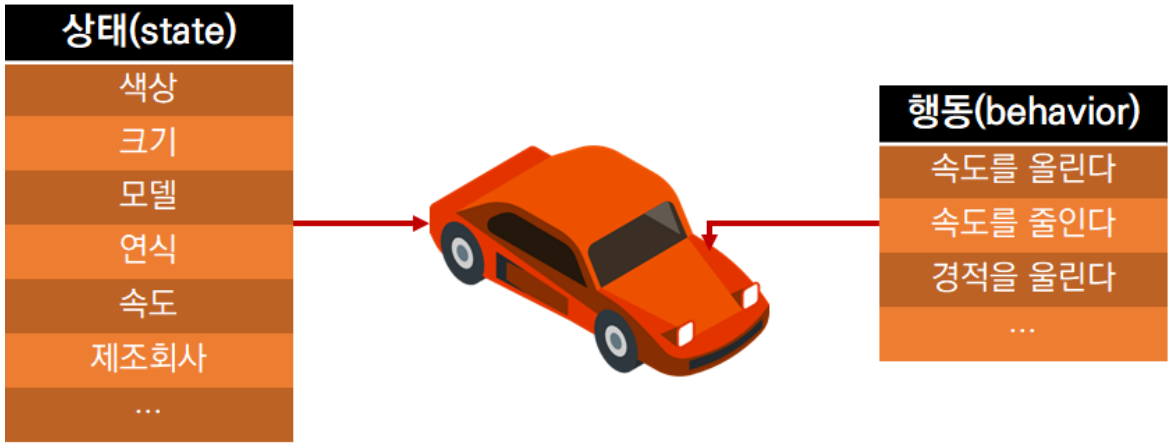
클래스(class)

- class는 물체의 설계도이다
- 설계도에는 물체의 속성(변수)과 물체의 동작(method)이 정의되어있다
- 설계도에 따라 만들어진 물체를 객체(object)라 한다
- 객체 = 속성(attribute) + 기능(method)
  - 객체는 상태(state)와 동작으로 나타냄
  - 객체의 상태는 객체의 속성이고, 객체의 동작(behavior)은 객체가 취할 수 있는 기능
  - 예를 들어 자동차에는 제조사, 모델, 년식, 색상, 속도 등의 속성이 있고, 가속, 감속, 좌회전, 우회전 등 자동차가 할 수 있는 기능이 있다.



클래스 정의

```
class 클래스이름(부모 클래스) : ◀ class 키워드로 정의
    def __init__(self, parameters):
        #인스턴스 변수 정의
        self.var1 = parameter1
        self.var2 = parameter2

    def method(self, parameters):
        body
```

클래스 정의

- 부모 클래스는 생략할 수 있다

```
class 클래스이름:
```

- *_init_*() 메서드에서 인스턴스 변수를 정의하고 자동 실행할 부분을 프로그램한다
  - *_init_*() 메서드는 객체를 생성할 때 자동 실행된다
- *_init_*() 이외의 method는 일반 프로그램의 함수와 비슷하게 프로그램한다
  - 모든 method의 첫 번째 매개변수는 **self**이다
- class 내부에서 다른 method를 호출하려면 **self.method()** 로 호출한다.
  이 때 self 매개변수는 생략한다

```
class Sample:
    def method1(self, par1, par2):
        pass

    def method2(self, par3):
        self.method1(par1, par2)
```

Car 클래스

```
class Car :
    # 초기화 메소드. self는 객체를 나타냄
    def __init__(self, color, speed):
        # 인스턴스 변수 정의
        self.color = color
        self.speed = speed

    #메소드의 첫 번째 매개변수는 self
    def speedUp(self, v):
        self.speed = self.speed + v
        return self.speed

    def speedDown(self, v):
        self.speed = self.speed - v
        return self.speed
```

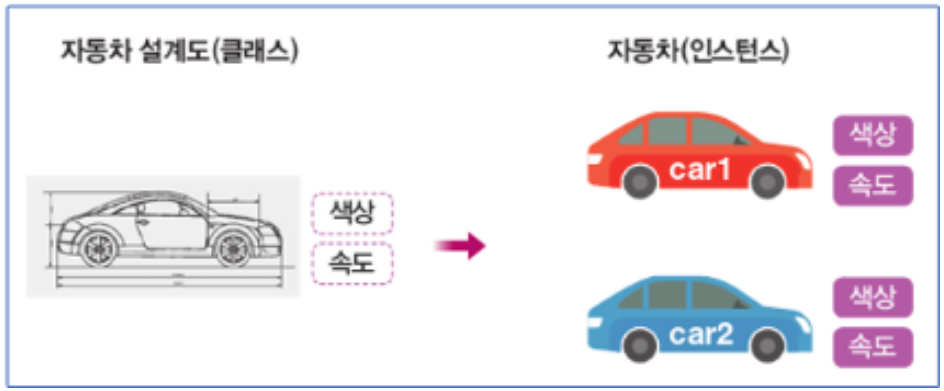
Car 객체

- 클래스를 인스턴스화하여 만들어진 것을 객체라 한다.
  - object = 클래스이름()
- 객체는 클래스에 정의된 변수(속성)와 메서드에 접근할 수 있다.
  - object.인스턴스변수
  - object.method()

```
# 객체 생성
# 인수는 __init__() 메서드의 매개변수와 일치
Mycar = Car("Red", 100)
# 객체 속성 접근
print(Mycar.color)
print(Mycar.speed)

#객체 메소드 호출
Mycar.speedUp(10)
print(Mycar.speed)

Mycar.speedDown(20)
print(Mycar.speed)
```

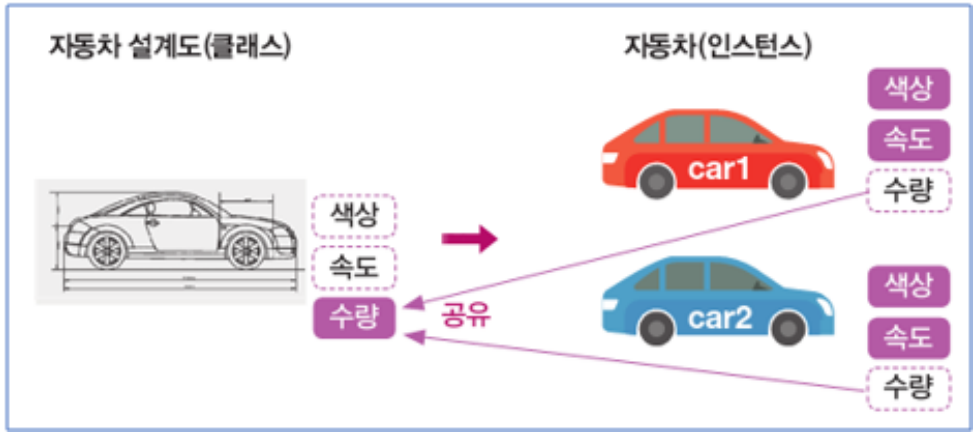
클래스 변수와 인스턴스 변수

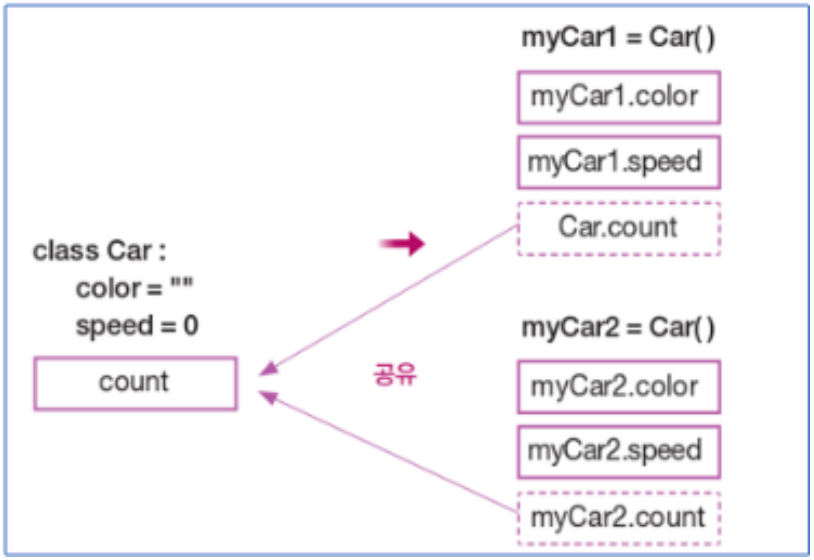
- 인스턴스 변수
  - self.var로 선언된 변수
  - 클래스 내부의 모든 메서드에서 사용 가능
  - 인스턴스를 생성해야 비로소 사용할 수 있는 변수이다.



클래스 변수와 인스턴스 변수

- 클래스 변수
  - 클래스 내부, 메서드 밖에서 정의된 변수
  - 모든 객체에서 같은 값을 갖는다
  - **클래스이름.var**로 사용하면 클래스 변수를 지정하고, **객체.var**로 사용하면 인스턴스 변수가 됨





참고: https://velog.io/asda²asda

클래스 변수와 인스턴스 변수

```
class Shark:
    # 클래스 변수
    animal_type = "fish"
    location = "ocean"

    # 생성자 함수(__init__)에서 인스턴스 변수 정의
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # 메서드 정의
    def set_followers(self, followers):
        print(self.name + " has " + str(followers) + " followers")
```

클래스 변수와 인스턴스 변수

```
# 클래스 객체 1
clowny = Shark("Clowny", 5)

# 인스턴스 변수 접근
print(clowny.name)

# 클래스 변수 접근
# 초기값은 클래스 변수 초기값
print(clowny.location)
print(clowny.animal_type)

# 인스턴스 변수를 변경해도 클래스 변수는 변하지 않는다
clowny.location = "East Sea" # 인스턴스 변수 변경
print(clowny.location)
print(Shark.location)
```

클래스 변수와 인스턴스 변수

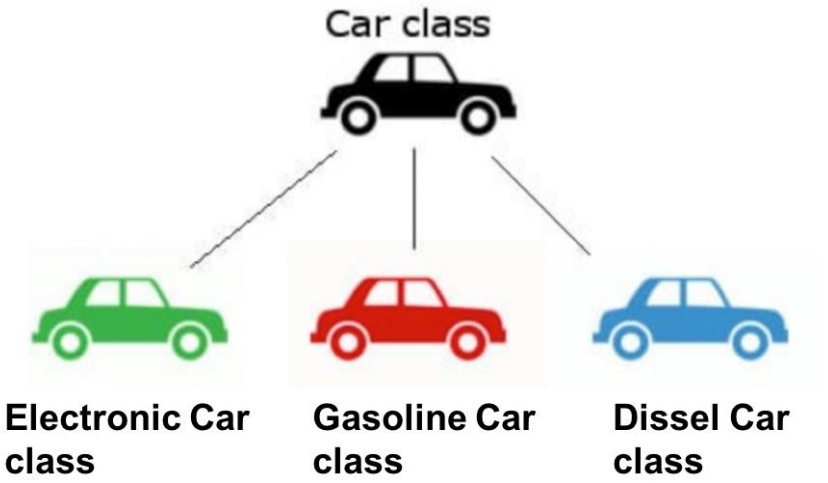
```
# 클래스 객체 2
stevie = Shark("Stevie", 8)

# 인스턴스 변수 접근
print(stevie.name)

# 메서드 호출
stevie.set_followers(77)

# 인스턴스 변수 접근
print(f'stevie가 사는 지역 {stevie.location}')
print(stevie.animal_type)
print(f'clowny가 사는 지역 {clowny.location}')
```

클래스 상속(inheritance)

- 부모 클래스를 상속 받은 자식 클래스는 부모 클래스의 속성과 메서드를 모두 물려받는다.

형식: class Child(Car):

- 부모클래스 : 상속하는 클래스
- 자식클래스 : 상속받는 클래스
- 자식 클래스가 부모 클래스의 내용을 가져다 쓸 수 있는 것



클래스 상속(inheritance)

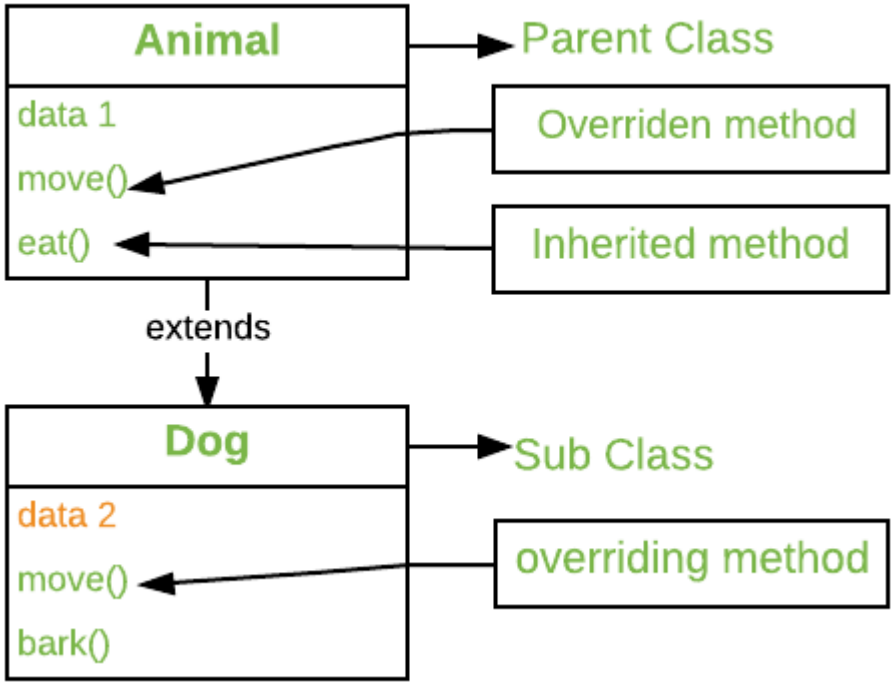
```
#부모 클래스
class People :
    def __init__(self, age=0, name=None):
        self.age = age
        self.name = name

    def introMe(self):
        print("My name is ", self.name, "and I am ", \
              str(self.age),"years old")
#자식 클래스
class Teacher(People) :
    def __init__(self, age=0, name=None, school=None) :
        # 부모 클래스의 초기화 함수는 자동 호출되지 않으므로 따로 호출해야 함.
        # 부모클래스 메서드 기능을 확장할 때 사용
        super().__init__(age, name) # super().method()
        self.school = school # 자식 클래스의 인스턴스 변수 추가

    #자식 클래스의 메서드
    def showSchool(self):
        print("My School is ", self.school)
```

클래스 상속(inheritance)

```
#부모 클래스 객체
p1 = People(29, "Lee")
p1.introMe()

#자식 클래스 객체
t1 = Teacher(22, "Kim", "동양미래대학교")
print("나이: ",t1.age)
print("이름: ",t1.name)
print("대학: ",t1.school)
t1.introMe() # 부모 클래스 메서드
t1.showSchool() # 자식 클래스 메서드
```

메서드 오버라이딩(method overriding)

- 자식 클래스에서 부모 클래스의 메서드를 다시 정의하는 것을 오버라이딩이라 한다.
- 하위 클래스의 메서드가 상위 클래스의 메서드와 이름, 매개변수 또는 시그니처, 반환 유형(또는 하위 유형)이 같을 때, 하위 클래스의 메서드가 상위 클래스의 메서드를 **오버라이드 한다고 함.**



참고: https://www.geeksforgeeks.org/

메서드 오버라이딩

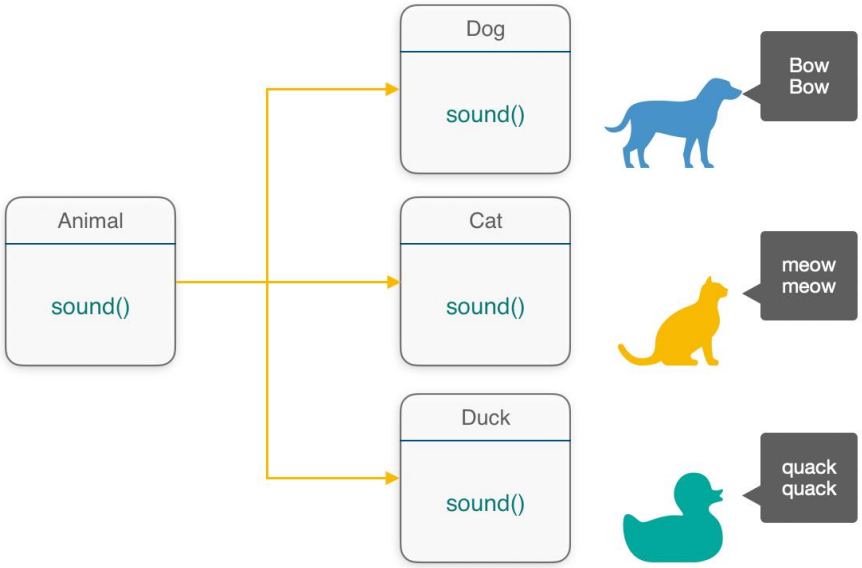
```
#부모 클래스
class People :
    def __init__(self, age=0, name=None):
        self.age = age
        self.name = name

    def introMe(self):
        print("My name is ", self.name, "and I am ", \
              str(self.age),"years old")

#자식 클래스
class Student(People) :
    def __init__(self, age=0, name=None, grade=None):
        super().__init__(age, name)
        self.grade = grade

    #부모 클래스 메소드를 재정의(오버라이딩)
    def introMe(self):
        super().introMe() # 부모 클래스의 메서드 호출
        print("My grade is ", self.grade)
```

메서드 오버라이딩

```
p1 = People(21, "Lee")
p1.introMe() # People의 introMe() 호출

s1 = Student(17, "Park", 2)
s1.introMe() # Student의 introMe() 호출
```

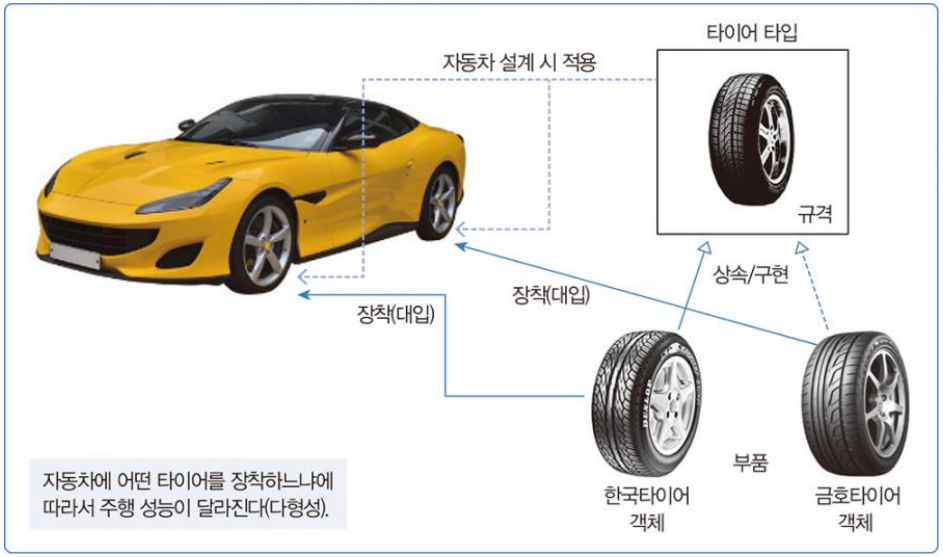
다형성(polymorphism)

- 다형성은 대입되는 **객체(인자)에 따라 메서드를 다르게 동작**하도록 구현하는 기술
  - 이는 실행도중 동일한 이름의 메서드 호출(오버라이드)에도 각각의 메서드를 선택할 수 있다.
  - 객체들의 타입(클래스)이 다르면 동일 메시지가 전달되어도 다른 동작을 하게 할 수 있음
- Animal이라는 부모객체에서 동물의 공통적인 ' 울음소리 ' 를 나타내는 sound()라는 메소드를 정의하였다고 가정한다.
  - 동물클래스인 Dog, Cat, Duck은 각각 Animal 클래스를 상속받아 sound()를 각각의 특성에 맞게 재정의 실행
    - Dog의 sound() : Bow Bow, Cat의 sound() : meow meow , Duck의 sound() : quack quack
  - Sound()라는 동일한 메소드를 전달하지만 각각의 클래스는 다르게 반응



다형성(polymorphism)

```
#다른 클래스에서 동일한 이름을 갖는 메서드를 정의한다
class Korean(object):
    def greeting(self):
        print("안녕하세요")

class American(object):
    def greeting(self):
        print("Hello")

def sayhello(people):
    people.greeting()
```



출처: 이것이 자바다(3판)

```
Kim = Korean() # Korean 클래스 객체
John = American() # Americal 클래스 객체

sayhello(Kim) # 동일한 함수인데 인수에 따라 출력이 다름
sayhello(John) # John을 전달하면 "Hello"
```