

# [FIM] FONDAMENTI DI INFORMATICA per medicina e chirurgia high tech

L10: Inheritance

Dott. Giorgio De Magistris

*demagistris@diag.uniroma1.it*

CORSO DI LAUREA IN MEDICINA E CHIRURGIA HIGH TECH



SAPIENZA  
UNIVERSITÀ DI ROMA

I3S

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE, INFORMATICA E STATISTICA

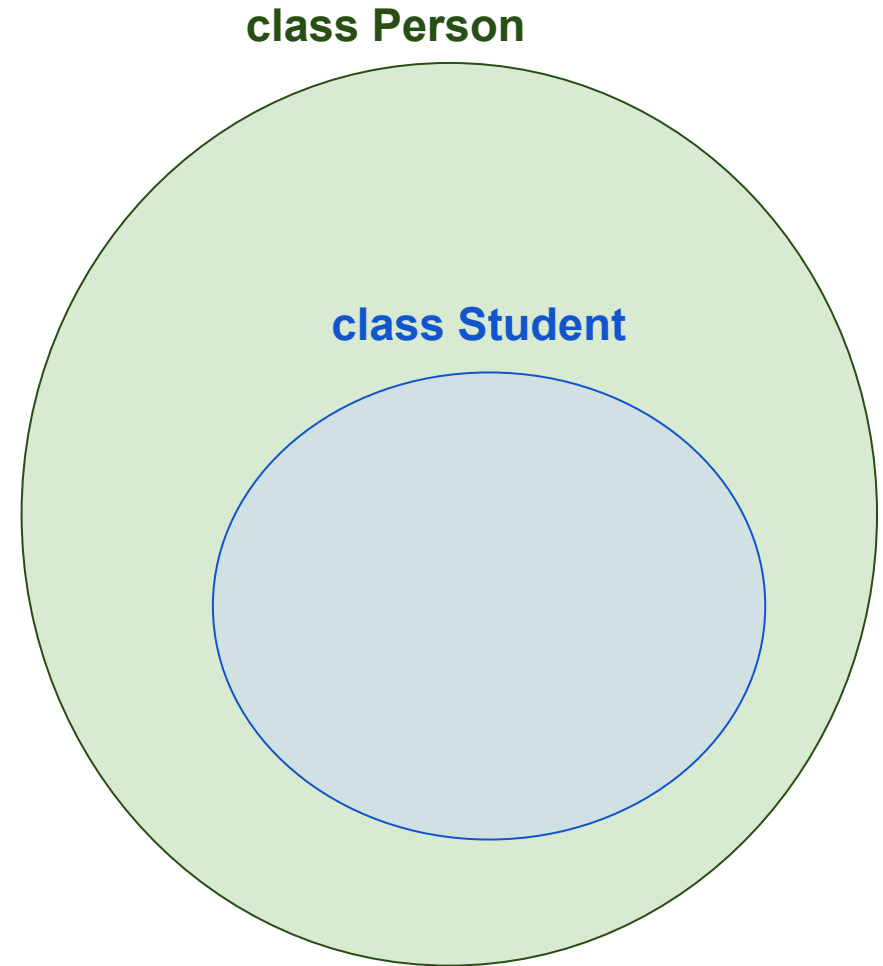
DIAG

DIPARTIMENTO DI INGEGNERIA INFORMATICA, AUTOMATICA E GESTIONALE

TUTTI I DIRITTI RELATIVI AL PRESENTE MATERIALE DIDATTICO ED AL SUO CONTENUTO SONO RISERVATI A SAPIENZA E AI SUOI AUTORI (O DOCENTI CHE LO HANNO PRODOTTO). È CONSENTITO L'USO PERSONALE DELLO STESSO DA PARTE DELLO STUDENTE A FINI DI STUDIO. NE È VIETATA NEL MODO PIÙ ASSOLUTO LA DIFFUSIONE, DUPLICAZIONE, CESSIONE, TRASMISSIONE, DISTRIBUZIONE A TERZI O AL PUBBLICO PENA LE SANZIONI APPLICABILI PER LEGGE

# Inheritance

- Allows to define a hierarchy of classes
- Class A is a subclass of class B if all instances of A can be considered instances of B
- A subclass inherits all the attributes and methods of the parent class



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

    def printInfo(self):
        print(f"name:{self.name}, surname:{self.surname}")

    def __str__(self):
        return f"{self.name},{self.surname}"

class Student(Person):
    def __init__(self, name: str, surname: str, student_id:str):
        super().__init__(name, surname)
        self.student_id = student_id

    def printInfo(self):
        print(f"name:{self.name}, surname:{self.surname}, student_id:{self.student_id}")

    def __str__(self):
        return super().__str__() + f",{self.student_id}"

p = Person("Mario", "Rossi")
print(p)
p.printInfo()

s = Student("Luigi", "Bianchi", "1234567")
print(s)
s.printInfo()
print(s.name)

print(p.student_id)
```

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

    def printInfo(self):
        print(f"name:{self.name}, surname:{self.surname}")

    def __str__(self):
        return f"{self.name},{self.surname}"

class Student(Person):
    def __init__(self, name: str, surname: str, student_id: str):
        super().__init__(name, surname)
        self.student_id = student_id

    def printInfo(self):
        print(f"name:{self.name}, surname:{self.surname}, student_id:{self.student_id}")

    def __str__(self):
        return super().__str__() + f",{self.student_id}"

p = Person("Mario", "Rossi")
print(p)                // Mario,Rossi
p.printInfo()           // name:Mario,surname:Rossi

s = Student("Luigi", "Bianchi", "1234567")
print(s)                // Luigi,Bianchi,1234567
s.printInfo()           // name:Luigi,surname:Bianchi,student_id:1234567
print(s.name)           // Luigi

print(p.student_id)     // Raises AttributeError
```

# Overriding

- Overriding occurs when a subclass provides a more specific implementation of a method defined in a subclass, the two methods have the same signature
- The interpreter uses the implementation provided by the most specific class

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

```
def printInfo(self):
    print(f"name:{self.name}, surname:{self.surname}")
```

```
def __str__(self):
    return f"{self.name}, {self.surname}"
```

```
class Student(Person):
    def __init__(self, name: str, surname: str, student_id: str):
        super().__init__(name, surname)
        self.student_id = student_id
```

```
def printInfo(self):
    print(f"name:{self.name}, surname:{self.surname}, student_id:{self.student_id}")
```

```
def __str__(self):
    return super().__str__() + f", {self.student_id}"
```

Overriding

# Overloading

- In statically typed languages we can have functions with the same name but different types or number of parameters, this is called overloading
- Python is a dynamically typed language, so in principle we could have at most two functions with the same name but different number of parameters
- However this is not allowed in Python, indeed the namespaces in Python are implemented with dictionaries, in which functions are addressed by their names, so if a function is redefined with a different parameter list the old function is replaced by the new function

```
>>> def foo(a,b):
...     return a+b
...
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'foo': <function foo at 0x7f692a245700>}
>>> def foo(a,b,c):
...     return a+b+c
...
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'foo': <function foo at 0x7f692a245790>}
>>> foo(1,2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: foo() missing 1 required positional argument: 'c'
>>>
```

# Operators Overloading

- A class can implement certain operations that are invoked by special syntax (such as arithmetic operations or subscripting and slicing) by defining methods with special names. This is Python's approach to operator overloading, allowing classes to define their own behavior with respect to language operators.
- These are the most common operators:

## Comparison operators

```
object.__lt__(self, other)
object.__le__(self, other)
object.__eq__(self, other)
object.__ne__(self, other)
object.__gt__(self, other)
object.__ge__(self, other)
object.__bool__(self)
```

## Operators for Container Objects

```
object.__len__(self)
object.__getitem__(self, key)
object.__setitem__(self, key, value)
object.__iter__(self)
object.__contains__(self, item)
```

## Math operators

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__truediv__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__pow__(self, other[, modulo])
```

## Operators for Callable Objects

```
object.__call__(self[, args...])
```

---

## **Slides distribuite con Licenza Creative Commons (CC BY-NC-ND 4.0) Attribuzione - Non commerciale - Non opere derivate 4.0 Internazionale**

### **PUOI CONDIVIDERLE ALLE SEGUENTI CONDIZIONI**

(riprodurre, distribuire, comunicare o esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato)

#### **Attribuzione\***

Devi riconoscere una menzione di paternità adeguata, fornire un link alla licenza e indicare se sono state effettuate delle modifiche. Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.

#### **Non Commerciale**

Non puoi utilizzare il materiale per scopi commerciali.

#### **Non opere derivate**

Se remixi, trasformi il materiale o ti basi su di esso, non puoi distribuire il materiale così modificato.

#### **Divieto di restrizioni aggiuntive**

Non puoi applicare termini legali o misure tecnologiche che impongano ad altri soggetti dei vincoli giuridici a questa licenza