OOP:
Composition and Aggregation

Inheritance - revision

Inheritance:

allows us to derive a new class based on existing (base) class

Models the **IS-A** relationship

For example

- Cat is a *subclass* of Animal
- Student is a *subclass* of Person

Composition/Aggregation

Composition/Aggregation:

allows us to create a new class using other existing classes as parts

Models the **HAS-A** relationship

Composition vs. Aggregation

Aggregation

implies a relationship where the child can exist independently of the parent.

Example: Module and Student

Composition

implies a relationship where the child cannot exist independently of the parent.

Example: House and Room

Aggregation - example

A college module can be defined by Module name and list of students taking that module.

We'll use Aggregation to make a class Module that will contain objects of the class Person

```
class Module:
    def __init__(self, name):
        self._name = name
        self._students =[]

    def get_module_name(self):
        return self._name

    def add_student(self, p):
        self._students.append(p)

    def get_students(self):
        return self._students

    def print_students(self):
        if self._students == []:
            print("No students")
        else:
            for i in range(len(self._students)):
                print(self._students[i])
```

```
class Person (object):

    def __init__(self, name, age, address):
        self._name = name
        self._address=address
        self._age = age

    def __str__(self):
        return "Person: name "+self._name+" age:"+str(self._age)
```

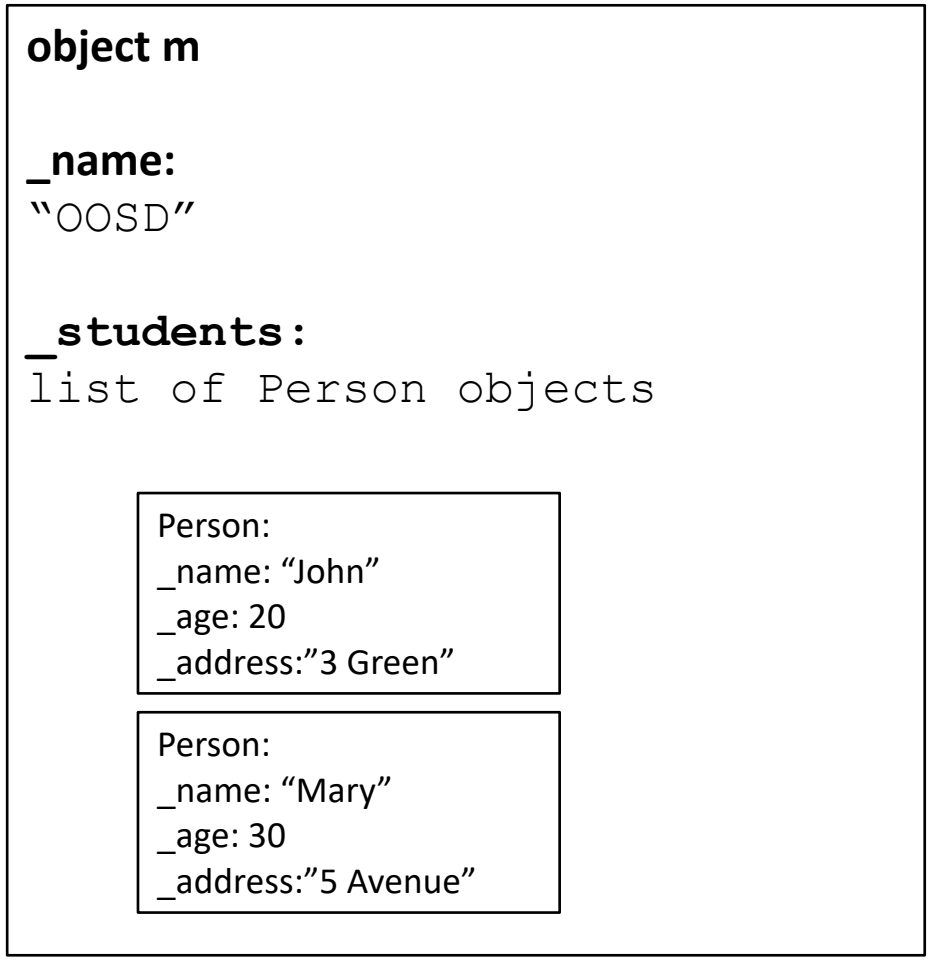
Example: Testing our classes

```
p1 = Person ("John", 20, "3 Green")
p2 = Person ("Mary", 30, "5 Avenue")

m = Module("OOSD")
m.add_student(p1)
m.add_student(p2)


print("Course: "+ m.get_module_name())

students = m.get_students()
m.print_students()
```



Keep in mind unwanted side-effects!

```
p1.change_age(15)   #you'll have to add the change_age method in Person class
m.print_students()
```

will print

```
Person: name John age: 15
Person: name Mary age: 30
```

Keep in mind unwanted side-effects!

However,

```
p1 = Person(“Ann”, 35, “Summerfield”)
m.print_students()
```

will still print

```
Person: name John age: 15
Person: name Mary age: 30
```

Example (cont.)

You can also populate the course with students without using individual variables

```
m.add_student(Person("John", 20, "3 Green"))
m.add_student(Person("Mary", 30, "5 Avenue"))

m.print_students()
```

Example: Car

A car has make, model, engine, tyres

class Car

- Make, Model
- Class Engline – size
- Tyres – brand, tire depth

Exercises

1. We-write the shapes classes from couple of weeks ago and instead of using two numbers, x and y, for the coordinates of the share, use an object of class Point instead.
2. Write a class Book – each book has a title (string) and one or more authors. Write a class to represent an author – each author has a name (string) and an email address (string)
3. Using the bank account classes we developed few weeks ago, write a class Bank. Every bank has a name, a list of savings accounts, and a list of current accounts.

Exercises (cont.)

4. You can write a class to represent anything you want. For example, write a class to represent an Address – each address has a country, city, street name and house number.

   Modify your Person class to use Address objects instead of string for representing the address.