



컴퓨팅 사고력을 키우는 SW 교육

파이썬

Chapter 08

살짝 맛보는 객체지향

- 01 클래스에 대해 알아보시다
- 02 생성자에 대해 알아보시다
- 03 인스턴스 변수와 클래스 변수의 차이는?
- 04 클래스의 상속에 대해 알아보시다

- 객체지향의 개념을 익힙니다.
- 클래스, 인스턴스를 구분하고 이해합니다.
- 객체지향 프로그래밍의 활용법을 배웁니다.



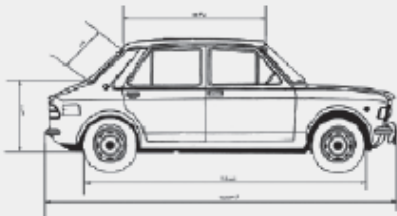
Section 01 클래스에 대해 알아봅시다

클래스에 대해 알아봅시다(1)

■ 클래스 개념

- 파이썬은 객체지향 개념을 적용할 수 있는 프로그래밍 언어
- 클래스의 모양과 생성

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



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

그림 11-3
자동차를 클래스로 구현

클래스에 대해 알아봅시다(2)

- 자동차의 속성은 필드(Field)라 함. 자동차의 기능은 함수(Function)형태로 구현. 클래스 안에서 구현된 함수는 메소드(Method)라 부름.

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

클래스에 대해 알아봅시다(3)

소스코드 11-1

(파일명 : 11-01.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
```

출력 결과

아무것도 나오지 않음

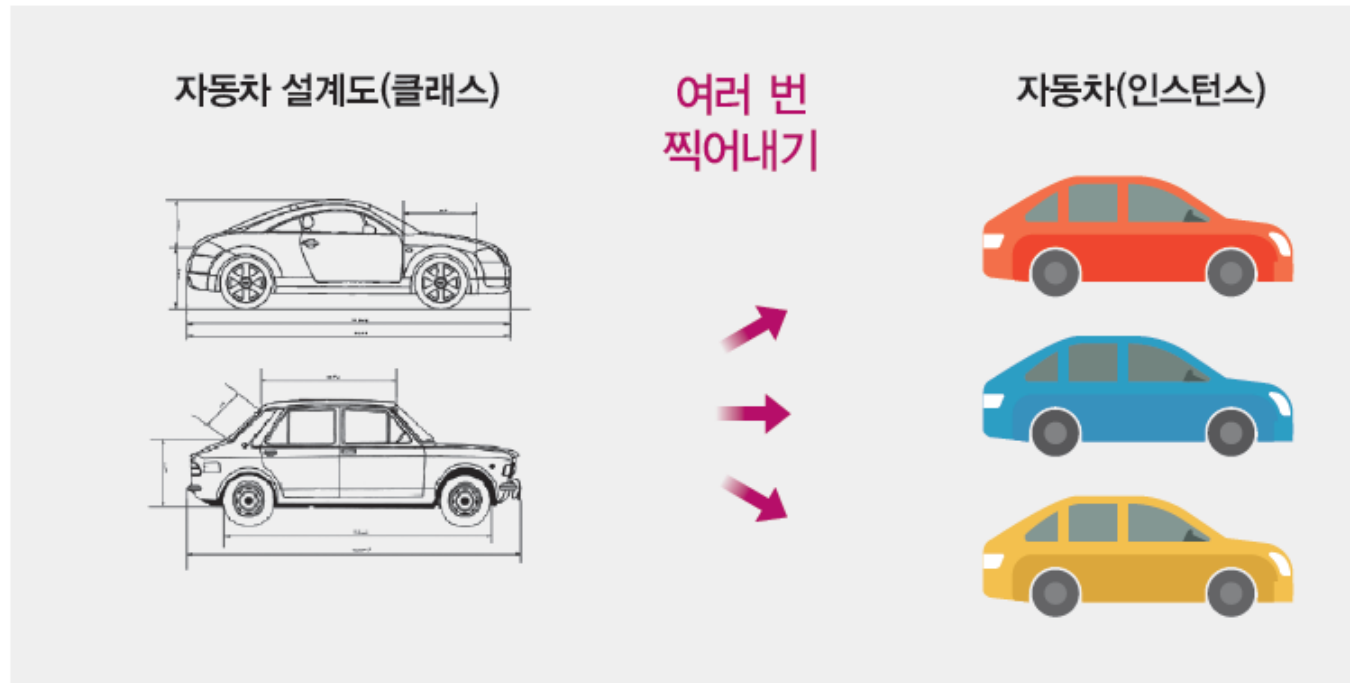
- 필드의 이름과 메소드의 매개변수를 영문 변수 명으로 변경, 메소드는 자동차의 속도(speed)를 변경시키는 내용으로 소스를 완성함
- self는 클래스 자신을 가리킴, 6행의 self.speed는 3행의 speed를 의미함. 즉 자신의 클래스에 있는 speed 변수라고 해석함

클래스에 대해 알아봅시다(4)

- 인스턴스 생성하기

그림 11-4

클래스와 인스턴스 개념



클래스에 대해 알아봅시다(5)


그림 11-5
클래스와 인스턴스 코드 구성

자동차 설계도(클래스)

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

자동차(인스턴스)

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



- 세 대의 자동차 인스턴스 생성을 실제 코드로 구현

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

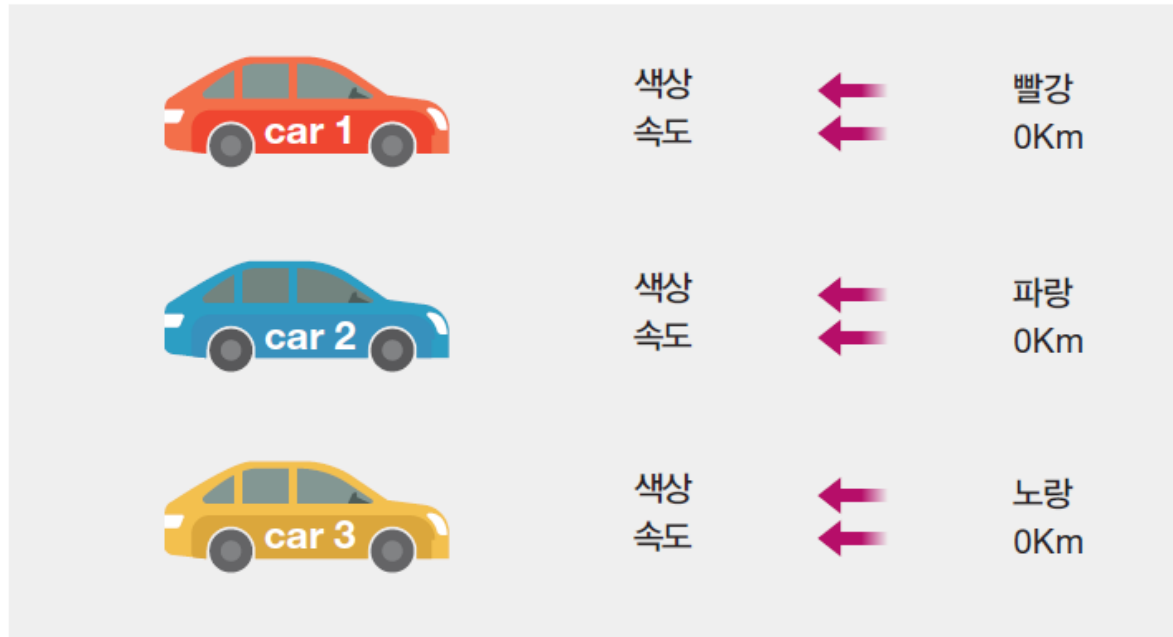
- 3개 인스턴스는 각각 자동차의 색상(color), 속도(speed) 필드를 가짐

클래스에 대해 알아봅시다(6)

■ 필드에 값 대입하기

- 각 인스턴스에는 별도의 필드가 존재, 각각에 별도의 값 대입 가능함

그림 11-6
인스턴스 필드에
값 대입하기



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

클래스에 대해 알아봅시다(7)

■ 메소드 호출하기

- Car 클래스에서 메소드는 upSpeed()와 downSpeed() 2개임

```
myCar1.upSpeed(30)
myCar2.upSpeed(60)
```

■ 클래스의 완전한 작동 코딩

소스코드 11-2

(파일명 : 11-02.py)

```
1  # 클래스 정의 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
```

클래스에 대해 알아봅시다(8)

```
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
21 myCar3=Car()
22 myCar3.color="노랑"
23 myCar3.speed=0
24
25 myCar1.upSpeed(30)
26 print("자동차1의 색상은 %s이며, 현재속도는 %d km 입니다." % (myCar1.
    color, myCar1.speed))
27
28 myCar2.upSpeed(60)
29 print("자동차2의 색상은 %s이며, 현재속도는 %d km 입니다." % (myCar2.
    color, myCar2.speed))
30
```

클래스에 대해 알아봅시다(9)

```
31 myCar3.upSpeed(0)
32 print("자동차3의 색상은 %s이며, 현재속도는 %d km 입니다." % (myCar3.
    color, myCar3.speed))
```

출력 결과

자동차1의 색상은 빨강이며, 현재속도는 30 km 입니다.
자동차2의 색상은 파랑이며, 현재속도는 60 km 입니다.
자동차3의 색상은 노랑이며, 현재속도는 0 km 입니다.

단계	작업	형식	예
1단계	클래스 생성	class 클래스이름 : // 필드 선언 // 메소드 선언	class Car : color="" def upSpeed (self, value) : ~~~
2단계	인스턴스 생성	인스턴스 = 클래스이름()	myCar1 = Car()
3단계	필드나 메소드 사용	인스턴스.필드이름 = 값 인스턴스.메소드()	myCar1.color = "빨강" myCar1.upSpeed(30)



Section 02 생성자에 대해 알아봅시다

생성자에 대해 알아봅시다(1)

■ 생성자의 의미

- 생성자는 인스턴스를 생성하면 무조건 호출되는 메소드

```
13행: myCar1=Car()  
14행: myCar1.color="빨강"  
15행: myCar1.speed=0
```

- 13행에서 인스턴스 생성 후 14,15행에서 따로 초기화를 하였는데, 13행 자체에서 인스턴스 생성과 동시에 필드값을 초기화 할 수 있음. 이 함수를 생성자라 함

■ 생성자의 기본

- 생성자는 `__init__()`라는 이름을 가짐

```
class 클래스이름 :  
    def __init__(self) :  
        // 이 부분에 초기화할 코드를 입력
```

생성자에 대해 알아봅시다(2)

- Car 클래스의 생성자

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

- 이제는 [소스코드 11-2]의 13행~15행 중에서 13행만 있으면 됨. 즉 인스턴스를 생성하면 자동으로 생성자가 호출

```
myCar1=Car()
```

생성자에 대해 알아봅시다(3)

- 기본 생성자 - 매개변수가 self만 있는 생성자

소스코드 11-3

(파일명 : 11-03.py)

```
1  # 클래스 정의 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()
```


생성자에 대해 알아봅시다(4)

```
19
20 print("자동차1의 색상은 %s이며, 현재속도는 %d km 입니다." % (myCar1.
    color, myCar1.speed))
21
22 print("자동차2의 색상은 %s이며, 현재속도는 %d km 입니다." % (myCar2.
    color, myCar2.speed))
```

출력 결과

자동차1의 색상은 빨강이며, 현재속도는 0 km 입니다.
자동차2의 색상은 빨강이며, 현재속도는 0 km 입니다.

- 매개변수가 있는 생성자
 - [소스코드 11-3]을 수정해서 매개변수가 있는 생성자를 사용해봄.
 - 인스턴스를 만들 때 초기값을 매개변수로 넘기는 방법을 사용해봄

생성자에 대해 알아봅시다(5)

소스코드 11-4

(파일명 : 11-04.py)

```
1  # 클래스 정의 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     ## [소스코드 11-3]의 upSpeed(), downSpeed() 함수와 동일
11
12 # 메인 코드 부분
13 myCar1 = Car("빨강", 30)
14 myCar2 = Car("파랑", 60)
15
16 ## [소스코드 11-3]의 20~22행과 동일
```

출력 결과

자동차1의 색상은 빨강이며, 현재속도는 30 km 입니다.
자동차2의 색상은 파랑이며, 현재속도는 60 km 입니다.

생성자에 대해 알아봅시다(6)

■ 객체지향 기본 프로그램 완성

소스코드 11-5

(파일명 : 11-05.py)

```
1  # 클래스 선언
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
```

생성자에 대해 알아봅시다(7)

```
16 # 변수 선언
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() ))
```

- 10행, 13행 : getName()과 getSpeed() 메소드를 만들고 자동차의 이름과 현재 속도를 반환함
- 23행, 24행 : name이나 speed 필드를 사용하지 않고 getName(), getSpeed() 메소드를 사용해서 값을 알아냄



Section 03 인스턴스 변수와 클래스 변수의 차이는?

인스턴스 변수와 클래스 변수의 차이는?(1)

- 인스턴스 변수

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

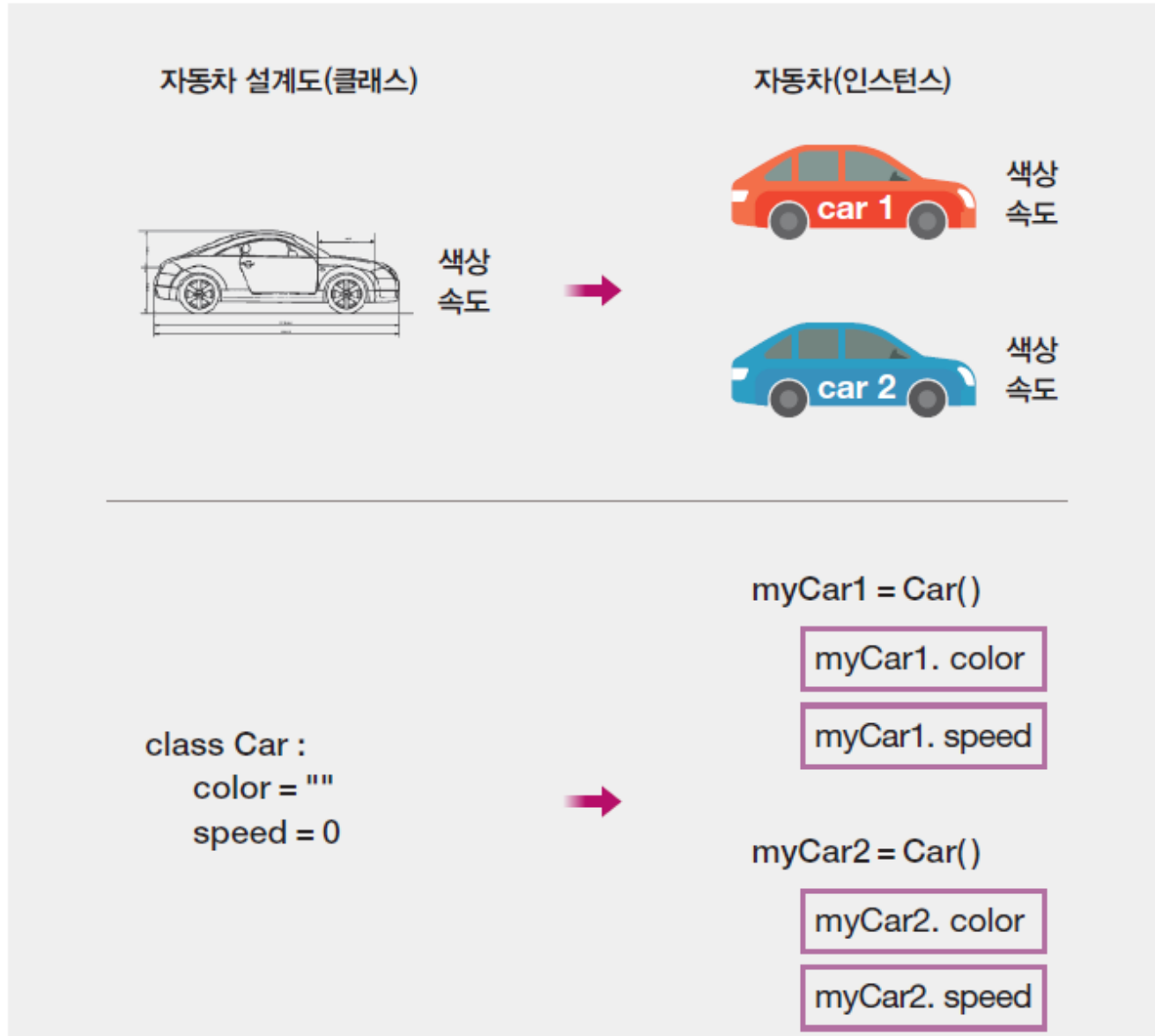
- 클래스를 이용하여 메인 코드 부분에서 인스턴스 만들기

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

인스턴스 변수와 클래스 변수의 차이는?(2)

그림 11-7

인스턴스 변수의 개념



인스턴스 변수와 클래스 변수의 차이는?(3)

■ 클래스 변수

- 클래스 안에
공간이 할당된
변수

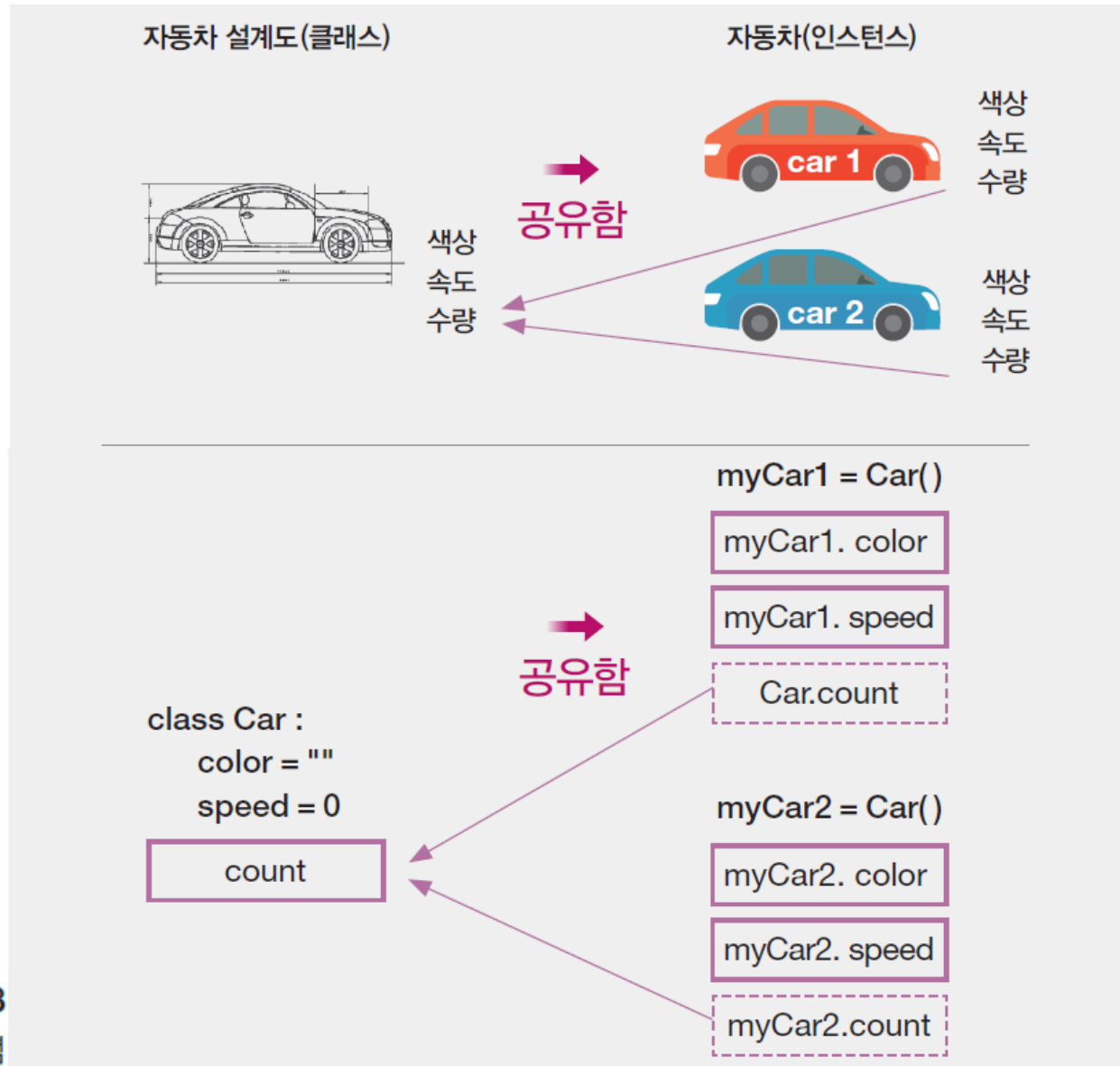


그림 11-8
클래스 변수의 개념

인스턴스 변수와 클래스 변수의 차이?(4)

- 클래스 변수 : class에서 선언된 변수로, 해당 클래스를 사용하는 모든 인스턴스에서 공유
으로 사용되는 변수
 - 클래스 변수에 접근시 '클래스이름.클래스변수명' 또는 '인스턴스.클래스변수명' 방식으로 접근가능.
- 인스턴스 변수 : __init__ 생성자안에 self로 선언된 변수로 인스턴스에서만 사용되는 변수

```
1 class Rectangle:
2     count = 0 # 클래스 변수
3
4     # 초기자(initializer)
5     def __init__(self, width, height):
6         # self.* : 인스턴스변수
7         self.width = width
8         self.height = height
9         Rectangle.count += 1
10
11     # 메서드
12     def calcArea(self):
13         area = self.width * self.height
14         return area
```

- 클래스 변수, 인스턴스 변수 사용법

```
1 # 인스턴스 생성
2 r = Rectangle(2, 3)
3
4 # 메서드 호출
5 area = r.calcArea()
6 print("area = ", area)
7
8 # 인스턴스 변수 액세스
9 r.width = 10
10 print("width = ", r.width)
11
12 # 클래스 변수 액세스
13 print(Rectangle.count)
14 print(r.count)
```

인스턴스 메소드, 정적 메소드, 클래스 메소드 차이는?

- 인스턴스 메소드 : 인스턴스 생성 후에 사용. 메소드 parameter에 self 포함됨
- 정적 메소드 : 인스턴스 변수 사용 불가. 메소드 이름 앞에 @staticmethod라는 decorator 선언해 주어야 함
- 클래스 메소드 : parameter에 cls 전달받고, 클래스 변수 사용 가능. 메소드 이름 앞에 @classmethod라는 decorator 선언

```
1  class Rectangle:
2      count = 0 # 클래스 변수
3
4      def __init__(self, width, height):
5          self.width = width
6          self.height = height
7          Rectangle.count += 1
8
9      # 인스턴스 메서드
10     def calcArea(self):
11         area = self.width * self.height
12         return area
13
14     # 정적 메서드
15     @staticmethod
16     def isSquare(rectWidth, rectHeight):
17         return rectWidth == rectHeight
18
19     # 클래스 메서드
20     @classmethod
21     def printCount(cls):
22         print(cls.count)
23
24
25     # 테스트
26     square = Rectangle.isSquare(5, 5)
27     print(square) # True
28
29     rect1 = Rectangle(5, 5)
30     rect2 = Rectangle(2, 5)
31     rect1.printCount() # 2
```



Section 04 클래스의 상속에 대해 알아봅시다

클래스의 상속에 대해 알아봅시다(1)

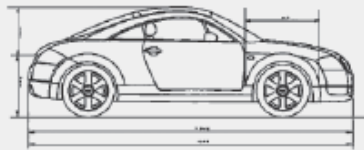
■ 상속의 개념

- 기존 클래스의 필드와 메소드를 그대로 물려받는 새로운 클래스를 만드는 것.

그림 11-9

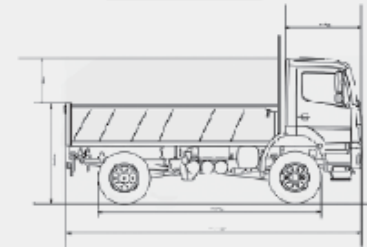
승용차와 트럭 클래스의 개념

승용차 클래스



class 승용차 :
필드 - 색상, 속도, 좌석수
메소드 - 속도 올리기()
 속도 내리기()
 좌석수 알아보기()

트럭 클래스

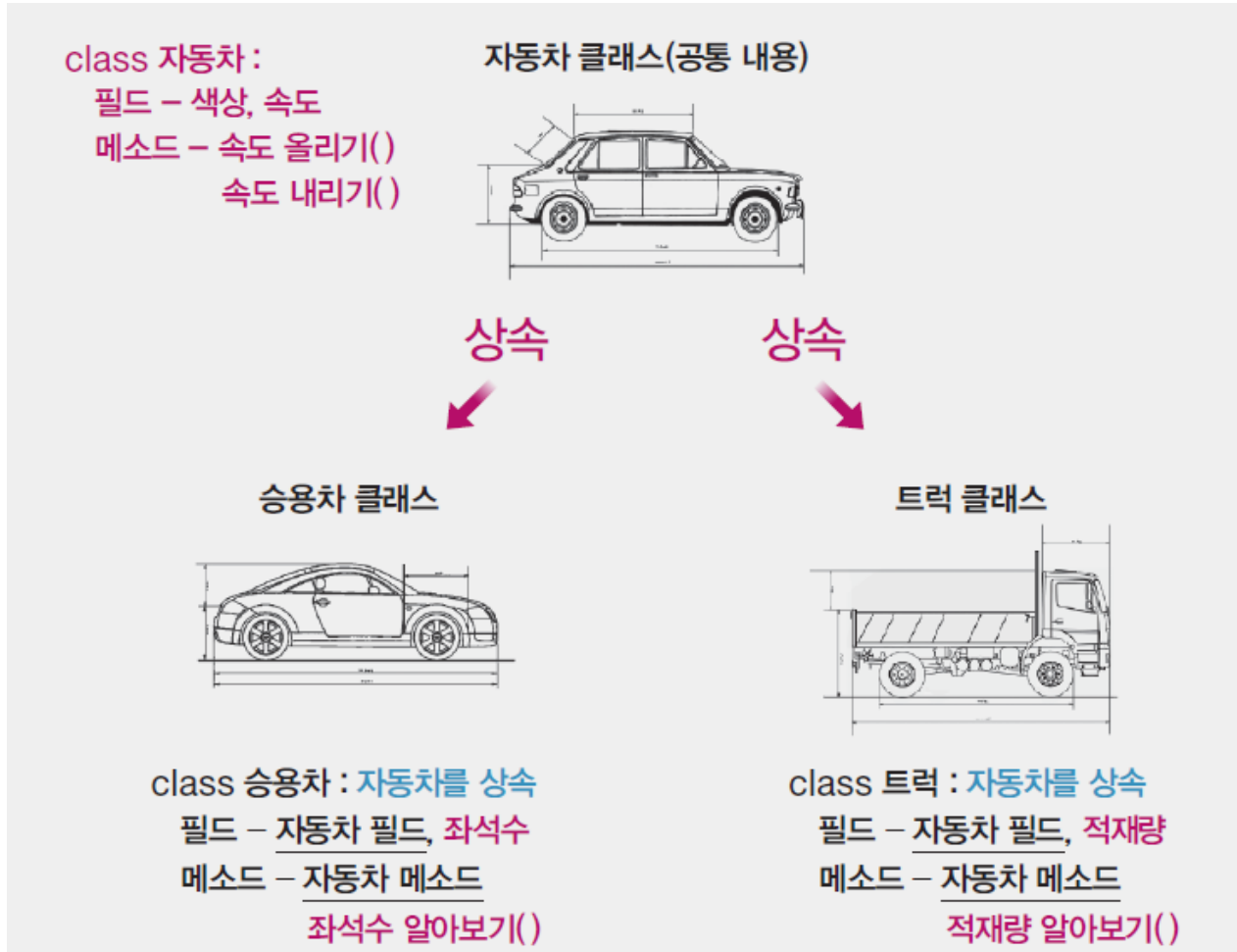


class 트럭 :
필드 - 색상, 속도, 적재량
메소드 - 속도 올리기()
 속도 내리기()
 적재량 알아보기()

클래스의 상속에 대해 알아봅시다(2)

- 기존의 두 클래스가 공통되는 것이 많음. 즉 공통된 특징을 '자동차' 라는 클래스로 만들고 승용차와 트럭은 클래스의 특징을 물려받아 각각에 필요한 필드와 메소드만 추가하면 효율적일 것임.

그림 11-10
상속의 개념



클래스의 상속에 대해 알아보시다(3)

- 상위 클래스인 자동차 클래스를 '슈퍼 클래스' 또는 '부모 클래스'라 하며, 하위 클래스인 승용차와 트럭 클래스는 '서브 클래스' 또는 '자식 클래스'라 함

```
class 서브 클래스(슈퍼 클래스) :  
    // 이곳에 서브 클래스의 내용을 코딩
```

클래스의 상속에 대해 알아봅시다(4)

■ 객체지향 활용 프로그램 완성

- 자동차 클래스를 만든 후, 승용차 클래스와 트럭 클래스가 자동차 클래스의 상속을 받음

소스코드 11-7

(파일명 : 11-07.py)

```
1  # 클래스 선언
2  class Car :
3      speed = 0
4
5      def upSpeed(self, value) :
6          self.speed = self.speed + value
7
8      def downSpeed(self, value) :
9          self.speed = self.speed - value
10
11 class Sedan(Car) :
12     seatNum = 0
13
14     def getSeatNum(self) :
15         return self.seatNum
16
```

클래스의 상속에 대해 알아보시다(5)

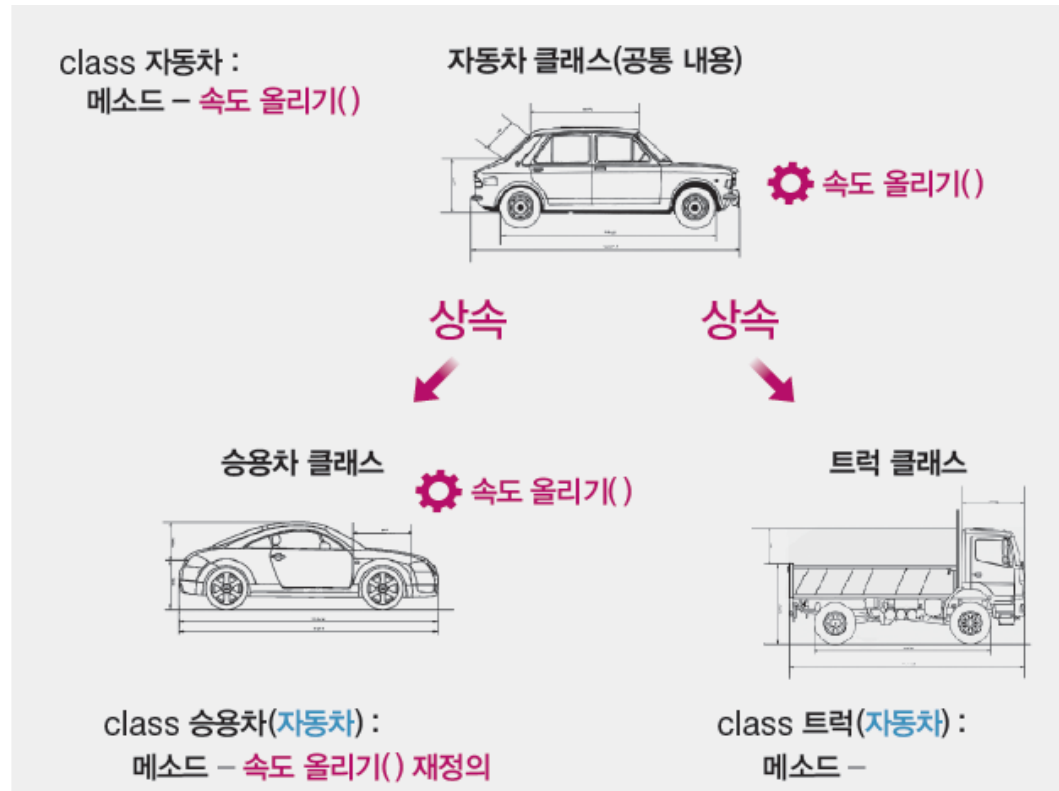
```
17 class Truck(Car) :
18     capacity=0
19
20     def getCapacity(self) :
21         return self.capacity
22
23 # 변수 선언
24 sedan1, truck1=None, None
25
26 # 메인 코드 부분
27 sedan1=Sedan()
28 truck1=Truck()
29
30 sedan1.upSpeed(100)
31 truck1.upSpeed(80)
32
33 sedan1.seatNum=5
34 truck1.capacity=50
35
36 print("승용차의 속도는 %d km, 좌석수는 %d개입니다." % (sedan1.speed,
    sedan1.getSeatNum() ))
37 print("트럭의 속도는 %d km, 총중량은 %d톤입니다." % (truck1.speed,
    truck1.getCapacity() ))
```


클래스의 상속에 대해 알아봅시다(6)

■ 메소드 오버라이딩(Overriding)

- 상위 클래스의 메소드를 하위 클래스에서 재정의하는 것. 다음의 그림에서 트럭은 속도에 제한이 없지만, 승용차는 안전상 속도가 최대 150km로 제한되어야 한다고 가정함.
- 슈퍼 클래스(자동차)를 상속받은 서브 클래스(승용차, 트럭)는 속도 올리기() 메소드를 상속받았지만, 승용차의 경우 속도의 제한이 필요해서 자동차의 속도 올리기()와 내용이 달라야 하므로 승용차 클래스에서 속도 올리기()를 다시 만들어서 사용함.

그림 11-11
메소드 오버라이딩의 개념
(다른 필드나 메소드는 생략함)



클래스의 상속에 대해 알아봅시다(7)

소스코드 11-8

(파일명 : 11-08.py)

```
1  ## 클래스 선언
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
16         print("현재 속도(서브 클래스) : %d" % self.speed)
17
18 class Truck(Car) :
19     pass
20
```

클래스의 상속에 대해 알아보시다(8)

```
21 # 변수 선언
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)
```

출력 결과

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



Thank You
