Formal Specification of a software system with UML

- In the previous lecture
  - Modelling the Uber App
    - Problems, Ideas, Tasks, Objects, Actions, Attributes
    - Storyboards and Use Cases
    - Hand prototyping
- In this class:
  - Formal specification of a software system
    - Introduction to the Unified Modelling Language, UML

What will we learn?

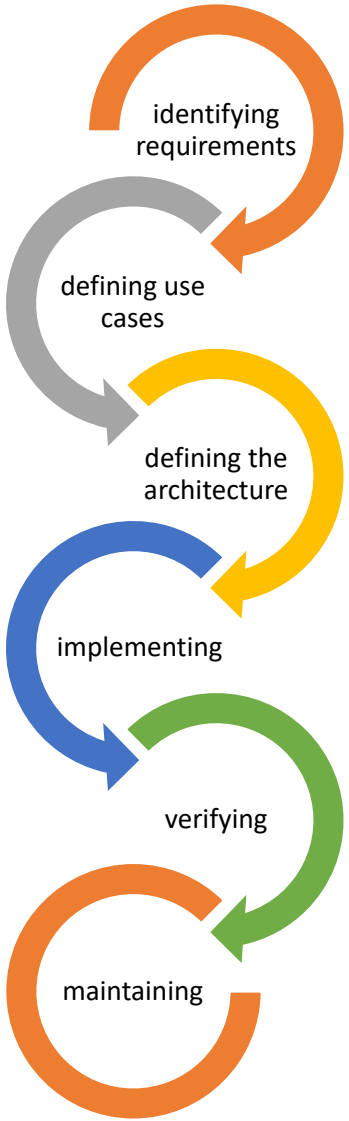
- What is UML and what it is good at
- Methodology associated with the use of UML
- Some of the UML tools
  - symbols
  - diagrams

What is UML ?

- Unified Modelling Language
- object-oriented standardised language
  - offers a set of standardised symbols and diagrams
    - that enables to describe software projects and systems
    - adopting formal, well-established principles
    - In an unambiguous way, that anyone can understand
- It offers a graphical method of communicating/explaining systems, their components, their functionalities and mode of operation

Methodology associated to UML

- When applied to the design and development of a software system
  - "waterfall" model
- Gathering requirements is the first step
  - we have already seen this when addressing the conceptualisation, modelling and design of applications
  - and that it was a step included in the iterative and cyclic approach
  - once a number of iterations has been conducted and a stable model obtained, we may proceed to specifying the system using UML
- Part of the UML tools can also be used during the initial iterative and cyclic process to communicate the idea to potential stakeholders



Requirements

- It is possible to identify functional and operational (non-functional) requirements

Functional

- Directly related with problems that the application aims to solve from the user perspective
- The tasks that the user will be able to accomplish with the system

Operational

- Related with the system performance
- For example, the minimum number of simultaneous users; maximum waiting time when doing a search; computacional resources needed; etc.

Gathering functional requirements

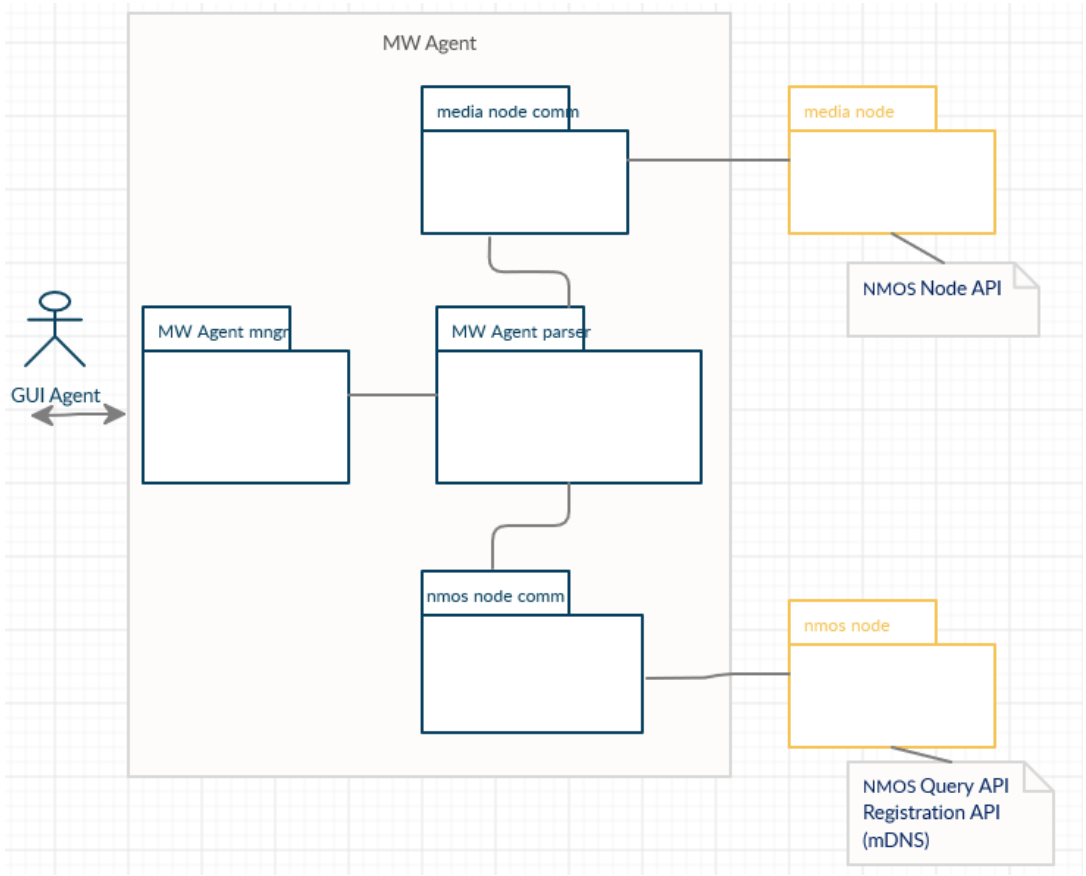
- Let us imagine an app to manage travels
  - *As a user, I want to be able to* register my expenses while traveling
  - *As a user, I want to be able to* create folders to separate my travels and related data
  - *As a user, I want to be able to* visualise information concerning one specific travel
  - *As a user, I want to be able to* select from my travels, those that I will share with my friends or contacts
  - etc.

Formal description with UML

- Once requirements are stable we can initiate the formal description of our system or application
  - Starting with the use cases
- UML offers a set of tools:
  - diagrams and symbols
    - actors, components, objects, classes
    - use case diagrams
    - class diagrams
    - sequence and collaboration diagrams
    - entity relationship diagrams

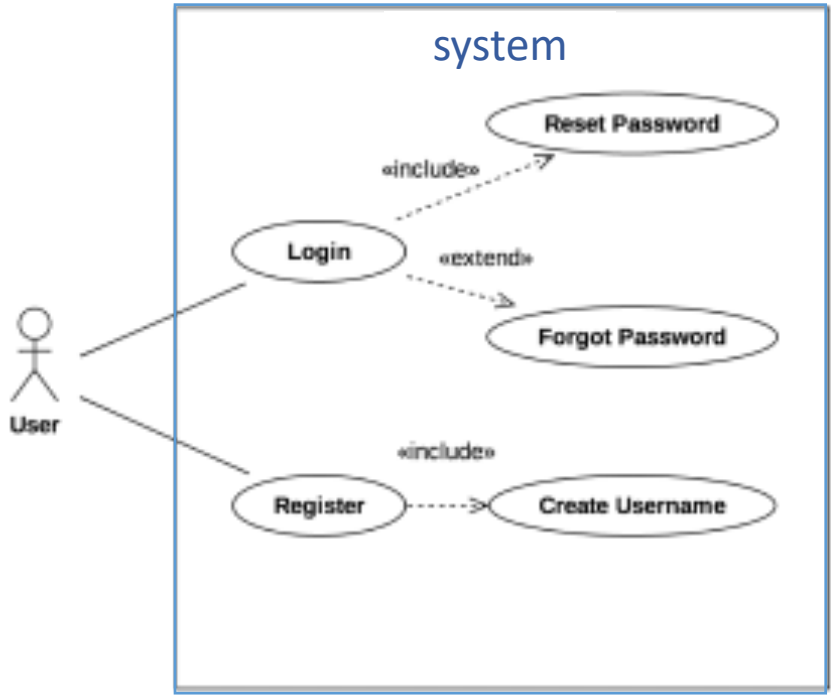
Formal description with UML - tools

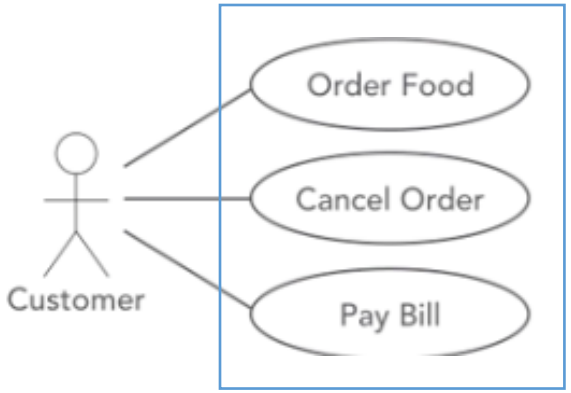
- Component diagram
  - overview of the system/application architecture
    - e.g., the software modules
    - with the identification of the users of the system
      - the actors

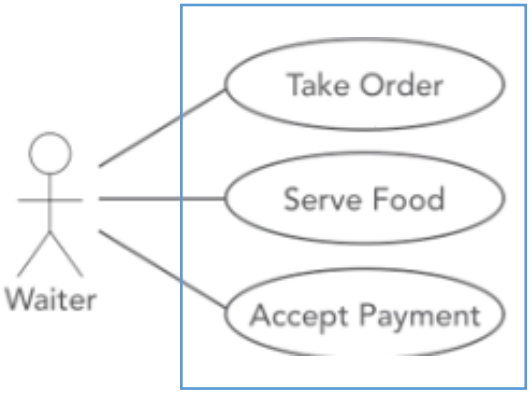


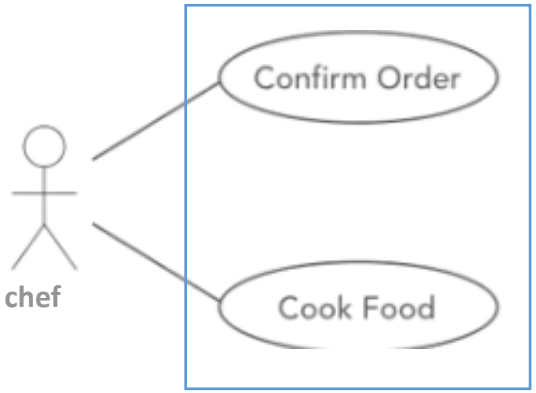
Formal description with UML - tools (2)

- use case diagrams
  - (more detail in the next block of slides)



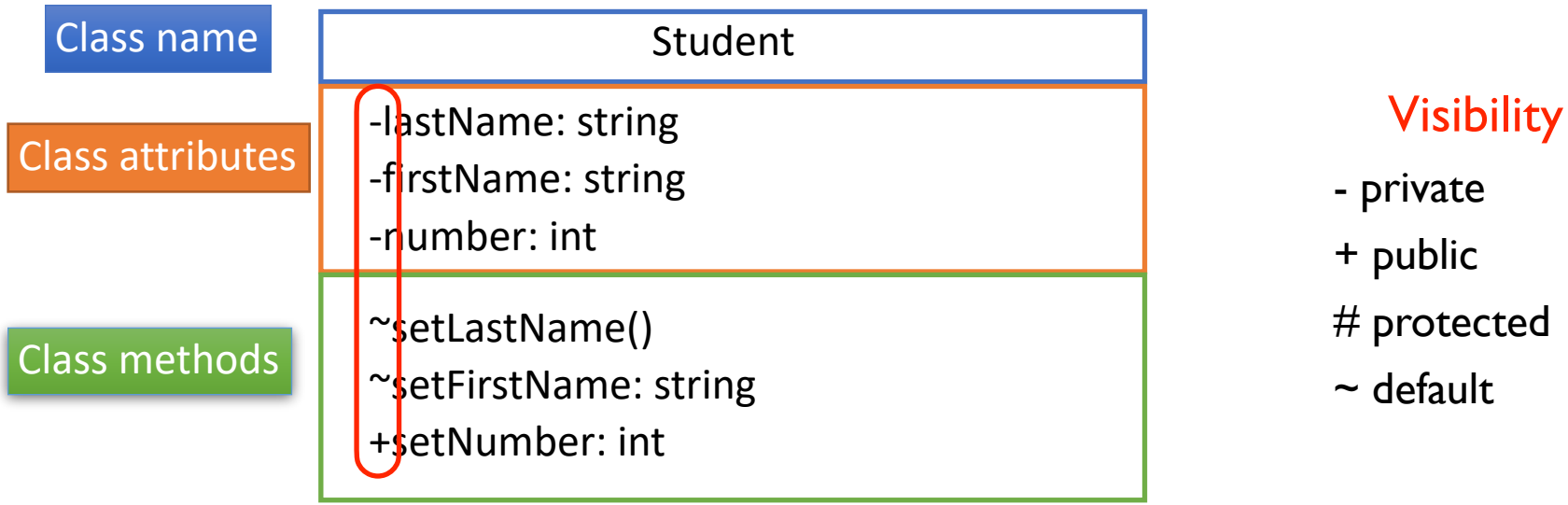






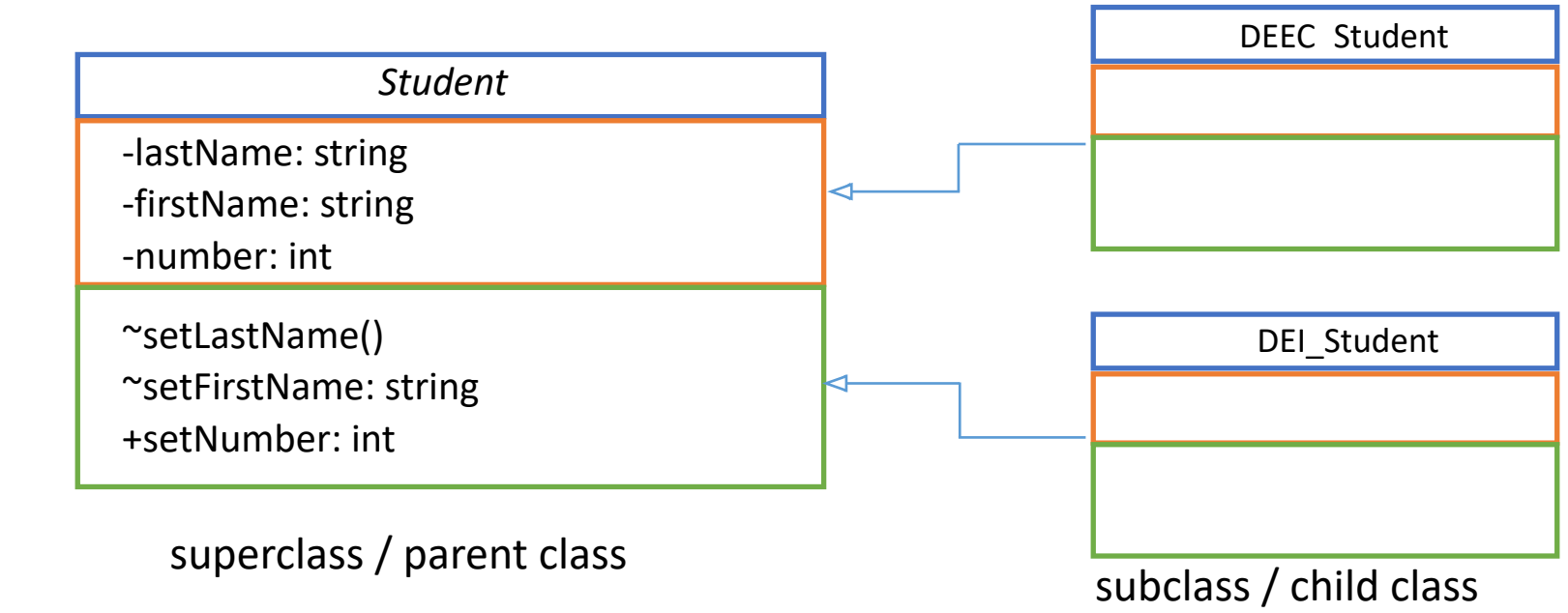
Formal description with UML - tools (3)

- class diagram
  - A class represents a type of object manipulated by the system and enables to define the properties of the objects and what can be done with the object
    - e.g., in a university system, there are students, teachers, administrative staff, departments, class rooms, etc. (class name)
      - students are identified by their names and numbers (class attributes)
      - it should be possible to define and modify names and the number (class methods)



Formal description with UML - relationships between classes

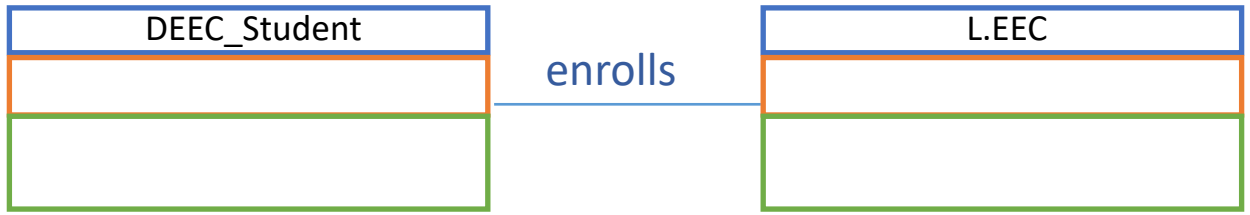
- It is possible to establish relationships between different classes
  - Inheritance (sub-classes that inherit all attributes and methods of the superclass)



- It is possible to add other attributes and methods to the sub-classes
- If a change needs to be made in the common attributes, it is only necessary to do it in the superclass

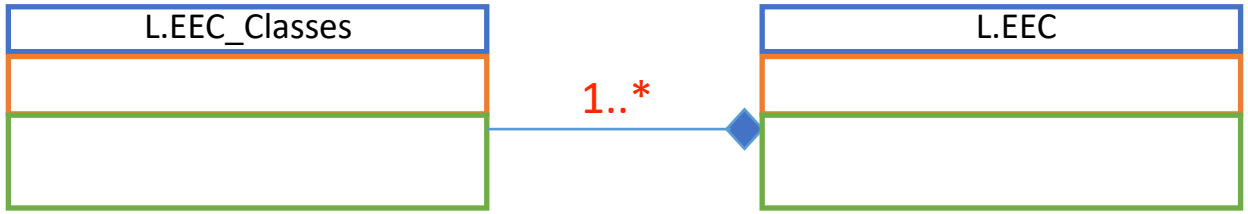
Formal description with UML - relationships between classes (2)

- Association (simple relations between classes)



- Aggregation, a special type of association, where instances of one class can belong to another class but can still exist even if not being part of that other class

- Composition, where a child class cannot exist without its parent class



Multiplicity

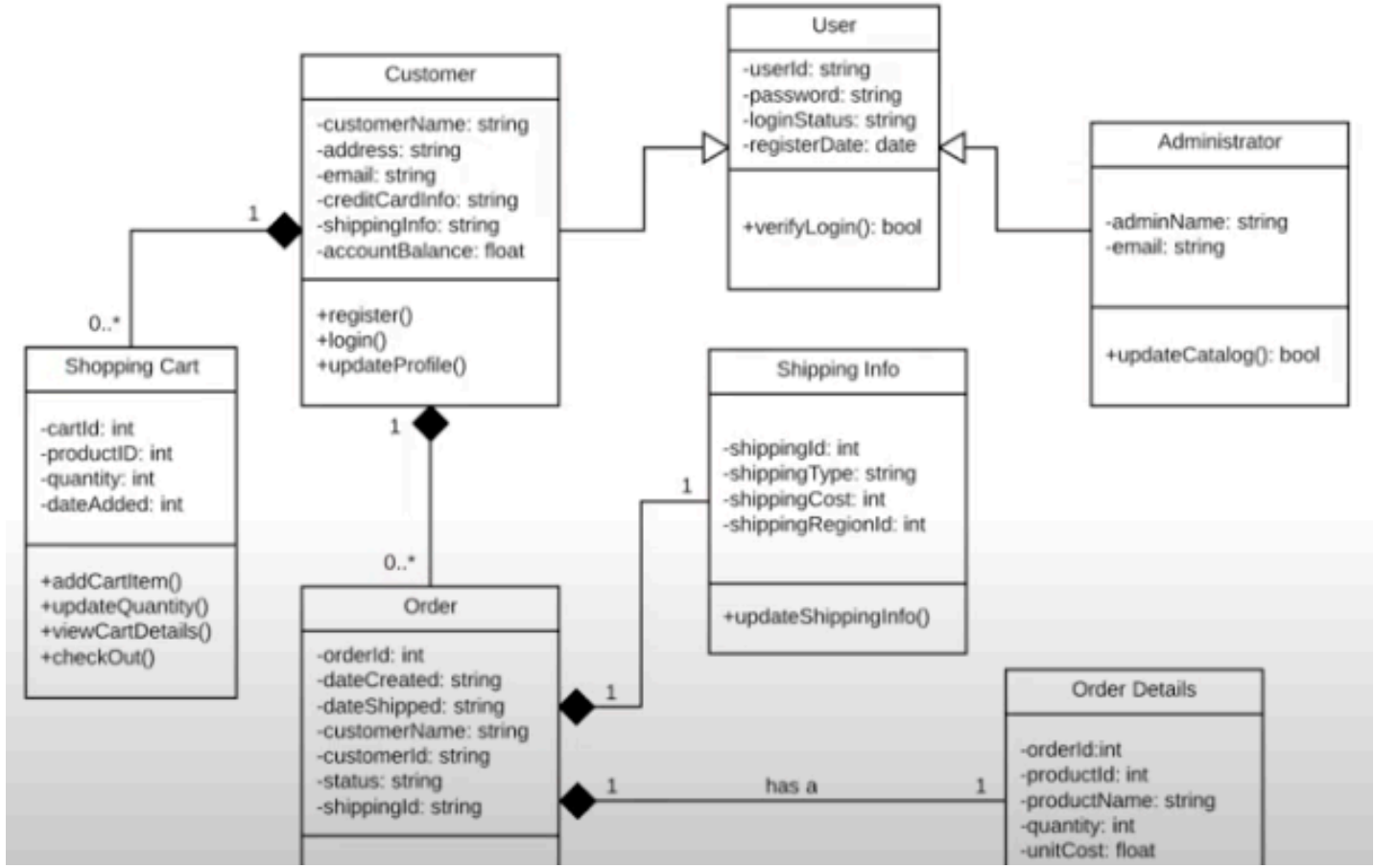
n: only n may exist (0, 1, …)

n..m: at least n may exist up to m

1..*: one to many

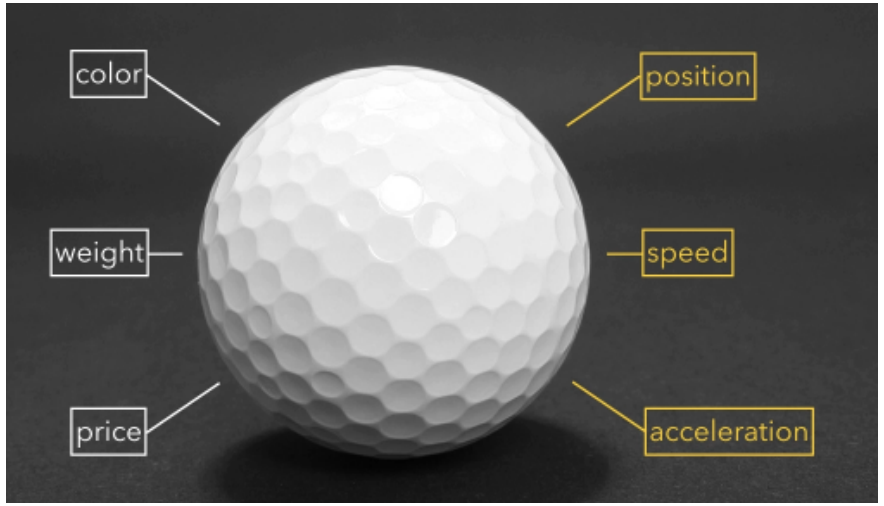
Formal description with UML - tools (4)

- A class diagram is the final step before starting writing code
  - when provided with it, a developer can write code even if he/she has not participate in the previous functional specification phases



Objects and classes

- To better understand what classes are …
  - Structured programming is organised in executable actions
  - Object-oriented programming is organised in objects
    - An object represents a thing just like in the real world
      - With properties that can be fixed/static, granting an identity to the thing
      - Or dynamic, describing instantaneous behaviour
        - A ball has always the same colour, a certain shape and a weight (static properties)
        - Its position and speed may change (behaviour)
      - In OO programming terms
        - Identity and Attributes
        - Behaviour

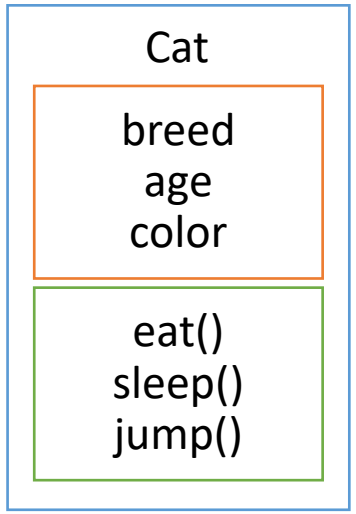


Objects and classes (2)

- In the real world

- in software, the objects are characterised in a similar fashion, using classes

Object	Class
identity / name	name
properties	attributes
behaviour	methods

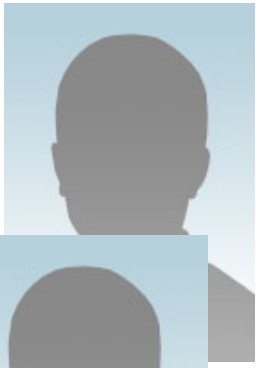


Class

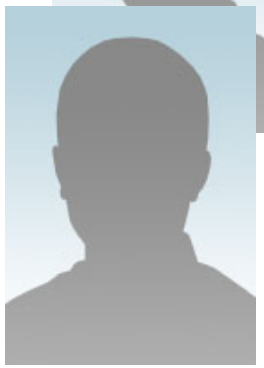
- allows to create abstractions of types of objects with a given set of attributes
  - objects of different types will correspond to different classes
    - ball (type 1, class 1), dog (type 2, class 2), table (type 3, class3), cat (type 4, class 4), etc
  - there may be different objects of the same type but with different values for the attributes
    - white ball with 5 cm radius; yellow ball with 10 cm radius
  - and with different values for the behaviour
    - white ball still vs yellow ball moving
- with one single class "cat" it is possible to create two different objects of the same type (one small black cat and one grey large cat)
- with one single class "student" it is possible to create two different objects each one with different values for the name and number attributes
  - two different instantiations

Class (2)

- In programming terms, a class is a concept that enables to group objects of the same type (with the same attribute and behaviour fields) and distinguish them with different values in (some of) those fields

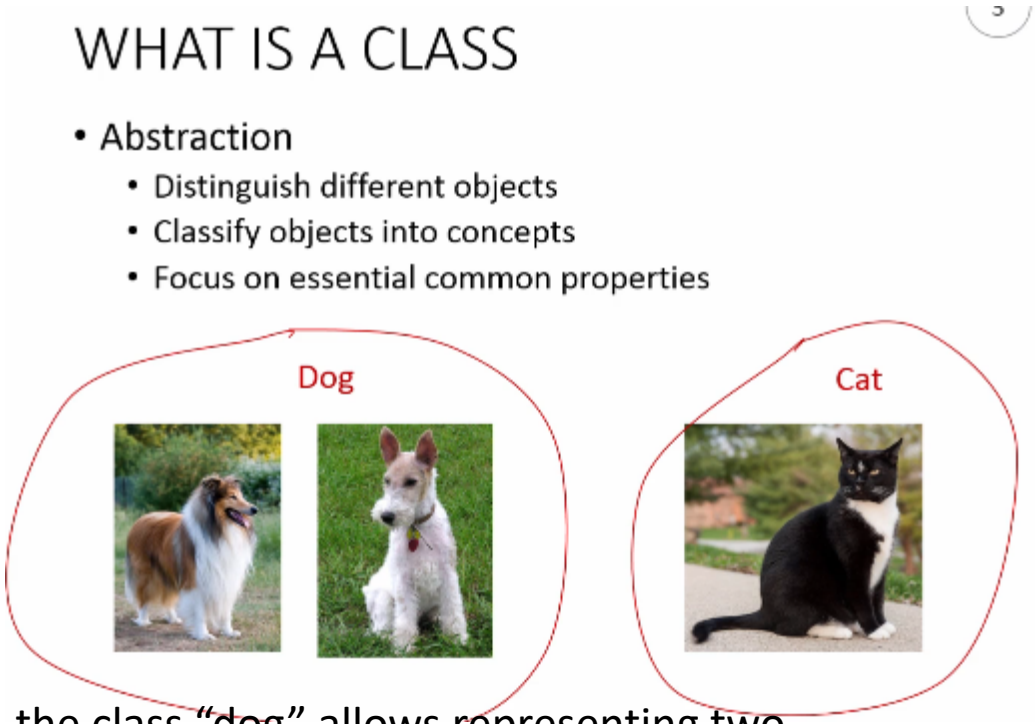


name: Fulaninho de Tal
student number: 201006844
institutional email: up201006844@fe.up.pt



name: Sicrano e Beltrano
student number: 200907954
institutional email up20097954@fe.up.pt

A "student" class enables to represent two objects (instances) for which the attributes name, student number and email have different values



the class "dog" allows representing two objects in which the attribute "breed" have different values

Class vs object

Class

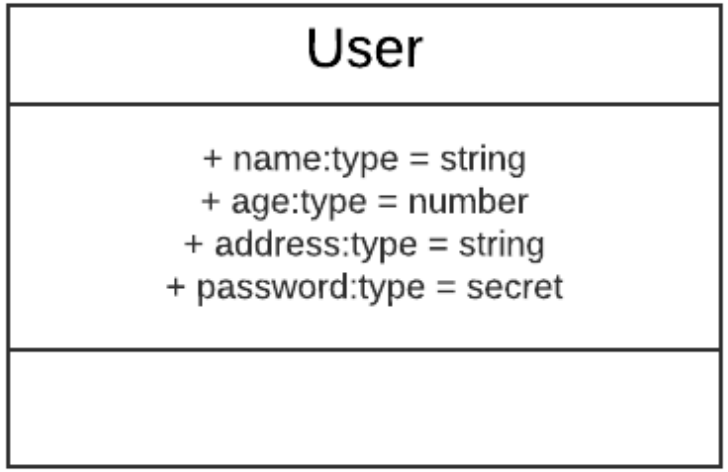
- kind of template
- represents the common structure and behaviour shared by the same type
- A collection of objects of similar type
- must be defined before creating an instance of the class

Object

- a thing, both tangible and intangible
- is comprised of data and operations that manipulate these data
- is called an instance of a class

Activity 4

- Recall the **Objects and Attributes** list when we prototyped the Uber-like app
  - Client
    - name, age, address, password
  - Calendar
    - with single date or time interval
  - Map
    - interactive with address / place of pickup and destination
  - Passengers
    - with associated numeric value
  - Cash register
    - with different associated variants (credit card, Paypal, MBWay,…)
  - Credit card
    - with fields for type, name of the holder, number, expiration date

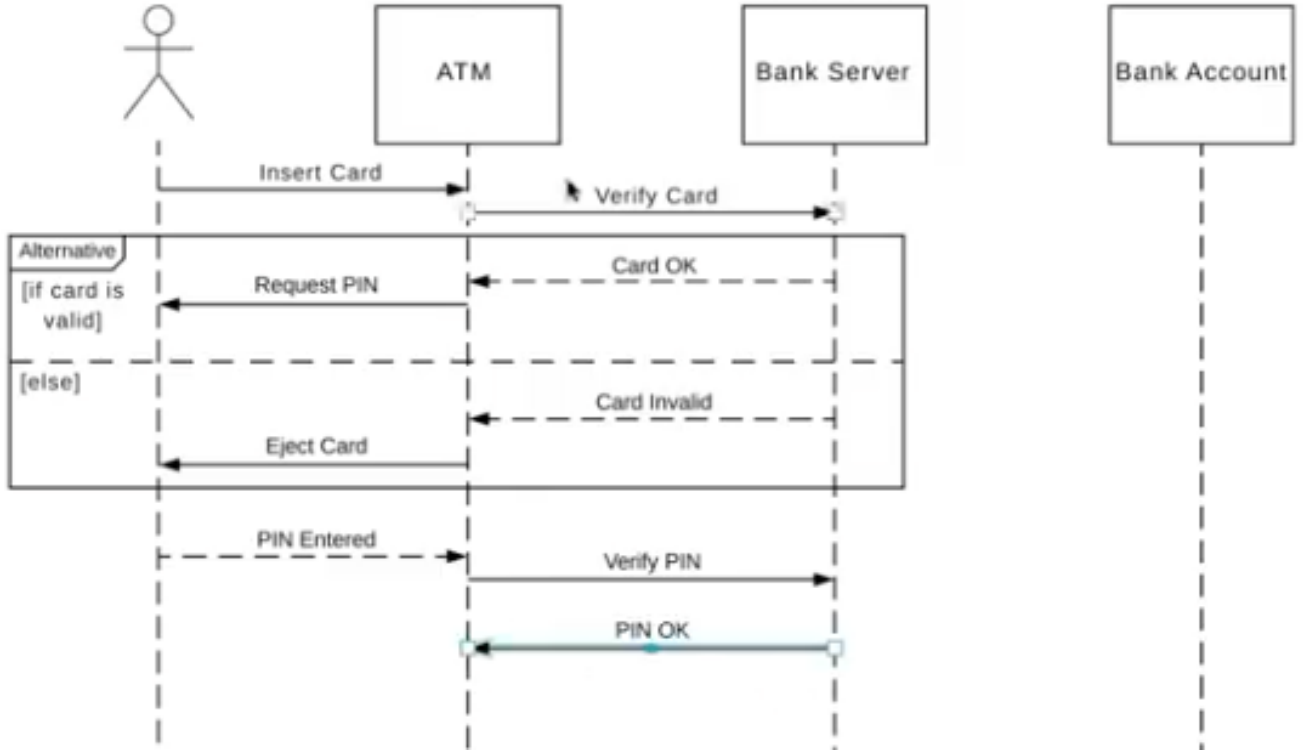


- Chose one or two objects from the list and model the respective classes using a UML Class Diagram

Formal description with UML - tools (5)

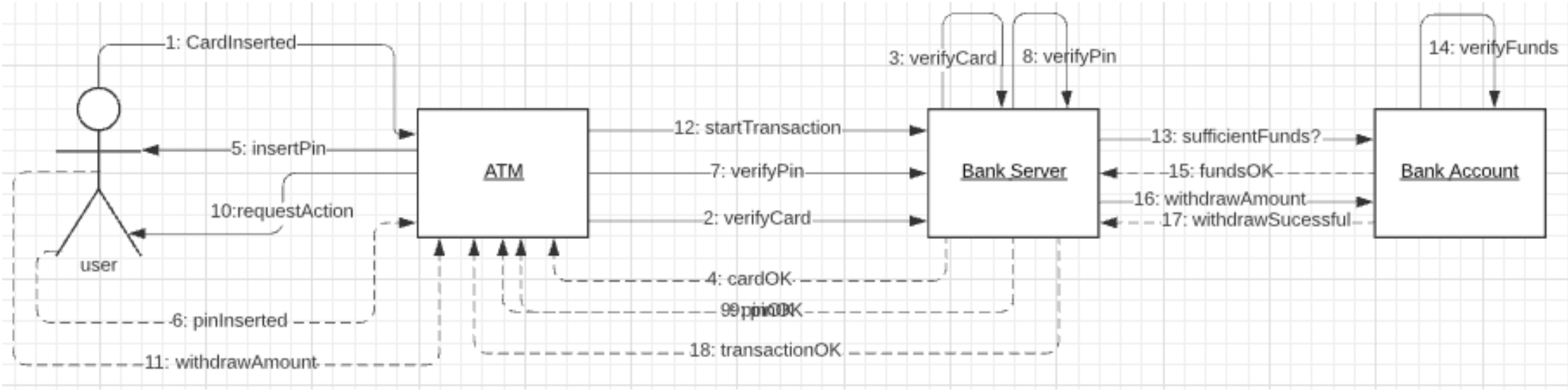
- Sequence and collaboration diagrams
  - they show how objects in a system, or classes in a software program, interact
    - and depict the interactions in the order they take place along a vertical timeline

Sequence diagram

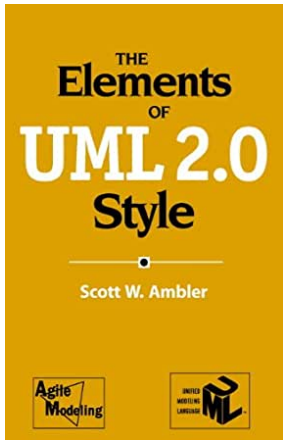


Formal description with UML - tools (6)

- collaboration diagrams
  - Provide the same information as the sequence diagrams but in a different way
    - the sequence is indicated using numbers and not a timeline

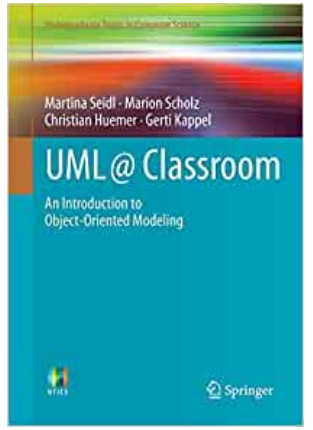


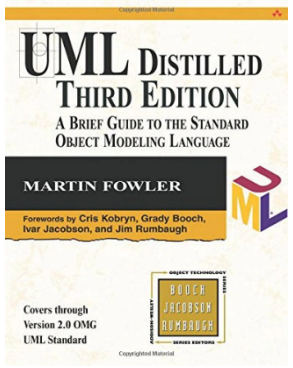
Additional Reading



Ambler, S. W. (2003). *The elements of UML style [agile modeling]*. Cambridge Univ. Press.

Kappel, M. S. M. B. C. H. G., Seidl, M., Scholz, M., Huemer, C., & Kappel, G. (2012). UML @ Classroom. In *Springer International Publishing*. Dpunkt.





Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)* (3rd ed.). Addison-Wesley Professional.