Introduction to OOP
Objects and classes

Course: Object Oriented Programming (OOP)
CTU, FS, U12110
Matouš Cejnek

Contents

- Brief history of OOP
- Motivation for OOP
- Objects and classes

What do we already know about OOP?

- Object oriented programming is a programming paradigm
- Objects, attributes and methods are the key elements in object oriented programming.

What we already know about functions?

A function is a set of instructions or procedures to perform a specific task.

Object functions are called methods.

What do we already know about memory management?

Memory management is a form of resource management applied to computer memory.

We recognize:

- Automatic memory management
- Manual memory management

Brief history of OOP

OOP history

- Terminology invoking "objects" and "oriented" in the sense of OOP made its first appearance at MIT in the late 1950s and early 1960s.
- In the 1970s, the first version of the Smalltalk programming language was developed.
- In 1986, the Association for Computing Machinery organised the first Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), which was unexpectedly attended by 1,000 people.

OOP history

- In the mid-1980s Objective-C was developed by Brad Cox and Bjarne Stroustrup
- In the early and mid-1990s object-oriented programming developed as the dominant programming paradigm
- Object-oriented features have been added to many previously existing languages, including Ada, BASIC, Fortran, Pascal, and COBOL.

OOP history

- Probably the most commercially important recent object-oriented languages are Java, C#, C++ and Visual Basic.NET (VB.NET).
- A number of languages have emerged that are primarily object-oriented (Python, Ruby, etc.).

Motivation for OOP

OOP motivation

- Save time and code
- Save even more time and code
- Work on abstract level
- Allow to start and test in broad scope and work on details later
- Simplification of massive projects

Basic concepts of OOP

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Basic concepts - Encapsulation

- Encapsulation refers to the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components.
- In most of the OOP languages, you can specify **private**, **protected** and **public** properties of an object.

Basic concepts - Inheritance

- Inheritance is concept of creating subclass from parent class in order to re-use some or all features of the parent class.
- It is basically a code sharing among classes.

Basic concepts - Polymorphism

- Polymorphism is the provision of a single interface to entities of different types.
- In other words, similar objects have similar properties. So they can be interchangeable.

Basic concepts - Abstraction

- Abstraction is concept of hiding unnecessary details to allow focus on a greater picture.
- Abstraction is related to generalization (using same/similar concepts to handle different objects etc.)

Objects and classes

Basic building blocks in OOP

- **Classes**
- **Objects** (instances of classes)
- **Methods** (functions, procedures)
- **Attributes** (fields, variables, or properties)

Class vs object vs instance

Class - template (blueprint) for objects

Object / instance - item created according to a class

Programmer writes code for classes, program creates instances and use them.

Object and instance are not totally the same, but in our scope we can interchange them freely.

OOP features common with other paradigms

- Variables - also know: parameters, references, attributes, …
- Procedures - functions, methods, …
- Members - name commonly used for class/object variables and/or methods

Access modifiers

- **Public** - It can be accessed everywhere.
- **Private** - Only the current class will have access
- **Protected** - Only the current class and subclasses will have access

This terminology is slightly language specific.

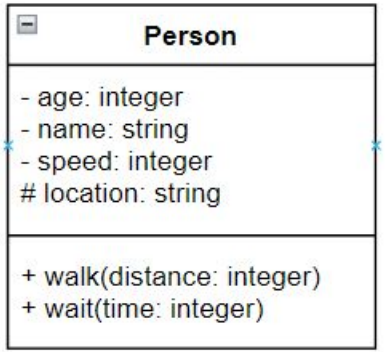
Example of access modifiers

All person related properties and functions are bundled in **Person class**

Private (**-** in UML)

Public (**+** in UML)

Protected (**#** in UML)



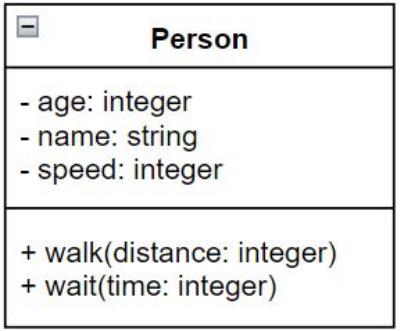
Instance attributes

- **Class attributes (static property)** – belong to the class as a whole; there is only one copy of each one
- **Instance attributes** – data that belongs to individual objects; every object has its own copy of each one

Example of instance properties

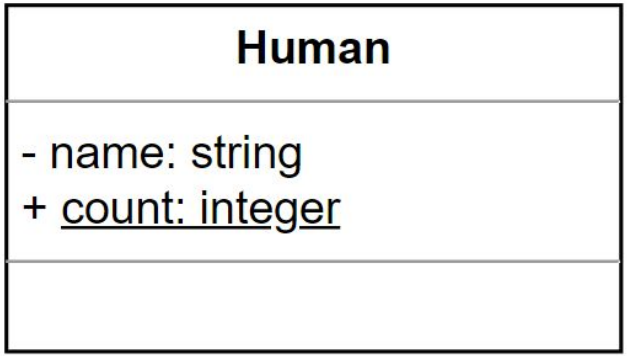
Class can have multiple properties - attributes common for all instances. Imagine multiple instances of the Human class:

	age	**speed**	**name**
Instance1	56	4.8	Alice
Instance2	28	2.6	Bob



Class vs instance properties

- Instance properties are properties with individual values for instances.
- Class properties (**static properties**) are properties with values common for all instances.



Methods

- **Class methods** – belong to the class as a whole and have access to only class variables and inputs
- **Instance methods** – belong to individual objects, and have access to instance variables for the specific object they are called on, inputs, and class variables
- **Static methods** – methods without access to class or instance variables

Note: it is language specific and quite confusing.

Example of methods

```
class Example:
    def method(self):
        pass

    @classmethod
    def classmethod(cls):
        pass

    @staticmethod
    def staticmethod():
        pass
```

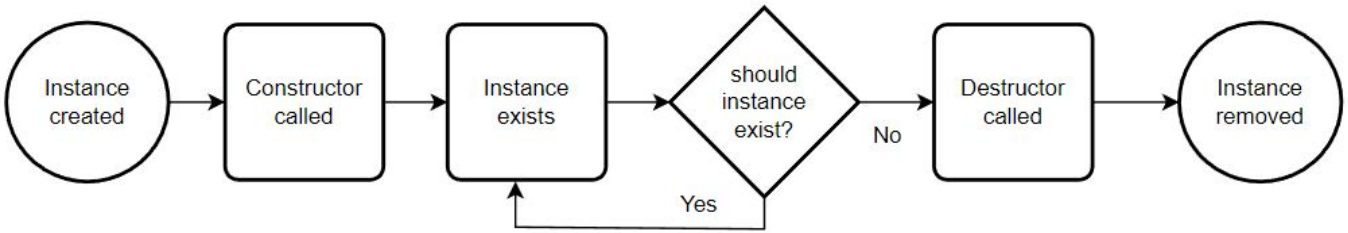
Constructor, destructor

- **Constructor** - this method is called during creation of an instance. Commonly used to prepopulate class attributes with provided inputs.
- **Destructor** - this method is called during destruction of an instance. This method should clean the instance from memory.

Constructor, destructor and garbage collector

The **garbage collector** attempts to reclaim memory which was allocated by the program, but is no longer referenced.

Some languages prefer **manual memory management**.



Constructor and destructor example

```
class Human():

    def __init__(self, name):
        self.name = name
        print(self.name, ": I am alive!")

    def __del__(self):
        print(self.name, ": Now it is over")

a = Human("A")
b = Human("B")
b = False
```

Resulting output:

A : I am alive!
B : I am alive!
B : Now it is over
A : Now it is over

Example: class property

```
class Human:
    count = 0
    def __init__(self):
        Human.count += 1
    def __del__(self):
        Human.count -= 1

print(Human.count)
humans = [Human() for _ in range(10)]
print(Human.count)
humans = None
print(Human.count)
```

Resulting output:

0

10

0

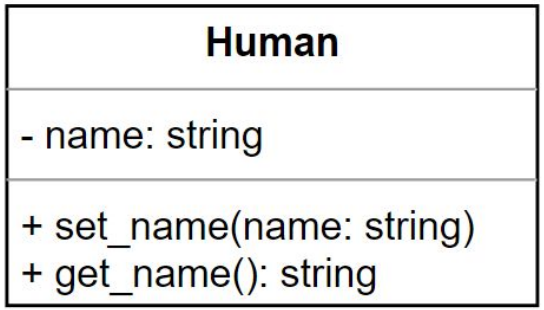
Magic methods

Magic methods are special methods. Implementation vary according to the language. In general:

- They have specific reserved namespace
- These methods are often called when some conditions are met
- Prime example is constructor and destructor.

Getter and setter methods

- Methods designed to get and set a property of an object.
- Setter can provide additional validation of the new value.

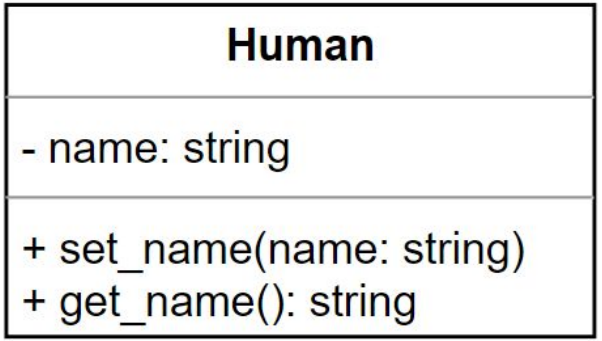


Getter and setter: simple example

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

    def get_name(self):
        return self._name

    def set_name(self, value):
        self._name = value
```



Bonus content

Some magic methods in Python

__len__: length (list, string, etc.)

__abs__: absolute values (numbers, etc.)

__add__ : + operator

__str__: string representation of the object

__ge__, __le__: greater than or equal, less than or equal (>=, <=)

Magic Methods in Python

Use dir() function to see members of an object:

```
>> dir(5)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
'__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__truediv__', '__trunc__', '__xor__', 'as_integer_ratio', 'bit_length',
'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```