산업 인공지능 - 실습 7

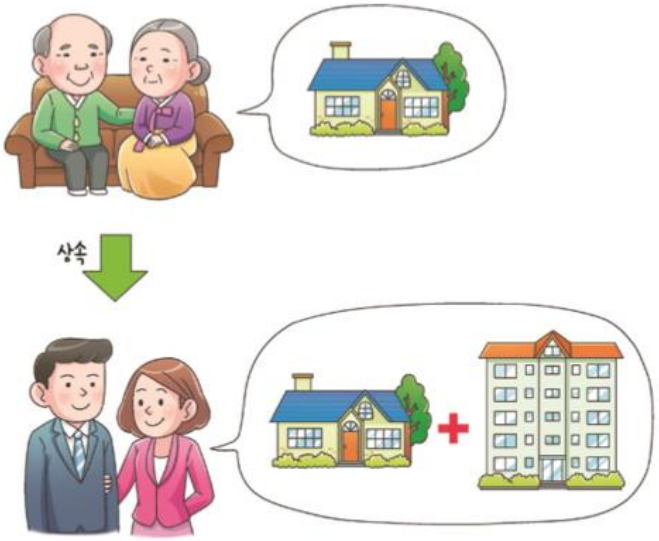
Python 프로그래밍

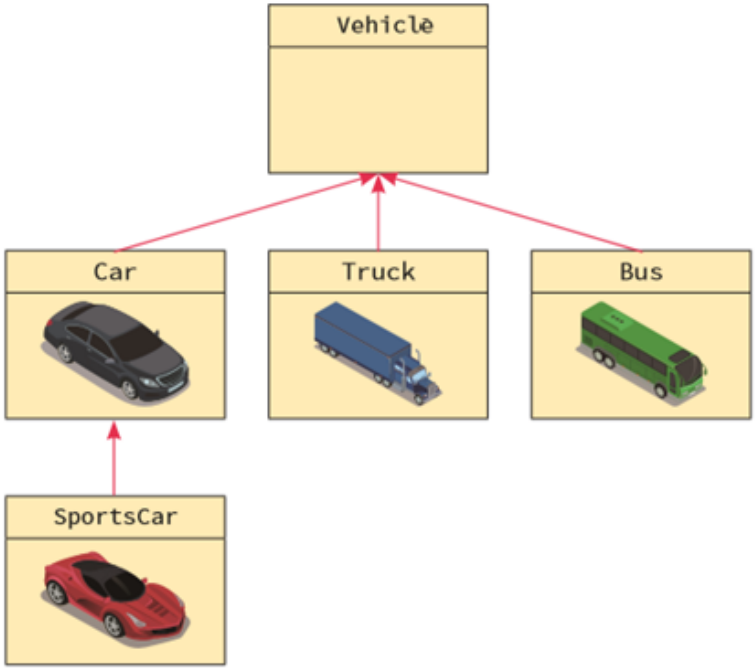
2021 Spring

1. 상속

❖ **상속(inheritance)**

- **기존의 클래스**로 부터 인스턴스 변수와 메소드를 상속받아 **새로운 클래스**를 만드는 메커니즘
- 필요시 **새로운 인스턴스 변수**와 **메소드 추가 및 교체** 가능





상속

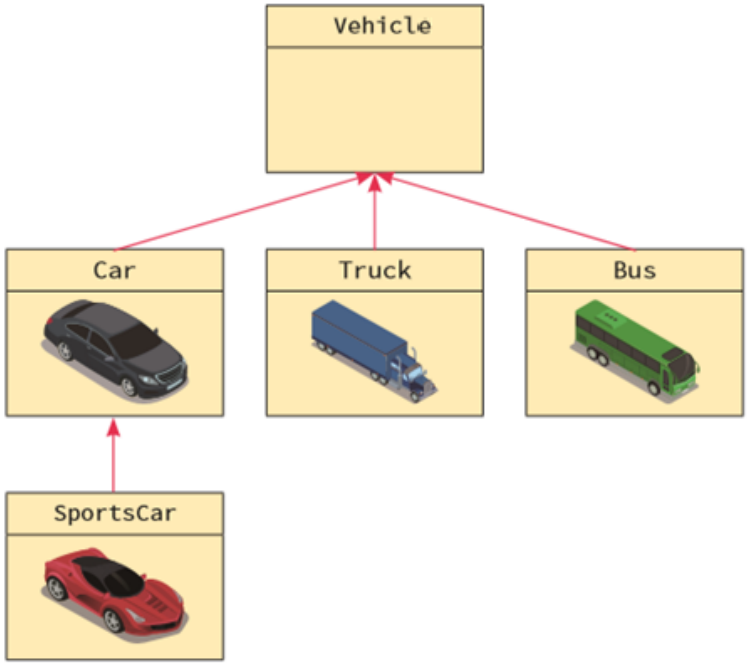
❖ 상속과 is-a 관계

- 클래스간의 is-a 관계 생성

- **부모 클래스(parent class, super class, base class)**
  - 기존 클래스로 상속되는 것

- **자식 클래스(child class, sub class, derived class)**
  - 상속을 통해 새로 만들어지는 클래스

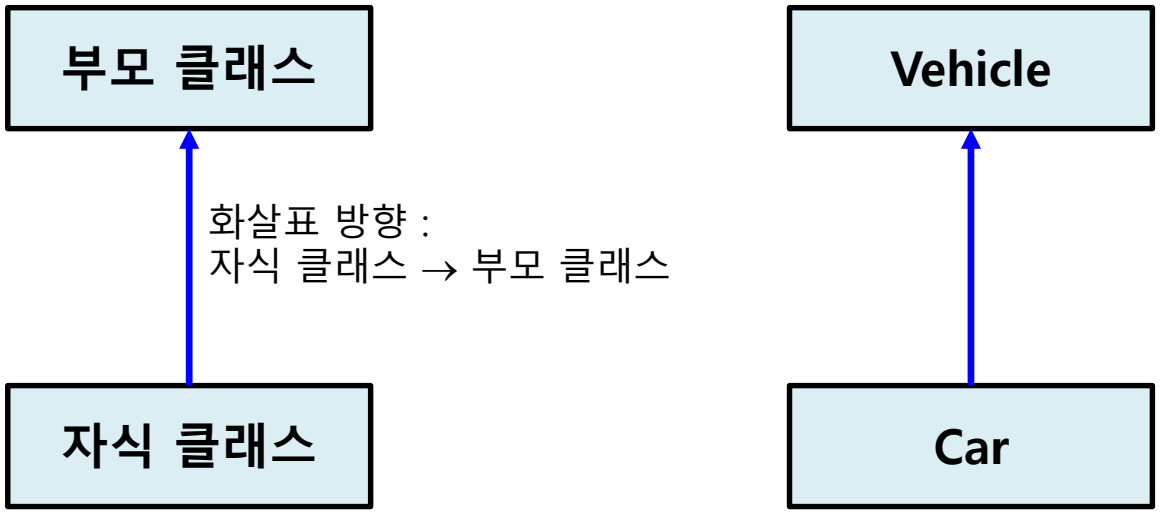
Car is a Vehicle
SportsCar is a Car
Truck is a Vehicle
Bus is a Vehicle



2. 클래스 상속의 구현

❖ **클래스 상속 정의**

- `class` **자식클래스** (**부모클래스**) :
  생성자
  메소드

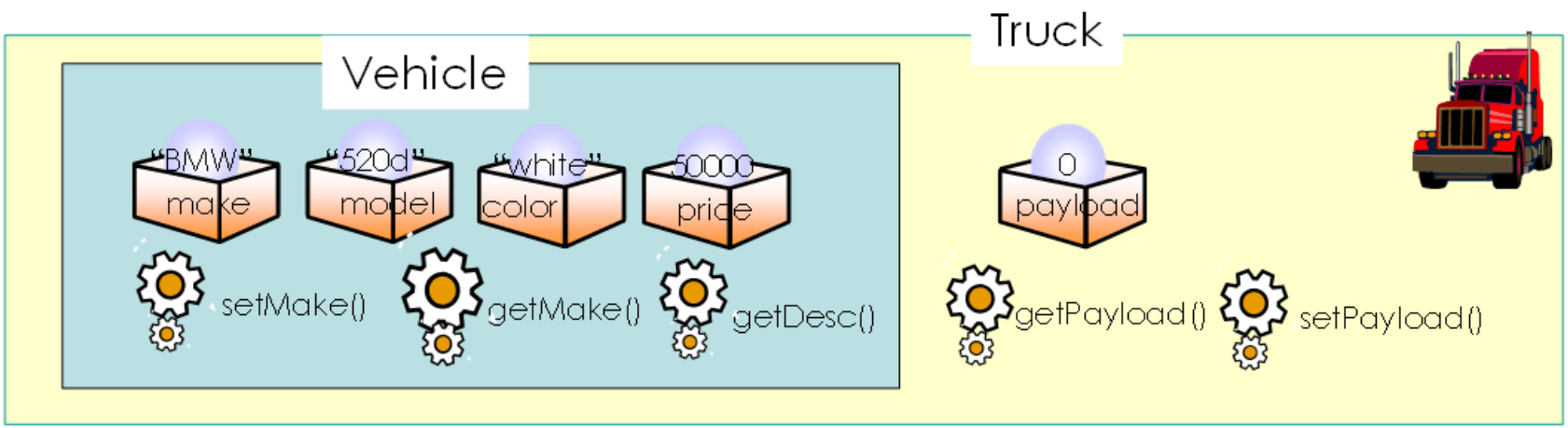


상속의 구현

```
# 일반적인 운송수단을 나타내는 클래스 .
class Vehicle:
    def __init__(self, make, model, color,  price):
        self.make = make            # 메이커
        self.model = model          # 모델
        self.color = color          # 자동차의 색상
        self.price = price          # 자동차의 가격

    def setMake(self, make):
        self.make = make

    def getMake(self):
        return self.make

    def getDesc(self):
        return "차량 =("+str(self.make)+","+\
                        str(self.model)+","+\
                        str(self.color)+","+\
                        str(self.price)+")"
```

상속의 구현

```
class Truck(Vehicle) :
    def __init__(self, make, model, color, price, payload):
        super().__init__(make, model, color, price)
        self.payload = payload

    def setPayload(self, payload):
        self.payload = payload

    def getPayload(self):
        return self.payload
```



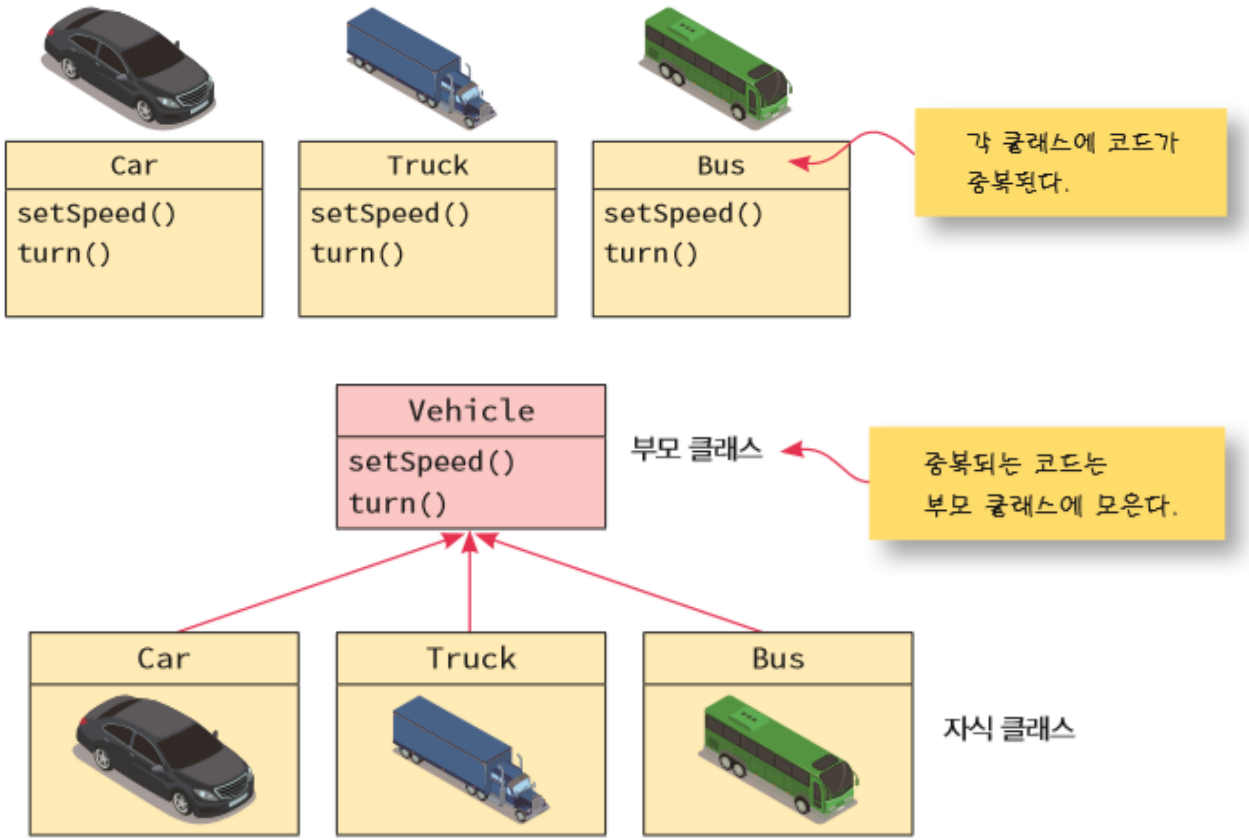
상속의 구현

```
def main():
    myTruck = Truck("Tisla", "Model S", "white", 10000, 2000)
    myTruck.setMake("Tesla")
    myTruck.setPayload(2000)
    print(myTruck.getDesc())

main()
```

```
차량 =(Tesla,Model S,white,10000)
```

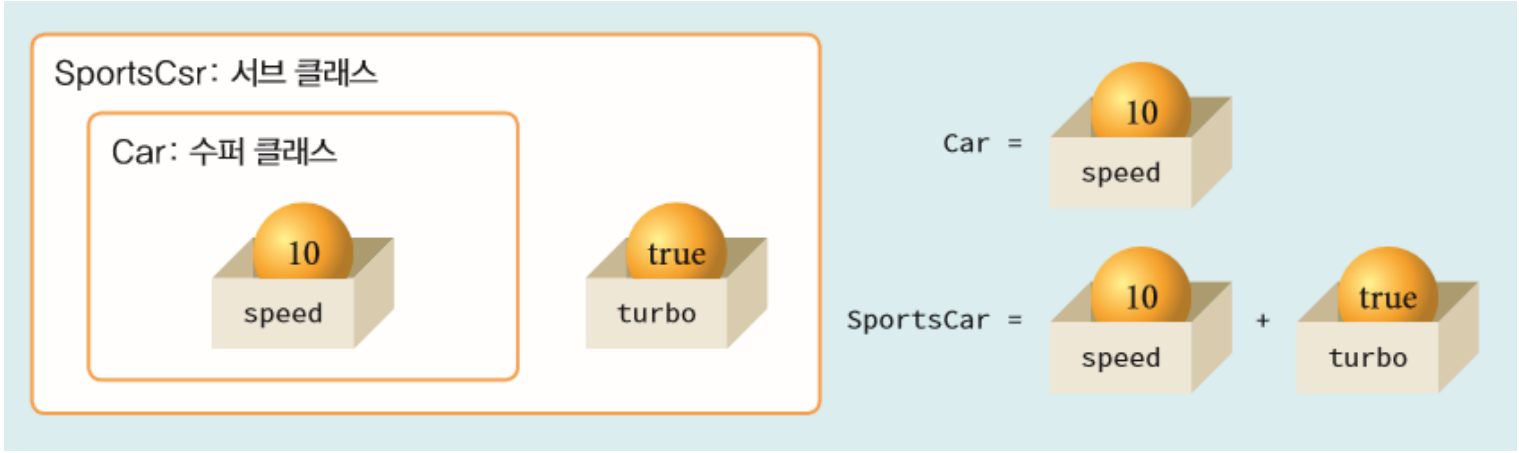
3. 상속 사용의 이유

❖ 상속 사용의 이유
- 기존 코드의 재사용
- 중복된 코드 축소



예: Sportscar 클래스

- 부모 클래스 **Car**
- 자식 클래스 **SportsCar**



예: Sportscar 클래스

```
class Car :
    def __init__(self, speed):
        self.speed = speed
    def setSpeed(self, speed):
        self.speed = speed
    def getDesc(self):
        return "차량 =("+str(self.speed) + ")"

class SportsCar(Car) :
    def __init__(self, speed, turbo):
        super().__init__(speed)
        self.turbo = turbo

    def setTurbo(self, turbo):
        self.turbo = turbo

obj = SportsCar(100, True)
print(obj.getDesc())
obj.setTurbo(False)
```

차량 =(100)

4. 부모 클래스 생성자 호출

❖ **부모 클래스 생성자 호출**

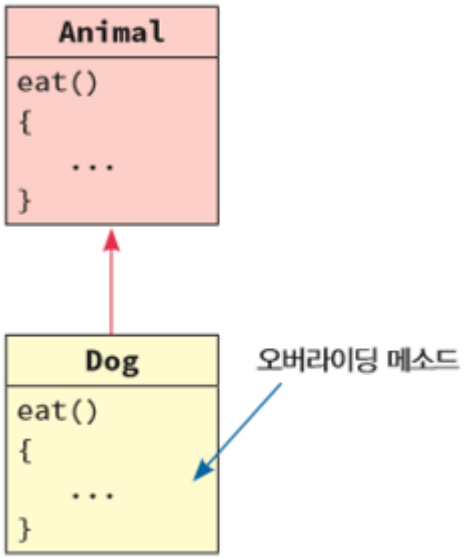
- 자식 클래스에서 명시적으로 호출

```
class ChildClass(ParentClass) :
    def __init__(self):
        super().__init__()
        ...
```

5. 메소드 오버라이딩

❖ **메소드 오버라이딩**(method overriding)

- 자식 클래스에서 **부모 클래스의 메소드**를 **재정의(redefine)**하는 것



메소드 오버라이딩

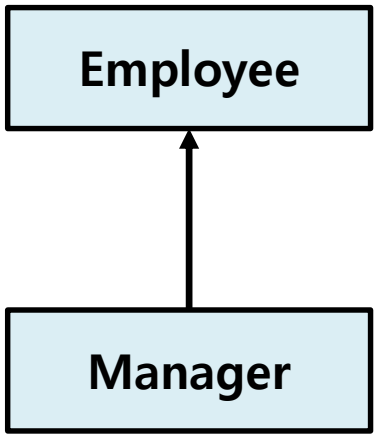
```
class Animal:
    def __init__(self, name=""):
        self.name=name
    def eat(self):
        print("동물이 먹고 있습니다. ")

class Dog(Animal):
    def __init__(self):
            super().__init__()
    def eat(self):
        print("강아지가 먹고 있습니다. ")

d = Dog();
d.eat()
```

```
강아지가 먹고 있습니다.
```

예: 직원과 매니저 클래스

❖ **직원과 매니저 클래스**

- 직원(**Employee**)은 월급만 있지만 매니저(**Manager**)는 월급외에 보너스가 있다.
- Employee 클래스를 상속받아서 Manager 클래스를 작성한다.
- Employee 클래스의 getSalary()는 Manager 클래스에서 재정의된다.



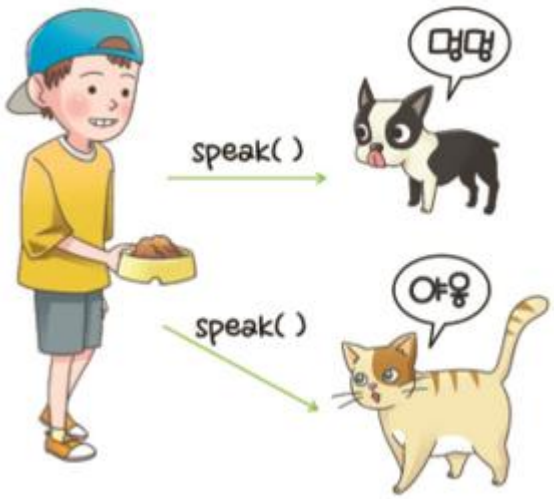
```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def  getSalary(self):
        return salary

class Manager(Employee):
    def __init__(self, name, salary, bonus):
        super().__init__(name, salary)
        self.bonus = bonus

    def  getSalary(self):
        salary = super().getSalary()
        return salary + self.bonus

    def __repr__(self):
        return "이름: " + self.name + "; 월급: " + str(self.salary) + \
               "; 보너스: " + str(self.bonus)

kim = Manager("김철수", 2000000, 1000000)
print(kim)
```

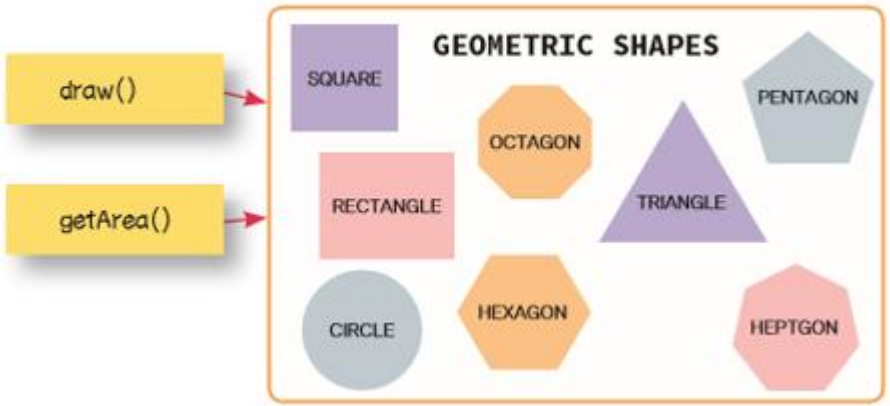
```
이름: 김철수; 월급: 2000000; 보너스: 1000000
```

6. 다형성

❖ **다형성(polymorphism)**

- **"많은(poly)+모양(morph)"**
- 하나의 식별자로 다양한 타입(클래스)을 처리하는 것





다형성

❖ 내장 함수와 다형성

```
>>> list = [1, 2, 3]          # 리스트
>>> len(list)
3

>>> s = "This is a sentense"    # 문자열
>>> len(s)
18

>>> d = {'aaa': 1, 'bbb': 2}    # 딕셔너리
>>> len(d)
2
```

다형성

❖ 상속과 다형성

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return '알 수 없음'

class Dog(Animal):
    def speak(self):
        return '멍멍!'

class Cat(Animal):
    def speak(self):
        return '야옹!'

animalList = [Dog('dog1'),
              Dog('dog2'),
              Cat('cat1')]

for a in animalList:
    print (a.name + ': ' + a.speak())
```

예: 추상메소드

❖ **예: Vehicle, Car, Truck 클래스**
- 일반적인 운송수단을 나타내는 **Vehicle** 클래스 상속
- **Car** 클래스와 **Truck** 클래스 작성

```
class Vehicle:
    def __init__(self, name):
        self.name = name

    def drive(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")

    def stop(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")
```

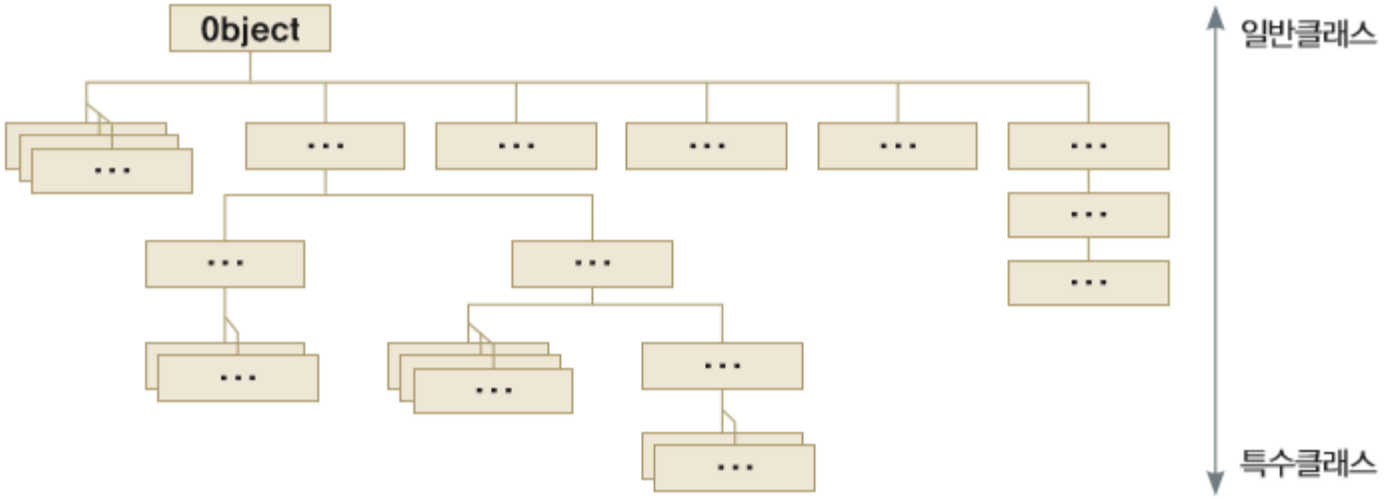
```
truck1: 트럭을 운전합니다.
truck2: 트럭을 운전합니다.
car1: 승용자를 운전합니다.
```

```
class Car(Vehicle):
    def drive(self):
        return '승용자를 운전합니다. '

    def stop(self):
        return '승용자를 정지합니다. '


class Truck(Vehicle):
    def drive(self):
        return '트럭을 운전합니다. '

    def stop(self):
        return '트럭을 정지합니다. '

cars = [Truck('truck1'), Truck('truck2'),  Car('car1')]

for car in cars:
    print(car.name + ': ' + car.drive())
```

7. Object 클래스

❖ **Object 클래스**

- 최상위의 클래스
- **모든 클래스가 상속받는 클래스**



Object 클래스

❖ **Object 클래스의 메소드**

메소드	메소드
`__init__ ( self [,args...] )` 생성자 (예) `obj = className(args)`	`__str__( self )` 문자열 표현 반환 (예) `str(obj)`
`__del__( self )` 소멸자 (예) `del obj`	`__cmp__ ( self, x )` 객체 비교 (예) `cmp(obj, x)`
`__repr__( self )` 객체 표현 문자열 반환 (예) `repr(obj)`	

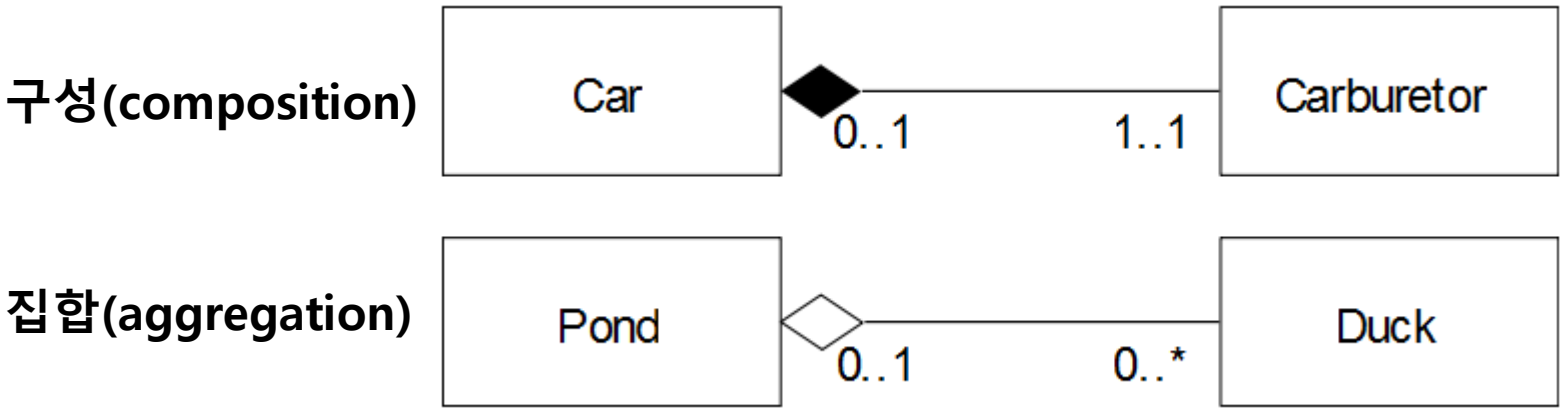
Object 클래스

```
class Book:
    def __init__(self, title, isbn):
        self.__title = title
        self.__isbn = isbn

    def __repr__(self):
        return "ISBN: " + self.__isbn + "; TITLE: " + self.__title

book = Book("The Python Tutorial", "0123456")
print(book)
```

```
ISBN: 0123456; TITLE: The Python Tutorial
```

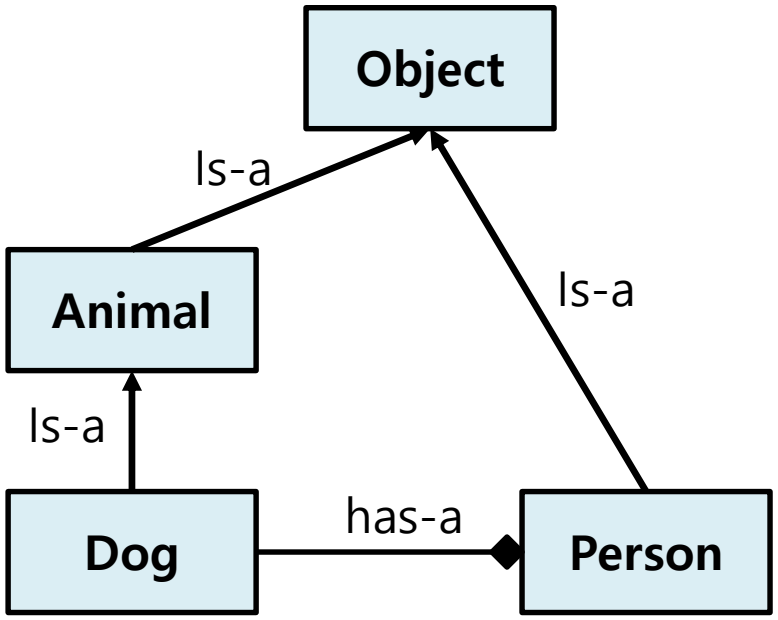
8. 클래스 관계

❖ 클래스 관계
- is-a 관계
  - 상속
- has-a 관계



클래스 관계

```
class Animal(Object):
    pass

class Dog(Animal):
    def __init__(self, name):
        self.name = name

class Person(Object):
    def __init__(self, name):
        self.name = name
        self.pet = None


dog1 = Dog("dog1")
person1 = Person("홍길동")
person1.pet = dog1
```



기계학습 – 신경망

[실습] sklearn의 MLP

```
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,confusion_matrix

wine = load_wine()
data = pd.DataFrame(data=wine['data'], columns=wine['feature_names'])
print(data.head())

X = wine.data
y = wine.target
X_train, X_test, y_train, y_test = train_test_split(X, y)

scaler = StandardScaler()
scaler.fit(X_train)
StandardScaler(copy=True, with_mean=True, with_std=True)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)
mlp.fit(X_train,y_train)
predictions = mlp.predict(X_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

Classes	3
Samples per class	[59,71,48]
Samples total	178
Dimensionality	13
Features	real, positive

```
   alcohol  malic_acid   ash  ...   hue  od280/od315_of_diluted_wines  proline
0    14.23        1.71  2.43  ...  1.04                          3.92   1065.0
1    13.20        1.78  2.14  ...  1.05                          3.40   1050.0
2    13.16        2.36  2.67  ...  1.03                          3.17   1185.0
3    14.37        1.95  2.50  ...  0.86                          3.45   1480.0
4    13.24        2.59  2.87  ...  1.04                          2.93    735.0

[5 rows x 13 columns]
```

```
[[11  0  0]
 [ 1 15  1]
 [ 0  0 17]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        11
           1       1.00      0.88      0.94        17
           2       0.94      1.00      0.97        17

    accuracy                           0.96        45
   macro avg       0.95      0.96      0.96        45
weighted avg       0.96      0.96      0.95        45
```

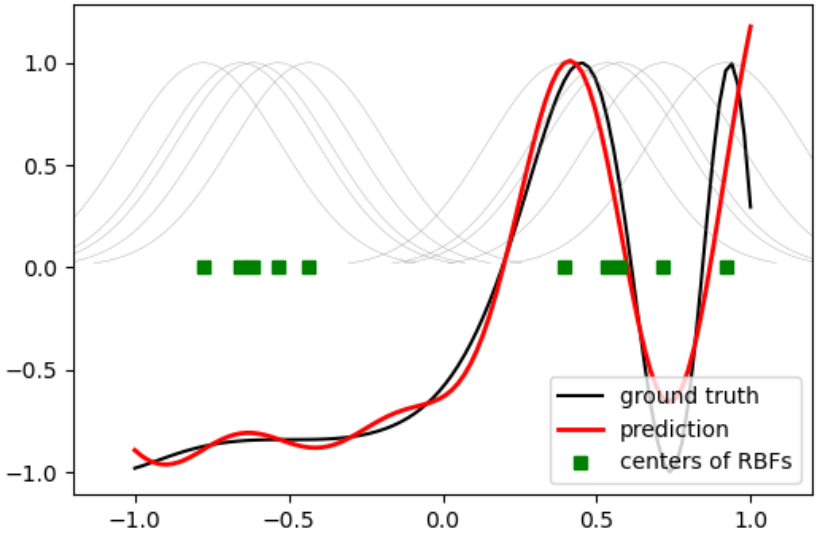
https://scikit-learn.org/

[실습] RBF망 학습

```
from scipy import *
from scipy.linalg import norm, pinv
from matplotlib import pyplot as plt
import numpy as np

class RBF:
    def __init__(self, indim, numCenters, outdim):
        self.indim = indim;   self.outdim = outdim;   self.numCenters = numCenters
        self.centers = [random.uniform(-1, 1, indim) for i in range(numCenters)]
        self.beta = 8
        self.W = random.random((self.numCenters, self.outdim))

    def basisFunc(self, c, d):
        assert len(d) == self.indim
        return np.exp(-self.beta * norm(c-d)**2)

    def activationFunc(self, X):
        G = np.zeros((X.shape[0], self.numCenters), float)
        for ci, c in enumerate(self.centers):
            for xi, x in enumerate(X):
                G[xi,ci] = self.basisFunc(c, x)
        return G

    def train(self, X, Y):
        rnd_idx = random.permutation(X.shape[0])[:self.numCenters]
        self.centers = [X[i,:] for i in rnd_idx]
        G = self.activationFunc(X)
        self.W = np.dot(pinv(G), Y)

    def predict(self, X):
        G = self.activationFunc(X)
        Y = np.dot(G, self.W)
        return Y
```

https://scikit-learn.org/

```
n = 100
x = mgrid[-1:1:complex(0,n)].reshape(n, 1)
y = np.sin(3*(x+0.5)**3 - 1)

rbf = RBF(1, 10, 1)
rbf.train(x, y)
z = rbf.predict(x)

plt.figure(figsize=(6, 4))
plt.plot(x, y, 'k-', label='ground truth')
plt.plot(x, z, 'r-', linewidth=2, label='prediction')
plt.plot(rbf.centers, np.zeros(rbf.numCenters), 'gs', label='centers of RBFs')

for c in rbf.centers:
    cx = np.arange(c-0.7, c+0.7, 0.01)
    cy = [rbf.basisFunc(np.array([cx_]), np.array([c])) for cx_ in cx]
    plt.plot(cx, cy, '-', color='gray', linewidth=0.2)

plt.xlim(-1.2, 1.2)
plt.legend( )
plt.show( )
```



https://scikit-learn.org/