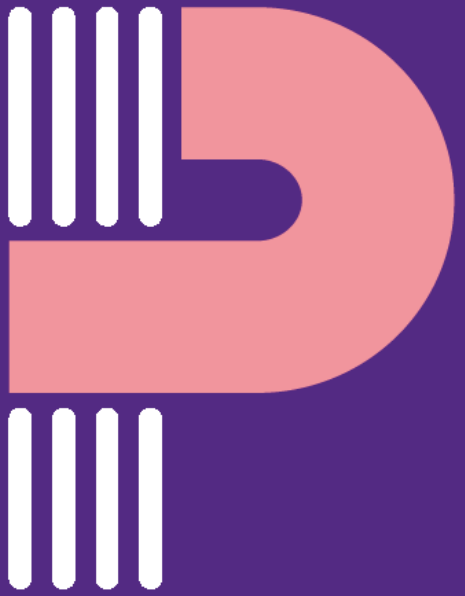
IT@COOKBOOK

한빛아카데미 Hanbit Academy, Inc.

실생활 융합 예제로 배우는

파이썬 마스터



PART 02. 파이썬 심화

10. 객체

목차

학습목표

- 클래스와 객체의 개념을 설명할 수 있습니다.
- 상속의 개념을 설명할 수 있습니다.
- 클래스와 객체를 이용하여 프로그램을 작성할 수 있습니다.
- 상속을 이용하여 프로그램을 작성할 수 있습니다.

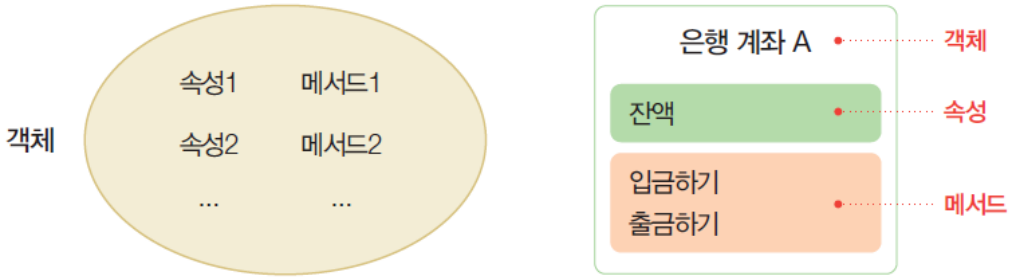
Section 10.1

객체와 클래스

1. 객체와 클래스

■ 객체의 개념

- 프로그래밍 기법 중 하나인 객체 지향 프로그래밍(object oriented programming)은 현실 세계에 존재하는 모든 것을 객체로 보고 그런 객체를 프로그램으로 작성하는 개념을 말함
- 현실 세계에 존재하는 모든 것이 객체(object) 가 될 수 있음
- 러한 객체는 속성과 메서드로 구성되는데, 속성(attribute) 은 객체의 상태를 의미하고 메서드(method)는 객체의 동작을 의미



(a) 객체의 구성 (b) 은행 계좌 A 객체

그림 10-1 객체의 개념

1. 객체와 클래스

■ 객체의 개념

- 예를 들어 '은행 계좌 A'라는 객체가 '잔액' 속성과 '입금하기'와 '출금하기' 메서드로 이루어진다고 가정
- [코드 10-1]에서 balance는 잔액을 저장할 변수이고, deposit은 계좌에 입금하는 함수이며 withdraw는 계좌에서 출금하는 함수
- deposit 함수와 withdraw 함수의 동작이 종료되어도 balance 변수 값은 유지되어야 하므로 함수 외부에 생성함
- 외부 변수를 함수 안에서 사용하기 위해 04행과 09행에서 global 키워드를 이용해 외부 변수인 balance를 사용한다고 선언하고 있음

1. 객체와 클래스

■ 객체의 개념

코드 10-1

```
01 balance = 0
02
03 def deposit(amount):
04   global balance
05   balance += amount
06   return balance
07
08 def withdraw(amount):
09   global balance
10   balance -= amount
11   return balance
12
13 print(deposit(100))
14 print(withdraw(30))
```

```
100
70
```

1. 객체와 클래스

■ 객체의 개념

- 만약 2개의 계좌를 관리하도록 이 프로그램을 확장한다면, 계좌가 2개이므로 잔액을 저장하는 변수가 2개여야 하고 입금하는 함수와 출금하는 함수도 2개씩 있어야 함

코드 10-2

```
01  balance1 = 0
02  balance2 = 0
03
04  def deposit1(amount):
05    global balance1
06    balance1 += amount
07    return balance1
08
09  def withdraw1(amount):
10    global balance1
11    balance1 -= amount
12    return balance1
13
14  def deposit2(amount):
```

1. 객체와 클래스

■ 객체의 개념

```
15    global balance2
16    balance2 += amount
17    return balance2
18
19  def withdraw2(amount):
20    global balance2
21    balance2 -= amount
22    return balance2
23
24  print(deposit1(100))
25  print(withdraw1(30))
26
27  print(deposit2(70))
28  print(withdraw2(20))
```

```
100
70
70
50
```

1. 객체와 클래스

■ 객체의 개념

- 만약 계좌가 2개가 아닌 10개라면 더 많은 변수와 함수가 필요하고 코드 길이가 상당히 길어질 것
- 계좌가 잔액, 입금하기, 출금하기로 이루어져 있고 여러 개의 계좌를 관리하는 프로그램을 작성하고자 한다면 객체를 이용하는 것이 바람직

1. 객체와 클래스

■ 객체의 개념

- [코드 10-3]은 객체를 이용해서 2개의 계좌를 관리하는 프로그램인데 [코드 10-2]보다 코드 길이가 짧아짐

코드 10-3

```
01 class Account:
02   def __init__(self):
03     self.balance = 0
04
05   def deposit(self, amount):
06     self.balance += amount
07     return self.balance
08
09   def withdraw(self, amount):
10     self.balance -= amount
11     return self.balance
12
13 a1 = Account()
14 print(a1.deposit(100))
15 print(a1.withdraw(30))
16
17 a2 = Account()
18 print(a2.deposit(70))
19 print(a2.withdraw(20))
```

```
100
70
70
50
```

1. 객체와 클래스

■ 클래스의 개념

- 파이썬에서 객체는 클래스를 이용해서 생성
- 클래스란 객체를 생성하는 틀(template)

그림 10-2 클래스와 객체의 의미

1. 객체와 클래스

■ 클래스의 개념

- 클래스class 는 같은 속성과 메서드를 갖는 객체들을 묶어 개념적으로 표현한 것
- 예를 들
- 어, '은행 계좌A' 객체와 '은행 계좌B' 객체는 '잔액', '입금하기', '출금하기'라는 속성과 메서드를 가지고 있음
- 이런 객체 여러 개를 묶음어 '은행 계좌'라는 클래스로 개념화할 수 있음

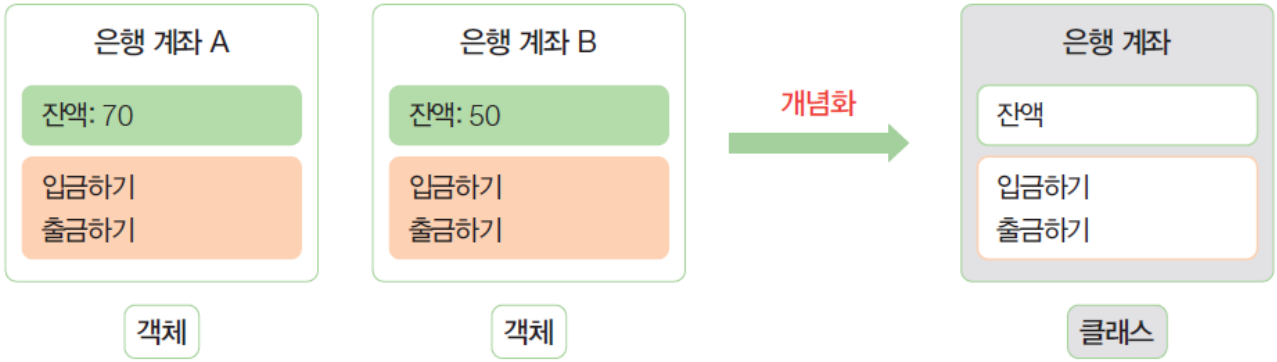


그림 10-3 객체와 클래스

- 클래스로부터 만들어진 객체를 그 클래스의 인스턴스(instance)라 함

1. 객체와 클래스

■ 파이썬의 클래스와 객체 생성

- 클래스는 class 키워드를 이용해 생성하며 일반적으로 클래스 이름의 첫 글자는 대문자로 함

| Syntax | 클래스 생성

```
class 클래스이름:
  속성1

  def 메서드1(self, …):
      …
def 메서드2(self, …):
      …
```

속성1, …, 메서드1, 메서드2, …를 갖는 '클래스이름' 클래스를 생성한다.

1. 객체와 클래스

■ 파이썬의 클래스와 객체 생성

- 클래스 안에 정의된 함수인 메서드는 클래스에 포함되어 있다는 점과 첫 번째 매개변수가 self라는 점을 제외하면 일반적인 함수를 정의하는 방법과 유사
- 첫 번째 매개변수인 self는 클래스 자신을 의미

1. 객체와 클래스

■ 파이썬의 클래스와 객체 생성

▪ 클래스 생성하기

- 은행 계좌 클래스인 Account를 생성하기
- Account 클래스는 잔액을 의미하는 balance속성과 입금하는 동작인 deposit 메서드, 출금하는 동작인 withdraw 메서드로 이루어짐

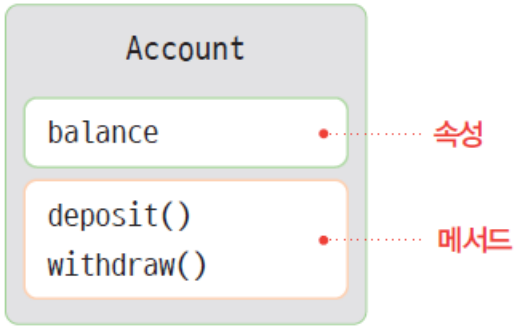


그림 10-4 Account 클래스 구조

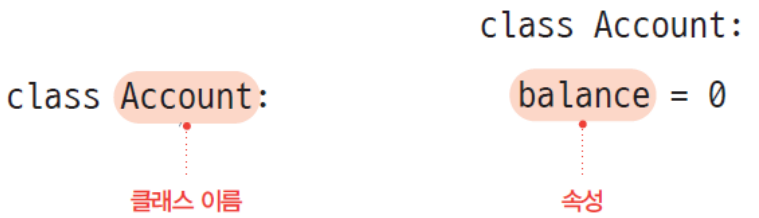
1. 객체와 클래스

■ 파이썬의 클래스와 객체 생성

▪ 클래스 생성하기

① **Account 클래스 구조 생성**: 클래스 이름의 첫 글자는 대문자로 함

② **Account 클래스의 속성 설정**: 잔액을 의미하는 속성인 balance의 초깃값을 0으로 설정



(a) Account 클래스 구조 생성 (b) balance 속성 설정

그림 10-5 Account 클래스 생성❶ – 클래스와 속성 생성

■ 파이썬의 클래스와 객체 생성

▪ 클래스 생성하기

③ **Account 클래스의 메서드 정의**:

- 입금하는 동작을 수행하는 deposit 메서드와 출금하는 동작을 수행하는 withdraw 메서드를 정의해야 함.
- deposit 메서드는 입금액인 amount를 잔액인 balance 속성에 더한 후 balance 값을 반환하는 동작을 수행
- 메서드의 첫 번째 매개변수는 클래스 자신을 의미하는 self
- 속성인 balance에 접근할 때는 self.balance와 같이 속성 이름 앞에 self.를 붙여야 함
- withdraw 메서드는 출금액인 amount를 잔액인 balance 속성에서 뺀 후 balance 값을 반환하는 동작을 수행
- deposit 메서드와 마찬가지로 첫 번째 매개변수는 self여야 함

■ 파이썬의 클래스와 객체 생성

▪ 클래스 생성하기

③ **Account 클래스의 메서드 정의**:

```
class Account:
  balance = 0
  def deposit(self, amount):
    self.balance += amount
    return self.balance

  def withdraw(self, amount):
    self.balance -= amount
    return self.balance
```

deposit 메서드 정의

withdraw 메서드 정의

그림 10-6 Account 클래스 생성❷ – 메서드 정의

■ 파이썬의 클래스와 객체 생성

▪ 클래스 생성하기

- Account 클래스를 생성하는 전체 코드는 [코드 10-4]

코드 10-4

```
01  class Account:
02    balance = 0
03
04    def deposit(self, amount):
05      self.balance += amount
06      return self.balance
07
08    def withdraw(self, amount):
09      self.balance -= amount
10      return self.balance
```

1. 객체와 클래스

■ 파이썬의 클래스와 객체 생성

▪ 객체 생성하기

- 객체는 클래스를 이용해서 생성할 수 있고, 형식은 다음과 같음

| Syntax | 객체 생성

```
객체이름 = 클래스이름()
```

'클래스이름' 클래스의 인스턴스인 '객체이름' 객체를 생성한다.

■ 파이썬의 클래스와 객체 생성

▪ 객체 생성하기

- 앞에서 정의한 Account 클래스를 이용해서 a1 객체를 생성하는 코드는 [코드 10-5]와 같음
- a1 객체를 Account 클래스의 인스턴스라 함

코드 10-5

```
01  a1 = Account()
```

- 이렇게 생성된 a1 객체에서 balance 속성 값은 0으로 초기화됨

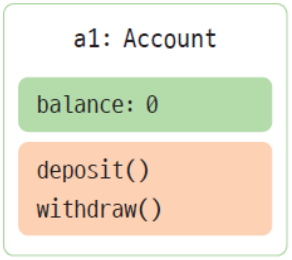


그림 10-7 Account 클래스의 인스턴스인 a1 객체

■ 파이썬의 클래스와 객체 생성

▪ 속성과 메서드에 접근하기

- 객체의 속성이나 메서드에 접근하려면 객체 이름에 마침표(.)를 붙이고 속성 이름 또는 메서드 이름을 적으면 됨

| Syntax | 속성 또는 메서드에 접근하기

```
객체이름.속성이름
객체이름.메서드이름()
```

객체의 속성 또는 메서드에 접근한다.

■ 파이썬의 클래스와 객체 생성

▪ 속성과 메서드에 접근하기

- a1 객체의 deposit 메서드를 다음과 같이 호출하면 매개변수인 100이 amount로 전달되고 self에는 a1 객체가 전달됨
- a1 객체의 balance 속성 값을 amount의 값인 100만큼 증가시키고 balance 값인 100을 반환함

```
a1.deposit(100)
```

■ 파이썬의 클래스와 객체 생성

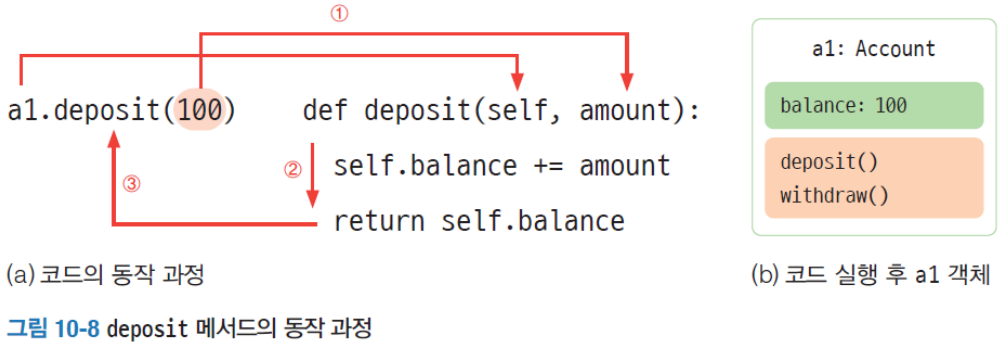
▪ 속성과 메서드에 접근하기

- 결국 [코드 10-6]을 실행하면 balance 속성 값은 초깃값 0에서 전달받은 100만큼 증가해서 100이 되고, 이 값을 출력

코드 10-6

```
01  print(a1.deposit(100))
```

```
100
```



(a) 코드의 동작 과정 (b) 코드 실행 후 a1 객체

그림 10-8 deposit 메서드의 동작 과정

■ 파이썬의 클래스와 객체 생성

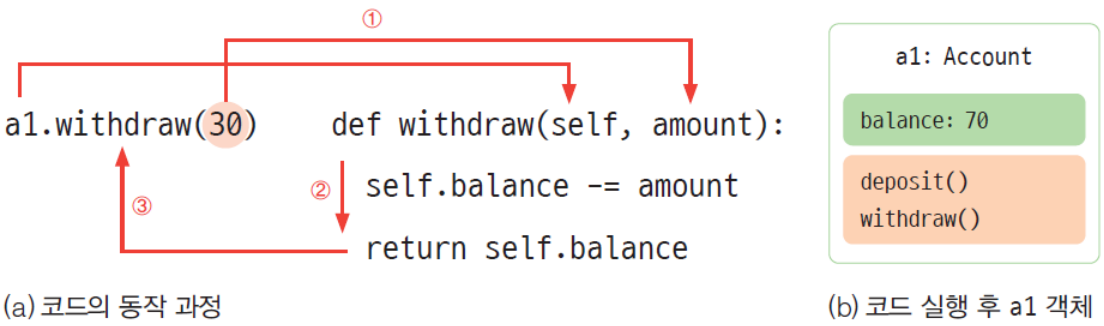
▪ 속성과 메서드에 접근하기

- 다음으로 [코드 10-7]에서 a1 객체의 withdraw 메서드를 호출하면 balance 속성 값이 전달받은 30만큼 감소하여 70이 되고, 이 값을 출력

코드 10-7

```
01  print(a1.withdraw(30))
```

```
70
```



(a) 코드의 동작 과정 (b) 코드 실행 후 a1 객체

그림 10-9 withdraw 메서드의 동작 과정

1. 객체와 클래스

■ 파이썬의 클래스와 객체 생성

▪ 속성과 메서드에 접근하기

- [코드 10-8]은 Account 클래스를 이용해서 또 다른 객체인 a2를 생성한 다음 a2에서 70을 입금하고 20을 출금

코드 10-8

```
01  a2 = Account()
02  print(a2.deposit(70))
03  print(a2.withdraw(20))
```

```
70
50
```

1. 객체와 클래스

■ 생성자

- 생성자(constructor): 객체가 생성될 때 자동으로 호출되는 메서드
- 일반적으로 초기화 동작을 담당
- 생성자의 이름은 __init__으로 정해져 있으며, self는 필수 매개변수로 맨앞에 위치해야 함

| Syntax | 생성자 생성

```
class 클래스이름:
  def __init__(self, …):
    …
```

생성자가 있는 '클래스이름' 클래스를 생성한다.

■ 생성자

▪ 속성을 0으로 초기화하기

- 앞에서 살펴본 Account 클래스에서 balance 속성을 0으로 초기화하는 동작을 생성자가 수행하도록 수정할 수 있음

```
class Account:
    balance = 0
```



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

생성자 설정

그림 10-10 생성자가 balance 속성을 초기화

■ 생성자

▪ 속성을 0으로 초기화하기

- balance 속성을 0으로 초기화하는 생성자를 포함하여 Account 클래스를 정의하는 전체 코드는 [코드 10-9]와 같음

코드 10-9

```
01  class Account:
02    def __init__(self):
03      self.balance = 0
04
05    def deposit(self, amount):
06      self.balance += amount
07      return self.balance
08
09    def withdraw(self, amount):
10      self.balance -= amount
11      return self.balance
```

■ 생성자

▪ 속성을 0으로 초기화하기

- 이와 같이 정의한 Account 클래스를 이용해서 [코드 10-10]과 같이 a1 객체를 생성하면 자동으로 생성자가 실행되어 balance 속성 값이 0으로 초기화됨

코드 10-10

```
01  a1 = Account()
```

■ 생성자

▪ 속성을 입금액으로 초기화하기

- Account 클래스를 이용해서 객체를 생성할 때 balance 속성의 초깃값을 입금하는 금액으로 바로 설정할 수 있음
- 생성자에 amount 매개변수를 추가하고 이 값이 바로 balance 속성의 초깃값이 되도록 함

```
class Account:
  def __init__(self, amount):        매개변수 추가
    self.balance = amount
...
```

그림 10-11 생성자에 매개변수를 추가해 초깃값 설정

■ 생성자

▪ 속성을 입금액으로 초기화하기

- 이런 내용을 반영하여 Account 클래스를 정의하는 전체 코드는 [코드 10-11]과 같음

코드 10-11

```
01  class Account:
02    def __init__(self, amount):
03      self.balance = amount
04
05    def deposit(self, amount):
06      self.balance += amount
07      return self.balance
08
09    def withdraw(self, amount):
10      self.balance -= amount
11      return self.balance
```

■ 생성자

▪ 속성을 입금액으로 초기화하기

- 이와 같이 정의된 Account 클래스를 이용해서 [코드 10-12]와 같이 a1 객체를 생성하면 50이 생성자의 amount 매개변수로 전달되어 balance 속성이 50으로 초기화됨

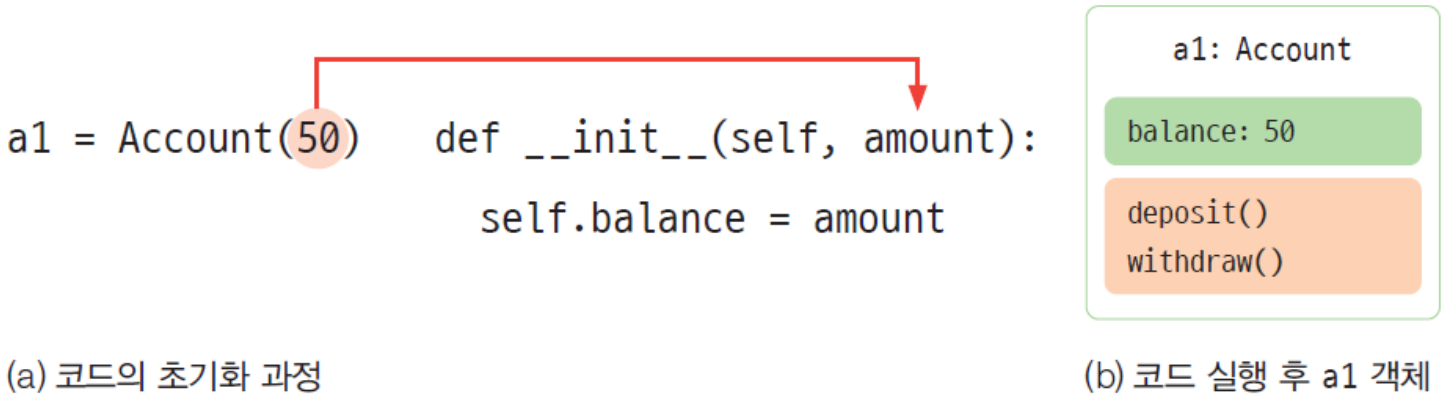
코드 10-12

```
01  a1 = Account(50)
```

■ 생성자

▪ 속성을 입금액으로 초기화하기

- 생성된 a1 객체의 구조는 다음과 같음



(a) 코드의 초기화 과정 (b) 코드 실행 후 a1 객체

그림 10-12 매개변수가 있는 생성자를 포함한 클래스로 a1 객체 생성하기

■ 생성자

▪ 속성을 입금액으로 초기화하기

- 이 상태에서 [코드 10-13]을 실행하면 balance 속성값은 초깃값 50에서 매개변수로 전달받은 100만큼 증가해서 150이 되고, 이 값을 반환받아 출력함

코드 10-13

```
01  print(a1.deposit(100))
```

```
150
```

■ 생성자

▪ 속성을 입금액으로 초기화하기

- 그런데 [코드 10-14]와 같이 객체를 생성할 때 실수로 매개변수를 전달하지 않으면 오류가 발생

코드 10-14

```
01  a2 = Account()
```

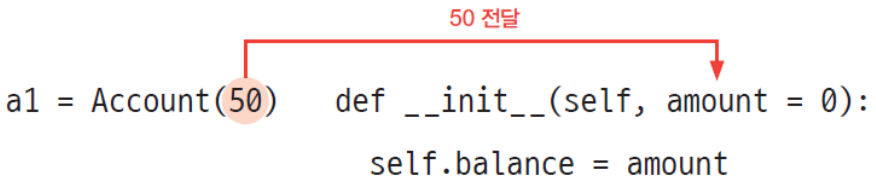
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-46b353017a16> in <cell line: 1>()
----> 1 a2 = Account()


TypeError: Account.__init__() missing 1 required positional argument: 'amount'
```

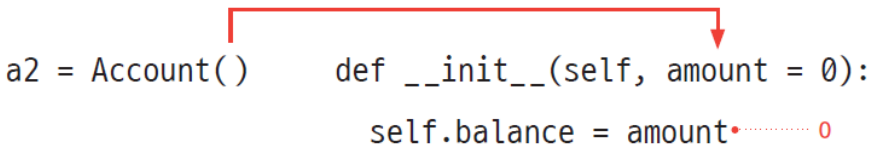
■ 생성자

▪ 속성을 입금액으로 초기화하기

- 이런 문제는 생성자의 매개변수에 초깃값을 지정하면 해결됨
- 객체를 생성할 때 50을 전달하면 생성자의 amount 매개변수의 값은 전달받은 50이 됨
- 반면 객체를 생성할 때 값을 전달하지 않으면 생성자의 amount 값은 초깃값인 0으로 지정됨



(a) 객체를 생성할 때 매개변수를 전달



(b) 객체를 생성할 때 매개변수를 전달하지 않음

그림 10-13 생성자의 매개변수에 초깃값 지정

■ 생성자

▪ 속성을 입금액으로 초기화하기

- [코드 10-15]는 이와 같이 초깃값을 지정하는 생성자를 갖는 Account 클래스를 이용해서 a1 객체와 a2 객체를 생성

코드 10-15

```
01  class Account:
02    def __init__(self, amount=0):
03      self.balance = amount
04
05    def deposit(self, amount):
```

■ 생성자

▪ 속성을 입금액으로 초기화하기

```
06      self.balance += amount
07      return self.balance
08
09    def withdraw(self, amount):
10      self.balance -= amount
11      return self.balance
12
13  a1 = Account(50)
14  print(a1.deposit(70))
15
16  a2 = Account()
17  print(a2.deposit(100))
```

```
120
100
```

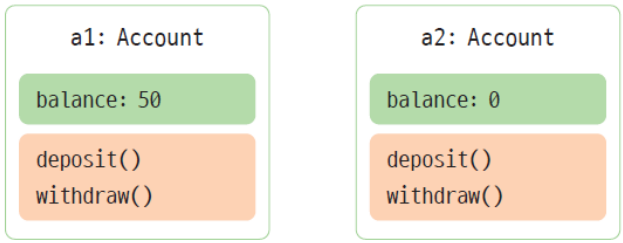


그림 10-14 Account 클래스로 생성한 a1 객체와 a2 객체

private 속성

- 객체 이름에 마침표(.)를 붙이고 속성 이름을 적으면 객체의 속성에 접근할 수 있다고 앞에서 설명함

```
객체이름.속성이름
```

private 속성

- [코드 10-16]의 14행에서는 이와 같이 속성에 접근해서 a1 객체의 balance 속성 값을 30으로 변경함

코드 10-16

```
01 class Account:
02   def __init__(self, amount=0):
03     self.balance = amount
04
05   def deposit(self, amount):
06     self.balance += amount
07     return self.balance
08
09   def withdraw(self, amount):
10     self.balance -= amount
11     return self.balance
12
13 a1 = Account(50)
14 a1.balance = 30
15 print(a1.balance)
```

```
30
```

private 속성

- 객체 지향 프로그래밍에서는 객체의 속성과 메서드를 하나의 캡슐처럼 묶어서 관리하는 캡슐화(encapsulation) 가 핵심
- 객체 내부에 구현된 세부 사항을 숨기고 외부에는 객체의 속성과 메서드에 접근하는 인터페이스만 제공하는 것
- 같은 맥락으로 보면 [코드 10-16]과 같이 객체 외부에서 객체의 속성에 직접 접근하는 것은 바람직하지 않음
- 이런 경우에 속성을 private으로 설정하여 접근하지 못하도록 할 수 있고, 속성 이름 앞에 __를 붙이면 그 속성은 private 속성이 됨

```
__속성이름
```

private 속성

- [코드 10-17]은 Account 클래스의 속성을 private으로 정의한 상태에서 14행에서 이 속성에 접근하려고 했으므로 오류가 발생

코드 10-17

```
01  class Account:
02    def __init__(self, amount=0):
03      self.__balance = amount
04
05    def deposit(self, amount):
06      self.__balance += amount
07      return self.__balance
08
09    def withdraw(self, amount):
10      self.__balance -= amount
11      return self.__balance
12
13  a1 = Account(50)
14  print(a1.__ balance)
```

```
⋮
AttributeError: 'Account' object has no attribute '__balance'
```

private 속성

private 속성 값 얻기

- private 속성 값을 얻으려면 속성 값을 반환하는 메서드를 이용 하면 됨
- [코드 10-18]의 13~14행에서 정의한 get_balance가 __balance 속성 값을 반환하는 메서드

코드 10-18

```
01  class Account:
02    def __init__(self, amount=0):
03      self.__balance = amount
04
05    def deposit(self, amount):
06      self.__balance += amount
07      return self.__balance
```

1. 객체와 클래스

■ private 속성

▪ private 속성 값 얻기

```
08
09    def withdraw(self, amount):
10        self.__balance -= amount
11        return self.__balance
12
13    def get_balance(self):
14        return self.__balance
15
16 a1 = Account(50)
17 print(a1.get_balance())
```

1. 객체와 클래스

LAB 원 클래스와 객체

- 원 클래스인 Circle을 생성하고 이 클래스를 이용해서 객체를 생성하기
- Circle 클래스는 반지름을 의미하는 __radius 속성과 반지름을 확인하는 get_radius 메서드, 반지름을 설정하는 set_radius 메서드, 원의 넓이를 구하는 get_area 메서드, 원의 둘레를 구하는 get_circumference 메서드로 이루어짐

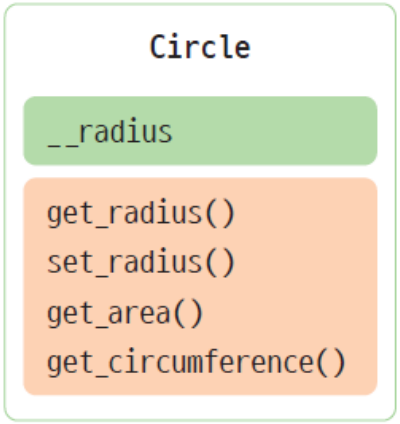


그림 10-15 Circle 클래스

1. 객체와 클래스

LAB 원 클래스와 객체

- Circle 클래스를 생성하고 이를 이용해 c1 객체를 생성해서 원의 넓이와 둘레 등을 구하는 프로그램을 만들어보기

코드 10-19

```
01  import math
02
03  class Circle:
04    def __init__(self, radius=10):
05      self.__radius = radius
06
07    def get_radius(self):
```

LAB 원 클래스와 객체

```
08      return self.__radius
09
10    def set_radius(self, radius):
11      self.__radius = radius
12
13    def get_area(self):
14      return math.pi*self.__radius**2
15
16    def get_circumference(self):
17      return 2*math.pi*self.__radius
18
19  c1 = Circle(7)
20  print('넓이:', c1.get_area())
21  print('둘레:', c1.get_circumference())
22  c1.set_radius(10)
23  print('넓이:', c1.get_area())
24  print('둘레:', c1.get_circumference())
```

```
넓이: 153.93804002589985
둘레: 43.982297150257104
넓이: 314.1592653589793
둘레: 62.83185307179586
```

1. 객체와 클래스

LAB 학생 클래스와 객체

- 학생 클래스인 Student를 생성하고 이 클래스를 이용해서 객체를 생성하기
- Student 클 래스는 이름을 의미하는 __name 속성, 수학 성적을 의미하는 __math 속성, 컴퓨터 성적을 의미 하는 __computer 속성, 이름을 확인하는 get_name 메서드, 성적 평균을 확인하는 get_average 메서드, 수학 성적을 설정하는 set_math 메서드, 컴퓨터 성적을 설정하는 set_computer 메서드로 이루어짐

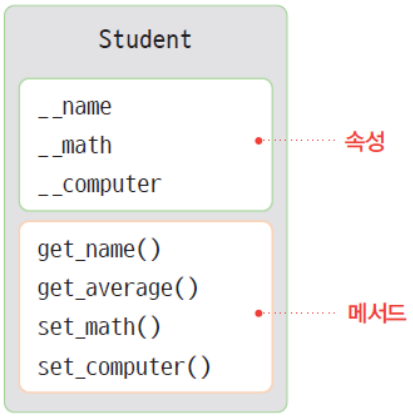


그림 10-16 Student 클래스

1. 객체와 클래스

LAB 학생 클래스와 객체

- 다음은 Student 클래스를 생성하고 이를 이용해 s1 객체를 생성한 다음 성적 평균을 구하고 컴퓨터 성적을 변경하는 등의 동작을 하는 프로그램

코드 10-20

```
01  class Student:
02    def __init__(self, name, math, computer):
03      self.__name = name
04      self.__math = math
05      self.__computer = computer
06
07    def get_name(self):
08      return self.__name
09
10   def get_average(self):
11      return (self.__math + self.__computer)/2
12
13    def set_math(self, math):
14      self.__math = math
```

LAB 학생 클래스와 객체

```
15
16    def set_computer(self, computer):
17        self.__computer = computer
18
19  s1 = Student('hanbit', 95, 89)
20  print(s1.get_name(), s1.get_average())
21  s1.set_computer(97)
22  print(s1.get_name(), s1.get_average())
```

```
hanbit 92.0
hanbit 96.0
```

Section 10.2

상속

2. 상속

■ 상속의 개념

- 상속(inheritance): 기존 클래스의 속성과 메서드를 공유하는 메커니즘
- 상속을 통해 정의되는 클래스를 자식 클래스(child class) 또는 하위 클래스(subclass)라고 함
- 상속해 주는 클래스를 부모 클래스(parent class) 또는 상위 클래스(superclass) 라 함

그림 10-17 부모 클래스와 자식 클래스의 관계

2. 상속

■ 상속의 개념

- 이과생 클래스와 문과생 클래스가 [그림 10-18]과 같다고 가정

그림 10-18 이과생 클래스와 문과생 클래스

2. 상속

■ 상속의 개념

- 여기에서 **이름** 속성, **컴퓨터성적** 속성, **컴퓨터성적설정** 메서드, **이름알아내기** 메서드는 두 개의 클래스에서 공통적
- 공통 속성과 메서드를 묶어 **학생**이라는 클래스를 정의하고, **이과생**과 **문과생** 클래스가 **학생** 클래스를 상속받게 함
- 그리고 **이과생** 클래스는 **과학성적** 속성과 **과학성적설정** 메서드로 구성하고, **문과생** 클래스는 **사회성적** 속성과 **사회성적설정** 메서드로 구성
- **이과생** 클래스에서 **성적평균알아내기** 메서드의 동작은 **문과생** 클래스의 **성적평균알아내기** 메서드의 동작과 다르므로 **이과생**과 **문과생** 클래스 각각에 **성적평균알아내기** 메서드를 정의함

2. 상속

■ 상속의 개념

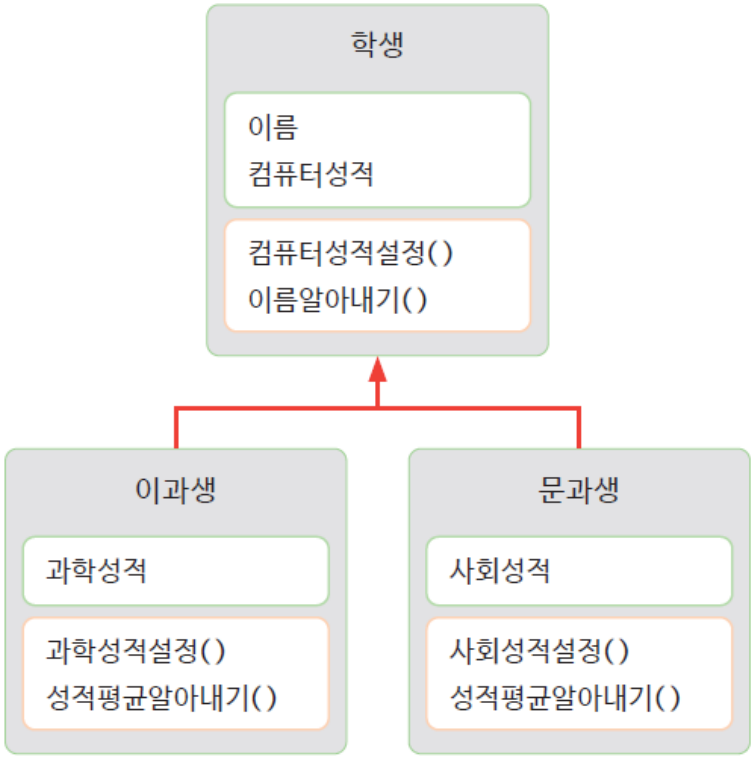


그림 10-19 학생 클래스를 상속받는 이과생 클래스와 문과생 클래스

2. 상속

■ 상속의 개념

- 이렇게 정의한 **이과생** 클래스와 **문과생** 클래스는 **학생** 클래스의 **이름** 속성, **컴퓨터성적** 속성, **컴퓨터성적설정** 메서드, **이름알아내기** 메서드를 상속받아 자신의 속성과 메서드인 것처럼 사용할 수 있음
- 만약 **컴퓨터성적설정** 메서드나 **이름알아내기** 메서드를 수정할 일이 생기면 [그림 10-18]과 같은 구조에서는 **이과생** 클래스와 **문과생** 클래스에서 모두 수정해야 하지만 [그림 10-19]와 같이 상속을 이용하면 학생 클래스만 수정하면 되므로 편리

2. 상속

■ 상속과 생성자

- 자식 클래스를 정의할 때 자식 클래스 이름 옆의 괄호 안에 부모 클래스 이름을 넣음
- 부모 클래스의 속성과 메서드는 자식 클래스에 상속되고, 자식 클래스는 상속받은 것에 새로운 속성과 메서드를 추가할 수 있음

| Syntax | 자식 클래스 생성

```
class 자식클래스이름(부모클래스이름):
  def 메서드1(self, …):
      …
  def 메서드2(self, …):
      …
```

'부모클래스이름'을 상속받는 '자식클래스이름'을 생성한다.

2. 상속

■ 상속과 생성자

- [그림 10-20]은 부모 클래스 Student를 상속하는 자식 클래스 Science와 Liberalarts를 정의하는 코드의 구조
- Student 클래스에 속한 속성과 메서드는 Science 클래스와 Liberalarts 클래스에 상속됨

```
class Student:
  def 메서드1(self, …):
    …

class Science(Student):
  def 메서드2(self, …):
    …

class Liberalarts(Student):
  def 메서드3(self, …):
    …
```

부모 클래스

Student를 상속받는 자식 클래스

그림 10-20 부모 클래스 Student를 상속받는 자식 클래스 Science와 Liberalarts

2. 상속

■ 상속과 생성자

- [그림 10-20]을 프로그램으로 작성해 보기
- 우선 학생을 의미하는 Student 클래스를 정의
- Student 클래스의 생성자는 이름인 name과 컴퓨터 성적인 computer를 초기화
- set_computer 메서드는 컴퓨터 성적을 설정하고, get_name 메서드는 이름을 반환

코드 10-21

```
01  class Student:
02    def __init__(self, name, computer):
03      self.name = name
04      self.computer = computer
05
06    def set_computer(self, computer):
07      self.computer = computer
08
09    def get_name(self):
10      return self.name
```

2. 상속

■ 상속과 생성자

- 다음으로 Student를 상속하는 자식 클래스인 Science를 정의
- Science 클래스의 생성자는 03행에서 부모 클래스의 생성자를 호출하여 name과 computer 속성을 초기화
- 또, 자식 클래스의 science 속성을 초기화
- set_science 메서드는 과학 성적을 설정하고, get_average 메서드는 성적 평균을 반환

코드 10-22

```
01  class Science(Student):
02    def __init__(self, name, computer, science):
03      super().__init__(name, computer)
04      self.science = science
05
06    def set_science(self, science):
07      self.science = science
08
09    def get_average(self):
10      return (self.science+self.computer)/2
```

2. 상속

■ 상속과 생성자

- Student를 상속하는 또 다른 자식 클래스인 Liberalarts를 정의
- Liberalarts 클래스의 생성자는 부모 클래스의 생성자를 호출해서 name과 computer 속성을 초기화
- 또한사회 성적인 social 속성을 초기화
- set_social 메서드는 사회 성적을 설정하고, get_average 메서드는 성적 평균을 반환

코드 10-23

```
01  class Liberalarts(Student):
02    def __init__(self, name, computer, social):
03      super().__init__(name, computer)
04      self.social = social
05
06    def set_social(self, social):
07      self.social = social
08
09    def get_average(self):
10      return (self.social+self.computer)/2
```

2. 상속

■ 상속과 생성자

- 부모 클래스인 Student, 자식 클래스인 Science와 Liberalarts의 관계는 [그림 10-21]과 같음
- Science 클래스와 Liberalarts 클래스는 부모 클래스인 Student 클래스의 name, computer 속성과 set_computer(), get_name() 메서드를 상속받음

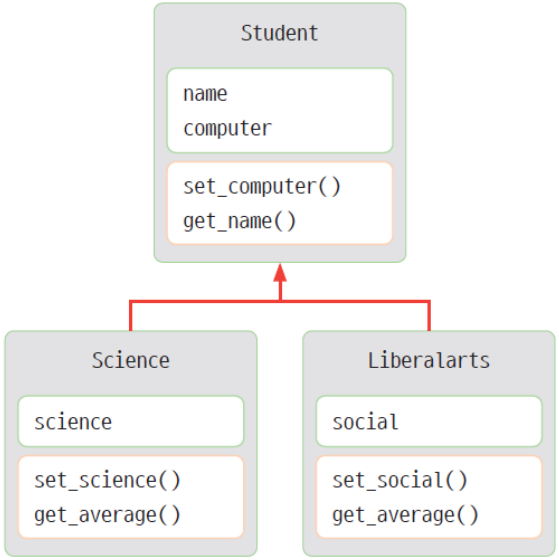


그림 10-21 Student 클래스를 상속받는 Science와 Liberalarts 클래스

■ 상속과 생성자

▪ 자식 클래스의 객체

- Science 클래스의 객체인 st1을 생성하는 방법은 [코드 10-24]와 같음

코드 10-24

```
01  st1 = Science('hanbit1', 90, 80)
```

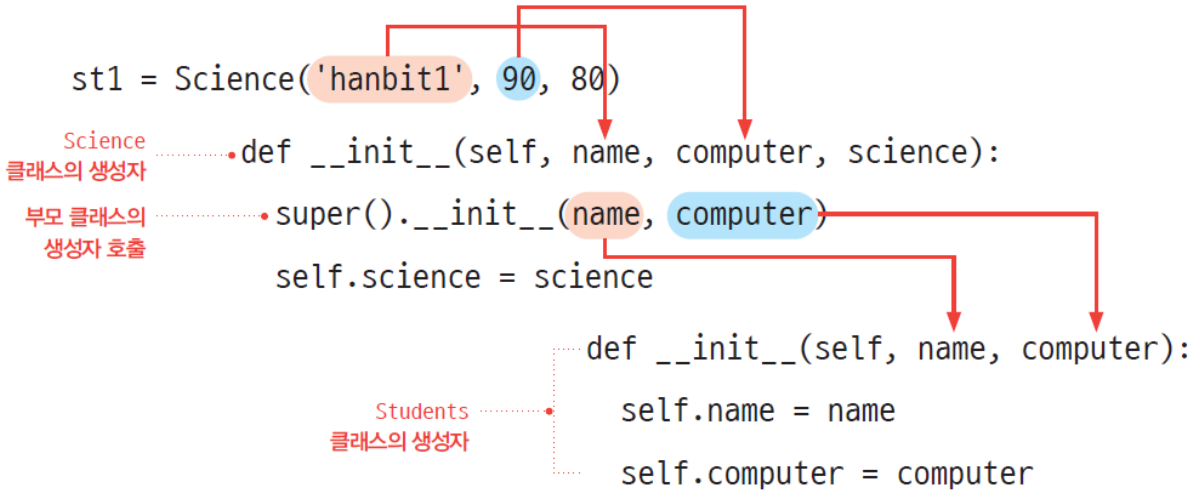


그림 10-22 st1 객체의 생성자 동작 과정

2. 상속

■ 상속과 생성자

▪ 자식 클래스의 객체

- Science 클래스의 객체를 생성할 때 생성자가 실행됨
- 생성자는 부모 클래스인Student 클래스의 생성자를 호출하여 name과 computer 속성을 초기화함
- 그리고 자신의 science 속성을 초기화
- name은 'hanbit1'이 되고, computer는 90, science는 80이 됨
- 만약 Science 클래스의 생성자가 Student 클래스의 생성자를 호출하지 않으면 name과 computer 속성이 생성되지 않아 이용할 수 없음

■ 상속과 생성자

▪ 자식 클래스의 객체

- [코드 10-25]는 st1 객체의 성적 평균을 확인하기 위해 get_average() 메서드를 호출
- 컴퓨터 성적인 computer와 과학 성적인 science의 평균인 85가 출력됨

코드 10-25

```
01  print(st1.get_average())
```

```
85.0
```

■ 상속과 생성자

▪ 자식 클래스의 객체

- [코드 10-26]은 st1 객체의 컴퓨터 성적인 computer를 100으로 변경하기 위해 set_computer() 메서드를 호출하고 평균 성적을 확인함
- set_computer()는 부모 클래스인 Student 클래스의 메서드인데 st1 객체로 상속되어 문제없이 동작함

코드 10-26

```
01  st1.set_computer(100)
02  print(st1.get_average())
```

```
90.0
```

■ 상속과 생성자

▪ 자식 클래스의 객체

- [코드 10-27]은 Liberalarts 클래스의 객체 st2를 생성한 후 성적 평균을 출력하고, 사회 성적을 변경한 다음 이름과 평균 성적을 출력함

코드 10-27

```
01  st2 = Liberalarts('hanbit2', 98, 88)
02  print(st2.get_average())
03  st2.set_social(96)
04  print(st2.get_name(), ':', st2.get_average())
```

```
93.0
hanbit2 : 97.0
```

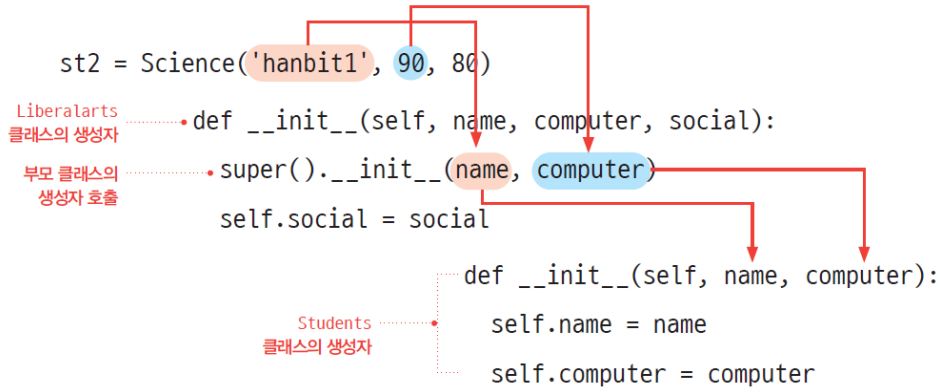


그림 10-23 st2 객체의 생성자 동작 과정

2. 상속

LAB 도형 클래스를 상속받는 원과 직사각형 클래스의 객체 생성

- Figure 클래스를 상속받는 Circle 클래스와 Rectangle 클래스를 생성하고, Circle 클래스와 Rectangle 클래스를 이용해서 객체를 생성해 보기
- Figure 클래스는 넓이를 의미하는 area 속성, 둘레를 의미하는 perimeter 속성, 넓이를 반환하는 get_area 메서드, 둘레를 반환하는 get_perimeter 메서드로 이루어짐
- 이를 상속받는 Circle 클래스는 반지름을 의미하는 radius 속성, 반지름을 설정하는 set_radius 메서드, 반지름을 반환하는 get_radius 메서드로 이루어짐
- 또한 Rectangle 클래스는 가로를 의미하는 width 속성, 세로를 의미하는height 속성, 가로를 설정하는 set_width 메서드, 세로를 설정하는 set_height 메서드, 가로를 반환하는 get_width 메서드, 세로를 반환하는 get_height 메서드로 이루어짐

2. 상속

LAB 도형 클래스를 상속받는 원과 직사각형 클래스의 객체 생성

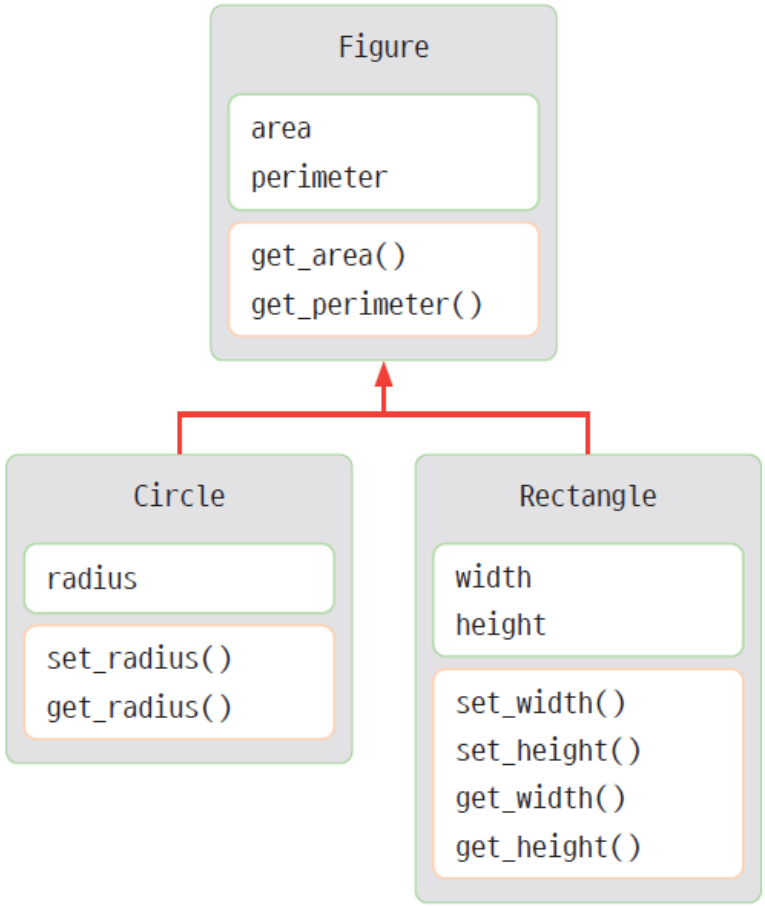


그림 10-24 Figure 클래스를 상속받는 Circle과 Rectangle 클래스

2. 상속

LAB 도형 클래스를 상속받는 원과 직사각형 클래스의 객체 생성

- 다음은 Figure 클래스를 상속받는 Circle과 Rectangle 클래스를 생성한 다음, Circle 클래스를 이용해서 f1 객체를 생성하고 Rectangle 클래스를 이용해서 f2 객체를 생성하여 이들 객체의 넓이와 둘레를 확인하는 프로그램

코드 10-28

```
01  class Figure:
02    def __init__(self, area, perimeter):
03      self.area = area
04      self.perimeter = perimeter
05
06    def get_area(self):
07      return self.area
08
09    def get_perimeter(self):
10      return self.perimeter
```

2. 상속

LAB 도형 클래스를 상속받는 원과 직사각형 클래스의 객체 생성

```
11
12  class Circle(Figure):
13    def __init__(self, radius=10):
14      area = 3.14*radius**2
15      perimeter = 2*3.14*radius
16      super().__init__(area, perimeter)
17      self.radius = radius
18
19    def set_radius(self, radius):
20      self.radius = radius
21      self.area = 3.14*radius**2
22      self.perimeter = 2*3.14*radius
23
24    def get_radius(self):
25      return self.radius
26
```

2. 상속

LAB 도형 클래스를 상속받는 원과 직사각형 클래스의 객체 생성

```
27 class Rectangle(Figure):
28   def __init__(self, width=10, height=10):
29     area = width*height
30     perimeter = 2*width+2*height
31     super().__init__(area, perimeter)
32     self.width = width
33     self.height = height
34
35   def set_width(self, width):
36     self.width = width
37     self.area = width*self.height
38     self.perimeter = 2*width+2*self.height
39
40   def set_height(self, height):
41     self.height = height
42     self.area = self.width*height
43     self.perimeter = 2*self.width+2*height
44
```

2. 상속

LAB 도형 클래스를 상속받는 원과 직사각형 클래스의 객체 생성

```
45    def get_width(self):
46      return self.width
47
48    def get_height(self):
49      return self.height
50
51  f1 = Circle(7)
52  print(f1.get_area(), f1.get_perimeter())
53  f1.set_radius(10)
```

```
54  print(f1.get_area(), f1.get_perimeter())
55
56  f2 = Rectangle(5, 4)
57  print(f2.get_area(), f2.get_perimeter())
58  f2.set_width(7)
59  print(f2.get_area(), f2.get_perimeter())
```

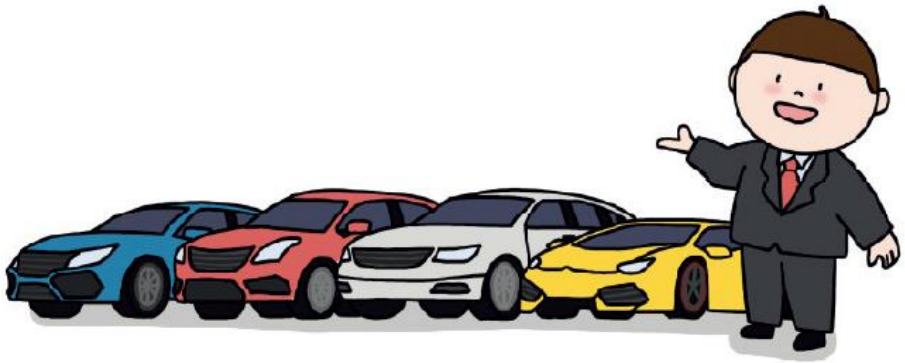
```
153.86 43.96
314.0 62.800000000000004
20 18
28 22
```

Section 10.3

[플러스 예제] 자동차 클래스와 객체

3. [플러스 예제] 자동차 클래스와 객체

- 친구들과의 여행에서 자동차를 렌트하려고 함
- 비용이 적게 드는 자동차를 선택하기 위해 자동차 모델별 이동 거리에 따른 연료량을 알고 싶음
- 자동차 객체와 상속을 이용해서 자동차 모델별 연료량을 계산하는 프로그램을 만들어보기



3. [플러스 예제] 자동차 클래스와 객체

■ 자동차 클래스 생성

- Car 클래스는 모델명을 의미하는 __model 속성, 연비를 의미하는 __fuelefficiency 속성, 모델명을 확인하는 get_model 메서드, 연비를 확인하는 get_fuelefficiency 메서드, 연비를 설정하는 set_fuelefficiency 메서드, 연료량을 구하는 get_fuelamount 메서드로 이루어짐

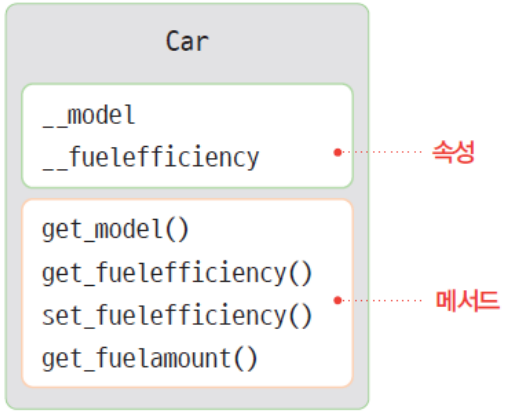


그림 10-25 Car 클래스

3. [플러스 예제] 자동차 클래스와 객체

■ 자동차 클래스 생성

- 클래스는 class 키워드를 이용해서 생성함
- 또한 객체를 생성할 때 자동으로 호출되는 메서드인 생성자를 정의함
- 생성자는 모델명인 __model 속성과 연비인 __fuelefficiency 속성을 초기화함
- 속성명 앞에 __를 붙여 private 속성으로 설정

```
class Car:
  def __init__(self, model, fuelefficiency):
    self.__model = model
    self.__fuelefficiency = fuelefficiency
```

3. [플러스 예제] 자동차 클래스와 객체

■ 자동차 클래스 생성

- 모델명인 __model 속성 값을 확인하는 get_model 메서드와 연비인 __fuelefficiency 속성 값을 확인하는 get_fuelefficiency 메서드를 정의

```
class Car:
⋮
  def get_model(self):
    return self.__model

  def get_fuelefficiency(self):
    return self.__fuelefficiency
```

3. [플러스 예제] 자동차 클래스와 객체

■ 자동차 클래스 생성

- 연비인 __fuelefficiency 속성 값을 설정하는 set_fuelefficiency 메서드를 정의

```
class Car:
⋮
  def set_fuelefficiency(self, fuelefficiency):
    self.__fuelefficiency = fuelefficiency
```

3. [플러스 예제] 자동차 클래스와 객체

■ 자동차 클래스 생성

- 연료량은 주행거리를 연비로 나눈 값
- 전달받은 주행거리를 연비인 __fuelefficiency로 나누어 연료량을 구하는 get_fuelamount 메서드를 정의

```
class Car:
⋮
  def get_fuelamount(self, distance):
    return distance/self.__fuelefficiency
```

3. [플러스 예제] 자동차 클래스와 객체

■ 자동차 객체 생성

- 정의된 Car 클래스를 이용해서 car1 객체를 생성하는 코드는 다음과 같음
- car1은 객체명이고, Car는 클래스명
- 그리고 'GV70'과 12는 생성자에게 전달되는 매개변수

```
car1 = Car('GV70', 12)
```

3. [플러스 예제] 자동차 클래스와 객체

■ 자동차 객체 생성

- 이렇게 생성된 car1 객체의 구조는 [그림 10-26]과 같음
- __model은 'GV70'으로 초기화되고 __fuelefficiency는 12로 초기화됨

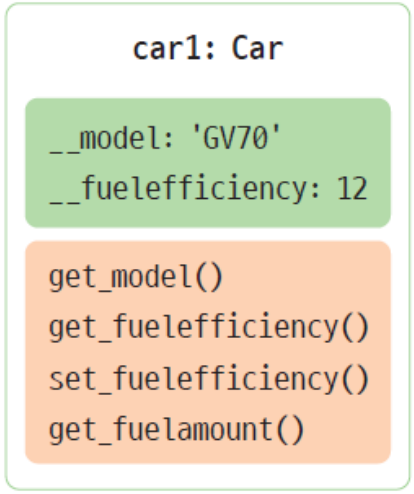


그림 10-26 car1 객체

3. [플러스 예제] 자동차 클래스와 객체

■ 메서드 활용

- 객체의 메서드에 접근하려면 객체이름에 마침표(.)를 붙이고 메서드이름을 적기

```
객체이름.메서드이름()
```

- car1 객체의 get_model 메서드를 호출하면 모델명인 __model 속성 값인 'GV70'을 반환하고, get_fuelefficiency 메서드를 호출하면 연비인 __fuelefficiency 속성 값인 12를 반환

```
print('모델명:', car1.get_model())
print('연비:', car1.get_fuelefficiency())
```

```
모델명: GV70
연비: 12
```

3. [플러스 예제] 자동차 클래스와 객체

■ 메서드 활용

- 연료량을 구하는 get_fuelamount 메서드를 호출
- 이때 주행거리를 매개변수로 전달해야 하는데 여기서는 240을 전달
- 전달한 240이 get_fuelamount 메서드의 distance에 저장되고, 주행거리와 연비를 이용해 계산한 연료량을 반환

```
print('연료량:', car1.get_fuelamount(240))
```

```
연료량: 20.0
```

3. [플러스 예제] 자동차 클래스와 객체

■ 메서드 활용

▪ 전체 프로그램

```
01 class Car:
02   def __init__(self, model, fuelefficiency):
03     self.__model = model
04     self.__fuelefficiency = fuelefficiency
05
06   def get_model(self):
07     return self.__model
08
09   def get_fuelefficiency(self):
10     return self.__fuelefficiency
11
12   def set_fuelefficiency(self, fuelefficiency):
13     self.__fuelefficiency = fuelefficiency
```

3. [플러스 예제] 자동차 클래스와 객체

■ 메서드 활용

▪ 전체 프로그램

```
14
15    def get_fuelamount(self, distance):
16        return distance/self.__fuelefficiency
17
18  car1 = Car('GV70', 12)
19  print('모델명:', car1.get_model())
20  print('연비:', car1.get_fuelefficiency())
21  distance = float(input('주행 거리(km): '))
22  print('연료량:', car1.get_fuelamount(distance))
```

```
모델명: GV70
연비: 12
주행 거리(km): 240
연료량: 20.0
```