기초빅데이터프로그래밍

객체지향 프로그래밍

서강대학교
SOGANG UNIVERSITY

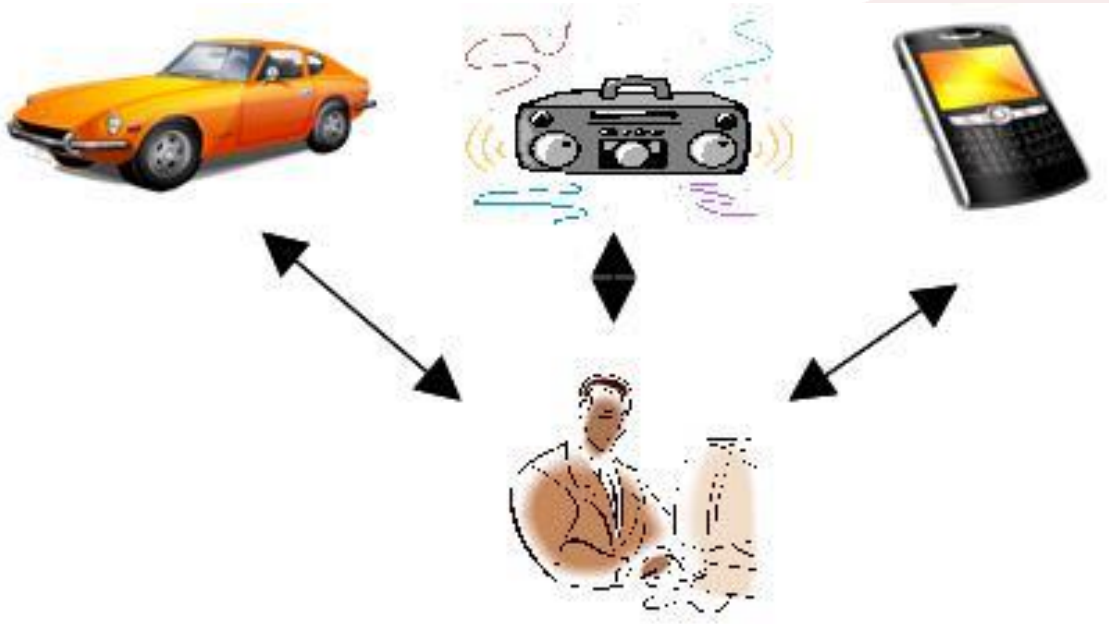
목차

목차

1 객체지향 프로그래밍이란?

- 객체지향 프로그래밍

  프로그램을 명령어들의 목록이 아니라, 객체들의 모임으로서, 객체와 객체와의 상호 관계를 중심으로 작성하자는 패러다임.



2 객체와 클래스

- **객체(object)**
  - **속성(Attribute)**과 **행위(Action)**를 가지고 있으며 이름을 붙일 수 있는 물체
    - 예: 자동차, 학생, 스마트 폰, 모니터, 키보드, 형광등, 책, 리모컨 등
- **클래스(class)**
  - 객체를 만들기 위한 설계도(blueprint)
- **인스턴스(instance):** 설계도에 따라 실제로 구현된 것
  - 인스턴스(instance)화 == 실체화 == 메모리에 구축한다
  - 클래스를 실체화 혹은 인스턴스화 시킨 것이 객체이다.

예로, class Programmer():

    pass

kim = Programmer()

kim이라는 객체는 Programmer라는 클래스의 인스턴스이다.

객체 생성예

```
student = {'name': '홍길동', 'year': 2, 'class': 3, 'student_id': 35}

print(student['name'], student['year'], student['class'], student['student_id'])
```

```
홍길동 2 3 35
```

```
class Student(object):
    def __init__(self, name, year, class_num, student_id):
        self.name = name
        self.year = year
        self.class_num = class_num
        self.student_id = student_id

    def introduce_myself(self):
        print(self.name, self.year, self.class_num, self.student_id)

student = Student('홍길동', 2, 3, 35)
student.introduce_myself()
```

```
홍길동 2 3 35
```

필요한 데이터에 접근하는 또 다른 방법

```
student = {'name': '홍길동', 'year': 2, 'class': 3, 'student_id': 35}

print(student['name'], student['year'], student['class'], student['student_id'])
```

```
홍길동 2 3 35
```

```
%%writefile stdu.py

name = '홍길동'
year = 2
class_num = 3
student_id = 35
```

```
Overwriting stdu.py
```

```
import stdu
print(stdu.name, stdu.year, stdu.class_num, stdu.student_id)
```

```
홍길동 2 3 35
```

Class

- 클래스 정의 문법

```
class 클래스_이름:
    클래스_본체
```

- Car 클래스의 속성과 행위

구분	코드	설명
클래스 이름	Car	자동차 클래스 이름
속성	_speed	속도
기능	get_speed	속도 가져오기
	start	출발
	accelerate	가속
	stop	정지

```
# car_example1.py

class Car:
    def __init__(self):
        self._speed = 0
        print("자동차가 생성되었습니다.")

    def get_speed(self):
        return self._speed

    def start(self):
        self._speed = 20
        print("자동차가 출발합니다.")

    def accelerate(self):
        self._speed = self._speed + 30
        print("자동차가 가속을 합니다.")

    def stop(self):
        self._speed = 0
        print("자동차가 정지합니다.")
```

```
if __name__ == "__main__":
    my_car = Car( )                                  # 1
    my_car.start()                                   # 2
    print("속도:", my_car.get_speed())               # 3
    my_car.accelerate()                              # 4
    print("속도:", my_car.get_speed())
    my_car.stop()                                    # 5
    print("속도:", my_car.get_speed())
```

#1: **Car 객체를 생성**, my_car라는 이름을 부여한다.
#2: my_car가 가진 start 메서드를 호출한다.
#3: get_speed 메서드를 호출하여 _speed 값을 반환 받아 화면에 출력
#4: accelerate 메서드를 호출하여 _speed값을 30만큼 증가
#5: stop 메서드를 호출하여 _speed 값을 0으로 설정

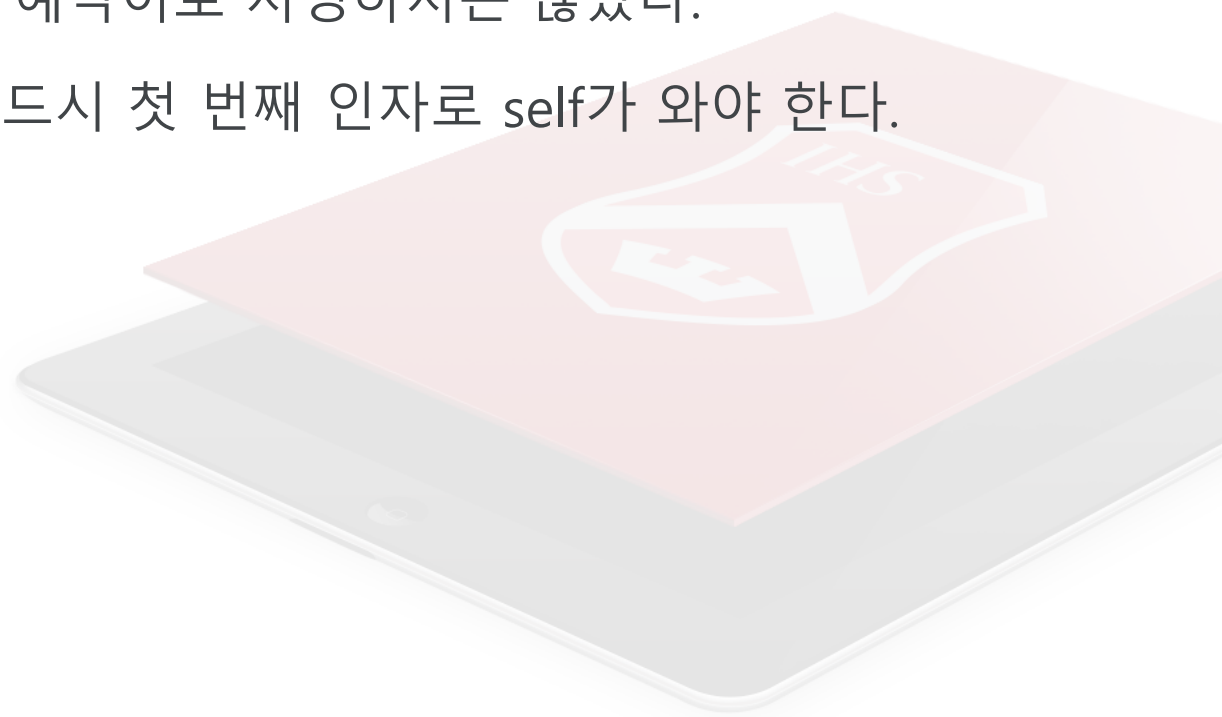
```
class Car:
    def __init__(self):
        self._speed = 0
        print("자동차가 생성되었습니다.")

    def get_speed(self):
        return self._speed

    def start(self):
        self._speed = 20
        print("자동차가 출발합니다.")

    def accelerate(self):
        self._speed = self._speed + 30
        print("자동차가 가속을 합니다.")

    def stop(self):
        self._speed = 0
        print("자동차가 정지합니다.")

if __name__ == "__main__":
    my_car = Car( )
    my_car.start()
    print("속도:", my_car.get_speed())
    my_car.accelerate()
    print("속도:", my_car.get_speed())
    my_car.stop()
    print("속도:", my_car.get_speed())
```

```
자동차가 생성되었습니다.
자동차가 출발합니다.
속도: 20
자동차가 가속을 합니다.
속도: 50
자동차가 정지합니다.
속도: 0
```

클래스명은 PEP 8 Coding Convention에 가이드된 대로 각 단어의 첫 문자를 대문자로 하는 **CapWords 방식**으로 명명한다.

3 self

- self는 객체 자신을 가리키는 특별한 변수이다.
- 현재 인스턴스 객체를 가리키는 것으로, C++나 자바의 this 키워드와 동일하지만, 예약어로 지정하지는 않았다.
- 객체 메서드에는 반드시 첫 번째 인자로 self가 와야 한다.



```
class Person:
    Name = "Default Name 홍길동"
    def Print(self):
        print("My Name is {0}".format(self.Name))
```

```
p1 = Person()
```

```
p1.Print()
```

```
My Name is Default Name 홍길동
```

```
p1.Name = "서경희"
```

```
p1.Print()    # bound method call
```

```
My Name is 서경희
```

```
Person.Print(p1)   # unbound method call
```

```
My Name is 서경희
```

Bound method call

```
class Foo:
    def func1():
        print("Function 1")
    def func2(self):
        print("function 2")
```

```
f=Foo()
f.func2()
```

```
function 2
```

```
f.func1()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-b3c056b64a8f> in <module>()
----> 1 f.func1()

TypeError: func1() takes 0 positional arguments but 1 was given
```

func1은 인자가 없는데 1개 받았다고 함. 즉, 첫 번째 인자로 항상 인스턴스가 전달되기 때문에 발생한 문제이다.

```
In [7]: class Foo:
            def func1():
                print("Function 1")
            def func2(self):
                print(id(self))
                print("function 2")
```

```
In [8]: f=Foo()
        id(f)
```

```
Out[8]: 2113307554592
```

```
In [9]: f.func2()
```

```
2113307554592
function 2
```

```
In [10]: f2=Foo()
         id(f2)
```

```
Out[10]: 2113307555712
```

```
In [11]: f2.func2()
```

```
2113307555712
function 2
```

f의 바인딩주소는 인스턴스의 주소값

```
In [12]: Foo.func1()

Function 1

In [13]: Foo.func2()

---------------------------------------------------------------------
----------
TypeError                                 Traceback (most recent
call last)
<ipython-input-13-1213fa831820> in <module>()
----> 1 Foo.func2()

TypeError: func2() missing 1 required positional argument: 'self'

In [14]: f3=Foo()
         id(f3)

Out[14]: 2113307556776

In [15]: Foo.func2(f3)

2113307556776
function 2

In [16]: f3.func2()

2113307556776
function 2
```

- 파이썬 클래스 자체가 하나의 네임스페이스이므로 인스턴스의 생성과 무관하게 클래스내의 메서드 호출가능(클래스와 인스턴스의 네임스페이스가 분리)
- Foo.func1: ㅇㅋ
- Foo.func2는 인자를 전달하지 않아 에러발생

- 인스턴스를 생성해서 그 인스턴스 f3 전달해줌

왜 OOP인가?

각각의 결과값을 유지해야 하는 2개의 계산기가 필요한 경우 함수 두 개로 해결했지만, 계산기가 5개, 10개로 **점점 더 많이 필요해진다면** 어떻게 해야 할 것인가?
그때마다 전역변수와 함수를 추가할 것인가?

In [1]:
```
result1 = 0
result2 = 0

def adder1(num):
    global result1
    result1 += num
    return result1

def adder2(num):
    global result2
    result2 += num
    return result2

print(adder1(3))
print(adder1(4))
print(adder2(3))
print(adder2(7))
```

```
3
7
3
10
```

[2]:
```
class Calculator:
    def __init__(self):
        self.result = 0

    def adder(self, num):
        self.result += num
        return self.result

cal1 = Calculator()
cal2 = Calculator()

print(cal1.adder(3))
print(cal1.adder(4))
print(cal2.adder(3))
print(cal2.adder(7))
```

```
3
7
3
10
```

계산기의 개수가 늘어나더라도 객체를 생성하기만 하면 되기 때문에 함수를 사용하는 경우와 달리 매우 간단해진다.

실습: 사칙연산하는 클래스 만들기

다음과 같이 사칙연산을 가능하게 하는 FourCal이라는 클래스를 만드시오.

```
In [4]: a=FourCal(4,2)
        b=FourCal(3,7)
        a.sum()
Out[4]: 6

In [3]: a.mul()
Out[3]: 8

In [4]: a.sub()
Out[4]: 2

In [5]: a.div()
Out[5]: 2.0

In [6]: b.sum()
Out[6]: 10

In [7]: b.mul()
Out[7]: 21

In [6]: b.sub()
Out[6]: -4

In [7]: b.div()
Out[7]: 0.4285714285714285
```

Class Members

- Class Members
  - **method**
  - **Property @**
  - **class variable**
  - **instance variable**
  - **Initializer __init__**
  - **Destructor __del__**

- 데이터를 표현하는 **field**와 행위를 표현하는 **method**로 구분하며, 파이썬에서는 이러한 field와 method 모두 그 객체의 attribute라고도 한다.
- 새로운 attribute를 동적으로 추가할 수 있으며, 메서드도 일종의 메서드 객체로 취급하여 attribute에 포함한다.

Initializer

- **__init__( )**
- 클래스로부터 새 객체가 생성될 때 마다 실행되는 특별한 메서드 (magic method)
- 인스턴스 변수의 초기화, 객체의 초기 상태를 만듦
- Python에서 클래스 생성자(Constructor)는 실제 런타임 엔진 내부에서 실행되는데, 이 생성자 실행 도중 클래스 안에 Initializer가 있는지 체크하여 만약 있으면 Initializer를 호출하여 객체의 변수 등을 초기화한다.

Destructor(소멸자)

- __del__
- 클래스로부터 객체가 소멸할 때 호출되는 특별한 메서드
- 객체의 reference counter(참조 카운터)가 0 이 되면 자동 호출
- 리소스 해제 등의 종료작업 수행

생성자와 소멸자의 예

```
class IceCream:
    def __init__(self, name, price):
        self.name = name
        self.price = price
        print(name + "의 가격은 "+ str(price) + "원 입니다.")
    def __del__(self):
        print(self.name + " 객체가 사라집니다")
```

```
obIce = IceCream("누가바", 800)
```

```
누가바의 가격은 800원 입니다.
```

```
ob3Ice = IceCream("월드콘", 1000)
```

```
월드콘의 가격은 1000원 입니다.
```

```
del ob3Ice
```

```
월드콘 객체가 사라집니다
```

소멸자의 예

```
class IceCream:
    def __init__(self, name, price):
        self.name = name
        self.price = price
        print(self.name + "의 가격은 "+ str(self.price) + "원 입니다.")
    def __del__(self):
        print(self.name + " 객체가 사라집니다")
```

```
obIce = IceCream("누가바", 800) #rc of obIce =1
```

누가바의 가격은 800원 입니다.

```
ob2Ice = IceCream("월드콘", 1000) #rc of ob2Ice =1
```

월드콘의 가격은 1000원 입니다.

```
obIce_copy = obIce  # rc of obIce =2
```

```
del obIce #rc of obIce =1
```

```
del obIce_copy #rc of obIce =0, 소멸자 호출
```

누가바 객체가 사라집니다

객체의 reference counter가 1 이상이면 del 구문을 사용해도 destructor가 호출되지 않는다

```
class Employee(object):
    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay
        self.email = first.lower()+'.'+last.lower()+'@sogang.ac.kr'

    def full_name(self):
        print(self.first + ' '+ self.last)

emp_1 = Employee('Jiho', 'Lee', 50000)
emp_2 = Employee('Minjung', 'Kim', 60000)
print(emp_1.email)
print(emp_2.email)
# emp_1의 풀네임 출력
emp_1.full_name()
```

```
jiho.lee@sogang.ac.kr
minjung.kim@sogang.ac.kr
Jiho Lee
```

```
class Employee(object):
    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay
        self.email = first.lower()+'.'+last.lower()+'@sogang.com'

    def __del__(self):
        print(self.last + " 퇴사했습니다")

    def full_name(self):
        print(self.first + ' '+ self.last)
```

```
emp_1 = Employee('Jiho', 'Lee', 50000)
emp_2 = Employee('Minjung', 'Kim', 60000)
print(emp_1.email)
print(emp_2.email)
emp_1.full_name()
```

```
jiho.lee@sogang.com
minjung.kim@sogang.com
Jiho Lee
```

```
del emp_1
```

```
Lee 퇴사했습니다
```

클래스 변수와 인스턴스 변수

클래스 변수가 하나의 클래스에 하나만 존재하는 반면, 인스턴스 변수는 각 객체 인스턴스마다 별도로 존재한다.

인스턴스 변수: 클래스 정의에서 메서드 안에서 사용되면서 "**self.변수명**"처럼 사용되며, 이는 **각 객체 별로 서로 다른 값을 갖는 변수이다**.

```
In [6]:
class Account:
    num_accounts = 0 #클래스 변수
    def __init__(self, name):
        self.name = name     #인스턴스 변수
        Account.num_accounts += 1
    def __del__(self):
        Account.num_accounts -= 1
```

```
In [8]:
kim = Account("kim")
kim.name
```

```
Out[8]: 'kim'
```

```
In [9]:
kim.num_accounts
```

```
Out[9]: 1
```

```
In [10]:
Account.num_accounts
```

```
Out[10]: 1
```

```
In [11]:
lee = Account("lee")
```

```
In [12]:
lee.name
```

```
Out[12]: 'lee'
```

```
In [13]:
lee.num_accounts
```

```
Out[13]: 2
```

```
In [14]:
Account.num_accounts
```

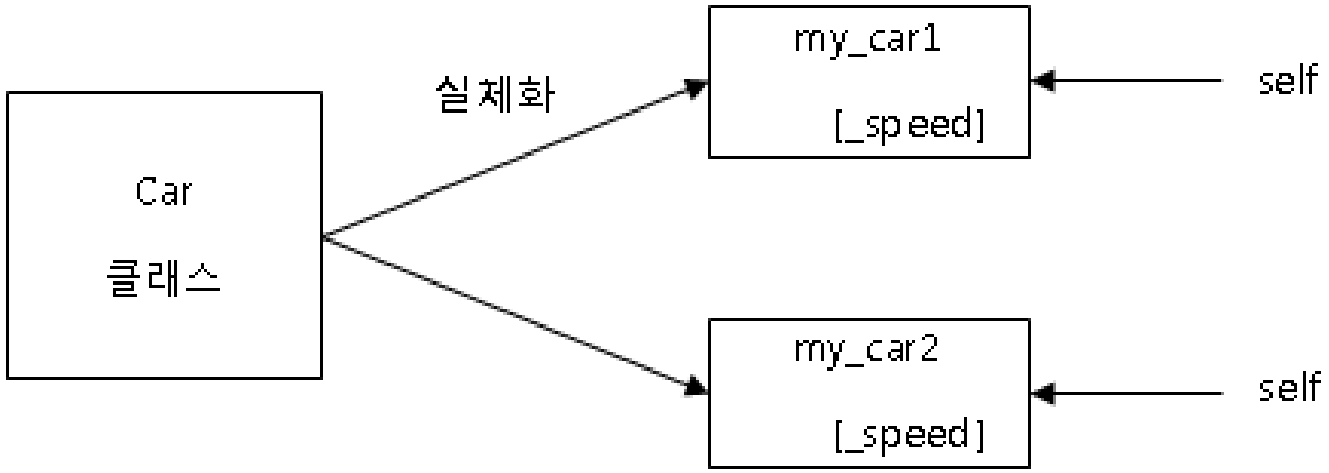
```
Out[14]: 2
```

```
# car_example2.py

class Car:
    def __init__(self):
        self._speed = 0

    def get_speed(self):
        return self._speed

    def start(self):
        self._speed = 20

    def accelerate(self):
        self._speed = self._speed + 30

    def stop(self):
        self._speed = 0
```

```
if __name__ == "__main__":
    my_car1 = Car( )
    my_car2 = Car( )
    my_car1.start( )
    my_car2.start( )
    my_car1.accelerate( )
    my_car2.accelerate( )
    my_car2.accelerate( )
    print("첫번째 자동차 속도:", my_car1.get_speed( ))
    print("두번째 자동차 속도:", my_car2.get_speed( ))
    my_car1.stop( )
    my_car2.stop( )
```

```
class Car:
    def __init__(self):
        self._speed = 0

    def get_speed(self):
        return self._speed

    def start(self):
        self._speed = 20

    def accelerate(self):
        self._speed = self._speed + 30

    def stop(self):
        self._speed = 0

if __name__ == "__main__":
    my_car1 = Car( )
    my_car2 = Car( )
    my_car1.start( )
    my_car2.start( )
    my_car1.accelerate( )
    my_car2.accelerate( )
    my_car2.accelerate( )
    print("첫번째 자동차 속도:", my_car1.get_speed( ))
    print("두번째 자동차 속도:", my_car2.get_speed( ))
    my_car1.stop( )
    my_car2.stop( )
```

```
첫번째 자동차 속도: 50
두번째 자동차 속도: 80
```



- **self를 이용해서 객체 자신이 가진 속성을 변화시키기 때문에 객체의 속성 값이 정확하게 관리된다.**

구분	코드	설명
클래스 이름	**Radio**	라디오 클래스 이름
속성		
기능	turn_on turn_off	라디오를 켠다 라디오를 끈다

구분	코드	설명
클래스 이름	**Car**	자동차 클래스 이름
속성	_speed _radio	속도 라디오
기능	get_speed start accelerate stop turn_on_radio turn_off_radio	속도 가져오기 출발 가속 정지 라디오를 켠다 라디오를 끈다

```
# car_with_radio.py

class Radio:
    def __init__(self):
        print("라디오가 생성되었습니다.")

    def turn_on(self):
        print("라디오를 켭니다.")

    def turn_off(self):
        print("라디오를 끕니다.")
```

```
class Car:
    def __init__(self):
        self._speed = 0
        print("자동차가 생성되었습니다.")
        self._radio = Radio( )

    def get_speed(self):
        return self._speed

    def start(self):
        self._speed = 20
        print("자동차가 출발합니다.")

    def accelerate(self):
        self._speed = self._speed + 30
        print("자동차가 가속을 합니다.")

    def stop(self):
        self._speed = 0
        print("자동차가 정지합니다.")

    def turn_on_radio(self):
        self._radio.turn_on( )

    def turn_off_radio(self):
        self._radio.turn_off( )
```

```
if __name__ == "__main__":
    my_car = Car( )
    my_car.start()
    my_car.turn_on_radio( ) # 1
    my_car.accelerate()
    my_car.turn_off_radio( ) # 2
    my_car.stop()
```

```python
class Radio:
    def __init__(self):
        print("라디오가 생성되었습니다.")

    def turn_on(self):
        print("라디오를 켭니다.")

    def turn_off(self):
        print("라디오를 끕니다.")

class Car:
    def __init__(self):
        self._speed = 0
        print("자동차가 생성되었습니다.")
        self._radio = Radio()

    def get_speed(self):
        return self._speed

    def start(self):
        self._speed = 20
        print("자동차가 출발합니다.")

    def accelerate(self):
        self._speed = self._speed + 30
        print("자동차가 가속을 합니다.")

    def stop(self):
        self._speed = 0
        print("자동차가 정지합니다.")

    def turn_on_radio(self):
        self._radio.turn_on( )

    def turn_off_radio(self):
        self._radio.turn_off( )
```

```python
if __name__ == "__main__":
    my_car = Car( )
    my_car.start()
    print("속도:", my_car.get_speed())
    my_car.turn_on_radio()
    my_car.accelerate()
    print("속도:", my_car.get_speed())
    my_car.turn_off_radio()
    my_car.stop()
    print("속도:", my_car.get_speed())
```

```
자동차가 생성되었습니다.
라디오가 생성되었습니다.
자동차가 출발합니다.
속도: 20
라디오를 켭니다.
자동차가 가속을 합니다.
속도: 50
라디오를 끕니다.
자동차가 정지합니다.
속도: 0
```