Introduction to Object-Oriented Programming: OOP

Rui S. Moreira
@ FTC/UFP

Software Development Methodologies

- Software systems are complex to understand, model and implement
- Need for standard processes to build software
  - Use CASE tools (Computer-Aided Software Engineering)
  - Standardize and automate entire software product life-cycle
- Methodologies to systematize and manage complexity
  - Facilitate communication between software teams
  - Reduce sources of errors/mistakes
  - Diminish time of development cycles
  - Ensure the quality of software products
  - Facilitate and easy maintenance

1st Generation Methodologies

- Initially software processes followed a simple sequence of development stages
  - Combining inspiration and basic programming principles
  - Use of software libraries provided reusability
  - Use of high-level programming languages
    - Better abstraction mechanisms
    - Reduction error occurrences
    - Speed development time
- Top-Down Development
  - Naturally divide a problem hierarchically in levels
    - Abstract each level entities/types + relationships/messages/functions
    - Define high-level routines that use simpler low-level sub-routines

Top-Down Methodologies

- Firstly, define/identify a generic high-level solution
- Then, successively refine the solution using 3 techniques:
  - Divide:
    - separate a problem in disjoint sub-problems
    - Tackle each problem separately
  - Progress:
    - Apply same method (divide/progress) to each level
    - Pursue finite solution (indivisible/simple problems to implement)
  - Analyze:
    - Analyze extensively and separately each level
    - Propose solutions for each problem
      - e.g. enumerate all interaction alternatives for the GUI and associated actions

Top-Down Methodologies

- In reality most of times real problems are too complex
  - Top-down division process is not easy and straightforward
  - Not all sub-problems are disjoint and tackled separately by atomic algorithms
- Top-down development is a systematic approach
  - But usually not providing the most efficient solutions

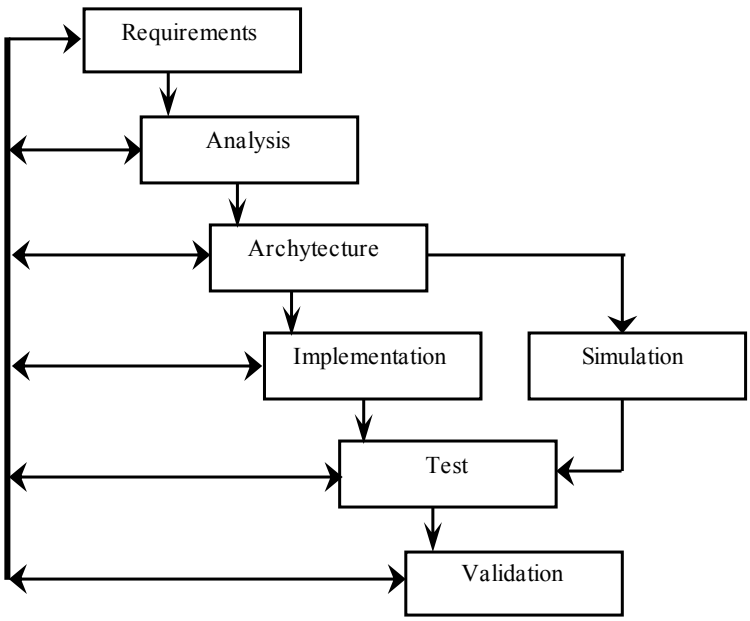
Object-Oriented Methodologies

- Next progresses in software development proposed a simpler Lego-like development strategy
  - Identify entities + behavior + exchanged messages
  - Map real objects and their relationships into programming constructs
  - Use of high-level OO programming languages
    - Classes (abstract generic structure and behavior of given entities) - types
    - Objects (instances of classes - one for each real entity) - instances
- Bottom-Up Development
  - Naturally divide a problem into entities and relationships
    - Abstract each simple entity structure and behavior - type
    - Build more complex entities (types) on top of simpler low-level entities (Lego strategy)

Bottom-Up Methodologies

- Build systems around generic entities
  - called classes = state + behavior
  - Not around functions or procedures (like in C)
- Mirror hardware construction:
  - Identify simple and indivisible entities (attributes + behavior)
    - Simpler to understand, build and test
  - Compose/reuse simpler entities to build more complex entities
    - Simplify development and manage/overcome complexity
    - Suited to evolution and reuse (combine classes)
  - Though may raise difficulties on identifying the right classes and relationships
    - Need for standard OO development methods and tools

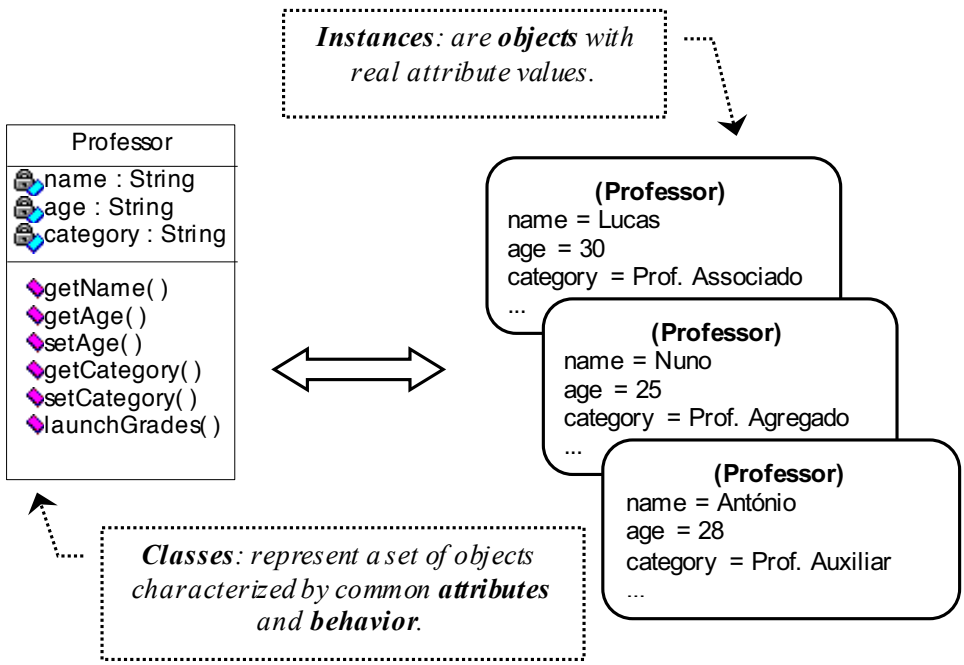
Waterfall OO Development Cycle



OO Programming Languages

- All OO languages map real world entities to software modules usually called **classes**
  - **Classes** are abstract types representing both the state and behavior of real entities
  - **Objects** are instances of classes (one for each real entity that we need to handle)
  - e.g. class Coin (abstracting different Euro coins) may be use to create several instances of 2€, 1€, 0.1€, 0.2€, 0.5€ coins
- Several programming languages provide an Object-Oriented Programming (OOP) approach:
  - Java, C++, Python, PHP, etc.

OOP – Classes versus Objects



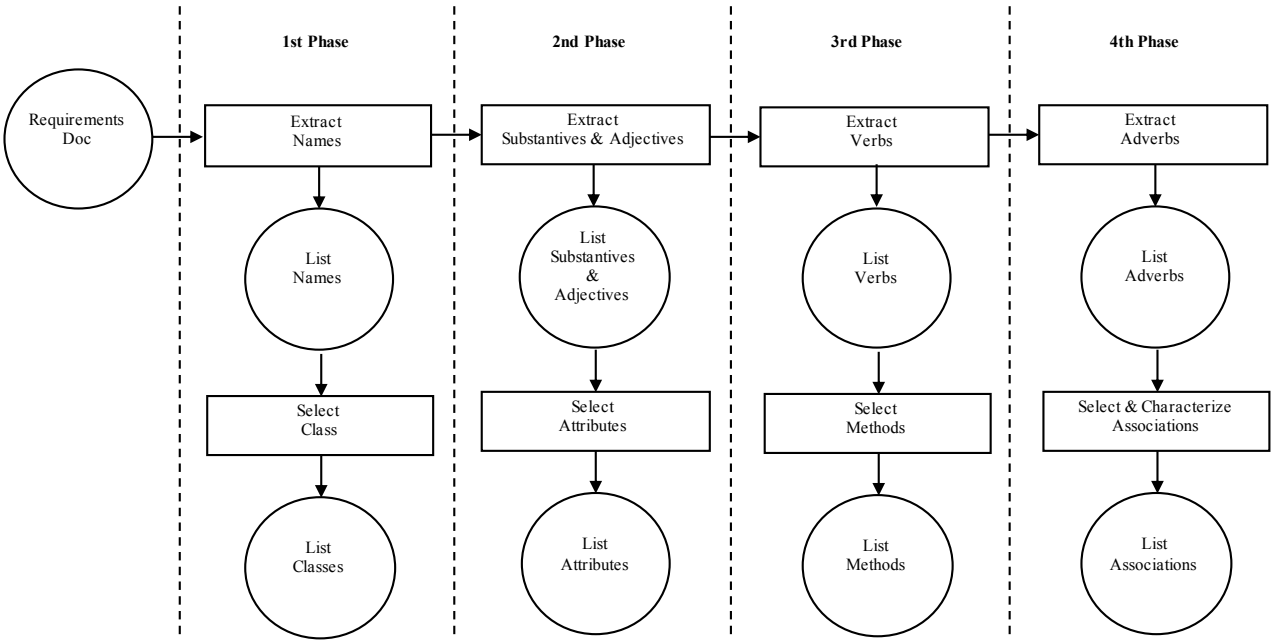
OO Languages Characteristics

- Encapsulation
  - Hide and protect the (inner) state and implementation details
  - Separate the public and private interface
- Relationships
  - Associations between objects (unary and binary)
    - Aggregation (e.g. grapes) & composition (e.g. car components) are stronger relationships
  - Objects communicate through messages
    - method invocations between objects
- Inheritance
  - Special associations between a generic super class and specific sub-classes;
  - sub-classes modify or add specific state + behavior
- Polymorphism
  - Language constructs assuming different shapes (e.g. several methods with same name; same variable referencing different objects, etc.)

Unified Modelling Language (UML)

- The analysis phase focus on finding
  - Entities (i.e. classes with attributes + methods)
  - Associations between classes
  - Interactions (exchanged messages)
- Support for several UML diagrams
  - Use Cases
  - Class diagrams
  - Message sequence charts
  - …

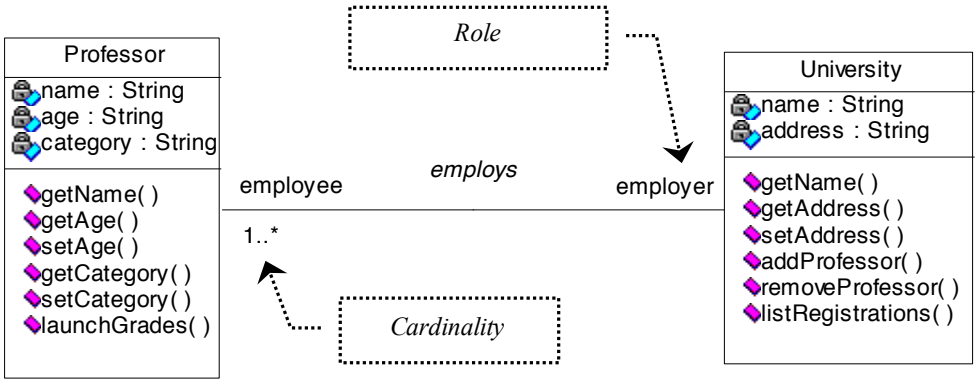
Heuristic to build class diagrams



Class diagrams

- Class names are written in the singular
- Not all names will origin classes
- Separate classes by domains (packages)
  - Interface classes
  - Control classes
  - Data classes

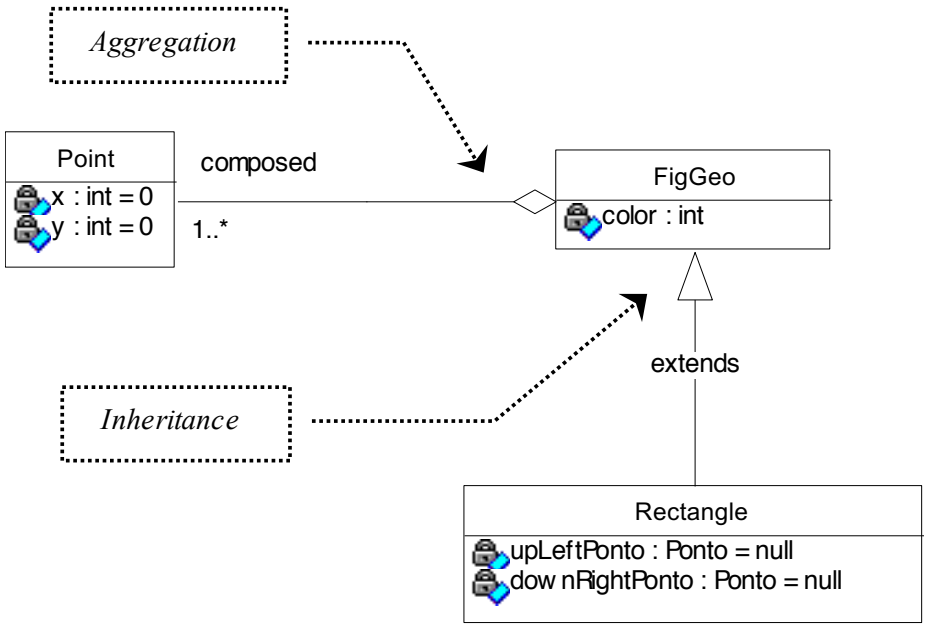
Characterizing associations



Cardinality of binary sssociations

Cardinality	Graphic Representation
1:0	A — 0 — B
1:1	C — 1 — D
1:n	E — 0..* — F
1:1+	G — 1..* — H
1:0 ou 1	I — 0..1 — J
1:2 a 6	K — 2-6 — L
1:2, 4 ou 6	M — 2,4,6 — N

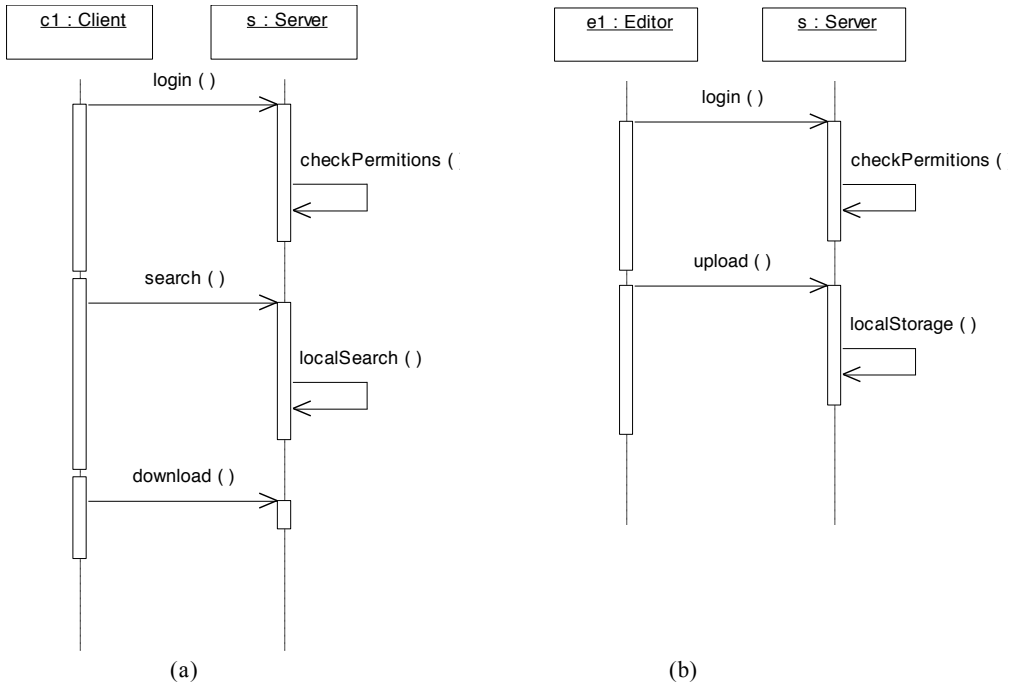
Special associations



Associations

- Inheritance models
  - is-a associations
    (generalizations or specializations)
- Aggregation models
  - part-of associations
  - owns associations
  - whole-part associations

Message exchange scenarios



CASE tools

- Computer-Aided Software Engineering tools
  - Support for UML diagrams
  - Standard graphical modelling tools
  - Generic to several software teams
  - Language independent
  - Analysis and modelling before coding
  - Support code generation
  - e.g. Rational Rose, …, ArgoUML, Gliffy, etc.