OOP - Inheritance

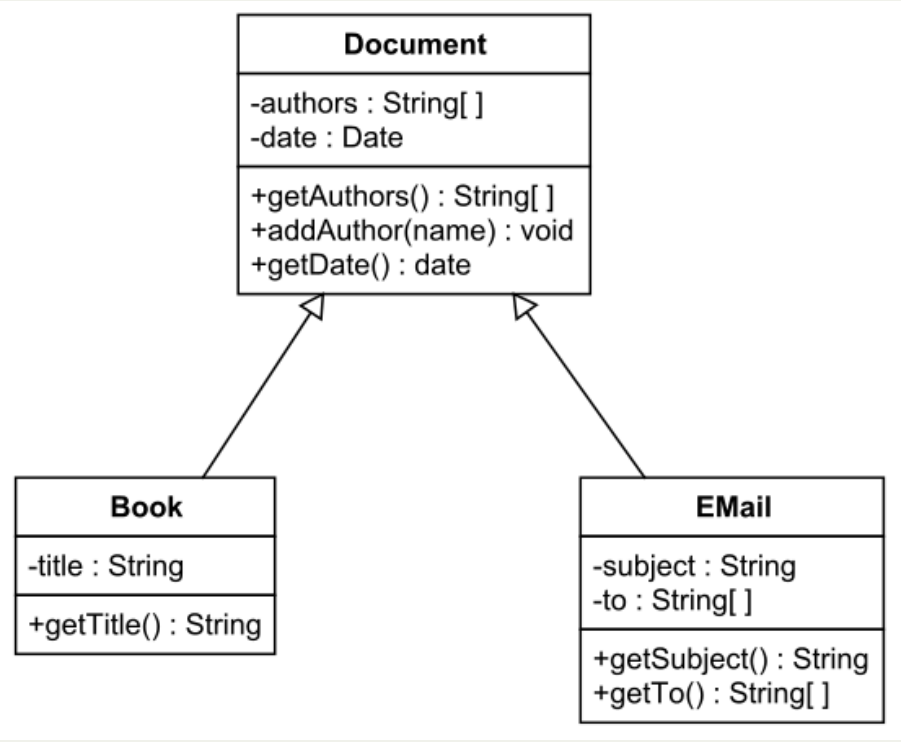
Instructors **Battista Biggio**, **Angelo Sotgiu** and **Leonardo Regano**

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence
University of Cagliari, Italy

Inheritance

Inheritance is the mechanism of building a class upon another class, maintaining a similar implementation.

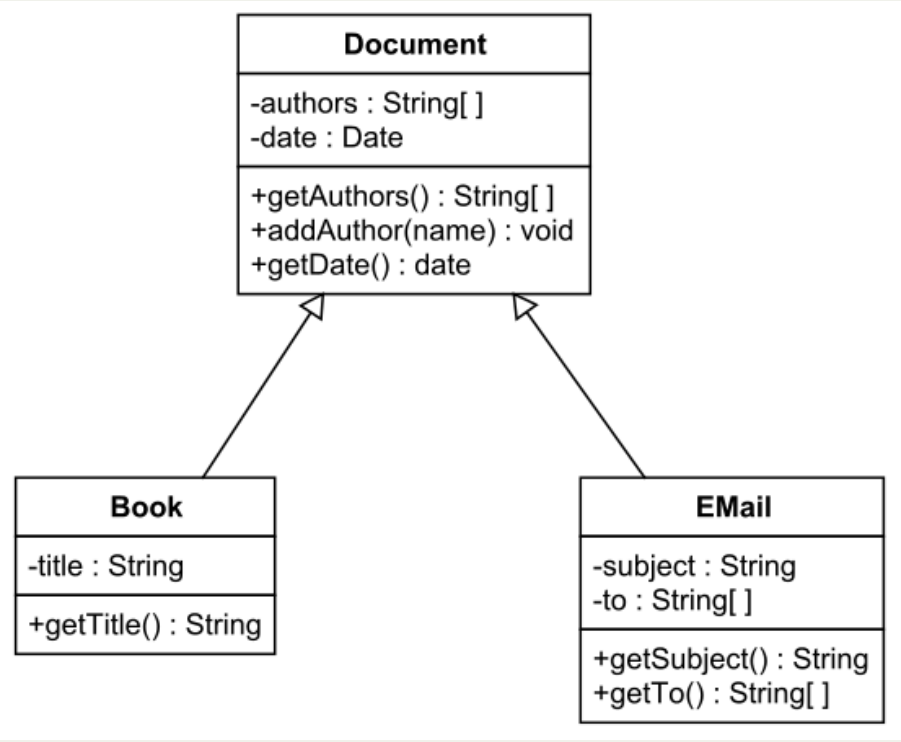
- The 'original' class is called Superclass, base class, or parent class
- The 'new' class is called Subclass, derived class, or child class



Inheritance

A class **subclasses** another class. The `Book` class and the `Mail` class subclass the `Document` class.

The Book and Email classes **inherit the attributes and methods** from the Document class. It is possible to modify the methods and **add** new **fields and methods**.



Main Forms of Inheritance

- **Specialization**: subclasses override methods from the superclass, keeping their other features.
- **Specification**: an *abstract* superclass defines methods that will be implemented by its subclasses.
- **Extension**: subclasses add new methods without altering inherited attributes/methods.
- **Combination**: a subclass inherit from multiple classes.

Inheritance - (*Python implementation*)

All Python classes are subclasses of the special class named `object`.

So all Python classes inherit all methods that we use, but we are usually not aware of them, such as the `__new__` class method.

```
class MySubClass1(object):
  pass


class MySubClass2:
  pass
```

Add new behavior to existing class

To define new behaviors we can add new methods to the subclass.

```
class MyClass:

  def __init__(self):
    # doSomething()
    ...

  def f1(self):
    ...


class MySubClass(MyClass):

  # it uses the __init__ and f1 method of the superclass

  def f2():
    ...
```

Change behavior to existing class

To change the behavior we can redefine (*override*) a method in the subclass.

```
class MyClass:

  def __init__(self):
    # doSomething()
    ...

  def f1(self):
    ...


class MySubClass(MyClass):

  # it uses the __init__  method of the superclass

  def f1(self):  # redefine the method
    ...
```

Change behavior to existing class

Sometimes we want the new method to do what the old method did, **plus other actions**.

```
class MyClass:

  def f1(self):
    # do_1()
    # do_2()


class MySubClass(MyClass):

  def f1(self):  # redefine the method
    # do_1()  # Problem: duplicate code
    # do_2()  # Problem: duplicate code
    # do_3()
```

Change behavior to existing class

Code maintenance is complicated. We have to update the code in two or more places. We need a way to **execute the original `f1()` method** on the `MyClass` class, and after the **new `f1()` method**.

```
class MyClass:

  def f1(self):
    # do_1()
    # do_2()


class MySubClass(MyClass):

  def f1(self):  # redefine the method
    super().f1()
    # do_3()
```

The `super()` function returns an instance of the parent class, allowing us to call the parent method directly.

A `super()` call can be made inside any method.
All methods can be modified via overriding and calls to `super()`.

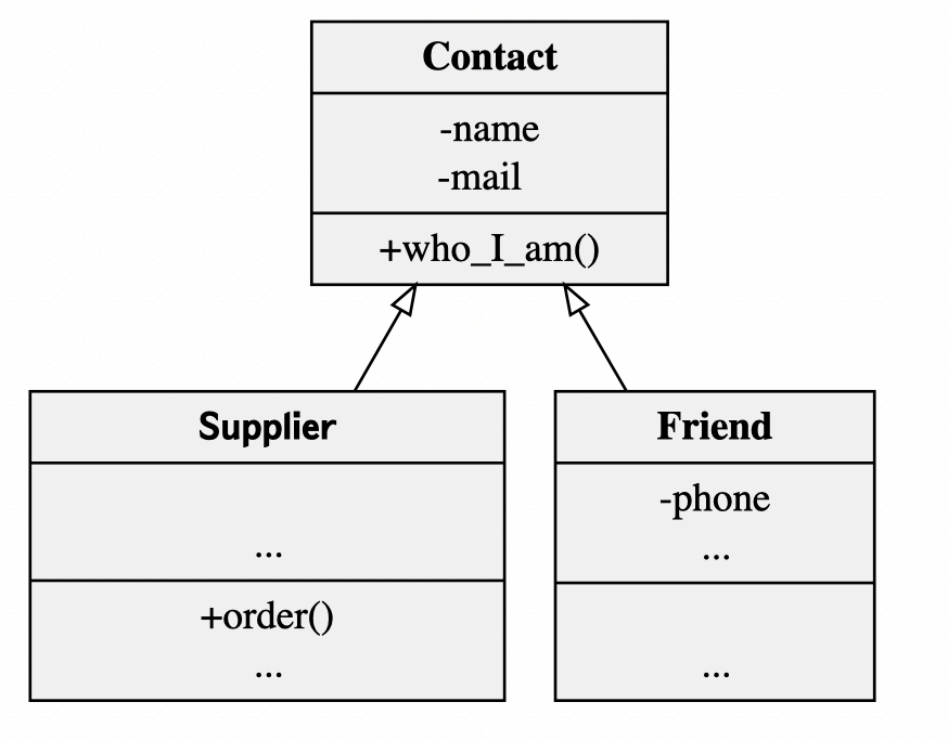
Example

- Define a `Contact` class. Instances have attributes **name** and a **email**.
- When the object is instantiated, the name and email are initialized, and the formal correctness of the email address is checked[1].
- Instances have a method `who_I_am()` that returns a string composed of name and email.

---

[1]: Trivial approach: email address must contain `@`, with other characters before and after `@`. Use the String `split()` method.

For a complete solution, you can use the **regular expression** module `re` - https://docs.python.org/3/library/re.html#



- `Supplier` (subclass of `Contact`) has a method `order()` to place purchase orders (*the method merely prints a string*).
- `Friend` (subclass of `Contact`) stores the phone number during its creation.

```
c = Contact("pippo", "pippo@gmail.com")
s = Supplier("pluto", "pluto@gmail.com")

print(c.name , c.email)
print(s.name , s.email)

# c.order("mouse")
# AttributeError: "Contact" object has no attribute "order"
s.order("mouse")

f = Friend("goofie", "goofie@gmail.com", "123123")
```