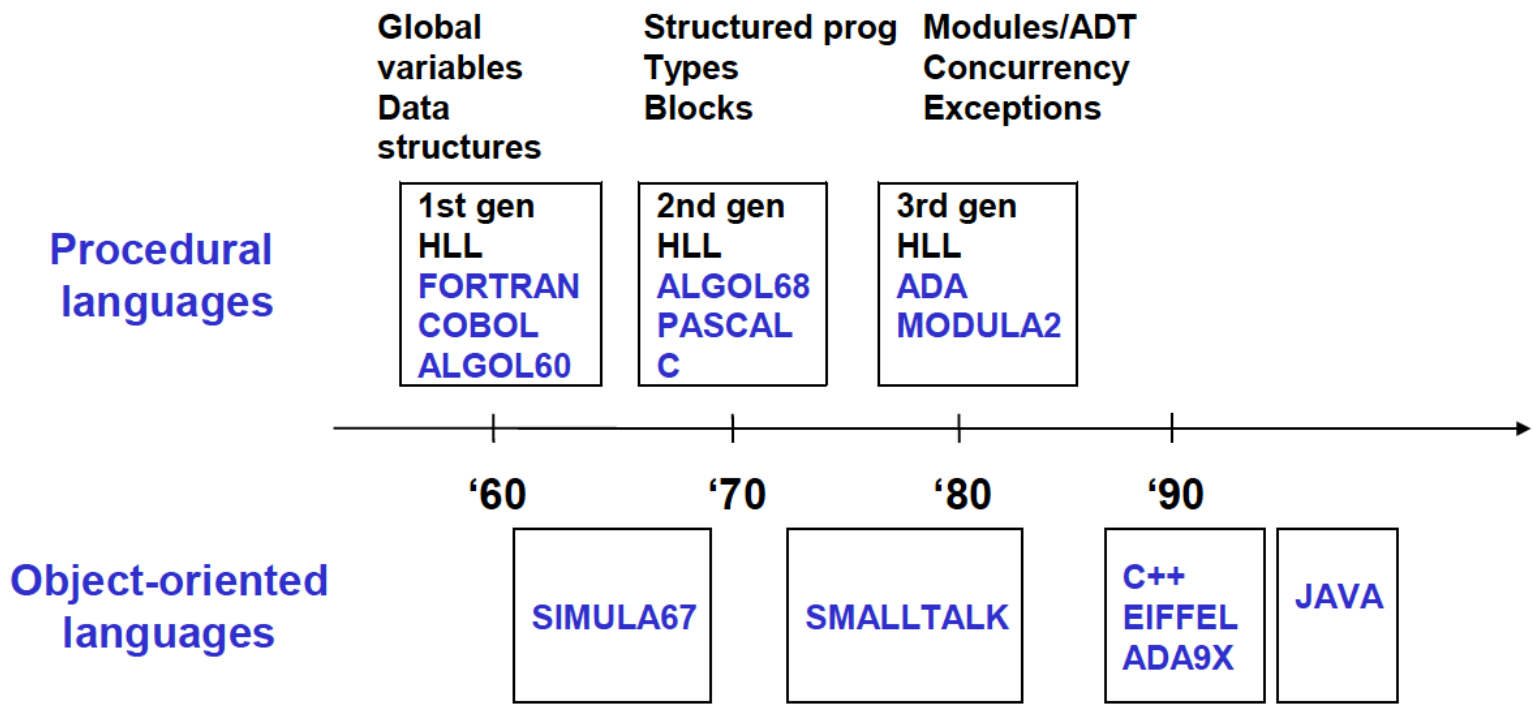
Introduction to Object Oriented Programming

Università di Modena e Reggio Emilia
Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)

ING

Languages Timeline



Why OOP?

- Procedural programming (e.g., Pascal, C) are not suitable for building large software infrastructures
- OOP addresses this issue and reduces development and maintenance costs for large and complex software projects

Limits of procedural programming

- Reuse of code limited
  - Data and procedures (functions) are separate. This makes it complex to reuse existing code in other projects
- Data protection limited
  - Unprotected data accessible from vast portions of the source code. After a certain stage, debug becomes a nightmare!
- Unsuitable for large systems
  - Large scale projects require large scale working force (many teams). Unprotected data, separate from functions makes it hard to decompose the global effort

Software crisis (1970)

- The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:
  - Projects running over-budget
  - Projects running over-time
  - Software was very inefficient
  - Software was of low quality
  - Software often did not meet requirements
  - Projects were unmanageable and code difficult to maintain
  - Software was never delivered

Software defects

- The book "Code Complete" by Steve McConnell summarize error expectations as follows:
  - Industry Average: about 15 - 50 errors per 1000 lines of delivered code
  - Microsoft Applications: about 10 - 20 defects per 1000 lines of code during in-house testing, and 0.5 defect per **KLOC** (**KLOC** IS CALLED AS 1000 lines of code)
  - The '*cleanroom development*' technique developed by Harlan Mills achieve rates as low as 3 defects per 1000 lines of code during in-house testing and 0.1 defect per 1000 lines of code in released product

Some data…

Software	Lines of codes	Bugs
Windows XP	45M	22K
Windows Vista	50M	25K
Debian 5.0	324M	4860K
Linux kernel 2.6.35	13M	195K
Mac OS X 10.4	86M	1290K
OpenOffice	1M	15K
Gimp	0.6M	9K

* http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

OOP Goal

- Build well-managed software which can be:
  - Secure, re-usable, reconfigurable, flexible, documentable, extensible

- Instead of focusing on algorithms, optimization and efficiency, the OOP focus on programming techniques to build software

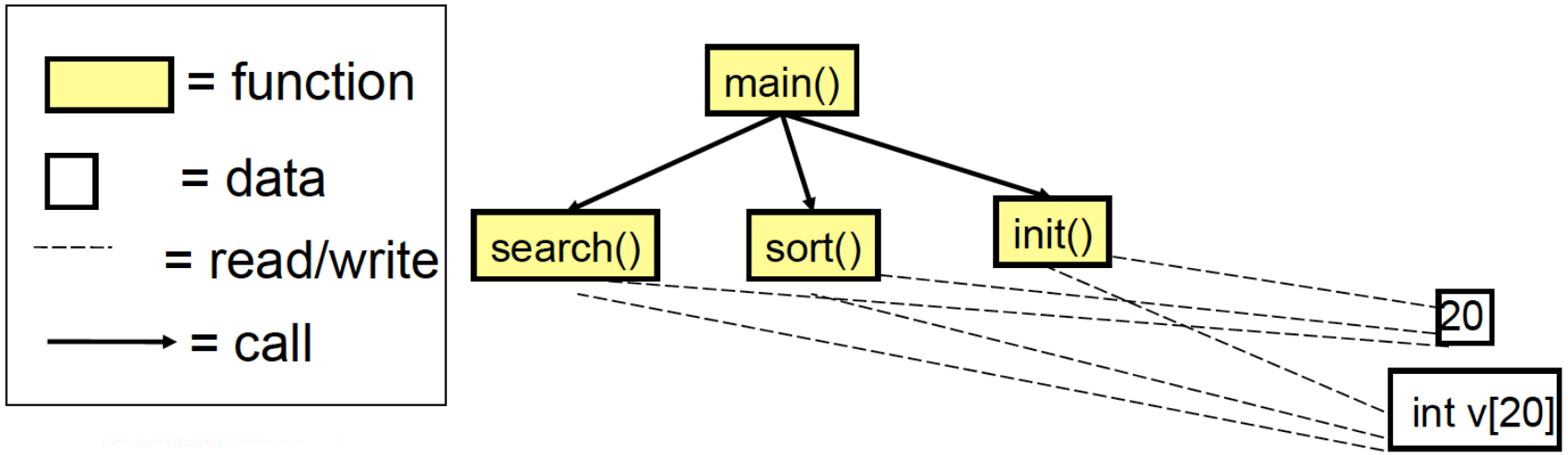
Which direction?

- The growing complexity of the reality we want to model requires a new approach so that:
  - At the project stage, allows to consider software as a set of well-defined ***entities*** (if required also complex)
  - At the implementation stage, allows to **identify** functions that uses some data and prevent them being accessed from other functions

Procedural Programming

```
int vect[20];
void sort() { /* sort */ }
int search(int n){ /* search */ }
void init() { /* init */ }
// …
int i;
void main(){
   init();
   sort();
   search(13);
}
```

Modules and relationships

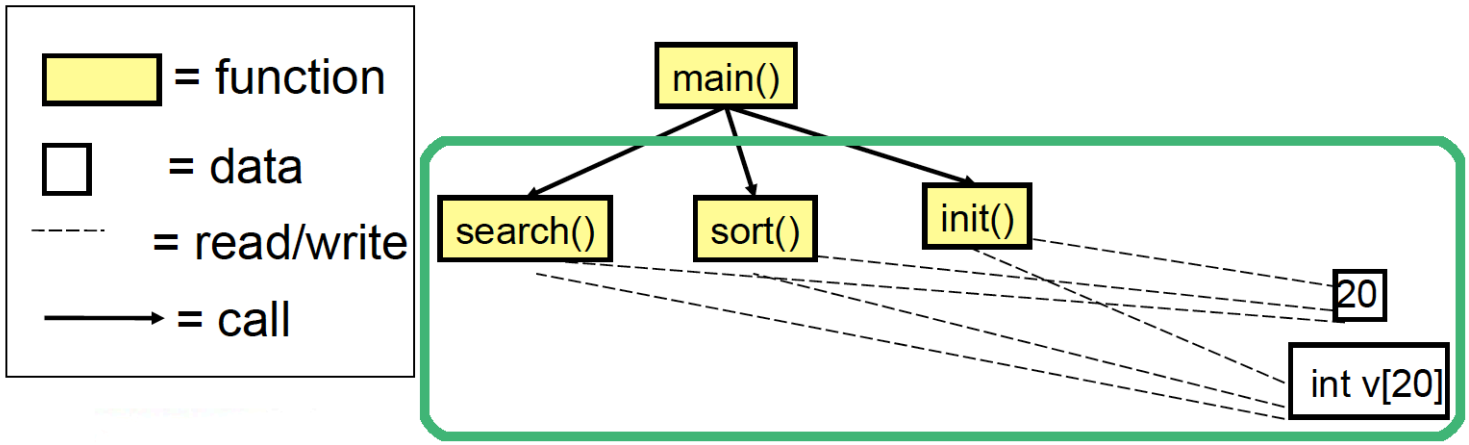


Issues

- There is no syntactic relationship between:
  - Vectors (int vect[20])
  - Operations on vectors (search, sort, init)
- There is no control over size:
  - for (i=0; i<=20; i++) { vect[i]=0; };
- Initialization
  - Actually performed?

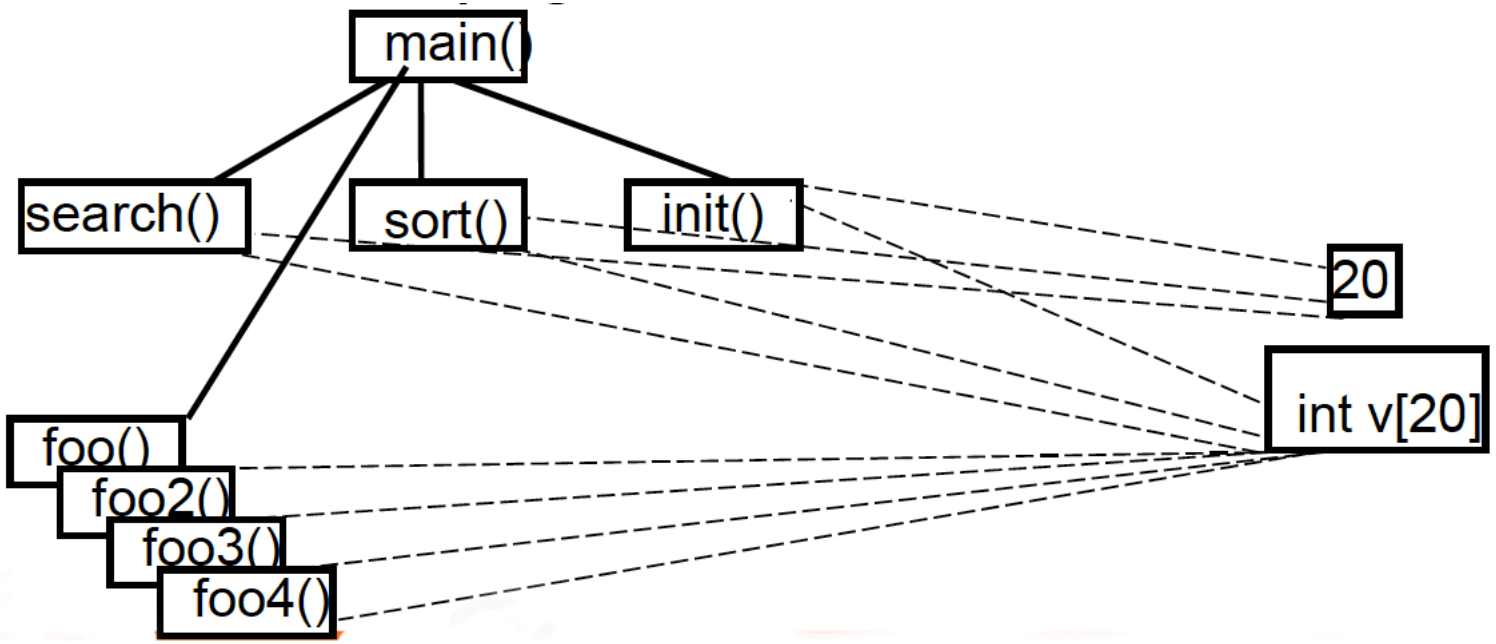
Issues

- It's not possible to consider a vector as a primitive and modular concept
- Data and functions cannot be modularized properly



Issues, on the long run

- External functions can read/write vector's data
- As time goes by, this leads to a growing number of relationships
- Source code becomes difficult to understand and maintain
  - Problem known as "Spaghetti code"



What is OO?

- Procedural Paradigm
  - Program defines data and then calls subprograms acting on data
- OO Paradigm
  - Program creates objects that encapsulate the data and procedures operating on data
- OO is simply a new way of organizing a program
  - Cannot do anything using OO that can't be done using procedural paradigm

Why OO?

- Programs are getting too large to be fully comprehensible by any person
- There is need of a way of managing very large projects
- Object Oriented paradigm allows:
  - programmers to use large blocks of code
  - without knowing all the picture
- Makes code reuse a real possibility
- Easier maintenance and evolution of code

Why OO?

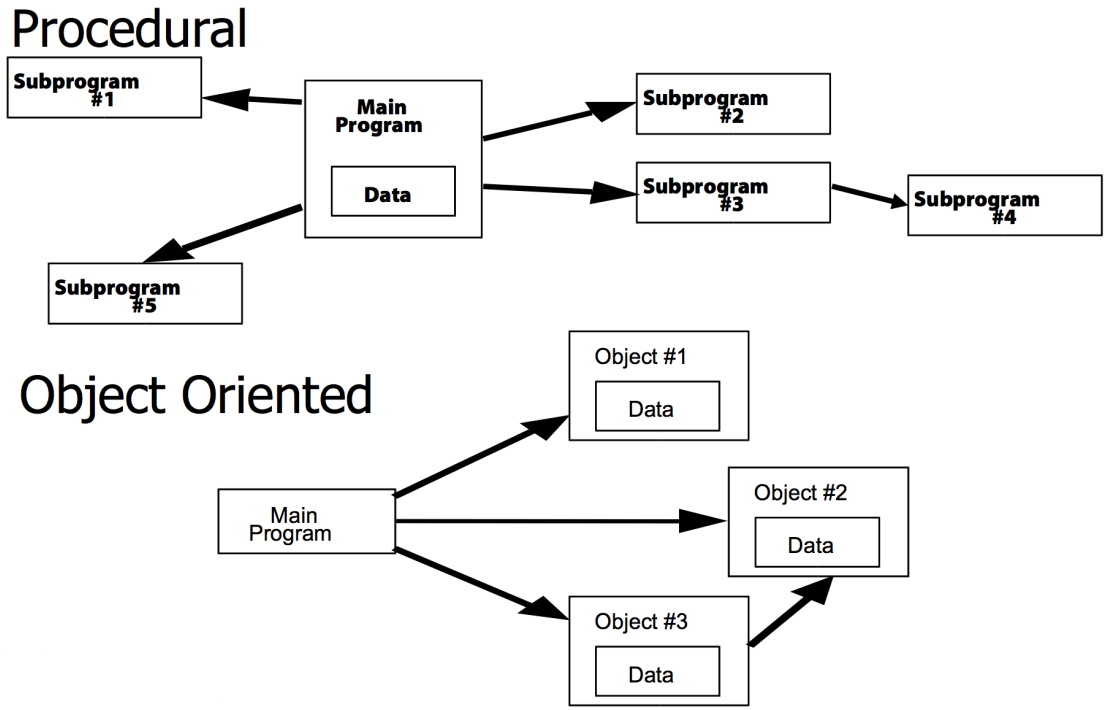
- Benefits only occur in larger programs
- Analogous to structured programming
  - Programs < 30 lines, spaghetti is as understandable and faster to write than structured
  - Programs > 1000 lines, spaghetti is incomprehensible, probably doesn't work, not maintainable
- Only programs > 1000 lines benefit from OO really

An engineering approach

- Given a system, with components and relationships among them, we have to:
  - Identify the components
  - Define component interfaces
  - Define how components interact each other through their interfaces
  - Minimize relationships among components

An engineering approach

- Objects add an additional abstraction layer
- More complex system can be built



Object-Oriented approach

- Defines a new component type
  - Object (and class)
  - Data and functions on data are within the same module
  - Allows defining a more precise interface
- Defines a new kind of relationship
  - Message passing
  - Read/write operations are limited to the object scope

Object-Oriented approach

```
class Vector {
    //data
    private int v[20];

    //interface
    Public Vector() {
        for(int i=0; i<20; i++) v[i]=0;
    }

    public sort(){ /*sort*/ }
    public search(int c){ /*search*/ }
}
```

Object-Oriented approach

- Use of the class Vector:

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.sort(); v1.search(22);

V1++; //Error: not an integer
V1.v[2] = 47; //Error: v[] is private
```

Class and object

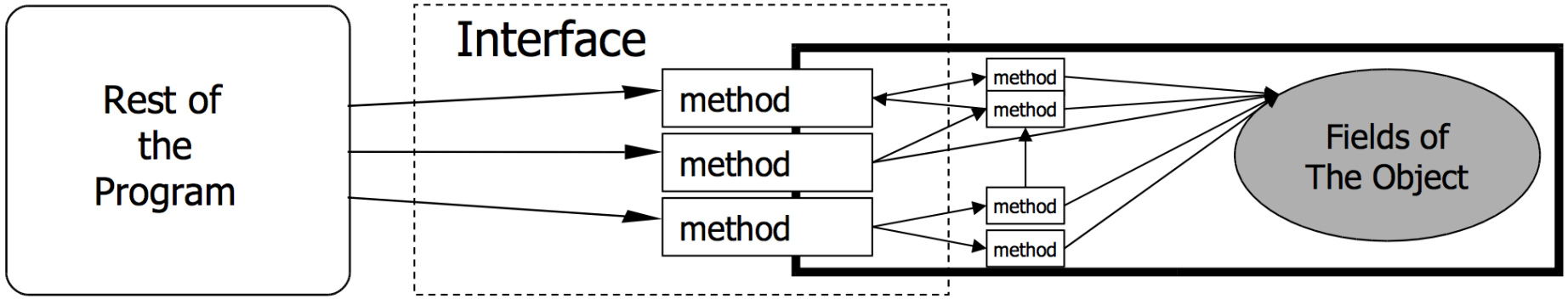
- Class (the description of object structure, i.e. type):
  - Data (ATTRIBUTES or FIELDS)
  - Functions (METHODS or OPERATIONS)
  - Creation methods (CONSTRUCTORS)
- Object (class instance)
  - State and identity

Class and object

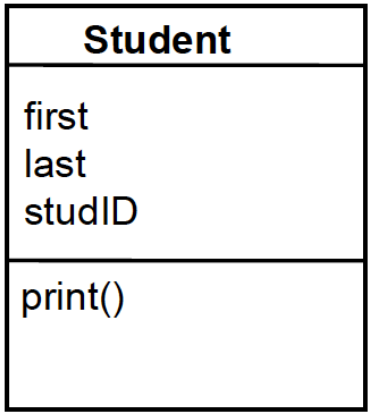
- A class is like a type definition. No data is allocated until an object is created from the class
- The creation of an object is called instantiation. The created object is often called an instance
- No limit to the number of objects that can be created from a class
- Each object is independent. Changing one object doesn't change the others

Class and interface

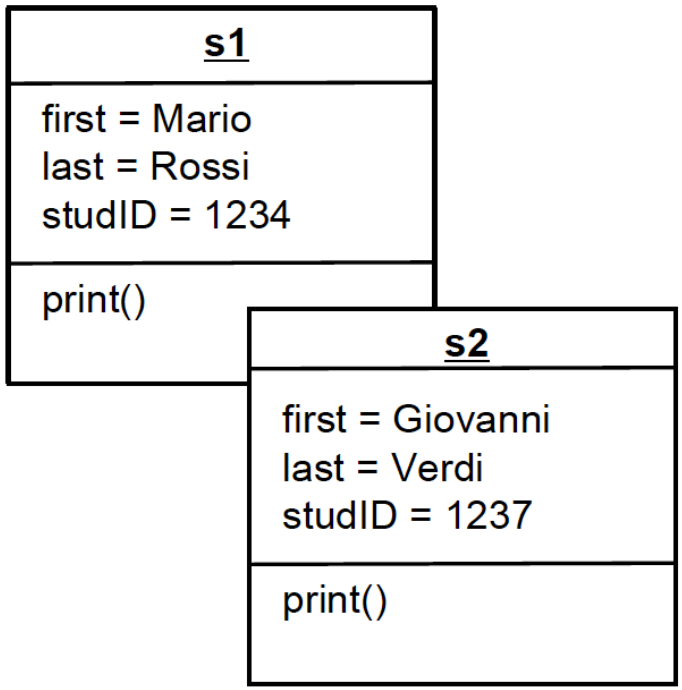
- A class interface represents operations (methods) that external objects can call.
- Methods are gateways to the internal state.



UML



class



objects

Client-Server Model

- Client
  - Knows the server
  - Uses a service
- Server
  - Does not know clients a-priori
  - Offers services

- **OOP implies a paradigm shift:**
  - Do not uses functions for processing data entities
  - Ask entities to deliver services

Client-Server Model

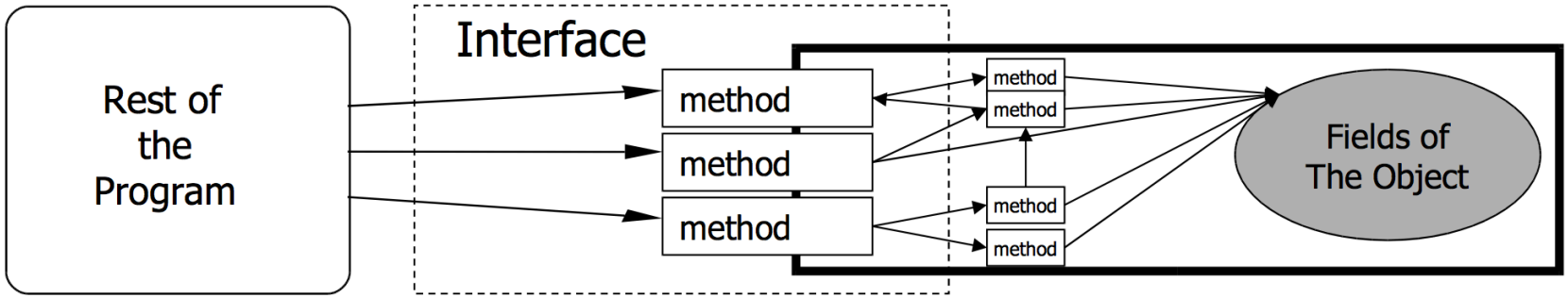
Procedural	**OOP**
operation(object, params)	*object.operation (params)*
Example insert(list, element)	Example list.insert(element)

Characteristics of OOP

- ***Encapsulation***
- ***Inheritance***
- ***Polymorphism***

Encapsulation

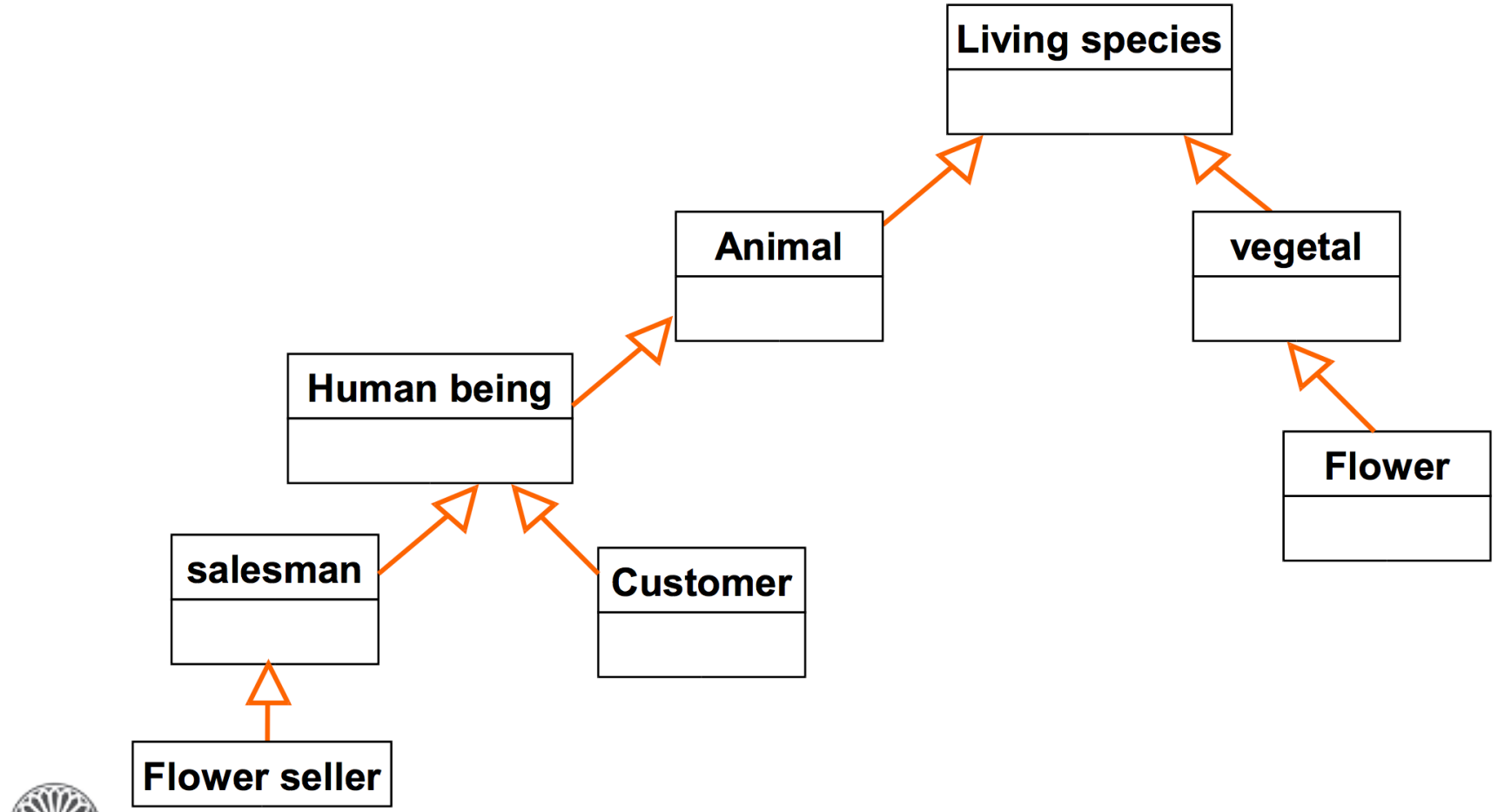
- Each object ”wrap” code and data and protect it’s data from the outside
- Each object handles it’s data
- External objects can use the object’s interface



Inheritance

- A class can derive from another class by extending it
- The derived class inherit variables and methods of the base class. The child class adds its own variables and methods
- Simplify the code reuse, build relations among classes

Inheritance



Polymorphism

- Same requests for a method may lead to different behaviors depending on:
  - The object which perform the operation
  - The execution context
  - Type of parameters passed as argument

Example: Courses management

- Management software for degree courses
- As the first step, we must identify which entities of the real world we should model as classes:
  - Degree courses(CdL)
  - Students
  - Exams
- But also
  - Counters
  - Date
  - …

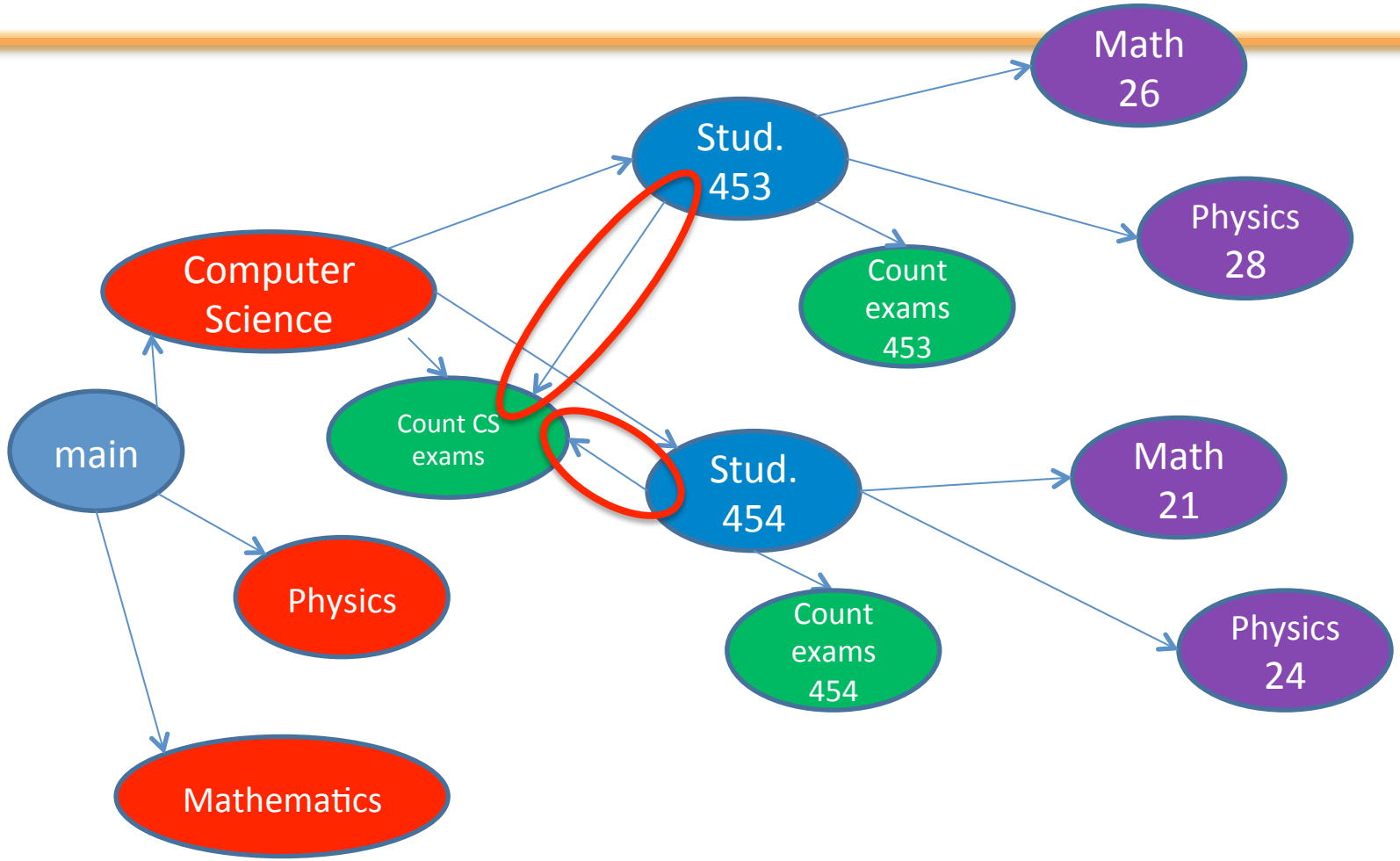
Define attribute and methods

- **Class CdL**
  - attributes: name, exams, enrolled students, global counter for passed examinations
  - methods: add student, get student, know how many exams a student succeed
- **Class Student**
  - attributes: name, surname, registration number, enroll date, enroll year, list of attended courses, number of passed exams
  - methods: update passed exams and number of exams, set enroll year, know if student passed a specific exam
  - NOT ALLOWED: Change enroll date or number of passed exams, change name/surname
- **Class Exam**
  - attributes: course name, passed/failed, attended date, grade
  - methods: set grade, read course name and obtained grade
  - NOT ALLOWED: change grade, change passed or fail state.
- **Class counter:** as mentioned above

Create objects

- At the execution time objects representing entities of the real world will be created:
  - One object **CdL** for each degree course
  - One object **Student** for each enrolled student. The same object will be attribute of the object CdL which belongs
- One object **Exam** for each passed exam. The object will be attribute of the object student who succeed it

Runtime


Math
26
Stud.
453
Physics
28
Computer
Science
Count
exams
453
Count CS
exams
main
Stud.
454
Math
21
Physics
Count
exams
454
Physics
24
Mathematics

Advantages of OOP

- Simplify the process of building software in a cooperative manner:
  - Different peoples develope the same classes
  - Each developer can easily test the class behaviour by initializating the corresponding object and calling the related methods
  - Respect the contract: Define an interface (methods and properties) for each class

Advantages of OOP (2)

- Simplify code management
  - Bugs on object data are easy to spot. Since data are not visible from the outside, errors occurs in the object that handles data
  - Changes on a specific class doesn't impact other classes (unless the interface is not modified)

Advantages of OOP(3)

- Support incremental design and development
  - Define new classes by extending the exsisting ones (e.g. new class student starting from an exsisting class people)
- Chance of rapid prototyping
  - Testing object without caring of entirely define the class behaviour

OOP's cost

- Needs a Object Oriented way of thinking
- Complex design (e.g., Which classes, How many?)
- Expensive infrastractures for simple applications