10장 객체와 클래스

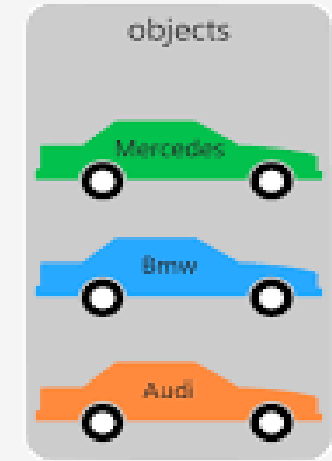
이 장의 내용

- 10.1 객체지향 프로그래밍
- 10.2 클래스 정의
- 10.3 객체 생성 및 메소드 호출
- 10.4 객체 변수와 클래스 변수
- 10.5 캡슐화
- 10.6 상속

10.1 객체지향 프로그래밍

객체지향 개념

- 객체지향 프로그램
  - 실세계에 있는 객체들을 프로그램 상에 표현하고
  - 이들 사이의 상호작용을 시뮬레이션하기 위한 것이다.
- 예
  - 인터넷 뱅킹, 수강 신청, 자동차, ...



객체

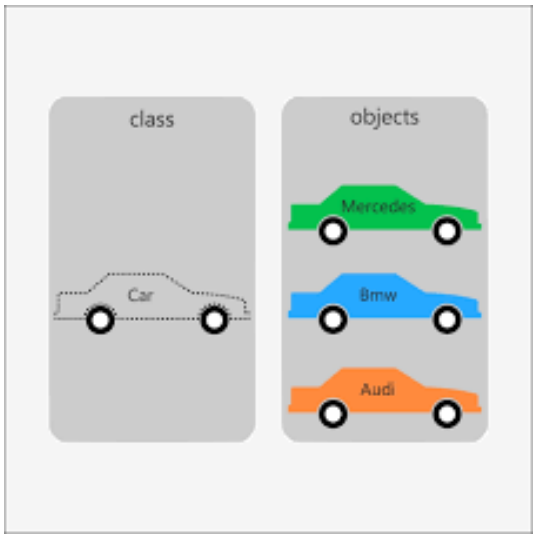
- 객체
  - 객체의 특성을 나타내는 속성(attribute)과
  - 객체에 대해서 할 수 있는 행동(behavior)으로 정의됨

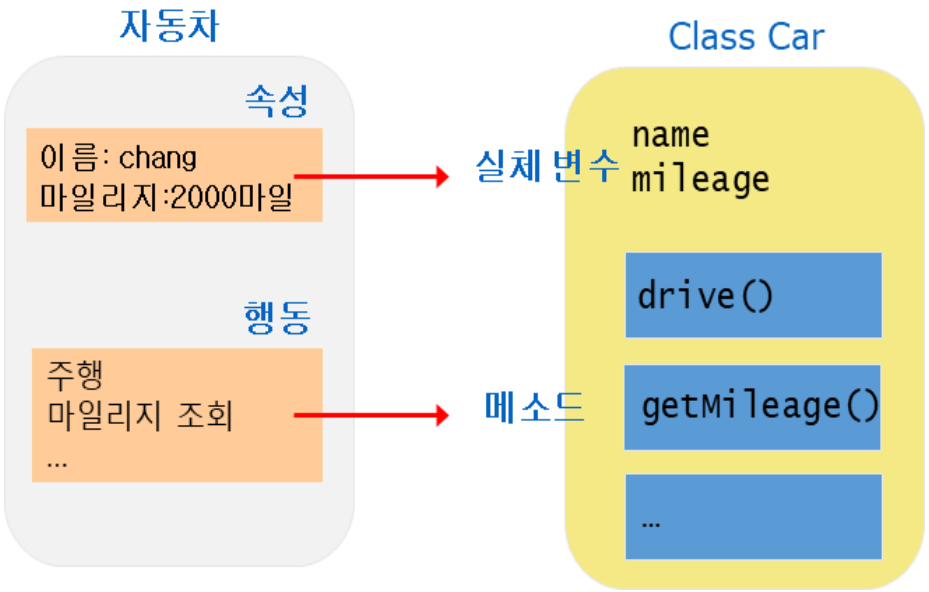
- 메소드
  - 객체를 대상으로 할 수 있는 행동들을 나타낸다
  - 객체의 메소드는 함수로 정의된다.

- 메소드 호출
  - 점 표기법으로 호출하며 정의된 함수 내의 코드를 실행한다.
  - 함수와 마찬가지로 메소드 호출시 인수를 전달 할 수 있다.

클래스

- 클래스
  - 객체를 정의한 것 : 객체의 속성과 행동을 정의
  - 객체에 대한 설계도
  - 객체지향 언어에서 일종의 자료형(data type)



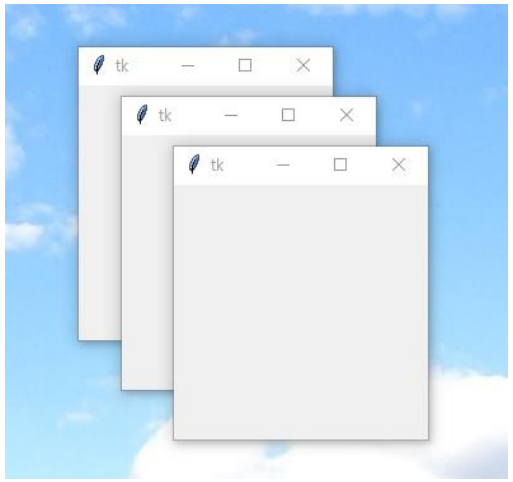


객체의 생성과 사용

- 실체화(instantiation)
  - 클래스로부터 객체들을 생성해내는 것을 실체화
  - 객체는 클래스의 하나의 실체라고 한다

- 예

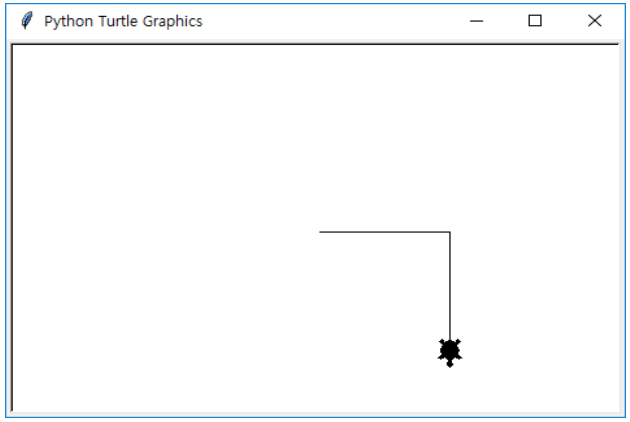
```
from tkinter import *
window1 = Tk()      # 윈도우 객체를 생성한다.
window2 = Tk()      # 윈도우 객체를 생성한다.
window3 = Tk()      # 윈도우 객체를 생성한다.
```



터틀 그래픽 예제

- 터틀 그래픽
  - 거북이 객체를 생성하여 이 객체를 이용하여 그림을 그릴 수 있다.
- 예

```
from turtle import *
t = Turtle()            # 거북 객체를 생성한다.
t.forward(100)          # forward는 거북 객체의 메소드
t.right(90)             # right는 거북 객체의 메소드
t.forward(100)
```



리스트 예제

- 리스트도 일종의 객체
- 리스트에 append, extend, insert, sort 등의 메소드 사용

```
>>> word = []
>>> word.append('I')
>>> word.extend(['love', 'programming’])
>>> word.insert(2, ‘파이썬’)
>>> word
['I', 'love', '파이썬', 'programming’]
>>> word.sort()
>>> word
['I', 'love', 'programming', '파이썬']
>>> word.reverse()
>>> word
['파이썬', 'programming', 'love', 'I']
```

10.2 클래스 정의

클래스의 정의

- 클래스
  - 객체의 속성과 메소드를 정의한 것이 클래스
  - 객체의 설계도와 같은 개념으로 볼 수 있다.

- 클래스는 class 예약어를 사용하여 정의된다.
  - 구문법

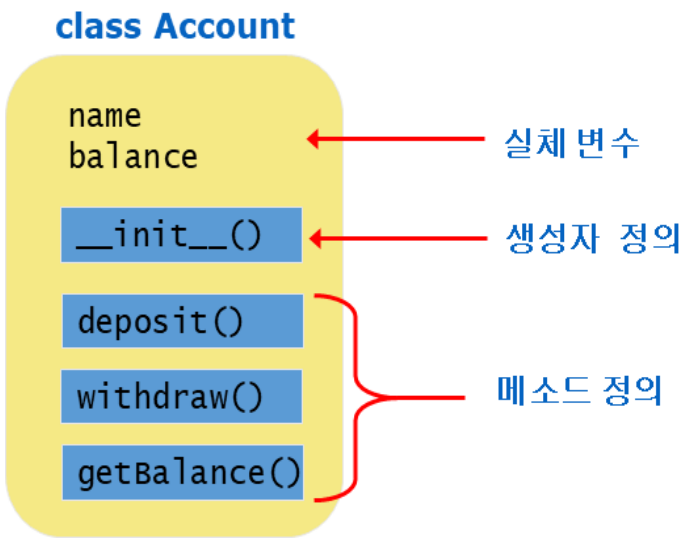
```
class 클래스명:
    def __init__(self, 매개변수리스트):
        ...

    def 메소드명(self, 매개변수리스트):
        ...
```

  - 의미

    클래스명을 갖는 새로운 클래스를 정의한다.

Account 클래스 정의




```
class Account:
    def __init__(self, name):
        self.name = name
        self.balance = 0

    def getBalance(self):
        return self.balance

    def deposit(self, amount):
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("잔액 부족")
        return self.balance
```

생성자 __init__ 메소드

- 객체의 실체변수를 __init__ 메소드를 사용하여 초기화 한다.

```
def __init__(self, name):
    self.name = name
    self.balance = 0

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

10.3 객체 생성 및 메소드 호출

객체의 생성

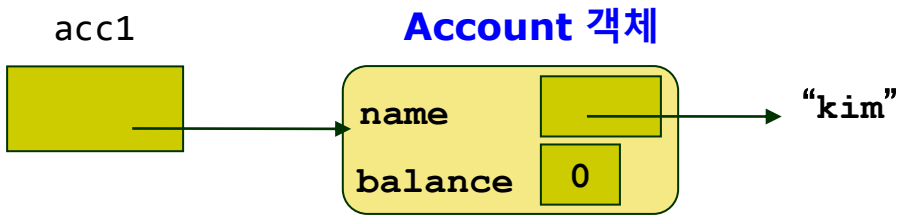
- 구문법

  변수 = 클래스명(인수리스트)

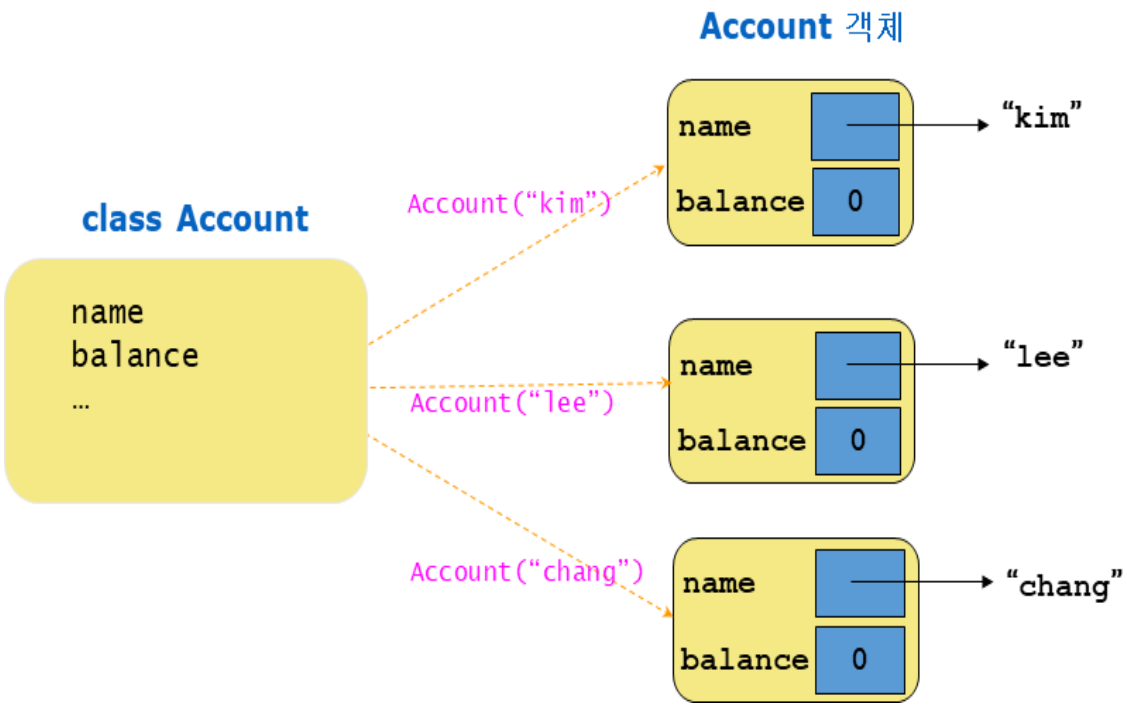
- 의미

  클래스명을 갖는 클래스의 객체를 생성하고 이를 변수가 참조한다.

- 예
  - acc1 = Account('kim')
  - 객체를 생성하고 변수에 배정
  - 변수가 생성된 객체를 참조한다.



클래스와 객체


Account 객체
class Account
name
balance
…
Account("kim")
Account("lee")
Account("chang")
name
balance
0
"kim"
name
balance
0
"lee"
name
balance
0
"chang"

필드 변수/메소드 호출

- 구문법

  (1) 참조변수.변수

  (2) 참조변수.메소드(인수리스트)

- 의미

  참조변수가 객체의
  (1) 필드 변수에 접근
  (2) 메소드를 호출한다.

```
>>> acc1.deposit(100000)

>>> acc1.getBalance()

100000

>>> acc1.name

'kim'

>>> acc1.balance

100000
>>> acc2 = Account('lee')
>>> acc2.deposit(200000)
>>> acc2.getBalance()
200000
>>> acc3 = Account('chang')
>>> acc3.deposit(300000)
>>> acc3.getBalance()
300000
```

10.4 객체 변수와 클래스 변수

객체 변수와 클래스 변수

- 객체 변수(object variable)
  - 각 객체마다 이 변수를 위한 기억 공간이 별도로 존재하는 필드 변수
  - 실체 변수(instance variable)
- 클래스 변수(class variable)
  - 클래스에 하나 존재하여 그 클래스의 모든 객체가 공유하는 변수
  - 정적 변수(static variable)

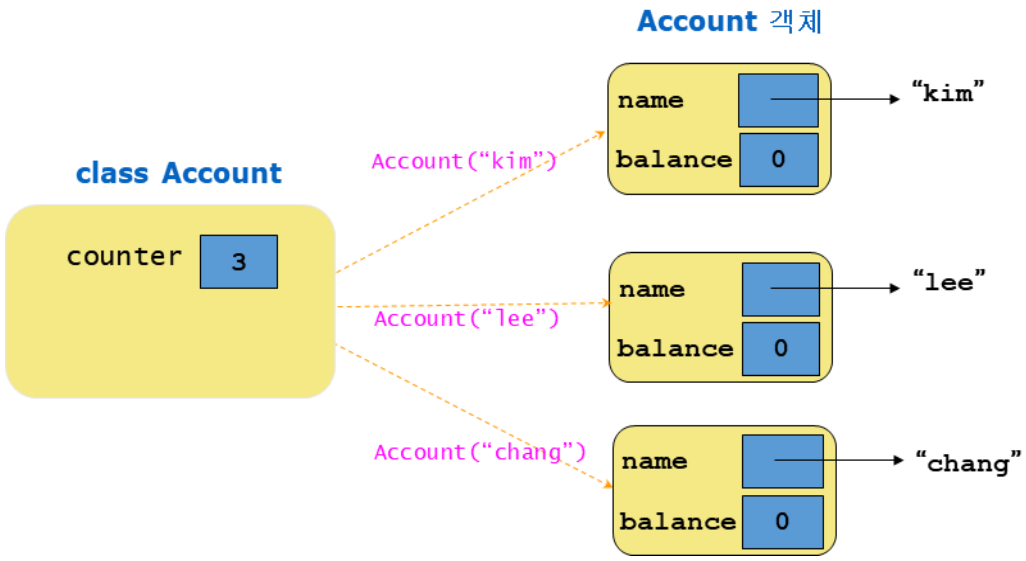
```
class Account:
    counter = 0
    def __init__(self, myname):
        self.name = myname
        self.balance = 0
        Account.counter += 1

    def __del__(self):
        Account.counter -= 1

    def getCounter(self):
        return Account.counter
...
```

예제

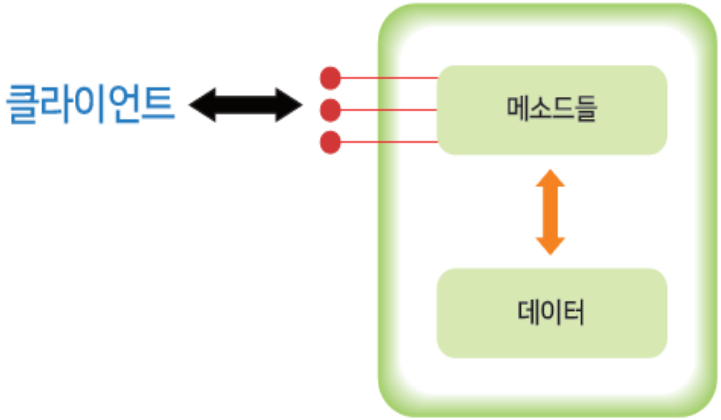
```
>>> kim = Account('kim')
>>> lee = Account('lee')
>>> chang = Account('chang')
>>> kim.getCounter()
3
>>> Account.counter
3
```



10.5 캡슐화

캡슐화의 필요성

- 추상 자료형(abstract data type)
  - 데이터(자료구조)와 관련된 연산(메소드)들을 한데 묶어 캡슐화하여 정의한 자료형
- 클래스
  - 데이터(자료구조)와 관련된 연산(메소드)들을 한데 묶어 캡슐화하여 정의한 일종의 추상 자료형이다.
- 캡슐화(encapsulation)
  - 데이터와 관련된 메소드들을 함께 선언하고 이 메소드들만 데이터에 접근하게 함.



접근 제어

- 캡슐화를 지원하기 위한 3가지 접근 제어(access modifier)
  - 공용(public) 접근 제어 : 기반, 파생, 외부 클래스 접근
  - 보호(protected) 접근 제어 : 기반클래스, (상속받은) 파생클래스 접근
  - 전용(private) 접근 제어 : 기반 클래스의 메소드 접근

공용(public)	전용(private)	보호(protected)
언더바로 시작하지 않는 이름	두 개의 언더바 __로 시작하는 이름	한 개의 언더바 _로 시작하는 이름

예제

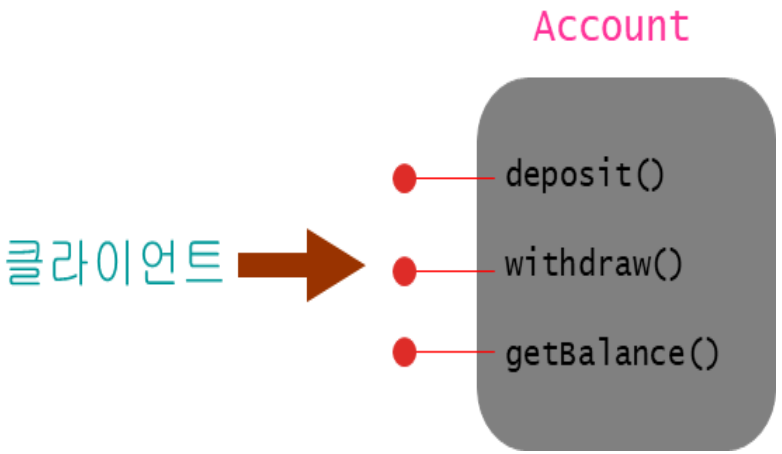
```
class Account:
   def __init__(self, name):
      self.__name = name
      self.__balance = 0

   def getBalance(self):
      return self.__balance

   def deposit(self, amount):
      self.__balance += amount
      return self.__balance

   ...
```

```
>>> my = Account('kim')
>>> my.__balance
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'Account' object has no
attribute '__balance'
>>> my.deposit(300000)
>>> my.getBalance()
300000
```



객체 출력을 위한 __str__() 메소드

- __str__() 메소드
  - 객체 정보 출력을 위한 특수 메소드
  - 출력하고자 하는 내용을 문자열 형태로 리턴

```
>>> print(acc1)
<__main__.Account object at 0x000001A2ACC7CC48>

>>> def __str__(self):

    msg = self.name + “의 계좌”

    return msg
```

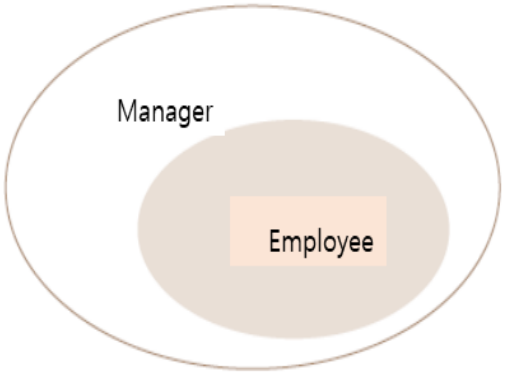
10.6 상속

상속

- 기존 클래스를 상속받아 새로운 클래스를 정의하는 것
  - 기존 클래스는 **부모 클래스** 또는 **슈퍼클래스**
  - 상속받아 새로 정의된 클래스는 **자식 클래스** 또는 **서브클래스**
  - 자식 클래스가 부모 클래스의 기능을 상속받아 쓰는 것!
  - 자식 클래스는 부모 클래스를 확장한(extend) 클래스!

상속의 예

- is-a 관계
  - 부모 클래스와 자식 클래스 사이에는 반드시 is-a 관계가 성립해야 한다.
  - 자식 클래스는 부모 클래스보다 구체적인 버전
  - (“The child is a more specific version of the parent”)
- 예
  - Manger is an employee



단일 상속

- 구문법

  class 자식클래스(부모클래스):

    ...

- 의미

  부모클래스를 상속받아 새로운 자식클래스를 정의한다

```
class Employee: # 일반 직원(Employee)
   def __init__(self, name, salary):
      self.name = name;
      self.salary = salary;

   def pay(self):
      return self.salary

class Manager(Employee): # 관리자(Manager)
   def __init__(self, name, salary, bonus):
      super().__init__(name, salary)
      self.bonus = bonus

   def pay(self):
      return self.salary + self.bonus

   def getBonus(self):
      return self.bonus
```

메소드 재정의

- 메소드 재정의(method overriding)
  - 자식 클래스는 부모로부터 상속받은 메소드를
  - 자신이 원하는 대로 재정의할 수 있다.
- 메소드 호출
  - 클래스 안에서 호출된 이름의 메소드를 먼저 찾고
  - 존재하지 않으면 부모 클래스의 메소드를 찾게 된다.
  - 재정의된 메소드가 실행된다.

```
>>> e = Employee('kim', 300)
>>> e.pay()
300
>>> e = Manager('lee', 300, 100)
>>> e.pay()
400
```

상속 예제

- Tultle 클래스를 상속 받는 my_turtle 클래스

```
from turtle import *
class my_turtle(Turtle):
  def set_turtle(self, color):
    self.color(color)
    self.shape('turtle')
```

- my_turtle 클래스에서는 컬러를 변경하고 화살표 모양을 거북으로 바꾸어 줄 수 있다.

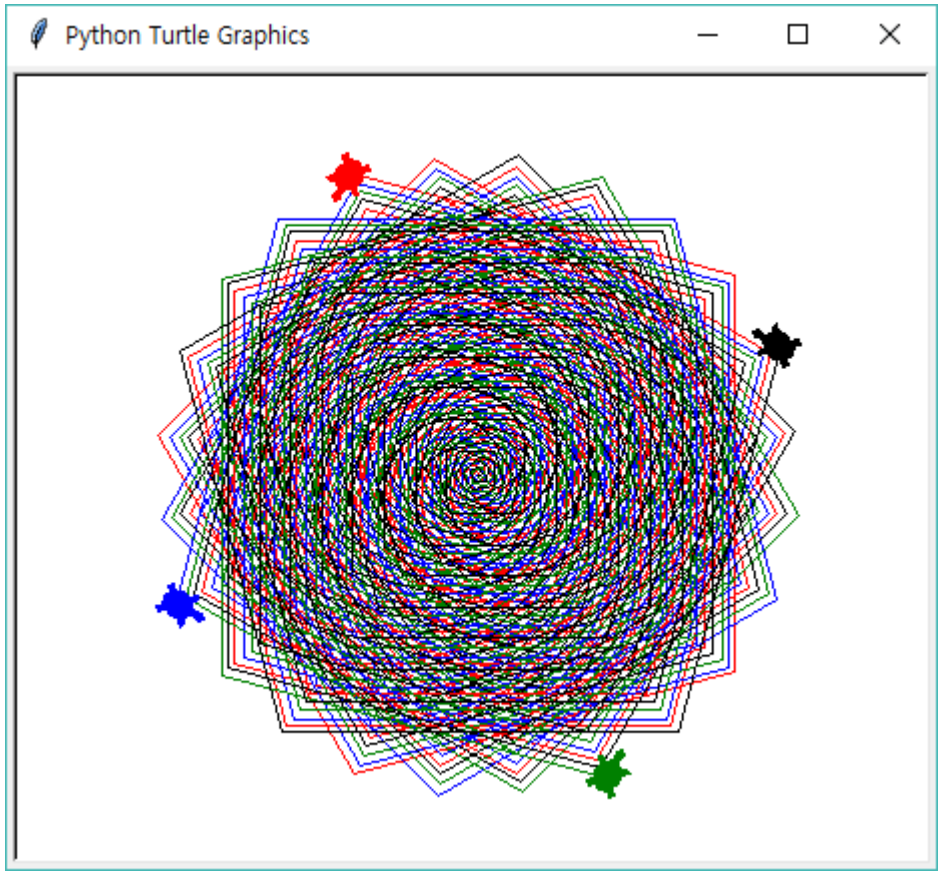
상속 예제(my_turtle)

```
# 다른 색을 갖는 네 마리 거북이 객체를 움직여 그리는 반복 사각형
t1 = my_turtle()
t2 = my_turtle()
t3 = my_turtle()
t4 = my_turtle()
t1.set_turtle('Red')
t2.set_turtle('Blue')
t3.set_turtle('Green')
t4.set_turtle('Yellow')
t2.left(90)
t3.left(180)
t4.right(90)
```

```
angle=75
t1.speed(0)
t2.speed(0)
t3.speed(0)
t4.speed(0)
for x in range(200):
    t1.forward(x)
    t2.forward(x)
    t3.forward(x)
    t4.forward(x)
    t1.left(angle)
    t2.left(angle)
    t3.left(angle)
    t4.left(angle)
```

상속 예제(my_turtle)

- 실행화면



다중 상속

- 다중 상속(multiple inheritance)
  - 여러 개의 클래스를 상속 받는 것

    - 구문법

      class 자식클래스(부모클래스1, …, 부모클래스N):

      …

    - 의미

      여러 부모클래스를 상속받아 새로운 자식클래스를 정의한다.

다중 상속 예

```
>>> class Person:
    def sleep(self):
        print('잠을 잡니다.')

>>> class Student(Person):
    def study(self):
        print('공부합니다.')
    def play(self):
        print('친구와 놉니다.')

>>> class Worker(Person):
    def work(self):
        print('일합니다.')
    def play(self):
        print('술을 마십니다.')
```

```
>>>class Arbeit(Student, Worker):
    def myjob(self):
        print('나는 알바 학생입니다:')
        self.sleep()
        self.play()
        self.study()
        self.work()

>>> a = Arbeit()
>>> a.myjob()
나는 알바 학생입니다:
잠을 잡니다.
친구와 놉니다.
공부합니다.
일합니다.
```