

객체지향 프로그래밍

학습목표

- 객체지향의 개념을 익힌다.
- 클래스, 인스턴스를 구분하고 이해한다.
- 객체지향 프로그래밍의 활용법을 익힌다.

SECTION 01 이 장에서 만들 프로그램

SECTION 02 클래스

SECTION 03 생성자

SECTION 04 인스턴스 변수와 클래스 변수

SECTION 05 클래스의 상속

SECTION 06 객체지향 프로그래밍의 심화 내용

요약

연습문제

응용예제



Section02 클래스

■ 클래스의 개념

- 클래스의 모양과 생성

```
class 클래스명 :  
    # 이 부분에 관련 코드 구현
```

- 현실 세계의 사물을 컴퓨터 안에서 구현하려고 고안된 개념
- 자동차를 클래스로 구현



```
class 자동차 :  
    # 자동차의 속성  
    색상  
    속도  
    # 자동차의 기능  
    속도 올리기()  
    속도 내리기()
```

그림 12-1 자동차를 클래스로 구현

Section02 클래스

- 자동차 클래스의 개념을 실제 코드로 구현
 - 자동차의 속성은 지금까지 사용 한 변수처럼 생성(필드(Field))
 - 자동차의 기능은 지금까지 사용한 함수 형식으로 구현
 - 클래스 안에서 구현된 함수는 함수라고 하지 않고 메서드라고 함.

```
class Car :  
    # 자동차의 필드  
    색상 = ""  
    현재_속도 = 0  
  
    # 자동차의 메서드  
    def upSpeed(증가할_속도량) :  
        # 현재 속도에서 증가할_속도량만큼 속도를 올리는 코드  
  
    def downSpeed(감소할_속도량) :  
        # 현재 속도에서 감소할_속도량만큼 속도를 내리는 코드
```

Section02 클래스

- 자동차 클래스를 완전한 파이썬 코드로

Code12-01.py

```
1 class Car :  
2     color = ""  
3     speed = 0  
4  
5     def upSpeed(self, value) :  
6         self.speed += value  
7  
8     def downSpeed(self, value) :  
9         self.speed -= value
```

5 ~ 6 행 : self . speed는 3행의 speed 의미
즉 자신의 클래스에 있는 speed 변수

출력 결과

아무것도 나오지 않음

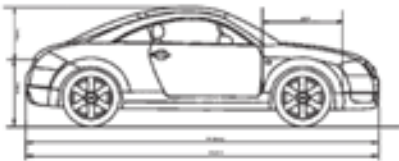
Section02 클래스

- Code12 - 01 . py 에 메서드 추가

```
def printMessage() :  
    print("시험 출력이다.")
```

- printMessage () 안에서는 필드를 사용하지 않으므로 이때는 self 생략이 가능
- 인스턴스의 생성(예 : 실제 생산되는 자동차)

자동차 설계도(클래스)



여러 번
찍어 내기



자동차(인스턴스)



그림 12-2 클래스와 인스턴스의 개념

Section02 클래스

- 인스턴스 구현 형식

자동차 설계도(클래스)

```
class 자동차 :  
    # 자동차의 속성  
    색상  
    속도  
    # 자동차의 기능  
    속도 올리기()  
    속도 내리기()
```

자동차(인스턴스)

```
자동차1 = 자동차()  
자동차2 = 자동차()  
자동차3 = 자동차()
```

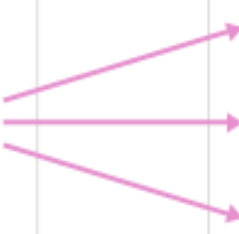


그림 12-3 클래스와 인스턴스 코드 구성

- 자동차 세 대의 인스턴스 생성 코드

```
myCar1 = Car()  
myCar2 = Car()  
myCar3 = Car()
```

Section02 클래스

- 필드에 값 대입



색상
속도



빨강
0km



색상
속도



파랑
0km



색상
속도



노랑
0km

그림 12-4 인스턴스의 필드에 값을 대입하는 개념

```
myCar1.color = "빨강"  
myCar1.speed = 0  
myCar2.color = "파랑"  
myCar2.speed = 0  
myCar3.color = "노랑"  
myCar3.speed = 0
```

Section02 클래스

- 메서드의 호출

```
myCar1.upSpeed(30)
```

```
myCar2.downSpeed(60)
```


Section02 클래스

■ 클래스의 완전한 작동 구현

Code12-02.py

```
1  ## 클래스 선언 부분 ##
2  class Car :
3      color = ""
4      speed = 0
5
6      def upSpeed(self, value) :
7          self.speed += value
8
9      def downSpeed(self, value) :
10         self.speed -= value
11
12  ## 메인 코드 부분 ##
13  myCar1 = Car()
14  myCar1.color = "빨강"
15  myCar1.speed = 0
16
17  myCar2 = Car()
18  myCar2.color = "파랑"
19  myCar2.speed = 0
20
```

2 ~ 10행 : Car 클래스 정의

3 ~ 4 행 : 자동차의 색상과 속도 필드 정의

6 ~ 7행, 9 ~ 10 행 : 매개변수로 추가 속도(value)를 받아 현재
속도(self . speed) 증가 또는 감소

13 ~ 15 행, 17 ~ 19 행, 21 ~ 23 행 : myCar1 ,
myCar2 , myCar3 인스턴스 생성하고, 색상과 속도 지정

Section02 클래스

```
21 myCar3 = Car()
22 myCar3.color = "노랑"
23 myCar3.speed = 0
24
25 myCar1.upSpeed(30)
26 print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.color, myCar1.speed))
27
28 myCar2.upSpeed(60)
29 print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.color, myCar2.speed))
30
31 myCar3.upSpeed(0)
32 print("자동차3의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar3.color, myCar3.speed))
```

25행 : myCar1 의 upSpeed (30) 메서드 호출하면 myCar1 의 필드 중 speed 필드가 30 으로 증가
26행 : myCar1 의 color 와 speed 필드 출력, 같은 방식으로 myCar2 는 28 ~ 29 행, myCar3 는 31 ~ 32 행에서 사용

출력 결과

자동차1의 색상은 빨강이며, 현재 속도는 30km입니다.
자동차2의 색상은 파랑이며, 현재 속도는 60km입니다.
자동차3의 색상은 노랑이며, 현재 속도는 0km입니다.

Section02 클래스

■ 클래스 사용 순서

단계	작업	형식	예
1단계	클래스 선언	<pre>class 클래스명: # 필드 선언 # 메서드 선언</pre>	<pre>class Car : color = " def upSpeed(self, value) : ...</pre>



2단계	인스턴스 생성	<pre>인스턴스 = 클래스명()</pre>	<pre>myCar1 = Car()</pre>
-----	---------	--------------------------	---------------------------



3단계	필드나 메서드 사용	<pre>인스턴스.필드명 = 값 인스턴스.메서드()</pre>	<pre>myCar1.color = "빨강" myCar1.upSpeed(30)</pre>
-----	------------	--	---

SELF STUDY 12-1

Code12-02.py의 upSpeed() 함수를 수정해서 속도가 150이 넘으면 최대 150으로 조절해 보자. 예를 들어 upSpeed(200)을 사용해도 출력은 150km가 되어야 한다.

힌트 if 문을 활용한다.

Section03 생성자

■ 생성자의 개념 : 인스턴스를 생성하면서 필드값을 초기화시키는 함수

■ 생성자의 기본

- 생성자의 기본 형태 : `__init__()`라는 이름

Tip • `__init__()`는 앞뒤에 언더바(_)가 2개씩, `init` 는 Initialize 의 약자로 초기화 의미
언더바가 2 개 붙은 것은 파이썬에서 예약해 놓은 것, 프로그램을 작성시 이 이름을 사
용해서 새로운 함수나 변수명을 만들지 말 것

```
class 클래스명 :  
    def __init__(self) :  
        # 이 부분에 초기화할 코드 입력
```

```
class Car :  
    color = ""  
    speed = 0  
  
    def __init__(self) :  
        self.color = "빨강"  
        self.speed = 0
```

Section03 생성자

■ 기본 생성자

- 매개변수가 self 만 있는 생성자

Tip • Code12 - 03 . py 에서는 코드양 줄이려고 Code12 - 02 . py 와 달리 myCar1 과 myCar2 만 사용

Code12-03.py

```
1  ## 클래스 선언 부분 ##
2  class Car :
3      color = ""
4      speed = 0
5
6      def __init__(self) :
7          self.color = "빨강"
8          self.speed = 0
9
10     def upSpeed(self, value) :
11         self.speed += value
12
13     def downSpeed(self, value) :
14         self.speed -= value
15
16  ## 메인 코드 부분 ##
17  myCar1 = Car()
18  myCar2 = Car()
19
20  print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.color, myCar1.speed))
21  print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.color, myCar2.speed))
```

6 ~ 8행의 생성자가 17 ~ 18 행에서 자동으로 값 초기화

출력 결과

자동차1의 색상은 빨강이며, 현재 속도는 0km입니다.
자동차2의 색상은 빨강이며, 현재 속도는 0km입니다.

Section03 생성자

- 매개변수가 있는 생성자

Code12-04.py

```
1  ## 클래스 선언 부분 ##
2  class Car :
3      color = ""
4      speed = 0
5
6      def __init__(self, value1, value2) :
7          self.color = value1
8          self.speed = value2
9
10     # Code12-03.py의 upSpeed(), downSpeed() 함수와 동일
11
12  ## 메인 코드 부분 ##
13  myCar1 = Car("빨강", 30)
14  myCar2 = Car("파랑", 60)
15
16  # Code12-03.py의 20~21행과 동일
```

6행의 생성자에서 매개변수 2개를 받도록 설정

출력 결과

자동차1의 색상은 빨강이며, 현재 속도는 30km입니다.
자동차2의 색상은 파랑이며, 현재 속도는 60km입니다.

Section03 생성자

■ [프로그램 1]의 완성

Code12-05.py

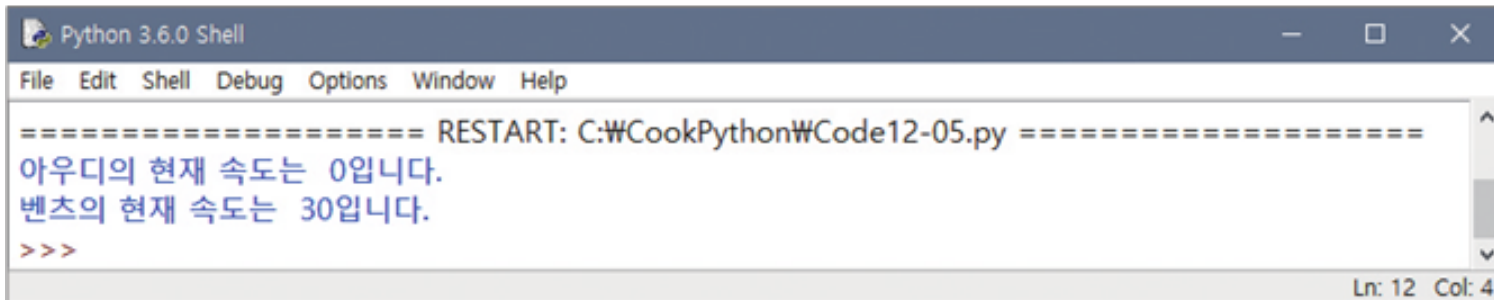
```
1  ## 클래스 선언 부분 ##
2  class Car :
3      name = ""
4      speed = 0
5
6      def __init__(self, name, speed) :
7          self.name = name
8          self.speed = speed
9
10     def getName(self) :
11         return self.name
12
13     def getSpeed(self) :
14         return self.speed
15
```

10행과 13행 : getName ()과 getSpeed () 메서드를
만들고 자동차의 이름과 현재 속도를 반환

Section03 생성자

```
16  ## 변수 선언 부분 ##
17  car1, car2 = None, None
18
19  ## 메인 코드 부분 ##
20  car1 = Car("아우디", 0)
21  car2 = Car("벤츠", 30)
22
23  print("%s의 현재 속도는 %d입니다." % (car1.getName(), car1.getSpeed()))
24  print("%s의 현재 속도는 %d입니다." % (car2.getName(), car2.getSpeed()))
```

23 ~ 24행 : name 이나 speed 필드를 사용하지 않고 getName (),
getSpeed () 메서드를 사용해서 값을 알아냄



```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:\CookPython\Code12-05.py =====
아우디의 현재 속도는 0입니다.
벤츠의 현재 속도는 30입니다.
>>>
Ln: 12 Col: 4
```

Section04 인스턴스 변수와 클래스 변수

■ 인스턴스 변수

- 예 : Car 클래스 2 개의 필드

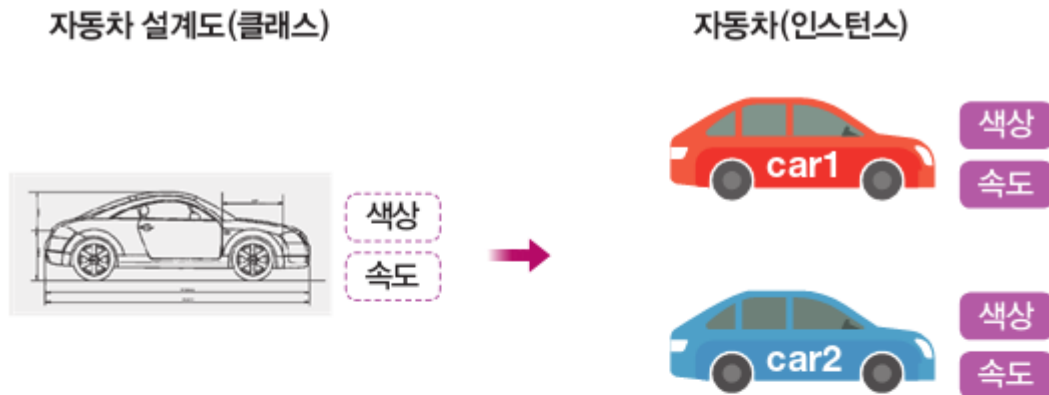
```
class Car :  
    color = ""    # 필드 : 인스턴스 변수  
    speed = 0     # 필드 : 인스턴스 변수
```

- 클래스를 이용해 메인 코드에서 인스턴스 만들기

```
myCar1 = Car()  
myCar2 = Car()
```

Section04 인스턴스 변수와 클래스 변수

■ 인스턴스 변수의 개념



```
class Car :  
    color = ""  
    speed = 0
```

```
myCar1 = Car()
```

```
myCar1.color
```

```
myCar1.speed
```

```
myCar2 = Car()
```

```
myCar2.color
```

```
myCar2.speed
```

그림 12-5 인스턴스 변수의 개념

Section04 인스턴스 변수와 클래스 변수

■ 클래스 변수

- 클래스 안에 공간이 할당된 변수, 여러 인스턴스가 클래스 변수 공간 함께 사용

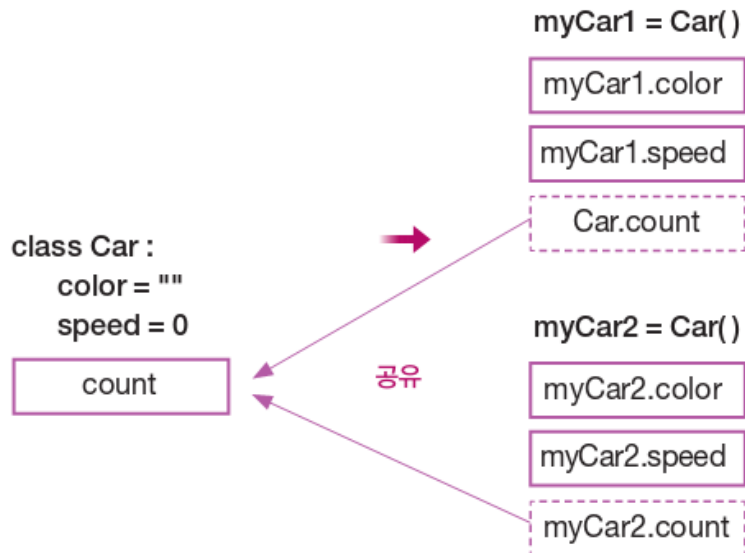
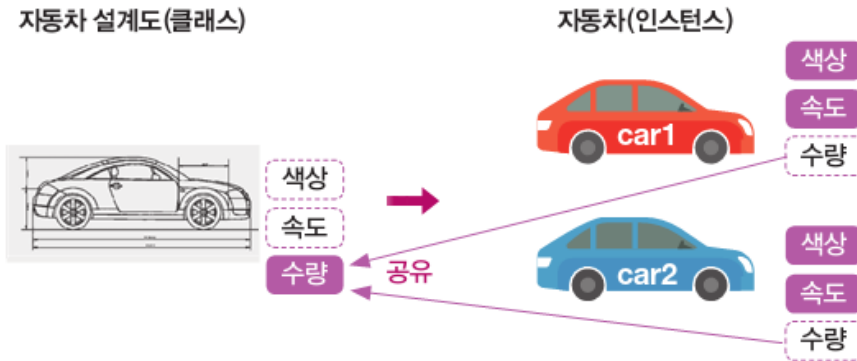


그림 12-6 클래스 변수의 개념

Section04 인스턴스 변수와 클래스 변수

- 자동차 생산 대수 확인 코드

Code12-06.py

```
1  ## 클래스 선언 부분 ##
2  class Car :
3      color = ""    # 인스턴스 변수
4      speed = 0     # 인스턴스 변수
5      count = 0     # 클래스 변수
6
7      def __init__(self) :
8          self.speed = 0
9          Car.count += 1
10
```

5행 : 클래스 변수 count 를 선언하고 0으로 초기화
7 ~ 9행 : 생성자 안에서 클래스 변수 에 접근하려고
클래스명. count 를 1 증가

Section04 인스턴스 변수와 클래스 변수

```
11  ## 변수 선언 부분 ##
12  myCar1, myCar2 = None, None
13
14  ## 메인 코드 부분 ##
15  myCar1 = Car()
16  myCar1.speed = 30
17  print("자동차1의 현재 속도는 %dkm, 생산된 자동차는 총 %d대입니다." % (myCar1.speed,
    Car.count))
18
19  myCar2 = Car()
20  myCar2.speed = 60
21  print("자동차2의 현재 속도는 %dkm, 생산된 자동차는 총 %d대입니다." % (myCar2.speed,
    myCar2.count))
```

출력 결과

자동차1의 현재 속도는 30km, 생산된 자동차는 총 1대입니다.

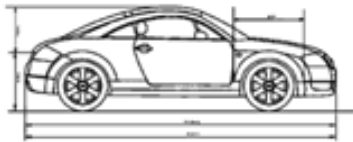
자동차2의 현재 속도는 60km, 생산된 자동차는 총 2대입니다.

Section05 클래스의 상속

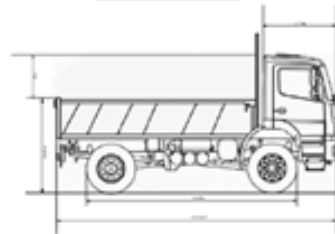
■ 상속의 개념

- 클래스의 상속(Inheritance) : 기존 클래스에 있는 필드와 메서드를 그대로 물려받는 새로운 클래스를 만드는 것

승용차 클래스



트럭 클래스



class 승용차 :

필드 - 색상, 속도, **좌석 수**

메서드 - 속도 올리기()

속도 내리기()

좌석 수 알아보기()

class 트럭 :

필드 - 색상, 속도, **적재량**

메서드 - 속도 올리기()

속도 내리기()

적재량 알아보기()

그림 12-7 승용차와 트럭 클래스의 개념

Section05 클래스의 상속

■ 상속의 개념

- 공통된 내용을 자동차 클래스에 두고 상속을 받음으로써 일관되고 효율적인 프로그래밍 가능
- 상위 클래스인 자동차 클래스를 슈퍼 클래스 또는 부모 클래스, 하위의 승용차와 트럭 클래스는 서브 클래스 또는 자식 클래스

class 자동차 :

필드 - 색상, 속도

메서드 - 속도 올리기()

속도 내리기()

자동차 클래스(공통 내용)



상속



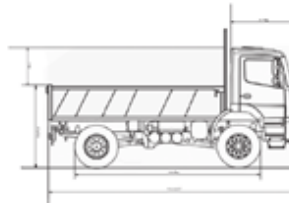
상속



승용차 클래스



트럭 클래스



class 승용차 : 자동차 상속

필드 - 자동차 필드, 좌석 수

메서드 - 자동차 메서드

좌석 수 알아보기()

class 트럭 : 자동차 상속

필드 - 자동차 필드, 적재량

메서드 - 자동차 메서드

적재량 알아보기()

그림 12-8 상속의 개념

Section05 클래스의 상속

- 상속을 구현하는 문법

```
class 서브_클래스(슈퍼_클래스) :  
    # 이 부분에 서브 클래스의 내용 코딩
```

Section05 클래스의 상속

■ 메서드 오버라이딩

- 상위 클래스의 메서드를 서브 클래스에서 재정의

class 자동차 :
 메서드 - 속도 올리기()

자동차 클래스(공통 내용)

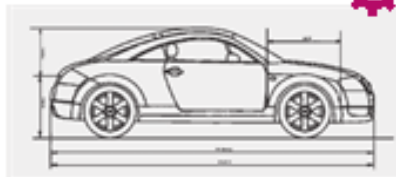


⚙️ 속도 올리기()

상속

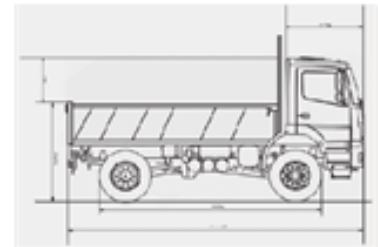
상속

승용차 클래스



⚙️ 속도 올리기()

트럭 클래스



class 승용차(자동차) :
 메서드 - 속도 올리기() 재정의

class 트럭(자동차) :
 메서드 -

그림 12-9 메서드 오버라이딩의 개념(다른 필드나 메서드는 그림에서 생략)

Section05 클래스의 상속

- 메서드 오버라이딩 구현 코드

Code12-07.py

```
1  ## 클래스 선언 부분 ##
2  class Car :
3      speed = 0
4      def upSpeed(self, value) :
5          self.speed += value
6
7          print("현재 속도(슈퍼 클래스) : %d" % self.speed)
8
9  class Sedan( Car ) :
10     def upSpeed(self, value) :
11         self.speed += value
12
13         if self.speed > 150 :
14             self.speed = 150
15
```

10 ~ 16행 : 서브 클래스(Sedan)의 upSpeed () 메서드 다시 만듦

Section05 클래스의 상속

```
16         print("현재 속도(서브 클래스) : %d" % self.speed)
17
18 class Truck(Car) :
19     pass
20
21 ## 변수 선언 부분 ##
22 sedan1, truck1 = None, None
23
24 ## 메인 코드 부분 ##
25 truck1 = Truck()
26 sedan1 = Sedan()
27
28 print("트럭 --> ", end = "")
29 truck1.upSpeed(200)
30
31 print("승용차 --> ", end = "")
32 sedan1.upSpeed(200)
```

32행 : Sedan 인스턴스의 upSpeed () 메서드 호출하면
10행에서 재정의된 upSpeed () 메서드를 호출
18행 : 서브 클래스(Truck)에는 아무런 내용 없어
슈퍼 클래스(Car)의 메서드를 그대로 상속
29행 : Truck 인스턴스의 upSpeed () 호출하면 4 ~ 7행
슈퍼 클래스(Car)의 upSpeed () 메서드 호출

출력 결과

트럭 --> 현재 속도(슈퍼 클래스) : 200
승용차 --> 현재 속도(서브 클래스) : 150

Section05 클래스의 상속

SELF STUDY 12-2

Code12-07.py에 Sonata 클래스를 추가해 보자. 단 Sonata 클래스는 Car → Sedan → Sonata 순서로 상속을 받도록 하자. Sonata 클래스에는 특별히 추가하는 필드나 메서드가 없다.

출력 결과

트럭 --> 현재 속도(슈퍼 클래스) : 200

승용차 --> 현재 속도(서브 클래스) : 150

소나타 --> 현재 속도(서브 클래스) : 150

Section05 클래스의 상속

■ [프로그램 2]의 완성

Code12-08.py

```
1  import turtle
2  import random
3
4  ## 클래스 선언 부분 ##
5  class Shape :          # 슈퍼 클래스
6      myTurtle = None
7      cx, cy = 0, 0      # 도형의 중심점
8
9      def __init__(self) :
10         self.myTurtle = turtle.Turtle('turtle')    # 거북이 생성
11
12         def setPen(self) :                          # 펜 색상과 두께 무작위로 뽑기
13             r = random.random()
14             g = random.random()
15             b = random.random()
16             self.myTurtle.pencolor((r, g, b))
17             pSize = random.randrange(1, 10)
18             self.myTurtle.pensize(pSize)
```

5 ~ 21행 : 슈퍼 클래스인 도형(Shape) 클래스 선언
6 ~ 7행 : 도형에 공통으로 사용할 필드 준비
9 ~ 10행 : Shape 생성자에서 거북이 생성
12 ~ 18행 : 색상, 선 두께를 무작위 추출하는 메서드 선언

Section05 클래스의 상속

```
19
20     def drawShape(self) :                # 서브 클래스에서 상속받아 오버라이딩
21         pass                             20 행 : drawShape ( ) 메서드는 서브 클래스에서 오버라이드
22
23 class Rectangle(Shape) :                # 서브 클래스
24     width, height = [0] * 2              23 ~ 48행 : 서브 클래스인 사각형( Rectangle )을 정의
25     def __init__(self, x, y) :           24행 : Rectangle 에서 필요한 속성인 폭과 넓이 준비
26         Shape.__init__(self)             25 ~ 30행 : 생성자로 26 행에서 슈퍼 클래스의 생성자 호출
27         self.cx = x                     27 ~ 28행 : 사각형의 중심을 x , y 로 받아 슈퍼 클래스 에서
28         self.cy = y                     상속받은 속성인 cx , cy 에 대입
29         self.width = random.randrange(20, 100)
30         self.height = random.randrange(20, 100)
31
32     def drawShape(self) :
33         # 네모 그리기
34         sx1, sy1, sx2, sy2 = [0] * 4     # 왼쪽 위 X, Y와 오른쪽 아래 X, Y
```

Section05 클래스의 상속

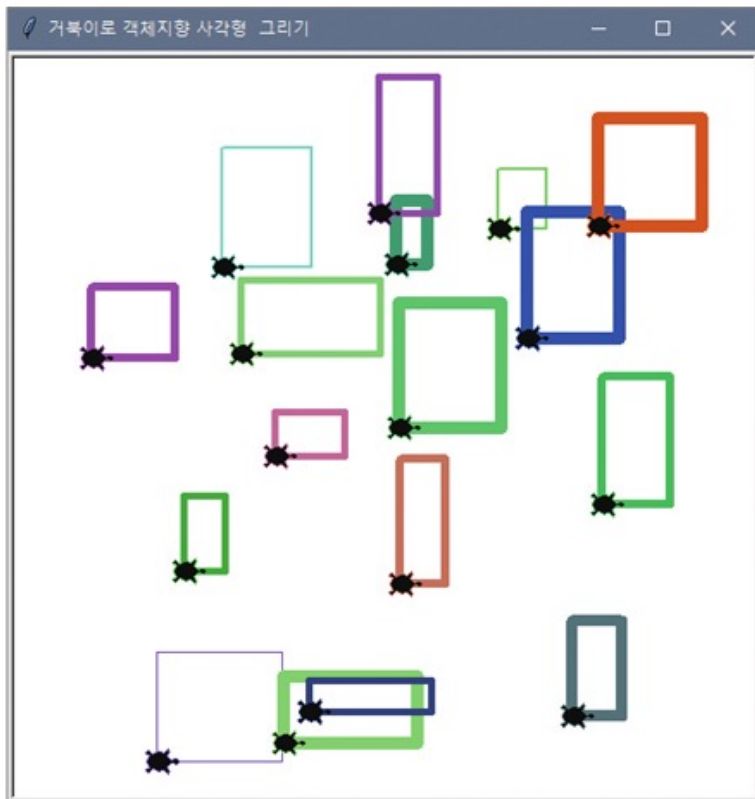
```
35
36     sx1 = self.cx - self.width / 2
37     sy1 = self.cy - self.height / 2
38     sx2 = self.cx + self.width / 2
39     sy2 = self.cy + self.height / 2
40
41     self.setPen()          # 부모 클래스 메서드
42     self.myTurtle.penup()
43     self.myTurtle.goto(sx1, sy1)
44     self.myTurtle.pendown()
45     self.myTurtle.goto(sx1, sy2)
46     self.myTurtle.goto(sx2, sy2)
47     self.myTurtle.goto(sx2, sy1)
48     self.myTurtle.goto(sx1, sy1)
49
```

32 ~ 48행 : 슈퍼 클래스의 drawShape () 메서드
오버라이드
36 ~ 39행 : 클릭한 점 중심으로 사각형의 왼쪽 위,
오른쪽 아래의 위치 계산
41 ~ 48행 : 사각형을 그림

Section05 클래스의 상속

```
50 ## 함수 선언 부분 ##
51 def screenLeftClick(x, y) :
52     rect = Rectangle(x, y)
53     rect.drawShape()
54
55 ## 메인 코드 부분 ##
56 turtle.title('거북이로 객체지향 사각형 그리기')
57 turtle.onscreenclick(screenLeftClick, 1)
58 turtle.done()
```

51 ~ 53행 : 마우스 클릭 때마다 클릭 위치 이용 사각형(Rectangle)
인스턴스 생성, drawShape () 메서드 실행해서 사각형 그림
57행 : 마우스 클릭하면 screenLeftClick () 메서드 실행하도록 설정



Section06 객체지향 프로그래밍의 심화 내용

■ 클래스의 특별한 메서드

- `__del__()` 메서드
 - 소멸자(Destructor), 생성자와 반대로 인스턴스 삭제할 때 자동 호출.
- `__repr__()` 메서드
 - 인스턴스를 `print()` 문으로 출력할 때 실행
- `__add__()` 메서드
 - 인스턴스 사이에 덧셈 작업이 일어날 때 실행되는 메서드, 인스턴스 사이의 덧셈 작업 가능
- 비교 메서드 : `__lt__()`, `__le__()`, `__gt__()`, `__ge__()`, `__eq__()`, `__ne__()`
 - 인스턴스 사이의 비교 연산자(<, <=, >, >=, ==, != 등) 사용할 때 호출

Code12-09.py

```
1  ## 클래스 선언 부분 ##
2  class Line :
3      length = 0
4      def __init__(self, length) :
5          self.length = length
```

Section06 객체지향 프로그래밍의 심화 내용

```
6         print(self.length, '길이의 선이 생성되었습니다.')
7
8     def __del__(self):
9         print(self.length, '길이의 선이 삭제되었습니다.')
10
11    def __repr__(self):
12        return '선의 길이 : ' + str(self.length)
13
14    def __add__(self, other):
15        return self.length + other.length
16
17    def __lt__(self, other):
18        return self.length < other.length
19
20    def __eq__(self, other):
21        return self.length == other.length
22
23    ## 메인 코드 부분 ##
24    myLine1 = Line(100)
25    myLine2 = Line(200)
```

Section06 객체지향 프로그래밍의 심화 내용

```
26
27 print(myLine1)
28
29 print('두 선의 길이 합 : ', myLine1 + myLine2)
30
31 if myLine1 < myLine2 :
32     print('선분 2가 더 기네요.')
33 elif myLine1 == myLine2 :
34     print('두 선분이 같네요.')
```

```
35 else :
36     print('모르겠네요.')
```

```
37
```

```
38 del(myLine1)
```

출력 결과

100 길이의 선이 생성되었습니다.

200 길이의 선이 생성되었습니다.

선의 길이 : 100

두 선의 길이 합 : 300

선분 2가 더 기네요.

100 길이의 선이 삭제되었습니다.

Section06 객체지향 프로그래밍의 심화 내용

■ 추상 메서드

- 서브 클래스에서 메서드를 오버라이딩 : 슈퍼 클래스에서는 빈 껍질의 메서드만 만들어 놓고 내용은 pass 로 채움

Code12-10.py

```
1  ## 클래스 선언 부분 ##
2  class SuperClass :
3      def method(self) :
4          pass
5
6  class SubClass1 (SuperClass) :
7      def method(self) :          # 메서드 오버라이딩
8          print('SubClass1에서 method()를 오버라이딩함')
9
10 class SubClass2 (SuperClass) :|
11     pass
12
13 ## 메인 코드 부분 ##
14 sub1 = SubClass1()
15 sub2 = SubClass2()
16
17 sub1.method()
18 sub2.method()
```

2 ~ 11행 : SuperClass 상속받은 SubClass1 과 SubClass2 만듦
14 ~ 15행 : 각 인스턴스 sub1 과 sub2 생성
17 ~ 18행 : 오버라이딩한 method () 호출

출력 결과

SubClass1에서 method()를 오버라이딩함

Section06 객체지향 프로그래밍의 심화 내용

- 3 ~ 4행 method () 수정
 - 오버라이딩하지 않았다는 Not Implement Error 발생

```
def method(self) :  
    raise NotImplementedError()
```

출력 결과

NotImplementedError

■ 멀티 스레드

- 프로그램 하나에서 여러 개를 동시에 처리할 수 있도록 제공하는 기능

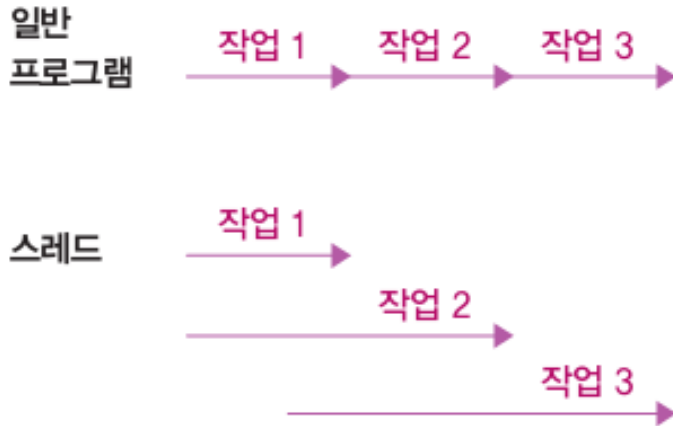


그림 12-10 일반 프로그램과 스레드의 차이

Section06 객체지향 프로그래밍의 심화 내용

- 자동차 세 대가 경주하는 코드

Code12-11.py

```
1 import time
2
3 ## 클래스 선언 부분 ##
4 class RacingCar :
5     carName = ''
6     def __init__(self, name) :
7         self.carName = name
8
9     def runCar(self) :
10         for _ in range(0, 3) :
11             carStr = self.carName + '~ 달립니다.\n'
12             print(carStr, end = '')
13             time.sleep(0.1)      # 0.1초 멈춤
14
```

4 ~ 13행 : RacingCar 클래스 정의
9 ~ 13행 : 자동차가 달린다는 것을 3번 출력하는
runCar () 메서드 만듦
13행 : 너무 빠른 출력 방지하려고 0.1 초 동안 멈춤

Section06 객체지향 프로그래밍의 심화 내용

```
15  ## 메인 코드 부분 ##
16  car1 = RacingCar('@자동차1')
17  car2 = RacingCar('#자동차2')
18  car3 = RacingCar('$자동차3')
19
20  car1.runCar()
21  car2.runCar()
22  car3.runCar()
```

20 ~ 22행 : 세 대가 차례로 출발

출력 결과

```
@자동차1~~ 달립니다.
@자동차1~~ 달립니다.
@자동차1~~ 달립니다.
#자동차2~~ 달립니다.
#자동차2~~ 달립니다.
#자동차2~~ 달립니다.
$자동차3~~ 달립니다.
$자동차3~~ 달립니다.
$자동차3~~ 달립니다.
```

Section06 객체지향 프로그래밍의 심화 내용

■ 자동차 세 대를 동시에 출발

Code12-12.py

```
1 import threading
2 import time
3
4 ## 클래스 선언 부분 ##
5 class RacingCar :
6     # Code12-11.py와 동일
7
8 ## 메인 코드 부분 ##
9 car1 = RacingCar('@자동차1')
10 car2 = RacingCar('#자동차2')
11 car3 = RacingCar('$자동차3')
12
13 th1 = threading.Thread(target = car1.runCar)
14 th2 = threading.Thread(target = car2.runCar)
15 th3 = threading.Thread(target = car3.runCar)
16
17 th1.start()
18 th2.start()
19 th3.start()
```

1행 : threading 모듈 임포트

13행 : threading . Thread (target =메서드 또는 함수,
args =(매개변수)) 형식 사용 스레드 생성

13행 : car1 . runCar () 메서드명을 스레드로 생성

13 ~ 15행 : 스레드 3개를 생성

17 ~ 19행 : 스레드 start ()

Section06 객체지향 프로그래밍의 심화 내용

출력 결과

```
@자동차1~~ 달립니다.  
#자동차2~~ 달립니다.  
$자동차3~~ 달립니다.  
@자동차1~~ 달립니다.  
#자동차2~~ 달립니다.  
$자동차3~~ 달립니다.  
@자동차1~~ 달립니다.  
#자동차2~~ 달립니다.  
$자동차3~~ 달립니다.
```

Section06 객체지향 프로그래밍의 심화 내용

■ 멀티 프로세싱

- 동시에 CPU 를 여러 개 사용

Code12-13.py

```
1 import multiprocessing
2 import time
3
4 ## 클래스 선언 부분 ##
5 class RacingCar :
6     # Code12-11.py와 동일
7
8 ## 메인 코드 부분 ##
9 if __name__ == "__main__" :
10     car1 = RacingCar('@자동차1')
11     car2 = RacingCar('#자동차2')
12     car3 = RacingCar('$자동차3')
13
14     mp1 = multiprocessing.Process(target = car1.runCar)
15     mp2 = multiprocessing.Process(target = car2.runCar)
16     mp3 = multiprocessing.Process(target = car3.runCar)
17
```

14행 : multiprocessing . Process (target =메서드 또는 함수, args =(매개변수)) 형식 사용 스레드 생성

Section06 객체지향 프로그래밍의 심화 내용

```
18 mp1.start() 18행 : 프로세스 시작
```

```
19 mp2.start()
```

```
20 mp3.start()
```

21

```
22 mp1.join()
```

```
23 mp2.join()
```

24 mp3.join()

→ 자식 스트레스가 종료될때 까지 기다림 → 부모 스트레스 종료

```
c:\#CookPython>python Code12-13.py
#자동차2~~ 두대입니다.
$자동차3~~ 세대입니다.
@자동차1~~ 한대입니다.
#자동차2~~ 두대입니다.
@자동차1~~ 한대입니다.
$자동차3~~ 세대입니다.
#자동차2~~ 두대입니다.
@자동차1~~ 한대입니다.
$자동차3~~ 세대입니다.

c:\#CookPython>
```



Thank You
