevel <= 7:
            self.volumeLevel = volumeLevel
    def channelUp(self):
        if self.on and self.channel < 120:
            self.channel += 1
    def channelDown(self):
        if self.on and self.channel > 1:
            self.channel -= 1
    def volumeUp(self):
        if self.on and self.volumeLevel < 7:
            self.volumeLevel += 1
    def volumeDown(self):
        if self.on and self.volumeLevel > 1:
            self.volumeLevel -= 1
```

TV 클래스의 이용

```
from TV import TV
```

```
def main():
```

```
    tv1 = TV()
```

```
    tv1.turnOn()
```

```
    tv1.setChannel(30)
```

```
    tv1.setVolume(3)
```

```
    tv2 = TV()
```

```
    tv2.turnOn()
```

```
    tv2.channelUp()
```

```
    tv2.channelUp()
```

```
    tv2.volumeUp()
```

```
    print("tv1의 채널은", tv1.getChannel(),  
          "이고 음량 크기는", tv1.getVolumeLevel(), "입니다.")  
    print("tv2의 채널은", tv2.getChannel(),  
          "이고 음량 크기는", tv2.getVolumeLevel(), "입니다.")
```

```
main() # main 함수를 호출한다.
```

정보 은닉 (data/information hiding)

- 앞의 Circle 클래스의 예에서
c = Circle(5)
c.radius = 5.4 # 인스턴스 변수에 직접 접근
print(c.radius) # 인스턴스 변수에 직접 접근
- 위와 같은 직접 접근의 문제
 - 인스턴스 변수의 값이 올바르게 않게 변경될 수 있음
 - 예를 들어 반지름이 음수가 되도록 할 수 있음
 - 클래스 관리하고 유지보수 하기 어렵게 만듦
 - Circle 클래스를 사용하여 프로그램을 만들었는데, 반지름이 음수가 되지 않도록 클래스를 수정하려고 할 때, 이미 Circle 클래스 사용 프로그램에서 반지름이 음수가 되도록 변수값을 변경해버린 경우, Circle 클래스 뿐만 아니라 이 프로그램도 수정해야 함

정보 은닉 (data/information hiding)

- 정보 은닉
 - 외부에서 객체의 인스턴스 변수를 직접 접근할 수 없도록 하는 것
 - `private` 변수로 정의
- `private` 변수
 - 변수 이름 앞에 밑줄 두 개를 추가
 - 클래스 내부에서만 접근 가능
- `private` 메소드도 동일하게 가능
- 본인이 작성한 프로그램에서만 내부적으로 사용되는 클래스의 경우, 은닉이 반드시 필요한 것은 아님

Circle 클래스의 수정 – private 변수

```
CircleWithPrivateRadius.py - D:\GitRepo\ScriptProgramming_2016-2\ch7\CircleWithPrivateRadius.py (3.5.2)
File Edit Format Run Options Window Help
import math

class CirclePrivate:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def getPerimeter(self):
        return 2 * self.__radius * math.pi

    def getArea(self):
        return self.__radius * self.__radius * math.pi

    def setRadius(self, radius):
        if radius >= 0:
            self.__radius = radius
```

- 아래와 같이 하면 어떻게 될까?

```
c = Circle(5)
c.__radius
```

접근자(getters)와 설정자(setters)

- private 변수와 메소드는 클래스 내부에서만 접근 가능
- 외부에서 이 변수 값을 이용해야 할 경우에는 어떻게 할까?
- 접근자와 설정자 이용
 - 접근자
 - 인스턴스 변수값을 반환하는 메소드
 - 설정자
 - 인스턴스 변수값을 설정하는 메소드
 - 일반적으로 get, set으로 시작
- 앞의 Circle 클래스 예제에서

```
c = Circle(5)  
c.getRadius()
```

접근자와 설정자 사용 이유

- 나중에 클래스를 수정할 때 용이함
- 설정자에서 매개 변수를 통해서 잘못된 값이 넘어오는 경우, 이를 차단할 수 있음
- 필요할 때마다 인스턴스 변수값을 계산하여 반환할 수 있음
- 어떤 변수에 대해 접근자만 제공하면 자동으로 읽기만 가능한 인스턴스 변수를 만드는 것이 됨

캡슐화 (encapsulation)

- 데이터와 메소드를 하나의 객체로 묶고 사용자로부터 데이터 필드와 메소드 구현을 감추는 것
 - 사용자는 공개된 인터페이스(메소드)가 무엇이고 그것을 어떻게 사용하는지 알면 내부 구현 내용을 알지 못해도 객체를 사용할 수 있음

객체지향적으로 생각하기

- 절차적 프로그래밍
 - 행동/행위 기반 방식 → 함수 설계에 초점
 - 데이터와 행위가 분리
 - 데이터를 함수에 전달하도록 요구함
 - 객체지향 프로그래밍
 - 객체와 객체의 연산에 초점
 - 데이터와 행위(메소드)를 객체에 결합
 - 객체의 속성과 행동이 연결되어 있는 실세계를 반영하는 방식으로 프로그램을 구성하는 것
- ✓객체의 사용은 소프트웨어의 재사용성을 높이고, 프로그램 개발과 유지를 쉽게 만든다

객체지향적으로 생각하기

- 코드 4.6 ComputeBMI.py의 사례
 - 다른 프로그램에서 BMI 계산하는 기능을 사용하고자 해도
 - 이 코드 자체를 다른 프로그램에서 재사용할 수 없음
- 키와 몸무게를 매개변수로 입력 받아 BMI를 계산하는 함수를 정의하여 사용할 수 있음
 - `def getBMI(weight, height):`
 - 이를 모듈화 하면 이 함수를 다른 프로그램에서 재사용할 수 있음
- 하지만, 여러 사람의 BMI 정보를 동시에 관리하기 위해서, 몸무게와 키, BMI를 각 사람의 이름 및 생일과 연결시켜야 하는 경우 어떻게 할 수 있을까?

객체지향적으로 생각하기

- 관련 데이터를 포함하는 객체를 생성
- BMI 클래스 정의

BMI	
-name: str -age: int -weight: float -height: float	이름 나이 몸무게(파운드 단위) 키(인치 단위)
BMI(name: str, age: int, weight: float, height: float)	특정 이름, 몸무게, 키 및 나이(기본값 20)를 사용하여 BMI 객체를 생성한다
getBMI(): float getStatus(): str	BMI를 반환한다. BMI 상태를 반환한다(예, 표준, 과체중 등)

이에 대한 getter 메소드는 생략