Introduction to Object-Oriented Programming

- **Conception et programmation orientées object**, B. Meyer, *Eyrolles*
- **Object-Oriented Software Engineering**, T. C. Lethbridge, R. Laganière, *McGraw Hill*
- **Design Patterns Explained**, A. Shalloway, J. R. Trott, *Addison-Wesley*
- **Design Patterns, Elements of Reusable Object-Oriented Software**, E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Addison-Wesley*

Before the object-oriented paradigm: functional decomposition

- Functional decomposition is a natural way to deal with complexity.
- It consists in breaking down (decomposing) a problem into the functional steps that compose it.

  e.g.: a cooking recipe, the instructions to assemble furniture, etc.
- This approach is often used because it is more natural.
- The problem with functional decomposition is that it does not help to prepare the code for possible changes in the future.

Many bugs originate with changes to code

The problem of requirements

What software developers say about the requirements they get from users:

- Requirements are incomplete
- Requirements are usually wrong
- Requirements (and users) are misleading
- Requirements do not tell the whole story

Requirements always change

The Object-Oriented Paradigm

- The object-oriented paradigm is centered on the concept of the object.
- Everything is focused on objects, not functions.
- The advantage of using objects is that the things that are defined are responsible for themselves: - Objects inherently know what type they are. - The data in an object allow it to know what state it is in and the code in the object allows it to function properly

What is an object?

Objects have been traditionnaly defined as data with *methods* (the object-oriented term for functions).

- At the conceptual level, an object is a set of responsibilities
- At the specification level, an object is a set of methods that can be invoked by other objects or itself
- At the implementation level, an object if code and data

What is a class?

- Objects are organized around the class
- A class is a definition of the behavior of an object
- It contains a complete description of:
    - The data elements that the object contains
    - The methods that the object can do
    - The way these data elements and methods can be accessed
- Since the data elements an object contains can vary, each object of the same type may have different data but will have the same functionality (as defined in the methods).

Objects are *instances* **of classes**

Review of Object-Oriented terminology

Object:

An entity with responsabilities. A special, self-contained holder of both data and methods tha operate on that data.

Class:

define the methods and data of an object of its type

Instance:

A particular object of a class

Instantiation:

The process of creating an instance of a class

Superclass:

A class from which other classes are *derived*. Contains the master definitions of attributes and methods that all derived classes will use (and possibly override)

Derived class:

A class that is specialized from a superclass. Contains all of the attributes and methods of the superclass bu may also contain other attributes or different methods implementations

Inheritance:

The way that a class is specialized

Attribute:

Data associated with an object

Method:

A function that is associated with an object

Visibility:

Objects do not have to expose everything (data and methods) to other objects. In object-oriented systems, the main types of accessibility are:

- *public*: anything can see it
- *protected*: only objects of this class and derived class can see it
- *private*: only objects from this class can see it

Encapsulation:

Typically defined as data-hiding, but better thought of as any kind of hiding (i.e., internal data members are to exposed externally)

Polymorphism:

The ability of related objects to implement methods that are specialized to their type