



LISBON  
SCHOOL OF  
ECONOMICS &  
MANAGEMENT

UNIVERSIDADE DE LISBOA

# Object Oriented Programming

Prof. Carlos J. Costa, PhD

# Traditional Perspective

- The traditional perspective in software development had adopted is algorithm perspective.
- In this view, the main software building block are procedures or functions

# Object oriented Approach

The main structural components of all systems are:

- Objects
- Class Objects

# Main Concepts

- Classes,
- Objects, and
- Instances

# Object

- Objects represent an entity and the basic building block.
- Object is something that takes up space in the real or conceptual world with which somebody may do things ( Booch et al . 1999)
- The objects have :
  - Name (or ID )
  - state
  - Operations (or behavior )

# Object

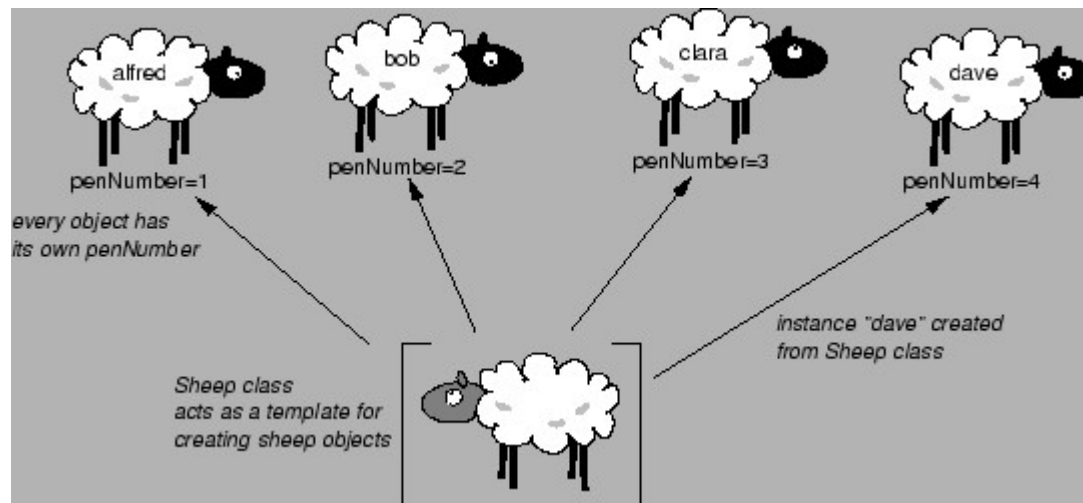
- Name (ID ) - The entire object must have a name that will differentiate from other objects in a context (eg my calculator )
- State - An object has state, which involves the object's properties together with the values of these properties (eg connected calculator)
- Operations (behavior ) - can do something with the object or the object can do something with another object (eg calculator does sums )

# Class

- A class is the description of a set of objects that share the same attributes, operations, relationships and semantics. (Eg calculators ).
- Class is the blue print of an object.

# Instance

- An object is an instance of a class.
- It is a concrete manifestation of an abstraction .  
(Eg " my calculator" is an instance of the class "calculating machines " ) .





# Main characteristics of the approach

- The object oriented approach has as main characteristics:
  - encapsulation
  - abstraction
  - inheritance
  - polymorphism

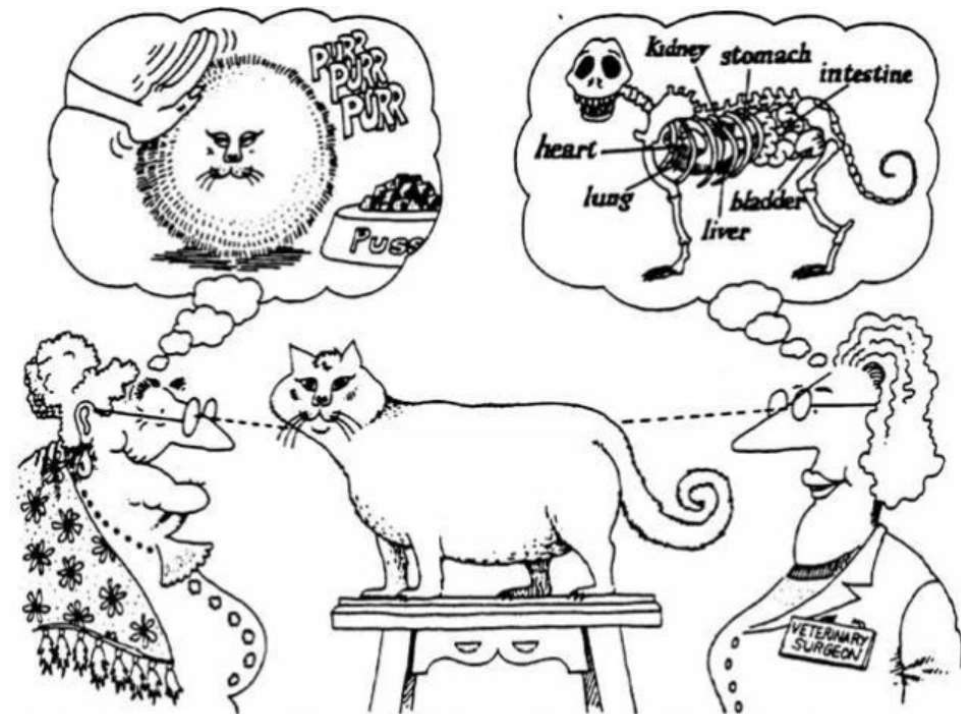
# Abstraction

- Abstraction is a principle which consists of ignoring the aspects of a subject that is not relevant for the present purpose, in order to concentrate on in those aspects that are really relevant .



# Abstraction

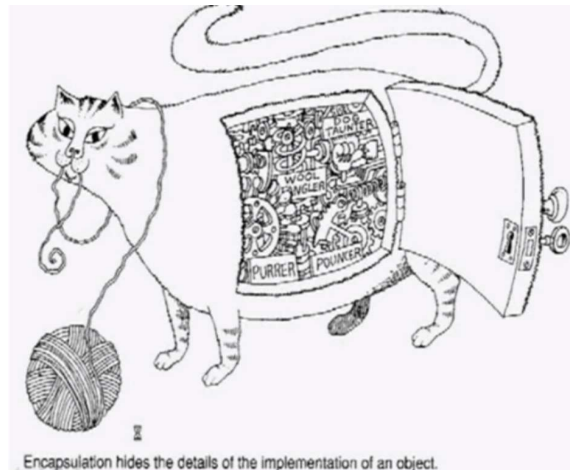
- Abstraction is the concise representation of a more complex object, focusing on the essential characteristics of the object .
- Good abstraction:
  - Appropriate ( If there is a real need can be satisfied )
  - appropriate level



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

# Encapsulation

- Encapsulation is the mechanism of hiding the implementation of the object, so that other system components do not have access to what is happening inside the object.

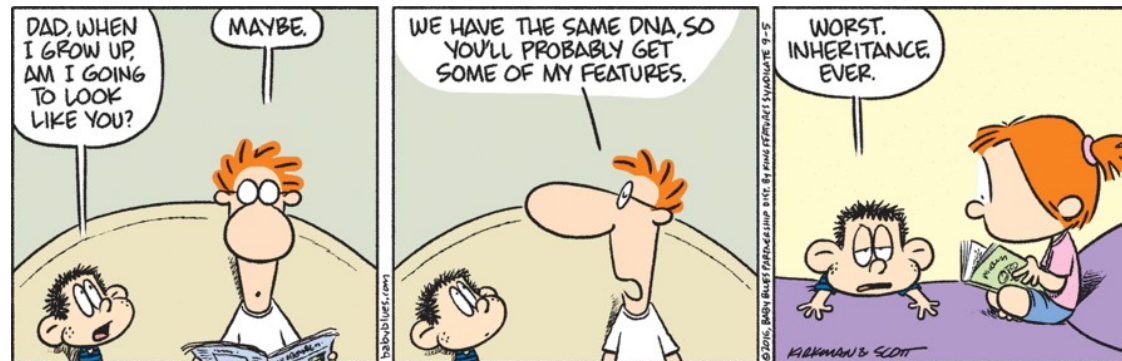


# Encapsulation

- This concept is associated with modularity , consisting in decomposing a system in a cohesive set of connected modules.
- Encapsulation is the mechanism of binding the data together and hiding them from outside world.
- Objects interact by message.

# Inheritance

- Inheritance is a mechanism that allows an object to incorporate all or part of the definitions of another object as part of itself (eg " doctor " and " optometrist ").
- Inheritance is the mechanism of making new classes from existing one.



# Polymorphism

- The word polymorphism means having many forms. In programming, polymorphism means same function name (but different signatures) being used for different types.

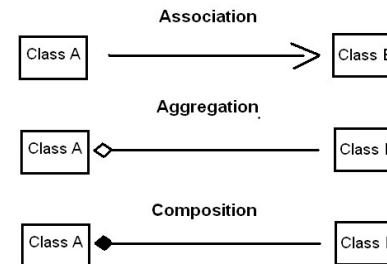
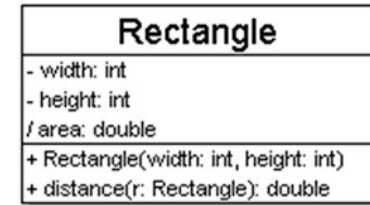
# Class Diagrams

- Diagrams that allow analyst
  - to specify the static structure of a system
  - according to the object-oriented approach .
- Used to describe the class model



# Class Diagrams

- Elements of a class diagram :
  - Classes
  - Relations between classes
    - Associations
    - Compositions
    - Aggregations
    - Generalizations



# Classe

## ID Class ( Class Name )

Campaign
code description annual Cost expected cost
pay() do Budget()

- Refers to specific objects, but the must abstract
- Nouns associated with the textual description of a problem
- Choose carefully the names
- using singular

## Attributes

- Values that characterize the objects of a class
- Types : Real, Integer , Text, Boolean , Enumerated , ...

## Operations

- Behaviors of the class ( service, method)

# Relationship

- A relationship UML establishes the connection between elements
- A relationship is graphically represented by a given type of line.
- In object-oriented modeling the three most important types of relationships are:
  - Associations
  - Generalizations
  - Dependencies

# Dependency

- A relationship of dependence, or simply dependence indicates that the change in the specification of an element can affect another element that uses it , but not necessarily the opposite.

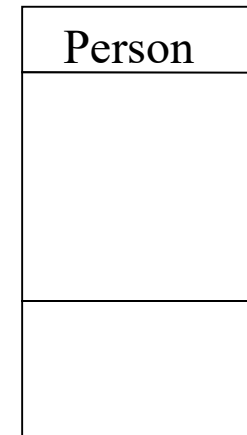


Now let's go to Python...

# Class

```
class Person:  
    pass # An empty block
```

```
p = Person()  
print(p)
```



## Result:

```
<__main__.Person object at 0x0000021D9EED60F0>
```

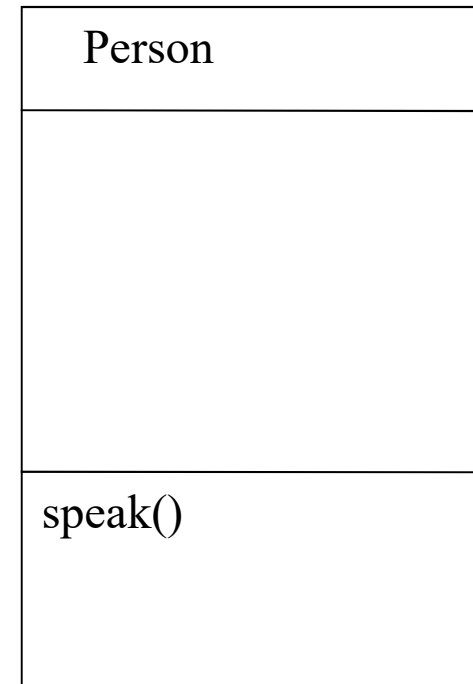
# Method

- define class with method

```
# class Person:  
    def speak(self):  
        print('Hello, how are you?')
```

- create object and call method

```
p = Person()  
p.speak()
```





# init method

- The first method **init()** is a special method,
- It is called class constructor or initialization
- Is a method that Python calls when you create a new instance of this class.

# init method

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

    def speak(self):
        print('Hello, my name is', self.name)

p = Person('Carlos')
p.speak()
```

Person
<code>__init__()</code> <code>speak()</code>

# self

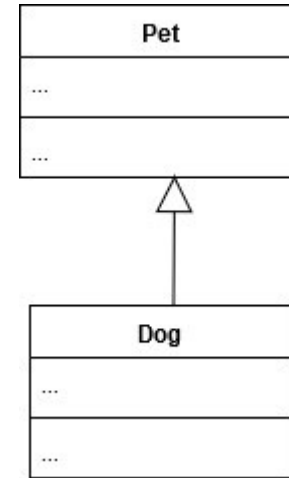
- The first argument of every class method, including `init`, is always a reference to the current instance of the class.
- By convention, this argument is always named `self`.
- In the `init` method, `self` refers to the newly created object;
- in other class methods, it refers to the instance whose method was called.

# Class Pet

```
class Pet(object):  
    def __init__(self, name, species):  
        self.name = name  
        self.species = species  
  
    def getName(self):  
        return self.name  
  
    def getSpecies(self):  
        return self.species  
  
    def __str__(self):  
        return "%s is a %s" % (self.name, self.species)
```

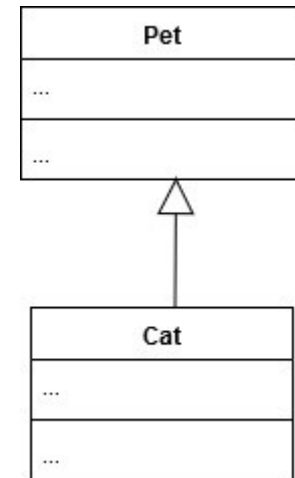
# Inheritance

```
class Dog(Pet):  
  
    def __init__(self, name, chases_cats):  
        Pet.__init__(self, name, "Dog")  
        self.chases_cats = chases_cats  
  
    def chasesCats(self):  
        return self.chases_cats
```



# Inheritance

```
class Cat(Pet):  
    def __init__(self, name, hates_dogs):  
        Pet.__init__(self, name, "Cat")  
        self.hates_dogs = hates_dogs  
  
    def hatesDogs(self):  
        return self.hates_dogs
```



```
myPet = Pet("Boby", "Dog")  
myDog = Dog("Boby", True)  
isinstance(myDog, Pet)  
isinstance(myDog, Dog)  
isinstance(myPet, Pet)  
isinstance(myPet, Dog)
```

# Bibliography

- **Bennet, S. McRobb, S & Farmer, R., *Object Oriented Systems Analysis and Design using UML*, MacGarw-Hill, 1999.**
- **Booch, G., Rumbaugh, J. & Jacobson, I, *The Unified Modeling Language User Guide*. Addison Wesley, 1999 (tradução portuguesa brasileira \_\_\_\_\_; *UML Guia do Usuário*; Campus, 2000).**
- **Costa, C. *Desenvolvimento para Web*, ITML Press, 2007**
- **Nunes, M & O'Neill, H. *Fundamental de UML*, FCA, 2001**
- **Silva, A & Videira, C., *UML, Metodologias e Ferramentas CASE*, Edições Centro Atlântico, 2001**
- **Terry, Q. *Visual Modeling With Rational Rose 2000 and UML*, Addison-Wesley. 2000.**
- ***Oxford Dictionary of Computing*, Oxford University Press.**