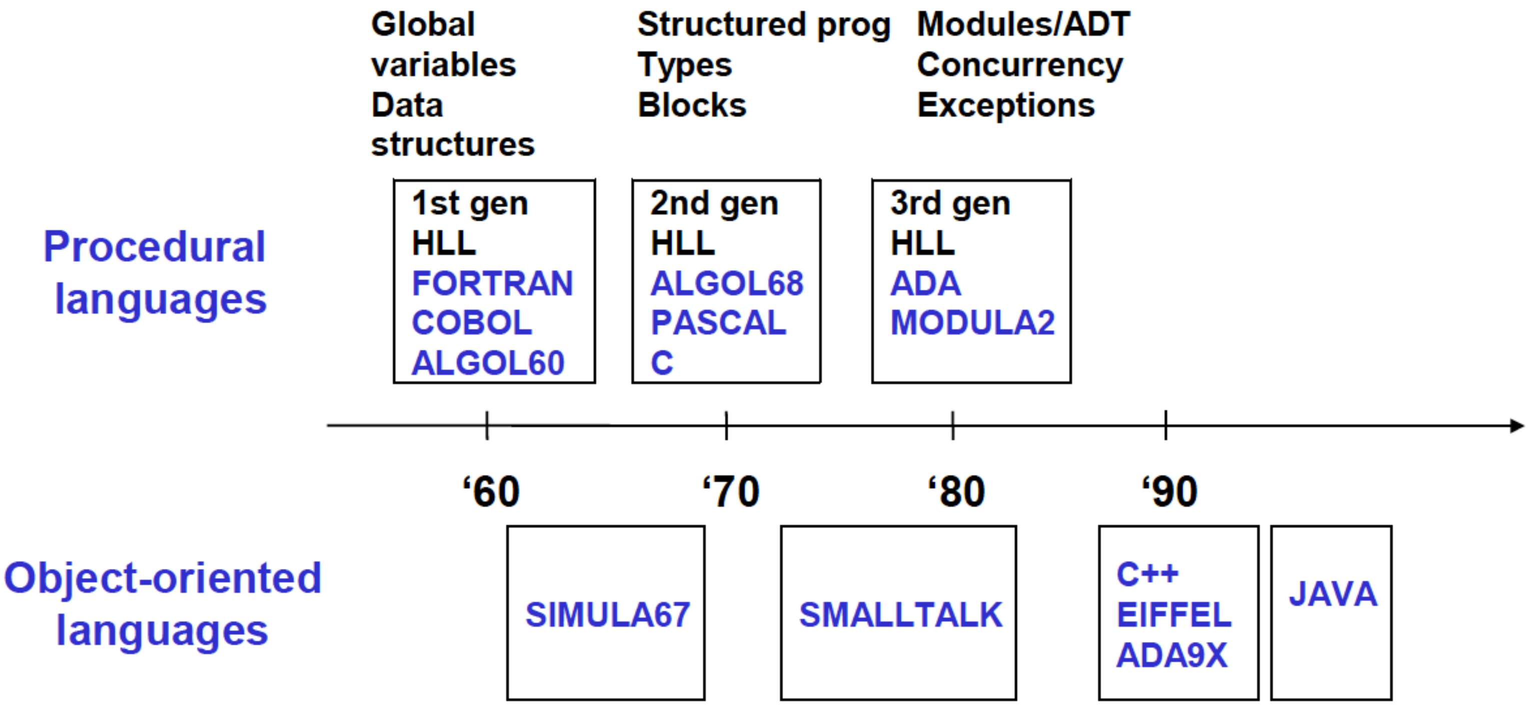
Introduction to Object Oriented Programming

Università di Modena e Reggio Emilia

Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)

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 1000 lines of delivered 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 delivered code

Some data…

Software	Lines of codes
Windows XP	45M
Windows 10	55M
Debian 7.0	419M
Linux kernel 3.6	16M
Mac OS X 10.4	86M
OpenOffice	1M
Gimp	0.6M

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

OOP Goal

- Build software being:
  - Secure, re-usable, reconfigurable, flexible, documentable, extensible

- Instead of focusing on algorithms, optimization and efficiency, **OOP focus on programming techniques**

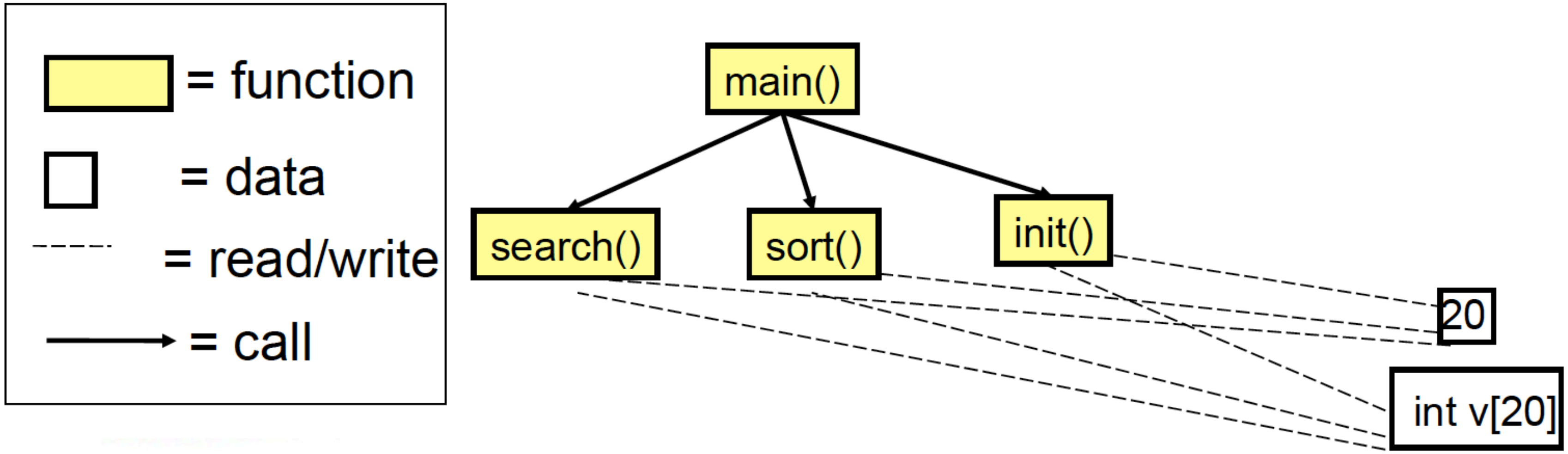
Which direction?

- The growing complexity of the reality we want to model requires approaches allowing:
  - During design, to consider software as a set of well-defined ***entities containing*** *both* ***data and functions*** *and* ***preventing other entities to access the same data***

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

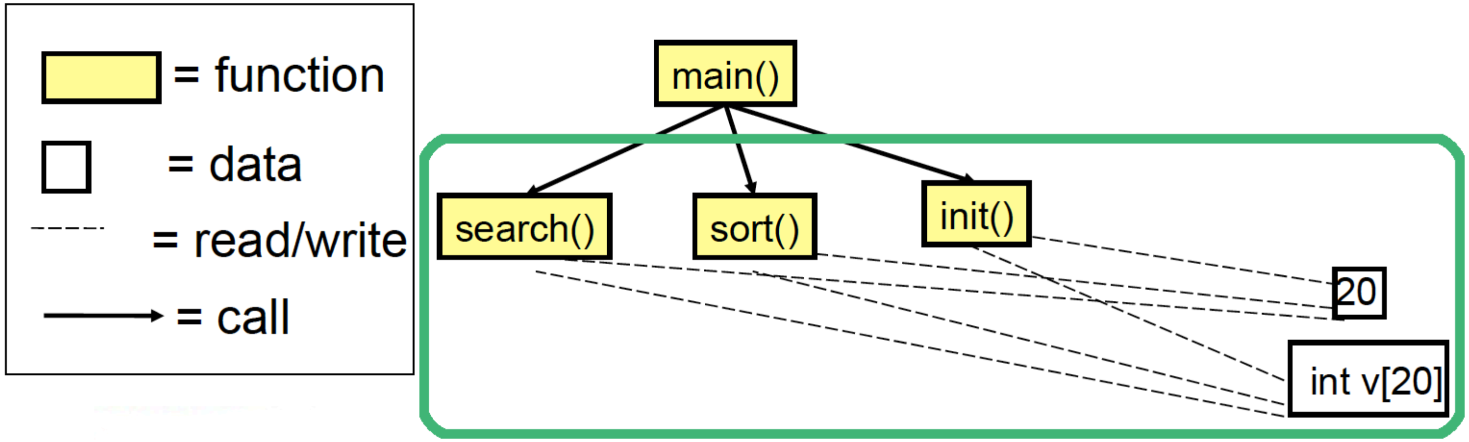


Issues

- 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?

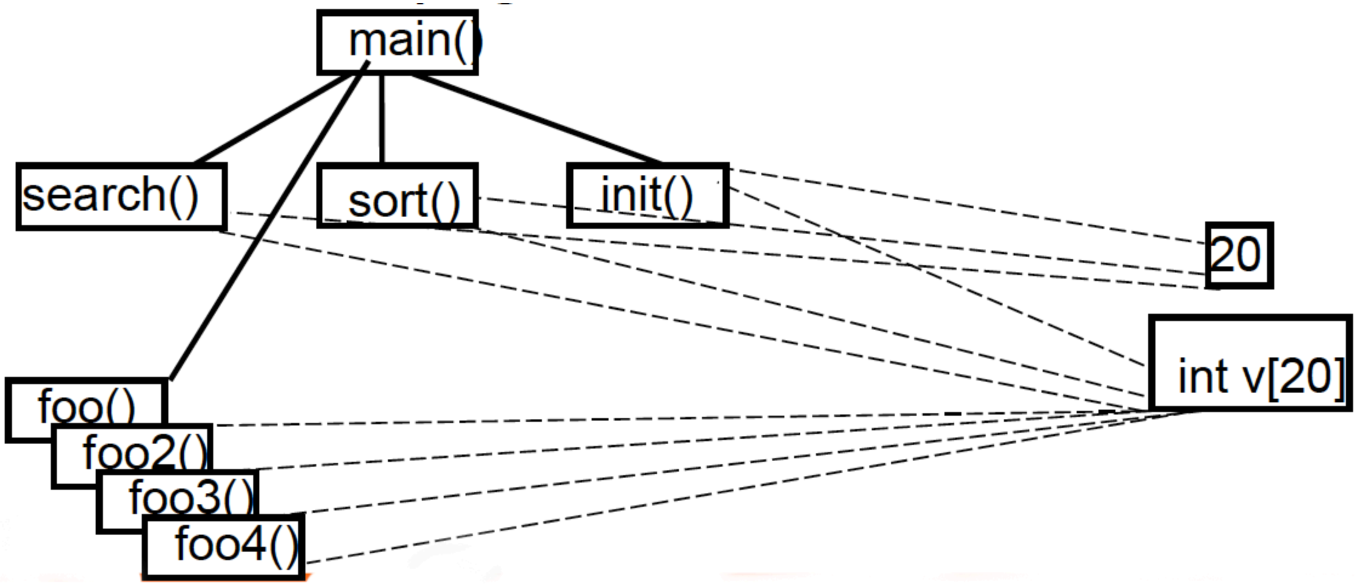
Issues

- 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, leading to a **growing number of relationships**
- Source code becomes **difficult to understand and maintain**



What is OO?

- Procedural Paradigm
  - Program defines data and then calls subprograms acting on data
- OO Paradigm
  - Program creates objects **encapsulating data and procedures operating on data**
- OO is only a new way of organizing code
  - 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 the whole picture
  - code reuse a real possibility
  - easier maintenance and evolution of code

Why OO?

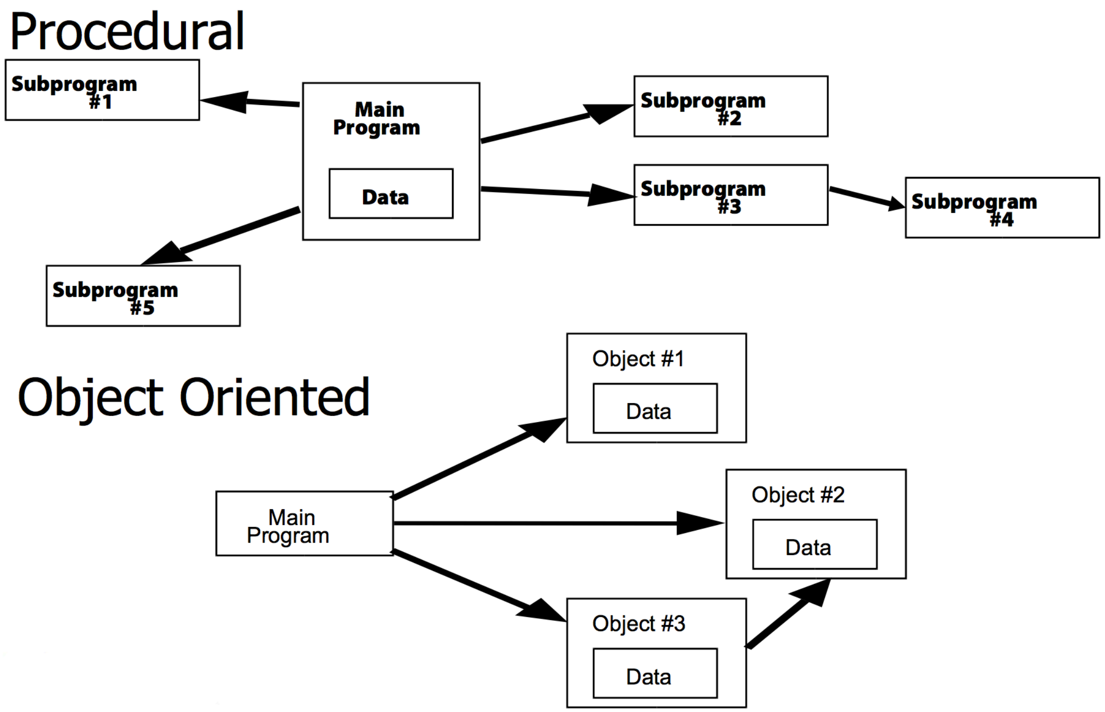
- Benefits only occur in larger programs
  - Programs < 30 lines, spaghetti is as understandable and faster to write than structured
  - Programs > 1K lines, spaghetti is incomprehensible, probably doesn't work, not maintainable
- Only programs > 1K lines benefit from OO really

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 systems can be built



Object-Oriented approach

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

Object-Oriented approach

```
class MyVector {
    // internal data (attributes)
    private int v[20];

    // external interface (methods)
    public MyVector() {
        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 MyVector:

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

v1++;          //Error: not an integer
v1.v[2] = 47; //Error: v[] is private
```

Class and object

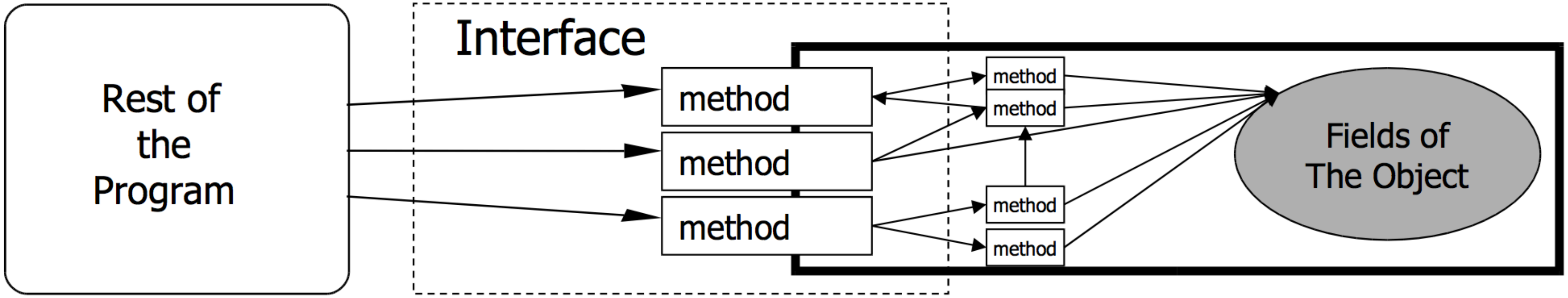
- Class (the description of object structure, i.e. type):
  - Data (ATTRIBUTES or FIELDS)
  - Functions (METHODS or OPERATIONS)
  - Creation methods (CONSTRUCTORS)
- Object (class instance)
  - Identity
  - State

Class and object

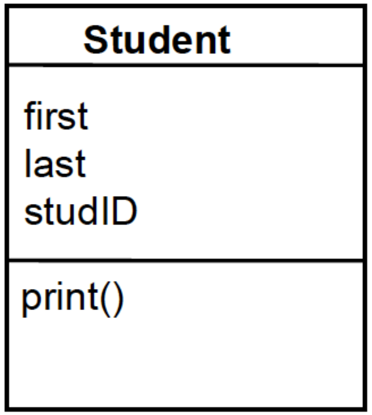
- A class is like a type definition. No data is allocated until an object is created from the class
- The creation of an object is called instantiation. The created object is often called an instance
- No limit to the number of objects that can be created from a class
- Each object is independent. Changing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

Class and interface

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



UML – Unified Modeling Language

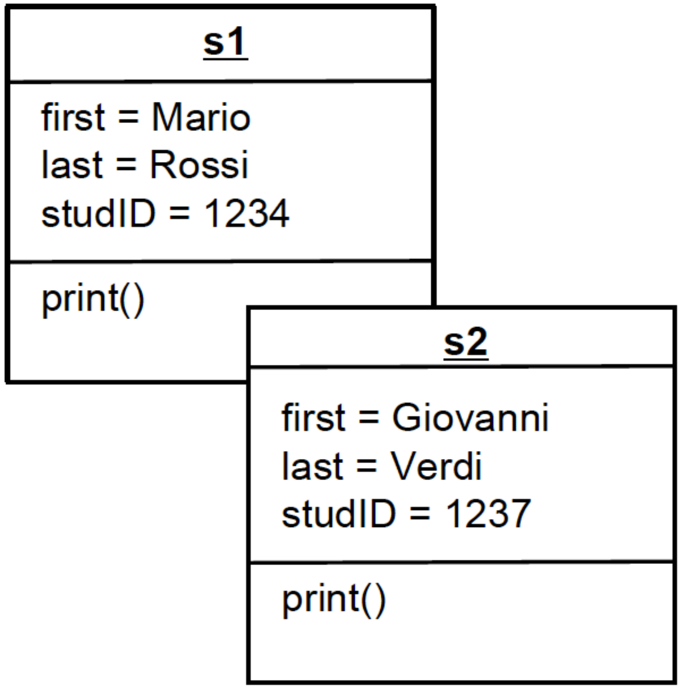


Student

first
last
studID

print()

class



s1

first = Mario
last = Rossi
studID = 1234

print()

s2

first = Giovanni
last = Verdi
studID = 1237

print()

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

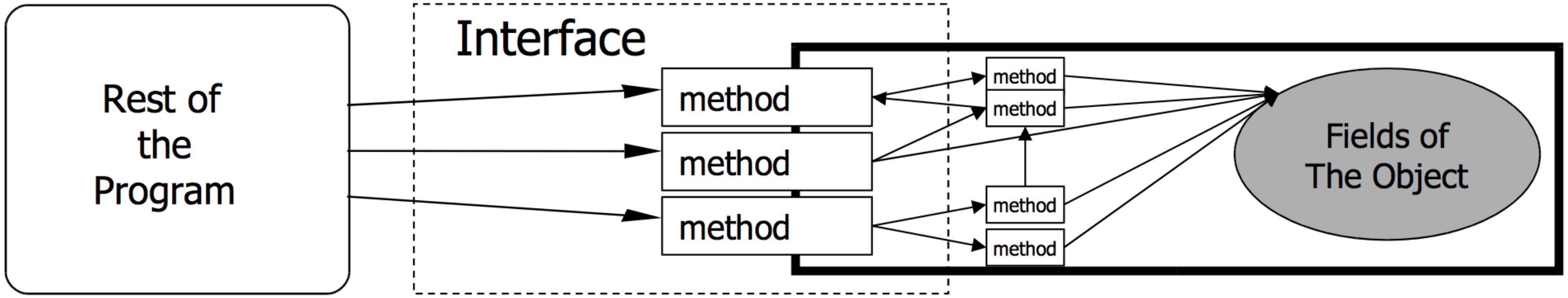
Procedural	**OOP**
operation(object, params)	*object.operation(params)*
For example:	For example:
insert(list, element)	*list.insert(element)*

OOP Key Features

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

Encapsulation

- Each object “wraps” code and data
- Each object handles its own data
- Other objects can use the object’s interface to require services



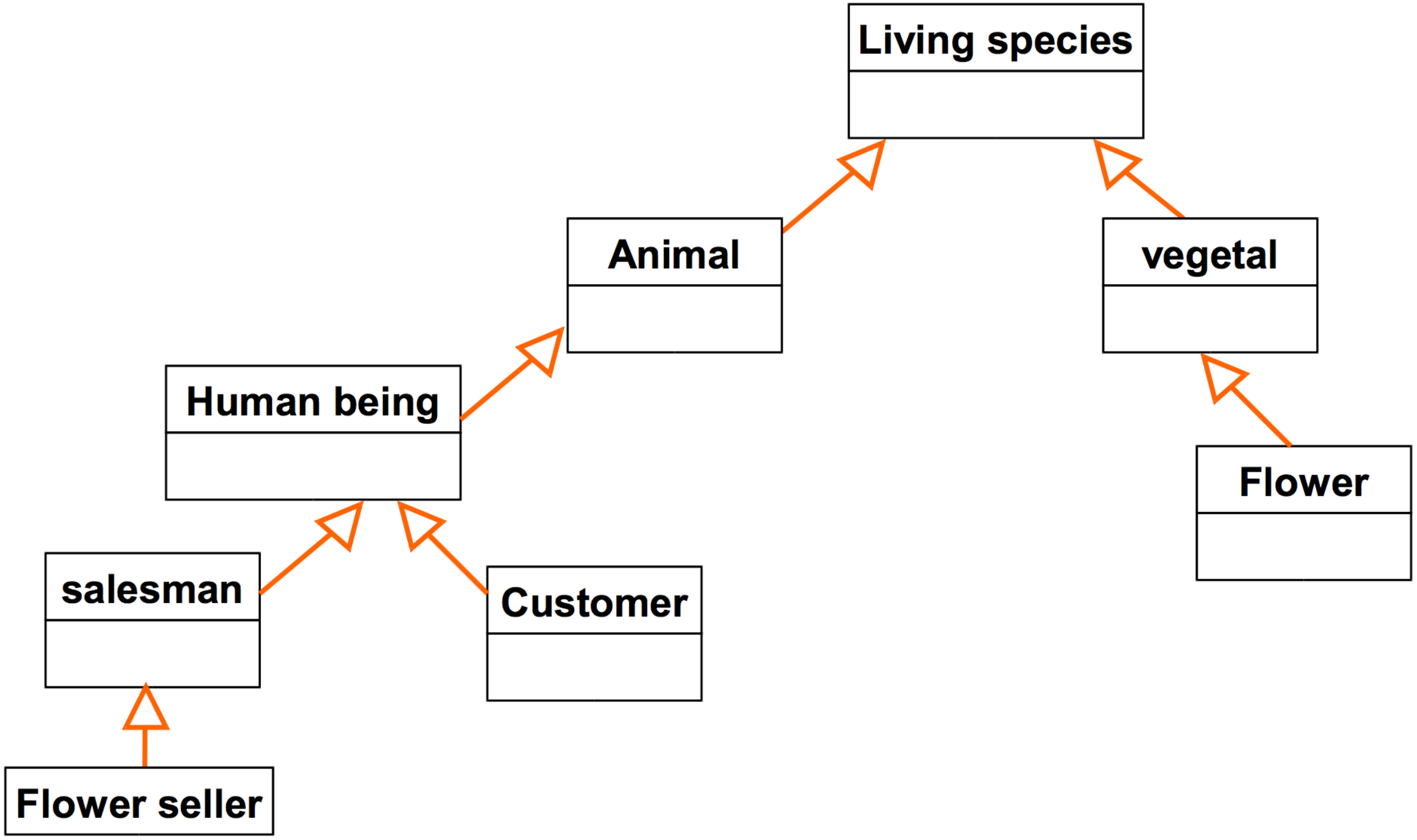
Encapsulation

```
class MyVector {
    // internal data (encapsulated data)
    private int v[20];

    // external interface (methods)
    public MyVector() {
        for(int i=0; i<20; i++) v[i]=0;
    }

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

Inheritance

- A class can derive from another class by extending it
- The derived class inherits 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



Inheritance

```
public class LivingSpecies {
  private boolean isAlive;
}

public class Animal extends LivingSpecies {
. . .
}

public class HumanBeing extends Animal {
. . .
}
```

Polymorphism

- Same requests for a method may lead to different behaviors depending on:
  - The actual object performing the operation (e.g., base class vs derived class)
  - Type of parameters passed as argument
  - Execution context

Example: Courses management

- Management software for courses
- As a first step, we must identify which real world entities should be modeled as classes:
  - Courses
  - Students
  - Exams
- But also
  - Counters
  - Date
  - …

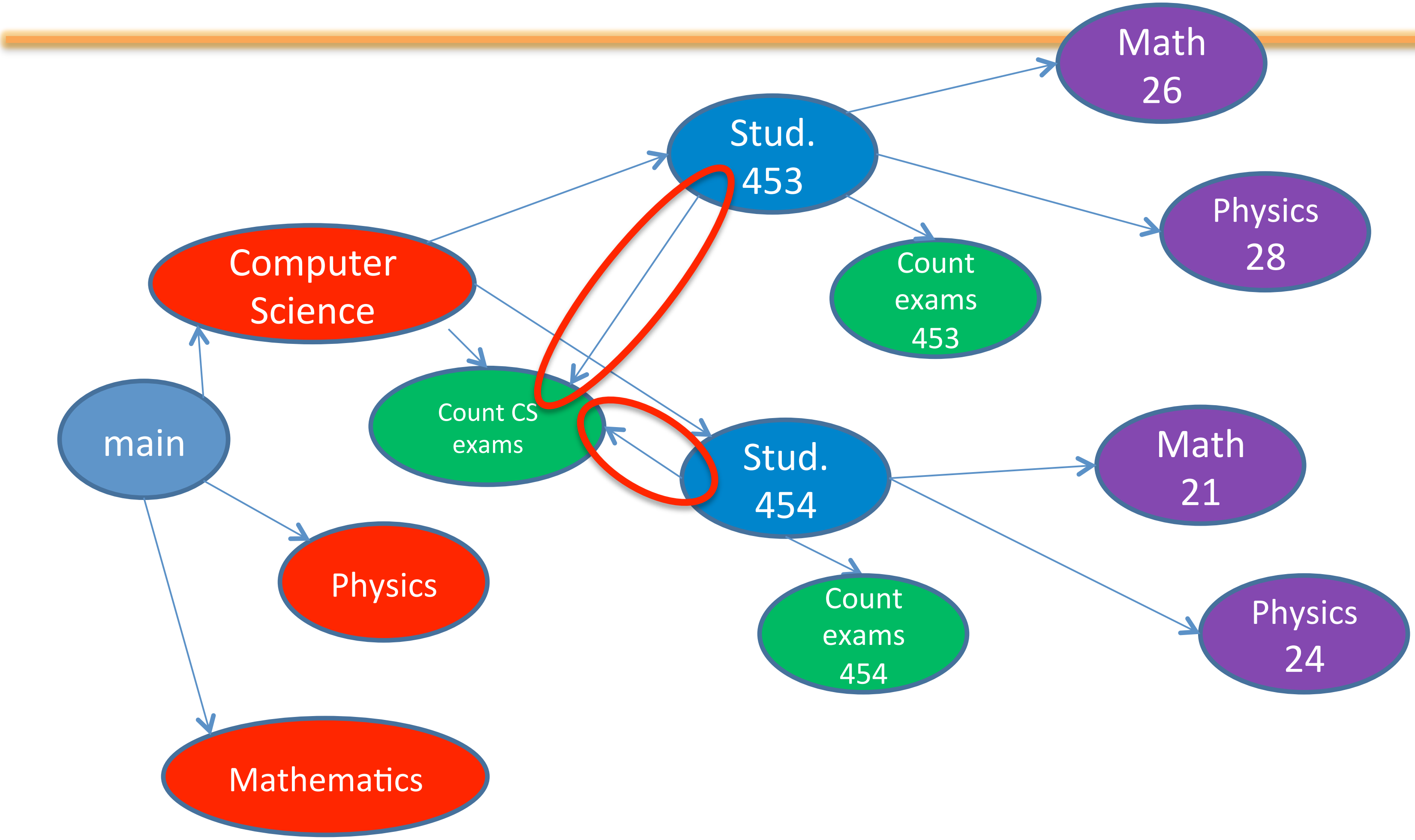
Define attribute and methods

- **Class Course**
  - 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 **Course** for each course
  - One object **Student** for each student. The same object will be attribute of the object Course 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 people developing different classes (on the same project!)
  - 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 mostly occurs in the object handling the data
  - Changes on a specific class do not 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