



# Designing Systems

Javier Esparza

Computer Science  
School of Informatics  
The University of Edinburgh



---

# Modelling, again

- Specifying systems is one of the hardest tasks in SE
    - Problem studied by many disciplines: Psychology, sociology, computer science, business administration
  - OO modelling is the current standard approach
    - Roots in programming for [simulation](#)
  - Key issues in OO modelling are
    - which classes do we need, and
    - how do they relate to each other
-

---

# The noun phrase identification approach

Underline the nouns in the informal description of the system and filter out those that should correspond to classes

The Informatics Teaching Office maintains a record of every student taking Informatics courses. This includes basic contact and identification information on the student, the student's Director of Studies, and the student's programme of study. The programme of study is a record of which courses the student is taking that year.

---

---

## Some criteria to retain a noun as a class

An objects of a class,

- is a **thing** you can interact with,  
(it can be sent messages, and it reacts to them)
  - that has a **behaviour**,
  - which depends on its **internal state**,  
(the internal state can change through interaction)
  - and has an **identity** that distinguishes it from other objects.  
(should each element of a list be an object?)
-

---

## Some criteria to reject a noun

- It is **redundant**  
(should there be Scottish and non-Scottish students ?)
  - It is **vague**  
(the **aim** of this **course** is to ...)
  - It denotes an **event or operation**  
(The **student's supervision** is the **responsibility** of the **Director of Studies**)
  - It is **outside the scope** of the system  
(do we have to model the ITO ?)
  - It is an **attribute**  
(**basic contact and identification information**, **year**)
-

- Classes are usually related to
    - Tangible or 'real-world' 'things' (course, person)
    - Rôles (student, director)
  - Associations are usually related to
    - Events: arrival, leaving, request
    - Interactions: meeting
-

---

# Associations

- We choose the classes Student, Director, Programme, Course
  - We now capture **associations**, i.e., relationships between classes
    - a Student **is directed by** a Director  
(but also **complains to, asks for an appointment to, receives a reprimand from, or changes** a Director)
    - a Student **follows** a Programme
    - a Student **takes** a course  
(but also **withdraws from, passes, fails . . .** a Course)
    - a Course **is part of** a Programme  
(maybe it **is a fundamental part** of a Programme, or it **is a complement to** a Programme)
-

---

# Multiplicities

- We may want to record the **multiplicity** of each association.
    - A student is directed by a single director, a single director directs between 30 and 60 students.
    - Each student does follows one programme of studies but there may may be many different programmes of study to follow.
    - Each programme will consist of between 3 and 6 courses to be followed as part of the programme.
  - Models must match the multiplicities (exact, ranges, arbitrary ...)
    - A multiplicity larger than one often involves that the class of the objects has to extend `Collection`
-

---

# Associations from an implementation point of view

In an implementation, classes A and B are associated if objects of class A

- can **send messages** to objects of class B;
  - can **create objects** of class B;
  - have objects (or collections of objects) of class B as **attributes**;
  - can **get messages** with objects of class B as **parameters**.
-

---

# Association classes

- Sometimes it is an association that leads to a class
  - Example: where should the marks of a Student in a Course be kept?
  - It may be adequate to have pairs (student, course) as objects of a class `IsTaking`
-

---

# Validating class models: CRC cards

- CRC: Class, Responsibility, Collaborator
    - **Class**. The name of the class
    - **Responsibilities**. The purpose of the class's existence. More abstract than the operations related to the class
    - **Collaborators**. The classes that help to carry out each responsibility.
-

---

# Use Cases and Interaction diagrams

- A **use case** is a task that an agent needs to perform with the help of the system
  - The realisation of a use case can be visualised by means of **interaction diagrams**
    - One vertical line for each agent
    - Time progresses from top to bottom
    - Messages modelled by horizontal lines starting and ending at a vertical line
-