AI VIETNAM
All-in-One Course

Object-Oriented Programming

Nguyen Dinh Vinh
Ph.D in Computer Science

Year 2023

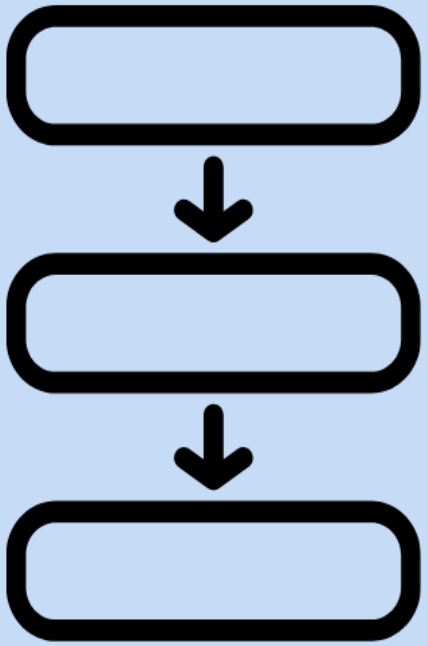
Outline

- Introduction to OOP
- Classes and Objects
- Object Relationships (Composition, Aggregation)
- Summarize
- Further reading

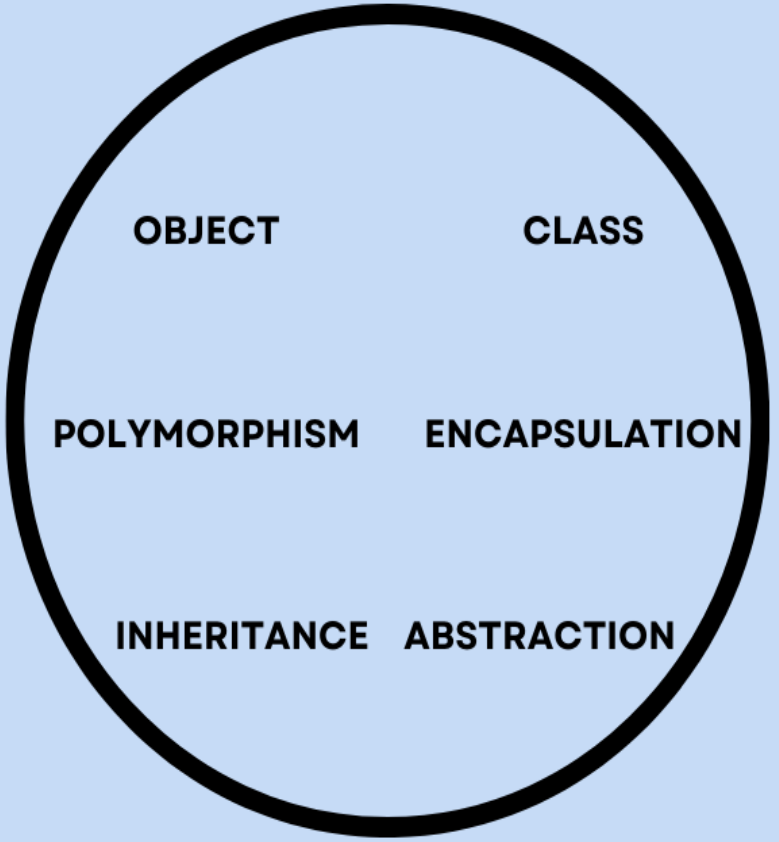
Outline

- Introduction to OOP
- Classes and Objects
- Object Relationships (Composition, Aggregation)
- Summarize
- Further reading

DIFFERENCES: PROCEDURAL AND OOPs PROGRAMMING



VS



PROCEDURAL PROGRAMMING

Image credit: https://www.boardinfinity.com/

OBJECT ORIENTED PROGRAMMING

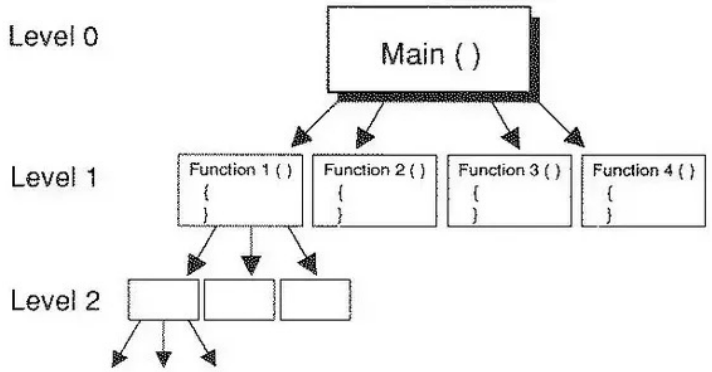
Review: Procedural programming

Develop a program to make a pizza NY Style Sauce and Cheese

1. **Prepare ingredient**
   - Tossing dough, adding sauce, Adding topping: Grated Reggiano cheese
2. **Bake :**
   - Bake for 25 minutes at 150 oC
3. **Cut:**
   - Cutting the pizza into diagonal slices
4. **Box:**
   - Place pizza in official PizzaStore box

Review: Procedural programming

Develop a program to make a pizza NY Style Sauce and Cheese



Top-down approach

```
# phát triển chương trình làm bánh NY Style Sauce and Cheese Pizza
def makeNYPizza():
  #bước 1 – chuẩn bị thành phần làm bánh
  prepareNYPizza()

  #bước 2 – nấu bánh
  bakeNYPizza()

  #bước 3 – cắt bánh thành các lát chéo
  cutNYPizza()

  #bước 4 – đóng gói
  boxNYPizza()

  #main program
makeNYPizza()
```

Review: Procedural programming

Develop a program to make a pizza NY Style Sauce and Cheese

1. **Prepare ingredient**
   - Tossing dough, adding sauce, Adding topping: Grated Reggiano cheese
2. **Bake :**
   - Bake for 25 minutes at 150 oC
3. **Cut:**
   - Cutting the pizza into diagonal slices
4. **Box:**
   - Place pizza in official PizzaStore box

```
ingredientPizza = ["Tossing dough", "Adding Sauce", "Adding topping: Grated Reggiano cheese"]

def prepareNYPizza():
    print("1. Prepare ingredient for making NY Style Pizza")
    for  index, value in enumerate(ingredientPizza):
      print(" 1.{}-{}".format(index+1, value))

def bakeNYPizza():
    print("2. Bake for 25 minutes at 150 oC")

def cutNYPizza():
    print("3. Cutting the pizza into diagonal slices")

def boxNYPizza():
    print("4. Place pizza in official PizzaStore box")
```

Review: Procedural programming

Develop a program to make a pizza NY Style Sauce and Cheese

1. **Prepare ingredient**
   - Tossing dough, adding sauce, Adding topping: Grated Reggiano cheese
2. **Bake :**
   - Bake for 25 minutes at 150 oC
3. **Cut:**
   - Cutting the pizza into diagonal slices
4. **Box:**
   - Place pizza in official PizzaStore box

```
makeNYPizza()
```

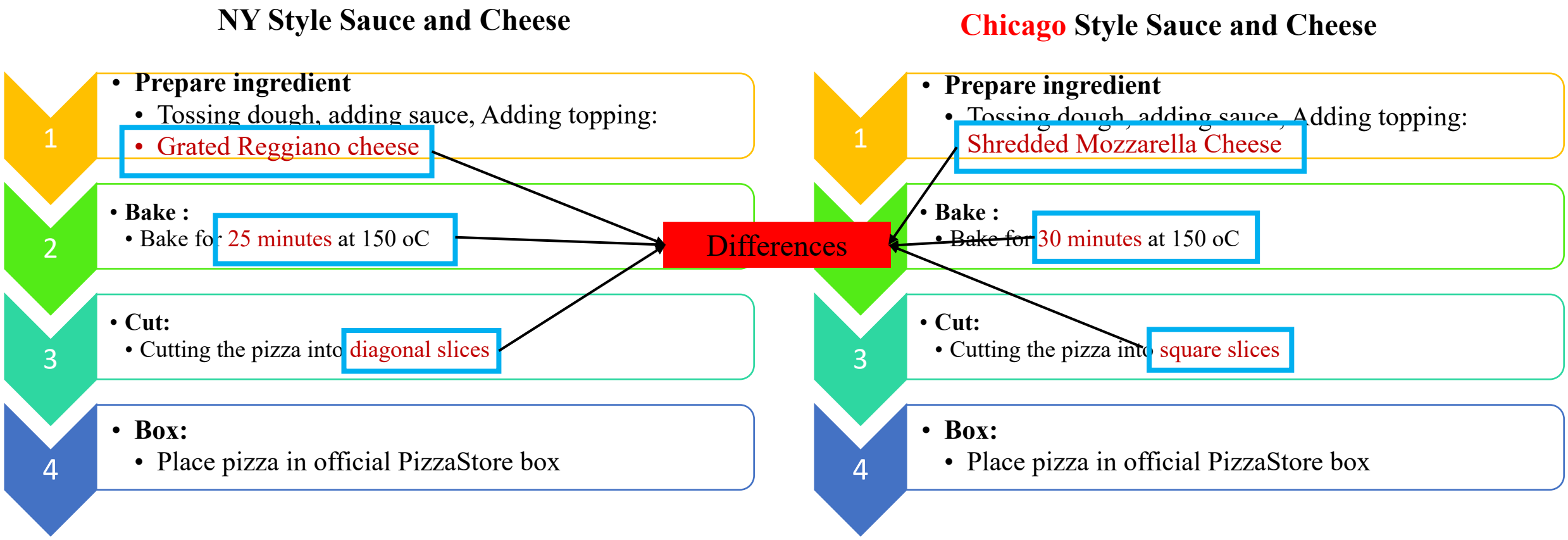
```
1. Prepare ingredient for making NY Style Pizza
 1.1-Tossing dough
 1.2-Adding Sauce
 1.3-Adding topping: Grated Reggiano cheese
2. Bake for 25 minutes at 150 oC
3. Cutting the pizza into diagonal slices
4. Place pizza in official PizzaStore box
```


Good Job
Well Done
U did it
Great

Review: Procedural programming

Previous: Develop a program to make a pizza NY Style Sauce and Cheese

Now, let's us make another pizza type: Chicago Style Sauce and Cheese



Chicago Style Sauce and Cheese

```
ingredientPizza = ["Tossing dough", "Adding Sauce",
                   "Adding topping: Grated Reggiano cheese",
                   "Add topping Shredded Mozzarella Cheese"]
```

Global variables

```
def prepareNYPizza():

def bakeNYPizza():

def cutNYPizza():

def boxNYPizza():
```

It creates problem in large program because we can't determine which global variables (data) are used by which function.

Global variables are accessed by all the function so any function can change its value at any time so all the function will be affected.

```
def prepareChiPizza():
    print("1. Prepare ingredient for making Chicago Style Pizza")
    for  index, value in enumerate([ingredientPizza[0], ingredientPizza[1], ingredientPizza[3]]):
      print(" 1.{}-{}".format(index+1, value))

def bakeChiPizza():
    print("2. Bake for 30 minutes at 150 oC")

def cutChiPizza():
    print("3. Cutting the pizza into square slices")

def boxChiPizza():
    print("4. Place pizza in official PizzaStore box")
```

New requirements => Affect to the existing solution of NY Style Sauce and Cheese

NY Style Sauce and Cheese

```
ingredientPizza = ["Tossing dough", "Adding Sauce",
                   "Adding topping: Grated Reggiano cheese"]

def prepareNYPizza():
    print("1. Prepare ingredient for making NY Style Pizza")
    for  index, value in enumerate(ingredientPizza):
      print(" 1.{}-{}".format(index+1, value))

def bakeNYPizza():
    print("2. Bake for 25 minutes at 150 oC")

def cutNYPizza():
    print("3. Cutting the pizza into diagonal slices")

def boxNYPizza():
    print("4. Place pizza in official PizzaStore box")
```

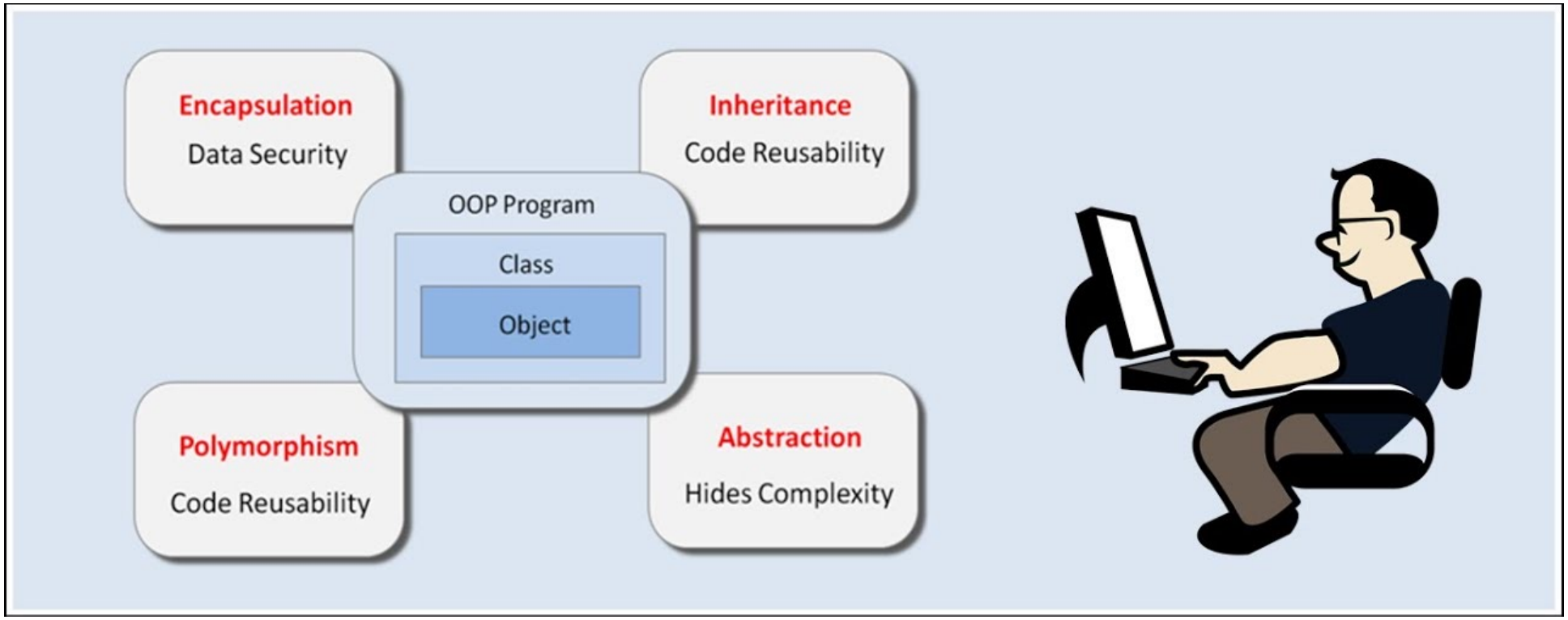
Modified NY Style Sauce and Cheese

```
ingredientPizza = ["Tossing dough", "Adding Sauce",
                   "Adding topping: Grated Reggiano cheese",
                   "Add topping Shredded Mozzarella Cheese"]

def prepareNYPizza():
    print("1. Prepare ingredient for making NY Style Pizza")
    for  index, value in enumerate(ingredientPizza[:-1]):
      print(" 1.{}-{}".format(index+1, value))

def bakeNYPizza():
    print("2. Bake for 25 minutes at 150 oC")

def cutNYPizza():
    print("3. Cutting the pizza into diagonal slices")

def boxNYPizza():
    print("4. Place pizza in official PizzaStore box")
```

New requirements => can not reuse the existing resources (codes)

- **Procedural programming**
- **Discussion**

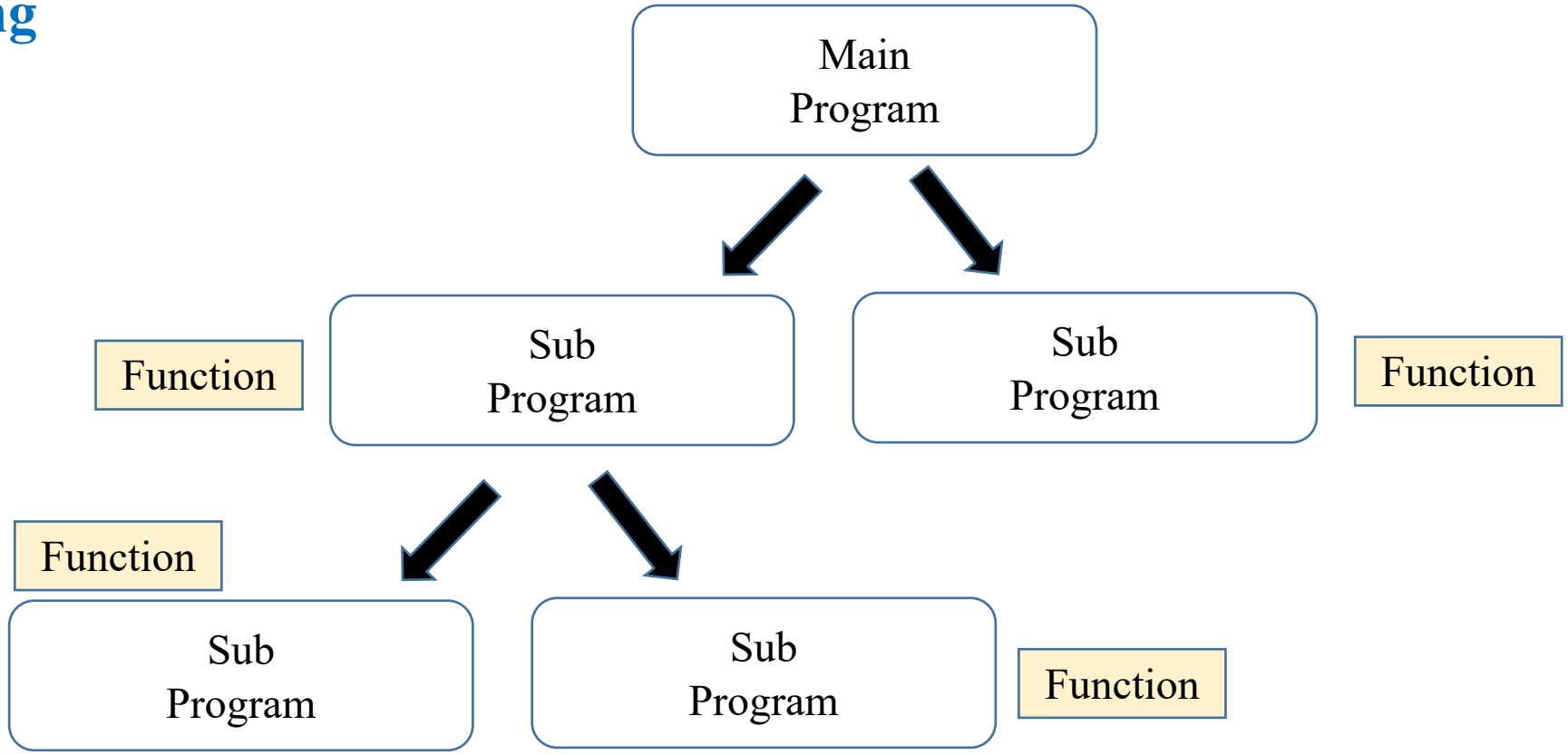
Solution: **Object Oriented Programming**



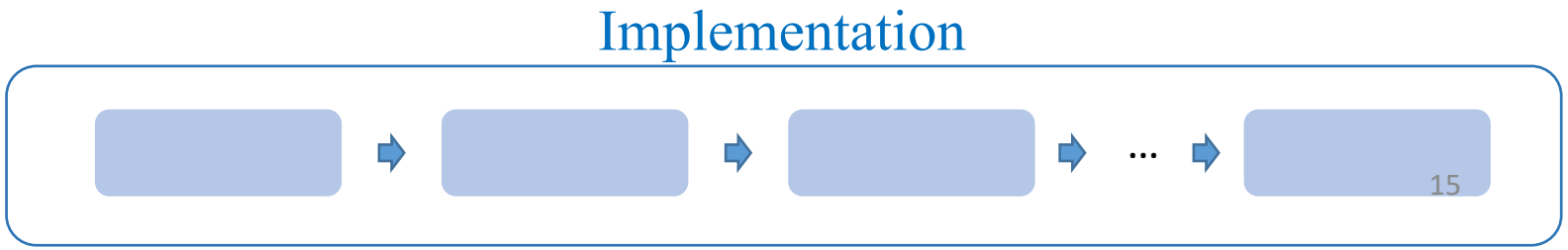
OOP Introduction

Procedural programming

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



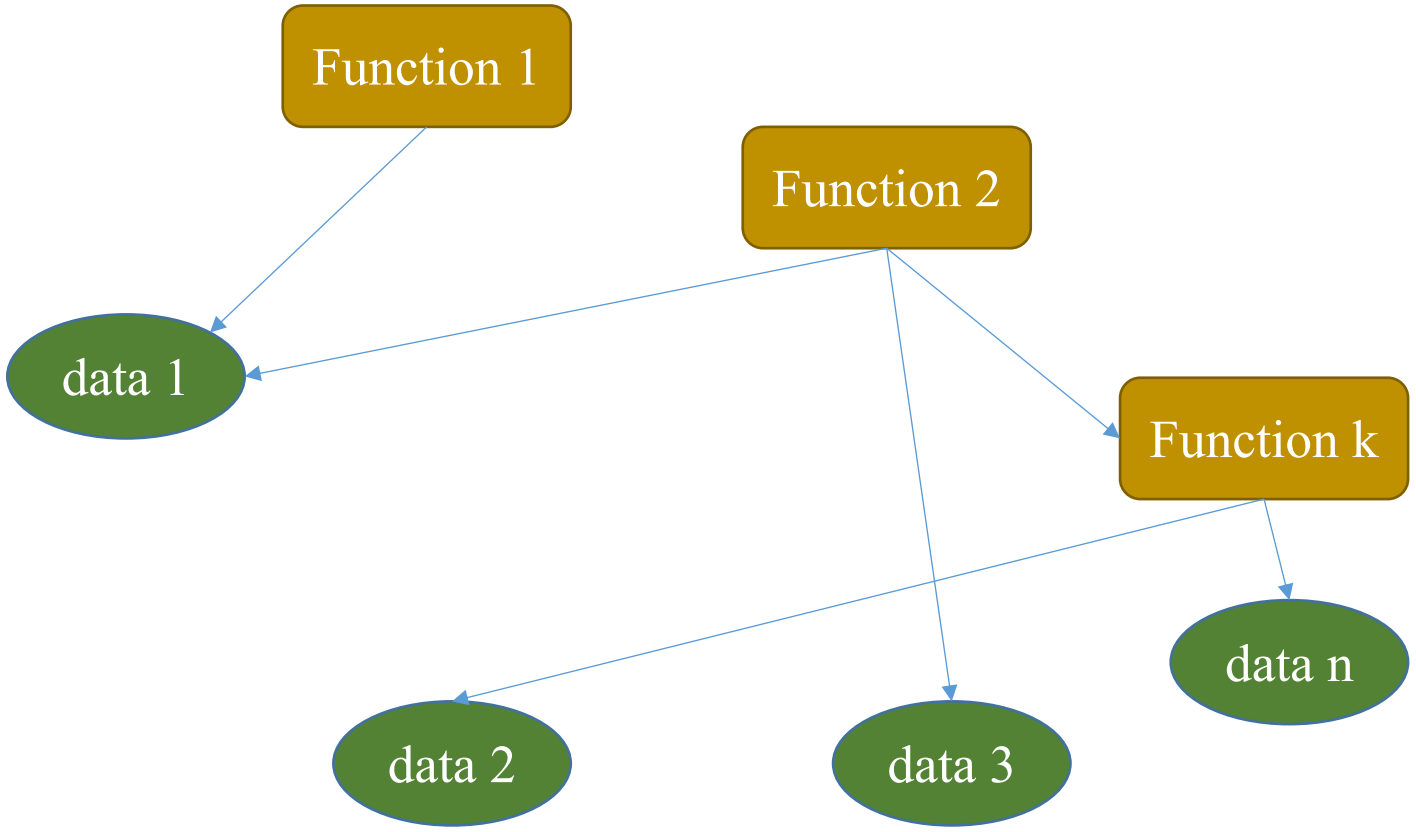
Programming languages: C



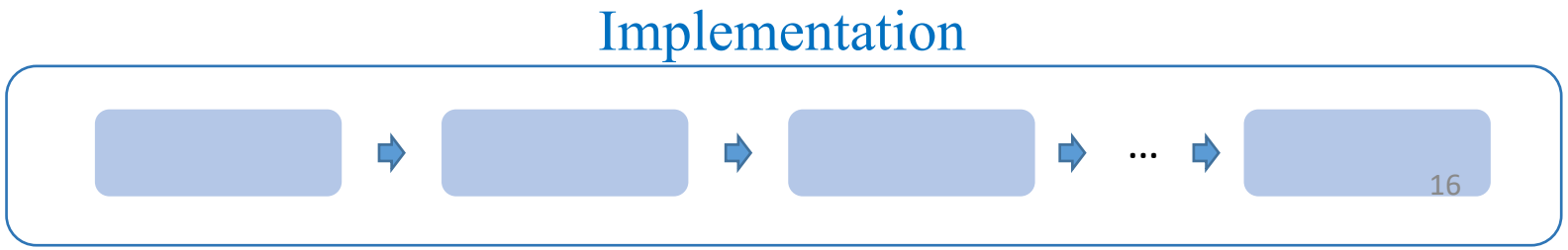
OOP Introduction

Procedural programming

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



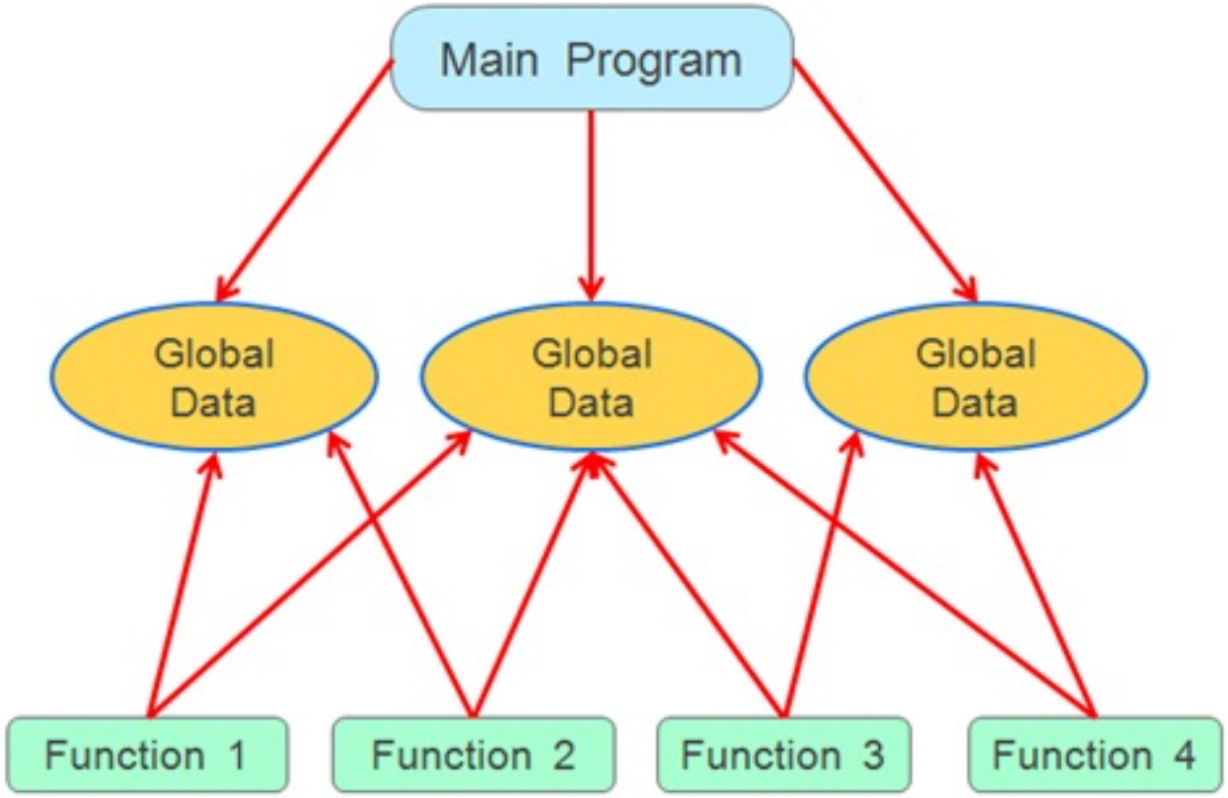
Programming languages: C, Fortran



OOP Introduction

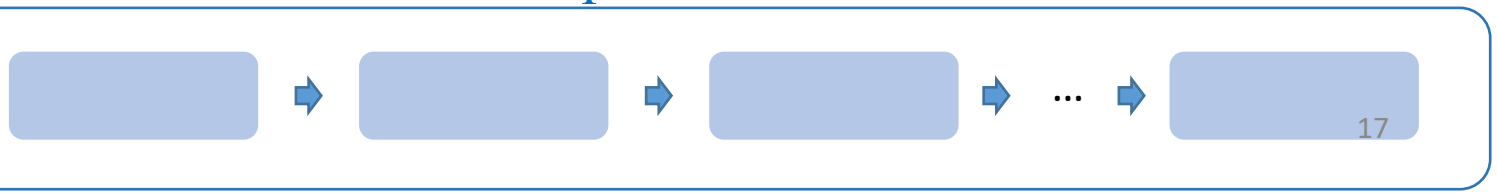
Procedural programming

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



Programming languages: C, Fortran

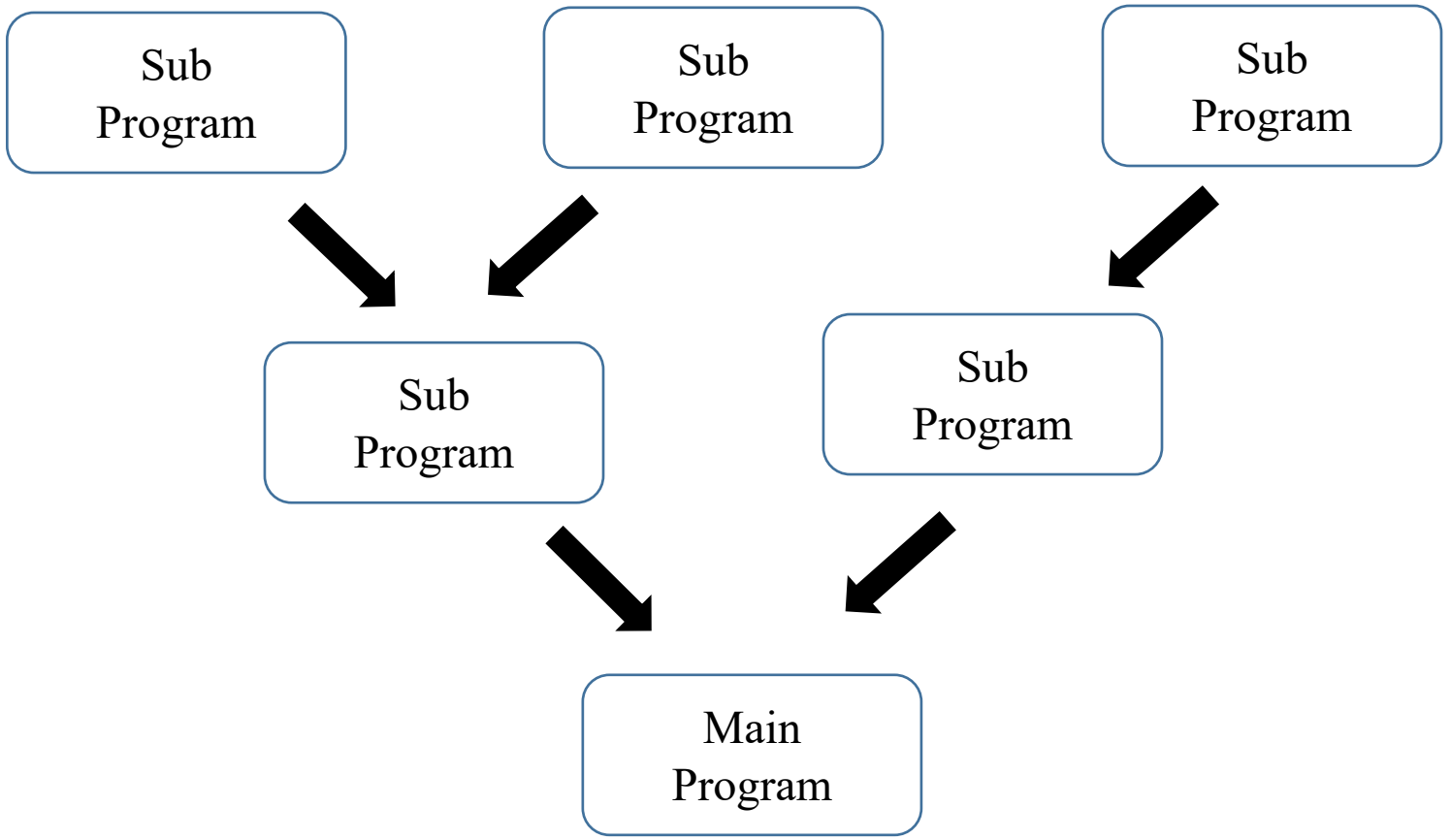
Implementation



OOP Introduction

OOP programming

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

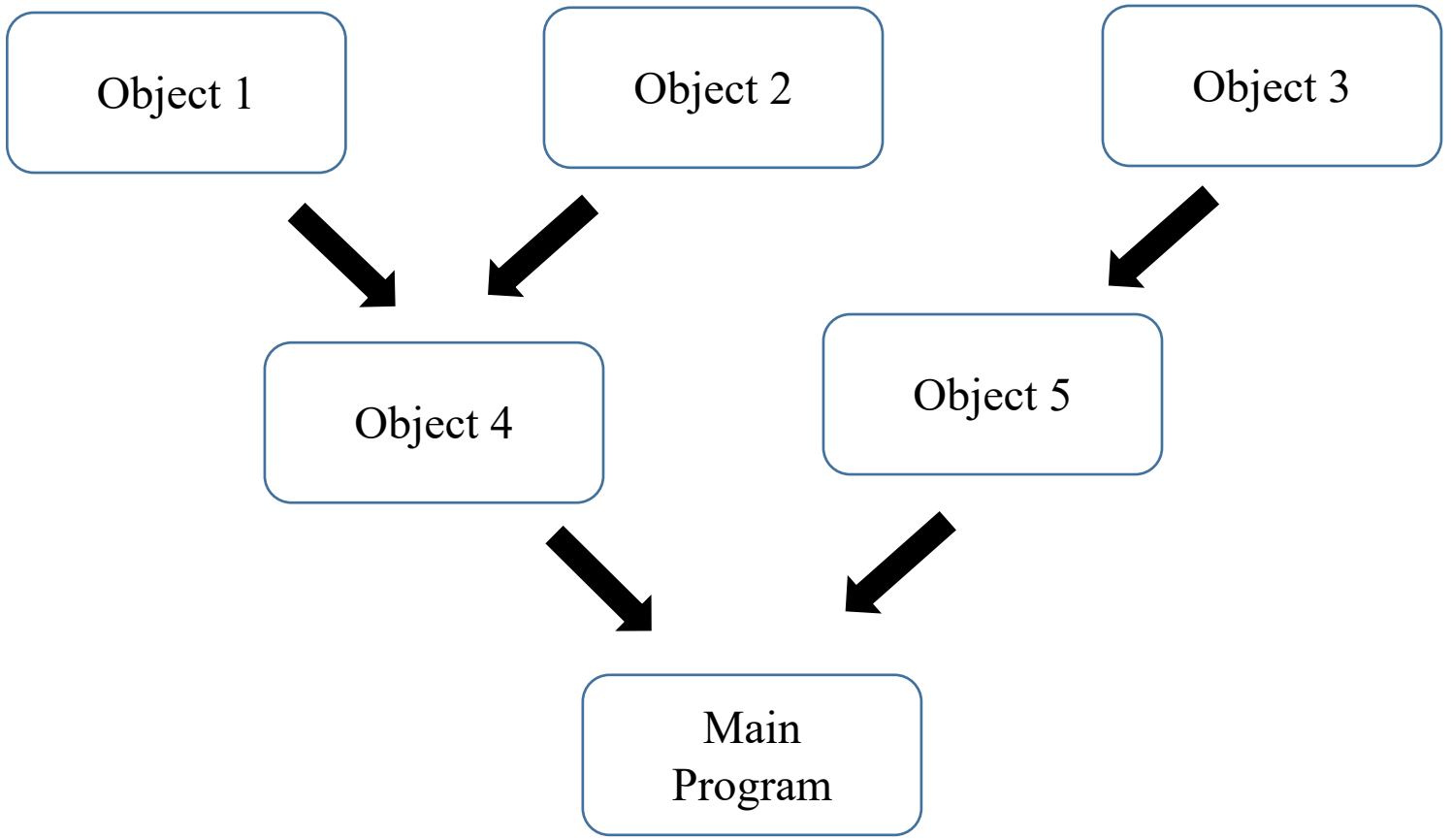


Programming languages:
Java, Python, C++

OOP Introduction

OOP programming

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

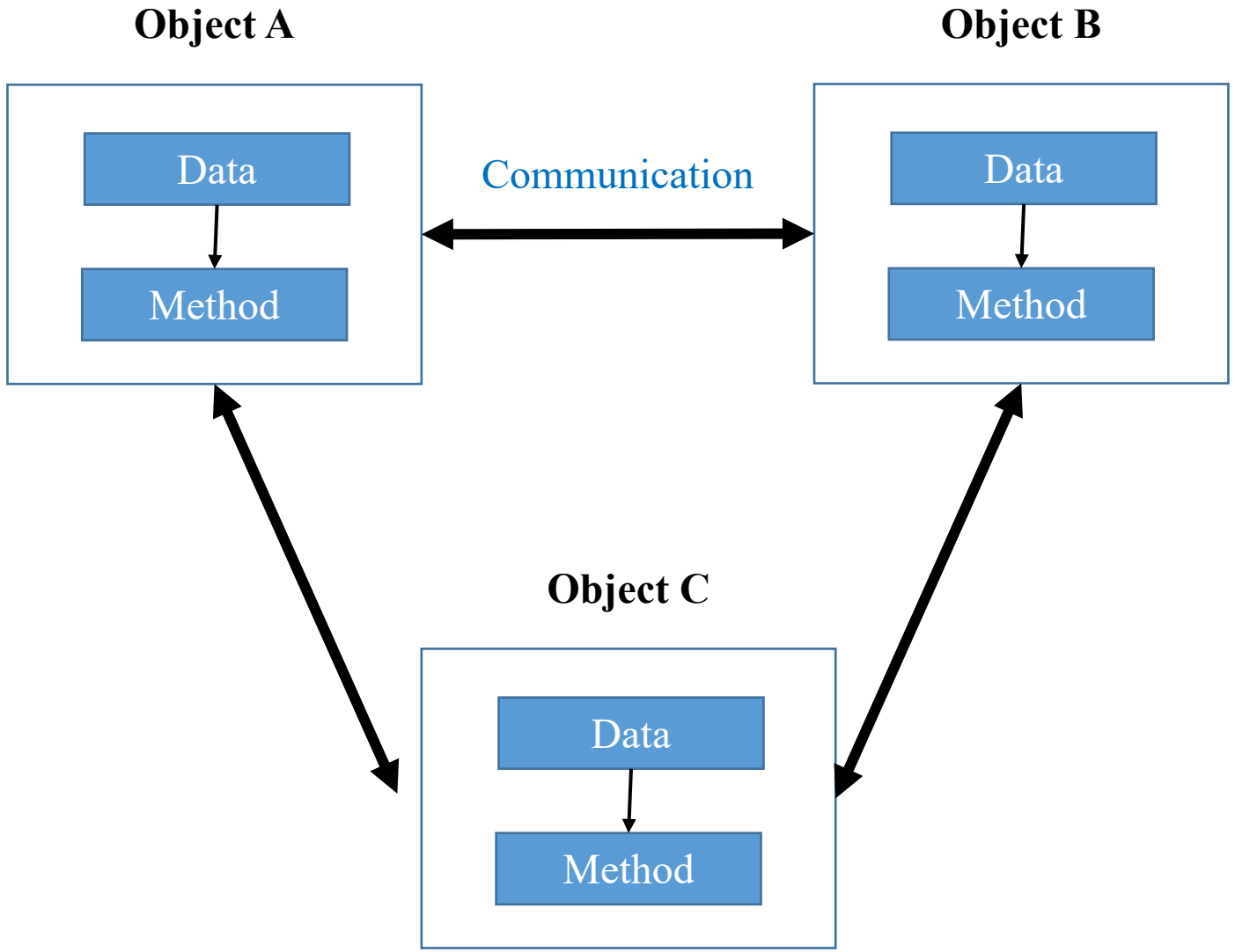


Programming languages:
Java, Python, C++

OOP Introduction

OOP programming

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



What is Object Oriented Programming

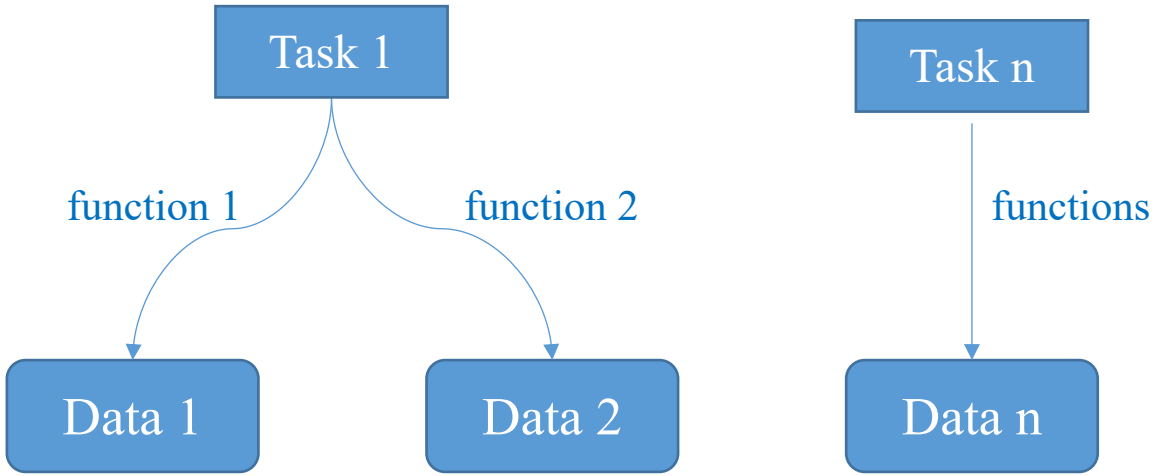
If you were to conduct a fast internet search on what object-oriented programming is, you'll find that OOP is defined as a programming paradigm that relies on the concept of **classes and objects**.

The dictionary meaning of an object is "an entity that exists in the real world", and oriented means "interested in a particular kind of thing or entity".

Introduction

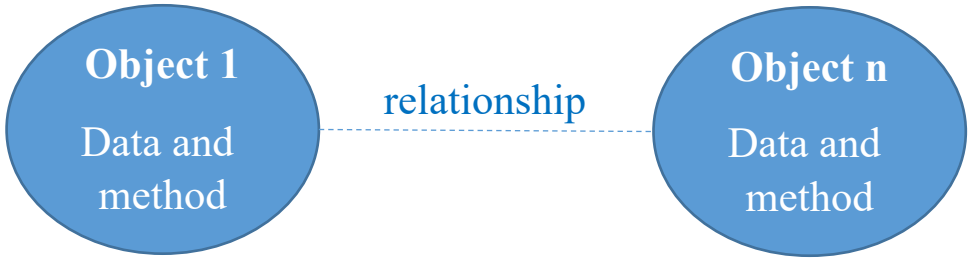
❖ What is OOP?

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

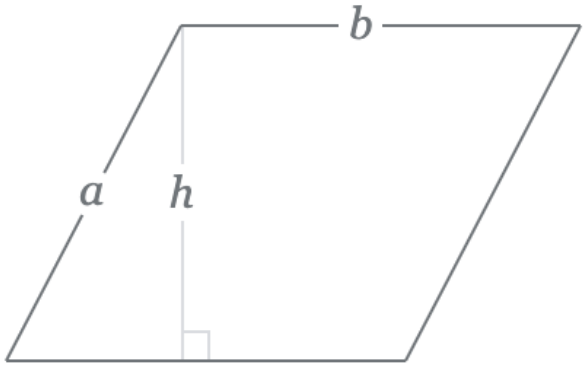
Reusable, security

Develop a program to calculate area of Parallelogram

$$A = b\,h$$

b Base — Enter value

h Height — Enter value

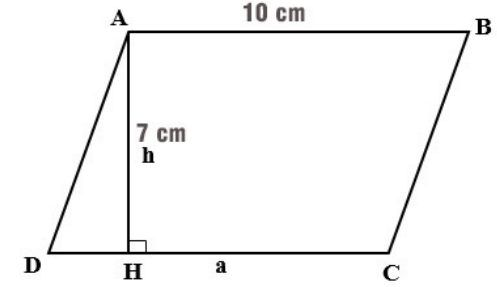


Develop a program to calculate area of Circle

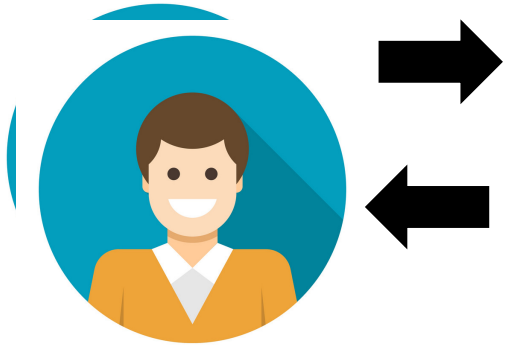
POP principle

Ê bồ **POP** tính giúp mình diện tích Parrellogram có base bằng 10 và height bằng 7 đi

Ôk bồ. Mình open mọi thứ, bạn cần gì thì cứ lấy. Nếu muốn sửa thông tin cũng được luôn nha.

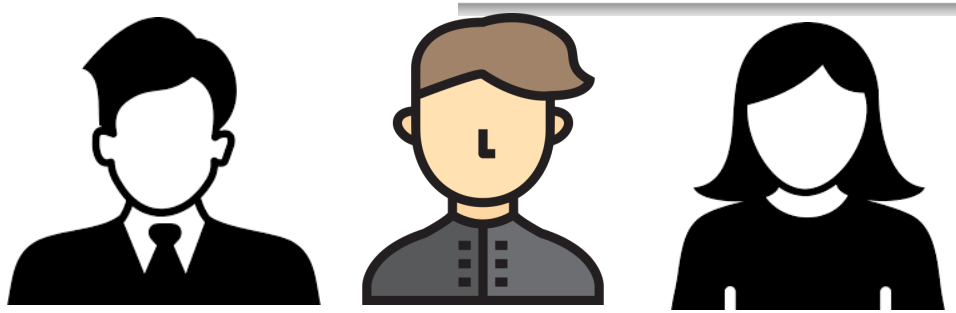


Diện tích hình bình hành bằng bao nhiêu?



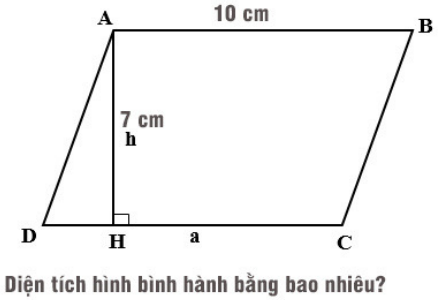
```
base = 10
height = 7

def computeAreaParallelogram():
  return base * height

area = computeAreaParallelogram()
print("Parallelogram's Area: {} cm2".format(area))
```

```
Parallelogram's Area: 70 cm2
```



Develop a program to calculate area of Circle

OOP principle



Diện tích hình bình hành bằng bao nhiêu?

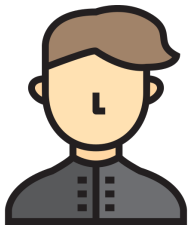
Ê bồ OOP tính giúp mình diện tích Parrellogram có base bằng 10 và heigh bằng 7 đi

Xin lỗi bồ nha, mình chỉ phục vụ các bạn nữ xinh đẹp thôi. Diện tích tự tính đi nha

Object: **Parallelogram**

It has information: base and height

It know how to calculate area : base * height

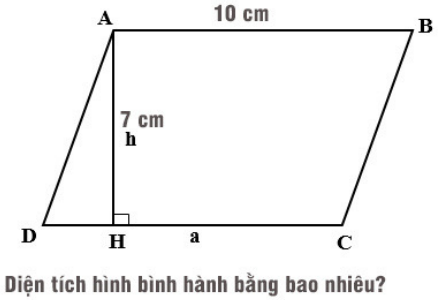


Ê bồ tèo OOP, cho mình biết thông tin base và height của bạn đi

Xin lỗi bạn nha. Mình không cho phép truy xuất trực tiếp thông tin base và height.

Develop a program to calculate area of Circle

OOP principle



Diện tích hình bình hành bằng bao nhiêu?

Ê bồ OOP tính giúp mình diện tích Parrellogram có base bằng 10 và heigh bằng 7 đi

Ok bồ, bạn nữ là mình phục vụ 24/24 cần gì có đó nha

Object: **Parallelogram**

It has information: base and height

It know how to calculate area : base * height

Structured of Object-Oriented Programming

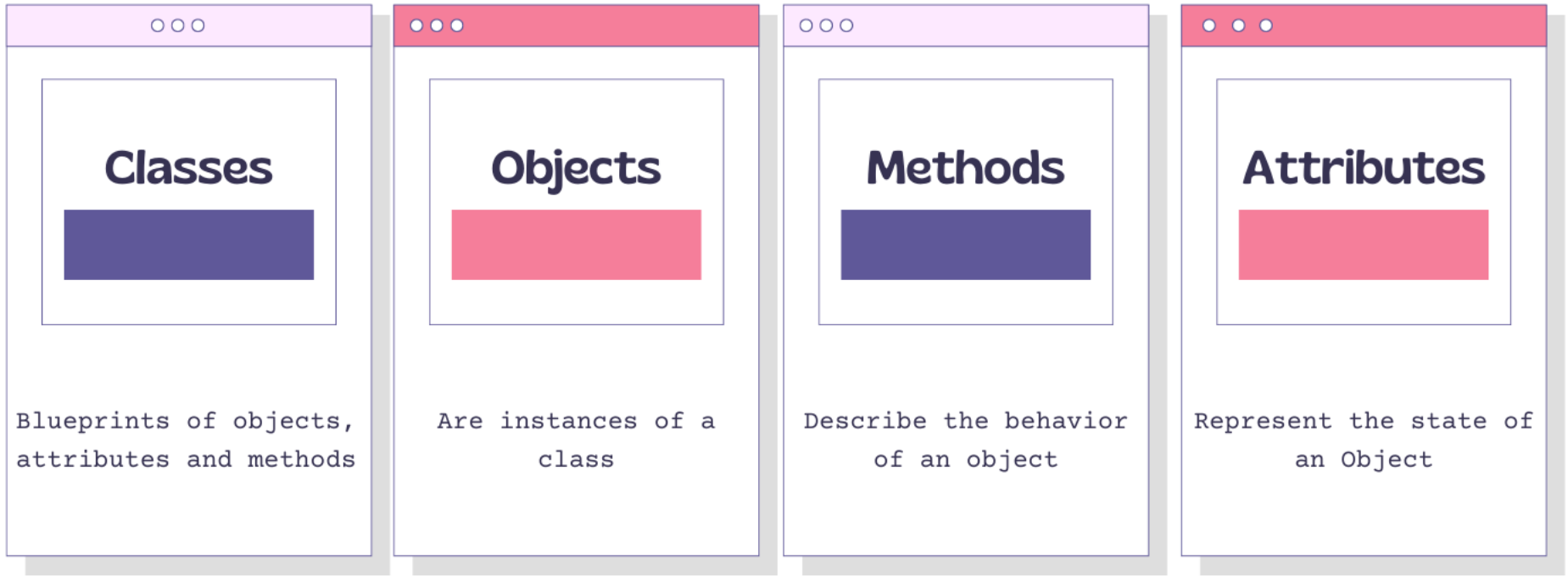
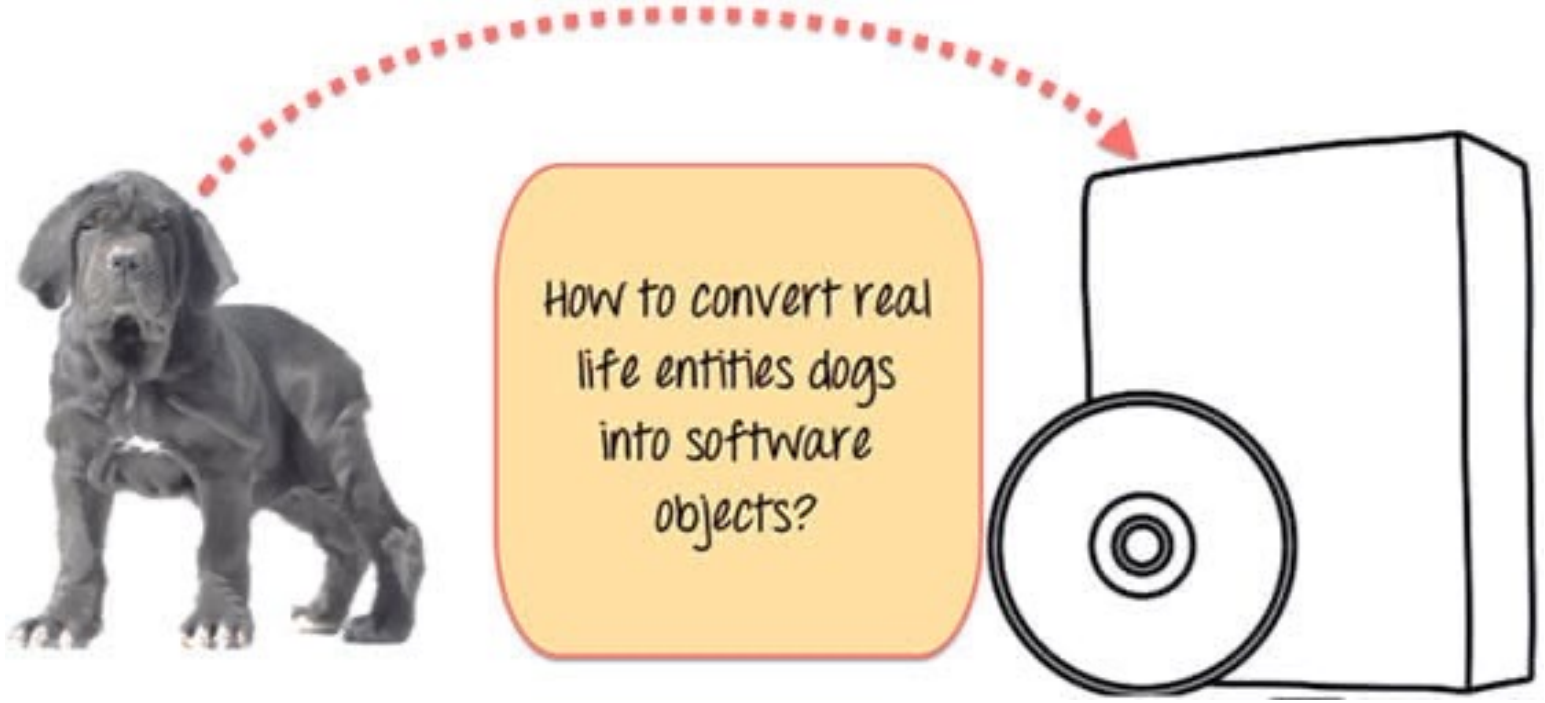


Image Credit: https://www.freecodecamp.org/news/what-is-object-oriented-programming/

Outline

- Introduction to OOP
- Classes and Objects
- Object Relationships (Composition, Aggregation)
- Summarize
- Further reading

Example


How to convert real life entities dogs into software objects?

Various Types of Dogs

Image Credit: https://www.thevetexpert.com/

Common Characteristics

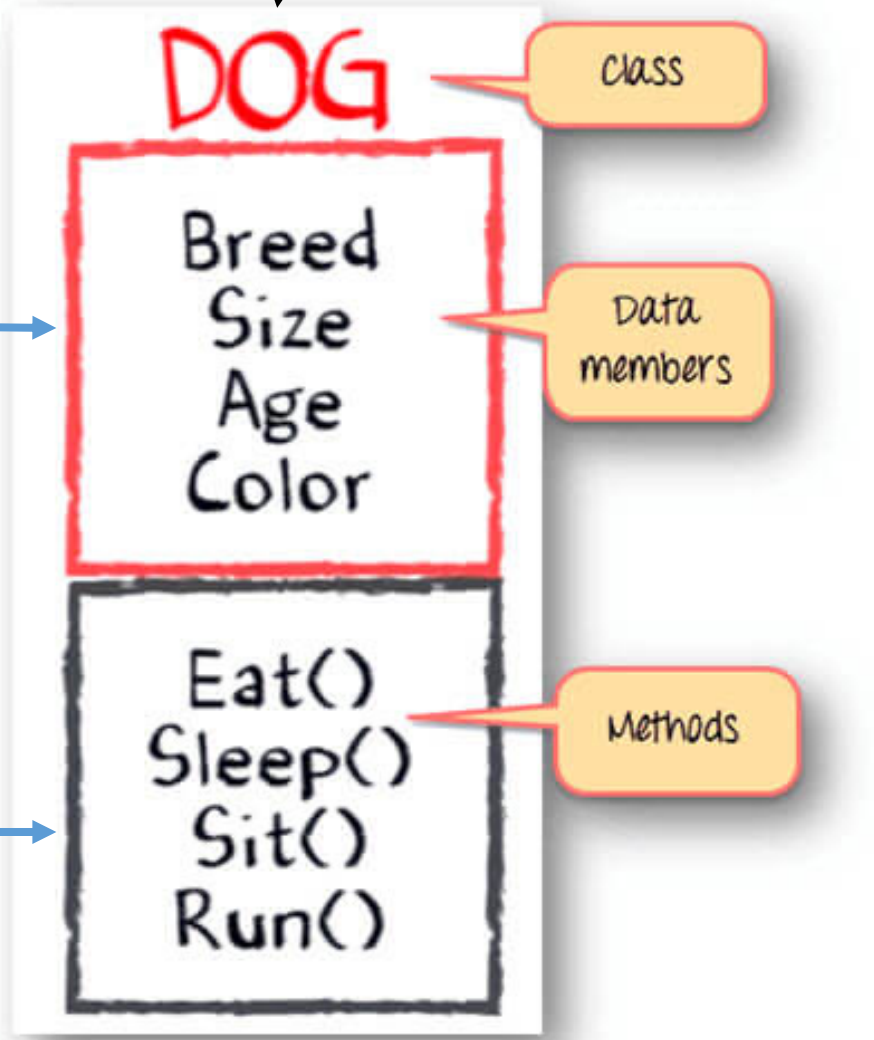
- Breed
- Size
- Age
- Color

Common Actions

- Eat
- Sleep
- Sit
- Run

Class in OOP

Class name



Common Characteristics

- Breed
- Size
- Age
- Color

Common Actions

- Eat
- Sleep
- Sit
- Run

Example

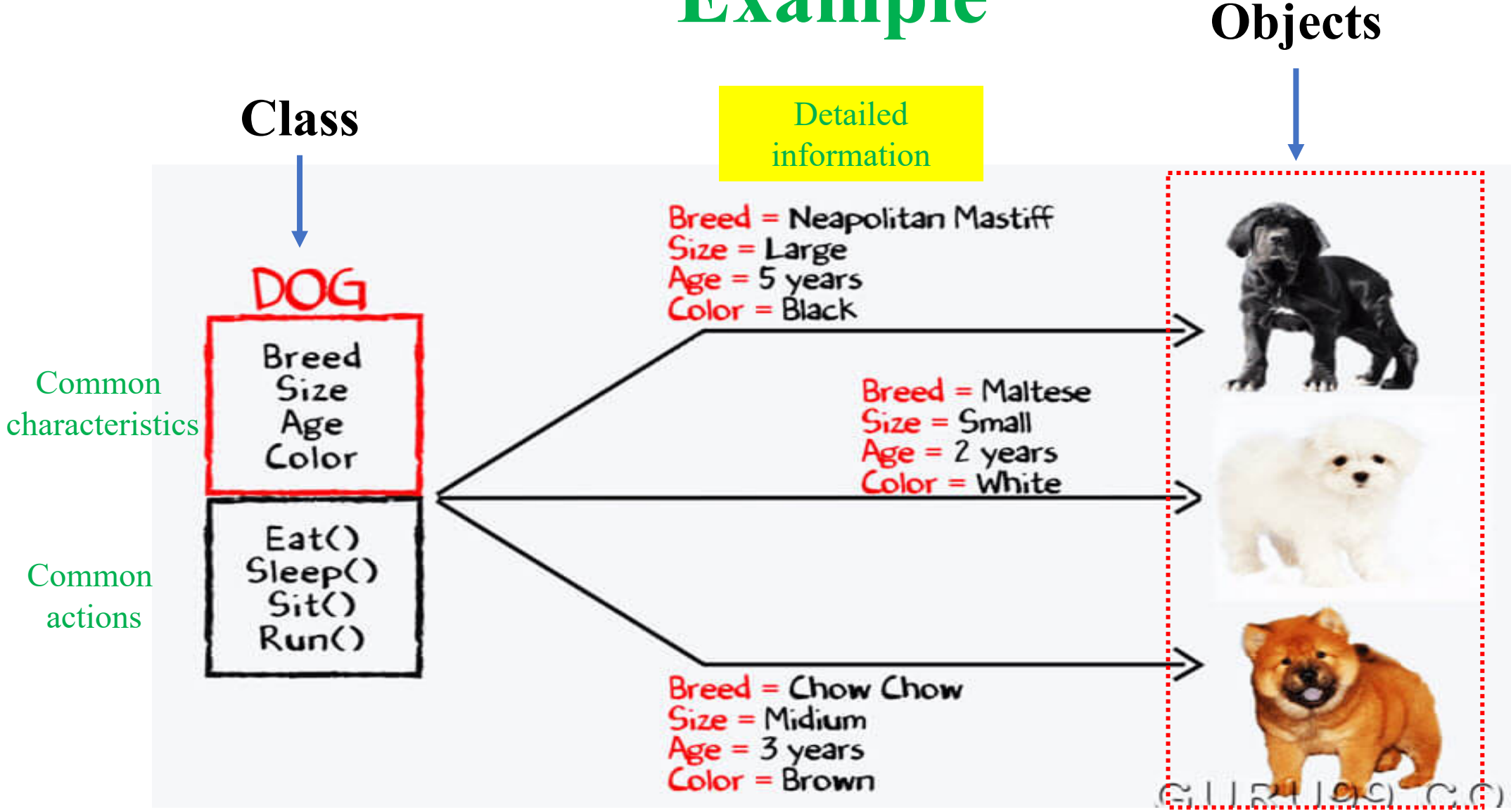


Image Credit: Guru99.com

Class and Object

- ✓A class is a blueprint for the object. Before we create an object, we first need to define the class.
- ✓We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.
- ✓Since many houses can be made from the same description, we can create many objects from a class.

(A)

(B)

Class and Object

❖ Classes and Objects

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

Fruit

Strawberry
Apple
Banana

Class and Object

❖ Classes and Objects

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

Animal	Cat Deer Tiger

Class and Object

❖ 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

Class and Object

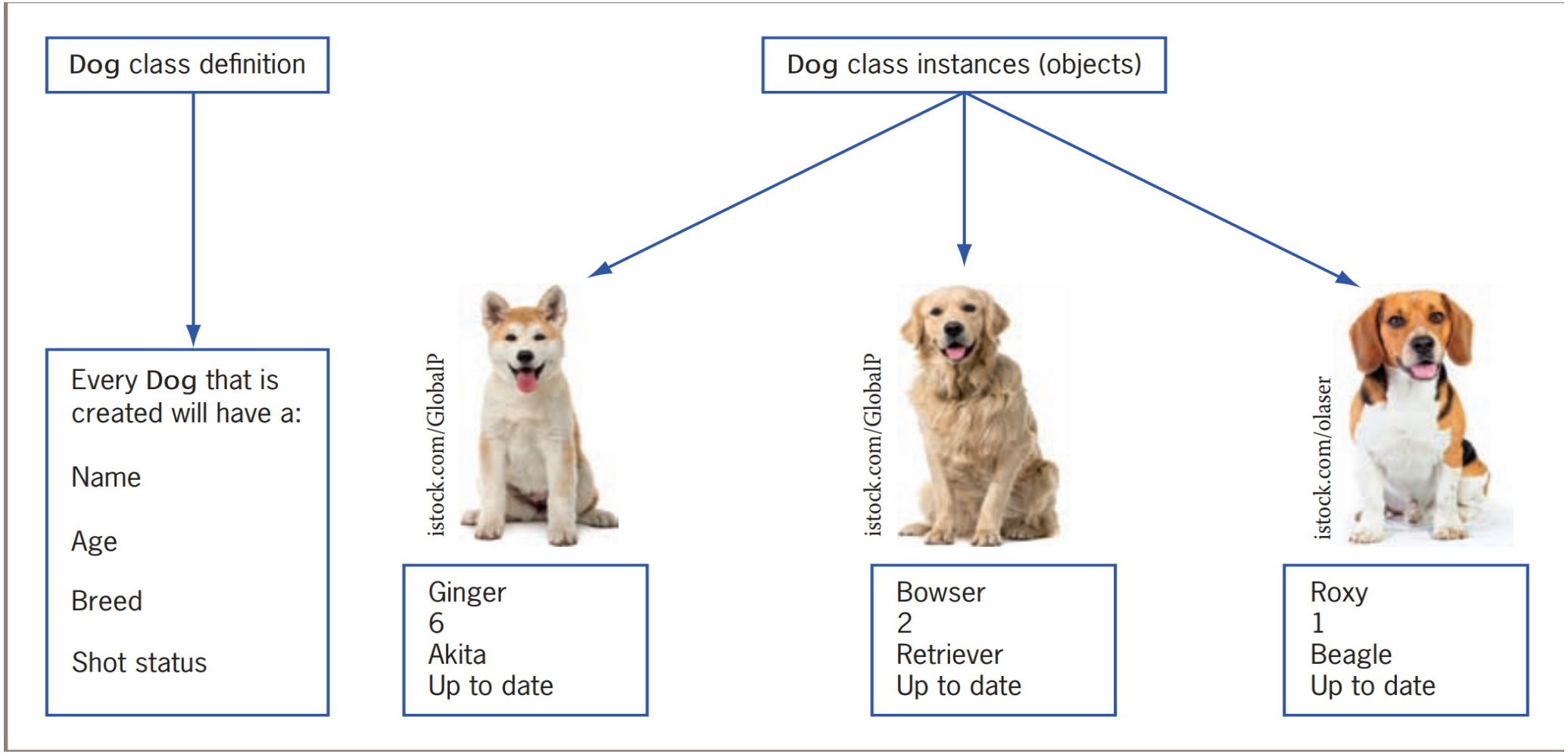
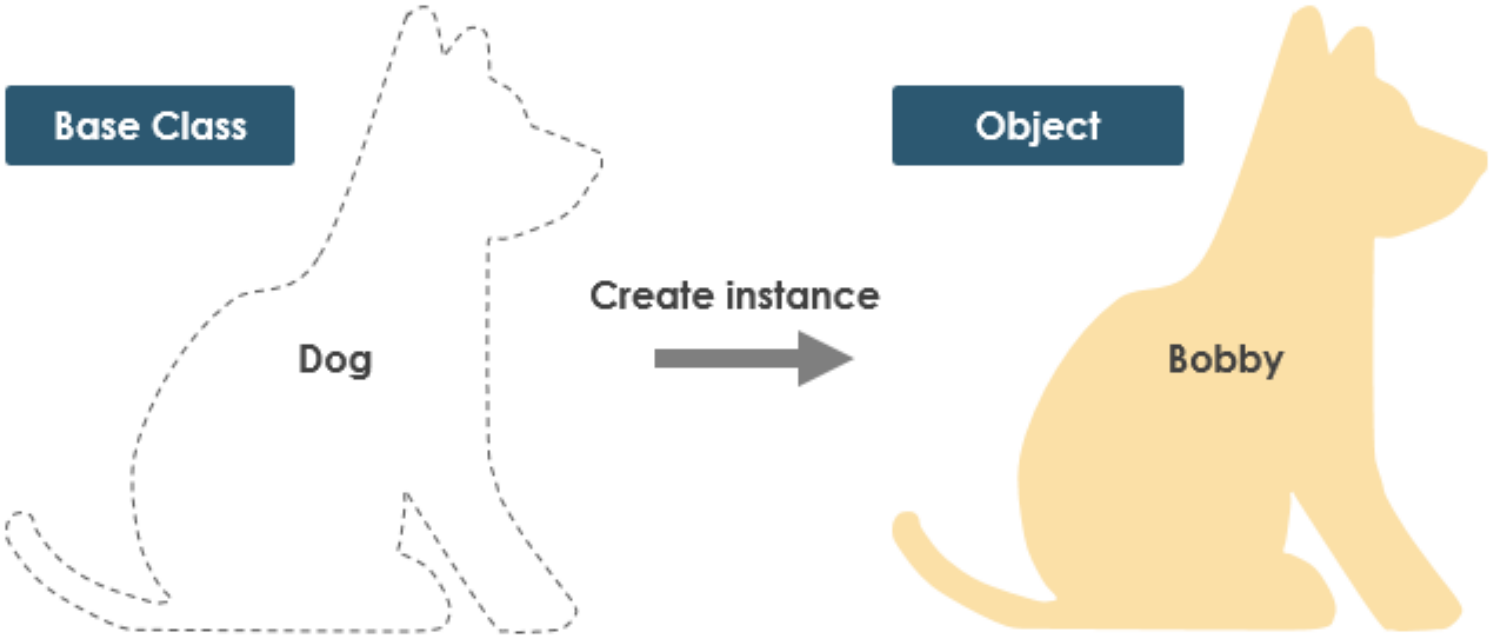


Image Credit: Guru99.com

Class and Object

❖ Classes and Objects

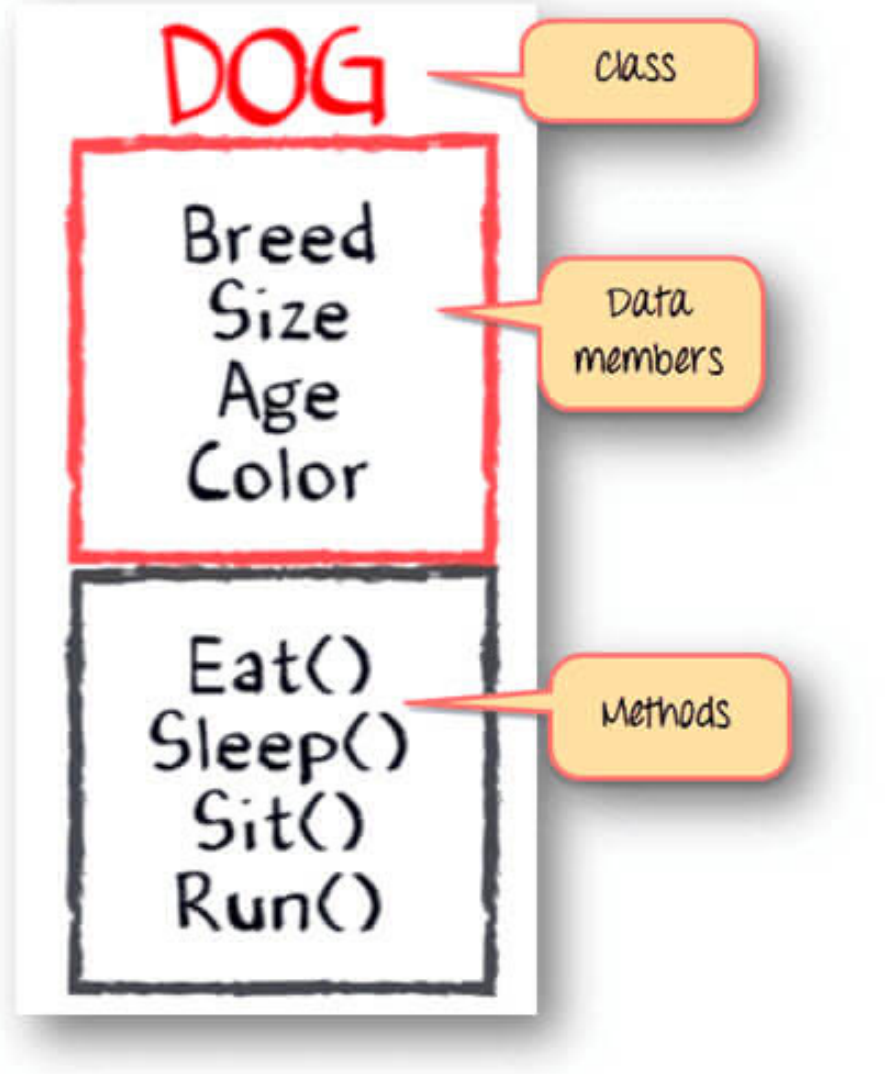


Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

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/

Review


DOG
Class
Breed
Size
Age
Color
Data members
Eat()
Sleep()
Sit()
Run()
Methods

Glance on creating Class in Python

What is seft.breed = newBreed?

```
1  #Declare class
2  class Dog:
3
4      # __init__ is known as the constructor
5      def __init__(self, newBreed):
6          self.breed = newBreed
7
8  # Object instantiation
9  rodger = Dog("USA")
10 tommy = Dog("Japan")
```

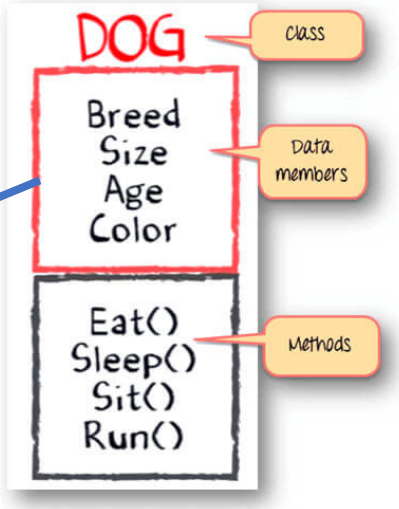
Class name

__init__() function is called every time an object is created.
=> This is call a constructor function of a class

Create Objects

Let's Add More Attributes to Class

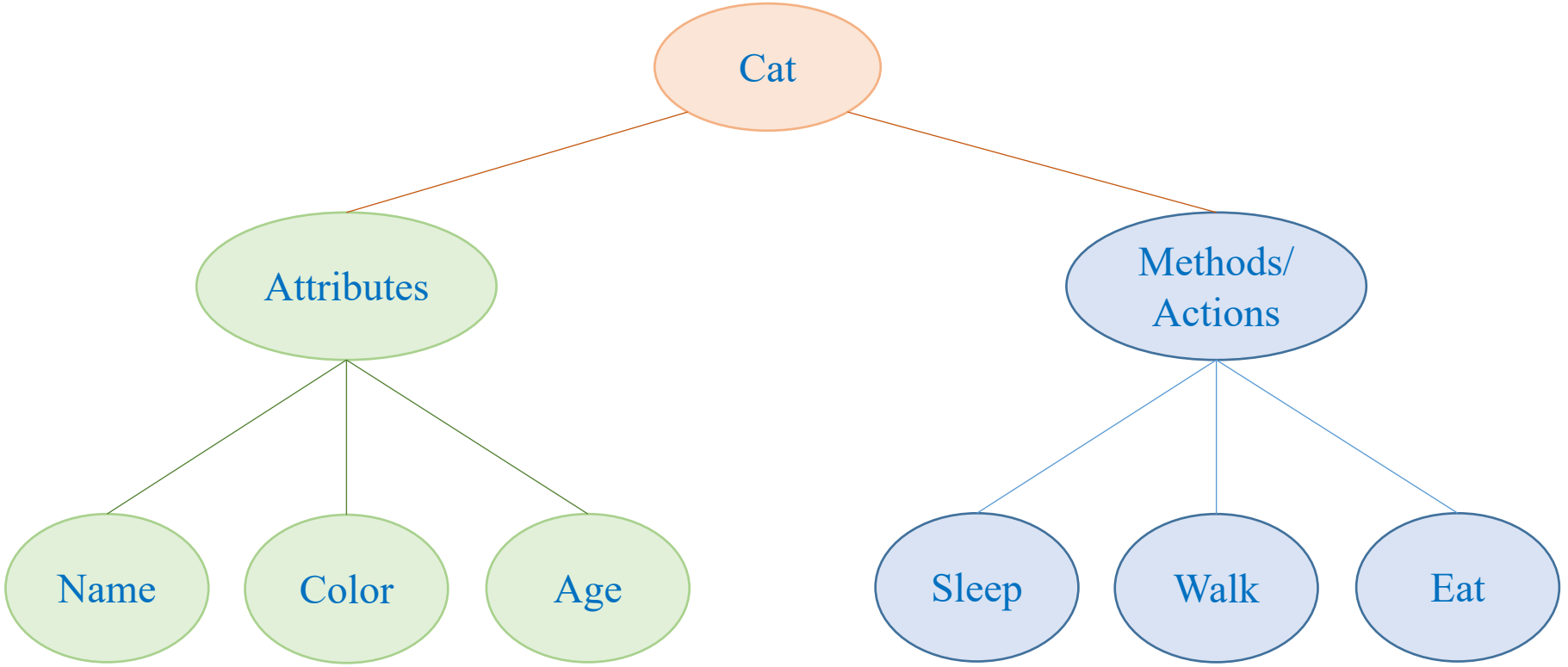
```
1  #Declare class
2  class Dog:
3
4      # __init__ is known as the constructor
5      def __init__(self, newBreed, newSize, newAge, newColor
           ):
6          self.breed = newBreed
7          self.size = newSize
8          self.age = newAge
9          self.color = newColor
10
11
12 # Object instantiation
13 rodger = Dog("USA", 10, 2, "Brown")
14 tommy = Dog("Japan", 8, 1, "Black")
```



Deeper in Class and Object

Classes and Objects

❖ Abstract view



Class Diagram

Cat
name color age
sleep walk eat

Class Diagram

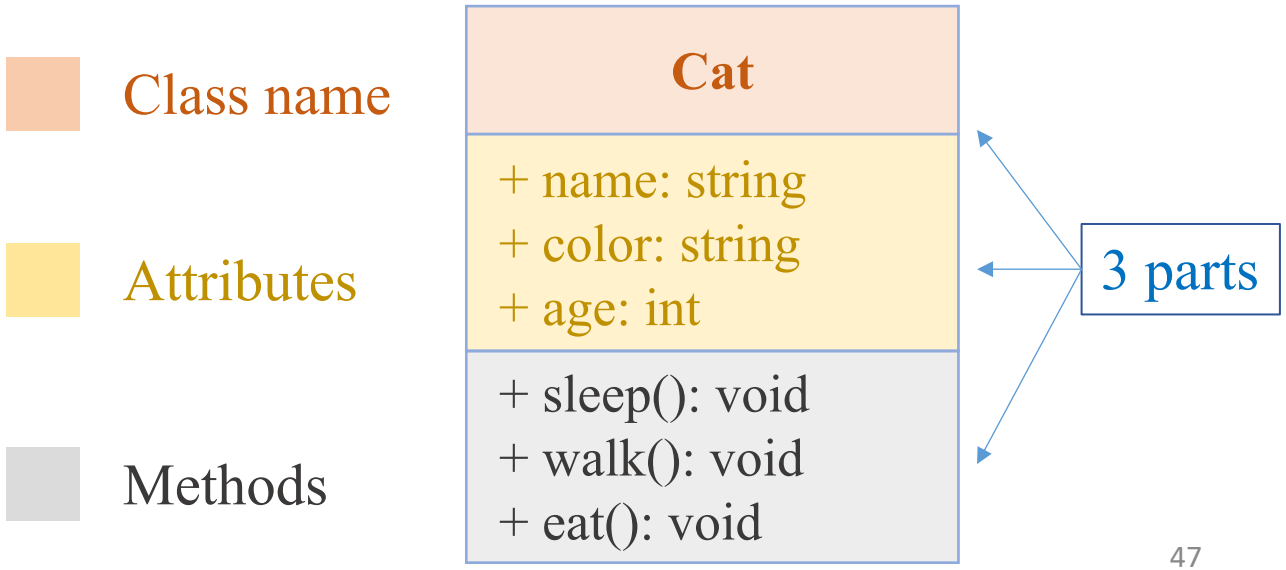
❖ Describe the structure of a system

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.



Objects and Classes

❖ Create a method 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.

```
object() is shorthand for object.__call__()
```

```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)
```

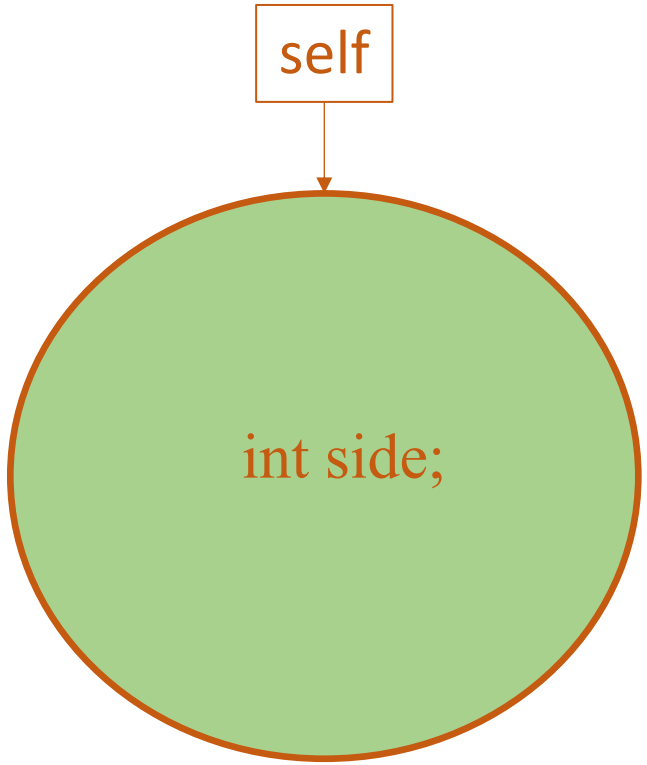
```
0
0
5
7
```

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

Cat
+ name: string + color: string + age: int
...

```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

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

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

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

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

```
1  class Cat:
2      def __init__(self, name):
3          self.__name = name
4
5      def getName(self):
6          return self.__name
7
8      def setName(self, name):
9          self.__name = name
10
11 # test
12 cat = Cat('Calico')
13 print(cat.getName())
14
15 cat.setName('Japanese Bobtail')
16 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

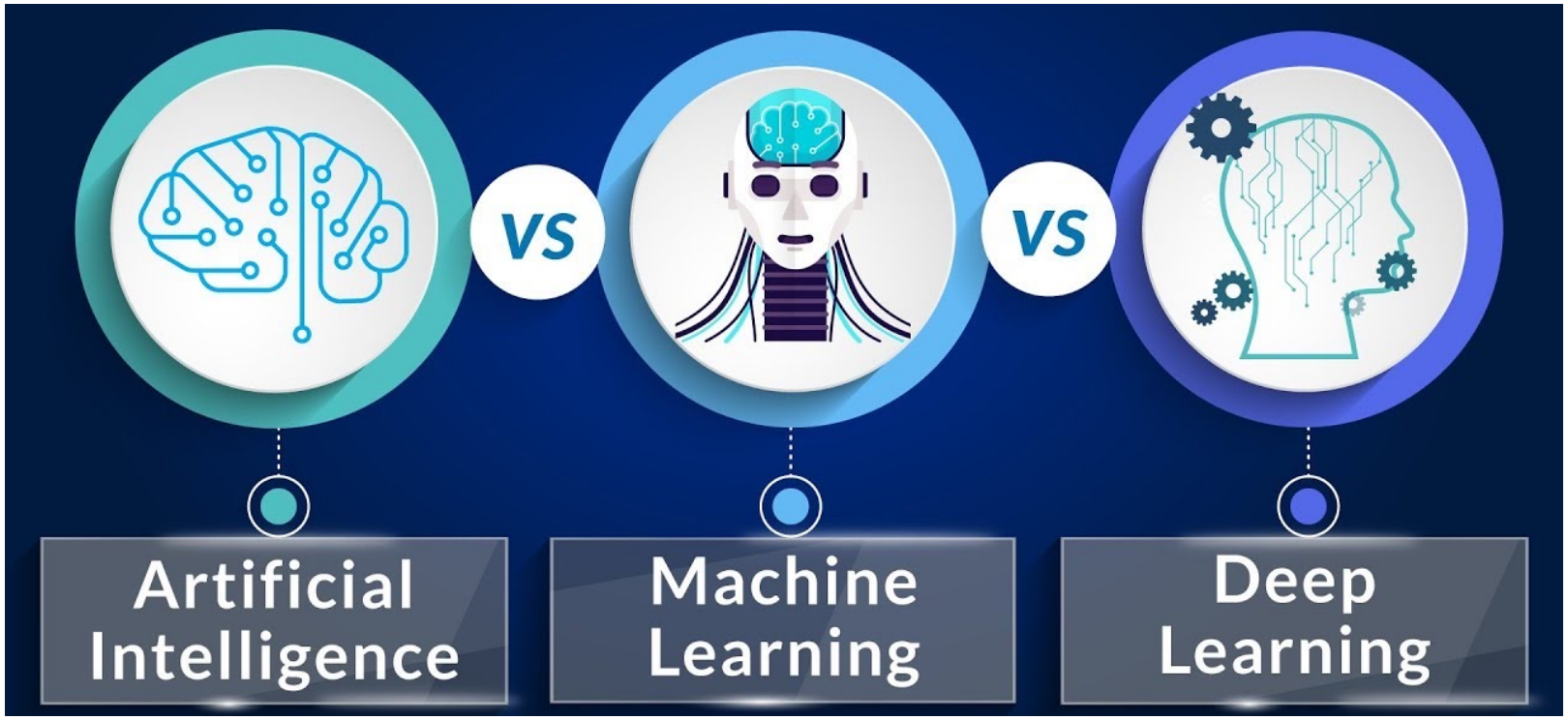
A cat includes a name, a color, and an age.

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

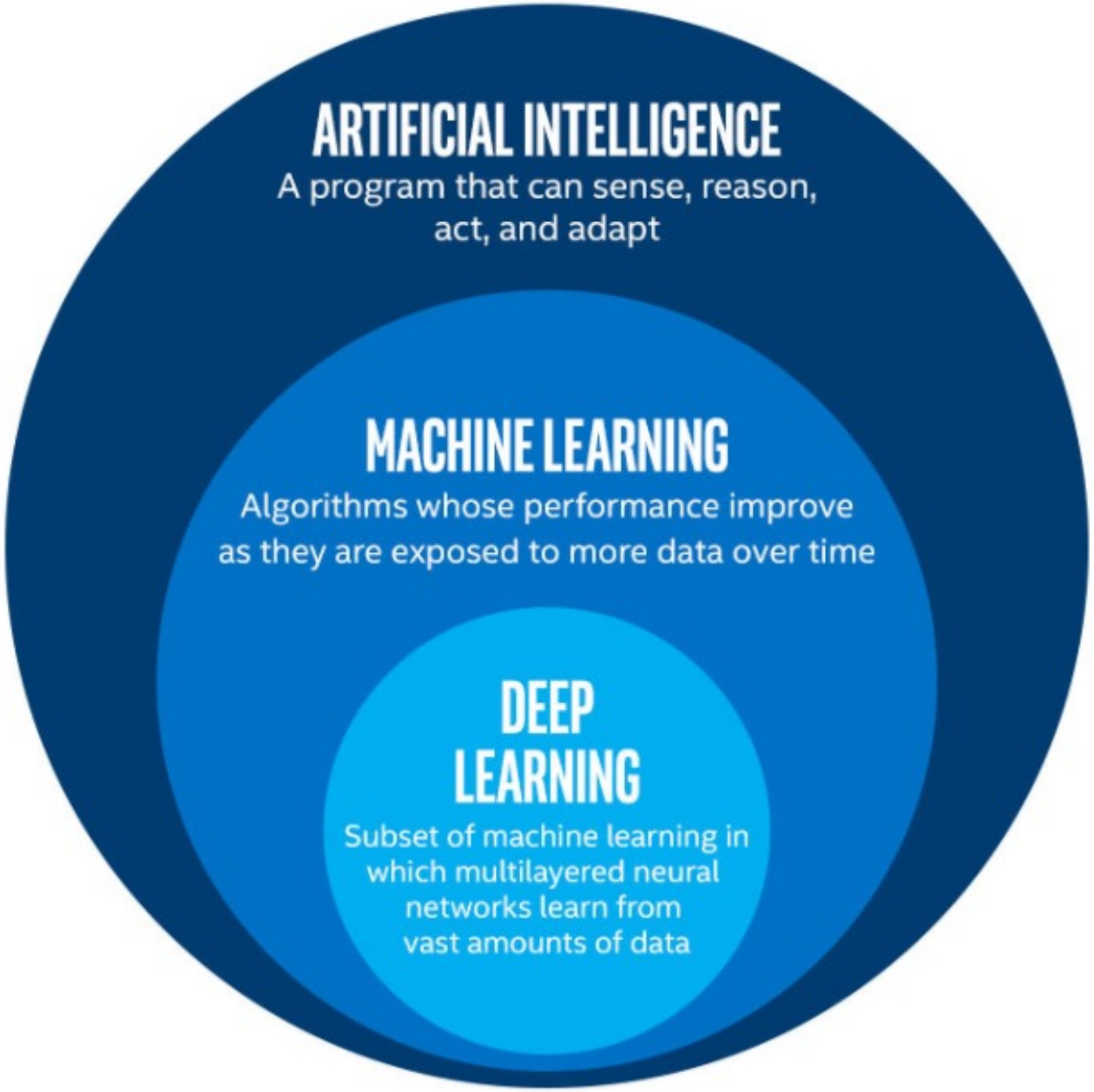
Outline

- Introduction
- Classes and Objects
- Object Relationships (Composition, Aggregation)
- Summarize
- Further reading

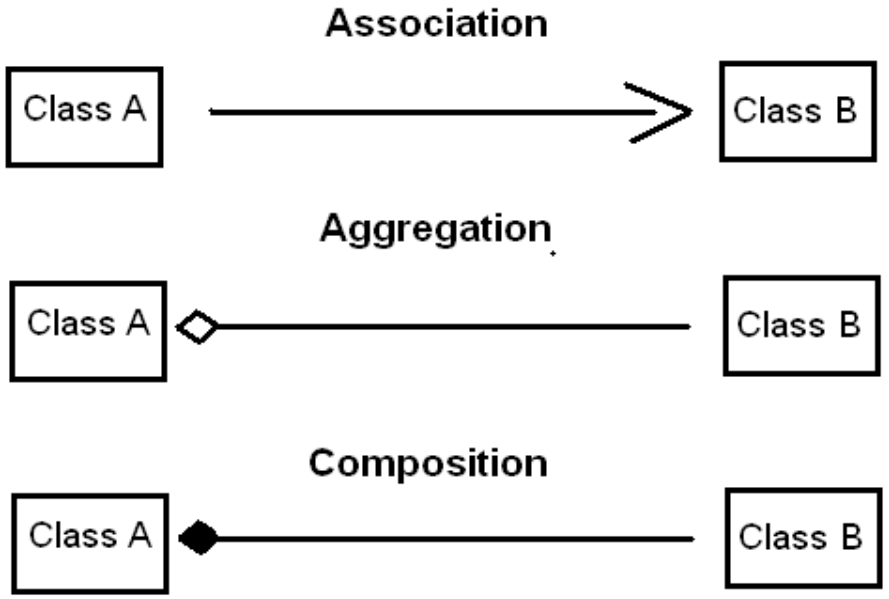
Happy Minute



Happy Minute



Object Relationships



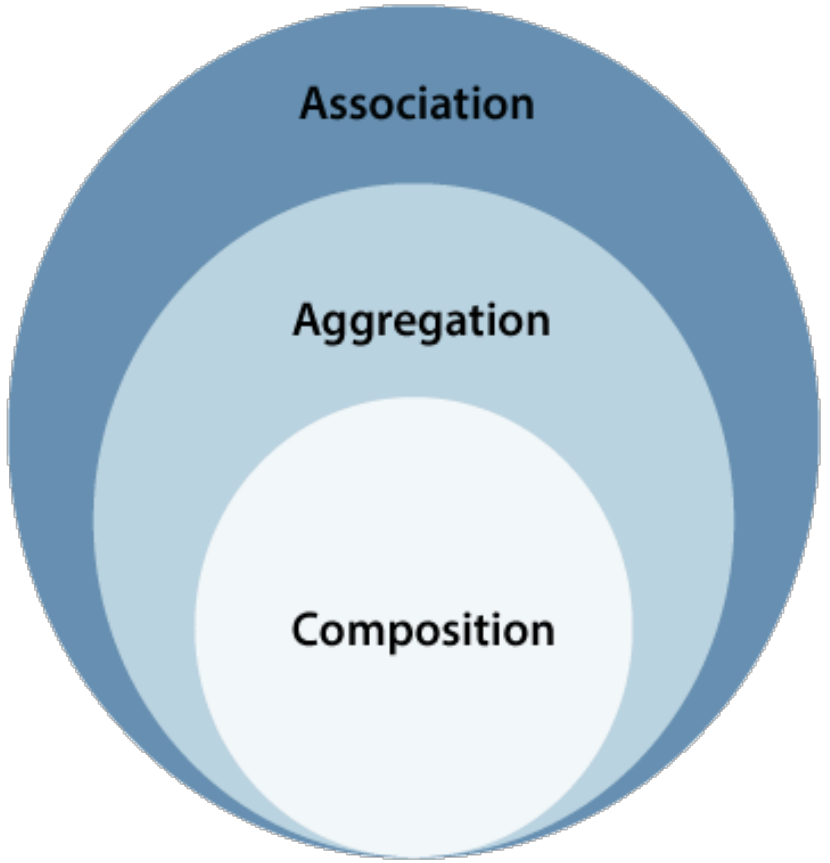
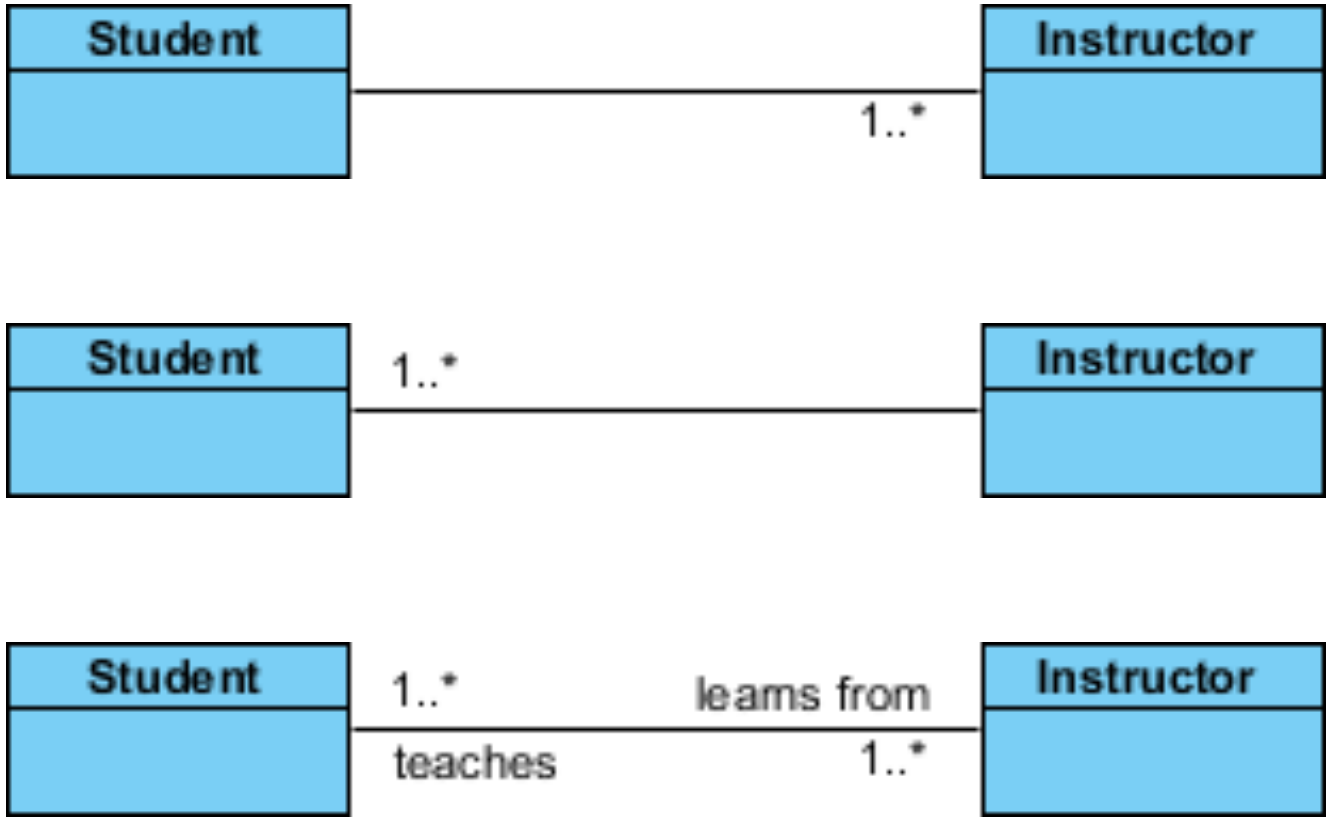


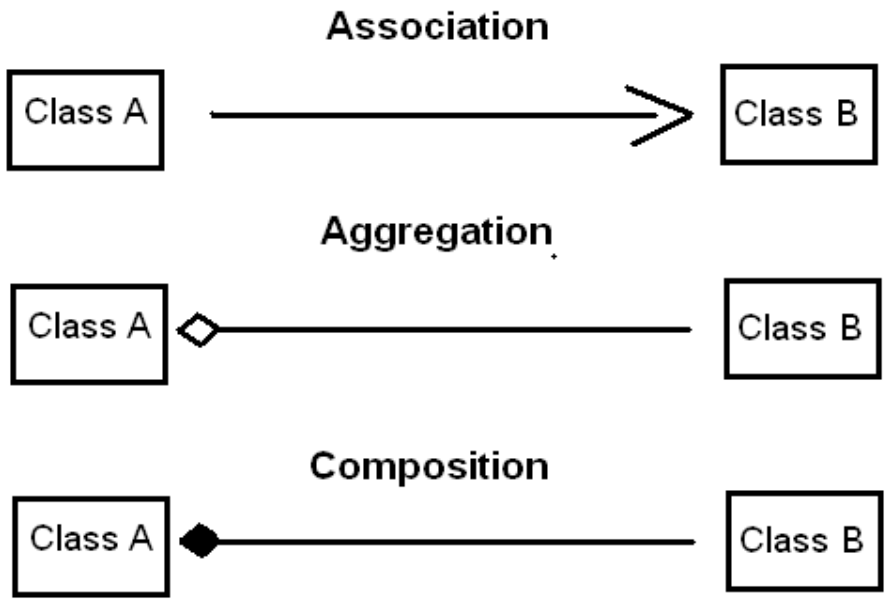
Image Credit:https://softwareengineering.stackexchange.com/

Association

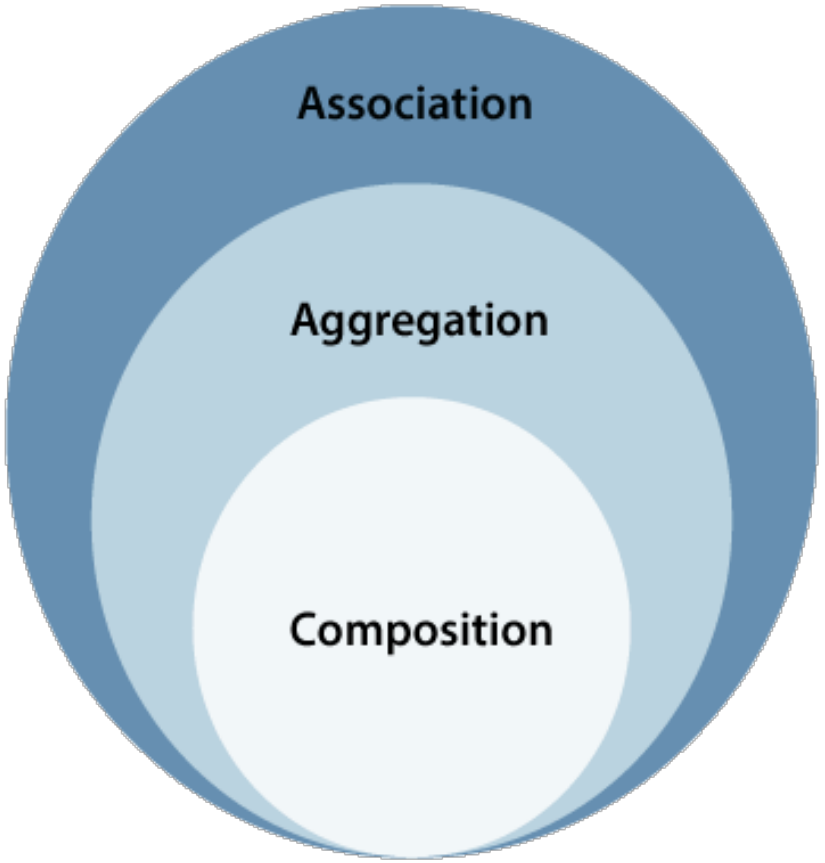
If two classes in a model need to communicate with each other, there must be a link between them, and that can be represented by an association (connector).



Object Relationships



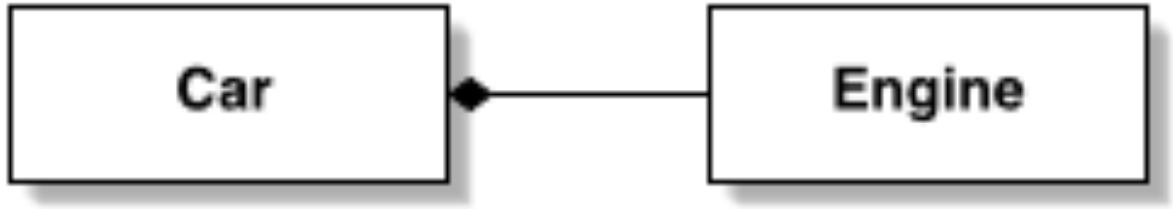
Aggregation and **Composition** are subsets of association meaning they are **specific cases of association.**



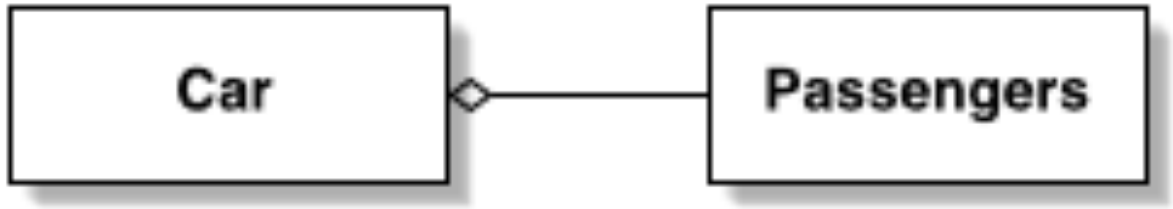
In both aggregation and composition object of one class "owns" object of another class. But there is a difference meaning

Object Relationships

Aggregation



Composition: every car has an engine.

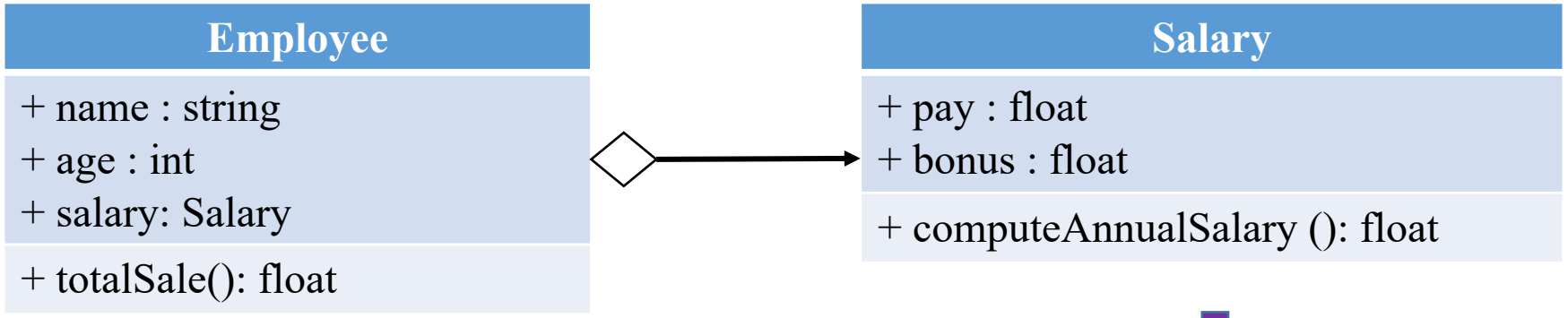


Aggregation: cars may have passengers, they come and go

Aggregation is one type of association between two objects describing the "**have/has a**" relationship, while Composition is a specific type of Aggregation which implies ownership.

Object Relationships

Aggregation: Example



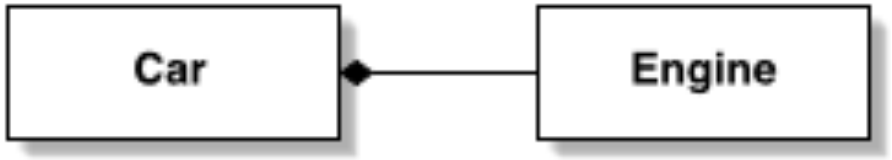

```python
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary # Aggregation

    def total_sal(self):
        return self.salary.computeAnnualSalary()
```

```python
class Salary:
    def __init__(self, pay, bonus):
        self.pay = pay
        self.bonus = bonus

    def computeAnnualSalary(self):
        return (self.pay*12)+self.bonus
```

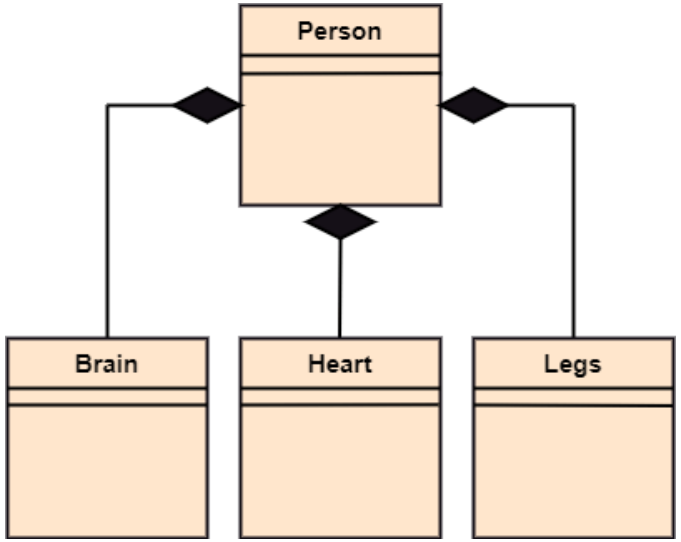
```python
salary = Salary(10000, 1500)
emp = Employee('AIVN', 25, salary)
print(emp.total_sal())
```

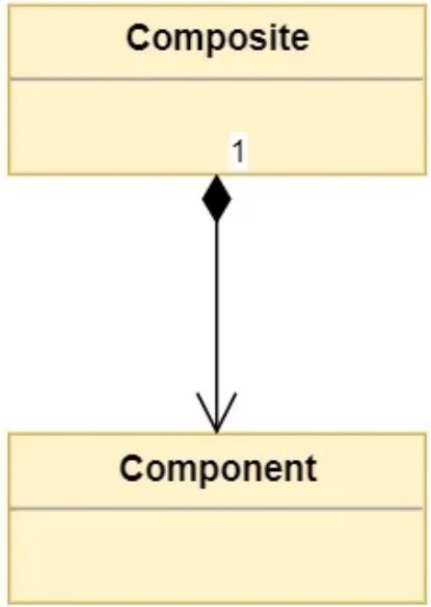
121500

Object Relationships
Composition



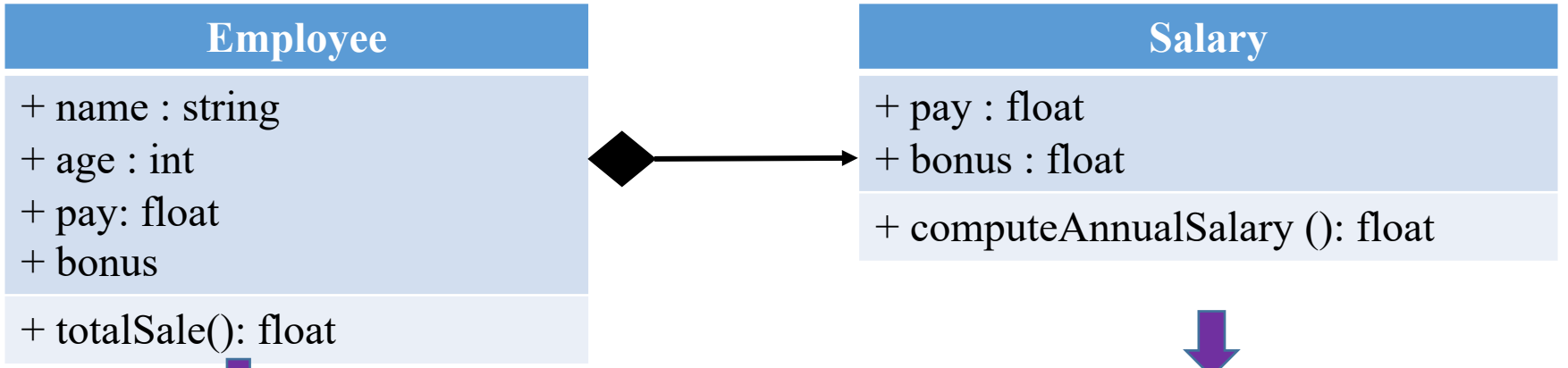
Composition: every car has an engine.





Composition is defined by the PART-OF relationship which means that one object IS PART-OF ANOTHER OBJECT

Object Relationships
Composition: Example



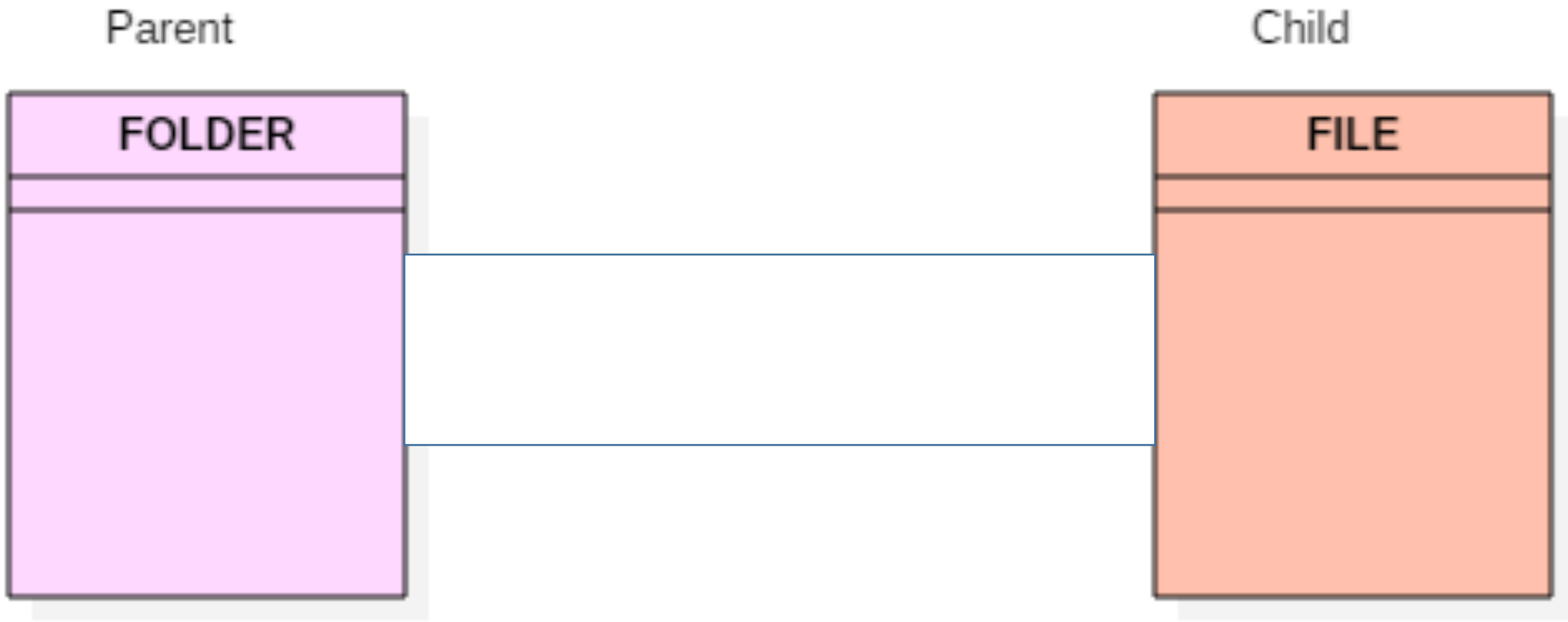

```python
class Employee:
    def __init__(self, name, age, pay, bonus):
        self.name = name
        self.age = age
        self.salary = Salary(pay, bonus) # composition

    def total_sal(self):
        return self.salary.computeAnnualSalary()
```

```python
class Salary:
    def __init__(self, pay, bonus):
        self.pay = pay
        self.bonus = bonus

    def computeAnnualSalary(self):
        return (self.pay*12)+self.bonus
```

```python
emp = Employee('AIVN', 25 ,10000, 1500)
print(emp.total_sal())
```

121500

Happy Minutes



Aggregation Example
Aggregate 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.

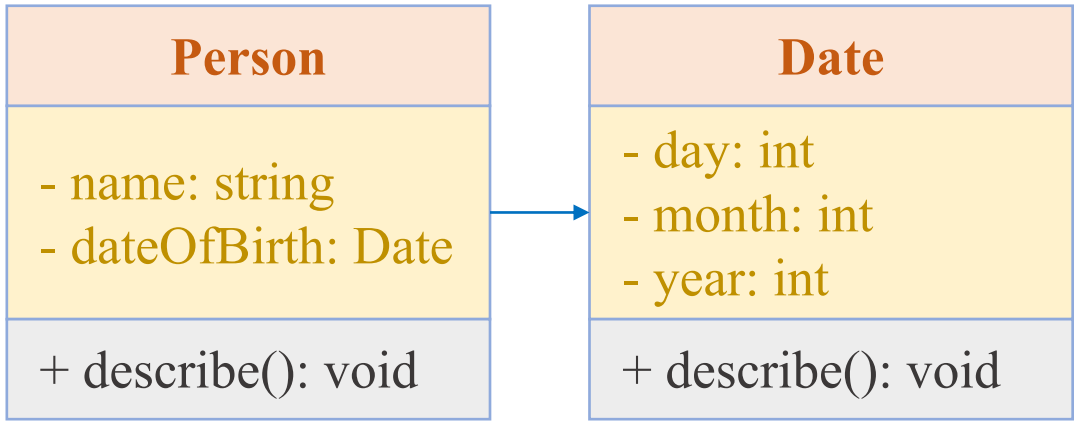
Draw a class diagram and implement in Python

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

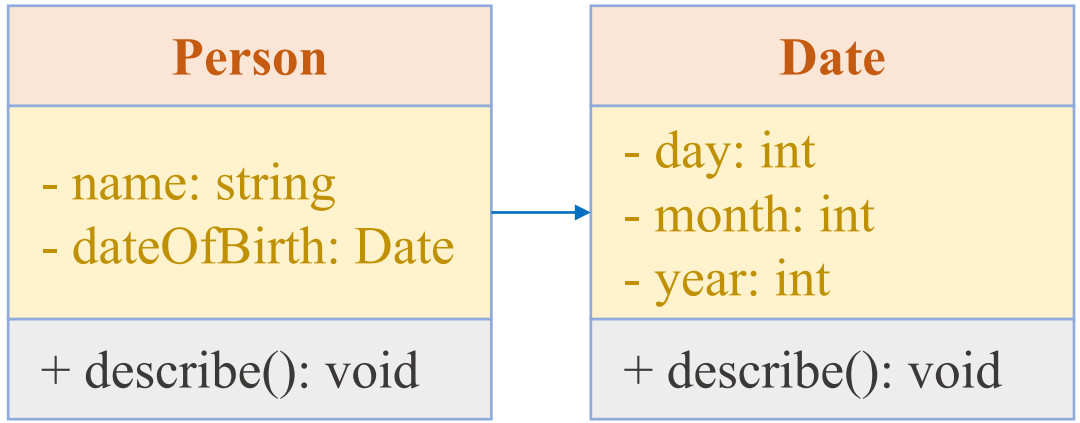



```python
class Date:
    def __init__(self, day, month, year):
        self.__day = day
        self.__month = month
        self.__year = year

    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

```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}')
```

Aggregation

```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
```

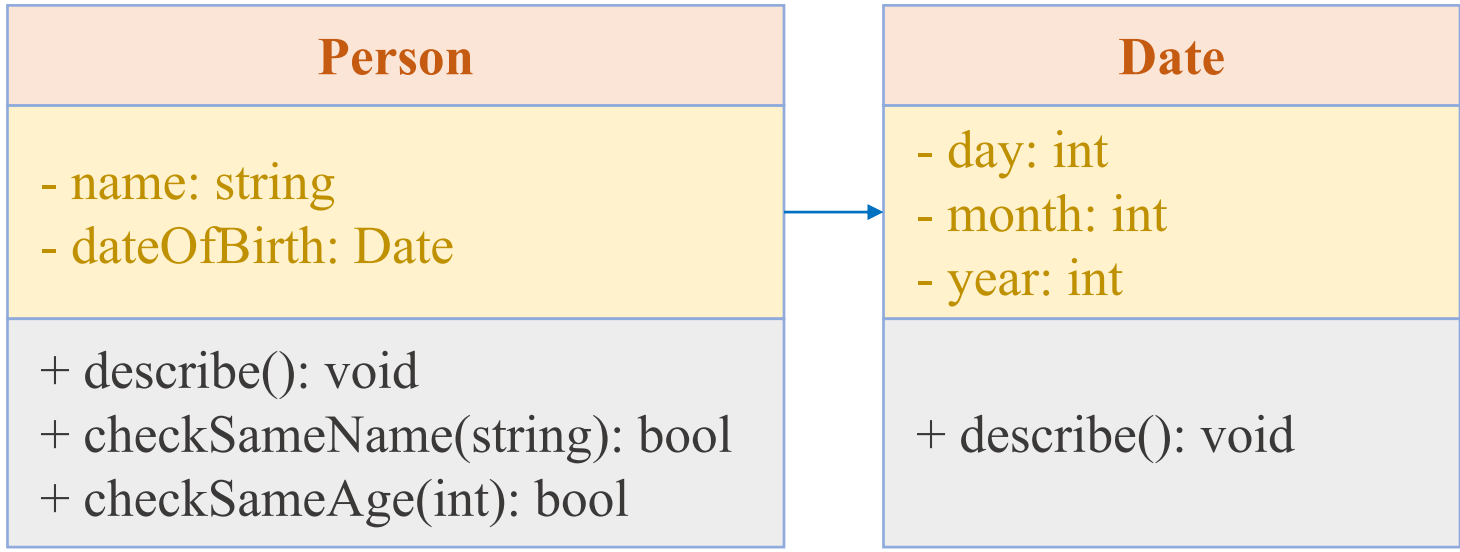
Class Data Type

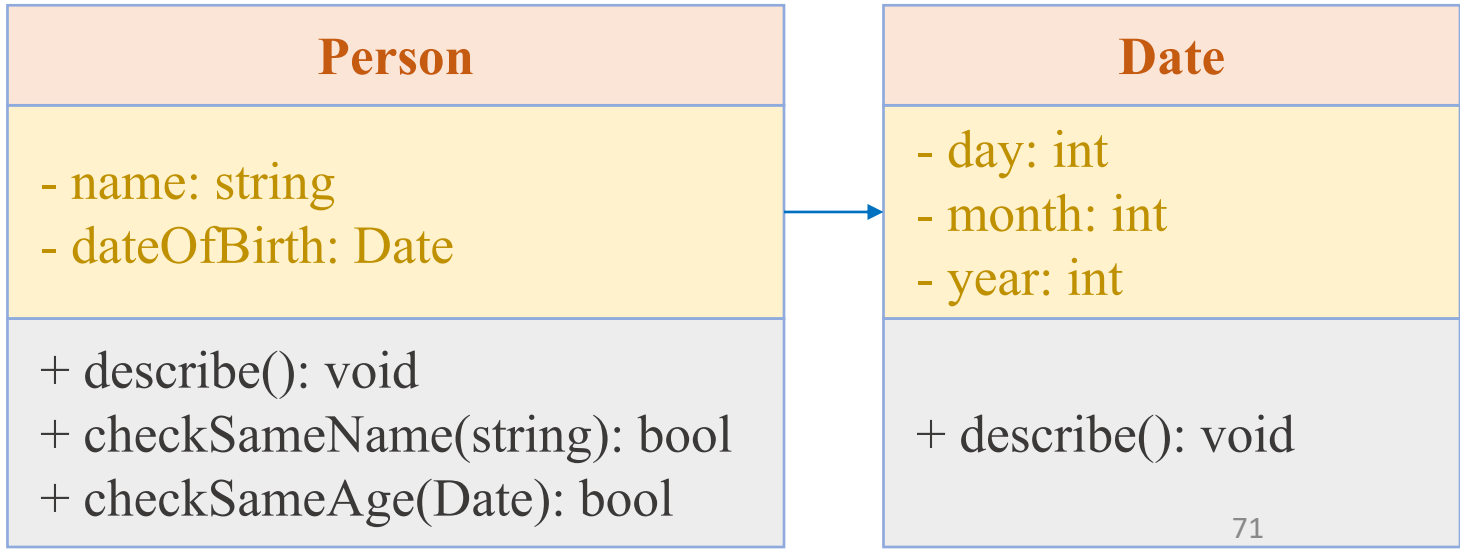
Using a class as a data type

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

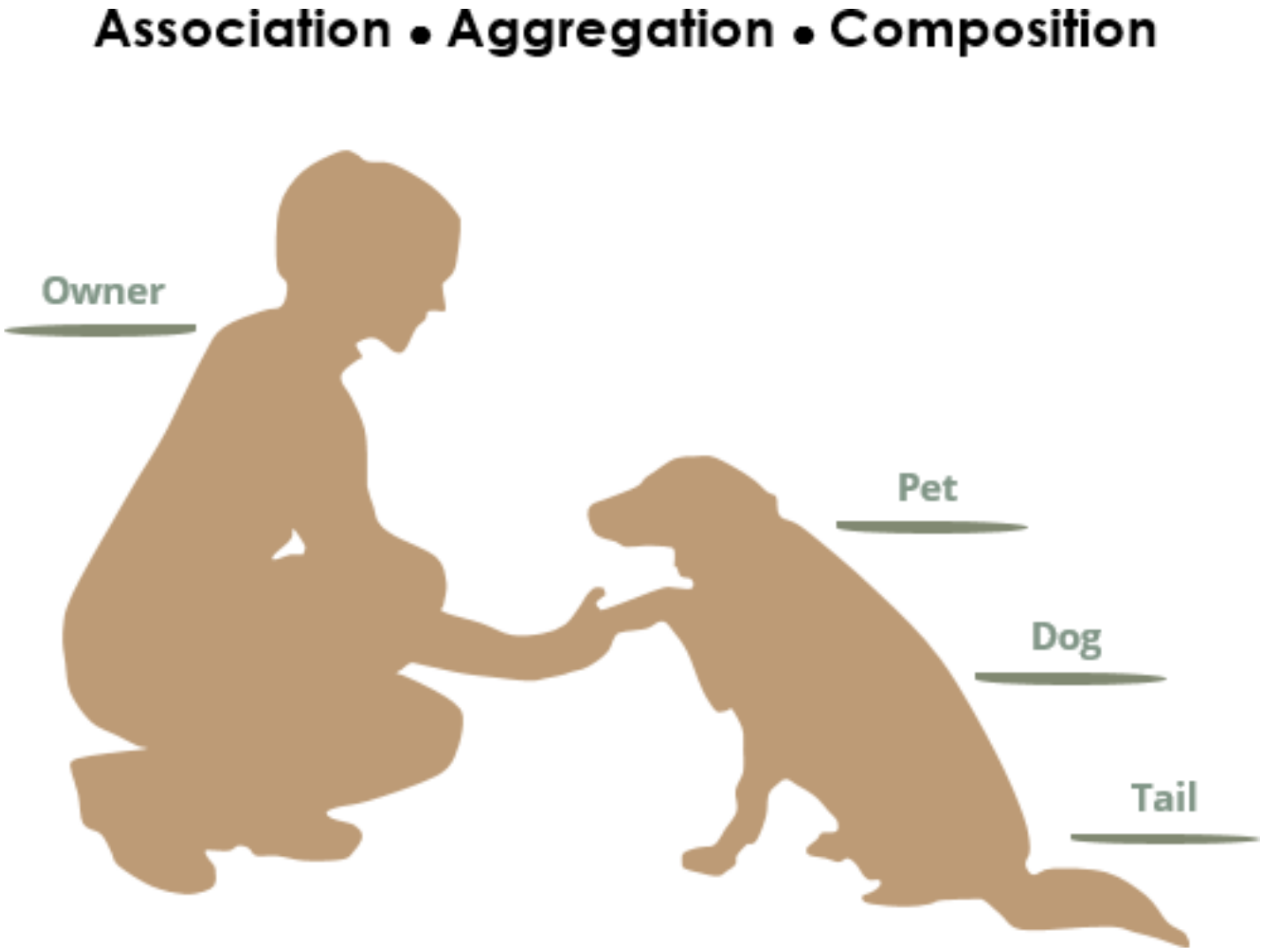
Write a method to check if two people have the same name.

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



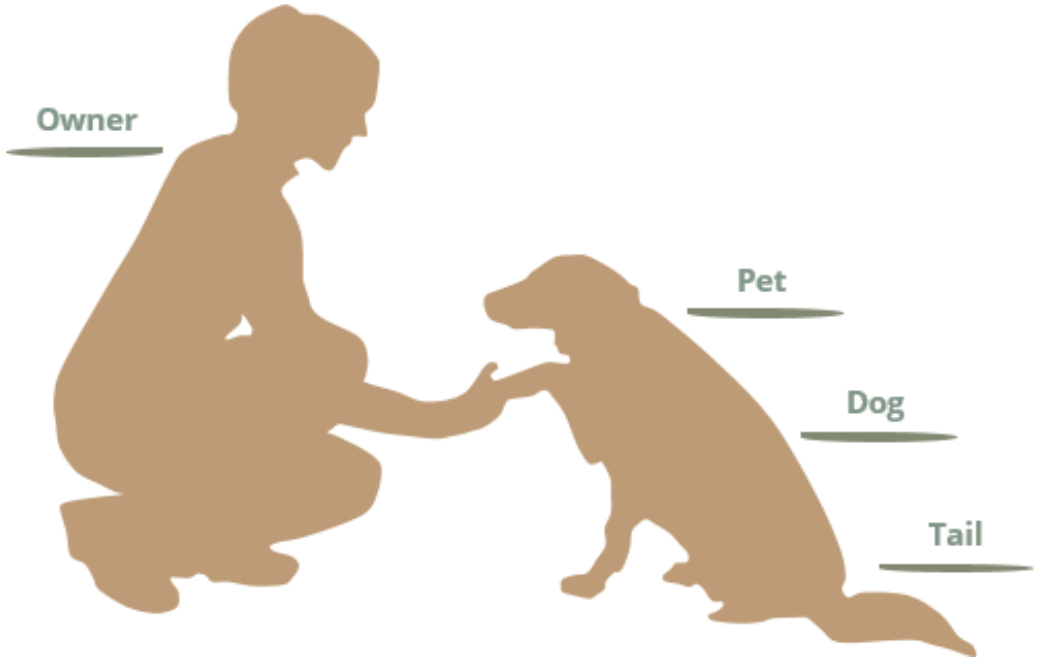


Case Study



Determine the relationship of Objects in the image

Association • Aggregation • Composition



- owners feed pets, pets please owners (**association**)
- a tail is a part of both dogs and cats (**aggregation** / **composition**)
- a cat is a kind of pet (**inheritance** / **generalization**)

Q1: What is main different between Aggregation and Composition

```python
class Employee:
    def __init__(self, name, age, pay, bonus):
        self.name = name
        self.age = age
        self.salary = Salary(pay, bonus) # composition

    def total_sal(self):
        return self.salary.computeAnnualSalary()
```

```python
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary # aggregation

    def total_sal(self):
        return self.salary.computeAnnualSalary()
```

Q2: When should we use Aggregation and Composition in OOP?

```
class Employee:
    def __init__(self, name, age, pay, bonus):
        self.name = name
        self.age = age
        self.salary = Salary(pay, bonus) # composition

    def total_sal(self):
        return self.salary.computeAnnualSalary()
```

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

    def total_sal(self):
        return self.salary.computeAnnualSalary()
```

Case study

Determine and implement the relationship between the following classes in Python

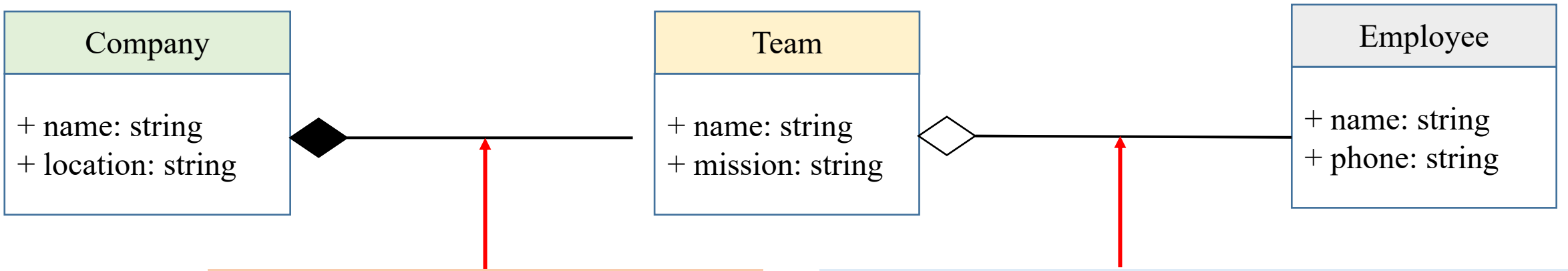
Company
+ name: string + location: string

Team
+ name: string + mission: string

Employee
+ name: string + phone: string

Case study

Determine and implement the relationship between the following classes in Python

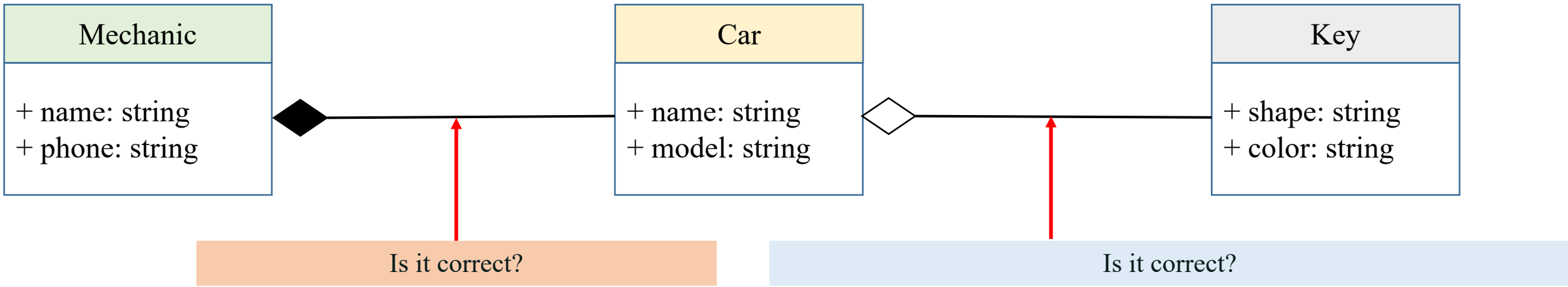


If the company object is deleted, the teams will have no reason to exist anymore.

When a team is deleted, the employees that were in the team, still exist. Employees might also belong to multiple teams. A team object does not "own" an employee object.

Case study

Determine and implement the relationship between the following classes in Python



Case study

Determine and implement the relationship between the following classes in Python

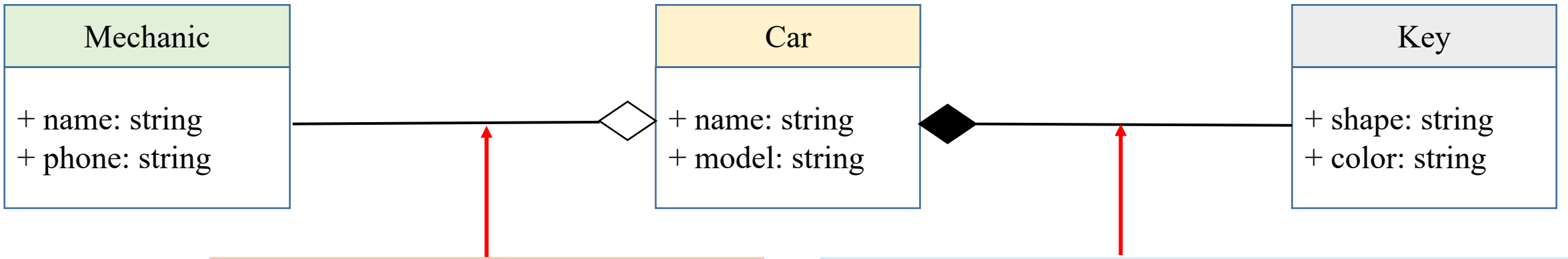
Mechanic
+ name: string + phone: string

Car
+ name: string + model: string

Key
+ shape: string + color: string

Case study

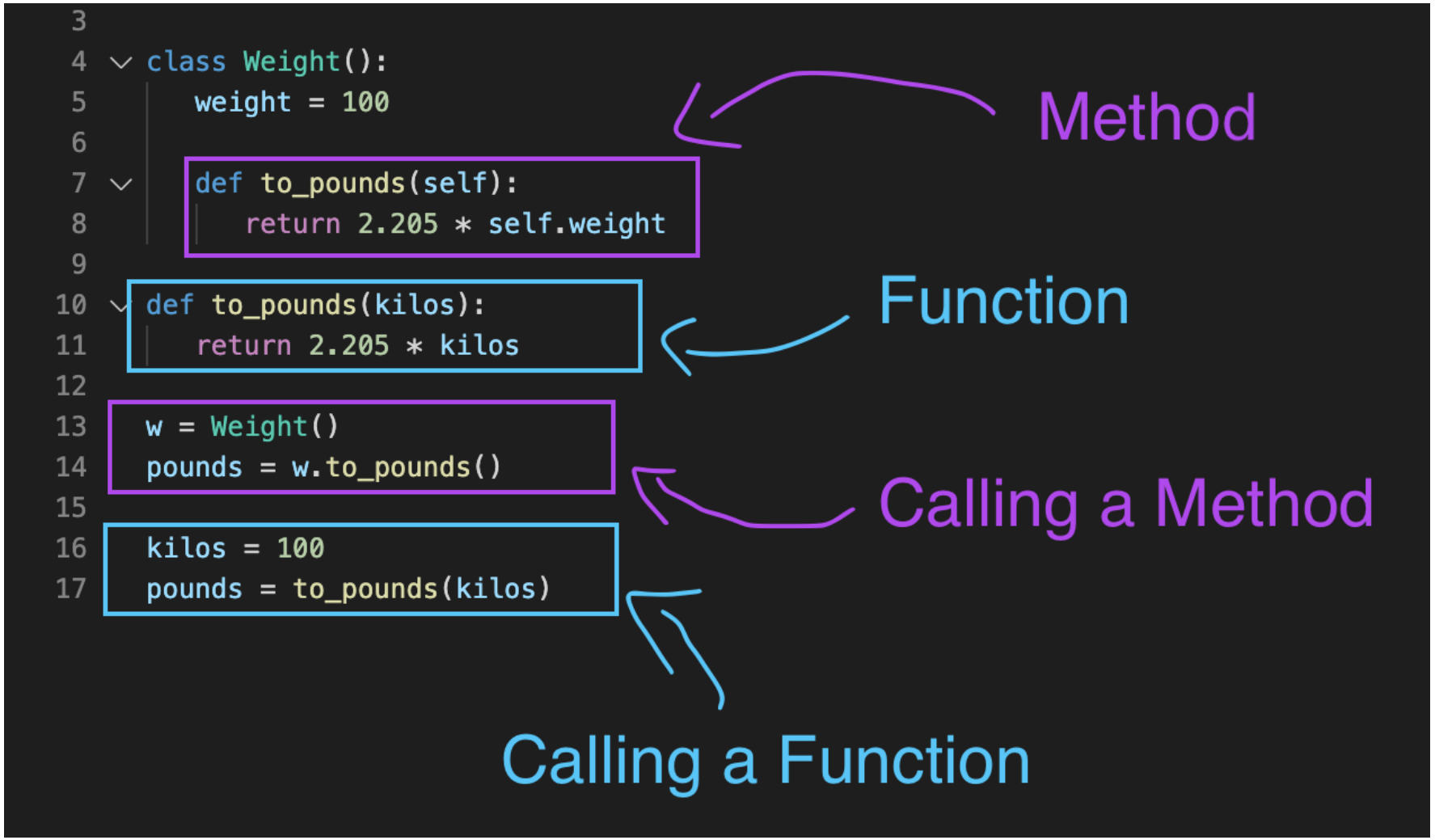
Determine and implement the relationship between the following classes in Python



A car needs a mechanic to work on it. A mechanic can also work on other cars at the same time. If a car object is deleted, the mechanic keeps working at the factory.

If the car object is deleted from the system, the key is useless and must be deleted also.

What are differences between Method and Function in Python


3
4 class Weight():
5 weight = 100
6
7 def to_pounds(self):
8 return 2.205 * self.weight
9
10 def to_pounds(kilos):
11 return 2.205 * kilos
12
13 w = Weight()
14 pounds = w.to_pounds()
15
16 kilos = 100
17 pounds = to_pounds(kilos)
Method
Function
Calling a Method
Calling a Function

Difference between Python Methods vs Functions

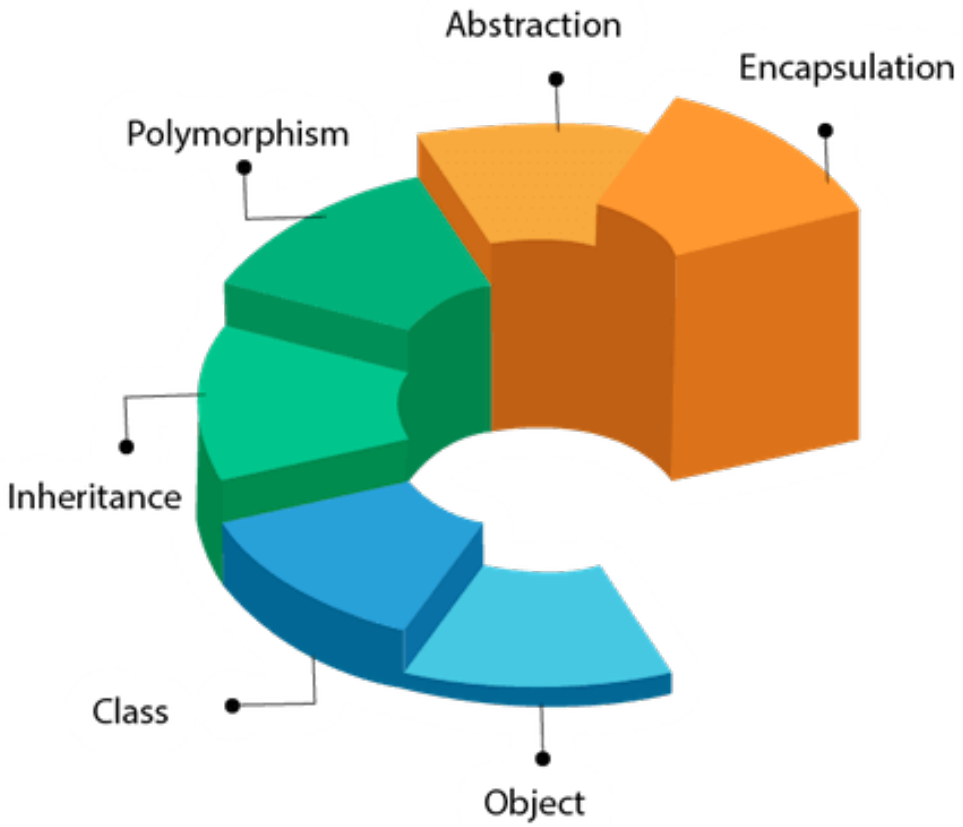
METHODS	FUNCTIONS
Methods definitions are always present inside a class.	We don’t need a class to define a function.
Methods are associated with the objects of the class they belong to.	Functions are not associated with any object.
A method is called ‘on’ an object. We cannot invoke it just by its name	We can invoke a function just by its name.
Methods can operate on the data of the object they associate with	Functions operate on the data you pass to them as arguments.
Methods are dependent on the class they belong to.	Functions are independent entities in a program.
A method requires to have ‘self’ as its first argument.	Functions do not require any ‘self’ argument. They can have zero or more arguments.

Outline

- Introduction to OOP
- Classes and Objects
- Object Relationships (Composition, Aggregation)
- Summarize
- Further reading

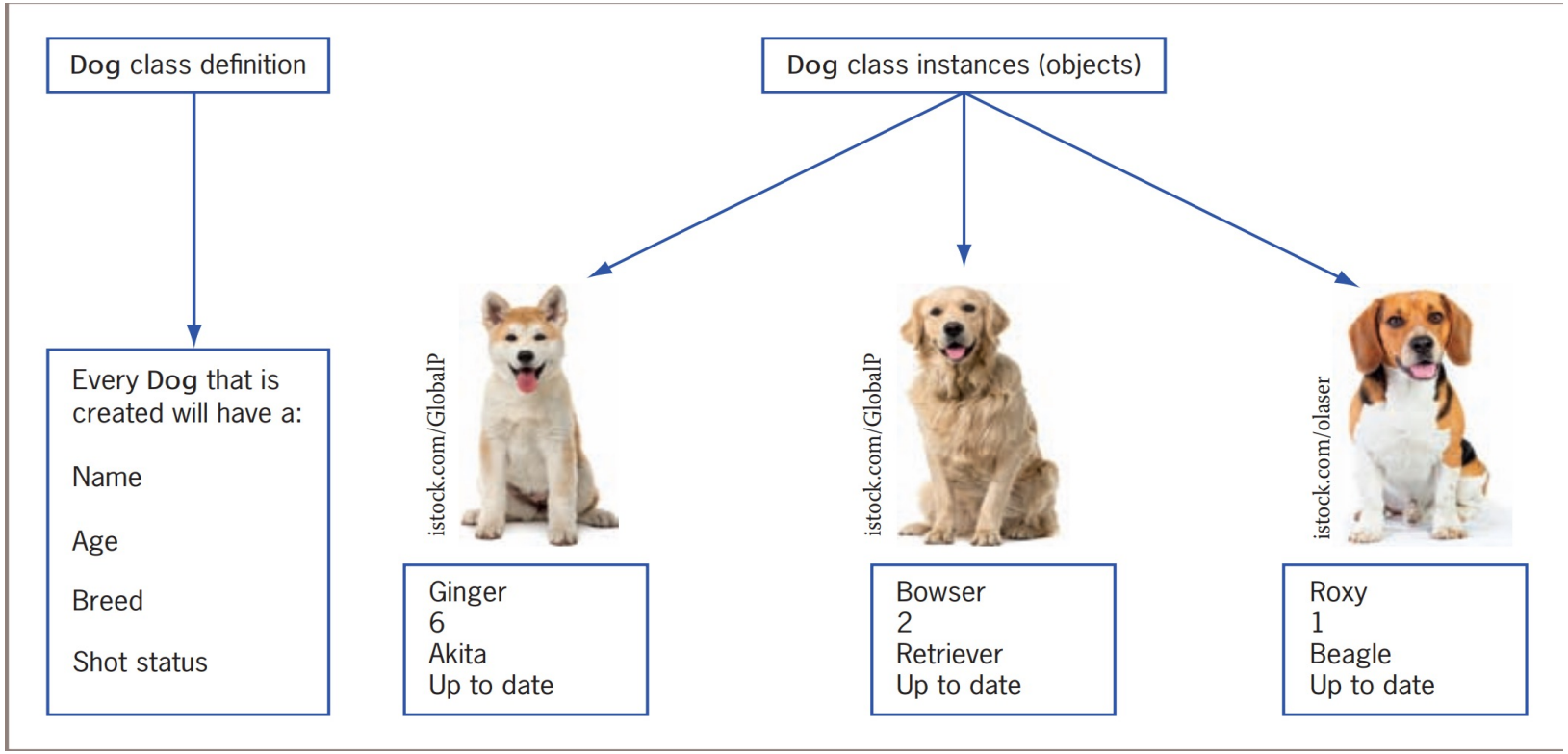
Summarize

➢ Introduction to OOP

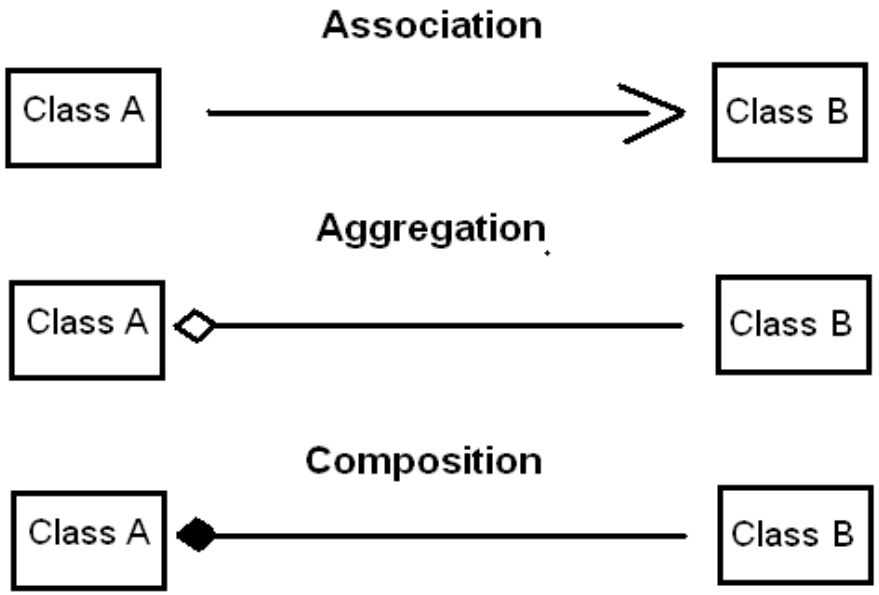


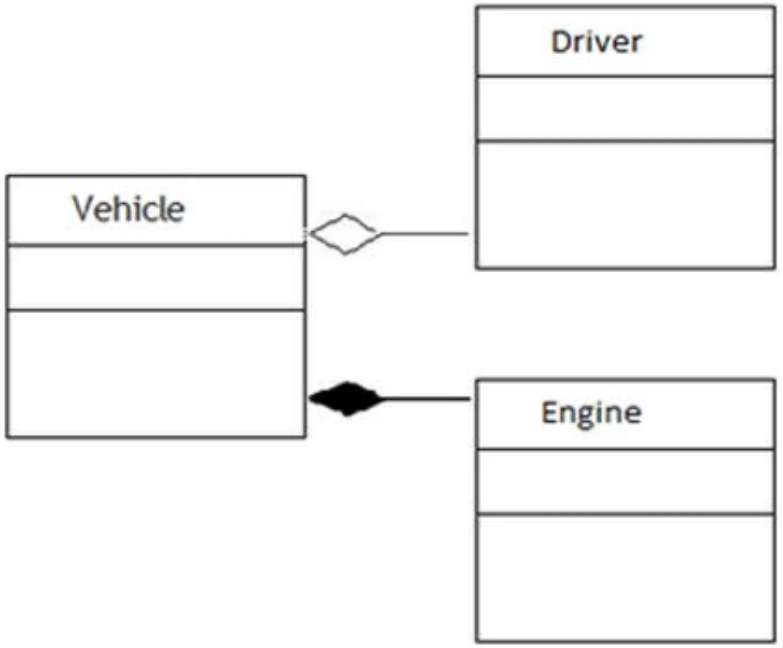
Summarize

➢ Classes and Objects



Object Relationships





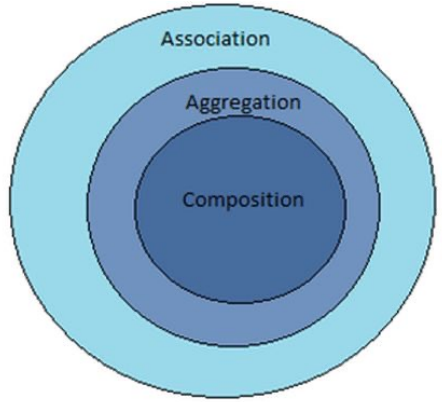
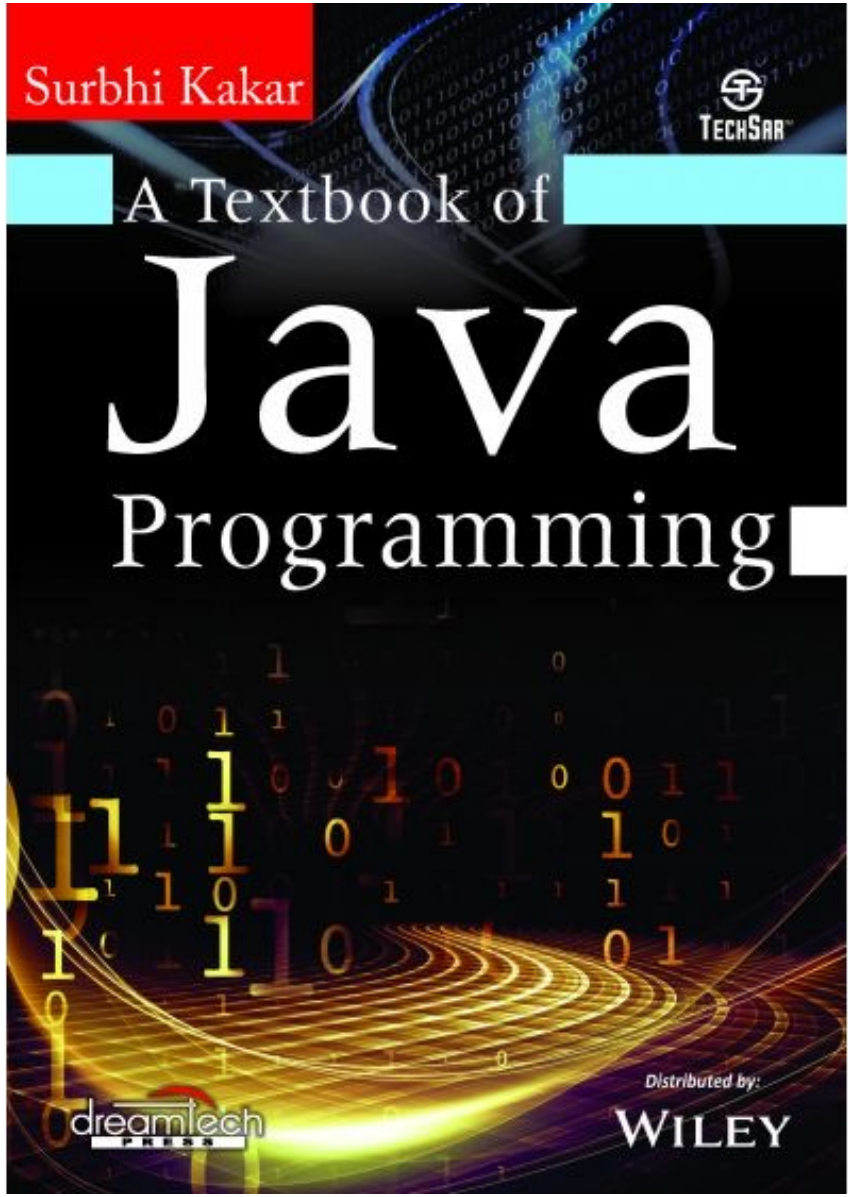


Image Credit:https://softwareengineering.stackexchange.com/

Exercise

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>
```

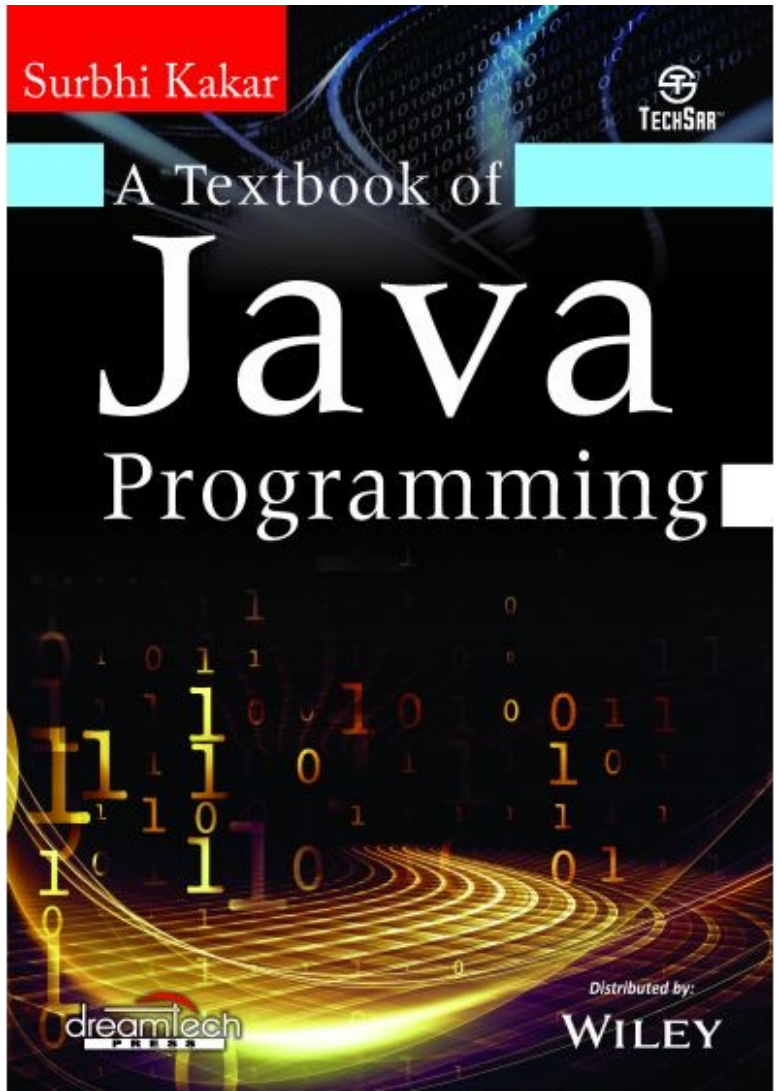
Book

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

...

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

Outline

- Advanced Object Relationships: Inheritance
- Access Modifier
- Module and Package
- Case study
- Exercises

NEXT TOPIC

Reference Books