

# Chapter 17: Object-Oriented Paradigm

- History of Programming Languages
- Abstract Data Type
- Python OOP Tutorial
- Motivation behind Python OOP

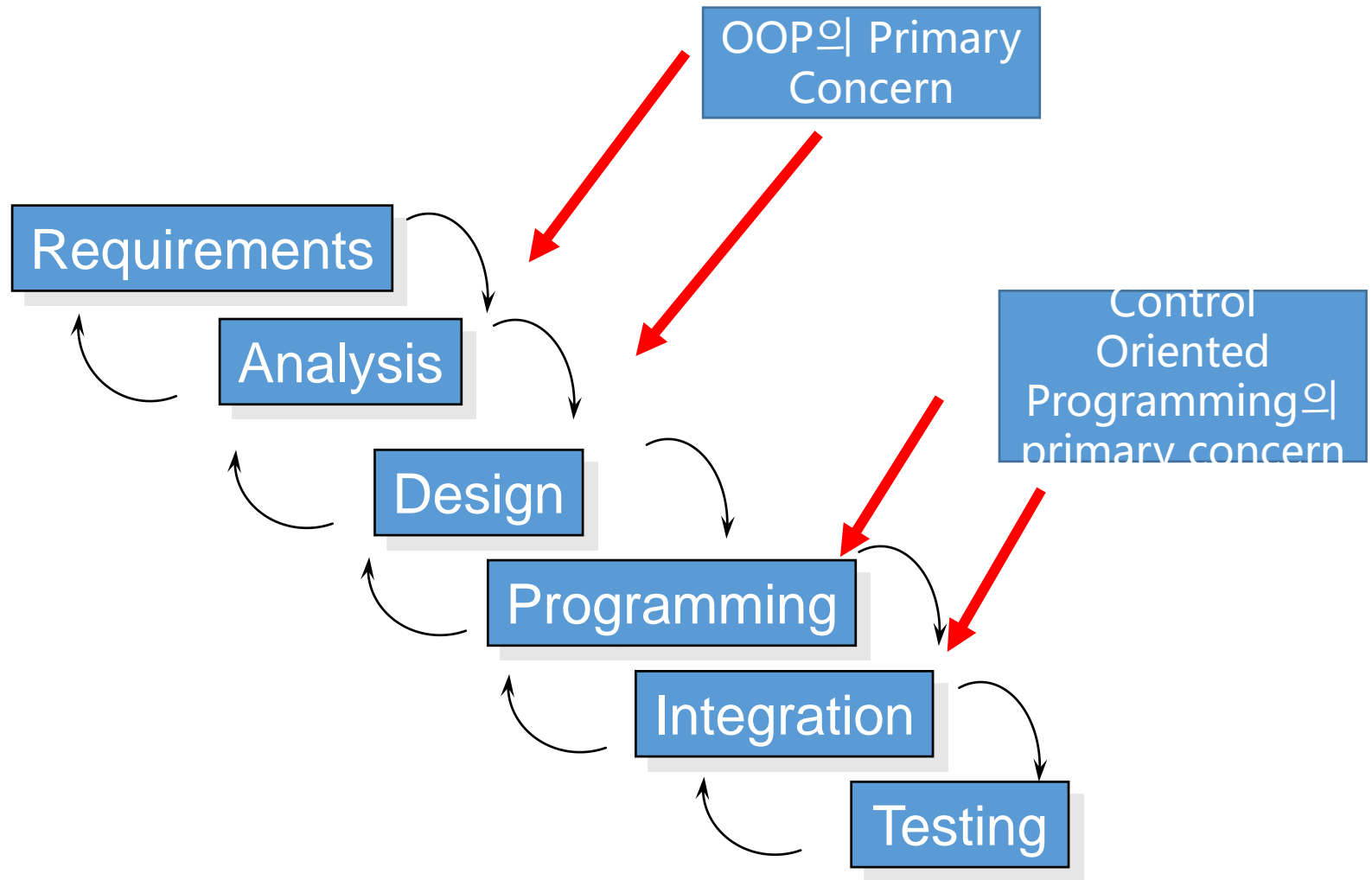
# High-Level Programming Paradigms

- Control-oriented Programming (before mid 80's)
  - Real world problem ➔ a set of functions
  - Data and functions are separately treated
  - Fortran, Cobol, PL/1, Pascal, C (1972, Bell Lab)
- Object-oriented Programming (after mid 80's)
  - Real world problem ➔ a set of classes
  - Data and functions are encapsulated inside classes
  - C++ (1983, Bell Lab)
  - Python (1991)
  - Java (1993)
  - and most Script Languages (Ruby, PHP, R,...)

# The Software Development Process: The WaterFall Model

- **Analyze the Problem**
  - Figure out exactly the problem to be solved.
- **Determine Specifications**
  - Describe exactly what your program will do. (not **How**, but **What**)
  - Includes describing the inputs, outputs, and how they relate to one another.
- **Create a Design**
  - Formulate the overall structure of the program. (*how* of the program gets worked out)
  - You choose or develop your own algorithm that meets the specifications.
- **Implement the Design (coding!)**
  - Translate the design into a computer language.
- **Test/Debug the Program**
  - Try out your program to see if it worked.
  - Errors (Bugs) need to be located and fixed. This process is called **debugging**.
  - Your goal is to find errors, so try everything that might “break” your program!
- **Maintain the Program**
  - Continue developing the program in response to the needs of your users.
  - **In the real world**, most programs are never completely finished – **they evolve over time**.

# Waterfall SW Development Model



# Typical Control-Oriented Programming:

## C code for TV operations

```
#include <stdio.h>
```

```
int power = 0;    // 전원상태 0(off), 1(on)
int channel = 1;  // 채널
int caption = 0;  // 캡션상태 0(off), 1(on)
```

```
main()
```

```
{
    power();
    channel = 10;
    channelUp();
    printf("%d□n", channel);

    displayCaption("Hello, World");
    // 현재 캡션 기능이 꺼져 있어 글짜 안보임

    caption = 1; // 캡션 기능을 켜다.
    displayCaption("Hello, World"); // 보임
}
```

```
power()
```

```
{
    if( power )
        { power = 0; } // 전원 off → on
    else { power = 1; } // 전원 on → off
}
```

```
channelUp()    { ++channel; }
```

```
channelDown() { --channel; }
```

```
displayCaption(char *text)
```

```
{
    // 캡션 상태가 on 일 때만 text를 보여준다.
    if( caption ) {
        printf( "%s □n", text);
    }
}
```

# Sample C program

(Function-based structure: **Top-Down Function Design**)

```
#include <stdio.h>
#include <stdlib.h>

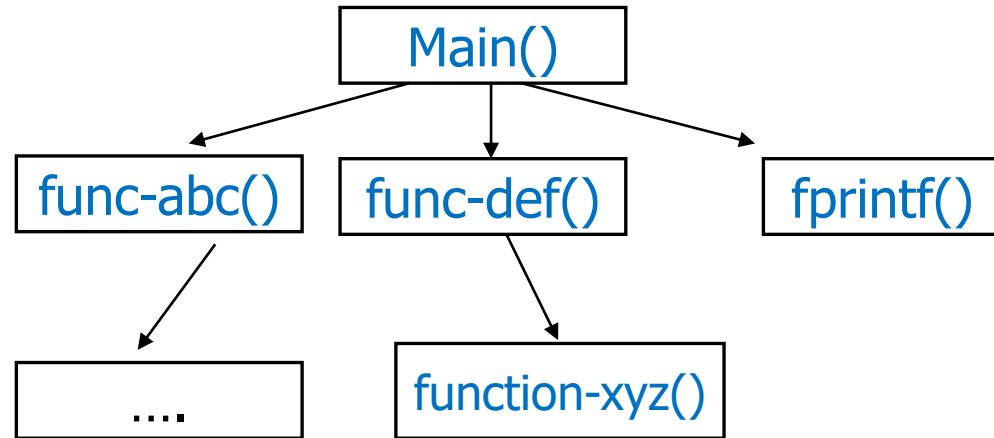
int main(int argc, char **argv) {

    int i, j, k, l;

    for(i=0; i < argc; i++) {
        func-abc();
        func-def();
        fprintf();
    }
}

func-abc ( ) { ..... }

func-def ( ) { ..... funct-xyx() }
```



# Struct in C

```
//includes and stuff
typedef struct Rectangle {
    int x,y;
    int width;
    int height;
} Rectangle;
```

```
void draw(Rectangle* obj) {
    printf("I just drew a nice
           %d by %d rectangle at
           position (%d, %d)!",
           obj->width,
           obj->height, obj->x,
           obj->y);
}
```

# OO Paradigm

- History of Programming Languages
- Abstract Data Type
- Python OOP Tutorial
- Motivation behind Python OOP



# History of Abstract Data Types

- An abstract data type is defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects
- Abstraction is simply the removal of unnecessary detail.
  - The idea is that to design a part of a complex system, you must identify what about that part others must know in order to design their parts, and what details you can hide.
  - The part others must know is the abstraction.
- Abstraction is information hiding
- Liskov, Barbara & Zilles, Stephen (1974). "Programming with Abstract Data Types". *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages*. SIGPLAN Notices.

- Basic Data Types
  - Integer
  - Floating Number
  - Character/String
  - Boolean
- Advanced Data Types
  - List (Array, Matrix)
  - Set
  - Dictionary
  - Tuple
- User-Defined Abstract Data Types (= Classes)
  - Student Data Type
  - Professor Data Type
  - Automobile Data Type
  - Bank-Account Data Type

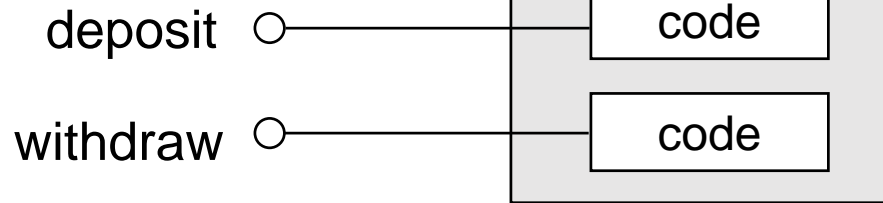
# Concept of Abstract Data Types [2/5]

**Object**

=

**Data + Operations**

Bank \_Account object



open ●  
close ●  
read ●  
write ●



myFile object

`myFile.open()` : myFile, please open yourself

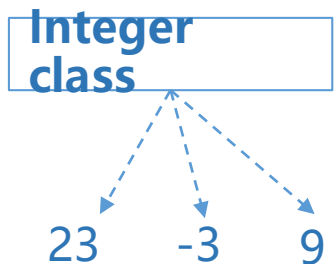
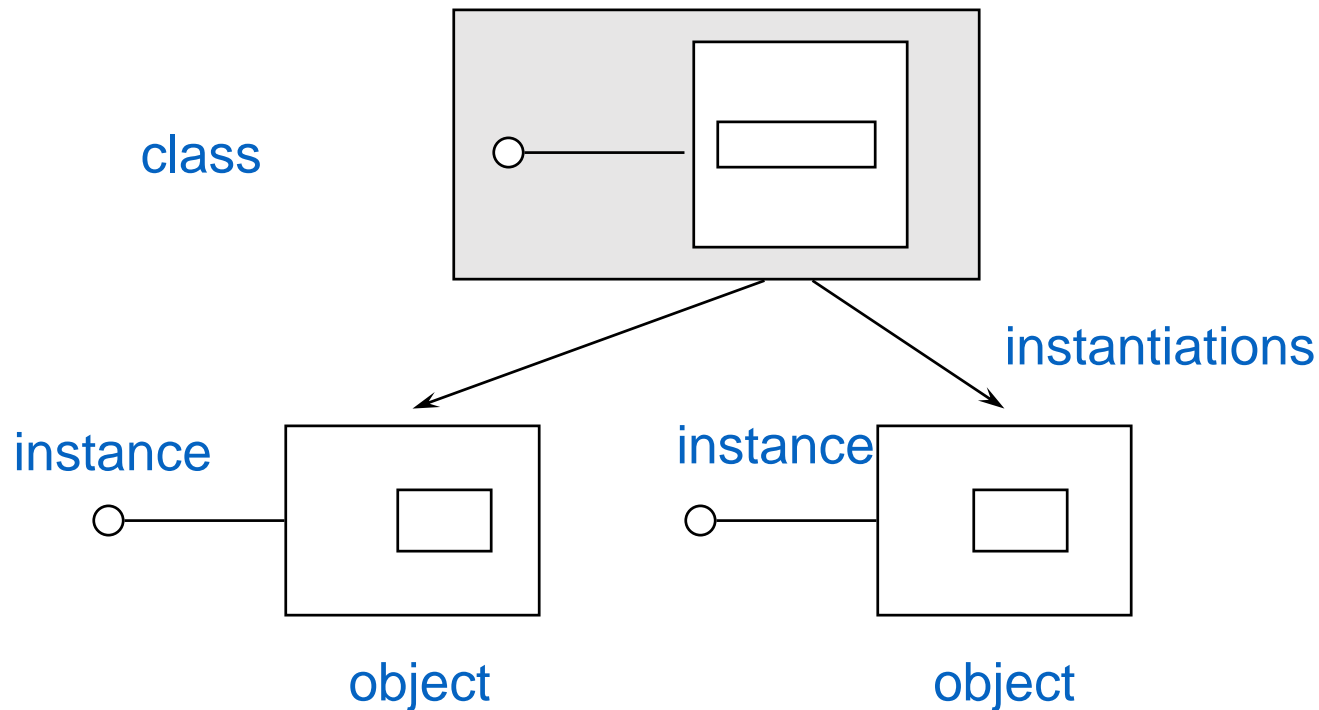
`myFile.read(ch)` : myFile, please give me the next char

`myFile.close()` : myFile, close yourself

# Concept of Abstract Data Types [3/5]

## Class

An **abstract data type** which define the **representation** and **behavior** of objects.

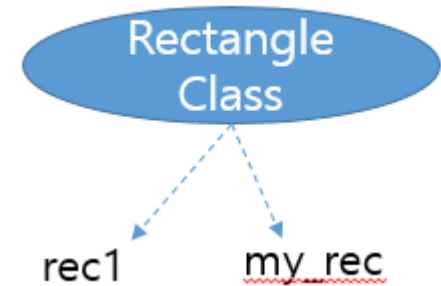
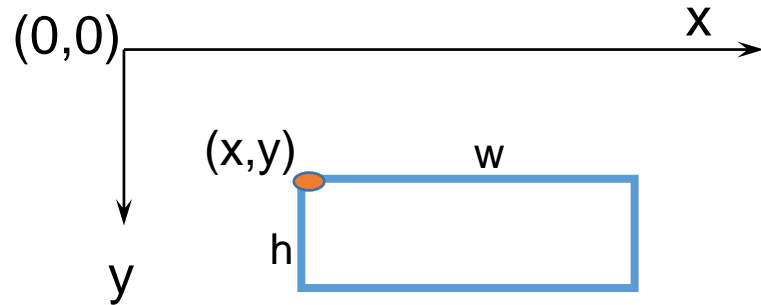


`X = 23`  
`Y = int(-3)`

# Concept of Abstract Data Types

[4/5]

## Define a Rectangle Class



Rectangle rec1, my\_rec  
rec1.create(5,5,10,5)  
my\_rec.create(10,10,10,5)

### Class **Rectangle**

variables

int **x, y, h, w**;

methods

**create** (int x1, y1, h1, w1)

{ x=x1; y=y1; h=h1; w=w1; }

**moveTo** (int x1, y1)

{ x=x1; y=y1; self.display(); }

**display** ()

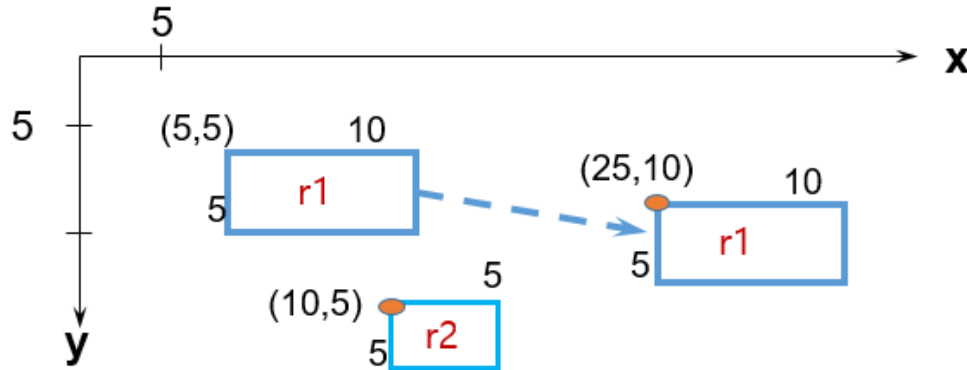
{ drawRectangle(x, y, h, w); }

End Class

# Concept of Abstract Data Types

[5/5]

## Example: Using Rectangle object



```
#Import Rectangle
```

```
Rectangle r1, r2;  
r1.create(5, 5, 10, 5);  
r1.display();  
r1.moveTo(25,10);
```

```
r2.create(10, 15, 5, 5);  
r2.display();
```

Rectangle object 를 직접  
manipulate 안했다면...

(5, 5, 5, 10) 에 (20, 5, 0, 0) 을  
더하여 (25,10,5,10)를  
만들고...

이런 rectangle 이 여러 개  
있다면...

# Why User-Defined Abstract Data Types?



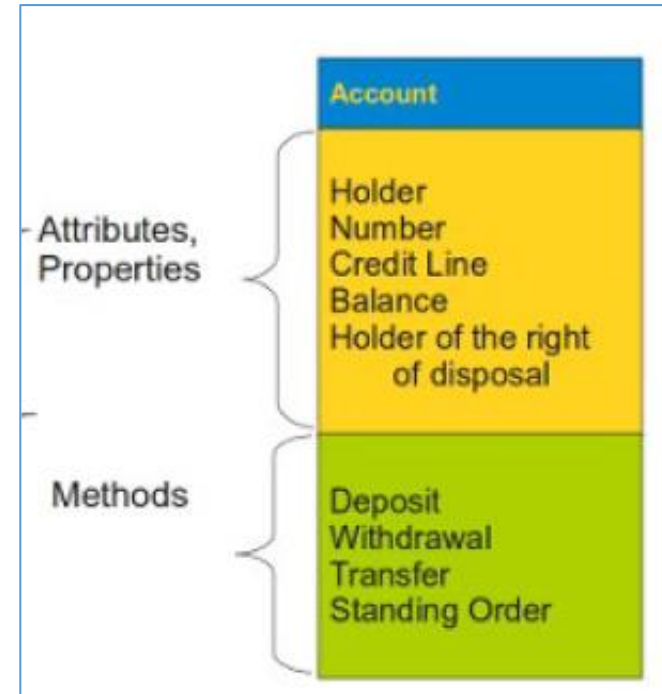
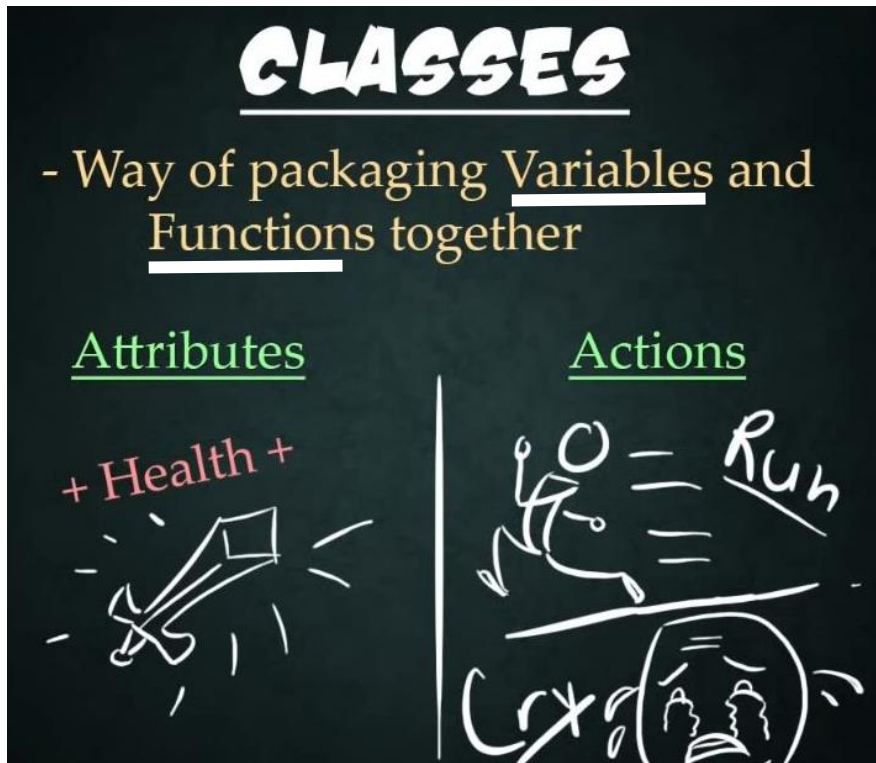
- Box를 표현하는 `width`, `length`, `height` 의 값들 & Box의 `volume` 을 고려
- Python에서 W, L, H 의 variable 을 써서 어떤 한 개 Box를 표현했다면....

`W = 10 / L = 15 / H = 12`

`Box_volume = W * L * H`

- (W, L, H) 3개의 data와 `Box_volume`이 합쳐져서 그 Box를 표현하고 있는데
  - Basic Data Type 만을 가지고는 이런 표현을 할 방법이 없네!
  - Variable들을 묶어서 복잡한 구조를 표현하면 좋을텐데... → `abstract data type`
- 앞페이지에서도 `rectangle` 을 (`x1`, `y1`, `width`, `height`) 로 표현하고 있는데...
  - `rectangle r1, r2` 하는 식으로 표현을 하니까 `r1` 과 `r2` 를 직접 `manipulate` 할 수 있네!!!!
- Box Class (abstract data type)를 만들고 필요한 object instance를 생성하여 coding에 이용하면 좋겠다!

# Inside Class



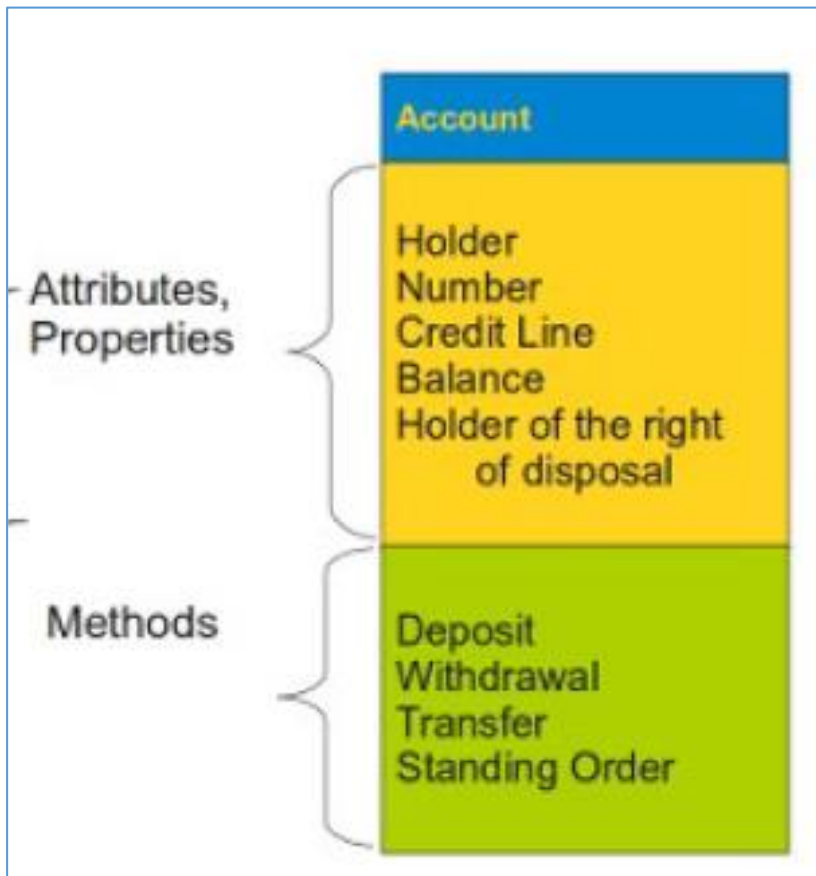
Attribute : Instance Variable, Variable, Property, Data Member

Action: Method, Instance Method, Procedure, Function, Member Function, Operation

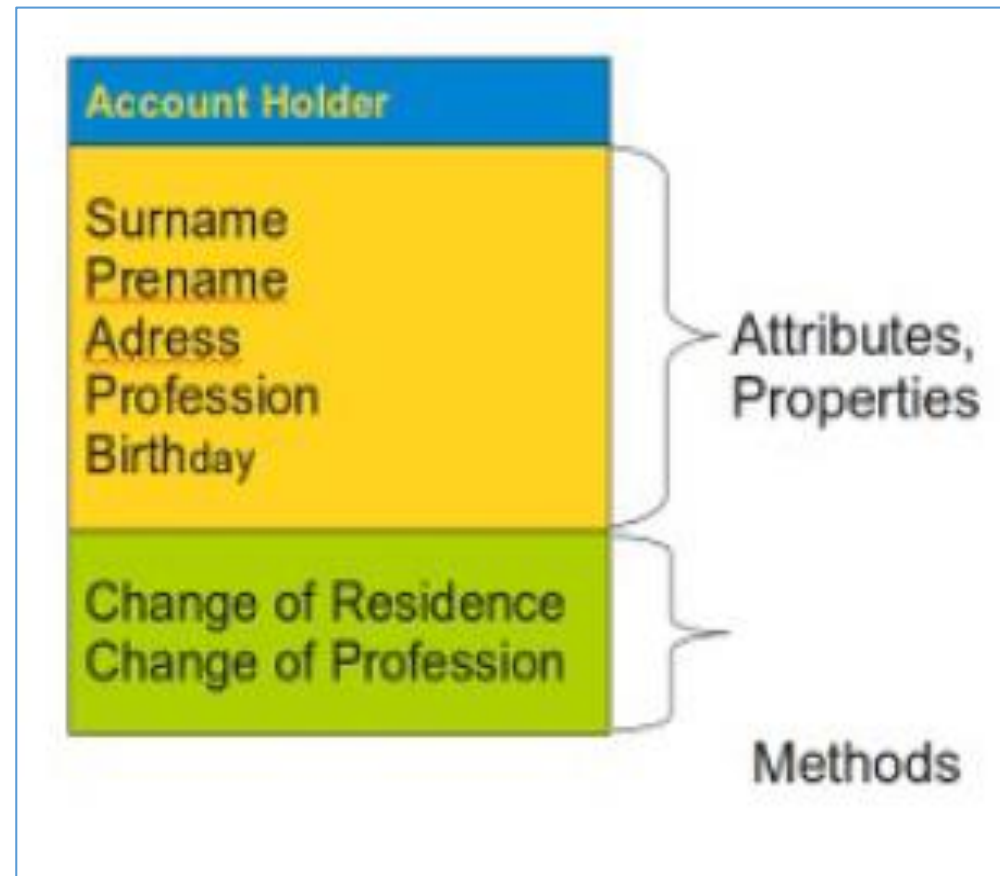


# Abstract Data Type of Banking Account [1/3]

## 은행 계좌 (Account)



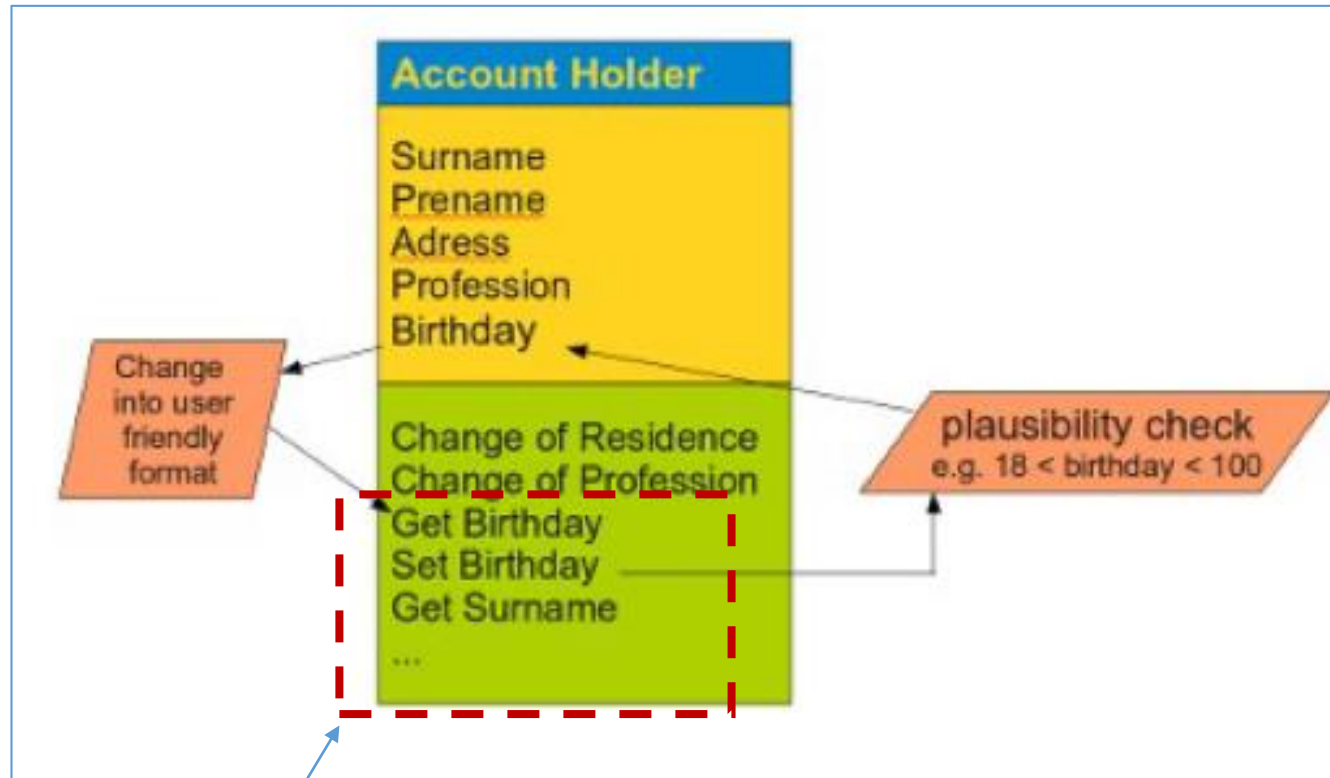
## 계좌 보유자 (Account Holder)



# Abstract Data Type of Banking Account [2/3]

Encapsulation of Data = Data Hiding

계좌 보유자



외부에 공개를 안하는 methods

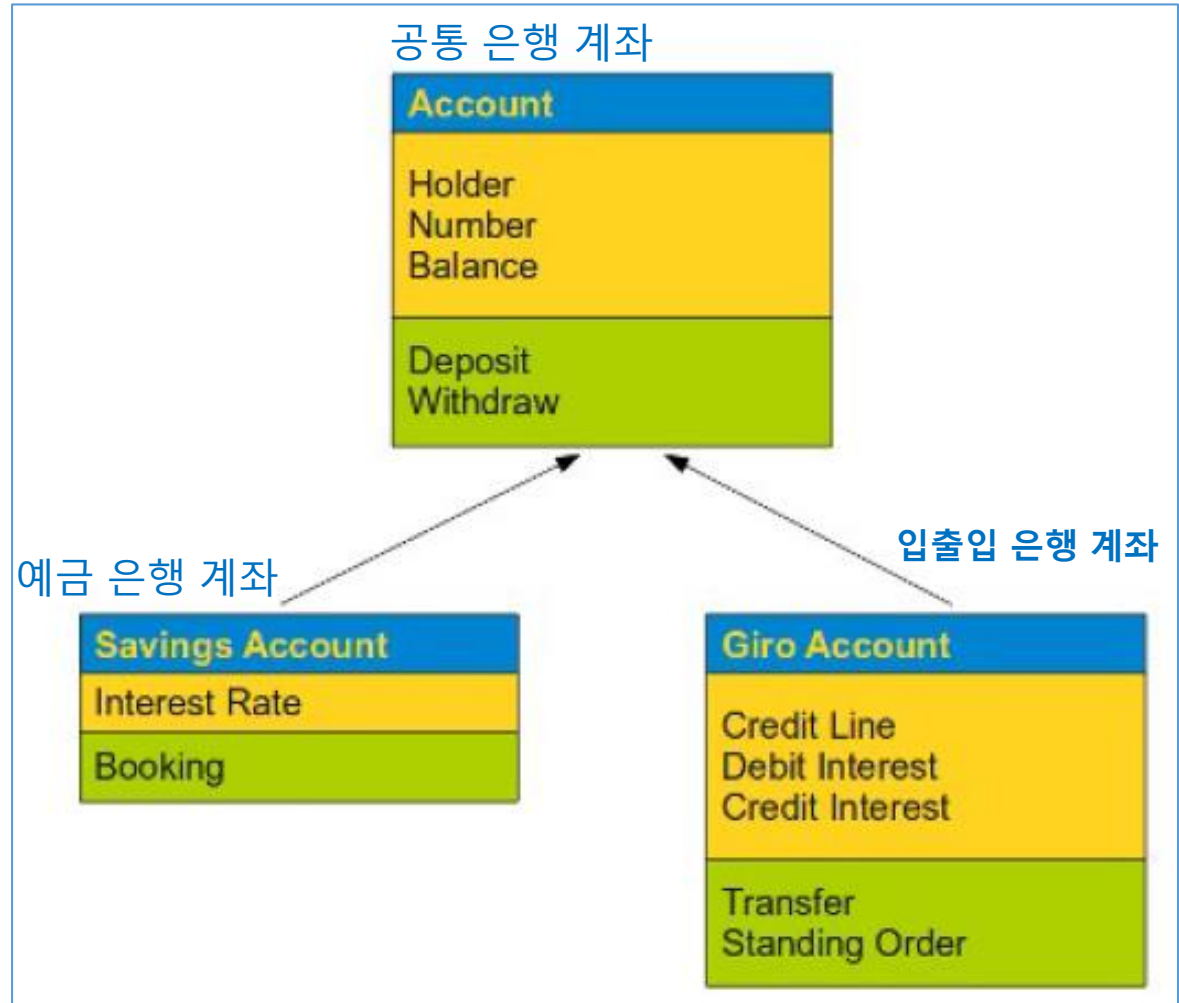
# Abstract Data Type of Banking Account [3/3]

Inheritance → Code Reusability → More Cleaner Way of Coding

은행 계좌



은행 계좌 Class Hierarchy

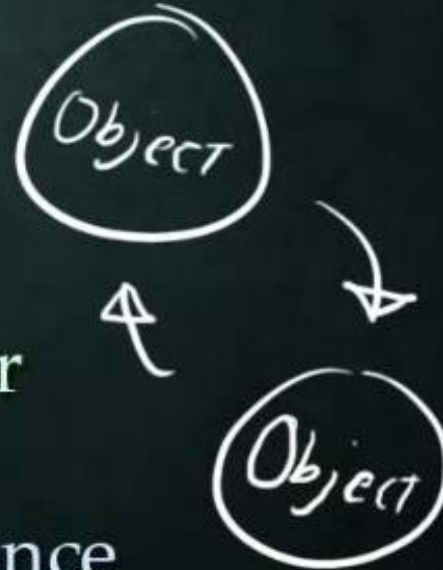


# OBJECT ORIENTED PROGRAMMING (OOP)

- Programming based around classes and instances of those classes (aka Objects)

- Revolves around how classes interact and Directly Affect one another

- We will focus on Inheritance

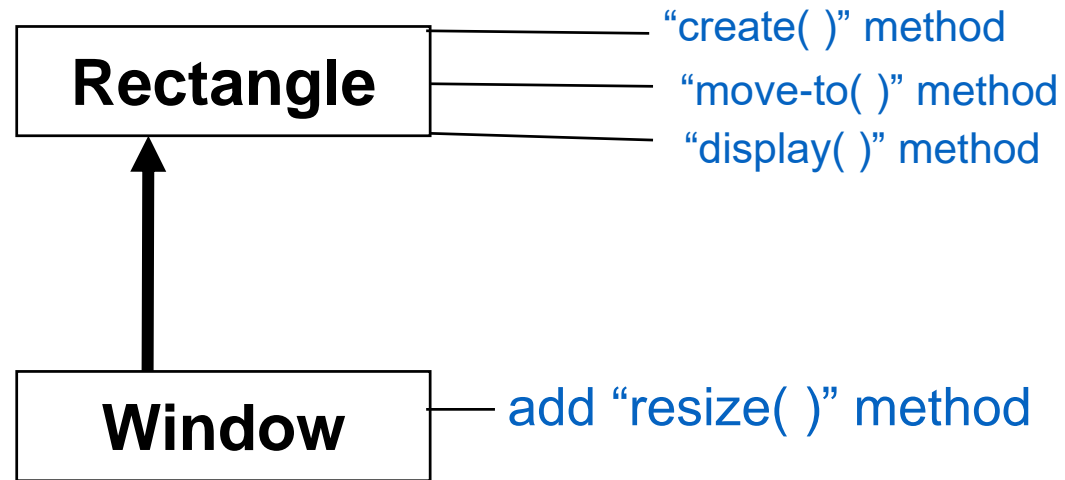


# Another Benefit of Abstract Data Types:

## Class Inheritance

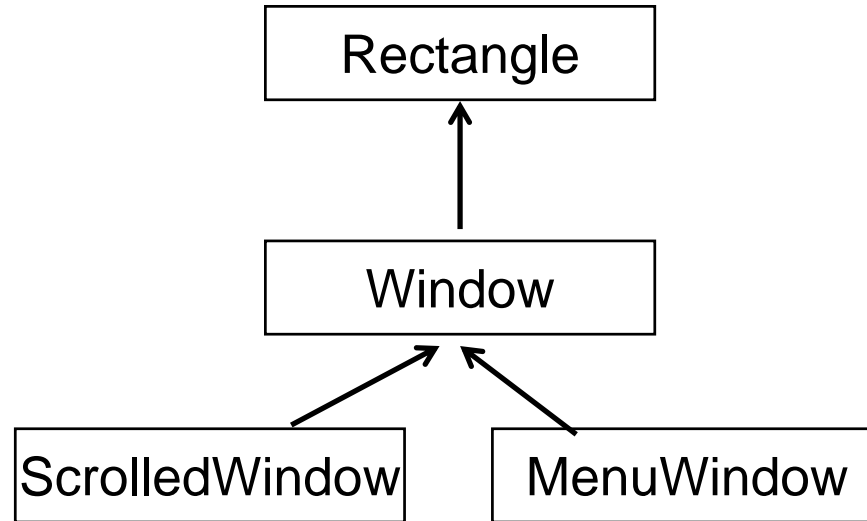
### Problem

Define a new class called Window, which is a rectangle, but also resizable.



```
Class Window
  inherit Rectangle
  add operation
    resize(int h1, w1)
      { h=h1; w=w1; display(); }
```

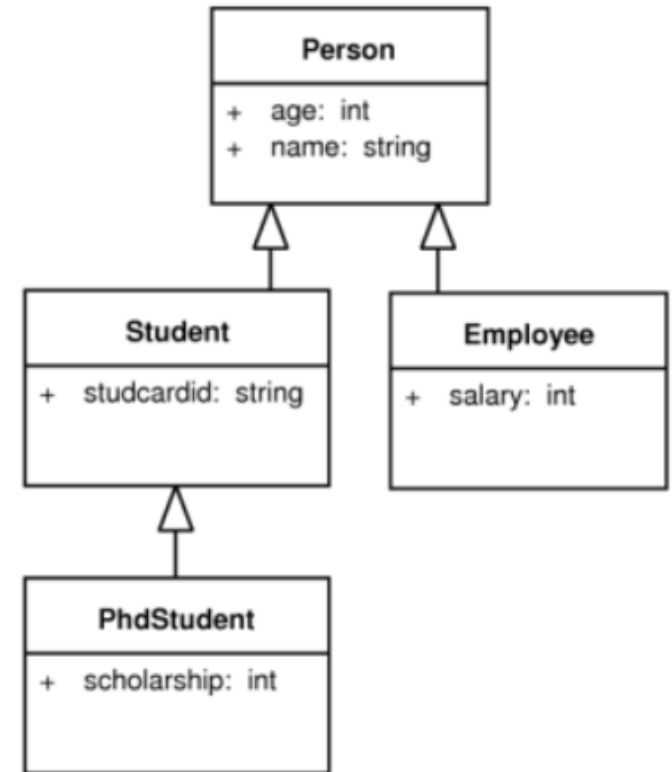
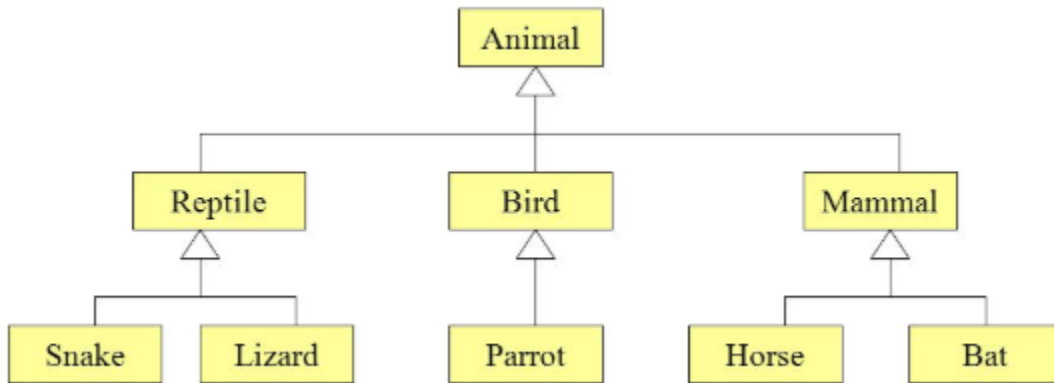
# Class Hierarchy provides Code Reusability and Cleaner Way of Programming



Inheritance builds class hierarchies which are reusable and opens the possibility of application frameworks for domain reuse

# Examples of Class Hierarchy

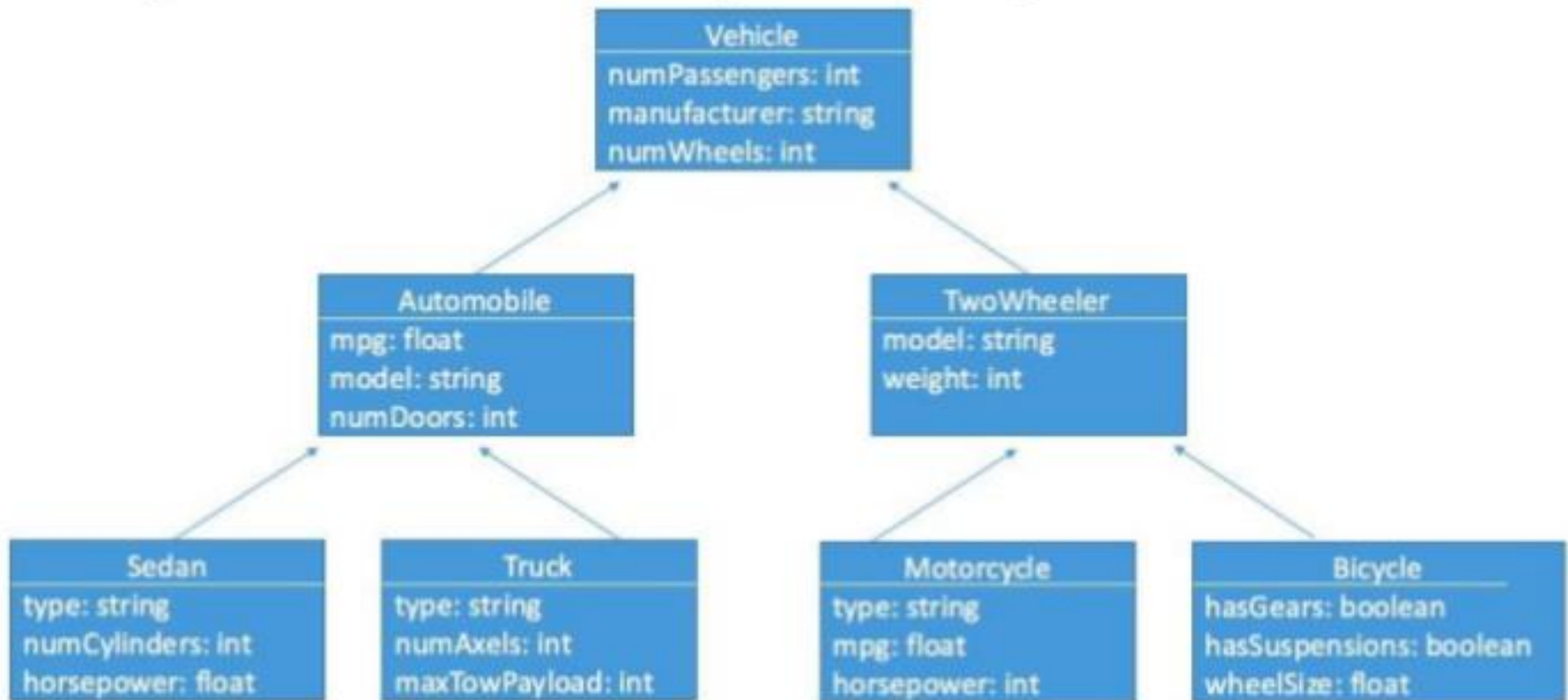
[1/4]





# Examples of Class Hierarchy

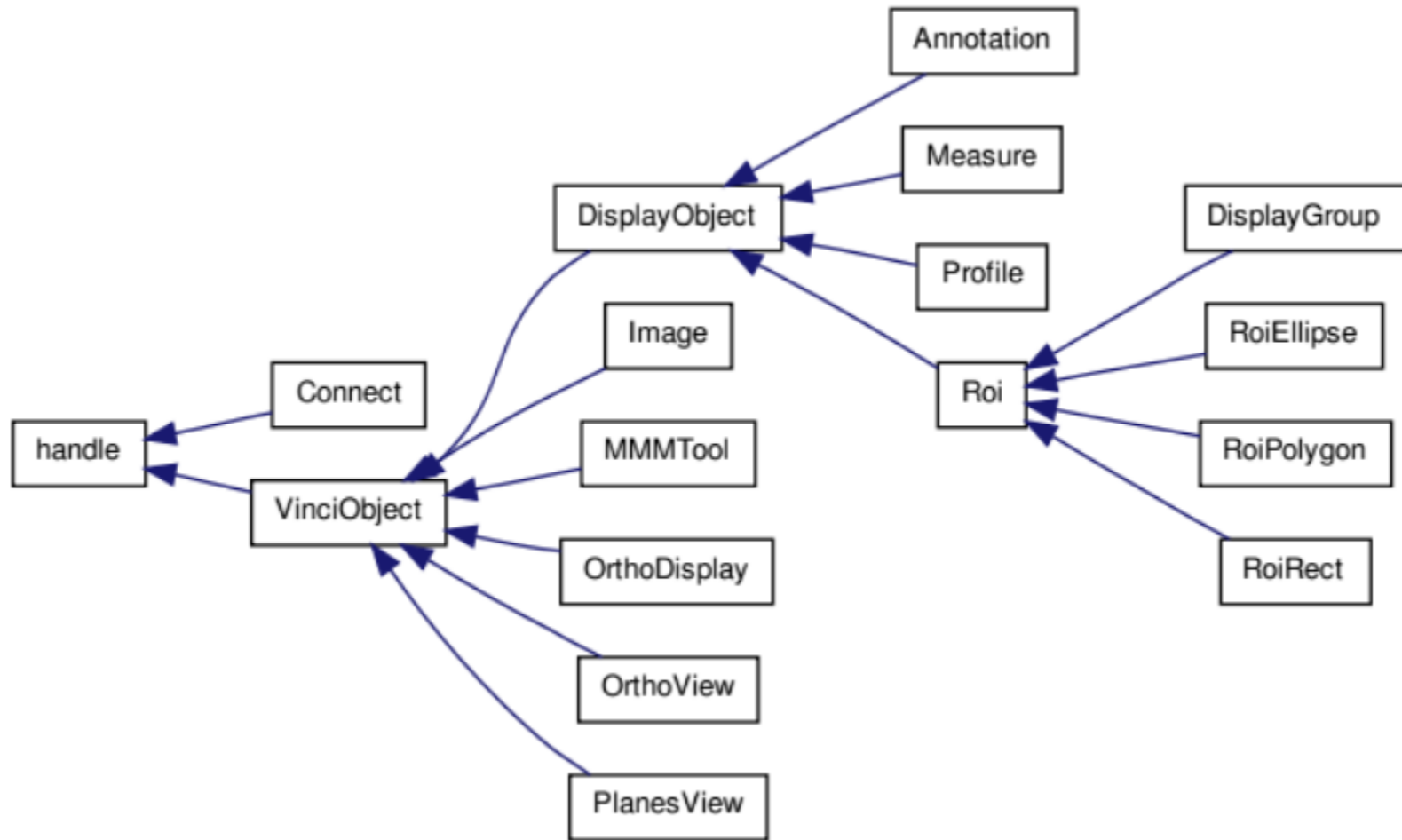
[2/4]





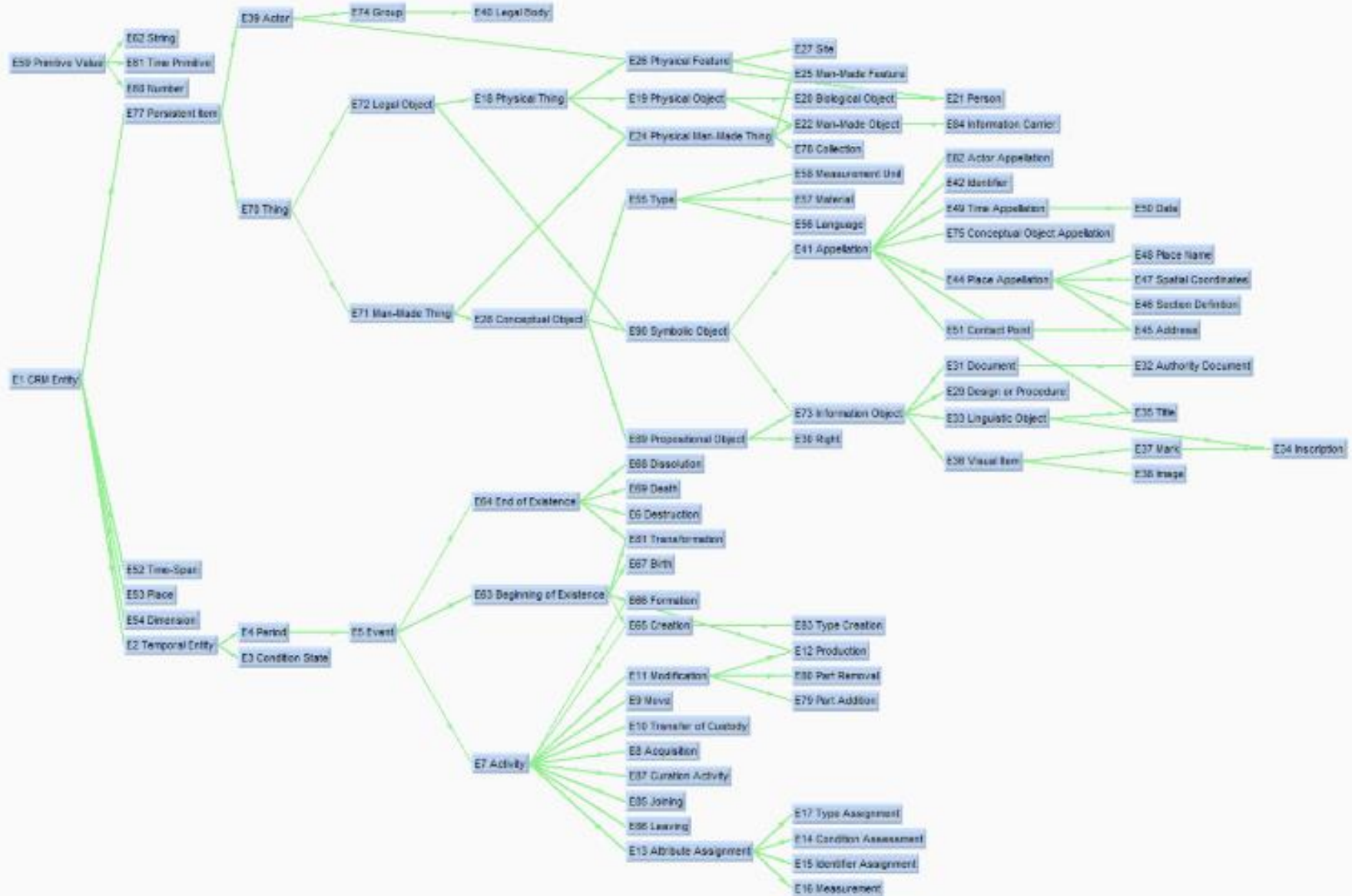
# Examples of Class Hierarchy

[3/4]



# Examples of Class Hierarchy

[4/4]



# OO Paradigm

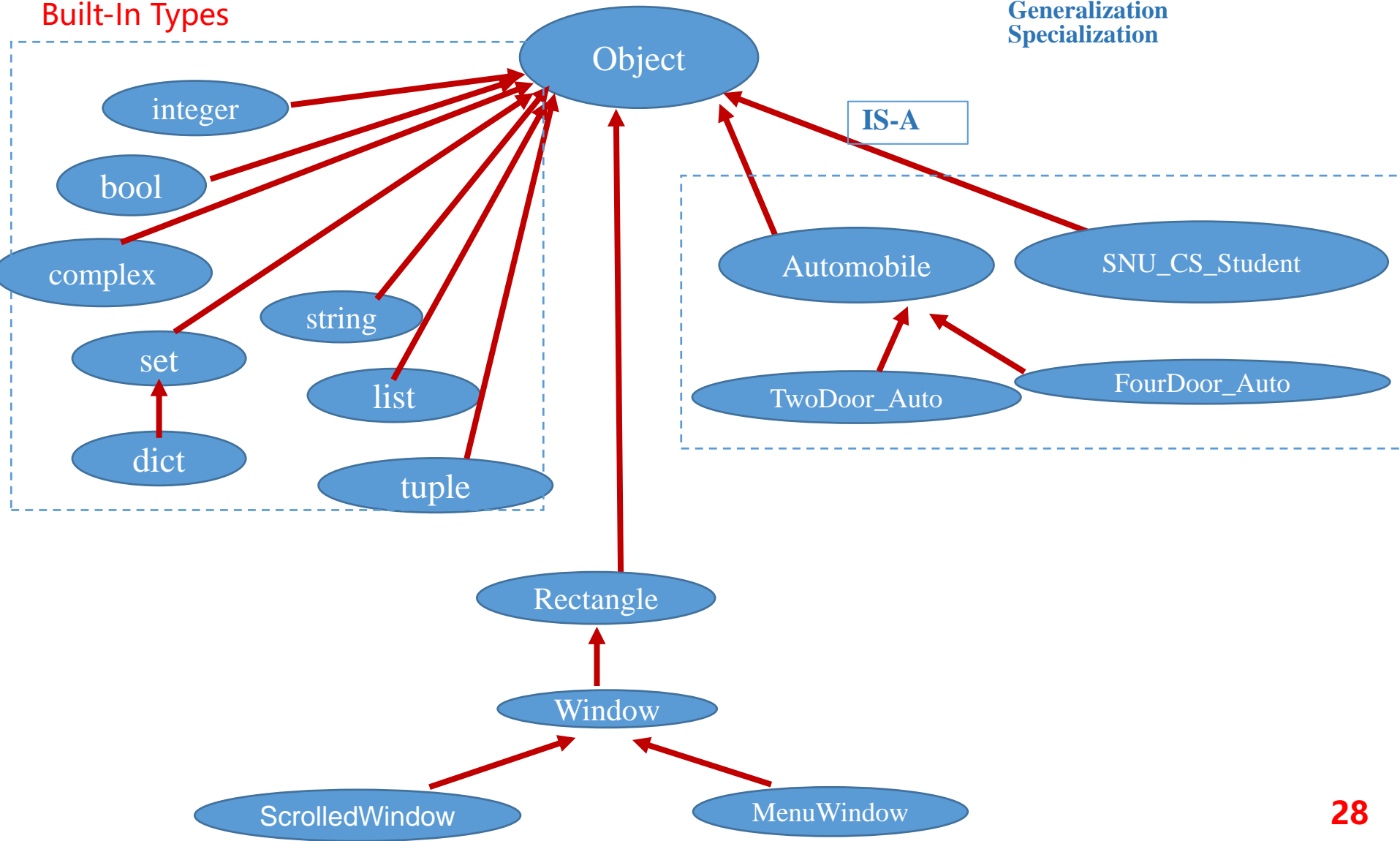
- History of Programming Languages
- Abstract Data Type
- [Python OOP Tutorial](#)
- Motivation behind Python OOP

# Everything is an Object in OOP

Python's type system (class structure)

Built-In Types

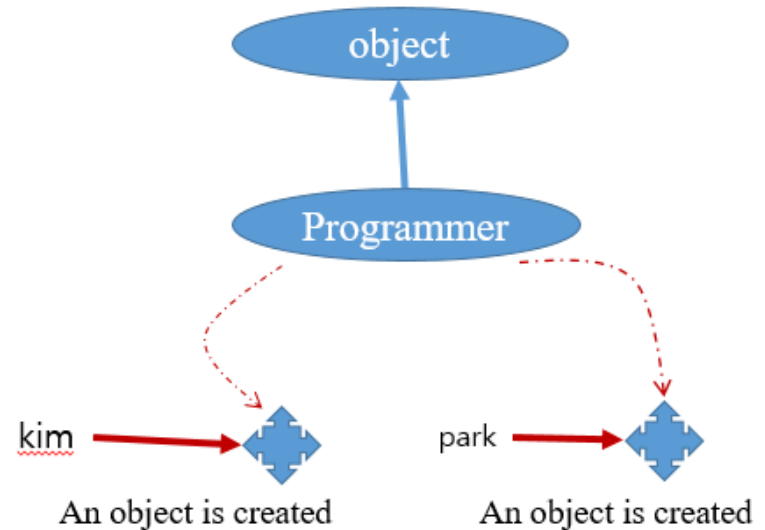
Generalization  
Specialization



# Very Simple Class and Its Instances

```
>>> class Programmer:  
>>>     pass
```

```
>>> kim = Programmer()  
>>> park = Programmer()
```



Begin with 'class' keyword

ALWAYS Capitalize First Letter of class Name

```
class Test:  
    pass
```

```
x = Test()
```

Create Class instance is just like calling a function!!

Class name의 첫글자를  
대문자로 하는것은 권고사항!

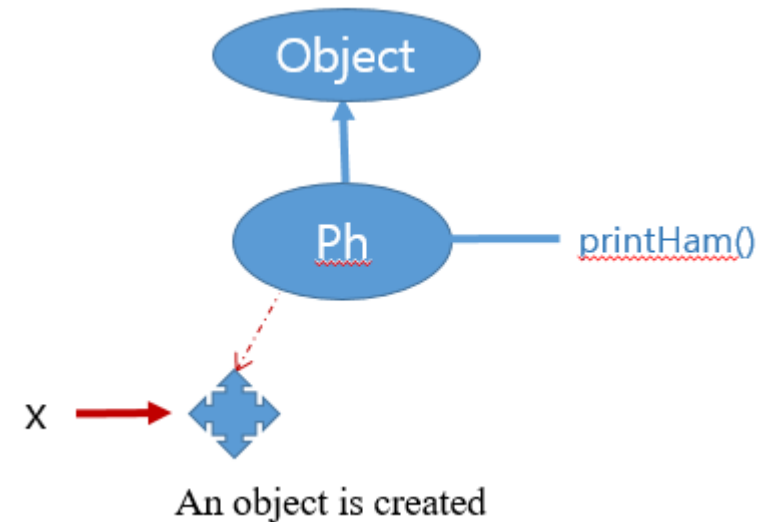
# Object에 속한 function들의 선언에는 “self“가 1st parameter로 있어야 한다

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> class Ph():
>>>     def printHam():
>>>         print("ham")

>>> x = Ph()
>>> x.printHam()
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    x.printHam()
TypeError: printHam() takes 0 positional arguments but 1 was given
>>> |
```

```
>>>
>>>
>>>
>>> class Ph():
>>>     def printHam(self):
>>>         print("ham")

>>> x = Ph()
>>> x.printHam()
ham
>>> |
```



- Ph 라는 class를 만들고자 한다
- Ph class 에는 두개의 variable (y, z) 와 한 개의 function (printHam( )) 을 두고자 한다

## INITIALIZATION

```
class Ph:
```

```
    def __init__(self):
```

```
        self.y = 5
```

```
        z = 5
```

```
    def printHam(self):
```

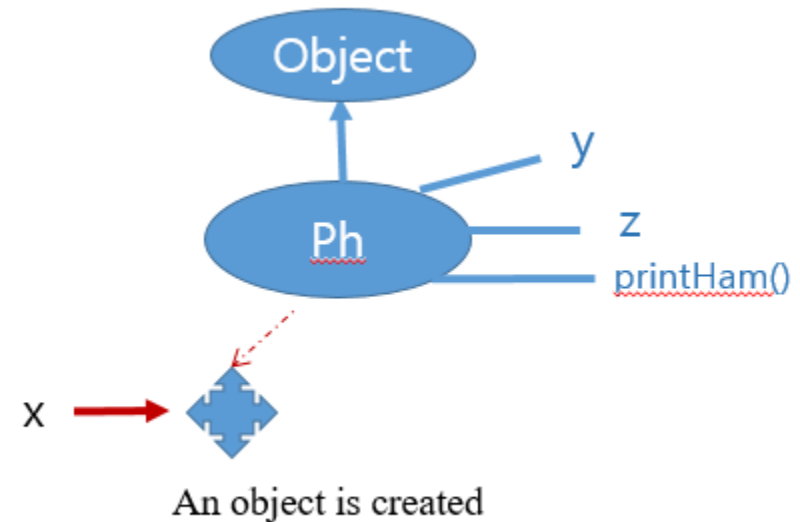
```
        print("ham" )
```

```
x = Ph()
```

```
x.printHam()
```

```
print(x.y)
```

```
print(x.z)
```



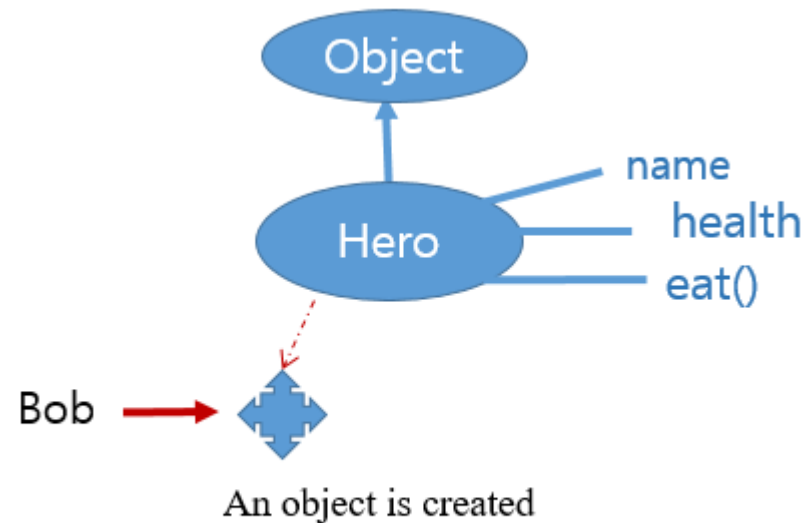
- Hero 라는 class를 만들고자 한다
- Hero class 에는 두개의 variable (name, health) 와 한개의 function ( eat() ) 을 두고자 한다

```
74 classes_example.py - C:\Users\The_Captain\Desktop\classes_example.py
File Edit Format Run Options Windows Help

class Hero:
    def __init__(self, name):
        self.name = name
        self.health = 100
    def eat(self, food):
        if (food == 'apple'):
            self.health -= 100
        elif (food == 'ham'):
            self.health += 20

Bob = Hero("Bob")
print(Bob.name)
print(Bob.health)
Bob.eat('ham')
print(Bob.health)

>>>
>>>
>>>
Bob
100
120
```





# Python Class Syntax

## Class Declaration

```
class class_name (superclasses):
```

```
    class_var1
```

```
    class_var2
```

```
    def __init__(self, para1, para2):
```

```
        self.ins_var1 = para1
```

```
        self.ins_var2 = para2
```

```
    def ins_func(self, para1, para2):
```

```
        <statement1>
```

```
        <statement2>
```

```
    @classmethod
```

```
    def class_func(cls):
```

```
        <statement1>
```

```
        <statement2>
```

인스턴스변수: Instance Variable  
인스턴스함수: Instance Function  
(= Instance Method)

클래스변수: Class Variable  
클래스함수: Class Function  
(= Class Method)

# Example: Service Class

[1/2]

- **Service** 라는 Class를 만들고자 한다
- Service Class 에는 `Sum(a, b)` 이라는 function만 가지고 있다
- Service Class를 만들고, **Service Instance (Object)** 를 생성한후에 **Pey** 라고 naming
- **Pey** 에게 `3 + 4` 의 작업을 시키려고 한다

```
>>> class Service:
...
...     def sum(self, a, b):                # 더하기 서비스
...         result = a + b
...         print("%s + %s = %s입니다." % (a, b, result))
...
>>>
```

```
>>> pey = Service()
```

```
>>> pey.sum(1,1)
1 + 1 = 2입니다.
```

# Example: Service Class [2/2]

- 앞페이지의 **Service Class** 내부에 `name` 이라는 instance variable을 두고 Service Class를 사용하는 사용자의 이름을 저장하여 사용하려고 한다

```
>>> class Service:
...
...     def __init__(self, name):
...         self.name = name
...     def sum(self, a, b):
...         result = a + b
...         print("%s님 %s + %s = %s입니다." % (self.name, a, b, result))
```

Instance variable

Instance method

```
>>> pey = Service("홍길동")
>>> pey.sum(1, 1)
홍길동님 1 + 1 = 2입니다.
```

```
>>> pey.name      # OK
>>> Service.name  # not
allowed
```

# Class Hierarchy

[1/2]

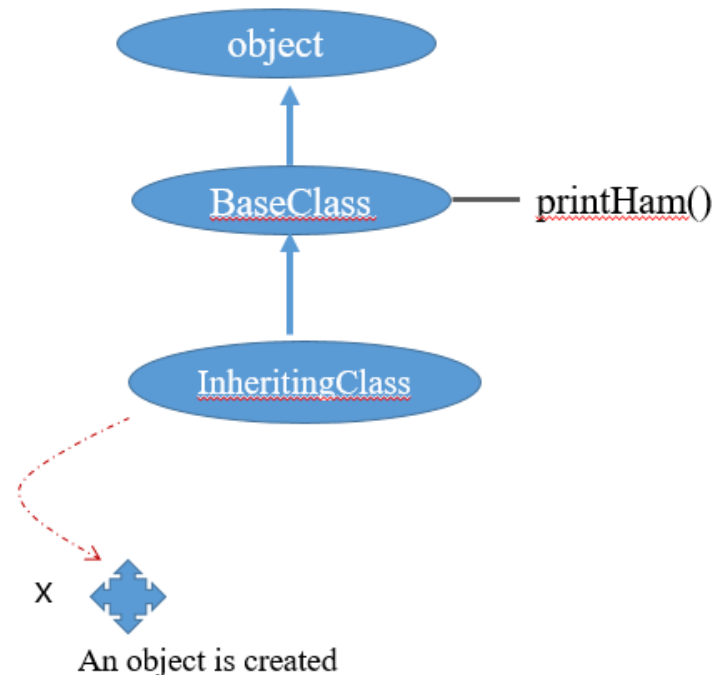
- Function 1개 (printHam( )) 이 들어 있는 “BaseClass” Class를 만든다
- “BaseClass” Class로 부터 subclass “InheritingClass” 를 만들려고 한다
- “InheritingClass”는 “BaseClass”로 부터 printHam( ) 을 상속 (Inheritance) 받는다.

## SIMPLE INHERITANCE EXAMPLE

```
class BaseClass (object):  
    def printHam(self):  
        prin ('ham')  
  
class InheritingClass (BaseClass):  
    pass
```

*Inherit from this class*

```
x = InheritingClass()  
x.printHam()
```



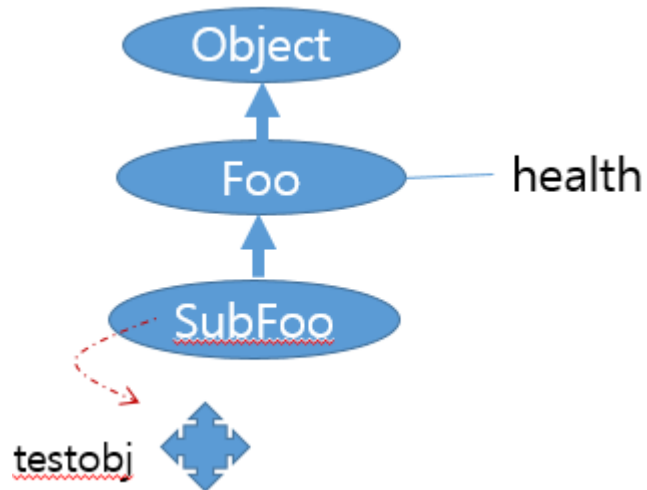
# Class Hierarchy

[2/2]

```
class Foo(object):  
    def __init__(self):  
        self.health = 100
```

```
Class SubFoo(Foo):  
    pass
```

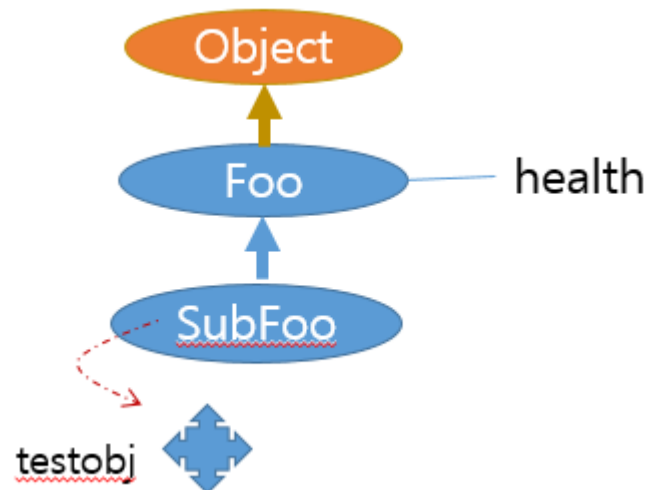
```
testobj = SubFoo()  
testobj.health
```



```
class Foo:  
    def __init__(self):  
        self.health = 100
```

```
Class SubFoo(Foo):  
    pass
```

```
testobj = SubFoo()  
testobj.health
```

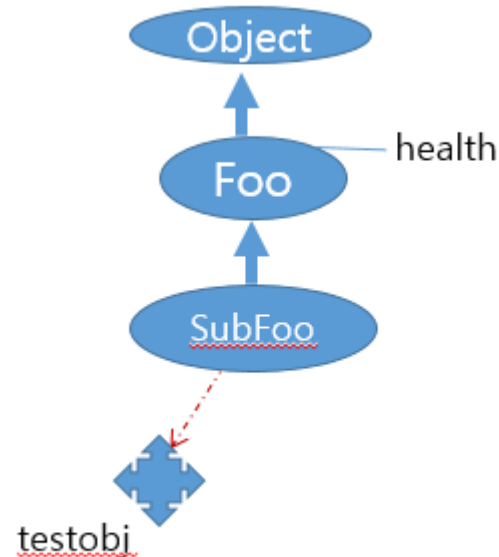


```
class Foo(object):  
    def __init__(self):  
        self.health = 100
```

```
class SubFoo(Foo):  
    pass
```

```
testobj = SubFoo()  
testobj.health
```

SubFoo class에서는  
Foo class의 instance  
variable 을 inherit 함



```
class Foo(object):  
    def __init__(self):  
        self.health = 100  
class SubFoo(Foo):  
    def __init__(self):  
        self.muscle = 200
```

```
testobj = SubFoo()  
testobj.health  
testobj.muscle
```

SubFoo class에서는  
Foo class의 instance  
variable 을 inherit  
안함

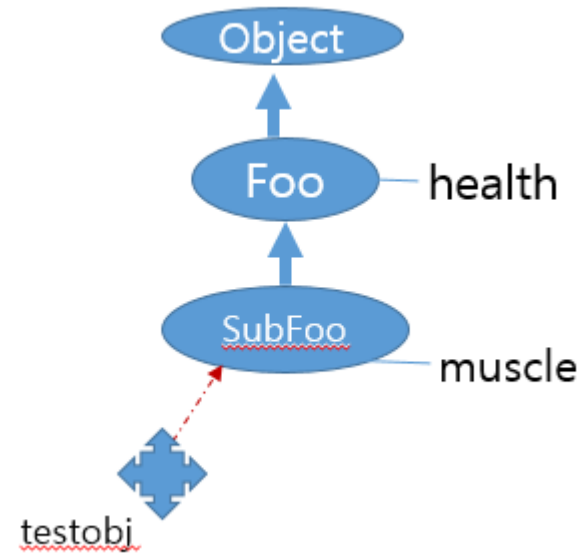
# Understanding Class Inheritance [2/5]

```
class Foo(object):  
    def __init__(self):  
        self.health = 100
```

```
class SubFoo(Foo):  
    def __init__(self):  
        super().__init__()  
        self.muscle = 200
```

```
testobj = SubFoo()  
testobj.health  
testobj.muscle
```

Superclass가  
여러개있을때는  
적절한것으로  
찾아서



SubFoo class에서는 Foo class의  
instance variable 을 inherit 받고  
자체적인 instance variable도  
선언

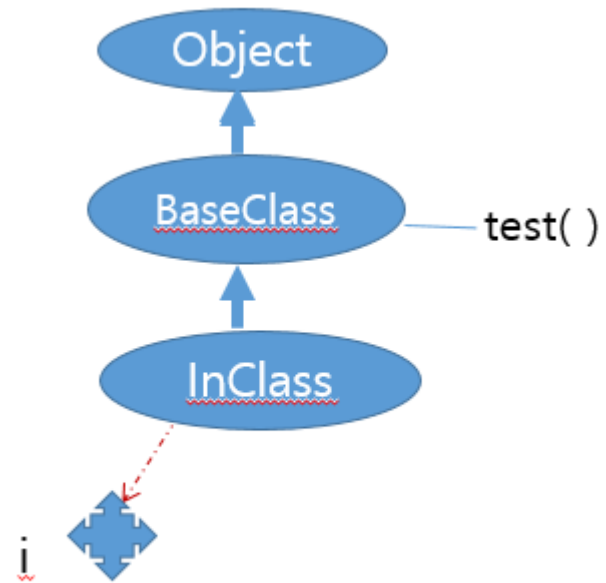
# Understanding Class Inheritance [3/5]

```
74 OverridingExample.py - C:/Users/The_Captain/Desktop/OverridingExample.py
File Edit Format Run Options Windows Help

class BaseClass(object):
    def test(self):
        print("ham")

class InClass(BaseClass):
    def test(self):
        print("hammer time")

i = InClass()
i.test()
```



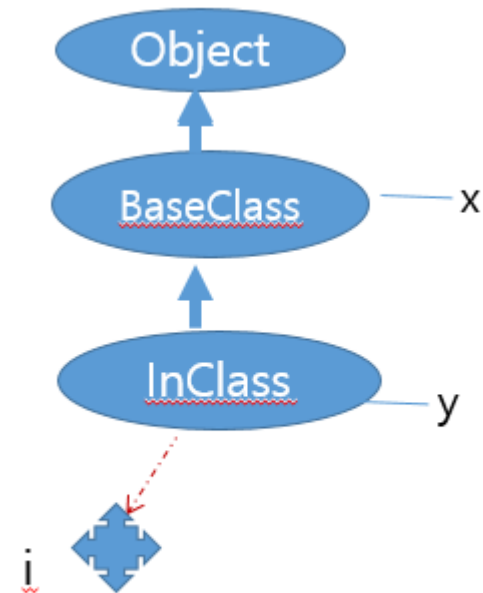
InClass class에 test()가 있어서  
BaseClass class의 test()를 override하므로  
BaseClass의 test()는 inherit 안됨



# Understanding Class Inheritance

[4/5]

```
class BaseClass(object):  
    def __init__(self):  
        self.x = 100  
class InClass(BaseClass):  
    def __init__(self):  
        super().__init__()  
        self.y = 200  
  
i = InClass()  
print("Object i's inherited variable:", i.x)  
print("Object i's locally defined variable:", i.y)
```



InClass class에 `__init__()`가 있어서  
BaseClass class의 `__init__()`를 override하므로  
BaseClass의 `__init__()`는 수행이 안된다.  
그러나 `super(InClass, self).__init__()`에 의해서  
BaseClass의 instance variable을 inherit 받는다

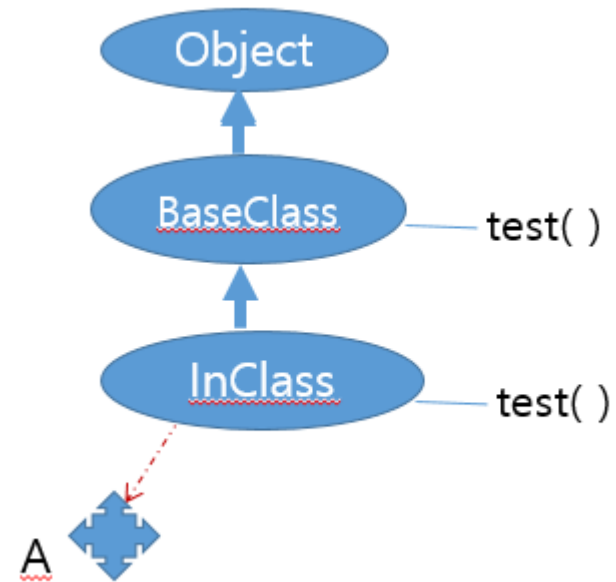
# Understanding Class Inheritance [5/5]

```
File Edit Format Run Options Windows Help
class BaseClass(object):
    def test(self):
        print("ham" )

class InClass(BaseClass):
    def test(self):
        print("hammer time")

print BaseClass.__subclasses__()

Python 2.7.6 Shell
```



\*\* InClass class에서는 BaseClass class의 test()을 inherit 받지 않고 같은이름의 test() 를 locally define했다

\*\* \_\_subclasses\_\_() 는 subclass들을 return하는 함수

\*\* \_\_superclasses\_\_() 는 존재를 안함

```
>>> A = InClass()
>>> A.test()
>>> BaseClass.__subclasses__()
>>> InClass.__superclasses__()
```

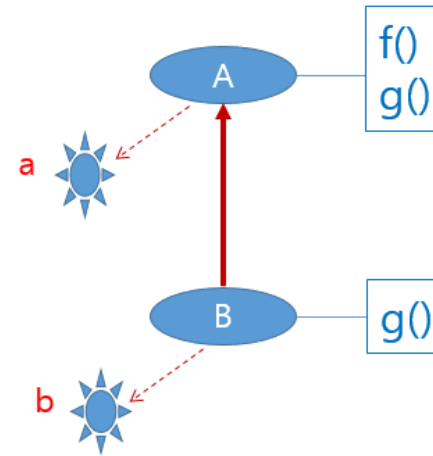
# What will be the output of the following program?

```
class A:
    def f(self):
        return self.g()

    def g(self):
        return 'A'

class B(A):
    def g(self):
        return 'B'

a = A()
b = B()
print(a.f(), b.f())
print(a.g(), b.g())
```



# OO Paradigm

- History of Programming Languages
- Abstract Data Type
- Python OOP Tutorial
- Motivation behind Python OOP

# OOP Motivational Example: Auto Volume Computation [1/2]

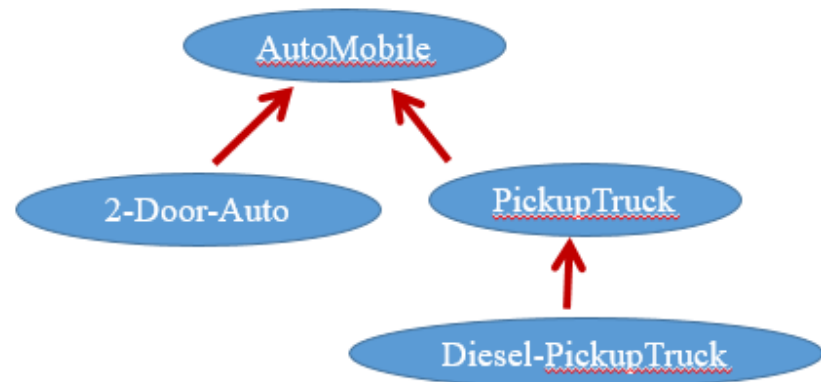
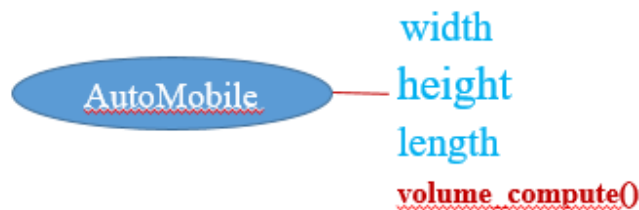
## Function-Oriented Python Version

```
def volume_compute(x, y, z):  
    return x * y * z  
  
def volume_compute1(x, y, z, l)  
    return x * y * z + l  
  
def Test():  
    print("My Automobile's volume is:", volume_compute(10, 15, 25))  
    print("Your PickupTruck's volume is:", volume_compute1(10, 15, 25, 1000))
```



**\*\* My Automobile, Your PickupTruck** 이라는 실체? (10, 15, 25), (10, 15, 25, 1000)?  
2-Door-Auto 혹은 Diesel-PickupTruck 같은 비슷한 자동차에 대해서 무언가를 하고 싶을때?

**\*\* In OOP**



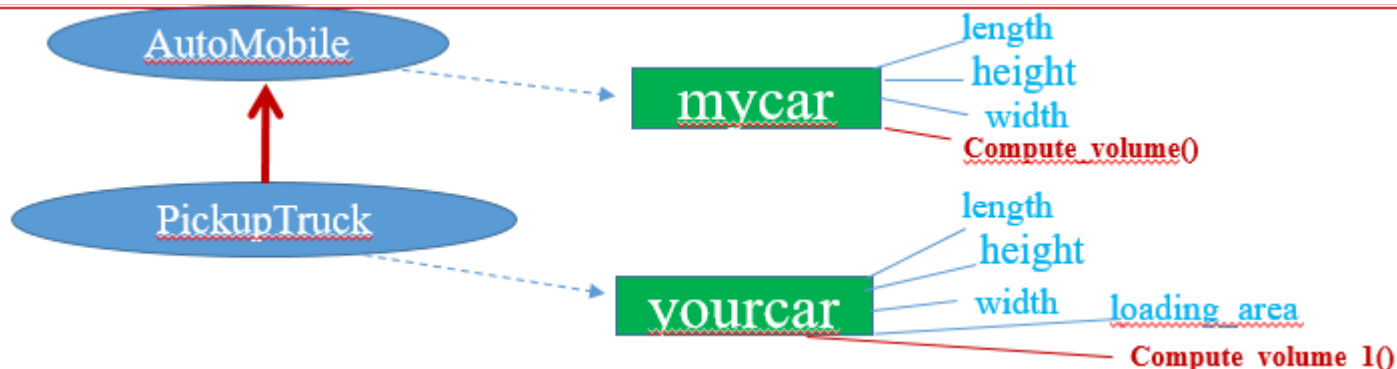
# OOP Motivational Example: Auto Volume Computation [2/2]

Object-Oriented

```
class Automobile(object):
    #
    def __init__(self, width, height, length):
        self.width = width
        self.height = height
        self.length = length
        print("A new Automobile instance is allocated")
    #
    def compute_volume(self):
        return self.width * self.height * self.length

class Pickup_Truck(Automobile):
    #
    def __init__(self, width, height, length, loading_area):
        Automobile.__init__(self, width, height, length)
        self.loading_area = loading_area
    #
    def compute_volume_1(self):
        return self.width * self.height * self.length + self.loading_area

def test():
    #
    mycar = Automobile(10,15,25)
    #
    print "Mycar\'s volume is ", mycar.compute_volume()
    #
    yourcar = Pickup_Truck(10,15,25,1000)
    print "Yourcar\'s volume is ", yourcar.compute_volume()
    print "Yourcar\'s volume with loading section is ", yourcar.compute_volume_1()
```



# Motivating Example of OO: Book Record

[1/5]

- Suppose we want to keep track of the books in you
- For each book, you want to store: `title`, `author`, `year_published`
- **Example**
  - Book1: “The Catcher in the Rye” , “J. D. Sallinger” , 1951
  - Book2: “The Brothers Karamazov” , “F. Dostoevsky” , 1880

- **Option1: Using Only Variables**

`book1Title = “The Catcher in the Rye”`

`book1Author = “J. D. Sallinger”`

`book1Year = 1951`

`book2Title = “The Brothers Karamazov”`

`book2Author = “F. Dostoevsky”`

`book2Year = 1880;`

- Example

- Book1: “The Catcher in the Rye” , “J. D. Sallinger” , 1951
- Book2: “The Brothers Karamazov” , “F. Dostoevsky” , 1880

- Option 2: Using List

```
book1 = [“The Catcher in the Rye”, “J.D. Sallinger”, 1951]
```

```
book2 = list()
```

```
book2.append(“The Brothers Karamazov”)
```

```
book2.append(“F. Dostoevsky”)
```

```
book2.append(1880)
```



- Example

- Book1: “The Catcher in the Rye” , “J. D. Sallinger” , 1951
- Book2: “The Brothers Karamazov” , “F. Dostoevsky” , 1880

- Option 3: Using Dictionary

```
book1 = {"title": "The Catcher in the Rye",  
         "author": "J.D. Sallinger",  
         "year": 1951}
```

```
book2 = dict()
```

```
book2["title"] = "The Brothers Karamazov"
```

```
book2["author"] = "F. Dostoevsky"
```

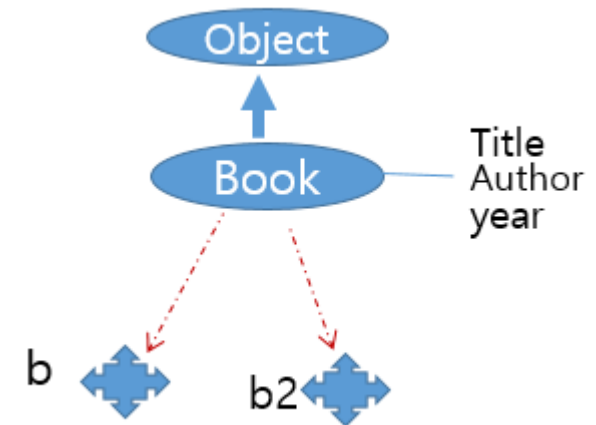
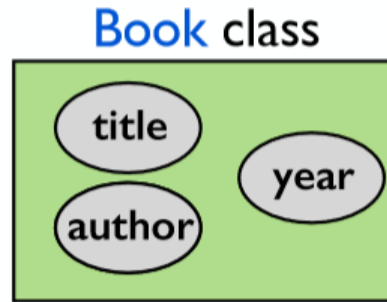
```
book2["year"] = 1880
```

# Motivating Example of OO: Book Record

[4/5]

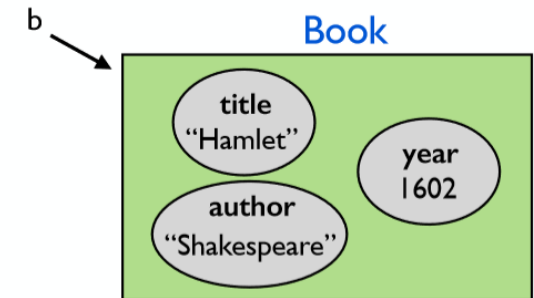
- Option 4: Defining a data type (Class) called Book

```
class Book(object):  
  
    def __init__(self):  
        self.title = None  
        self.author = None  
        self.year = None
```



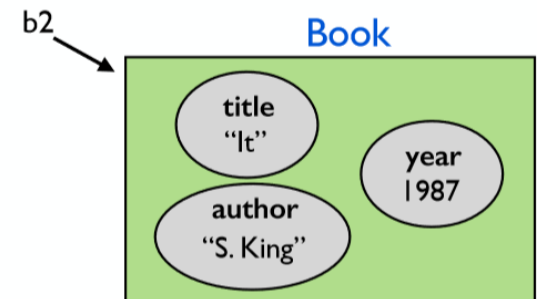
```
b = Book()  
b.title = "Hamlet"  
b.author = "Shakespeare"  
b.year = 1602
```

b refers to an **object**  
of type **Book**.



```
b2 = Book()  
b2.title = "It"  
b2.author = "S. King"  
b2.year = 1987
```

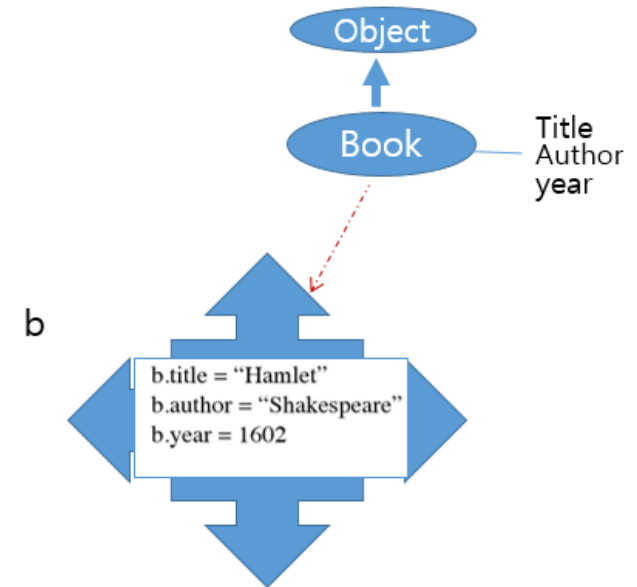
b2 refers to another **object**  
of type **Book**.



## Initializing fields at object creation

```
class Book(object):  
    def __init__(self, title, author, year):  
        self.title = title  
        self.author = author  
        self.year = year
```

```
b = Book("Hamlet", "Shakespeare", 1602)
```



# OOP Motivational Example: Calculator [1/4]



만약 한 프로그램에서 **2개의 계산기**가 필요하다면 어떻게 해야 할까?

각각의 계산기는 각각의 결과값을 유지해야 하므로 adder function 1개로는 결과값을 따로 유지할수 없다

## Using Functions Only

```
result = 0
```

```
def adder(num):  
    global result  
    result += num  
    return result
```

```
print(adder(3))  
print(adder(4))
```

3

7

## Using Functions Only

```
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

# OOP Motivational Example: Calculator [2/4]

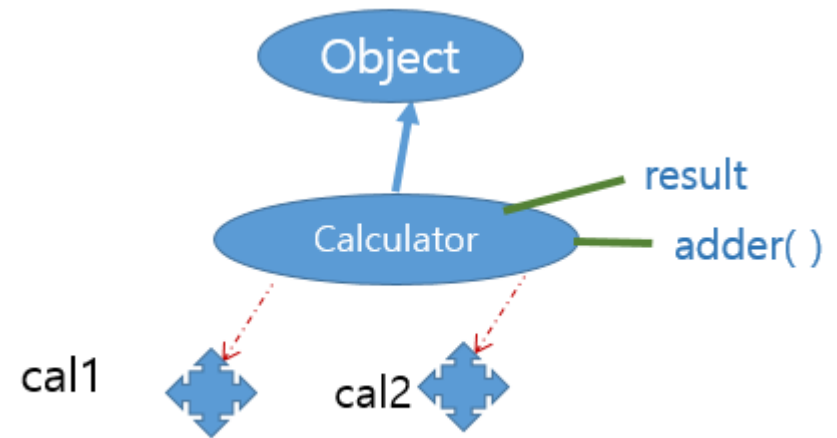
만약 10개의 계산기가 필요하다면 어떻게 해야 할까?

각각의 계산기는 각각의 결과값을 유지해야 하므로 10개의 adder function을 각각 만들어야 하나?

## Using Classes and Objects

```
class Calculator:  
    def __init__(self):  
        self.result = 0  
  
    def adder(self, num):  
        self.result += num  
        return self.result
```

```
cal1 = Calculator()  
cal2 = Calculator()
```



<code>print(cal1.adder(3))</code>	3
<code>print(cal1.adder(4))</code>	7
<code>print(cal2.adder(3))</code>	3
<code>print(cal2.adder(7))</code>	10

# OOP Motivational Example: Calculator [3/4]

## Using Classes and Objects

```
>>> class FourCal:
...     def __init__(self, first, second):
...         self.first = first
...         self.second = second
...     def sum(self):
...         result = self.first + self.second
...         return result
...     def mul(self):
...         result = self.first * self.second
...         return result
...     def sub(self):
...         result = self.first - self.second
...         return result
...     def div(self):
...         result = self.first / self.second
...         return result
```

```
>>> a = FourCal(4,2)
```

```
>>> b = FourCal(3,7)
```

```
>>> a.sum()
```

```
6
```

```
>>> a.mul()
```

```
8
```

```
>>> a.sub()
```

```
2
```

```
>>> a.div()
```

```
2
```

```
>>> b.sum()
```

```
10
```

```
>>> b.mul()
```

```
21
```

```
>>> b.sub()
```

```
-4
```

```
>>> b.div()
```

```
0
```

# OOP Motivational Example: Calculator [4/4]

Using Classes and  
Objects

`__init__()` 를 안쓰고 `setdata()` function을 써도 가능!

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...     def sum(self):
...         result = self.first + self.second
...         return result
...     def mul(self):
...         result = self.first * self.second
...         return result
...     def sub(self):
...         result = self.first - self.second
...         return result
...     def div(self):
...         result = self.first / self.second
...         return result
```

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 7)
>>> a.sum()
6
>>> a.mul()
8
>>> a.sub()
2
>>> a.div()
2
>>> b.sum()
10
>>> b.mul()
21
>>> b.sub()
-4
>>> b.div()
0
```

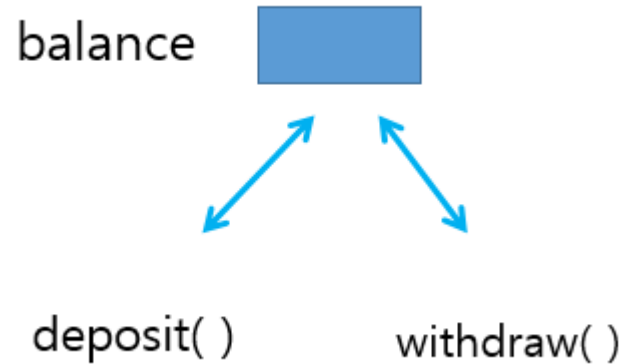
# OOP Motivational Example: Bank Account [1/4]

## Using Functions Only

```
balance = 0

def deposit(amount):
    global balance
    balance += amount
    return balance

def withdraw(amount):
    global balance
    balance -= amount
    return balance
```



**\*\* 여러 개의 account를 만들려고 한다면?**



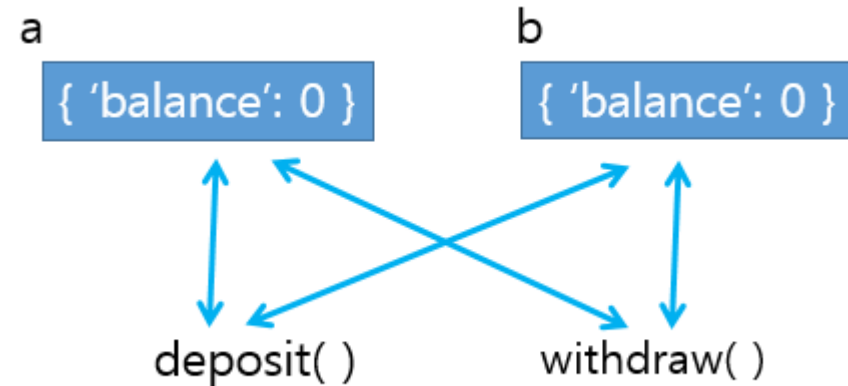
# OOP Motivational Example: Bank Account [2/4]

## What If Functions with Dictionary

```
def make_account():  
    return {'balance': 0}  
  
def deposit(account, amount):  
    account['balance'] += amount  
    return account['balance']  
  
def withdraw(account, amount):  
    account['balance'] -= amount  
    return account['balance']
```

With this it is possible to work with multiple accounts

```
>>> a = make_account()  
>>> b = make_account()  
>>> deposit(a, 100)  
100  
>>> deposit(b, 50)  
50  
>>> withdraw(b, 10)  
40  
>>> withdraw(a, 10)
```



**\*\* 여러 개의 account 생성은 해결되어지만**

**\*\* 다른종류의 account (예, minimum balance account를 만들고 싶다면?)**

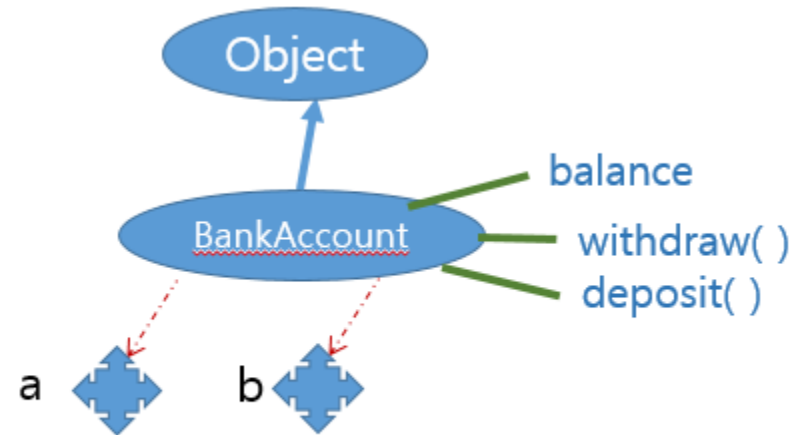
# OOP Motivational Example: Bank Account [3/4]

## Using OOP (Classes and Objects)

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

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

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



**\*\* BankAccount**라는 user-defined data type을 만들었으므로 필요할때마다 account instance를 만들어서 사용하면 됨!

```
>>> a = BankAccount()
>>> b = BankAccount()
>>> a.deposit(100)
100
>>> b.deposit(50)
50
>>> b.withdraw(10)
40
>>> a.withdraw(10)
90
```

# OOP Motivational Example: Bank Account [4/4]

## Using Classes and Objects

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

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

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



**\*\* BankAccount라는 class를 만들었으므로 필요할때마다 subclass를 만들어 사용하면 됨!**

Subclass에서 `__init__()`을 자체적으로 만들면 superclass의 instance variable을 받을수 없다.

Superclass의 instance variable도 받고, 자체적으로 instance variable를 추가하고자 하면 superclass의 `__init__()`를 구체적으로 call해야 한다

```
class MinimumBalanceAccount(BankAccount):
    def __init__(self, minimum_balance):
        BankAccount.__init__(self)  ← super().__init__(self)
        self.minimum_balance = minimum_balance

    def withdraw(self, amount):
        if self.balance - amount < self.minimum_balance:
            print('Sorry, minimum balance must be maintained.')
        else:
            BankAccount.withdraw(self, amount)
```

다른방법: `super(MinimumBalanceAccount, self).__init__()`

# Practice of Class Hierarchy [1/6]

- 박씨집안 (HousePark) 사람들은 성이 박(Park) 이고 김씨집안 (HouseKim) 사람들은 성이 김(Kim) 이다
- 박씨집안 (HousePark) 과 김씨집안 (HouseKim) 은 여행을 좋아하는 공통점이 있다.
  - 박씨집안 사람들은 여행을 갈때에 어디로 여행간다는 내용을 공표한다
  - 김씨집안 사람들은 여행을 갈때에 어디로 얼마동안 여행간다는 내용을 공표한다
- 어느날 두집안의 박응용군과 김줄리엣양은 사귀다가 결혼을 했다
- 그러나 나중에 성격차로 싸움하고는 이혼을 했다

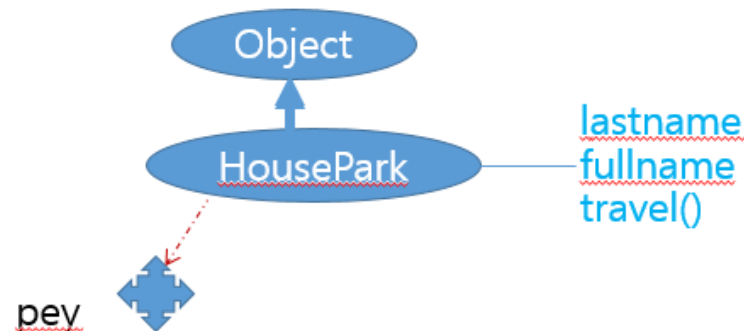
# Practice of Class Hierarchy [2/6]

```
>>> class HousePark:
...     lastname = "박"
...     def __init__(self, name):
...         self.fullname = self.lastname + name
...     def travel(self, where):
...         print("%s, %s여행을 가다." % (self.fullname, where))
```

Class variable

```
>>> pey = HousePark()
TypeError: __init__() takes exactly 2 arguments (1 given)
```

```
>>> pey = HousePark("응용")
>>> pey.travel("태국")
박응용, 태국여행을 가다.
```

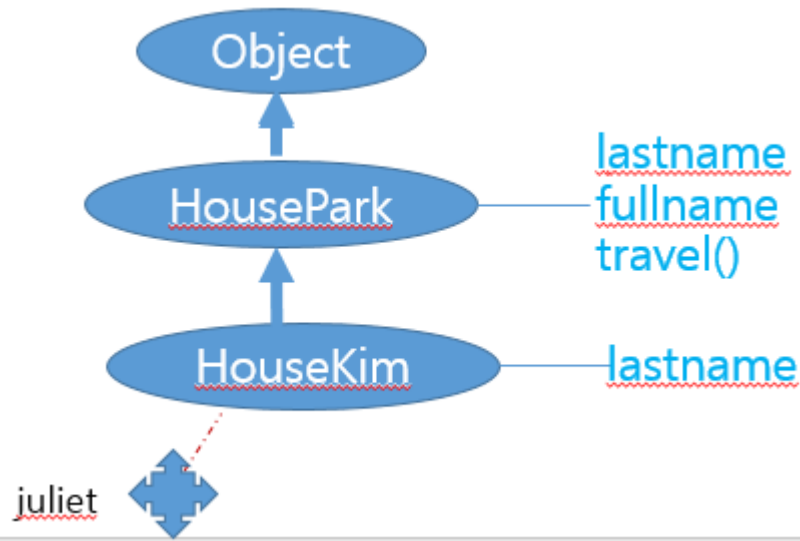


# Practice of Class Hierarchy [3/6]

```
>>> class HousePark:
...     lastname = "박"
...     def __init__(self, name):
...         self.fullname = self.lastname + name
...     def travel(self, where):
...         print("%s, %s여행을 가다." % (self.fullname, where))
```

```
>>> class HouseKim(HousePark):
...     lastname = "김"
```

```
>>> juliet = HouseKim("줄리엣")
>>> juliet.travel("독도")
김줄리엣, 독도여행을 가다.
```

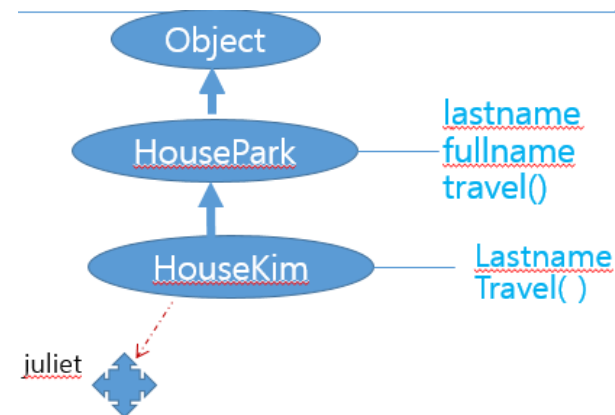


# Practice of Class Hierarchy [4/6]

```
>>> class HousePark:
...     lastname = "박"
...     def __init__(self, name):
...         self.fullname = self.lastname + name
...     def travel(self, where):
...         print("%s, %s여행을 가다." % (self.fullname, where))
```

```
>>> class HouseKim(HousePark):
...     lastname = "김"
...     def travel(self, where, day):
...         print("%s, %s여행 %d일 가네." % (self.fullname, where, day))
```

```
>>> juliet = HouseKim("줄리엣")
>>> juliet.travel("독도", 3)
김줄리엣, 독도여행 3일 가네.
```



# Practice of Class Hierarchy [5/6]

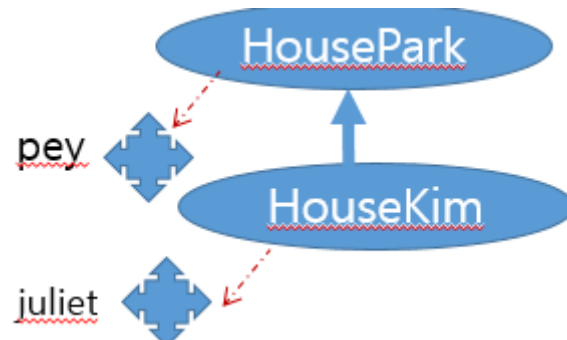
```
class HousePark:
    lastname = "박"
    def __init__(self, name):
        self.fullname = self.lastname + name
    def travel(self, where):
        print("%s, %s여행을 가다." % (self.fullname, where))
    def love(self, other):
        print("%s, %s 사랑에 빠졌네" % (self.fullname, other.fullname))
    def __add__(self, other):
        print("%s, %s 결혼했네" % (self.fullname, other.fullname))
```

```
class HouseKim(HousePark):
    lastname = "김"
    def travel(self, where, day):
        print("%s, %s여행 %d일 가네." % (self.fullname, where, day))
```

User-defined class의 instance에  
+를 쓰려면 class내부에  
\_\_add\_\_()가 define 되어있어야 함

```
pey = HousePark("응용")
juliet = HouseKim("줄리엣")
pey.love(juliet)
pey + juliet
```

박응용, 김줄리엣 사랑에 빠졌네  
박응용, 김줄리엣 결혼했네



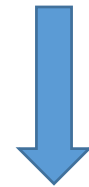


# Practice of Class Hierarchy [6/6]

```
class HousePark:
    lastname = "박"
    def __init__(self, name):
        self.fullname = self.lastname + name
    def travel(self, where):
        print("%s, %s여행을 가다." % (self.fullname, where))
    def love(self, other):
        print("%s, %s 사랑에 빠졌네" % (self.fullname, other.fullname))
    def fight(self, other):
        print("%s, %s 싸우네" % (self.fullname, other.fullname))
    def __add__(self, other):
        print("%s, %s 결혼했네" % (self.fullname, other.fullname))
    def __sub__(self, other):
        print("%s, %s 이혼했네" % (self.fullname, other.fullname))

class HouseKim(HousePark):
    lastname = "김"
    def travel(self, where, day):
        print("%s, %s여행 %d일 가네." % (self.fullname, where, day))
```

```
pey = HousePark("응용")
juliet = HouseKim("줄리엣")
pey.travel("부산")
juliet.travel("부산", 3)
pey.love(juliet)
pey + juliet
pey.fight(juliet)
pey - juliet
```



박응용 부산여행을 가다.  
김줄리엣 부산여행 3일 가네.  
박응용, 김줄리엣 사랑에 빠졌네  
박응용, 김줄리엣 결혼했네  
박응용, 김줄리엣 싸우네  
박응용, 김줄리엣 이혼했네

- 를 instance에 쓰려면  
\_\_sub\_\_()가 define 되어야 함