

Software Requirement

- Requirements: expected **services** of the system and **constraints** that the system must obey
- Functional Requirements
 - What the system must achieve
- Non-functional Requirements
 - Software quality: How well the system can do its job, etc
- Domain Requirements
 - Easy to omit as domain experts may think they are “obvious”

Functional Requirements

- Functions, tasks, or behaviors the system must fully support.
 - How user of the system use the system
- The “skeleton” of the system requirements
 - Should be captured in early iterations
- Need to distinguish “core functions” from “features”

Non-Functional Requirements

- Constraints placed on various attributes of system functions or tasks
- Equally important compared to functional requirements
 - Separate software products from software practices
- Sources
 - Domain: i.e. Human can tolerate up to 150ms delay in voice communication
 - Legacy: i.e. QWERTY keyboard
 - User: i.e. User want to operate the interface with one hand
 - Regulation: The system should switch to backup and resume within 1ms after the primary program crashes

Examples of Non-Functional Requirements

- User interface and human factors:
 - What type of user will be using the system?
 - Will more than one type of user be using the system?
 - What sort of training will be required for each type of user?
 - Is it particularly important that the system be easy to learn?
 - Is it particularly important that users be protected from making errors?
 - What sort of input/output devices for the human interface are available, and what are their characteristics?

Examples of Non-Functional Requirements

- Performance characteristics
 - Are there any speed, throughput, or response time constraints on the system?
 - Are there size or capacity constraints on the data to be processed by the system?
- Error handling and extreme conditions
 - How should the system respond to input errors?
 - How should the system respond to extreme conditions?

Examples of Non-Functional Requirements

- Quality issues
 - What are the requirements for reliability?
 - Must the system trap faults?
 - What is the maximum time for restarting the system after a failure?
 - Is it important that the system be portable (able to move to different hardware or operating system environments)?
- System Modifications
 - What parts of the system are likely candidates for later modification?
 - What sorts of modifications are expected (levels of adaptation)?
 - Might unwary adaptations lead to unsafe system states?

Identifying Non-functional Requirements

- Certain constraints are related to the design solution that are unknown at the requirements stage.
- Certain constraints are highly subjective and can only be determined through complex, empirical evaluations.
- Non-functional requirements tend to conflict and contradict.
- There is no ‘universal’ set of rules and guidelines for determining when nonfunctional requirements are optimally met.

Requirement Elicitation

- Step 1: **(Business analyst)** develops common understanding of the problem domain with (customers) and (domain experts)
- Step 2: **(Business analyst)** explains the problem to (the development team) and develop a design strategy
- Step 3: **(Business analyst)** presents the design strategy to the customer, and agree on technical solutions

Business analysts

- Need to be familiar with the **problem domain** and **development techniques**
- The bridge between the customers and the development team
 - To the customers:
 - Explain in domain language what can/cannot be achieved with existing constraints
 - Hide technical details when explaining the technical solution to the customers
 - Create user manual
 - To the development team:
 - Reformulate the domain problem as mathematical problems

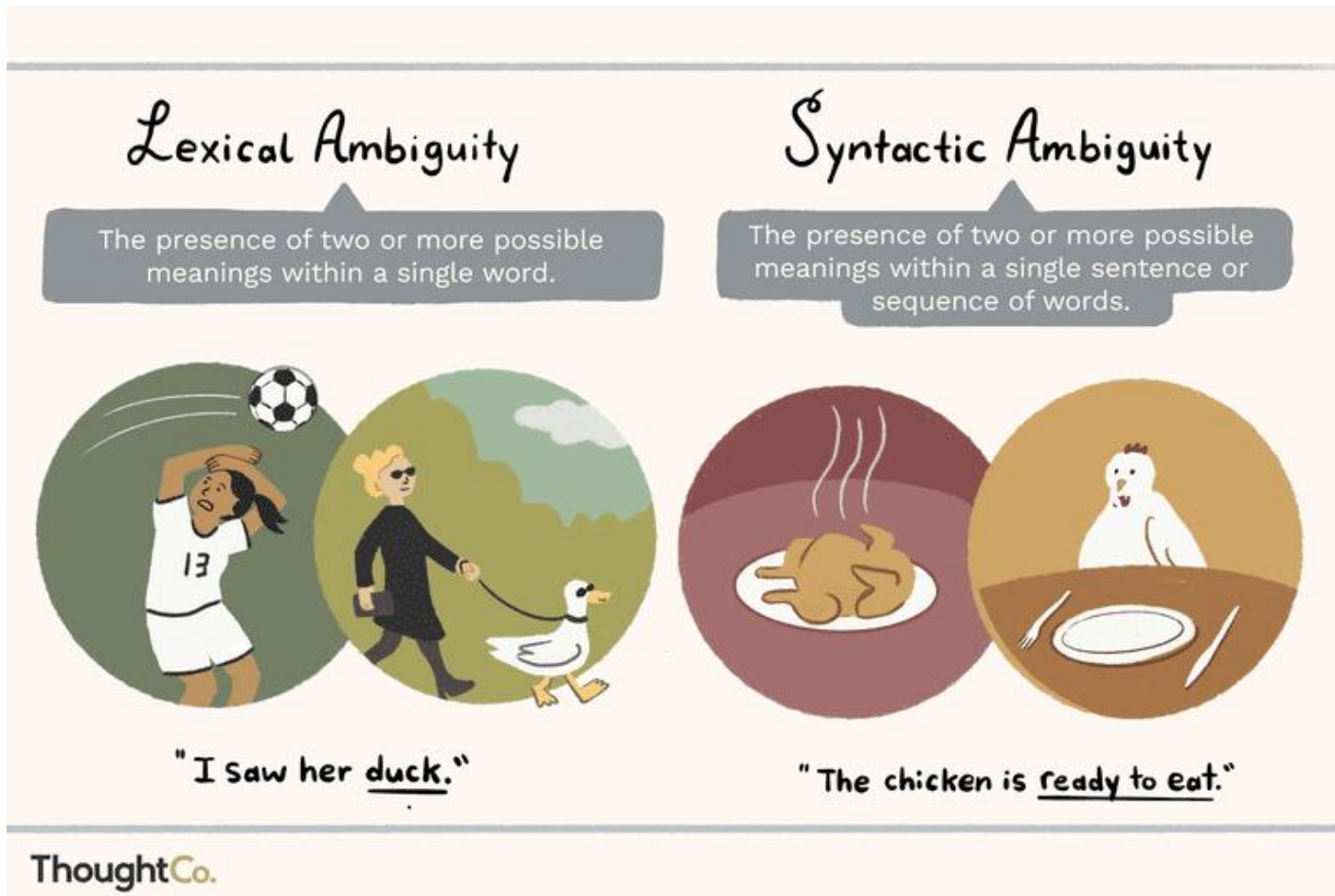
Common Problems During Requirement Elicitation

- Problem of scope
 - What environmental condition the system will operate in?
- Problem of understanding
- Problem of volatility
 - User needs evolve over time

Problem of Understanding

- The customer fails to explain their needs well.
 - Need a common language
- The analyst may not understand the customer's need.
 - Need to study the problem domain
- The customer may not know what he/she wants
 - The team should identify customer needs from the problem domain
- The analyst may not clearly convey the requirements to the development team
 - Problem abstraction

Natural Languages Are Prone to Ambiguities



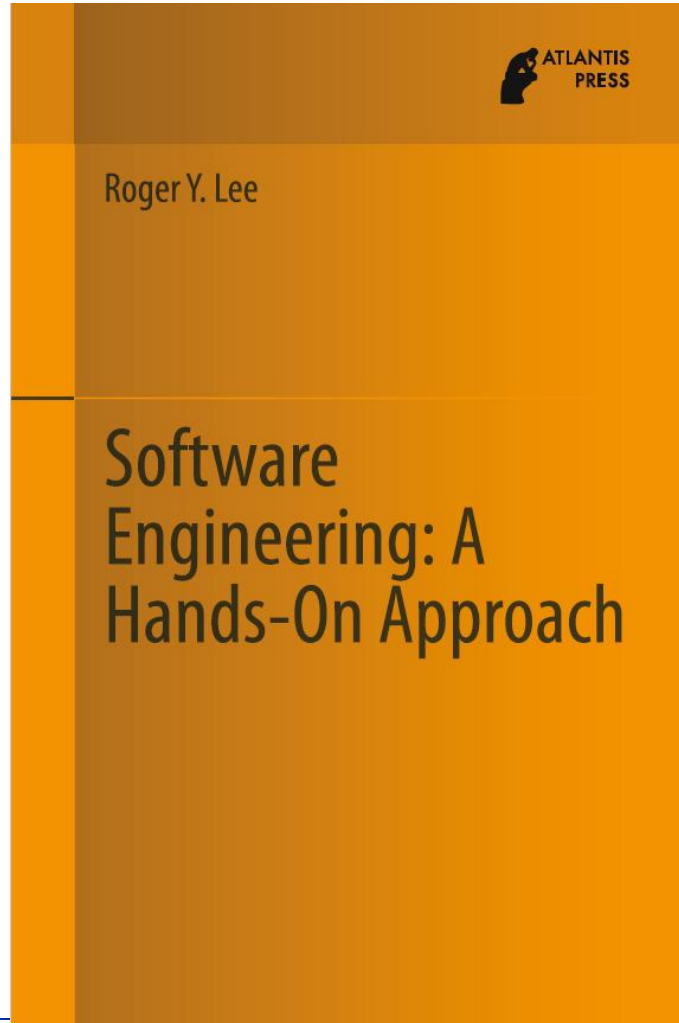
We need a widely used formal language

Communications among various stakeholders

- Need a common language for communication
- Unified Modeling Language (UML)
- Recognized as an international standard
- It's just a tool, not a solution

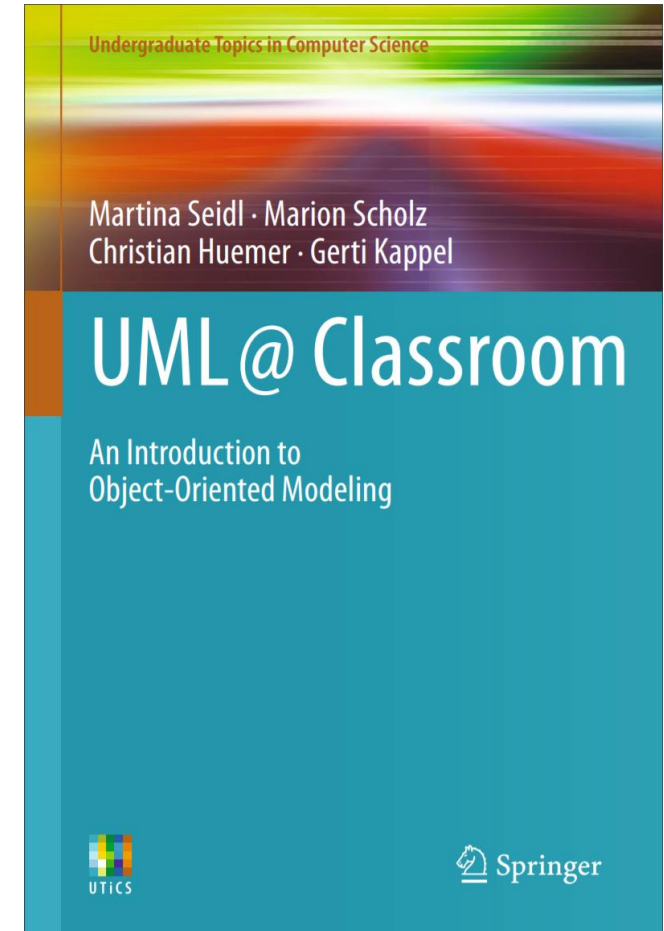


Reference Book



Reference for UML

- Freely available online
- Search from our library website



Object-Oriented Design



Procedure-Oriented Software Design

- Describe problems in terms of functions: $y=f(x)$
- Behaviors hard to describe as procedure



Procedure-Oriented Software Design

- Sensitive to requirement changes
- Nothing reusable
- Less intuitive (Communication problems)
- No information hiding

```
graduate()
{
    returnCafe();
    dropClass();
    returnBook();
}
```



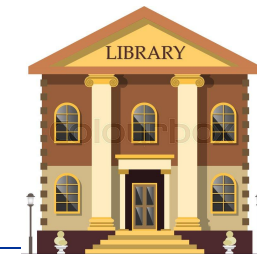
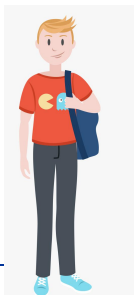
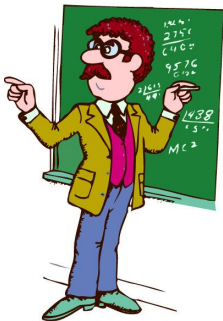
deposit()

deposit()

Joe	202001	\$100	Yes
Jane	202002	\$200	No

Joe	202001	CS132
Jane	202002	CS233

Joe	202001	Book 1
Jane	202002	Book 2



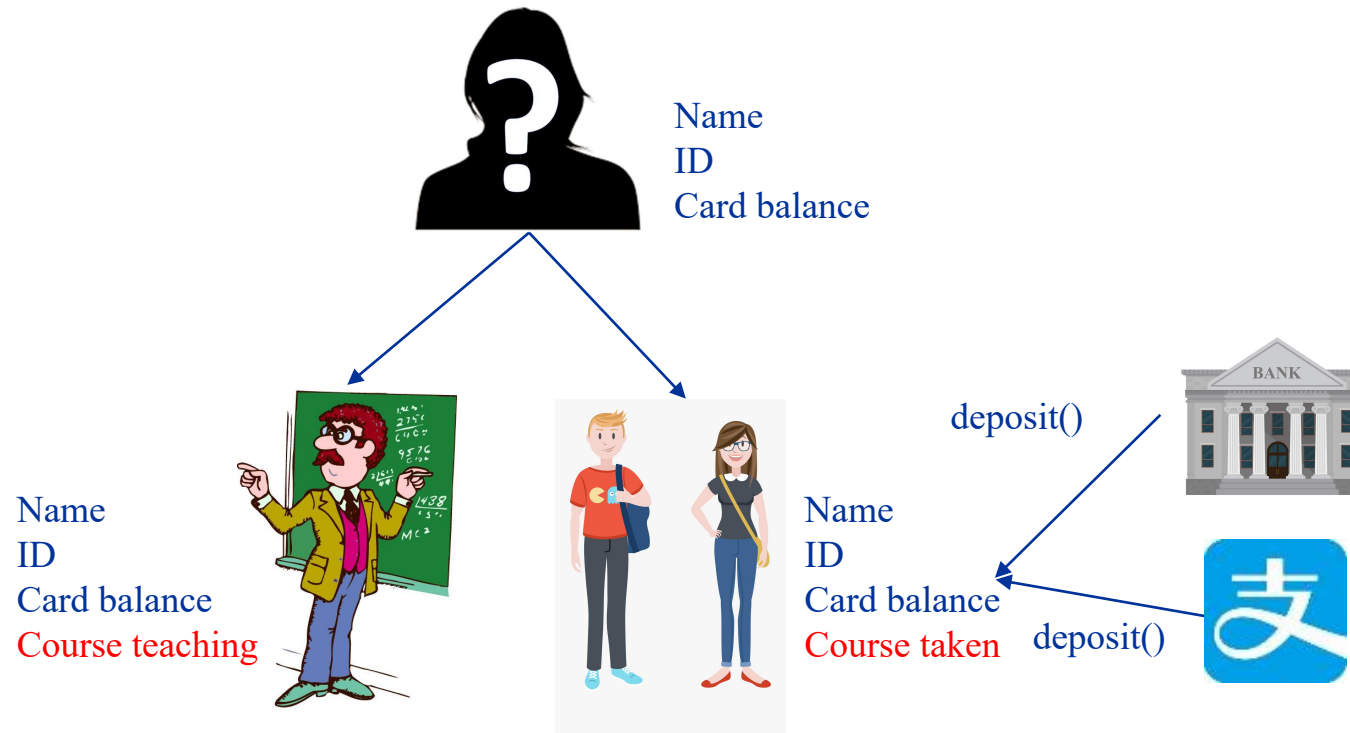
Engineering

DIY Community in Electrical Engineering

- Standardized “building blocks”
 - Easily accessible
- Standardized interface
 - Interchangeable components
- Can we define a software system as a collection of objects of various types that interact with each other through well-defined interfaces?

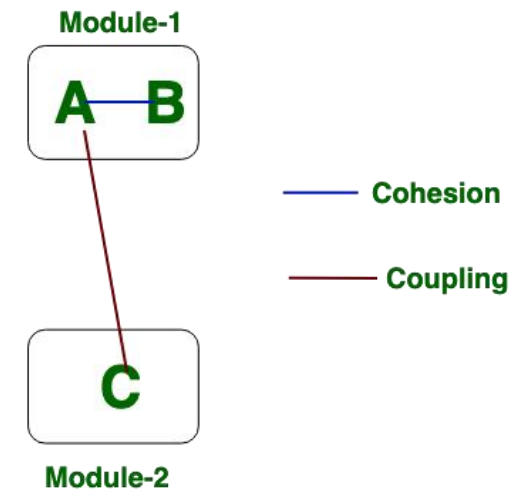
Object-Oriented Software Design

- Describe problems as objects and interactions between objects
- Much more intuitive



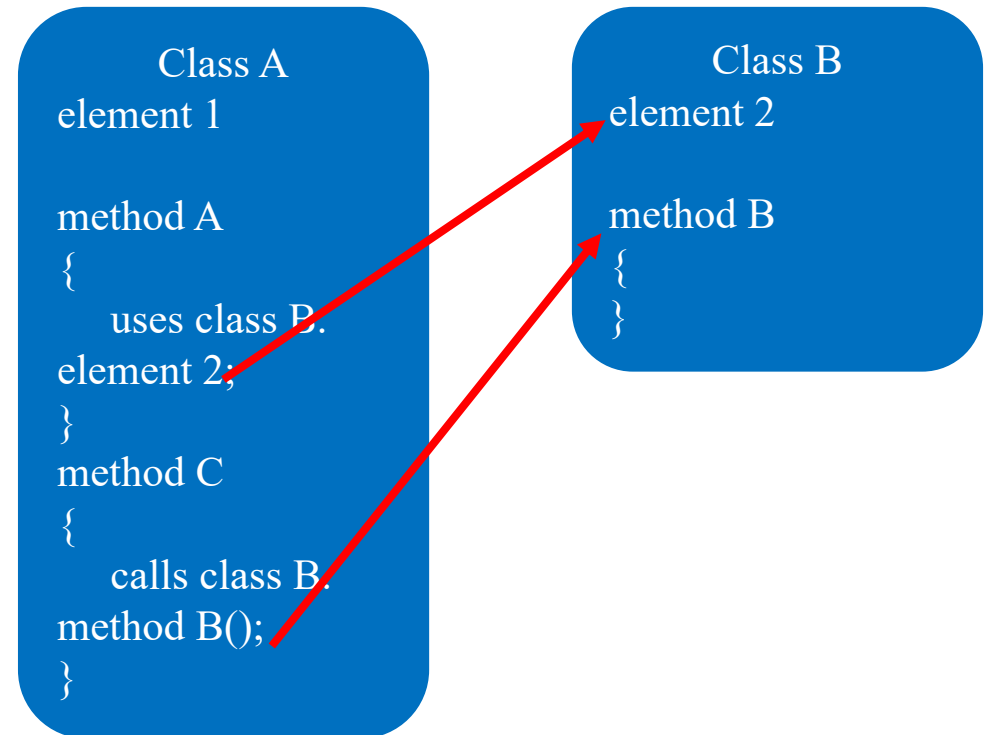
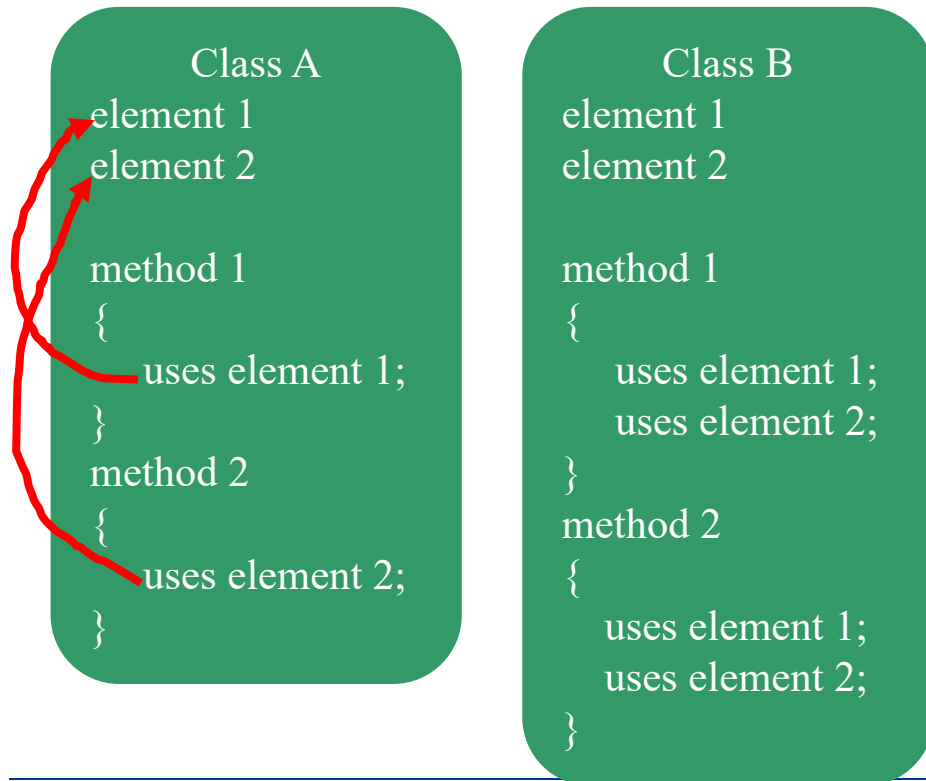
Benefits of OO

- Modularity: Decompose a system into a set of **cohesive** and **loosely coupled** modules
 - Reusability
 - Accidental vs. deliberate reuse
 - Encapsulation and information hiding
 - Interfaces
 - Access levels
 - Reduce coupling
- Inheritance: a relationship between different classes in which one class shares attributes of one or more different classes



Cohesion vs. Coupling

- Low vs. high cohesion
- Tight Coupling (**avoid**)



Design Choices

- A method of an object may only call methods of:
 - The object itself.
 - An argument of the method.
 - Any object created within the method.
 - Any **direct** properties/fields of the object.
- **Don't talk to strangers!**
- When one wants a dog to walk, one does not command the dog's legs to walk directly; instead one commands the dog which then commands its own legs.

The Unified Modeling Language (UML)



History of UML

- Resulted from the convergence of notations from three leading object-oriented methods (The Three Amigos)
 - OMT (James Rumbaugh)
 - OOSE (Ivar Jacobson)
 - Booch (Grady Booch)
- 1995 Unified Method 0.8
- 1997 Unified Modeling Language 1.0
 - Object Management Group (OMG)
- Currently at UML 2.0
- Has a very good eco-system and is still evolving

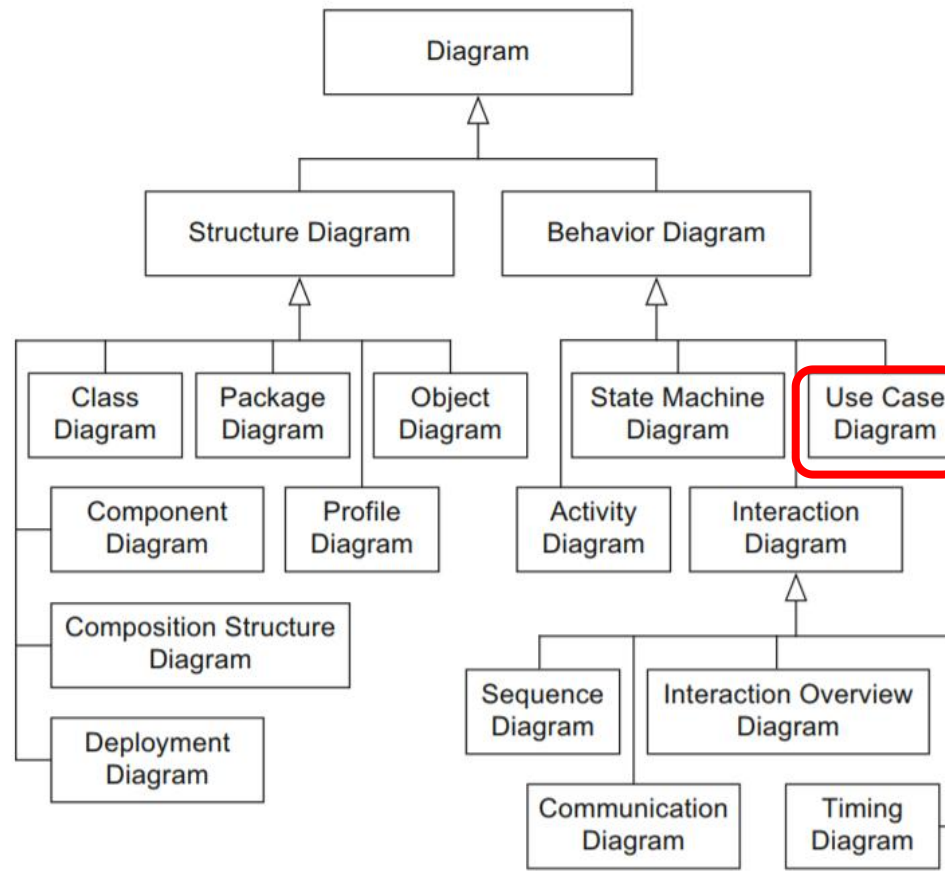
UML Partners

- IBM
 - NEC
 - Oracle
 - ...
-
- Very impressive list
 - Specialized to solve problems in different domains

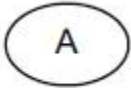
Unified Modeling Language (UML)

- Visualizing and documenting **analysis** and **design** effort.
- Unified because it ...
 - Combines main preceding OO methods (Booch by Grady Booch, OMT by Jim Rumbaugh and OOSE by Ivar Jacobson)
- Modelling because it is ...
 - Primarily used for visually modelling systems. Many system views are supported by appropriate models
- Language because ...
 - It offers a syntax through which to express modelled knowledge

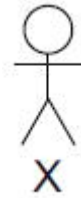
UML Diagrams



Use Cases

- The customer's requirements of the system (**Functional**)
- Who's interacting with the system?
- How they are using it?
- Use case 
 - Verb + **Noun**
 - i.e. Check deposit
- Use case needs to benefit the actor
 - i.e. Withdraw money is a use case, fill in the withdraw form is not

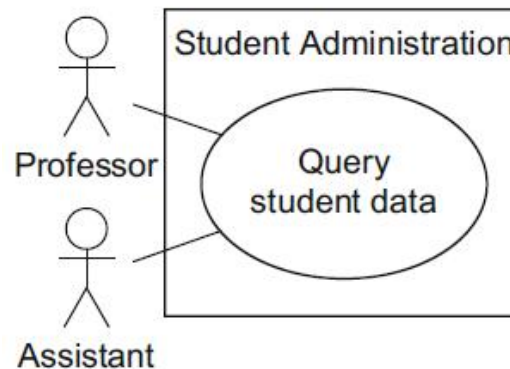
Actor



- Actor
 - Don't confuse with stakeholder
 - Not all stakeholders are actors in certain use cases
 - Not necessarily human. i.e. server
- Active actor: who initiates the use case
- Primary/secondary actor:
 - Who benefits from executing the use case
- Represents roles, not user
 - One user can perform different role, therefore represents multiple actors

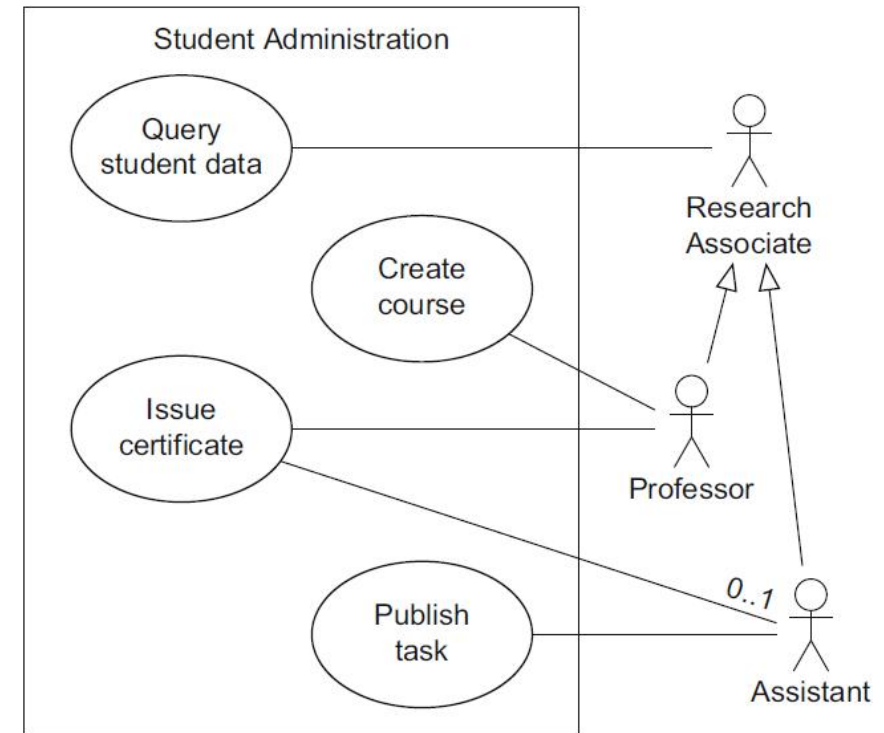
Associations

- Associate actors with use cases
- Every actor should interact with at least one use case
- Every use case should have at least one associated actor
- Actors are outside of system boundary
- Some stakeholders are inside the system as business workers



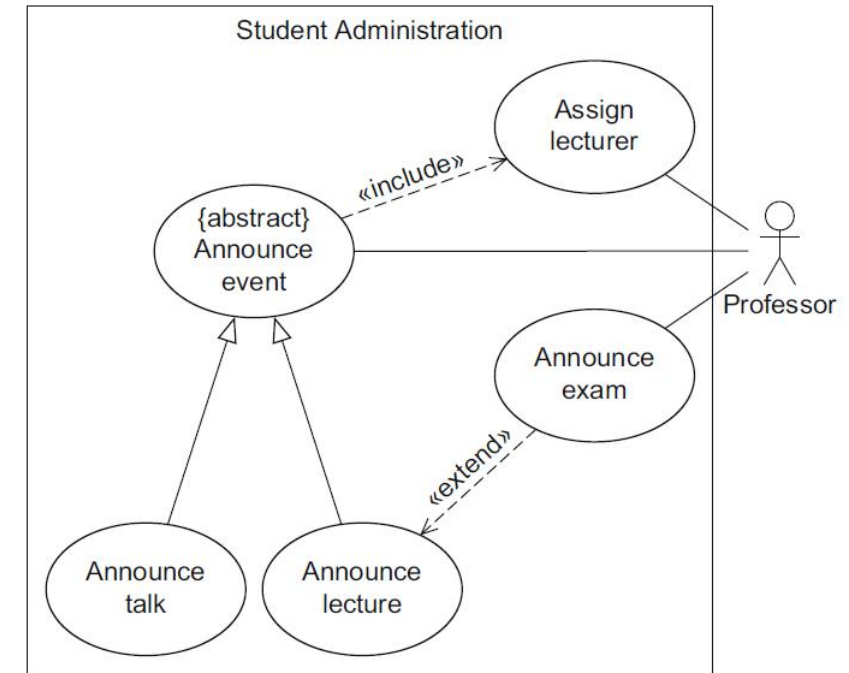
Relationships between Actors

- Generalization/inheritance
- Abstract actor
 - No instance



Relationships between Use Cases

- Include
 - The included use case is part of base use case
 - Like a subroutine
 - The included use case can be executed individually
- Extend
 - Optional
 - The extending use case and the base use case both can execute individually
 - The extending use case is triggered by a **condition** at an **extension point**
- Generalization



Use Case Document Example

Name:	Reserve lecture hall
Short description:	An employee reserves a lecture hall at the university for an event.
Precondition:	The employee is authorized to reserve lecture halls. Employee is logged in to the system.
Postcondition:	A lecture hall is reserved.
Error situations:	There is no free lecture hall.
System state in the event of an error:	The employee has not reserved a lecture hall.
Actors:	Employee
Trigger:	Employee requires a lecture hall.
Standard process:	(1) Employee selects the lecture hall. (2) Employee selects the date. (3) System confirms that the lecture hall is free. (4) Employee confirms the reservation.
Alternative processes:	(3') Lecture hall is not free. (4') System proposes an alternative lecture hall. (5') Employee selects the alternative lecture hall and confirms the reservation.

Identify the actors

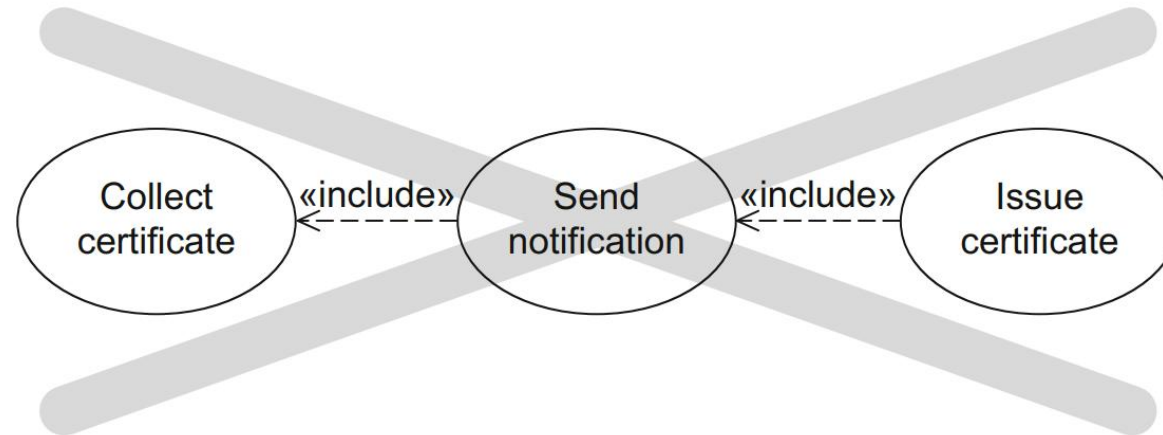
- Who uses the main use cases?
- Who needs support for their daily work?
- Who is responsible for system administration?
- What are the external devices/(software) systems with which the system must communicate?
- Who has an interest in the results of the system?

Derive the use cases

- What are the main tasks that an actor must perform?
- Does an actor want to query or even modify information contained use cases in the system?
- Does an actor want to inform the system about changes in other systems?
- Should an actor be informed about unexpected events within the system?

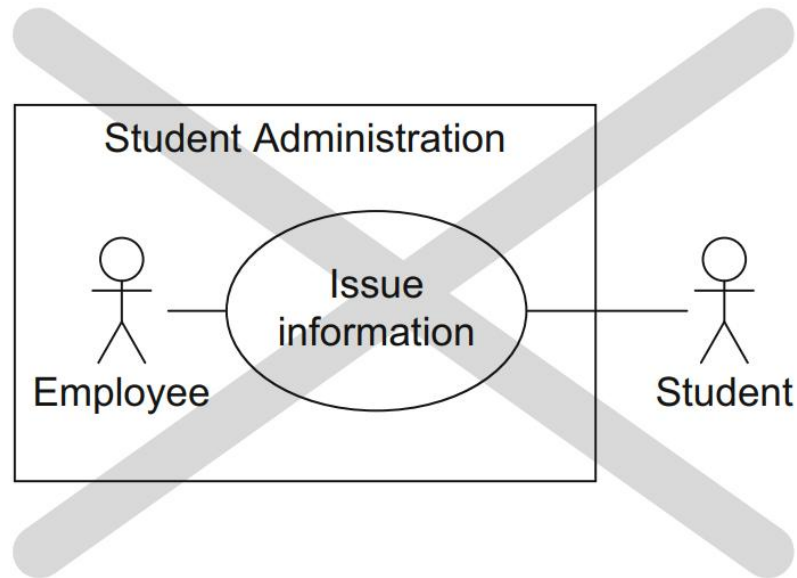
Common Problems

1. Modeling Processes



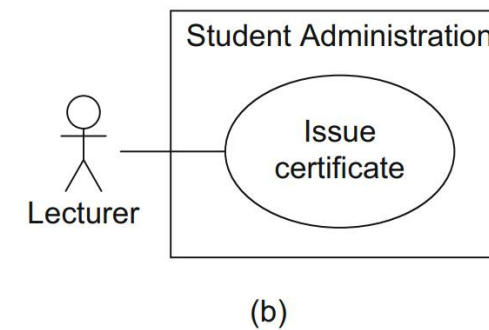
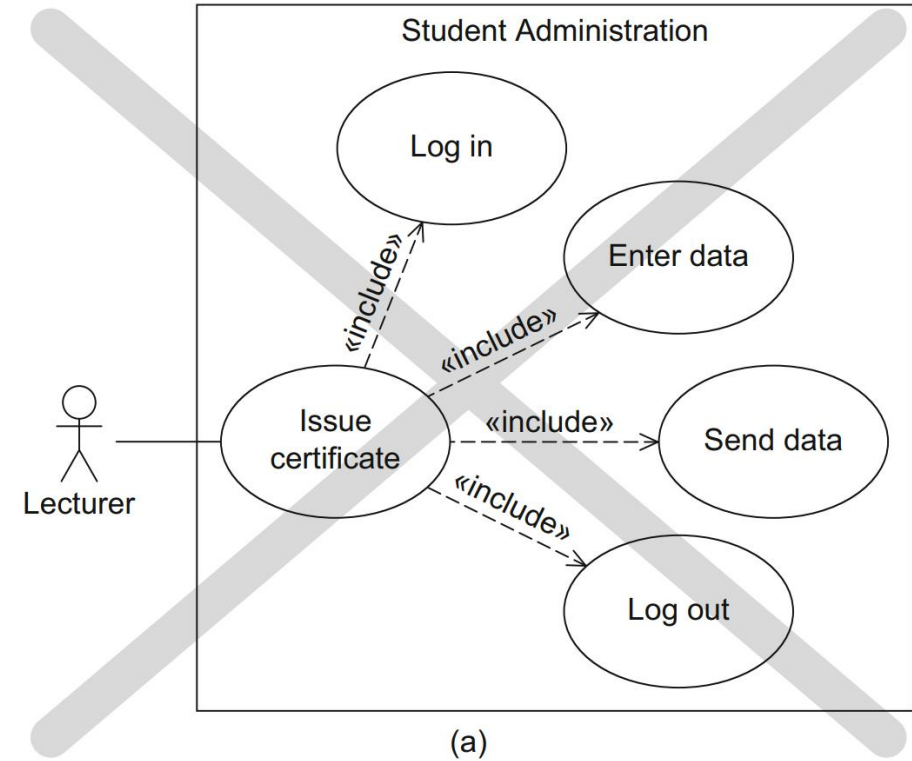
Common Problems

2. Setting wrong system boundary



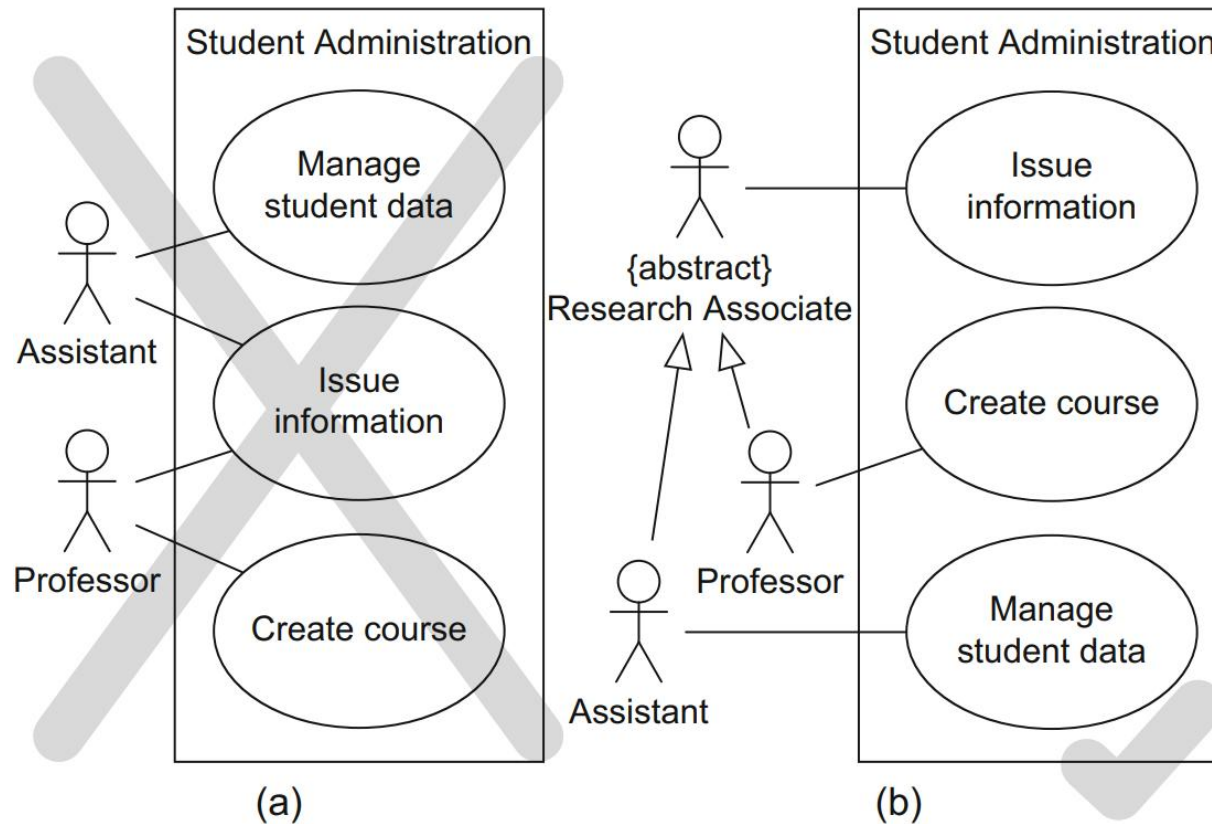
Common Problems

3. Functional decomposition



Common Problems

4. Incorrect association



Example

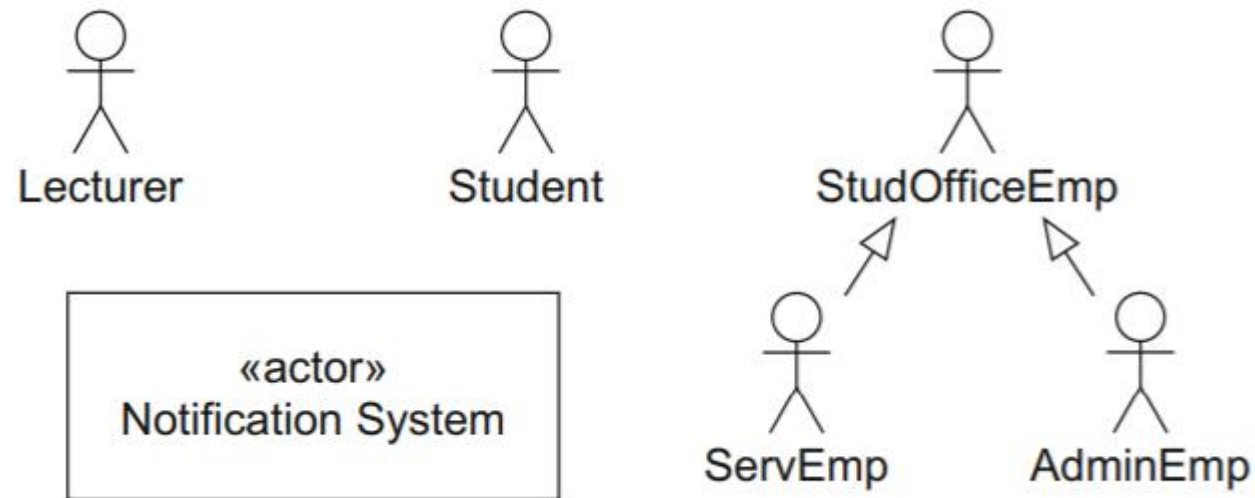
- Many important administrative activities of a university are processed by the student office. Students can register for studies (matriculation), enroll, and withdraw from studies here. Matriculation involves enrolling, that is, registering for studies.
- Students receive their certificates from the student office. The certificates are printed out by an employee. Lecturers send grading information to the student office. The notification system then informs the students automatically that a certificate has been issued.
- There is a differentiation between two types of employees in the student office: a) those that are exclusively occupied with the administration of student data (service employee, or ServEmp), and b) those that fulfill the remaining tasks (administration employee, or AdminEmp), whereas all employees (ServEmp and AdminEmp) can issue information.
- Administration employees issue certificates when the students come to collect them. Administration employees also create courses. When creating courses, they can reserve lecture halls.

Example

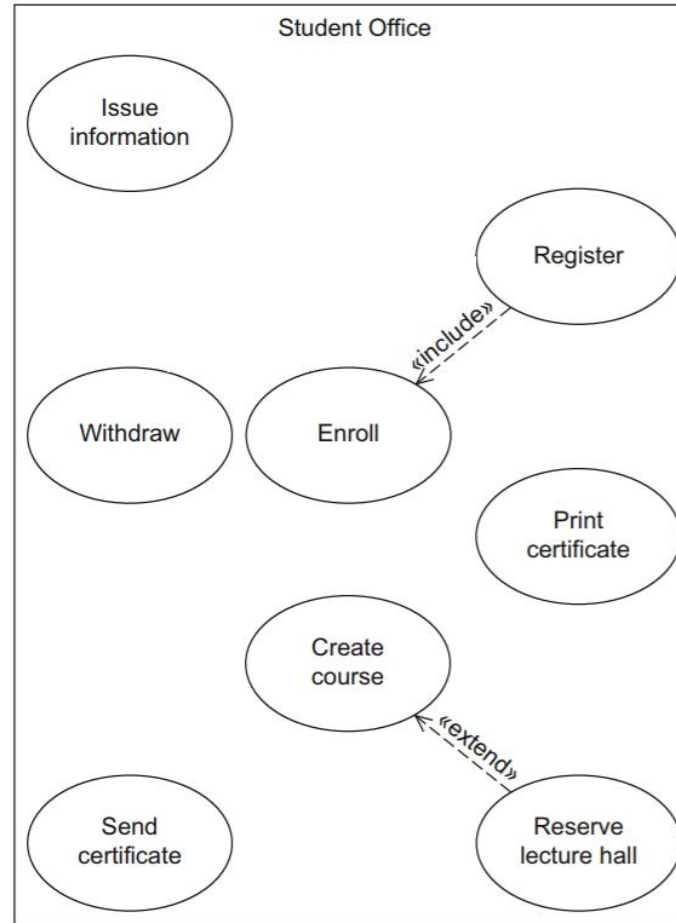
- Many important administrative activities of a university are processed by the student office. **Students** can register for studies (matriculation), enroll, and withdraw from studies here. Matriculation involves enrolling, that is, registering for studies.
- Students receive their certificates from the student office. The certificates are printed out by an employee. Lecturers send grading information to the student office. The notification system then informs the students automatically that a certificate has been issued.
- There is a differentiation between two types of employees in the student office: a) those that are exclusively occupied with the administration of student data (service employee, or ServEmp), and b) those that fulfill the remaining tasks (administration employee, or AdminEmp), whereas all employees (ServEmp and AdminEmp) can issue information.
- Administration employees issue certificates when the students come to collect them. Administration employees also create courses. When creating courses, they can reserve lecture halls.

Example: Actors

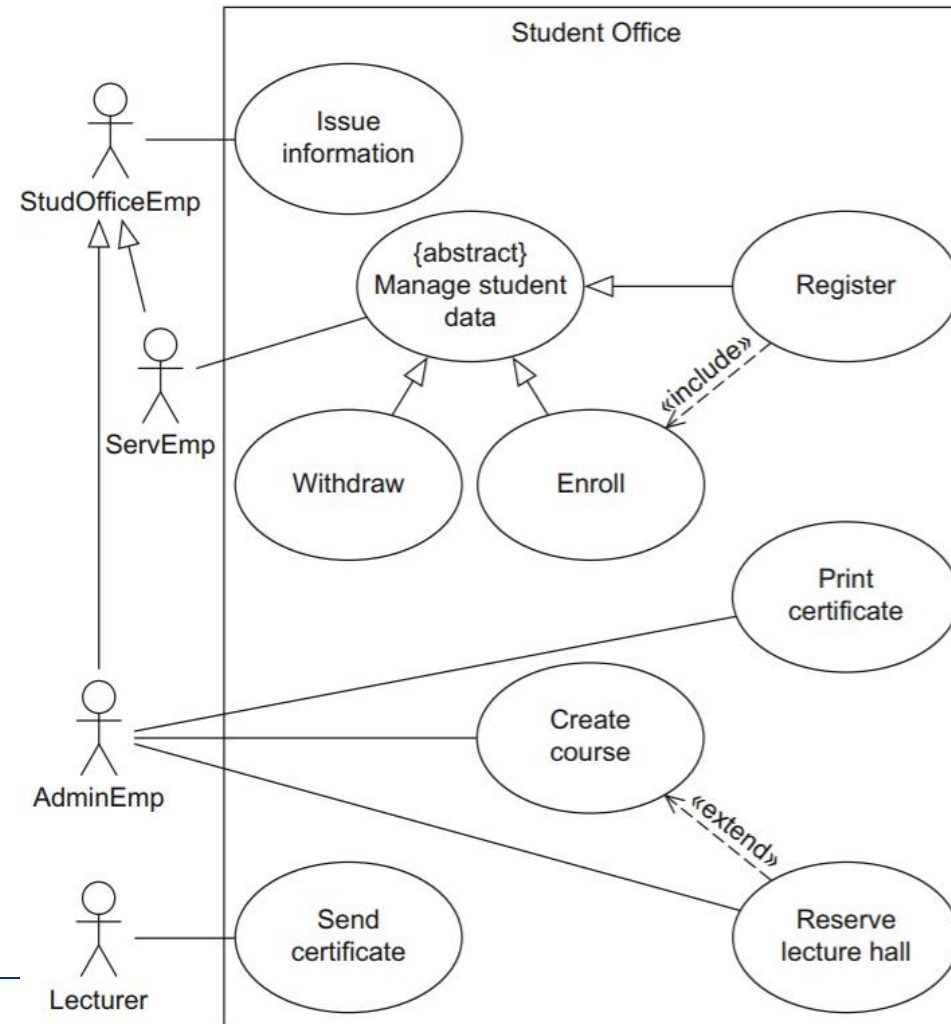
- Students and Notification system are not part of this use case



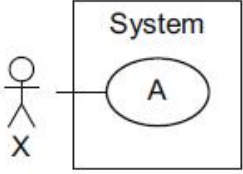
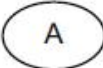
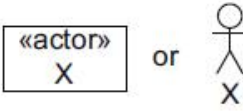
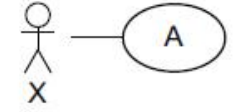
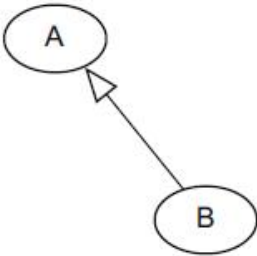
Example: Use Cases

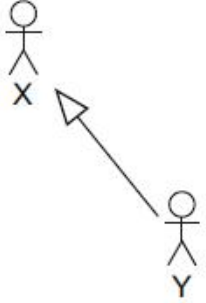
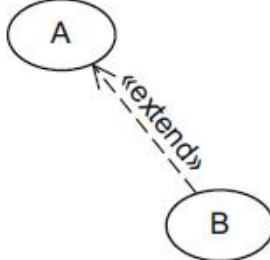
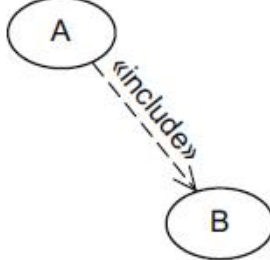


Example

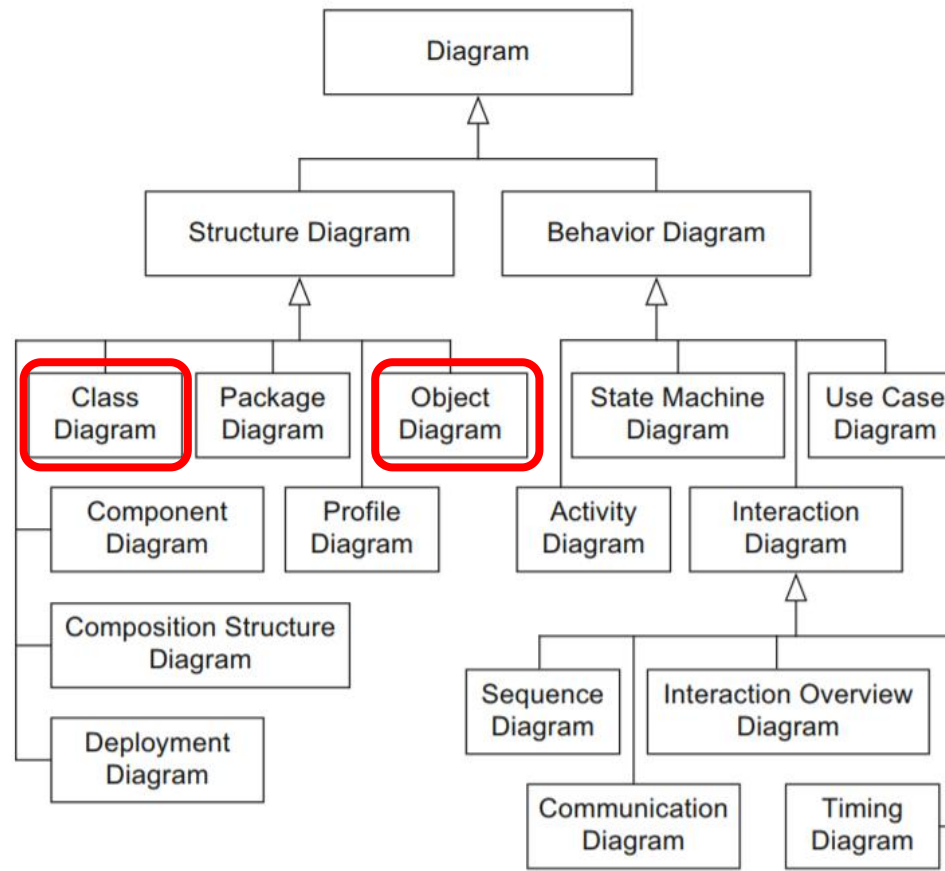


Summary: Use Case Diagram

Name	Notation	Description
System		Boundaries between the system and the users of the system
Use case		Unit of functionality of the system
Actor		Role of the users of the system
Association		X participates in the execution of A
Generalization (use case)		B inherits all properties and the entire behavior of A

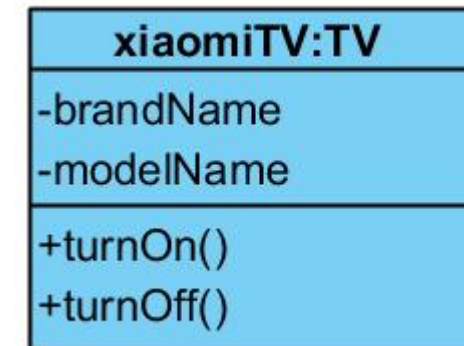
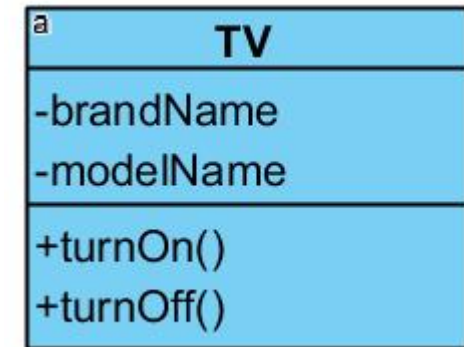
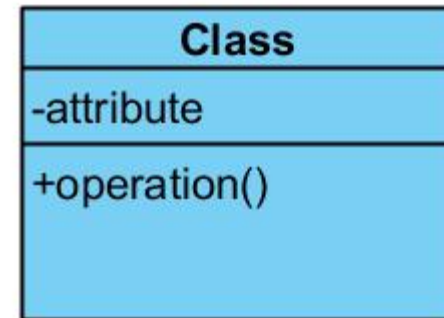
Generalization (actor)		Y inherits from X; Y participates in all use cases in which X participates
Extend relationship		B extends A: optional incorporation of use case B into use case A
Include relationship		A includes B: required incorporation of use case B into use case A

UML Diagrams



Class and object

- Class: The basic component of object-oriented approaches
- Each class has a list of **attributes** and **operations**
- Objects are *instances* of classes
- Visibility:
 - + global: accessible to all
 - - private: accessible within the obj
 - # protected: only accessible by its sub-classes



Attributes

- Name
 - Noun clause, lowercase first letter, then uppercase for latter words
 - i.e. gradStudent, firstName
- Data Type
 - i.e. String
- Multiplicity: how many value it can contain
 - [min .. max]: i.e. [0 .. 1]
- Example: address: String [1..*]= “Huanke Rd 199”
- Which attributes to include depends on the stage of development
 - The closer to implementation, the more detailed the models are

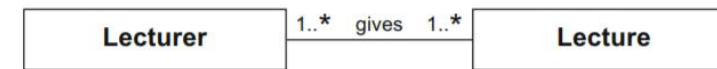
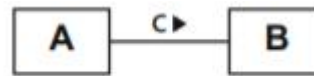
Operations

- Name
 - Verb clause: i.e. `getGrade()`
- Parameters
 - Direction: in, out, inout
 - Name
 - Data type
- Return value
 - Only need a data type
- Example
 - `getName(out fn: String, out in: String): void`
 - `updateLastName(in newName: String): boolean`

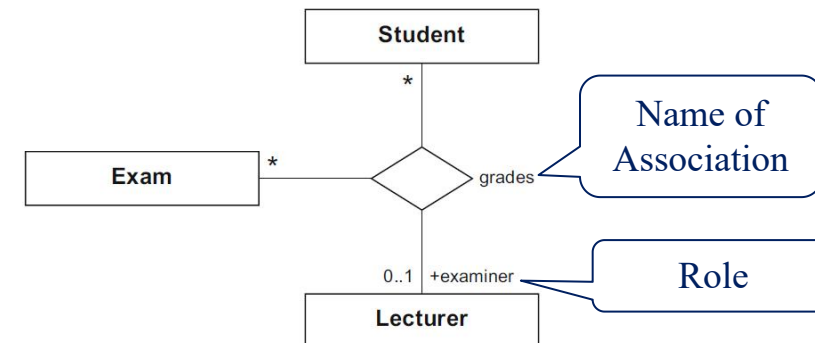
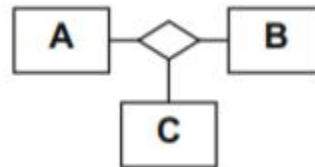
Association

- Possible relationships between instances of the classes.
- Can access attributes and operations with corresponding visibility

- Binary association



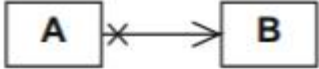
- N-nary association

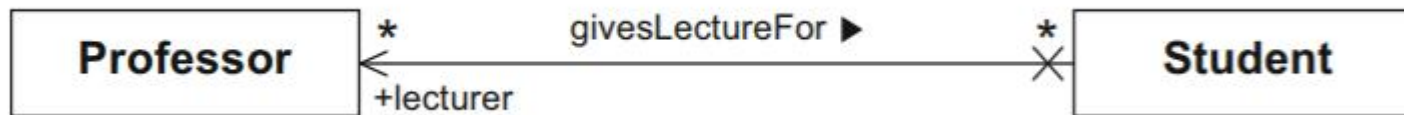


- Multiplicity (Cardinality)

- The number of objects that may be associated with exactly **one** object of the opposite side

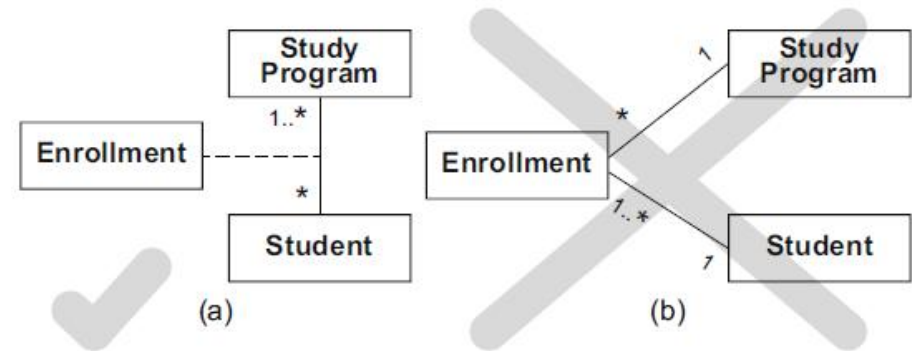
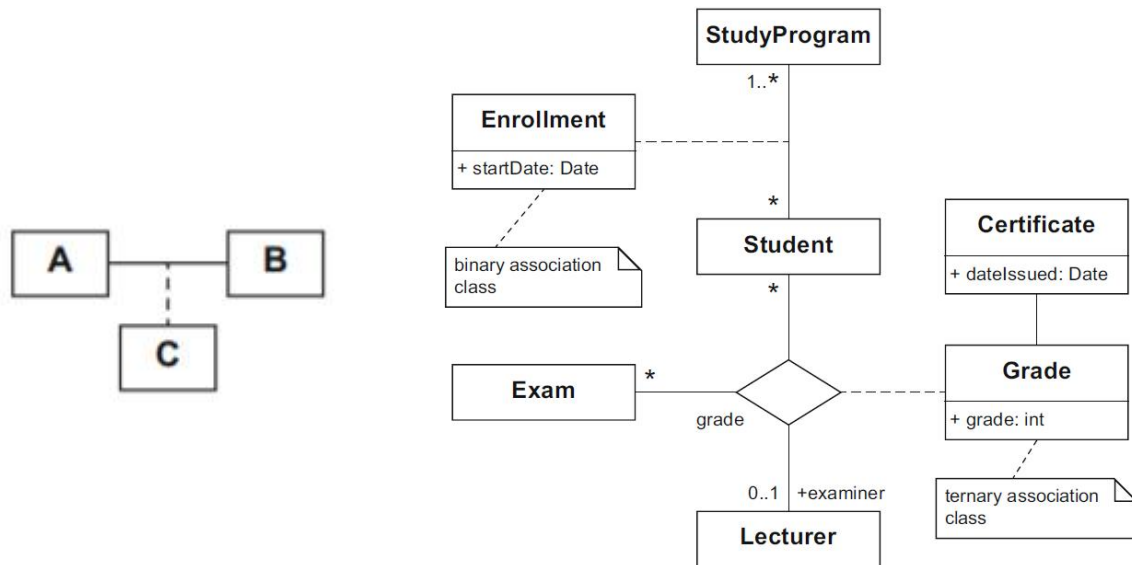
Navigability

- By default the information sharing is bi-directional
- Non-navigability 
 - A can access visible information of B
 - B cannot access information of A



Association Class

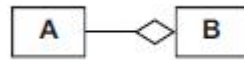
- Has the property of both a class and an association
- Cannot be simply replaced by a “normal” class
- In (b), a student can enroll a study program multiple times



Aggregation

- A special form association: A is part of B

- Shared aggregations



- A can also be part of something else
- When B is gone, A can still exist



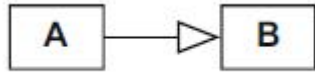
- Compositions

- A specific part can only be contained in **at most one** composite object at one specific point in time.
- A much stronger bond (normally physical)

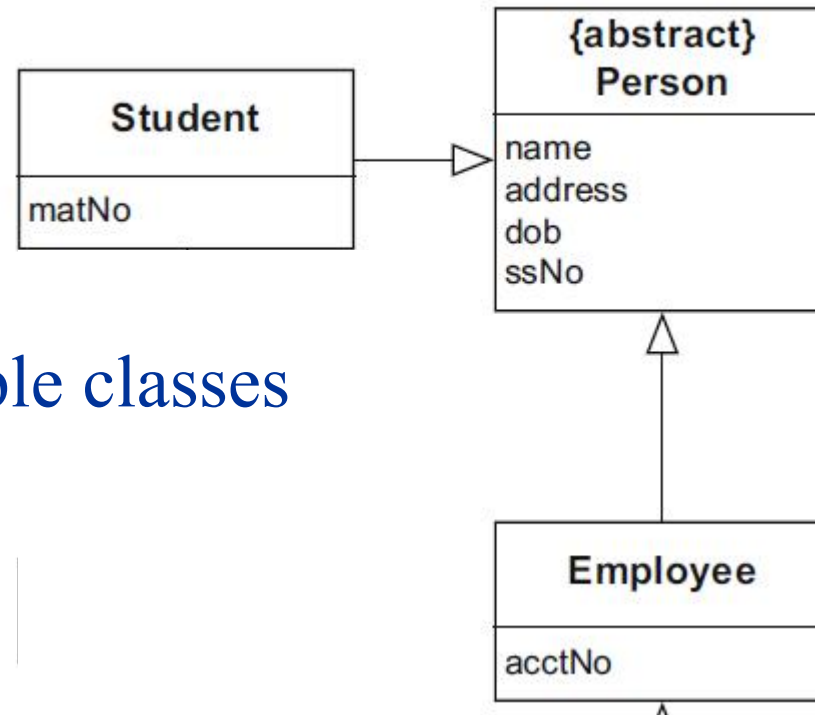


Generalization/Inheritance

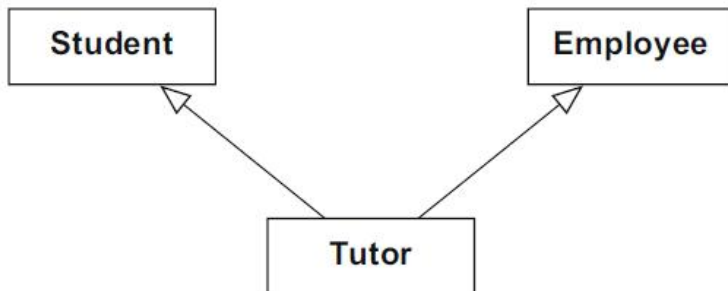
- Highlight common attributes and methods of objects and classes



- Abstract class
 - No instances



- A class can inherit from multiple classes



An Example

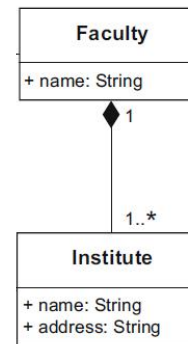
- A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.
- Each faculty is led by a dean, who is an employee of the university.
- The total number of employees is known. Employees have a social security number, a name, and an e-mail address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates teach courses. They are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.

An Example

- A university consists of multiple **faculties** which are composed of various **institutes**. Each faculty and each institute has a **name**. An **address** is known for each institute.
- Each faculty is led by a dean, who is an **employee** of the university.
- The total number of employees is known. Employees have a **social security number**, a **name**, and an **e-mail address**. There is a distinction between research and **administrative personnel**.
- **Research associates** are assigned to at least one institute. The **field of study** of each research associate is known. Furthermore, research associates can be involved in **projects** for a certain number of **hours**, and the **name**, **starting date**, and **end date** of the projects are known. Some research associates teach **courses**. They are called **lecturers**.
- Courses have a unique number (**ID**), a **name**, and a **weekly duration** in hours.

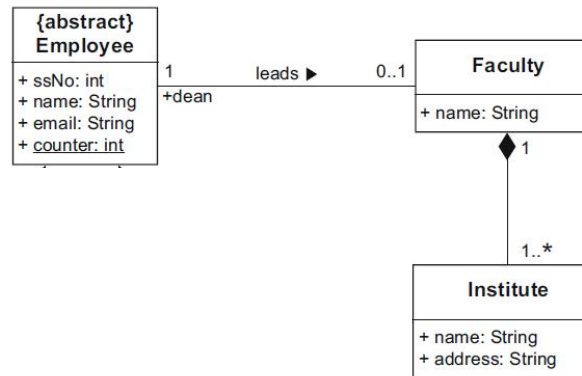
An Example

- A university consists of multiple **faculties** which are composed of various **institutes**. Each faculty and each institute has a **name**. An **address** is known for each institute.



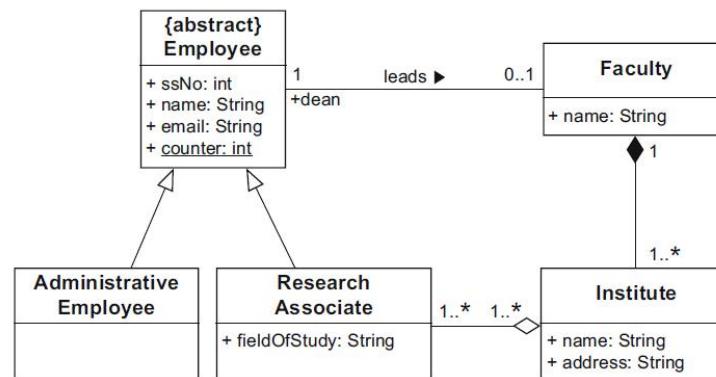
An Example

- Each faculty is led by a dean, who is an **employee** of the university.



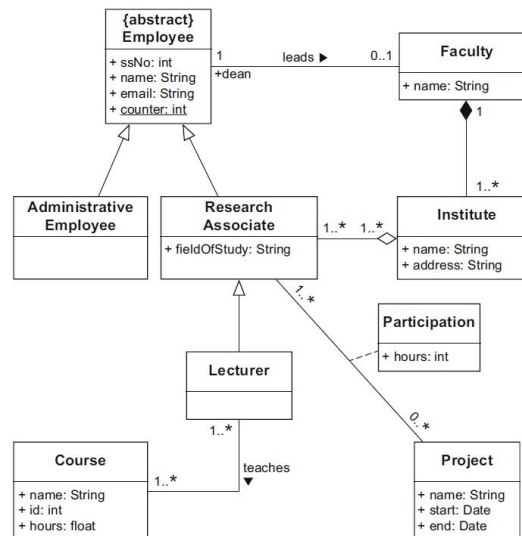
An Example

- The total number of employees is known. Employees have a social security number, a name, and an e-mail address. There is a distinction between research and administrative personnel.
- Research associates are assigned to at least one institute.

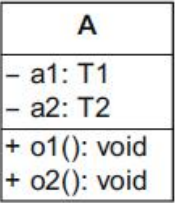

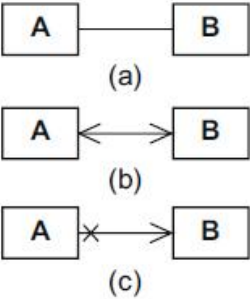
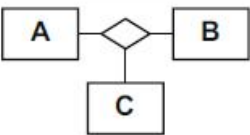


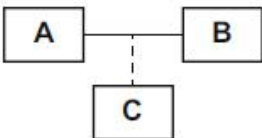
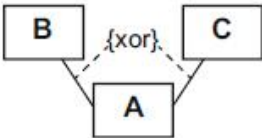

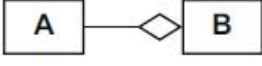
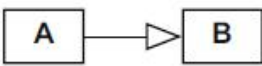


An Example

- The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates teach courses. They are called lecturers.
- Courses have a unique number (ID), a name, and a weekly duration in hours.



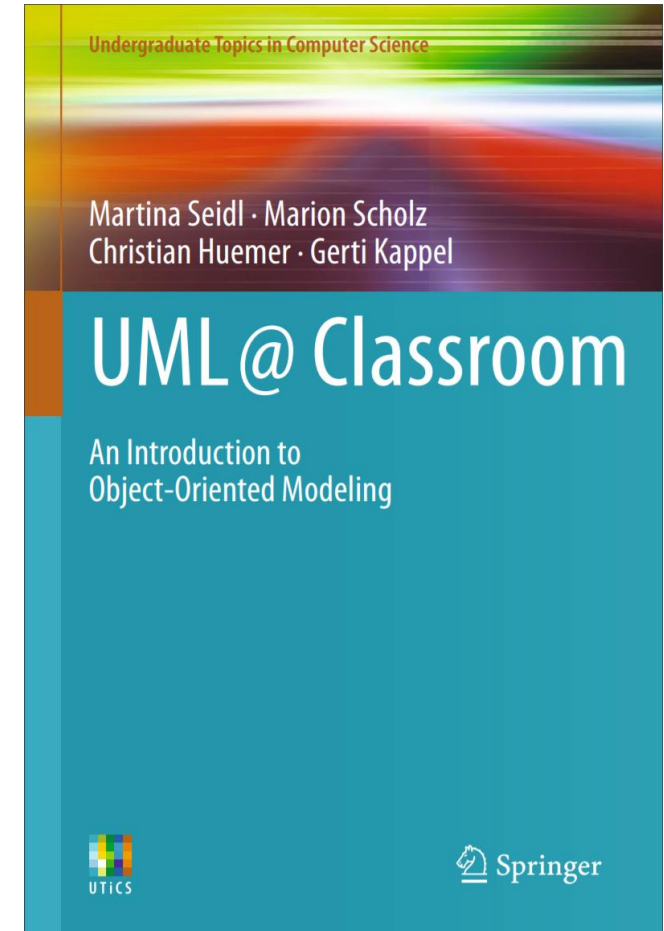
Summary: Class Diagram

Name	Notation	Description
Class		Description of the structure and behavior of a set of objects
Abstract class		Class that cannot be instantiated
Association		Relationship between classes: navigability unspecified (a), navigable in both directions (b), not navigable in one direction (c)
N-ary association		Relationship between N (in this case 3) classes

Association class		More detailed description of an association
xor relationship		An object of A is in a relationship with an object of B or with an object of C but not with both
Strong aggregation = composition		Existence-dependent parts-whole relationship (A is part of B ; if B is deleted, related instances of A are also deleted)
Shared aggregation		Parts-whole relationship (A is part of B ; if B is deleted, related instances of A need not be deleted)
Generalization		Inheritance relationship (A inherits from B)
Object		Instance of a class
Link		Relationship between objects

Reference for UML

- Freely available online
- Search from our library website



UML Drawing Tools

- Microsoft Visio can draw basic UML diagrams
 - Available from the library
- Visual Paradigm (Community edition)
 - <https://www.visual-paradigm.com/download/community.jsp>
- IBM Rational Rose
 - Cracked version online (not recommended)