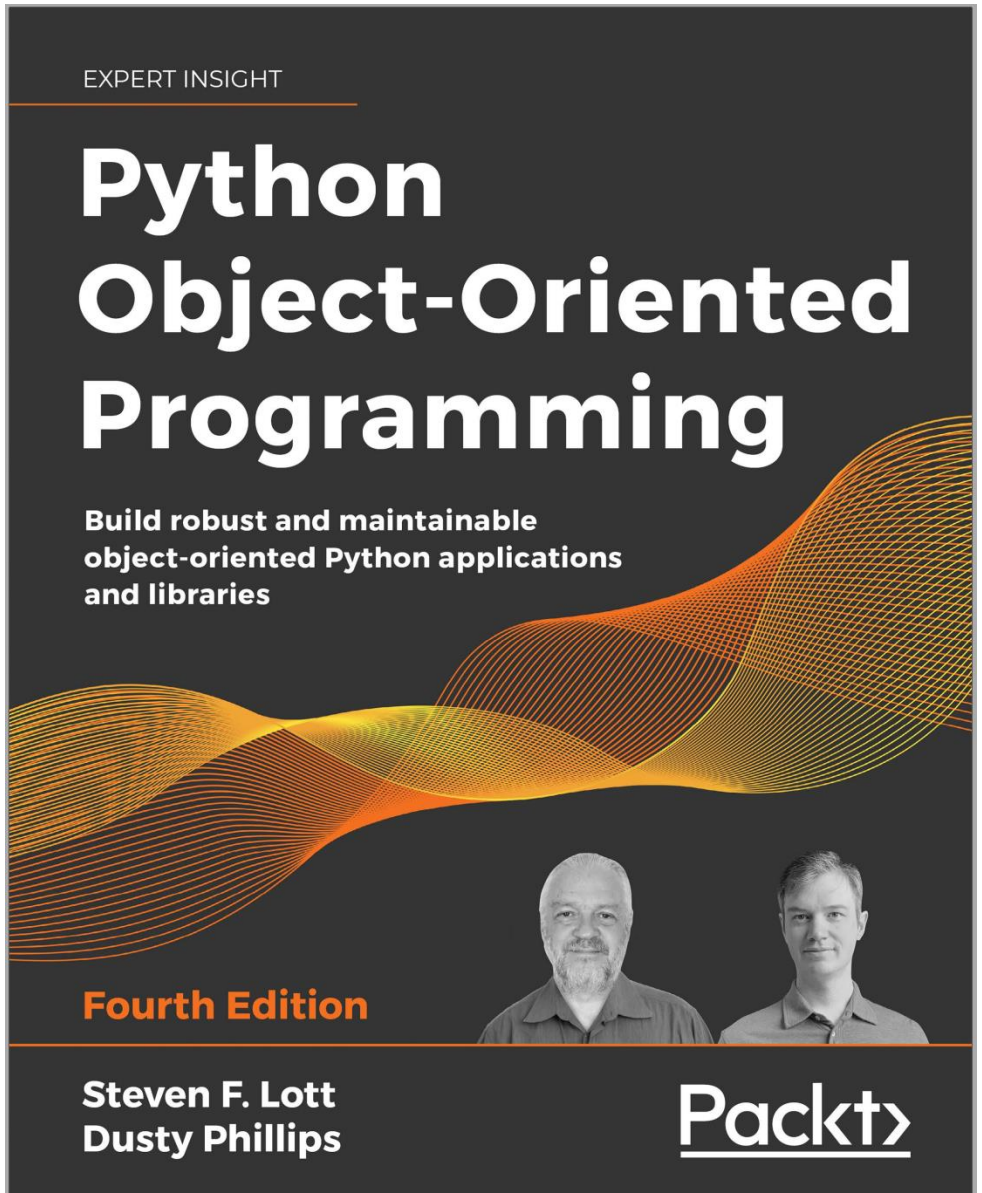
AI VIETNAM
All-in-One Course

Object-Oriented Programming

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2022

Reference Books



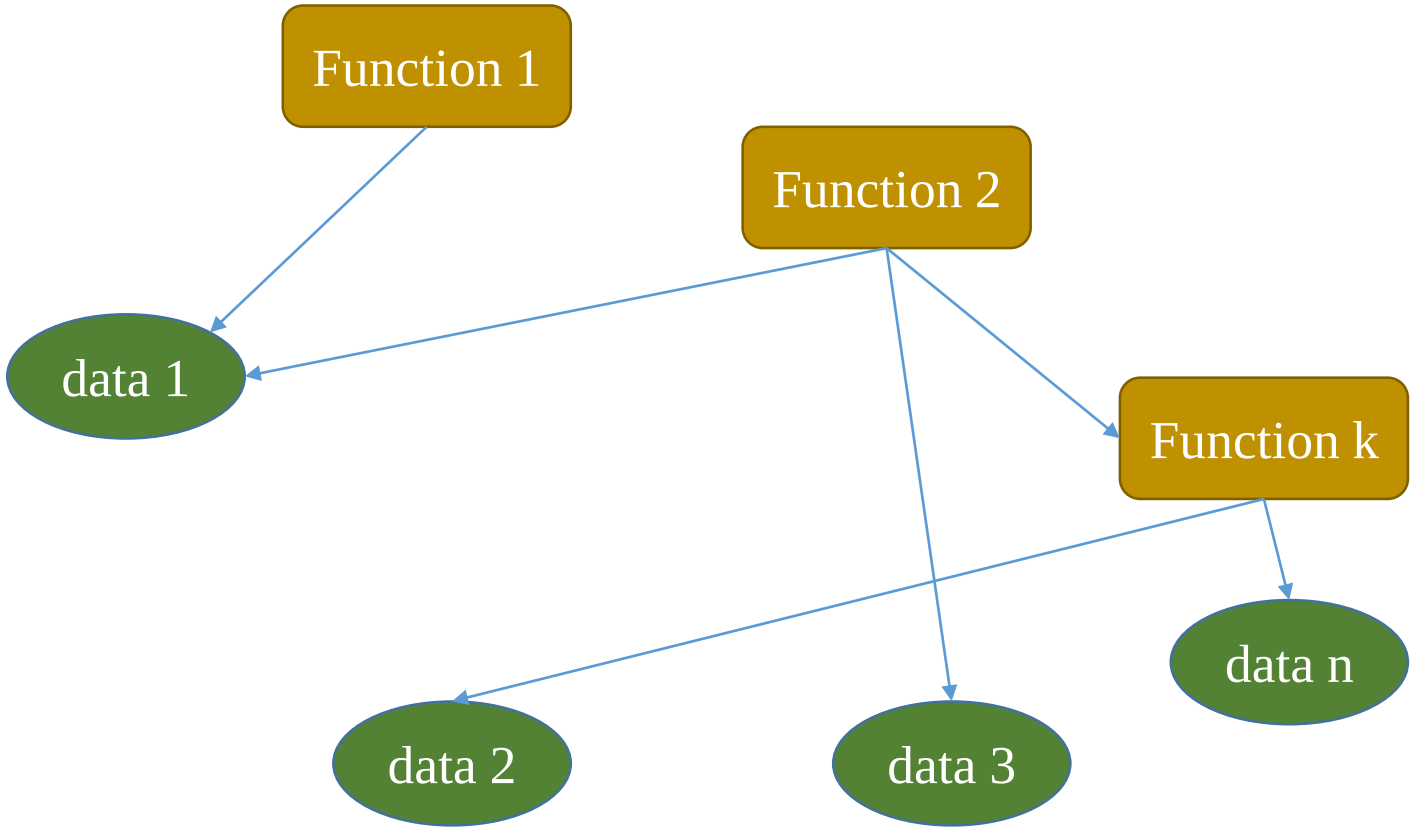
Outline

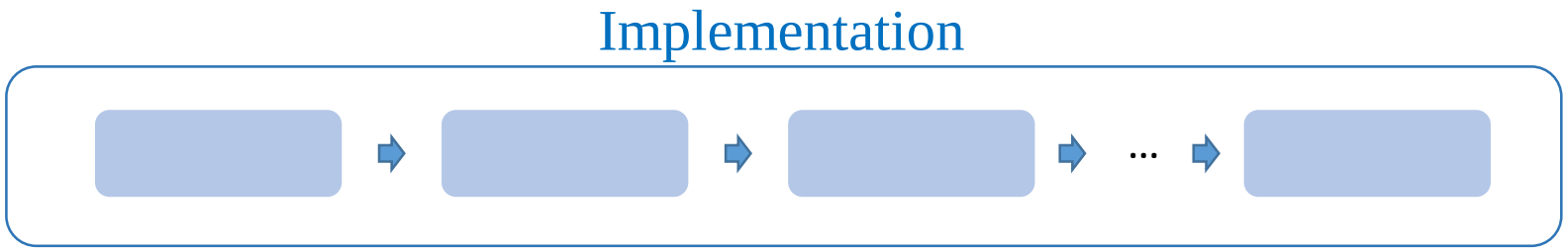
- Introduction
- Classes and Objects
- Iterator and Class Data Type
- Inheritance
- Exercises

OOP Introduction

Procedural programming

- Top-down approach
- Security
- Maintenance
- Reusability
- Implementation





OOP Introduction

OOP programming

- Bottom-up approach
- Security
- Maintenance
- Reusability
- Implementation

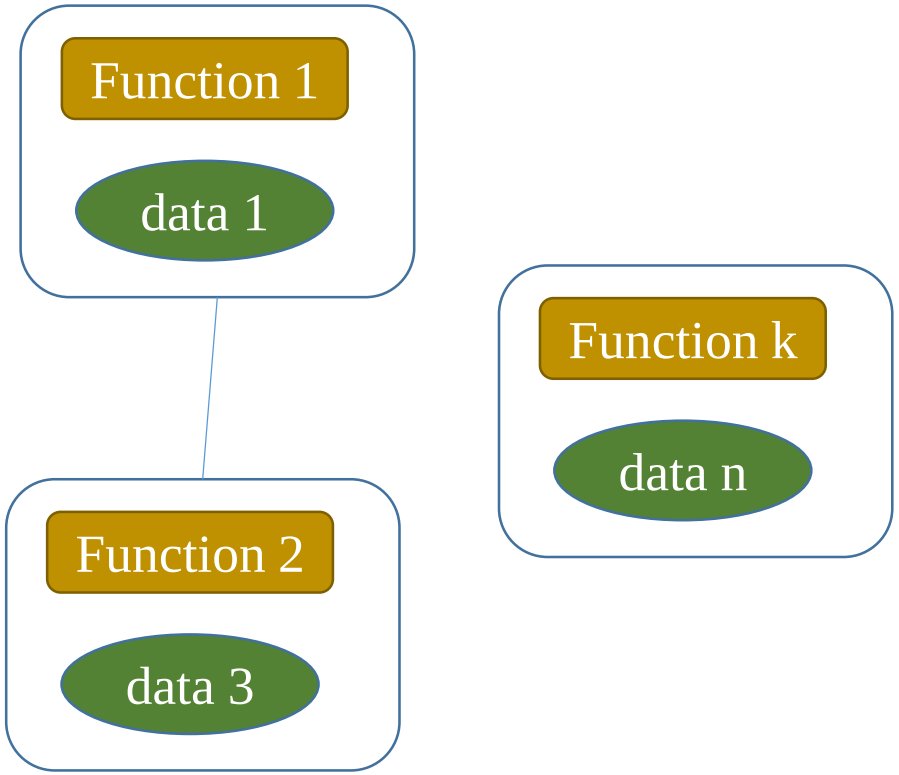
Access modifiers

Inheritance

Objects/Classes

Encapsulation

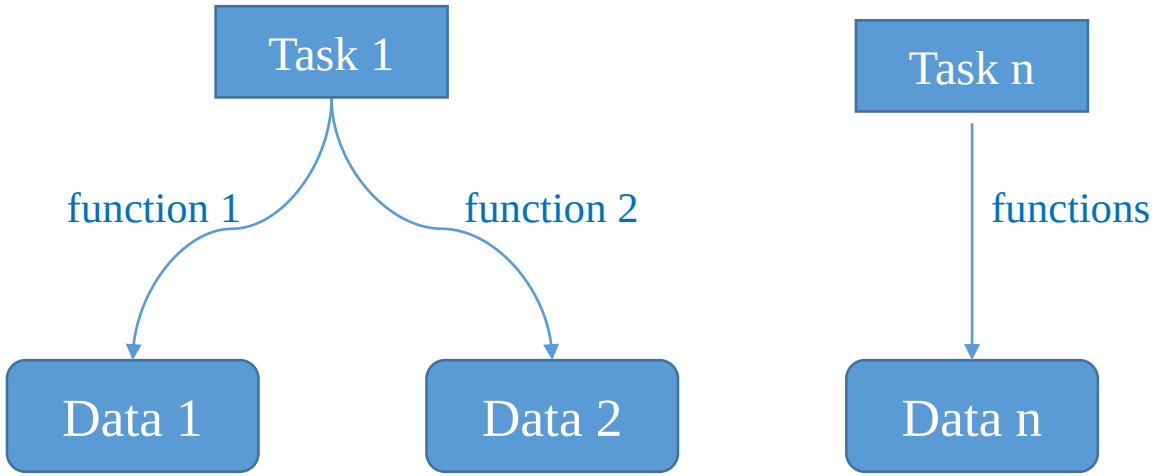
Polymorphism



Introduction

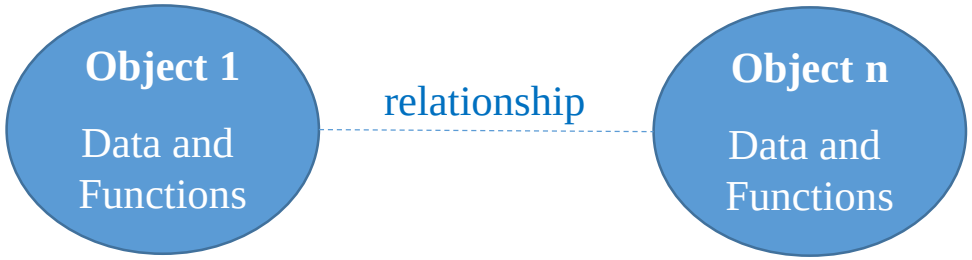
❖ What is OOP?

Writing functions that perform operations on the data



Procedural programming

Creating objects that contain both data and functions



Object-oriented programming

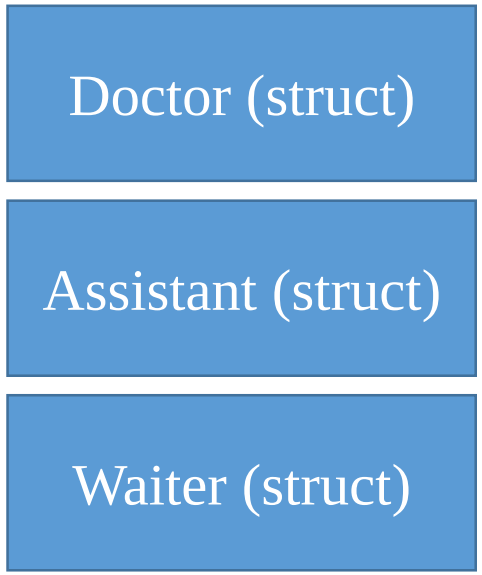
Introduction

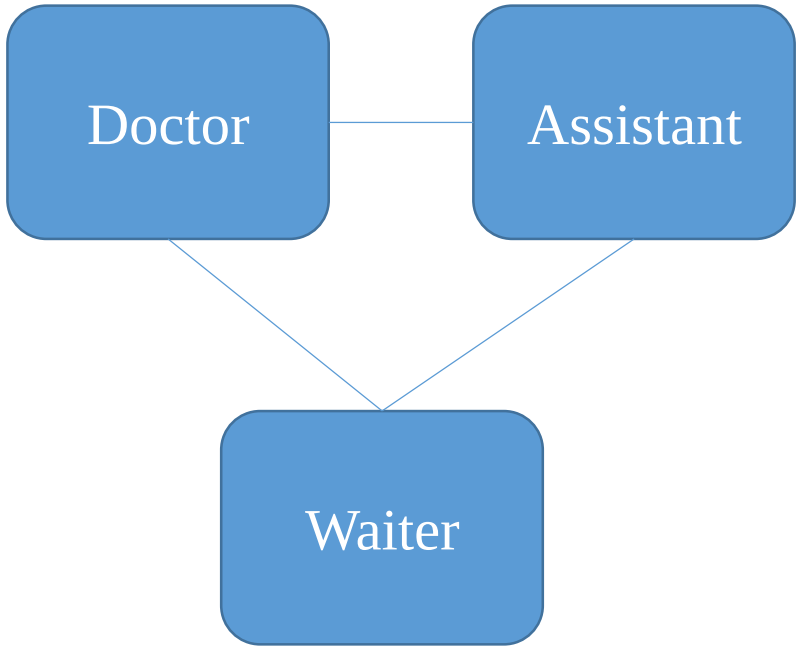
❖ OOP advantages

A clear structure

Easier to maintain, modify and debug

Reusable





Introduction

❖ Classes and Objects

A class is a template for objects, and an object is an instance of a class.

Fruit

Strawberry
Apple
Banana

Introduction

❖ Classes and Objects

A class is a template for objects, and an object is an instance of a class.

Animal

Cat
Deer
Tiger

Introduction

❖ Classes and Objects

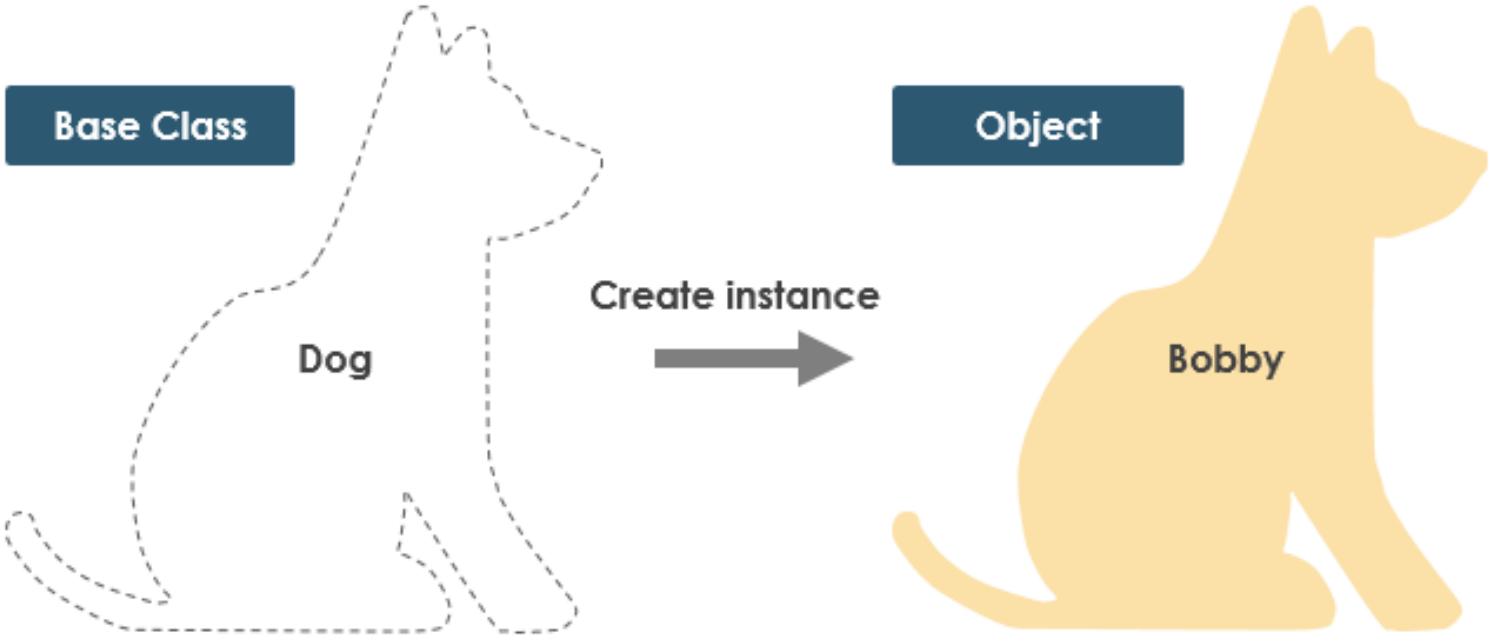
A class is a template for objects, and an object is an instance of a class.

Cat

Japanese Bobtail
Scottish Fold
Calico

Introduction

❖ Classes and Objects



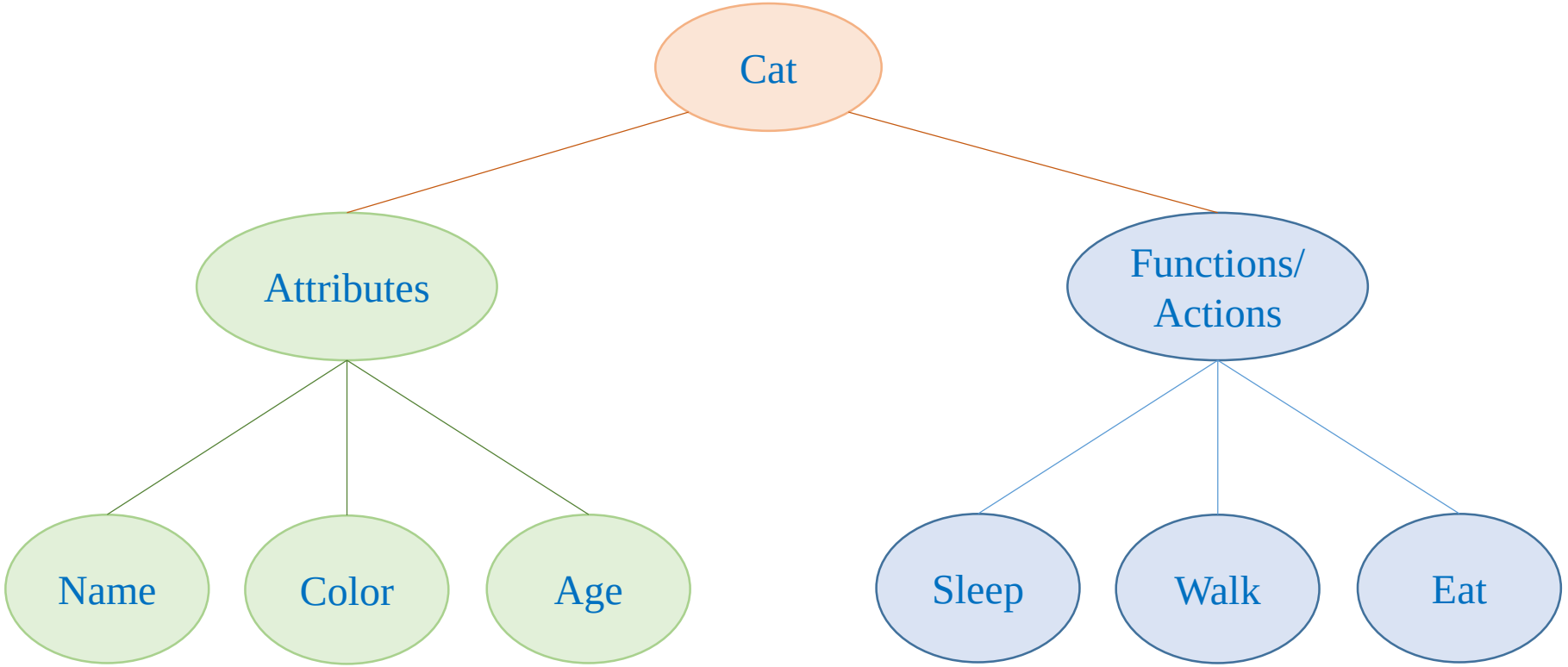
Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Classes and Objects

❖ Abstract view



Class Diagram

Cat
name color age
sleep walk eat

Class Diagram

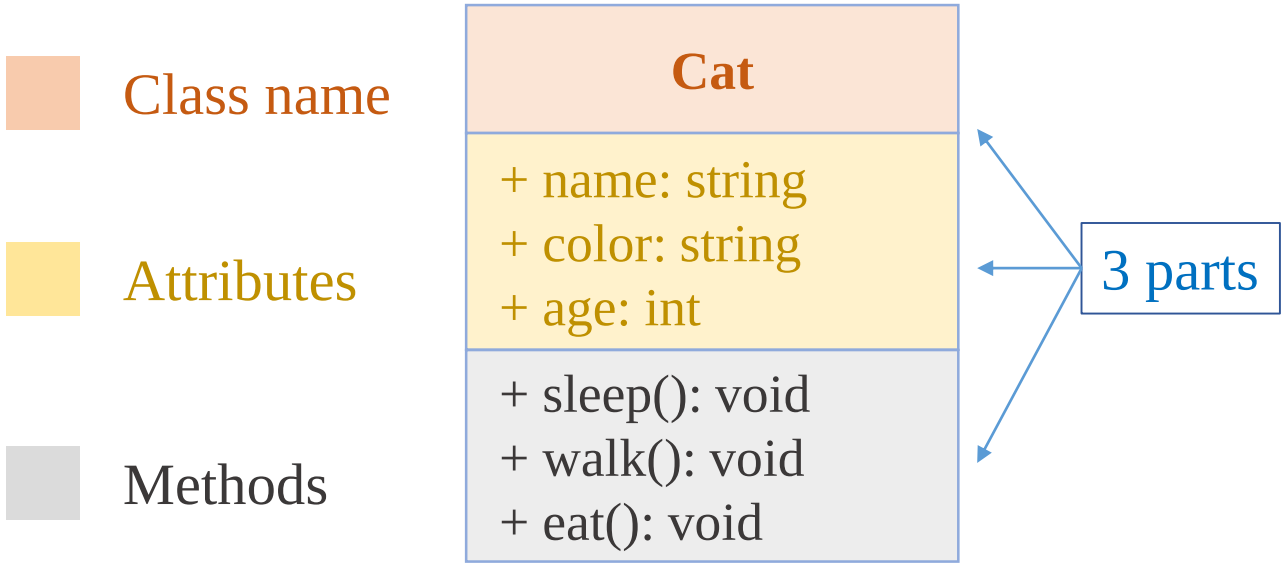
❖ Describe the structure of a system

Classes
their attributes
operations (methods)
relationships among objects

Access modifiers
- private
+ public

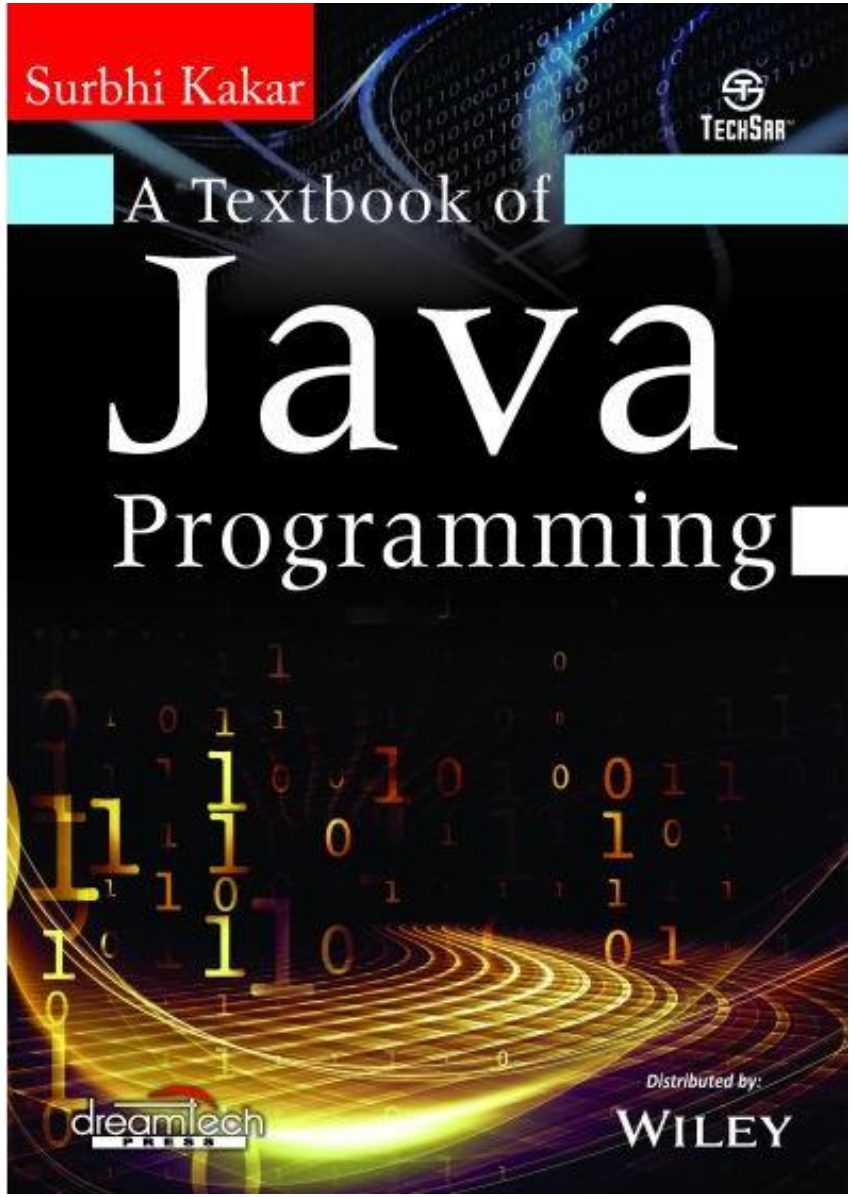
A cat includes a name, a color, and an age. The daily activities of the cat consists of sleeping, walking, and eating.

Draw a class diagram for the above description. All the attributes and methods are publicly accessed.



OOP Introduction

Detect Class and its Members




```xml
<?xml version="1.0" encoding="UTF-8"?>
<book id="java101">
  <author>Surbhi Kakar</author>
  <title>Java Programming</title>
  <genre>Computer</genre>
  <price>30.0</price>
  <publish_date>2010-08-01</publish_date>
  <publisher>Dream Tech Press</publisher>
  <description>A description here.</description>
</book>
```

Book

+ author: string
+ title: string
+ genre: string
+ price: double
+ date: string
+ publisher: string
+ description: string

...

Objects and Classes

❖ Create a function in a class

The __init__() function is called automatically every time the class is being used to create a new object.

The self parameter is a reference to the current instance of the class.

__call__() function: instances behave like functions and can be called like a functions.

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def sum(self):
        return self.x + self.y

    def __call__(self):
        return self.x*self.y
```

```python
point = Point(4, 5)
print(point.sum())
print(point())
```

```
9
20
```

Objects and Classes

❖ Create a class

```
# create a class
class Point:
    x = 0
    y = 0
```

```
point = Point()
print(type(point))
print(point)
print(point.x)
print(point.y)
```

```
<class '__main__.Point'>
<__main__.Point object at 0x00000196835AD860>
0
0
```

Using init() function

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

```
point = Point(4, 5)
print(point.x)
print(point.y)
```

```
4
5
```

Change property values

```
point = Point()
print(point.x)
print(point.y)

point.x = 5
point.y = 7
print(point.x)
print(point.y)
```

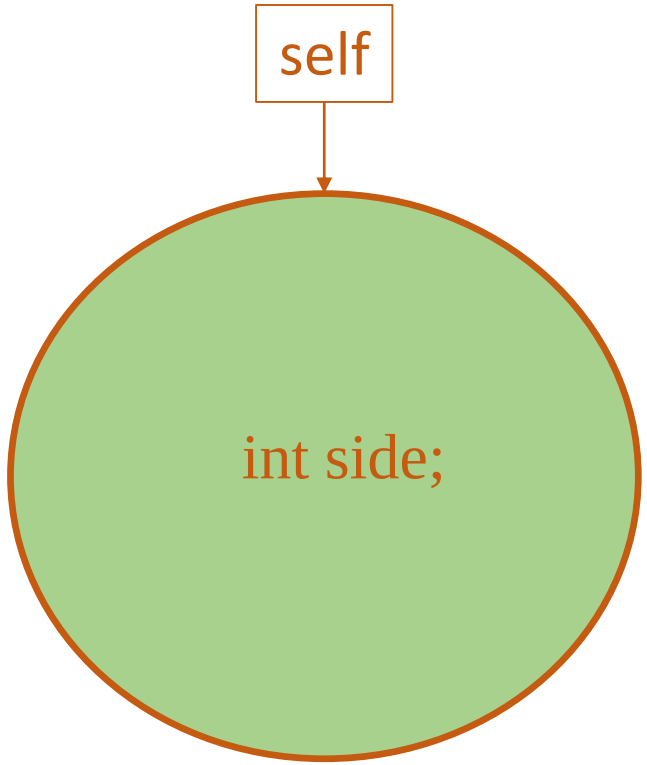
```
0
0
5
7
```

self Keyword

Must be the first argument of methods

Used to create and access data members

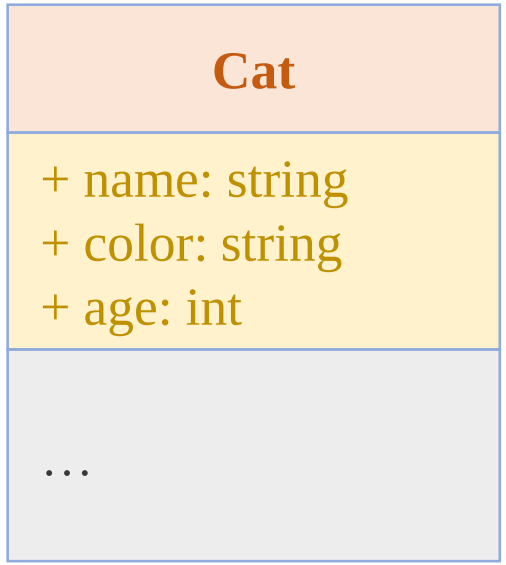
int side;




```python
class Square:
    def __init__(self, side):
        self.side = side

    def computeArea(self):
        return self.side*self.side

# test sample: side=5 -> 25
square = Square(5)
area = square.computeArea()
print(f'Square area is {area}')
```

```
Square area is 25
```

Classes and Objects




```python
class Cat:
    def __init__(self, name, color, age):
        self.name = name
        self.color = color
        self.age = age

# test
cat = Cat('Calico', 'Black, white, and brown', 2)
print(cat.name)
print(cat.color)
print(cat.age)
```

```
Calico
Black, white, and brown
2
```

We have a new data type

cat = Cat('Calico', 'BW', 2)

variable — create an object

Naming conventions

For class names

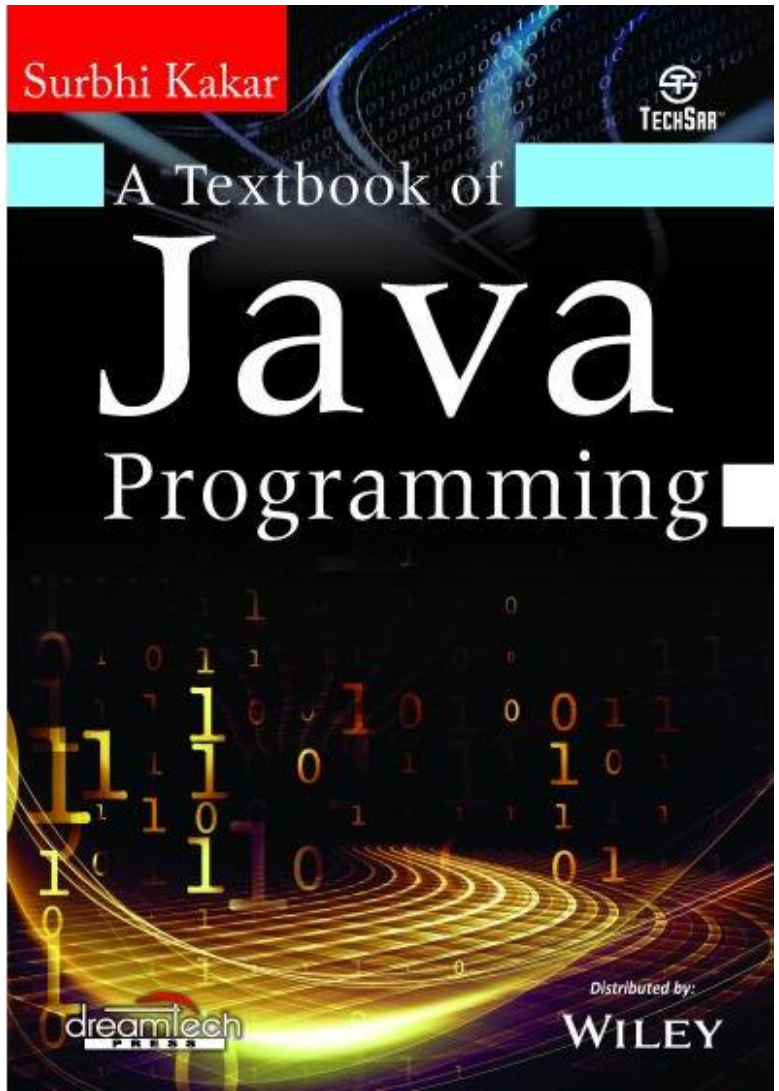
- Including words concatenated
- Each word starts with upper case

For attribute names

- Including words concatenated
- Each word starts with upper case except the first word

Exercise

Implementation




```xml
<?xml version="1.0" encoding="UTF-8"?>
<book id="java101">
  <author>Surbhi Kakar</author>
  <title>Java Programming</title>
  <genre>Computer</genre>
  <price>30.0</price>
  <publish_date>2010-08-01</publish_date>
  <publisher>Dream Tech Press</publisher>
  <description>A description here.</description>
</book>
```

Book
+ author: string + title: string + genre: string + price: double + date: string + publisher: string + description: string
…

Implement the Book class in Python

Classes and Objects

Back to the Cat example

Cat
+ name: string …
…

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

# test
cat = Cat('Calico')
print(cat.name)

cat.name = 'Japanese Bobtail'
print(cat.name)
```

```
Calico
Japanese Bobtail
```

Classes and Objects

Problem and Solution:

Step 1 – implement getter and setter functions

Cat
+ name: string …
+ getName(): string + setName(string): void …

Access modifiers
- private
+ public

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

    def getName(self):
        return self.name

    def setName(self, name):
        self.name = name

# test
cat = Cat('Calico')
print(cat.getName())

cat.setName('Japanese Bobtail')
print(cat.getName())
```

Classes and Objects

Solution: Step 2 – Using private for attributes

Cat
- name: string …
+ getName(): string + setName(string): void …

Access modifiers
- private
+ public

```
print(cat.__name)
```

```
---------------------------------------------------------
AttributeError                            Traceback (mo
Input In [3], in <cell line: 1>()
----> 1 print(cat.__name)

AttributeError: 'Cat' object has no attribute '__name'
```

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

    def getName(self):
        return self.__name

    def setName(self, name):
        self.__name = name

# test
cat = Cat('Calico')
print(cat.getName())

cat.setName('Japanese Bobtail')
print(cat.getName())
```

```
Calico
Japanese Bobtail
```

Classes and Objects

Takeaways

Use the private access modifiers for typical attributes

Create getter and setter functions to access protected attributes

Use the public access modifiers for the getter and setter functions

Access modifiers
- private
+ public

A cat includes a name, a color, and an age.

Cat
- name: string - color: string - age: int
+ getName(): string + setName(string): void + getColor(): string + setColor(string): void + getAge(): int + setAge(int): void

Outline

- Introduction
- Classes and Objects
- Iterator and Class Data Type
- Inheritance
- Exercises

Iterator

1. Create iterator using iter()
2. Get value using next()

```
# iterator

a_list = [1, 2, 3]

# get an iterator
a_iter = iter(a_list)

print(next(a_iter))
print(next(a_iter))
print(next(a_iter))
```

```
1
2
3
```

```
# iterator

a_list = [1, 2, 3]

# get an iterator
a_iter = iter(a_list)

print(next(a_iter))
print(next(a_iter))
print(next(a_iter))
print(next(a_iter))
```

```
1
2
3

---------------------------------------------------------------------------
StopIteration                                             Traceback
<ipython-input-1-251788936f93> in <module>
      9 print(next(a_iter))
     10 print(next(a_iter))
---> 11 print(next(a_iter))

StopIteration:
```

Iterator

```python
class AdditionValue:
    def __init__(self, value, maxValue=0):
        self.value = value
        self.maxValue = maxValue

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        if self.n <= self.maxValue:
            result = self.n + self.value
            self.n += 1

            return result
        else:
            raise StopIteration
```

```python
# create an object
numbers = AdditionValue(5, 2)

for number in numbers:
    print(number)
```

```
5
6
7
```

__iter__(): set initialization for the iterator

__next__(): get a value of the iterator
update the state of the iterator

Using raise StopIteration

AI VIETNAM
All-in-One Course

Iterator

```python
class AdditionValue:
    def __init__(self, value, maxValue=0):
        self.value = value
        self.maxValue = maxValue

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        if self.n <= self.maxValue:
            result = self.n + self.value
            self.n += 1

            return result
        else:
            raise StopIteration
```

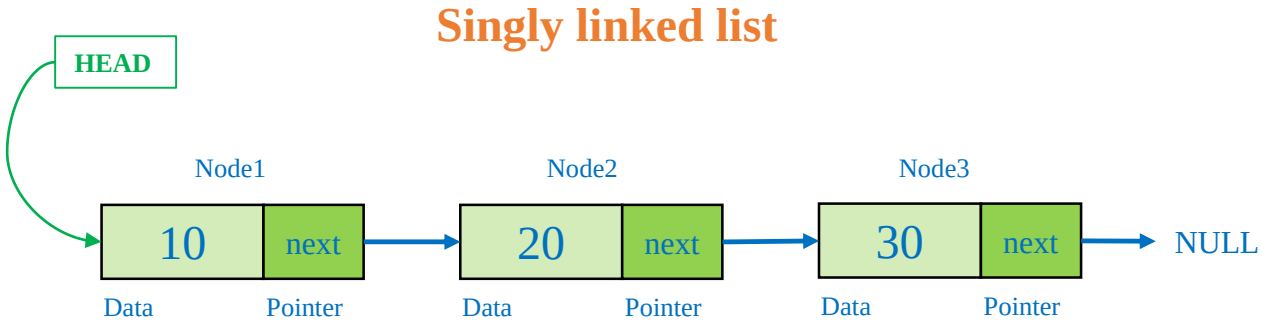
```python
# create an object
numbers = AdditionValue(5, 2)

# create an iterator
a_iter = iter(numbers)

# get iterator elements
print(next(a_iter))
print(next(a_iter))
print(next(a_iter))
print(next(a_iter))
```

```
5
6
7
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Input In [10], in <cell line: 11>()
      9 print(next(a_iter))
     10 print(next(a_iter))
---> 11 print(next(a_iter))

Input In [7], in AdditionValue.__next__(self)
     15     return result
     16 else:
---> 17     raise StopIteration

StopIteration:
```

Iterator

❖ List




```python
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None
```

Iterator

❖ List

```python
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def add(self, data):
        newNode = Node(data)

        # Update the new node
        newNode.next = self.head
        self.head = newNode
```

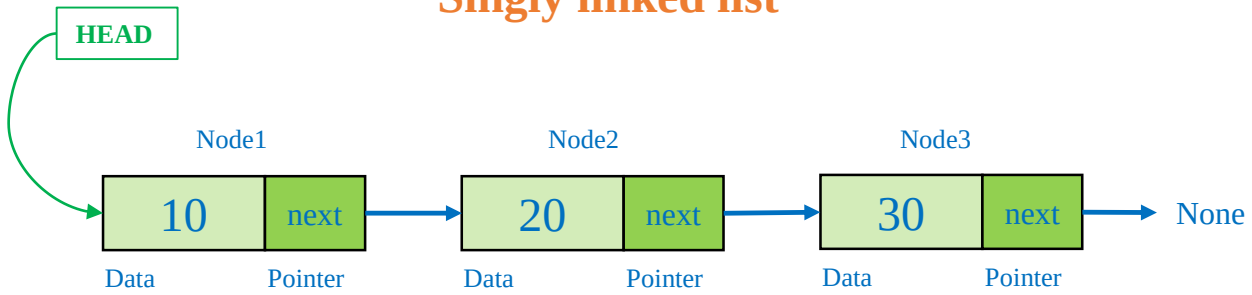
```python
# create a linked list
linkedList = SinglyLinkedList()

# create three nodes
n1 = Node('Mon')
n2 = Node('Tue')
n3 = Node('Wed')

# set head
linkedList.head = n1

# Link nodes
n1.next = n2
n2.next = n3
```

AI VIETNAM
All-in-One Course

Iterator

❖ List

Singly linked list




```python
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None
```

```python
for node in linkedList:
    print(node.data)
```

```
Wed
Tue
Mon
```

```python
class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def add(self, data):
        newNode = Node(data)
        newNode.next = self.head

        self.head = newNode

    def __iter__(self):
        self.current = self.head
        return self

    def __next__(self):
        if self.current is not None:
            node = self.current
            self.current = self.current.next

            return node
        else:
            raise StopIteration
```

Lists and Classes

Sort a list of numbers

```
aList = [1, 5, 3, 7, 4, 9]
aList.sort()
print(aList)
```

```
[1, 3, 4, 5, 7, 9]
```

```
aList.sort()
for square in aList:
    square.describe()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [5], in <cell line: 1>()
----> 1 aList.sort()
      2 for square in aList:
      3     square.describe()

TypeError: '<' not supported between instances of 'Square' and 'Square'
```

```
class Square:
    def __init__(self, side):
        self.side = side

    def computeArea(self):
        return self.side*self.side

    def describe(self):
        print(f'Side is {self.side}')
```

```
s1 = Square(3)
s2 = Square(8)
s3 = Square(1)
s4 = Square(6)
s5 = Square(5)

aList = [s1, s2, s3, s4, s5]
for square in aList:
    square.describe()
```

```
Side is 3
Side is 8
Side is 1
Side is 6
Side is 5
```

Lists and Classes

Sort a list of squares

```python
class Square:
    def __init__(self, side):
        self.side = side

    def computeArea(self):
        return self.side*self.side

    def describe(self):
        print(f'Side is {self.side}')
```

```python
s1 = Square(3)
s2 = Square(8)
s3 = Square(1)
s4 = Square(6)
s5 = Square(5)

aList = [s1, s2, s3, s4, s5]
aList.sort(key=lambda x: x.side)
for square in aList:
    square.describe()
```

```
Side is 1
Side is 3
Side is 5
Side is 6
Side is 8
```

Lists and Classes

Sort a list of squares

```python
class Square:
    def __init__(self, side):
        self.side = side

    def computeArea(self):
        return self.side*self.side

    def describe(self):
        print(f'Side is {self.side}')
```

```python
s1 = Square(3)
s2 = Square(8)
s3 = Square(1)
s4 = Square(6)
s5 = Square(5)

aList = [s1, s2, s3, s4, s5]
```

```python
def compare(x):
    return x.side

aList.sort(key=compare)
for square in aList:
    square.describe()
```

```
Side is 1
Side is 3
Side is 5
Side is 6
Side is 8
```

Class Data Type

Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

Write a function to check if two people have the same name.

Write a function to check if two people have the same date of birth.

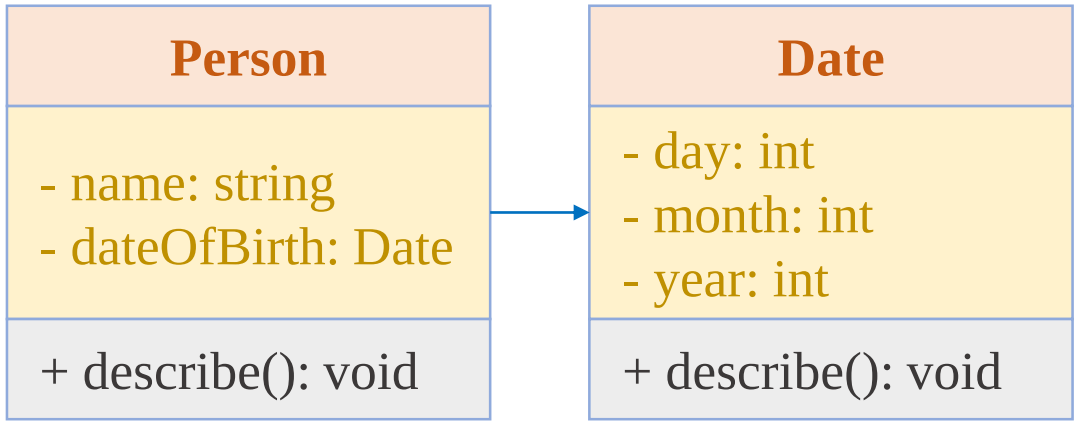
Draw a class diagram and implement in Python

Class Data Type

Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

Draw a class diagram and implement in Python

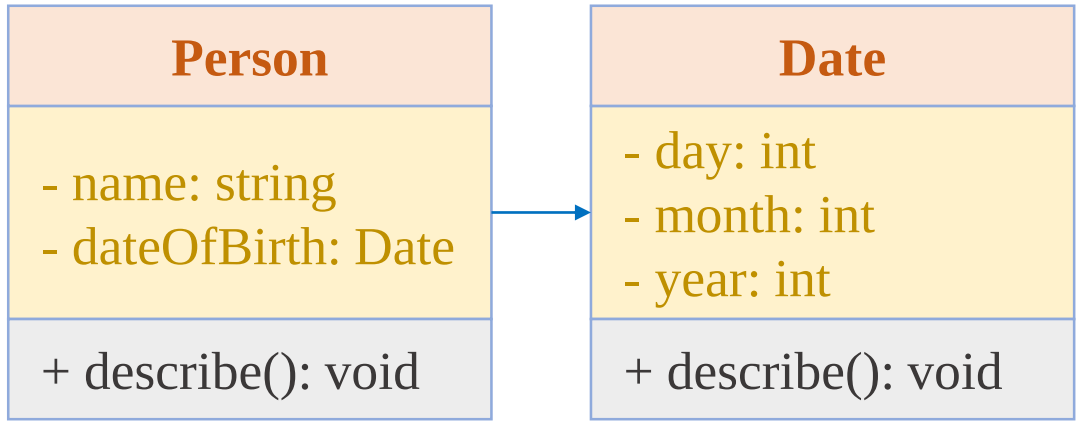



```python
class Date:
    def __init__(self, day, month, year):
        self.__day = day
        self.__month = month
        self.__year = year

    def getDay(self):
        return self.__day

    def getMonth(self):
        return self.__month

    def getYear(self):
        return self.__year
```

Class Data Type

Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.



Using Date as a data type

```python
class Person:
    def __init__(self, name, dateOfBirth):
        self.__name = name
        self.__dateOfBirth = dateOfBirth

    def describe(self):
        # print name
        print(self.__name)

        # print date
        day = self.__dateOfBirth.getDay()
        month = self.__dateOfBirth.getMonth()
        year = self.__dateOfBirth.getYear()
        print(f'{day}/{month}/{year}')
```

```python
date = Date(10, 1, 2000)
peter = Person('Peter', date)
peter.describe()
```

```
Peter
10/1/2000
```

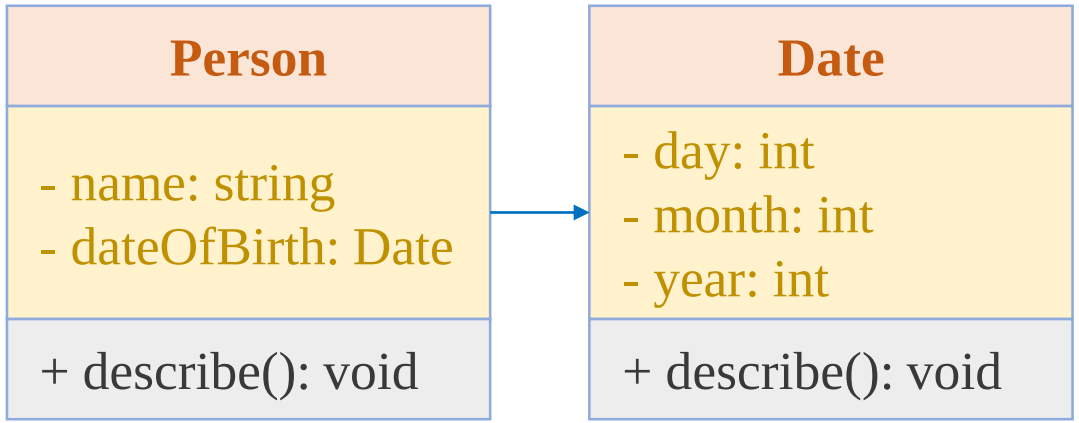
Class Data Type

Using a class as a data type

Aggregation

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

Draw a class diagram and implement in Python

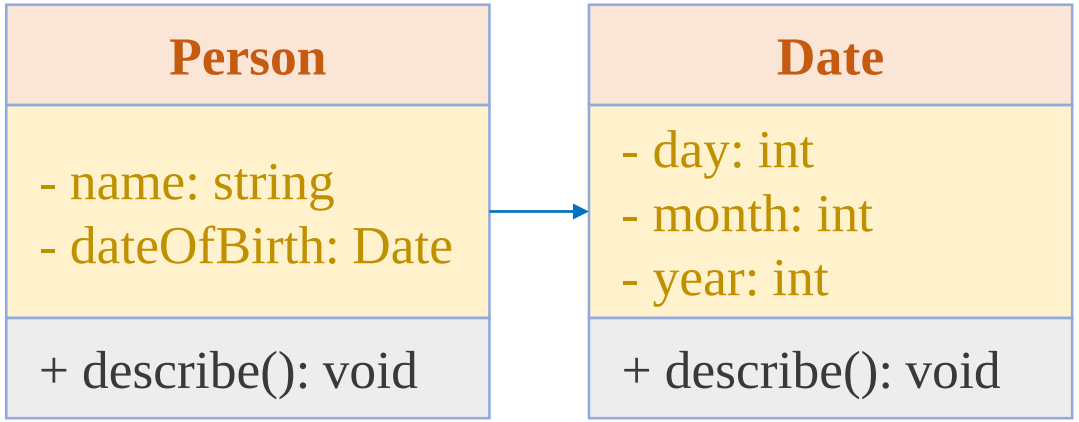



```python
class Date:
    def __init__(self, day, month, year):
        self.__day = day
        self.__month = month
        self.__year = year

    def getDay(self):
        return self.__day

    def getMonth(self):
        return self.__month

    def getYear(self):
        return self.__year

    def describe(self):
        print(f'{self.__day}/{self.__month}/{self.__year}')
```

Class Data Type

Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.



Using Date as a data type

```python
class Person:
    def __init__(self, name, dateOfBirth):
        self.__name = name
        self.__dateOfBirth = dateOfBirth

    def describe(self):
        # print name
        print(self.__name)

        # print date
        self.__dateOfBirth.describe()
```

```python
date = Date(10, 1, 2000)
peter = Person('Peter', date)
peter.describe()
```

```
Peter
10/1/2000
```

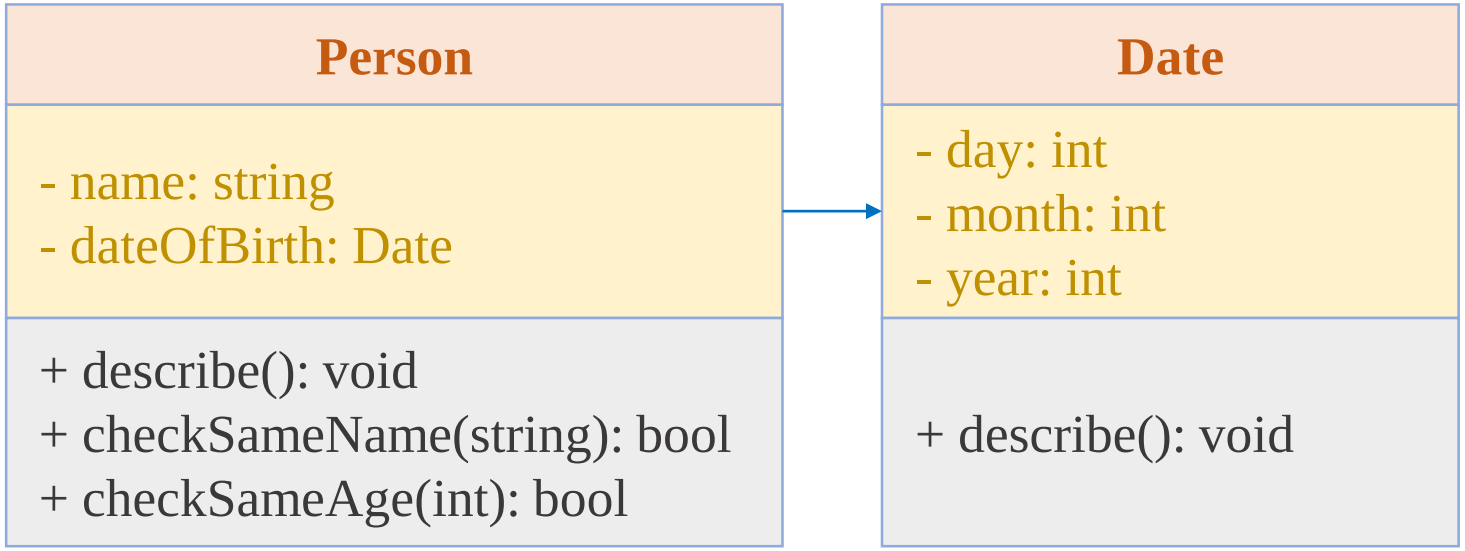
Class Data Type

Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

Write a function to check if two people have the same name.

Write a function to check if two people have the same date of birth.



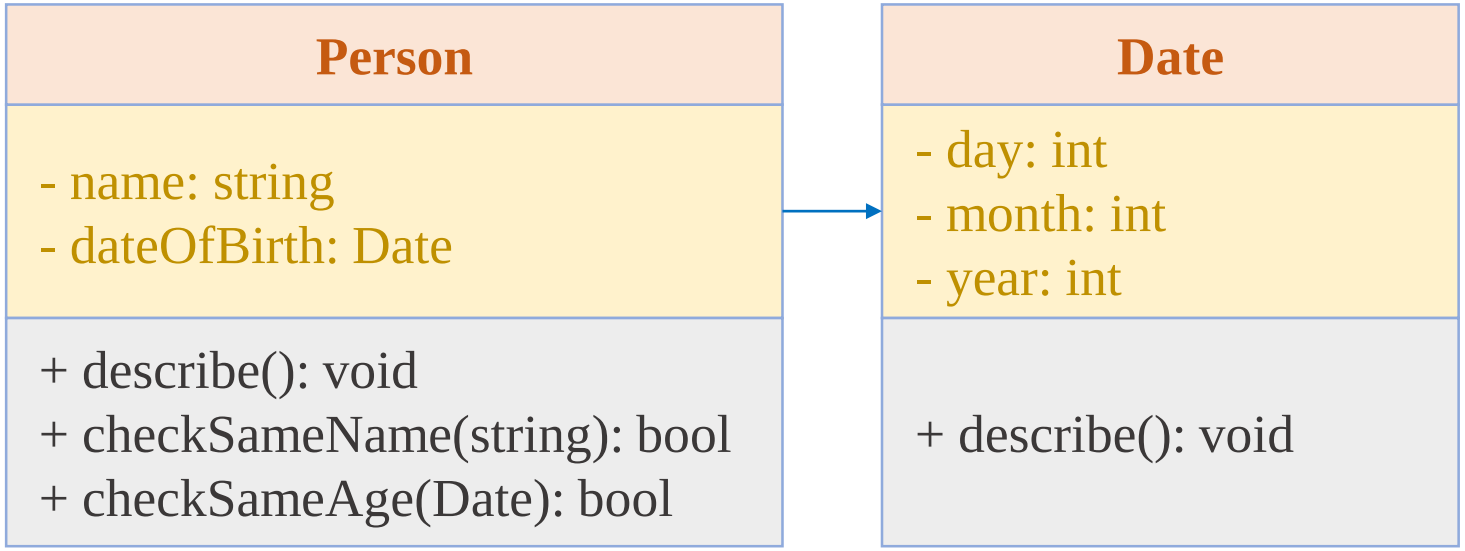
Class Data Type

Using a class as a data type

A person comprises a name in string and a date of birth. A date consists of day, month, and year.

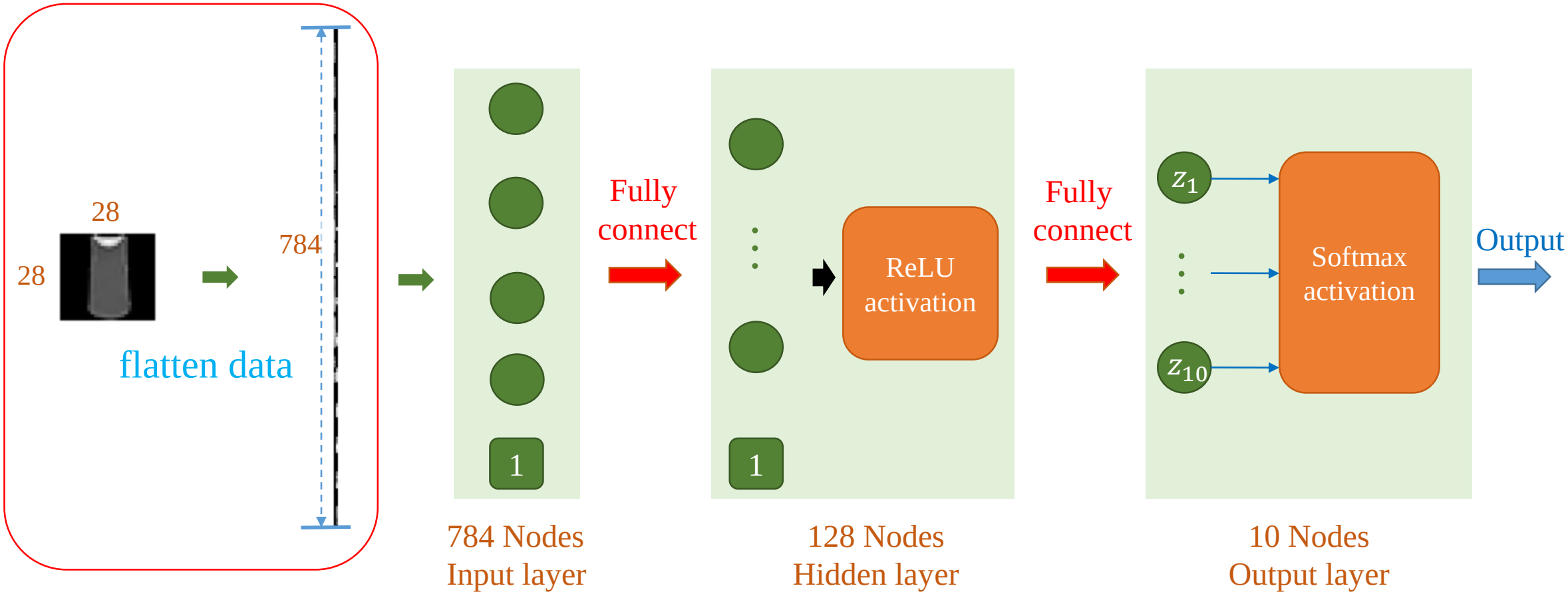
Write a function to check if two people have the same name.

Write a function to check if two people have the same date of birth.



Custom MLP for Fashion-MNIST

❖ ReLU and SGD



Custom MLP for Fashion-MNIST

❖ Custom ReLU

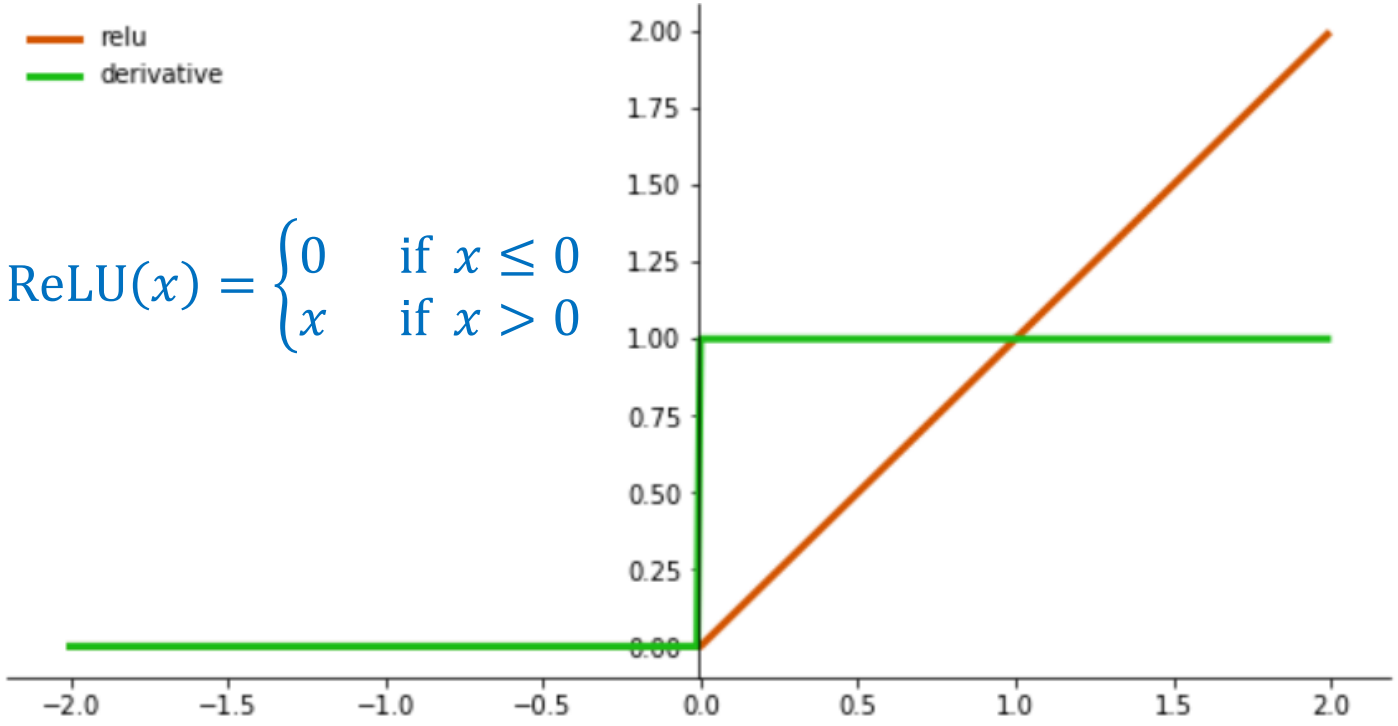
init method

Initialize values/variables necessary for a class

call method

Forward computation max(0, x)

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$




```python
class MyReluActivation(tf.keras.layers.Layer):
    def __init__(self):
        super(MyReluActivation, self).__init__()

    def call(self, inputs):
        return tf.maximum(inputs, 0)
```

Custom MLP for Fashion-MNIST

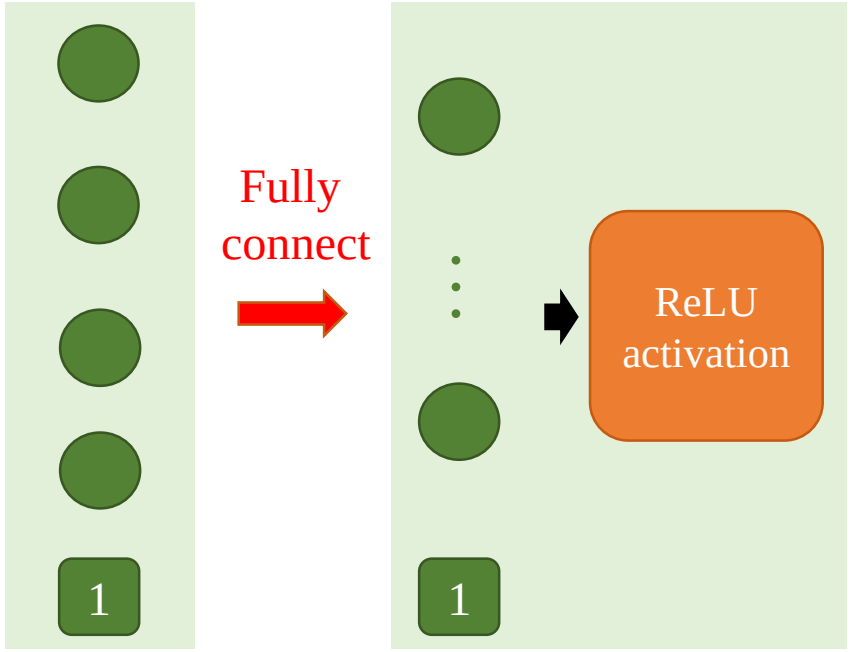
Custom Dense

init method
Initialize values/variables necessary for a class
build method
Do something using the shape of the input tensors
call method
Forward computation $z = x\theta$

$\theta = [\theta_1 \quad \theta_2 \quad \dots \quad \theta_{128}]$

num_outputs = 128

input_shape = (None, 784)

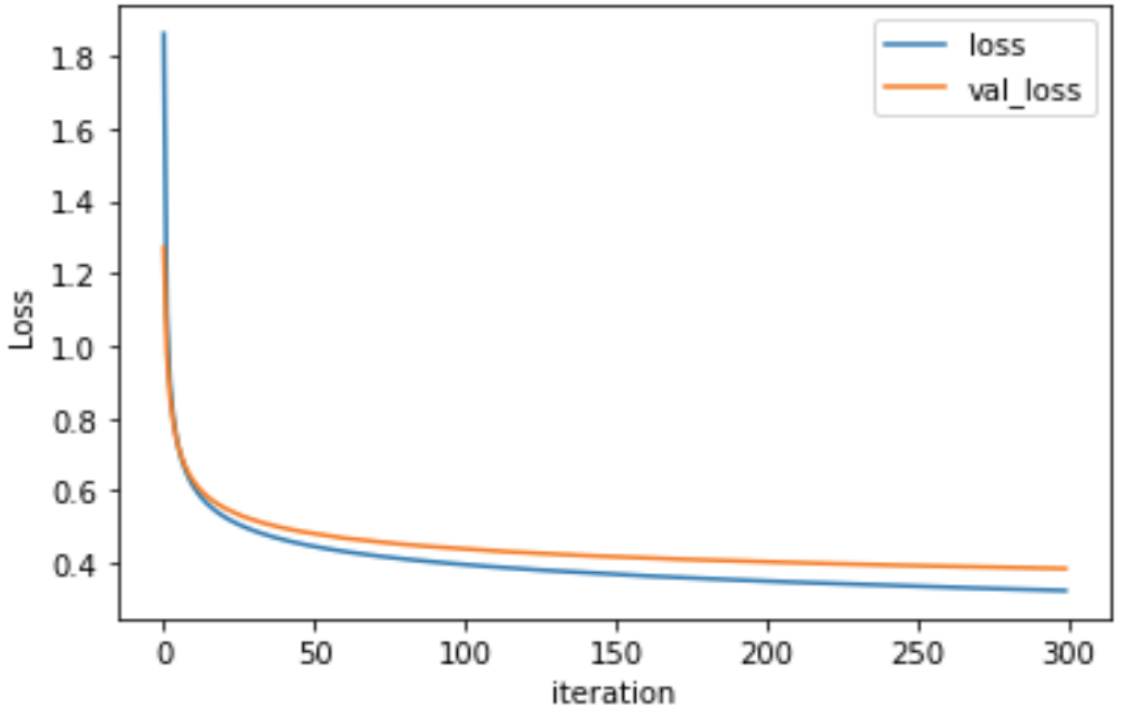


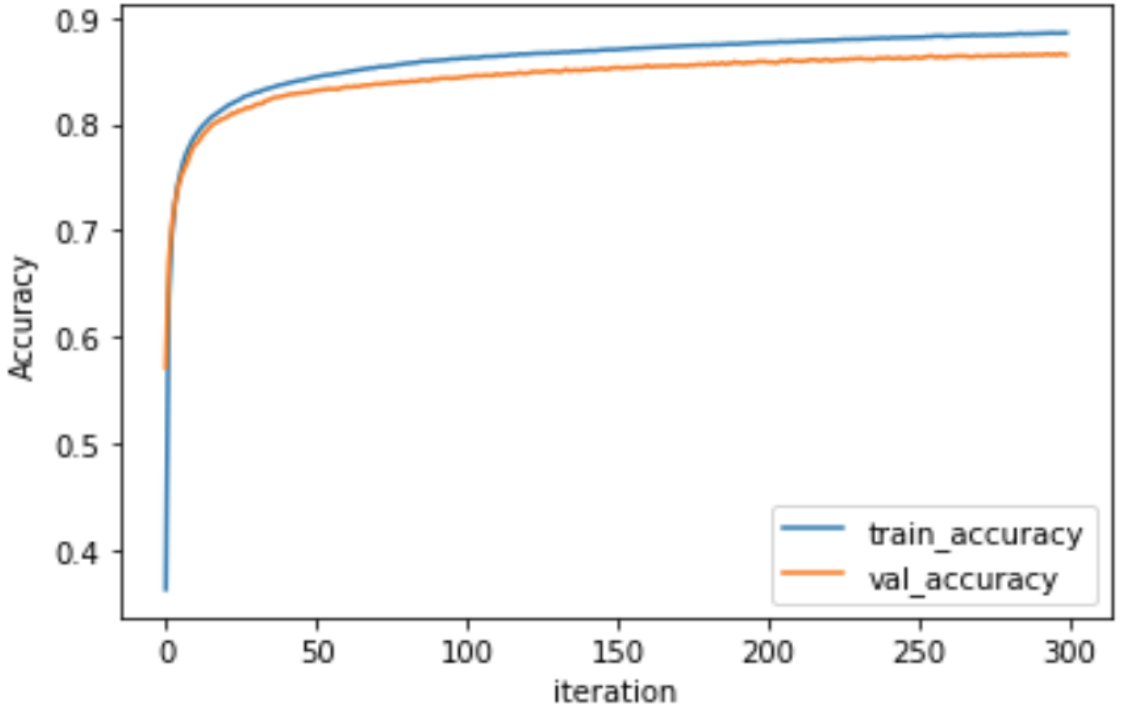

```
1  class MyDenseLayer(tf.keras.layers.Layer):
2      def __init__(self, num_outputs):
3          super(MyDenseLayer, self).__init__()
4          self.num_outputs = num_outputs
5
6      def build(self, input_shape):
7          kernel_shape = [int(input_shape[-1]), self.num_outputs]
8          self.kernel = self.add_weight("kernel", shape=kernel_shape)
9
10     def call(self, inputs):
11         return tf.matmul(inputs, self.kernel)
```

Custom MLP for Fashion-MNIST

Result

```python
import tensorflow as tf
import tensorflow.keras as keras

# create model
model = keras.Sequential()
model.add(keras.Input(shape=(784,)))
model.add(MyDenseLayer(128))
model.add(MyReluActivation())
model.add(MyDenseLayer(10))
model.add(tf.keras.layers.Softmax())

# optimizer and loss
opt = tf.keras.optimizers.SGD(0.001)
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

# training
batch_size = 256
history = model.fit(X_train, y_train, batch_size,
                    validation_data=(X_test, y_test),
                    epochs=300, verbose=2)
```





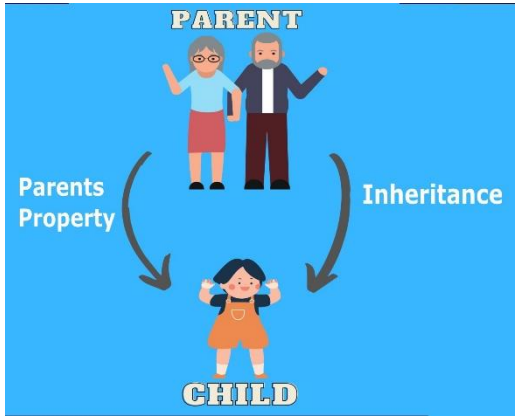
Outline

- Introduction
- Classes and Objects
- Iterator and Class Data Type
- Inheritance
- Exercises

Inheritance

❖ Introduction

Mechanism by which one class is allowed to inherit the features (attributes and methods) of another class.



Super Class: The class whose features are inherited is known as superclass (a base class or a parent class).

Subclass: The class that inherits the other class is known as subclass (a derived class, extended class, or child class).

The subclass can add its own attributes and methods in addition to the superclass attributes and methods.

Inheritance

❖ Introduction

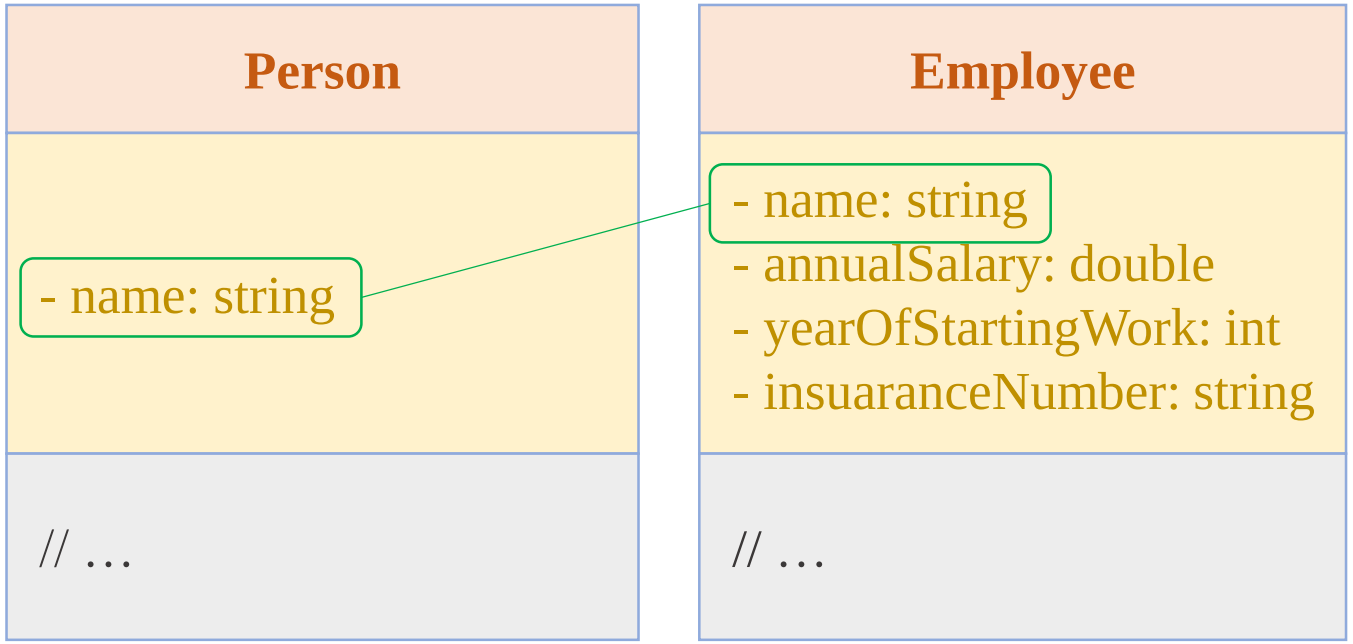
Create a class called Employee whose objects are records for an employee. This class will be a derived class of the class Person.

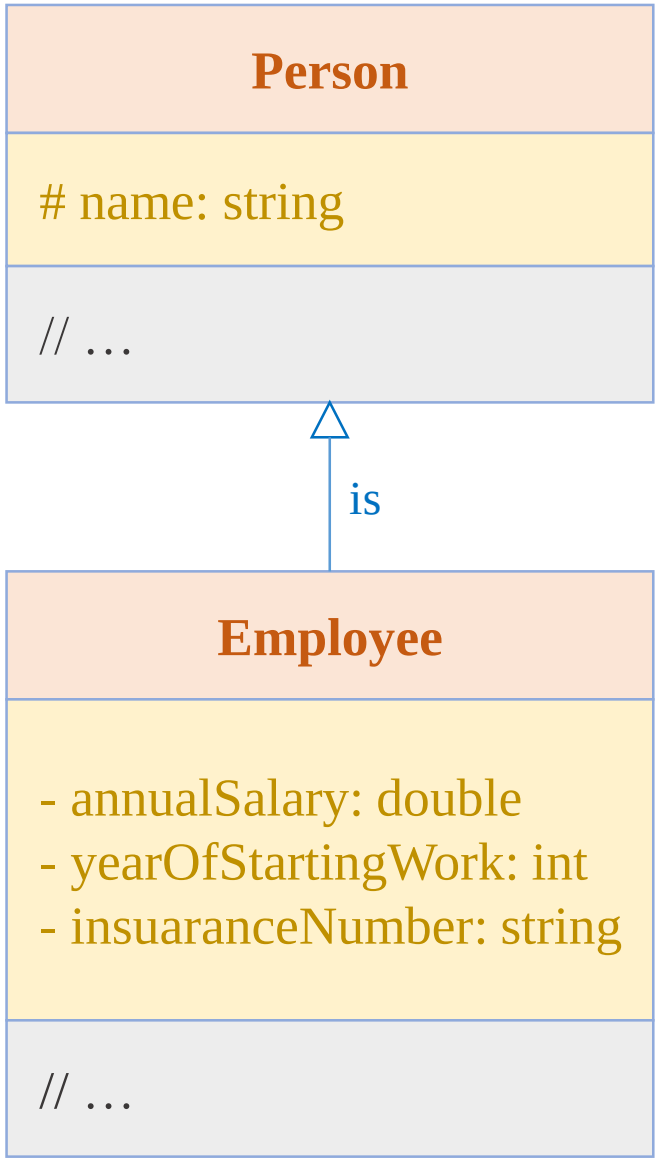
An employee record has an employee's name (inherited from the class Person), an annual salary represented as a single value of type double, a year the employee started work as a single value of type int and a national insurance number, which is a value of type String.

Person
- name: string
// …

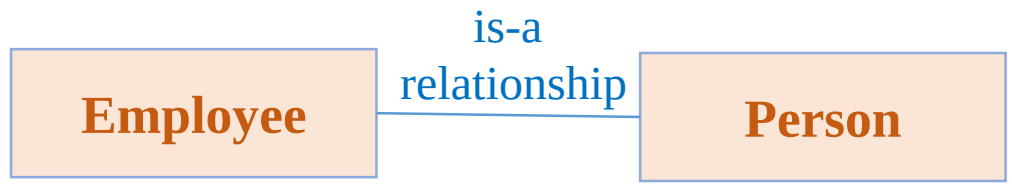
Employee
- name: string - annualSalary: double - yearOfStartingWork: int - insuaranceNumber: string
// …

Inheritance





Access modifiers
- private
+ public
# protected

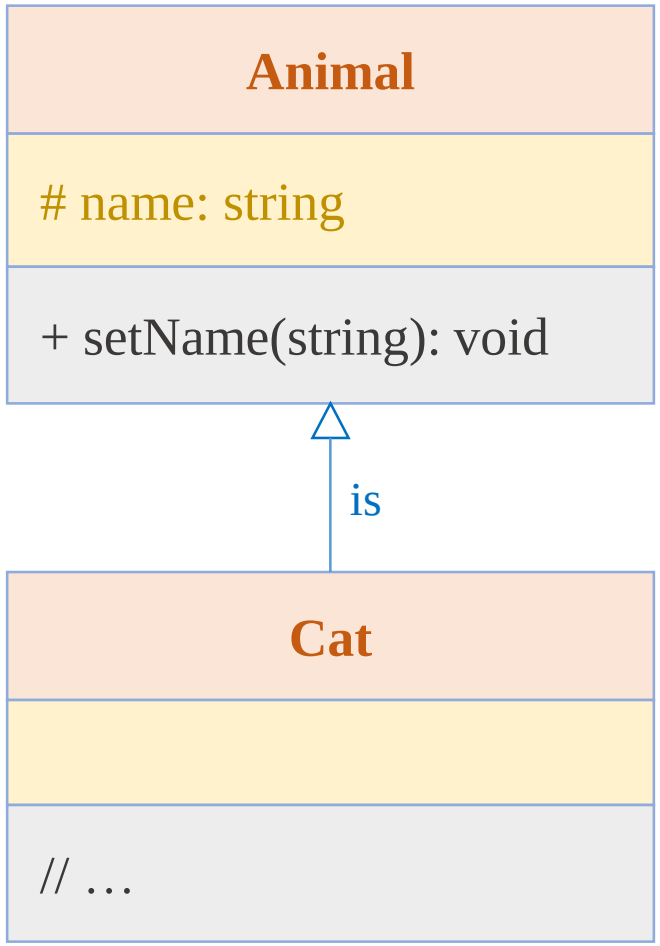


An employee is a person.

Inheritance

Inherit attributes and methods from one class to another

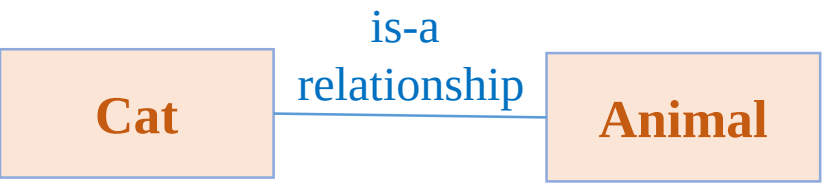
Benefit: Code reusability



Derived class (child) - the class that inherits from another class

Base class (parent) - the class being inherited from

Derived class : Base class



A cat is an animal.

```python
class Animal:
    def __init__(self, name):
        self._name = name

    def setName(self, name):
        self._name = name

    def describe(self):
        print(f'Name: {self._name}')

class Cat(Animal):
    def __init__(self, name):
        Animal.__init__(self, name)

cat = Cat('Calico')
cat.describe()
```

```
Name: Calico
```

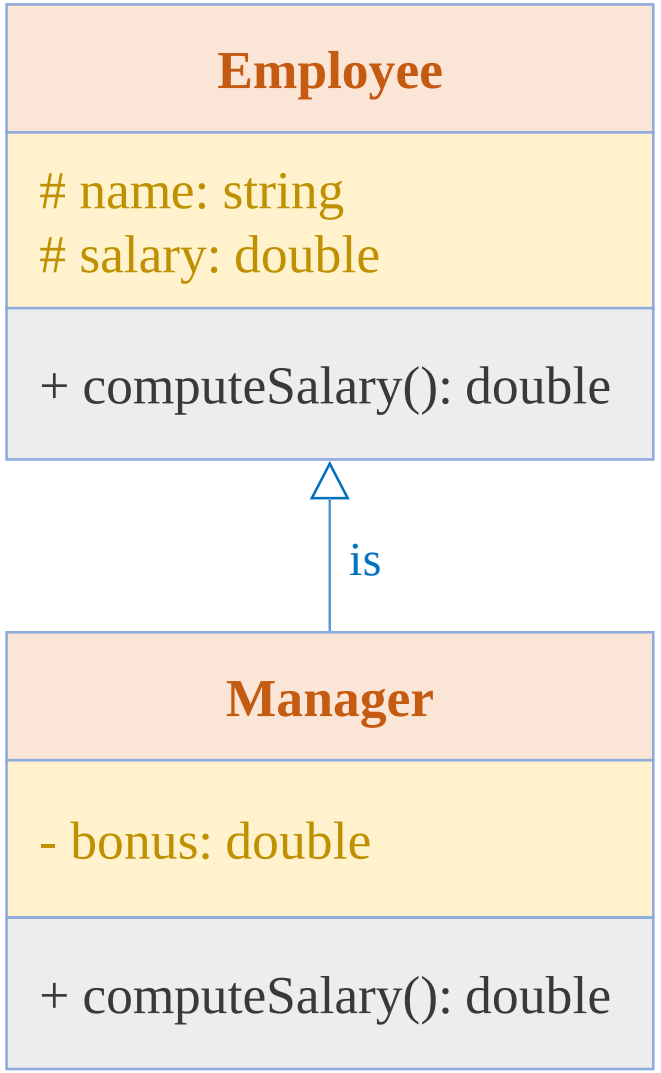
Inheritance

❖ Introduction

To extend an existing class

UML Annotation
'-' stands for 'private'
'#' stands for 'protected'
'+' stands for 'public'

Super Class



Subclass

What features does a manager inherit?

Inheritance

❖ As a template

A class Animal that has a method sound() and the subclasses of it like Dog, Lion, Horse, Cat, etc.

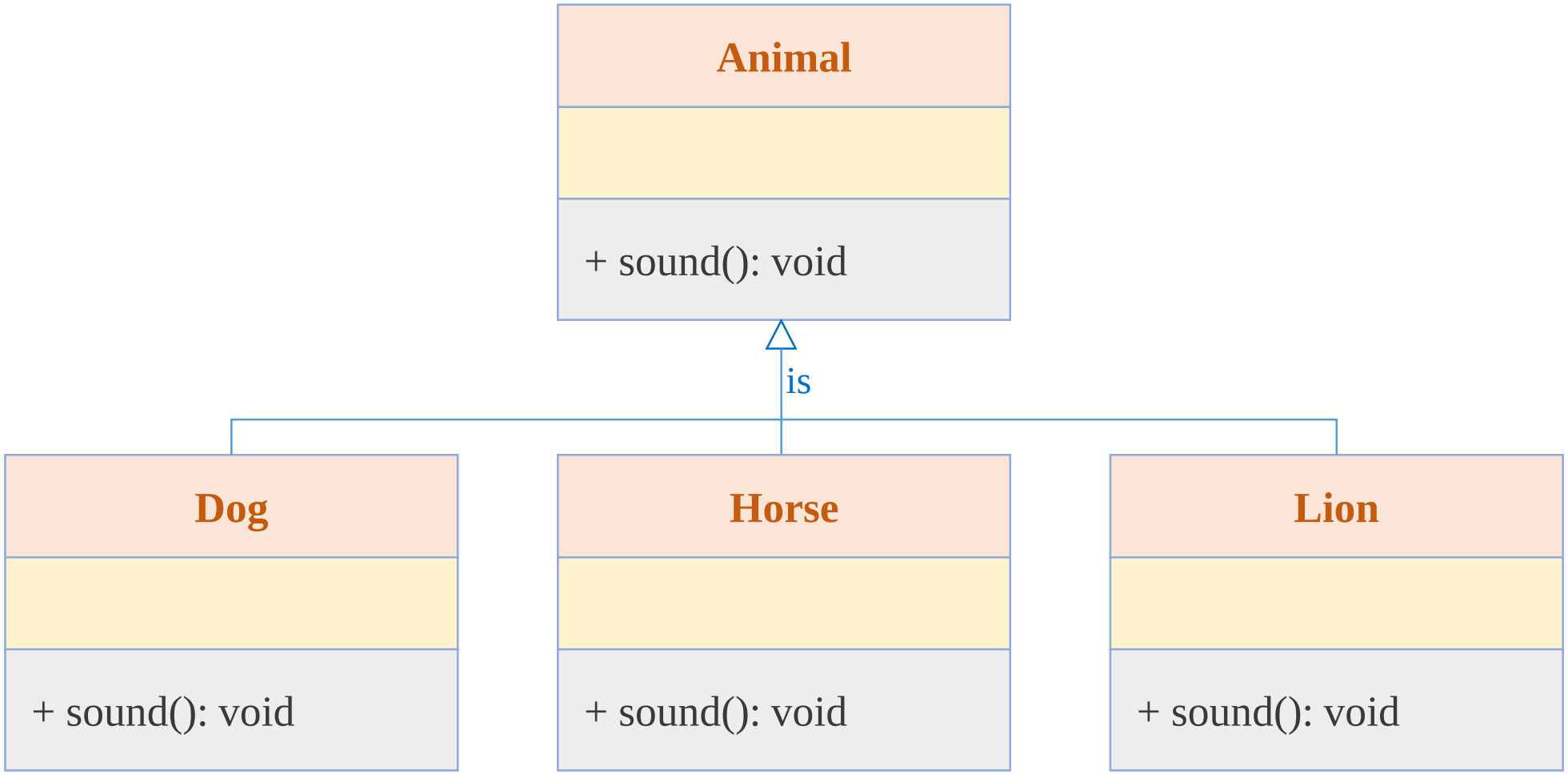
Since the animal sound differs from one animal to another, there is no point to implement this method in parent class.

This is because every child class must override this method to give its own implementation details, like Lion class will say “Roar” in this method and Dog class will say “Woof”.

https://beginnersbook.com/2013/05/java-abstract-class-method/

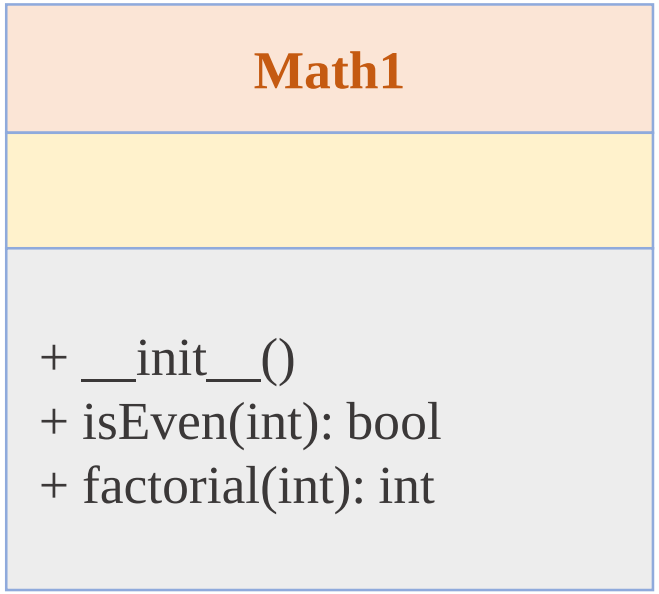
Inheritance

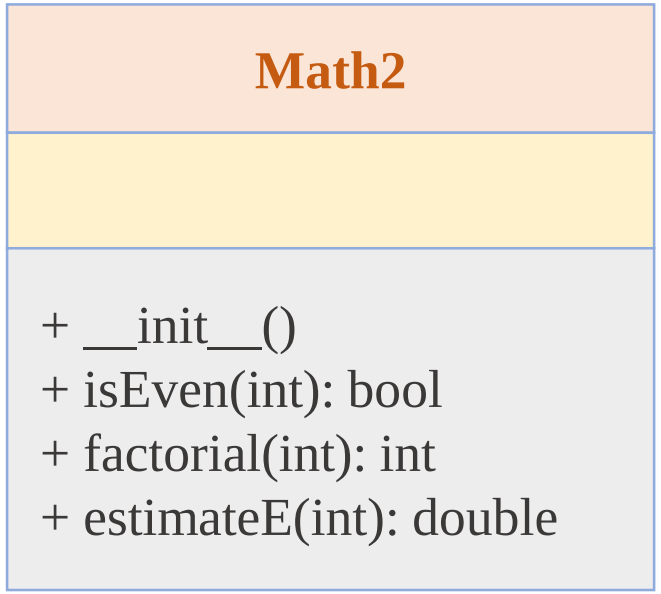
❖ As a template



Example

❖ **Implement the two classes below**





Example

Implement the two classes below

Math1
+ __init__() + isEven(int): bool + factorial(int): int

```python
class Math1:
    def isEven(self, number):
        if number%2:
            return False
        else:
            return True

    def factorial(self, number):
        result = 1

        for i in range(1, number+1):
            result = result*i

        return result
```

```python
# test Math1
math1 = Math1()

# isEven() sample: number=5 -> False
# isEven() sample: number=6 -> True
print(math1.isEven(5))
print(math1.isEven(6))

# factorial() sample: number=4 -> 24
# factorial() sample: number=5 -> 120
print(math1.factorial(4))
print(math1.factorial(5))
```

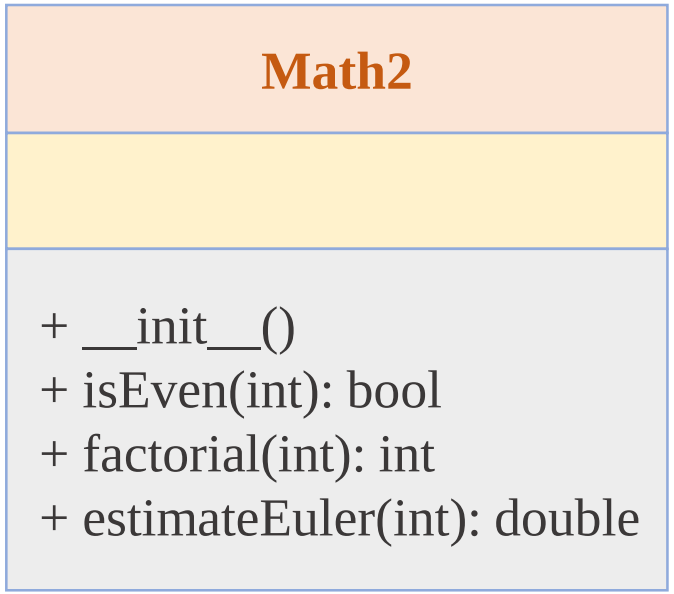
```
False
True
24
120
```

Example

Implement the two classes below

$$e = 2.71828$$

$$e \approx 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!}$$



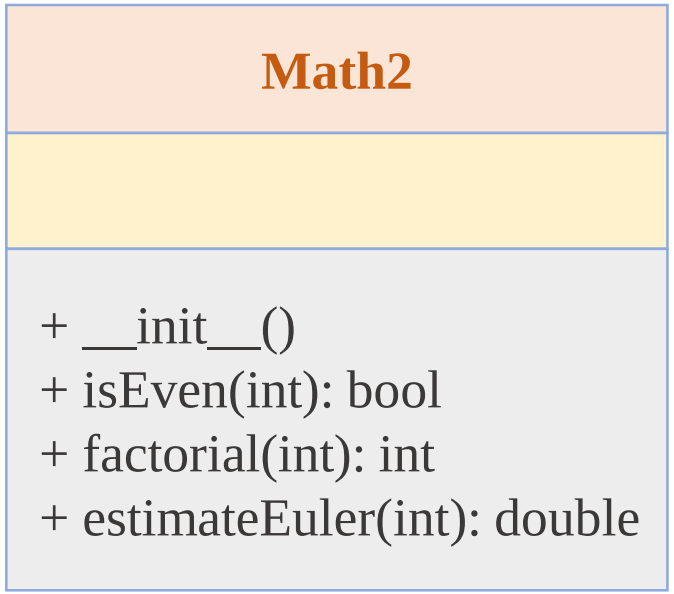

```python
class Math2:
    def isEven(self, number):
        if number%2:
            return False
        else:
            return True

    def factorial(self, number):
        result = 1

        for i in range(1, number+1):
            result = result*i

        return result

    def estimateEuler(self, number):
        result = 1

        for i in range(1, number+1):
            result = result + 1/self.factorial(i)

        return result
```

Example

Implement the two classes below

$$e = 2.71828$$

$$e \approx 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{n!}$$




```python
# test Math2
math2 = Math2()

# isEven() sample: number=5 -> False
# isEven() sample: number=6 -> True
print(math2.isEven(5))
print(math2.isEven(6))

# factorial() sample: number=4 -> 24
# factorial() sample: number=5 -> 120
print(math2.factorial(4))
print(math2.factorial(5))

# estimateEuler() sample: number=2 -> 2.5
# estimateEuler() sample: number=8 -> 2.71
print(math2.estimateEuler(2))
print(math2.estimateEuler(8))
```

```
False
True
24
120
2.5
2.71827876984127
```

AI VIETNAM
All-in-One Course

Example

How to reuse an existing class?

Math1

+ __init__()
+ isEven(int): bool
+ factorial(int): int

```
class Math1:
    def isEven(self, number):
        if number%2:
            return False
        else:
            return True

    def factorial(self, number):
        result = 1

        for i in range(1, number+1):
            result = result*i

        return result
```

Math2

+ __init__()
+ isEven(int): bool
+ factorial(int): int
+ estimateEuler(int): double

```
class Math2:
    def isEven(self, number):
        if number%2:
            return False
        else:
            return True

    def factorial(self, number):
        result = 1

        for i in range(1, number+1):
            result = result*i

        return result

    def estimateEuler(self, number):
        result = 1

        for i in range(1, number+1):
            result = result + 1/self.factorial(i)

        return result
```

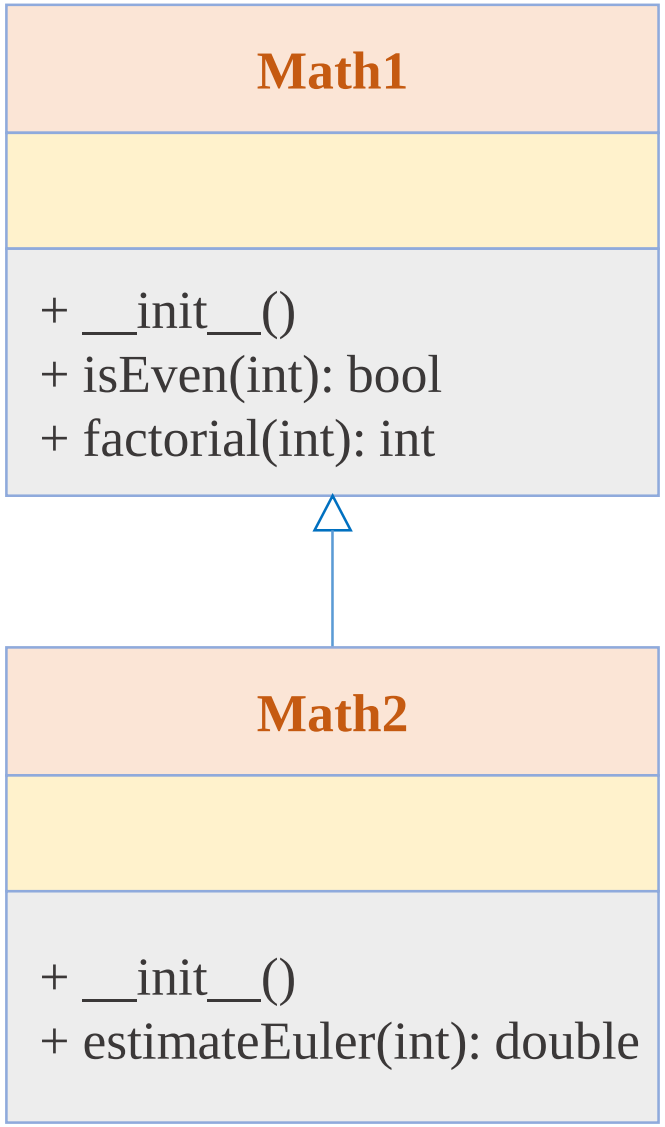
Example

❖ Inheritance

Math1: super class or parent class

Math2: child class or derived class

Child classes can use the public and protected attributes and methods of the super classes.



```python
class Math1:
    def isEven(self, number):
        if number%2:
            return False
        else:
            return True

    def factorial(self, number):
        result = 1

        for i in range(1, number+1):
            result = result*i

        return result
```

```python
class Math2(Math1):
    def estimateEuler(self, number):
        result = 1

        for i in range(1, number+1):
            result = result + 1/self.factorial(i)

        return result
```

```python
# test Math2
math2 = Math2()

# isEven() sample: number=5 -> False
# isEven() sample: number=6 -> True
print(math2.isEven(5))
print(math2.isEven(6))

# factorial() sample: number=4 -> 24
# factorial() sample: number=5 -> 120
print(math2.factorial(4))
print(math2.factorial(5))

# estimateEuler() sample: number=2 -> 2.5
# estimateEuler() sample: number=8 -> 2.71
print(math2.estimateEuler(2))
print(math2.estimateEuler(8))
```

```
False
True
24
120
2.5
2.71827876984127
```

```python
class Math1:
    def isEven(self, number):
        if number%2:
            return False
        else:
            return True

    def factorial(self, number):
        result = 1

        for i in range(1, number+1):
            result = result*i

        return result
```

```python
class Math2(Math1):
    def estimateEuler(self, number):
        result = 1

        for i in range(1, number+1):
            result = result + 1/super().factorial(i)

        return result
```

```python
# test Math2
math2 = Math2()

# isEven() sample: number=5 -> False
# isEven() sample: number=6 -> True
print(math2.isEven(5))
print(math2.isEven(6))

# factorial() sample: number=4 -> 24
# factorial() sample: number=5 -> 120
print(math2.factorial(4))
print(math2.factorial(5))

# estimateEuler() sample: number=2 -> 2.5
# estimateEuler() sample: number=8 -> 2.71
print(math2.estimateEuler(2))
print(math2.estimateEuler(8))
```

```
False
True
24
120
2.5
2.71827876984127
```

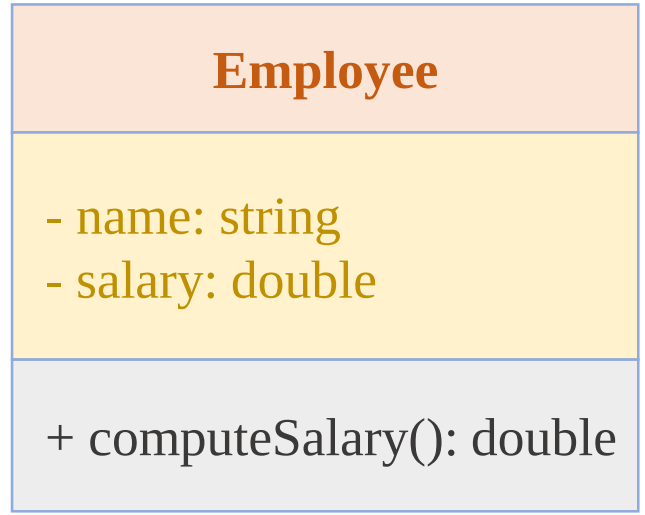
Another Example

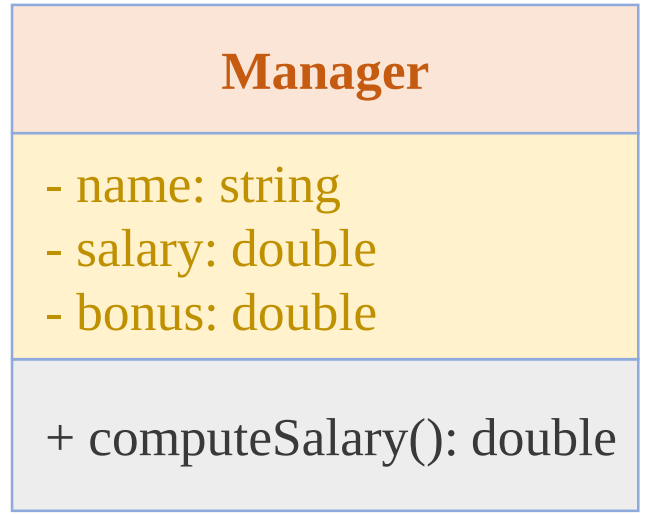
Employee-Manager Example: Simple requirement

A standard employee of company X includes his/her name and base salary. For example, Peter is working for X, and his base salary is 60000$ a year. Implement the Employee class and the computeSalary() method to compute the final salary for an employee. The salary for an employee is his/her base salary.

A manager includes his/her name, base salary, and bonus. The final salary for the manager comprises the base salary and a bonus. For example, Mary is a manager in the company. Her base salary and bonus are 60000$ and 20000$ a year, respectively. Yearly, she gets paid 80000$ a year. Implement the Manager class and the computeSalary() method to compute the final salary.

Demo





Another Example

Employee-Manager Example

A standard employee of company X includes his/her name and base salary. For example, Peter is working for X, and his base salary is 60000$ a year.

Implement the Employee class and the computeSalary() method to compute the final salary for an employee. The salary for an employee is his/her base salary.

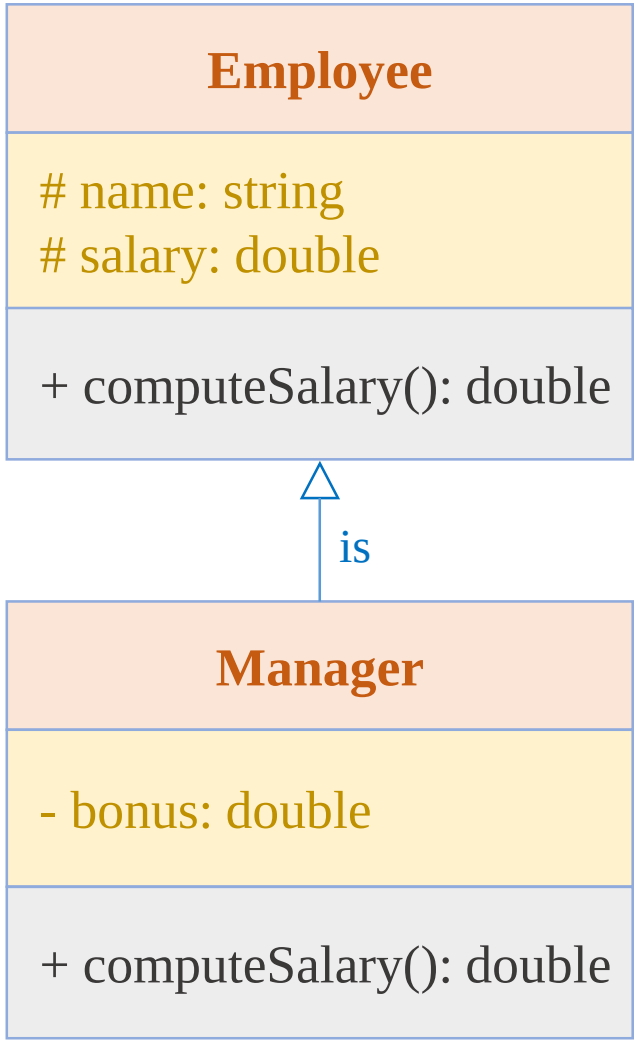
A manager is an employee who has the name and base salary attributes. However, the final salary for the manager comprises the base salary and a bonus.

For example, Mary is a manager in the company. Her base salary and bonus are 60000$ and 20000$ a year, respectively. Yearly, she gets paid 80000$ a year.

Implement the Manager class and the computeSalary() method to compute the final salary.

Another Example

Employee-Manager




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

    def computeSalary(self):
        return self._salary

class Manager(Employee):
    def __init__(self, name, salary, bonus):
        self._name = name
        self._salary = salary
        self.__bonus = bonus

    def computeSalary(self):
        return super().computeSalary() + self.__bonus
```

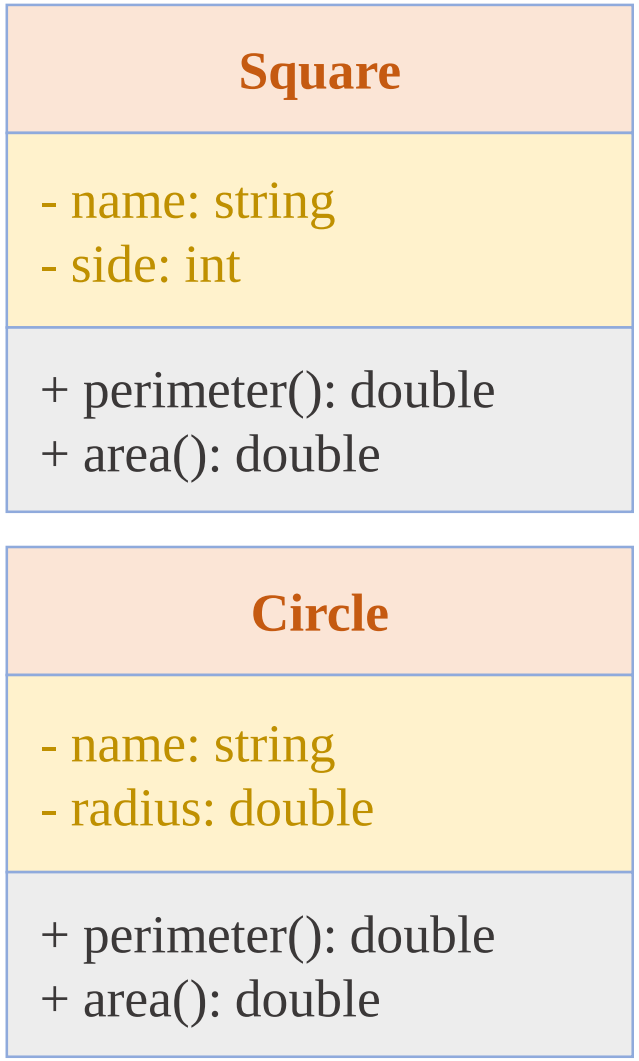
```python
peter = Manager('Peter', 100, 20)
salary = peter.computeSalary()
print(f'Peter Salary: {salary}')
```

```
Peter Salary: 120
```

Example

❖ Inheritance recognition

Squares and circles are both examples of shapes. there are certain questions one can reasonably ask of both a circle and a square (such as, ‘what is the area?’ or ‘what is the perimeter?’) but some questions can be asked only of one or the other but not both (such as, ‘what is the length of a side?’ or ‘what is the radius?’)



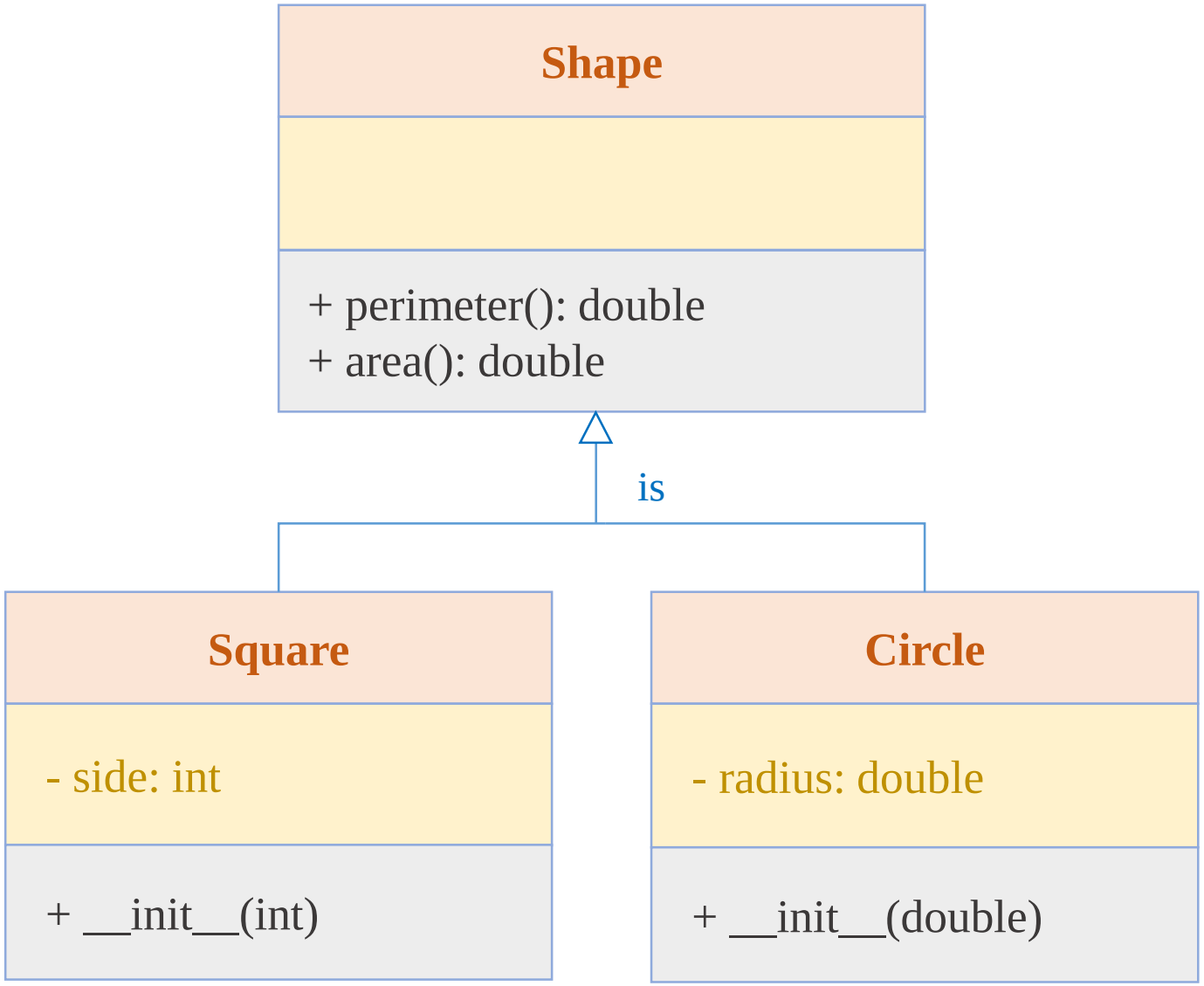
Example

❖ Inheritance recognition

Shape does not know how to compute its perimeter and area

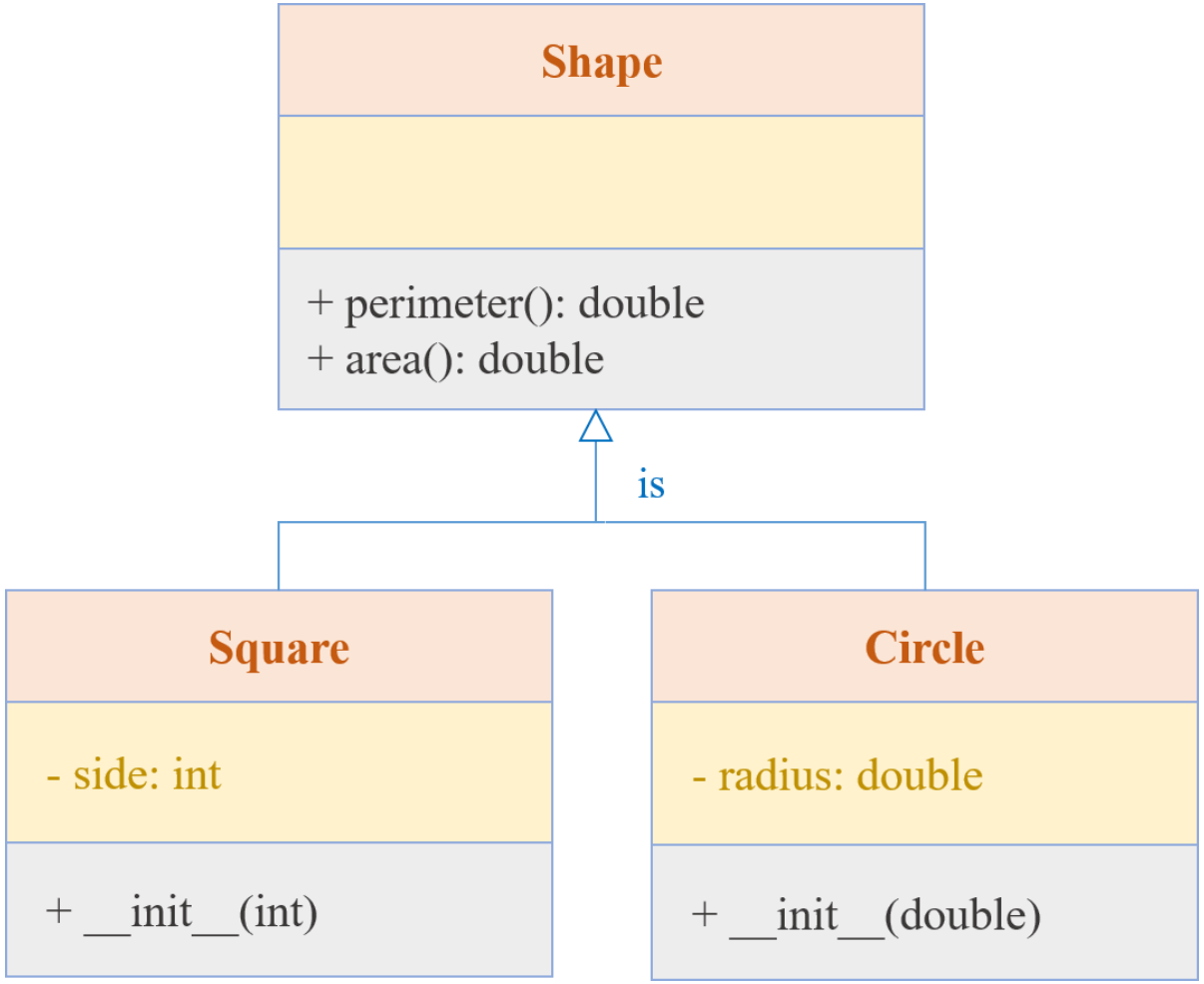
Use @abstractmethod to ask its child to implement them

Using pass in the abstract method



Example

❖ Inheritance recognition



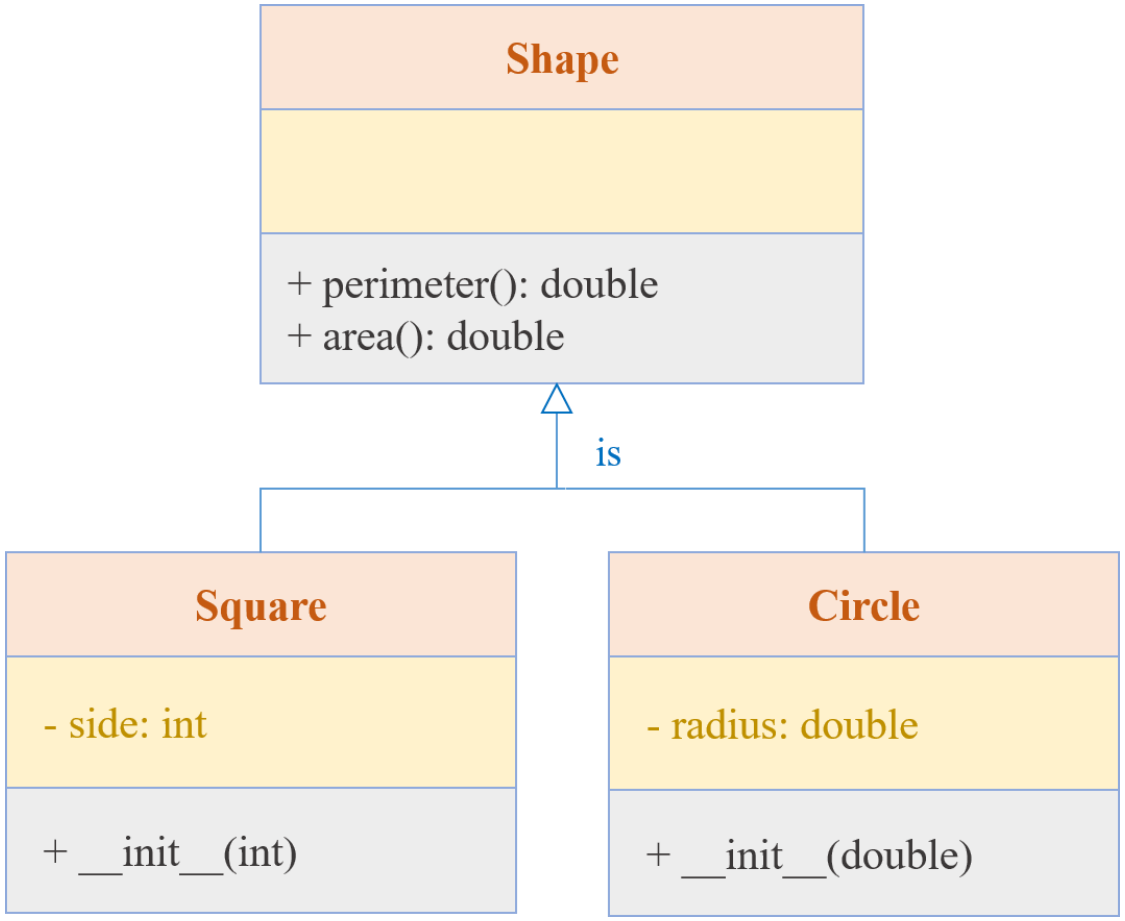

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def computeArea(self):
        pass
```

```python
class Square(Shape):
    def __init__(self, side):
        self.__side = side

    def computeArea(self):
        return self.__side*self.__side
```

```python
square = Square(5)
print(square.computeArea())
```

Example

❖ Inheritance recognition




```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def computeArea(self):
        pass
```

```python
class Cicle(Shape):
    def __init__(self, radius):
        self.__radius = radius
```

```python
circle = Cicle(5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [8], in <cell line: 1>()
----> 1 circle = Cicle(5)

TypeError: Can't instantiate abstract class Cicle with abstract method computeArea
```

Example

❖ Inheritance recognition

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def computeArea(self):
        pass
```

```python
import math

class Cicle(Shape):
    def __init__(self, radius):
        self.__radius = radius

    def computeArea(self):
        return self.__radius*self.__radius*math.pi
```

```python
class Square(Shape):
    def __init__(self, side):
        self.__side = side

    def computeArea(self):
        return self.__side*self.__side
```

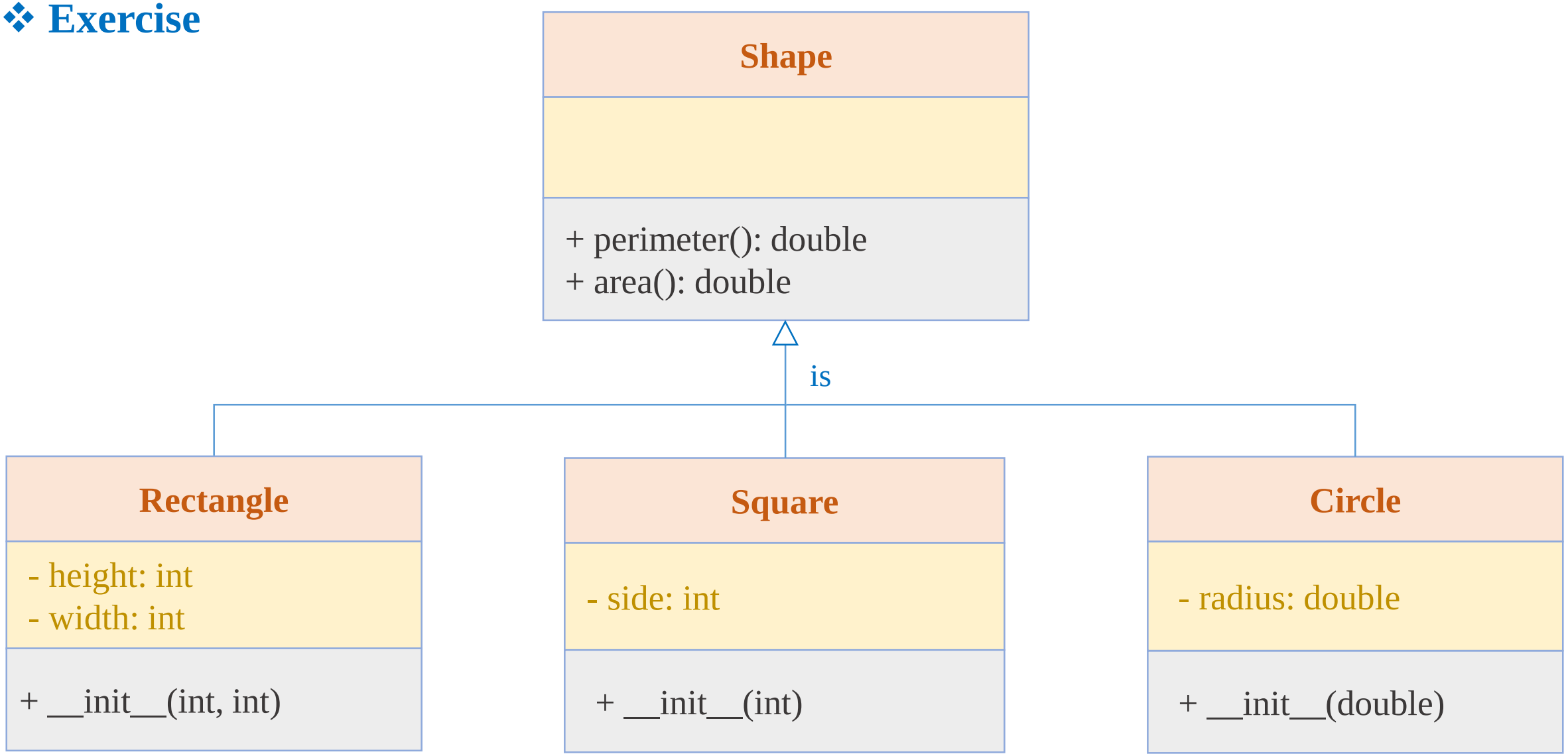
```python
circle = Cicle(5)
print(circle.computeArea())
```

```
78.53981633974483
```

```python
square = Square(5)
print(square.computeArea())
```

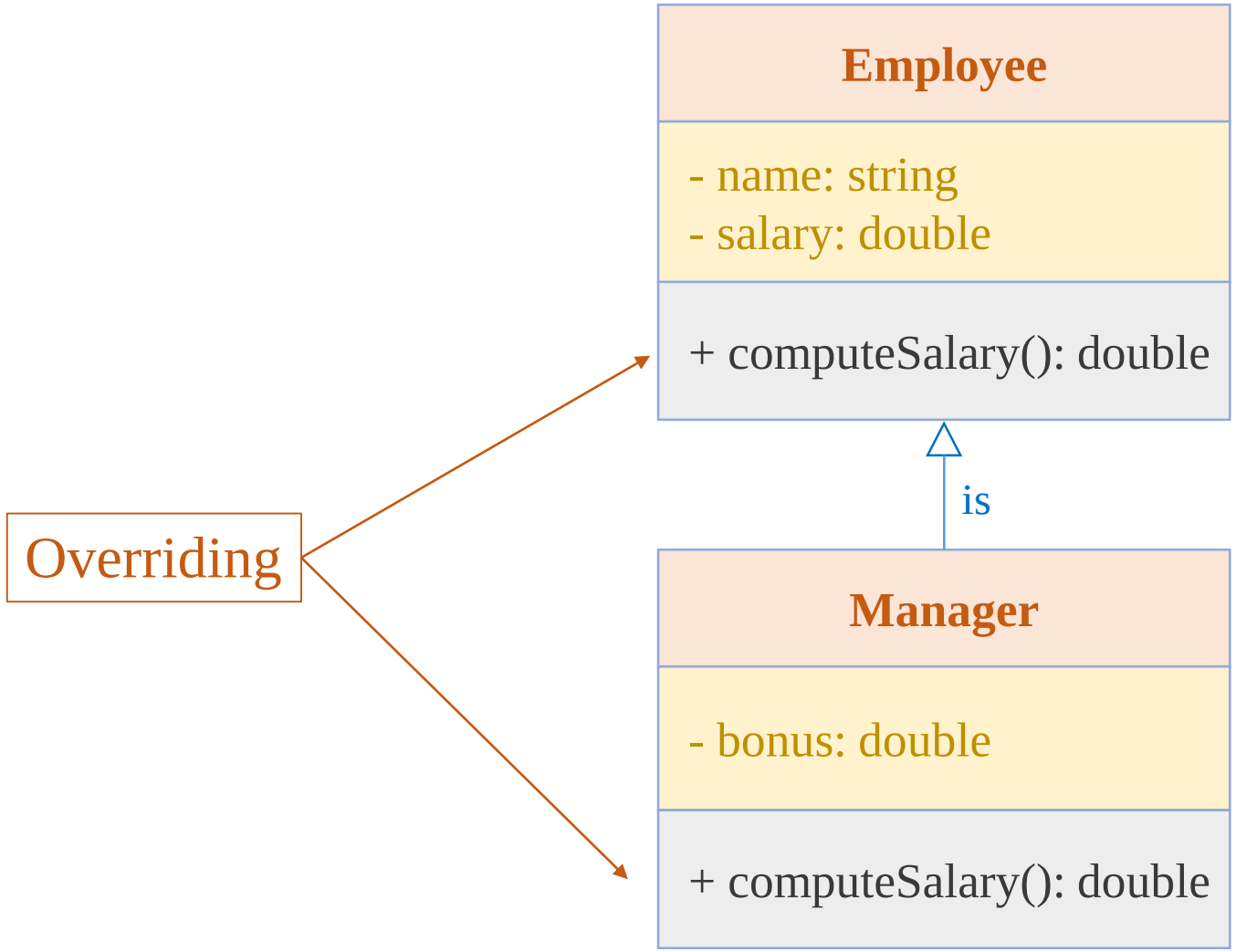
Example

❖ Exercise



Overriding

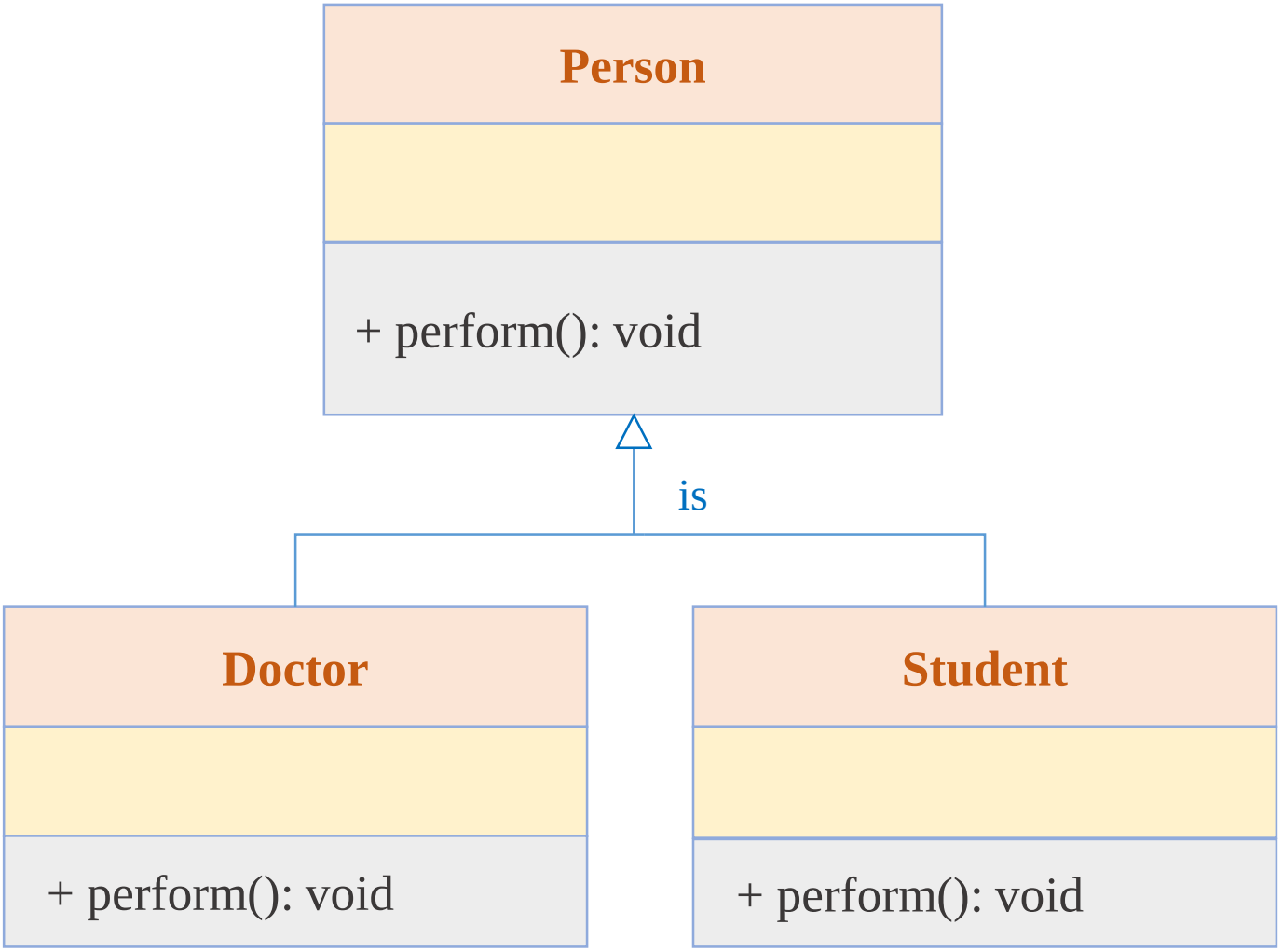
Overriding is a feature that allows a child class to provide a specific implementation of a method that is already provided by its super-class.



AI VIETNAM
All-in-One Course

Overriding

Example




```python
class Person:
    def perform(self):
        print('Person activities')

class Doctor(Person):
    def perform(self):
        print('Doctor activities')

class Student(Person):
    def perform(self):
        print('Student activities')
```

```python
person = Person()
person.perform()
```

```
Person activities
```

```python
doctor = Doctor()
doctor.perform()
```

```
Doctor activities
```

Exercise

A vector in 2D includes x and y.

1) Implement a constructor with no parameters. This constructors set 1 to x and y.

2) Implement a constructor with two parameters x and y.

3) Implement a destructor

4) Write a method to compute the addition between a vector and another vector

5) Write a method to compute the dot product between two vectors.

6) Check if two vectors are the same

Vector
- x: int - y: int
+ __init__(int, int) + add(Vector): void + dotProduct(Vector): int + check(Vector): bool

Dot Product

$$\vec{x}.\vec{y} = \sum_{1}^{n} x_i y_i$$

$\vec{x} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$ $\quad \vec{u} = \begin{pmatrix} 4 \\ 4 \end{pmatrix} + \vec{x} = \begin{pmatrix} 6 \\ 7 \end{pmatrix}$

$\vec{y} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$ $\quad \vec{v} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} + \vec{y} = \begin{pmatrix} 7 \\ 5 \end{pmatrix}$