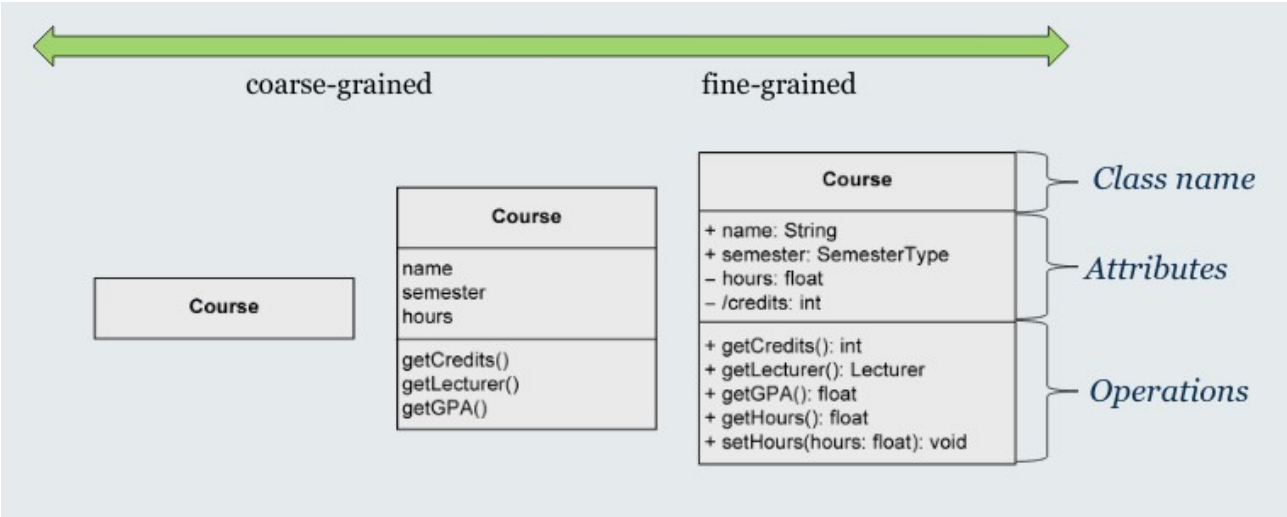
UML CLASS DIAGRAMS AND CONFIGURATION MANAGEMENT

CLASS DIAGRAMS

Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.

- An object class can be thought of as a general definition of one kind of system object.
- An association is a link between classes that indicates that there is some relationship between these classes
- When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, a doctor, etc.

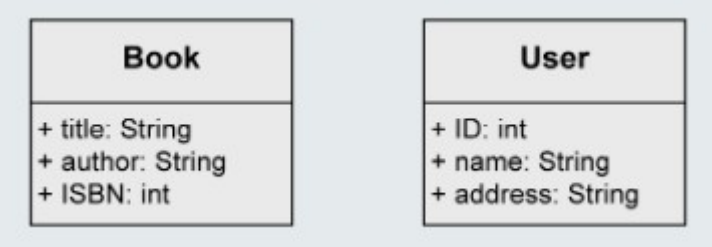


CREATING A CLASS DIAGRAM

- It is not possible to completely extract classes, attributes and associations from a natural language text automatically.
- Some guidelines:
  - Nouns often indicate classes
  - Adjectives often indicate attribute values
  - Verbs often indicate operations

e.g.:

The library management system stores users with their unique ID, name and address as well as books with their title, author and ISBN number. Ann Foster wants to use the library.



FROM OBJECTS TO CLASS

Individuals of a system often have identical characteristics and behaviour

A class is a construction plan for a set of similar objects of a system.

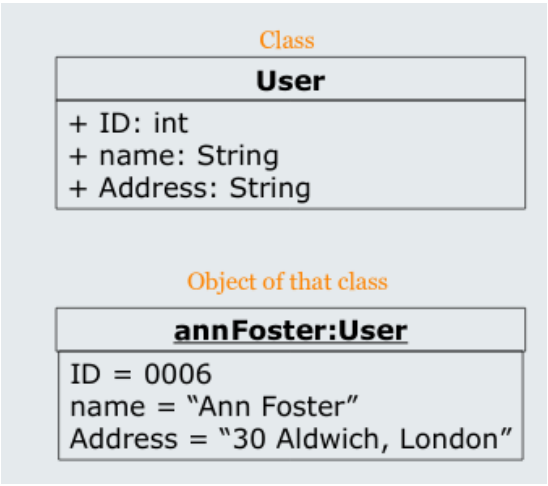
- Objects are instances of classes

Attributes: structural characteristics of a class

- Different value for each instance (=object)

Operations: behaviour of a class

- Identical for all objects of a class
- Not depicted in object diagram



EXAMPLE – UNIVERSITY INFORMATION SYSTEM

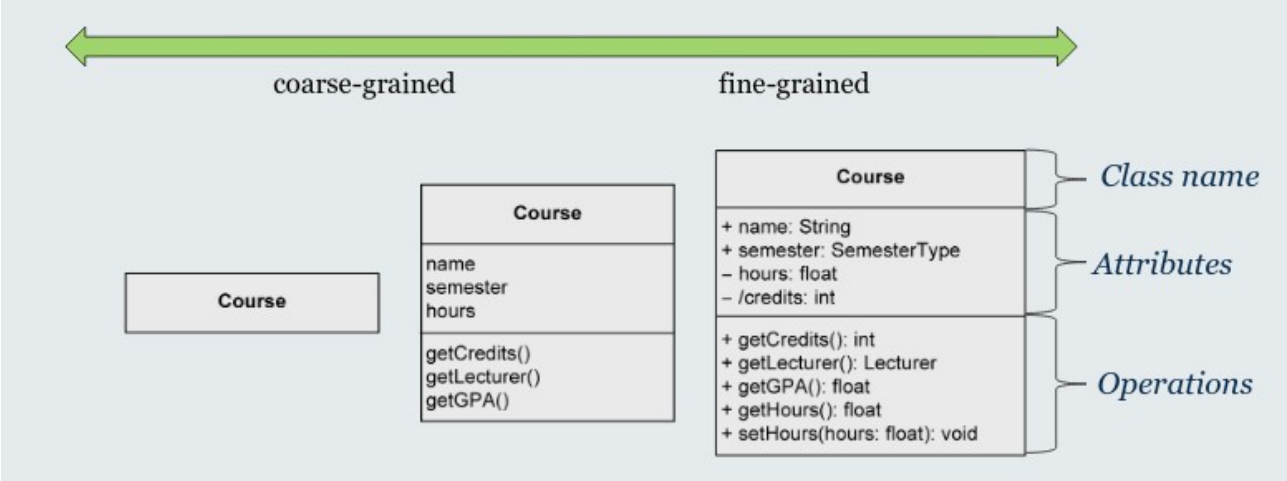
A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.

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

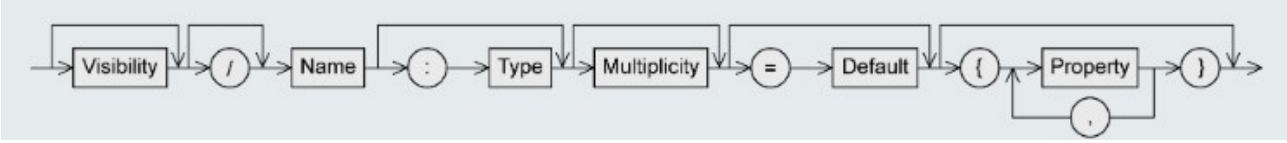
The total number of employees is known. Employees have an SSN, a name, and an email address. There is a distinction between research and administrative personnel.

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 hold courses. Then they are called lecturers.

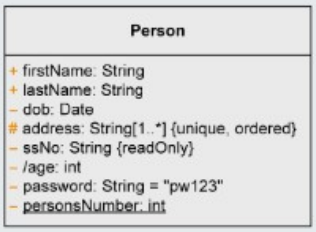
Courses have a unique number (ID), a name, and a weekly duration in hours.



ATTRIBUTE SYNTAX

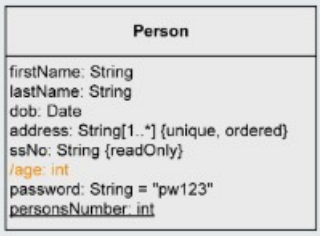


VISIBILITY



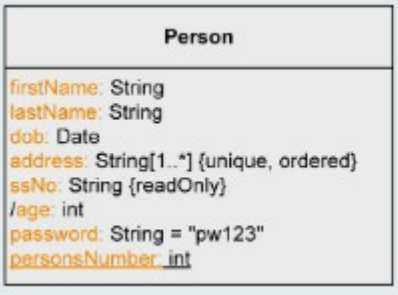
- Who is permitted to access the attribute
  - + ... public: everybody
  - - ... private: only the object itself
  - # ... protected: class itself and subclasses
  - ~ ... package: classes that are in the same package

DERIVED ATTRIBUTE



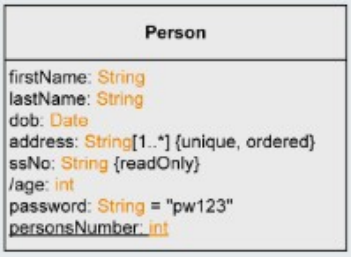
- Attribute value is derived from other attributes
  - **age**: calculated from the date of birth

NAME

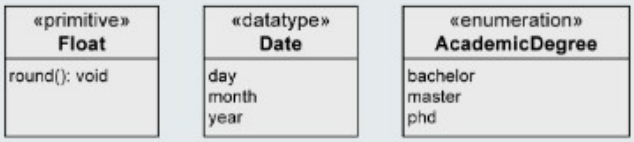


- Name of the attribute

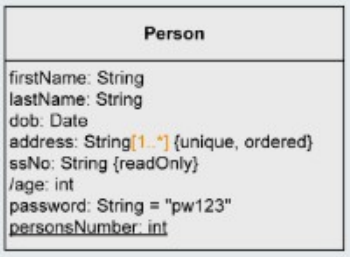
TYPE



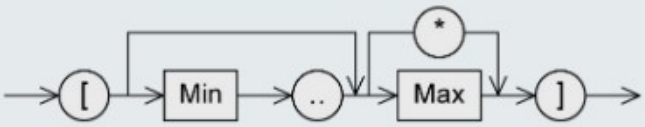
- Type
  - User-defined classes
  - Data type
    - Primitive data type
      - Pre-defined: Boolean, Integer, String
      - User-defined: «**primitive**»
      - Composite data type: «**datatype**»
    - Enumerations: «**enumeration**»



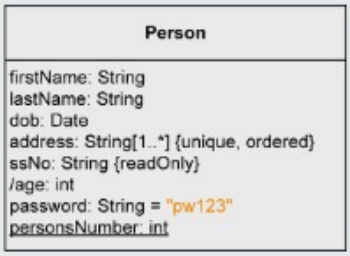
MULTIPLICITY



- Number of values an attribute may contain
- Default value: **1**
- Notation: [min..max]
  - no upper limit: **[*]** or **[0..*]**

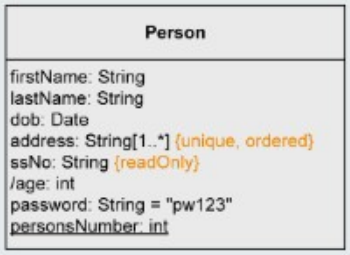


DEFAULT VALUE



- Default value
  - Used if the attribute value is not set explicitly by the user

PROPERTIES



- Pre-defined properties
  - `{readOnly}` ... value cannot be changed
  - `{unique}` ... no duplicates permitted
  - `{non-unique}` ... duplicates permitted
  - `{ordered}` ... fixed order of the values
  - `{unordered}` ... no fixed order of the values
- Attribute specification
  - Set: `{unordered, unique}`
  - Multi-set: `{unordered, non-unique}`
  - Ordered set: `{ordered, unique}`
  - List: `{ordered, non-unique}`

EXAMPLE 2: IDENTIFYING THE ATTRIBUTES

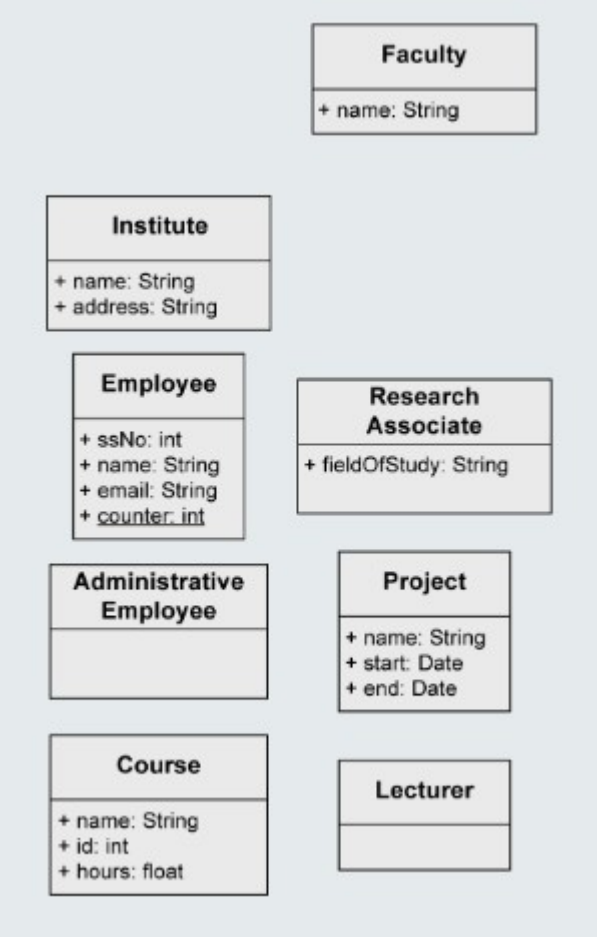
A university consists of multiple faculties which are composed of various institutes. Each faculty and each institute has a name. An address is known for each institute.

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

The total number of employees is known. Employees have an SSN, a name, and an email address. There is a distinction between research and administrative personnel.

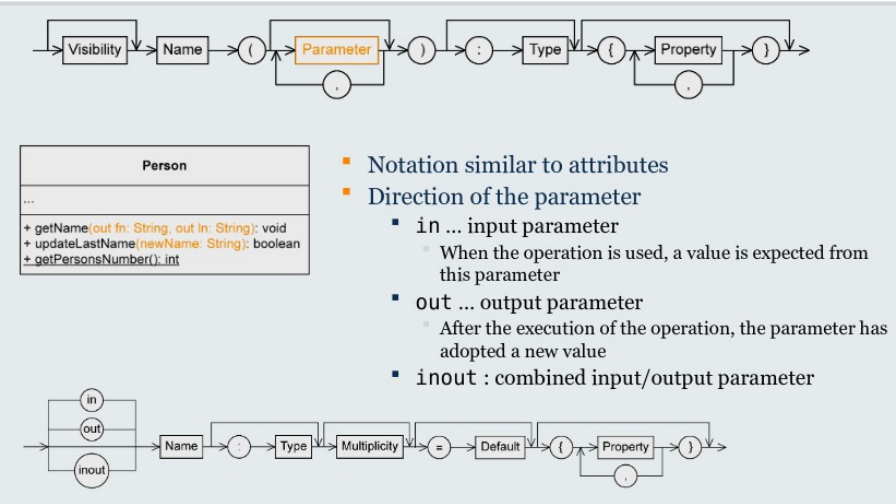
Research associates are assigned to at least one institute. The field of study of each research associate is known. Furthermore, research associates can be involved in projects for a certain number of hours, and the name, starting date, and end date of the projects are known. Some research associates hold courses. Then they are called lecturers.

Courses have a unique number (ID), a name, and a weekly duration in hours.

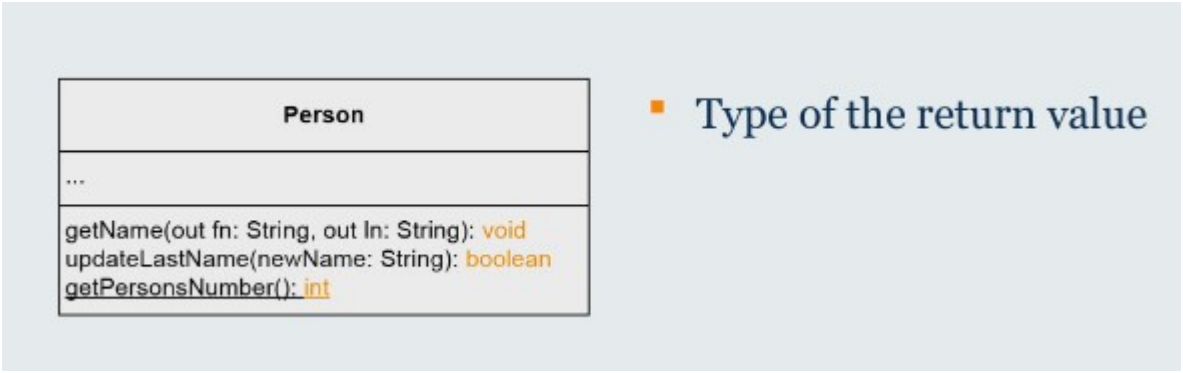


OPERATION SYNTAX

PARAMETERS



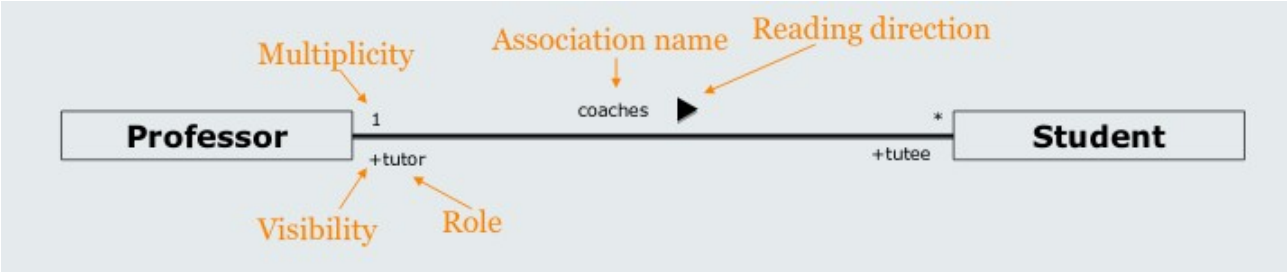
TYPE



CLASS DIAGRAMS AND RELATIONSHIPS

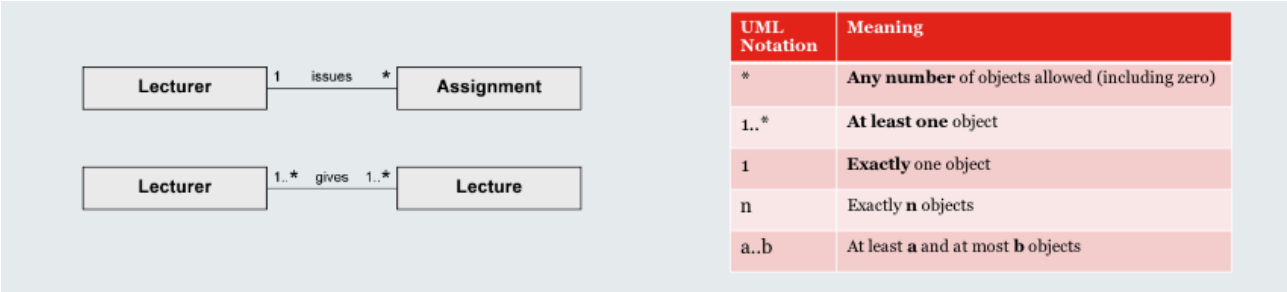
BINARY ASSOCIATION

- Define relationships between instances (objects) of classes.
- There is an association between two classes if an instance of one class must know about the other in order to perform its work
- Role names and multiplicities at both ends
- In uni-directional associations, role name at start of arrow is optional
- Formalise requirements such as “Each professor may coach a set of students, and each set of students is coached by exactly one professor”



MULTIPLICITY AND ROLE

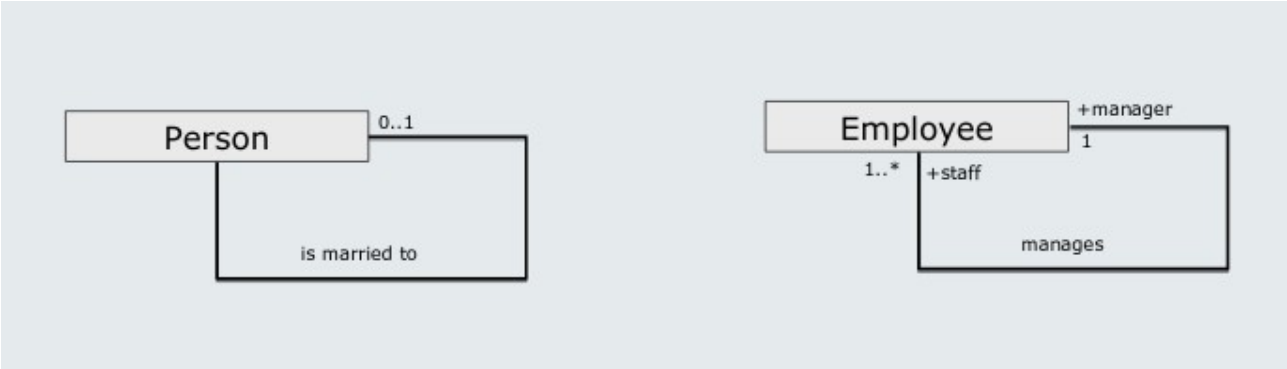
- Multiplicity: Number of objects that may be associated with exactly one object of the opposite side



UML Notation	Meaning
*	**Any number** of objects allowed (including zero)
1..*	**At least one** object
1	**Exactly** one object
n	Exactly **n** objects
a..b	At least **a** and at most **b** objects

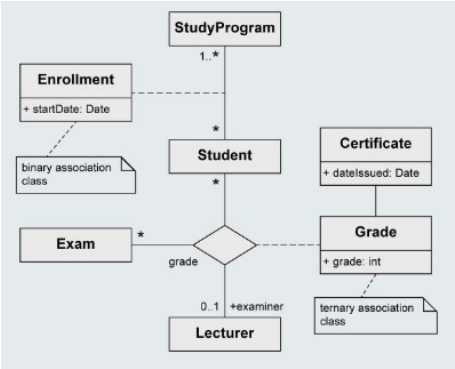
- Role: describes the way in which an object is involved in an association relationship

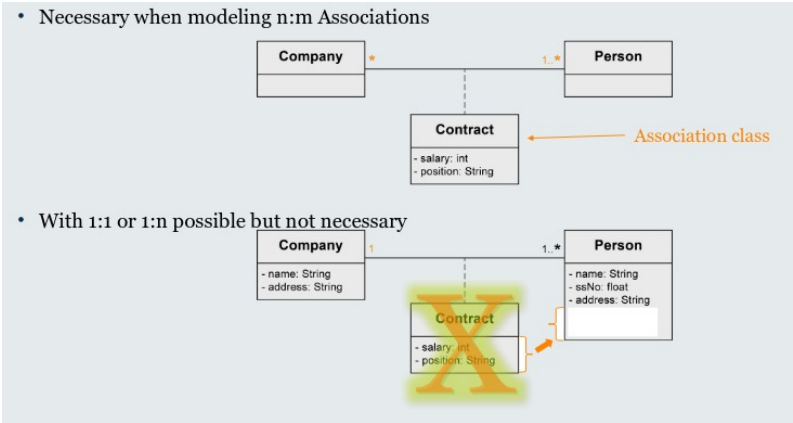
UNARY ASSOCIATION EXAMPLE



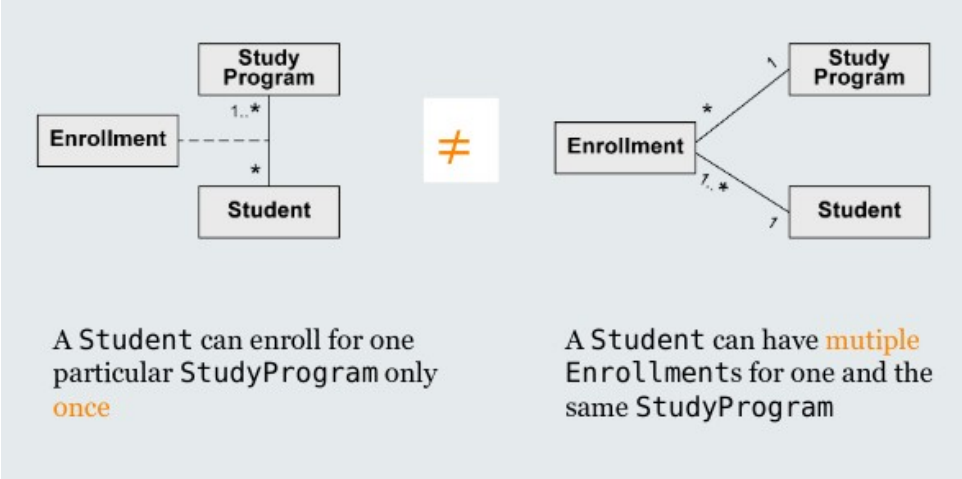
ASSOCIATION CLASS

Assign attributes to the relationship between classes rather than to a class itself





VS. REGULAR CLASS

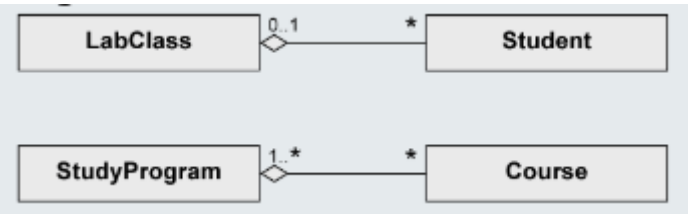


AGGREGATION

- A special form of association
- Used to express that a class is part of another class
- Properties of the aggregation association:
  - Transitive: if B is a part of A and C is a part of B, C is also a part of A.
  - Asymmetric: it is not possible for A to be a part of B and B to be a part of A simultaneously

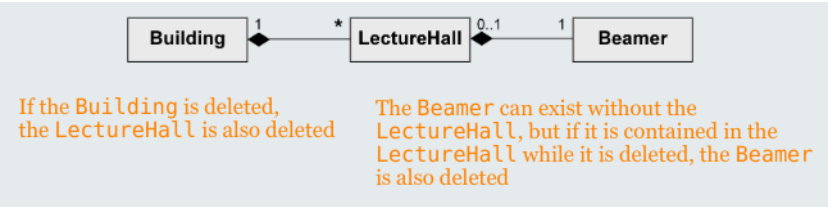
SHARED AGGREGATION

- Expresses a weak belonging of the parts to a whole
  - = Parts also exist independently of the whole
- Multiplicity at the aggregating end may be >1
  - = One element can be a part of multiple other elements simultaneously
- Spans a directed acyclic graph
- Syntax: Hollow diamond at the aggregating end
- Example:
  - Student is a part of LabClass
  - Course is a part of StudyProgram



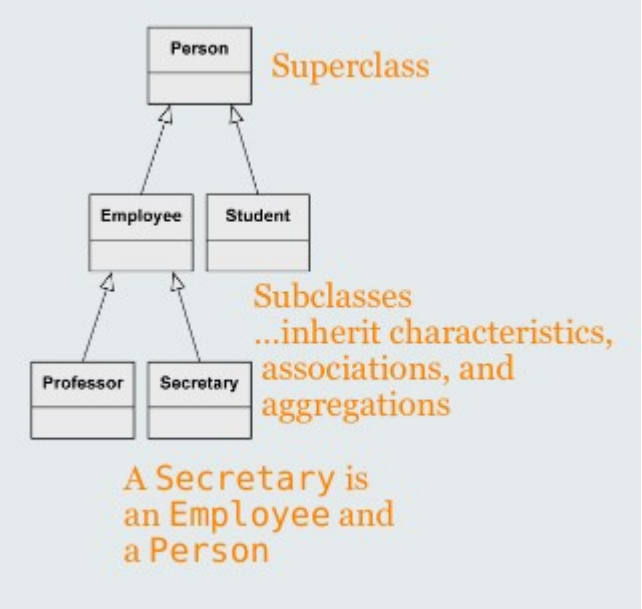
COMPOSITION

- Existence dependency between the composite object and its parts
- One part can only be contained in at most one composite object at one specific point in time
  - Multiplicity at the aggregating end max. 1
  - The composite objects form a tree
- If the composite object is deleted, its parts are also deleted
- Syntax: Solid diamond at the aggregating end
- Example:
  - Beamer is a part of LectureHall is a part of Building



GENERALIZATION

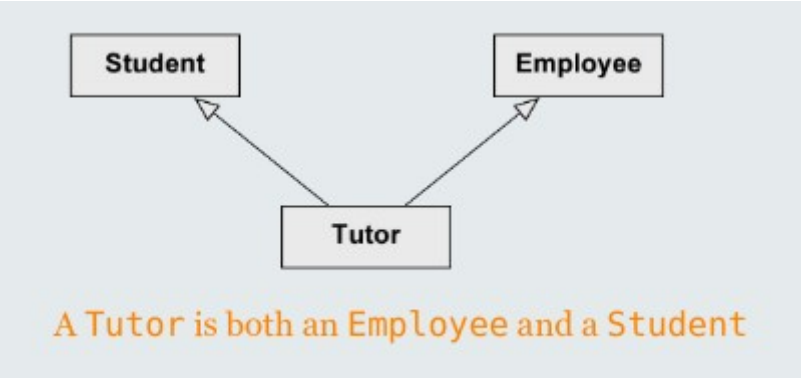
- Characteristics (attributes and operations), associations and aggregations that are specified for a general class (superclass) are passed on to its subclasses.
- Every instance of a subclass is simultaneously an indirect instance of the superclass.
- Subclass inherits all characteristics, associations, and aggregations of the superclass except private ones.
- Generalizations are transitive.



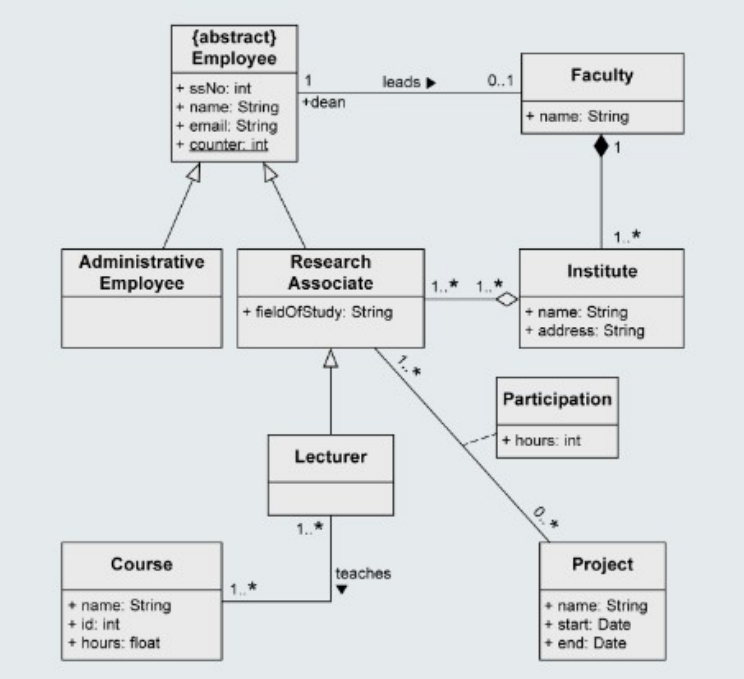
MULTIPLE INHERITANCE

UML allows multiple inheritance, such that a class may have multiple superclasses.

e.g.:



COMPLETE CLASS DIAGRAM



CONSISTENCY RULES

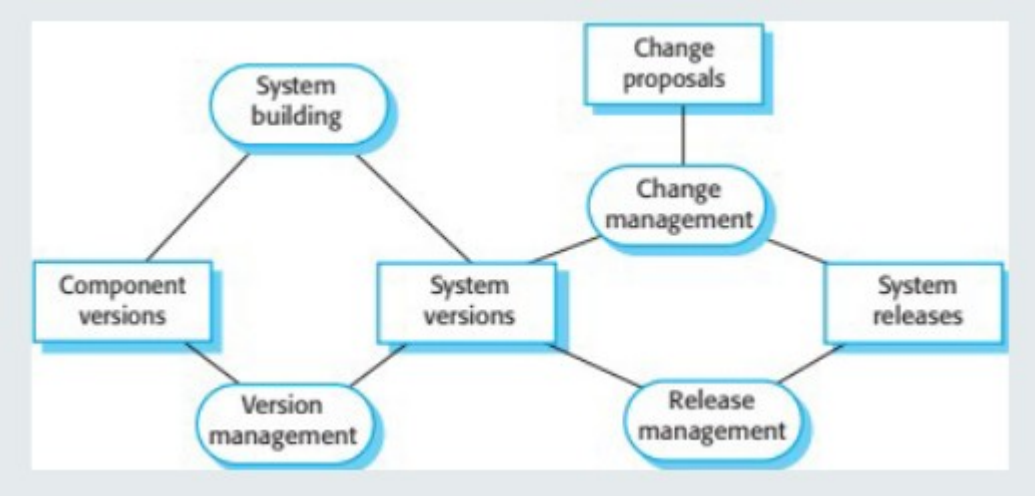
- NO two attributes with same name
- NOT an attribute and a role with same name
- NO two associations from one class with same role names at their other ends – even if they end at different classes.

CONFIGURATION MANAGEMENT

- Software systems are constantly changing during development and use
- Configuration management is concerned with the policies, processes and tools for managing changing software systems
- You need CM because it is easy to lose track of what changes and component versions have been incorporated into each system version
- CM is essential for team projects to control changes made by different developers

CM ACTIVITIES

- Version management
  - Keeping track of multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.
- System building
  - The process of assembling program components, data and libraries, then compiling these to create an executable system
- Change management
  - Keeping track of requests for changes to the software from customers and developers, working out the costs and impact of changes and deciding the changes should be implemented
- Release management
  - Preparing software for external release and keeping track of the system versions that have been released for customer use.



AGILE DEVELOPMENT AND CM

Agile development, where components and systems are changed several tiles per day, is impossible without using CM tools.

- The definitive versions of components are held in a shared project repository and developers copy these into their own workspace.
- They make changes to the code and then use system building tools to create a new system on their own computer for testing. Once they are happy with the changes made, they return the modified components to the project repository.

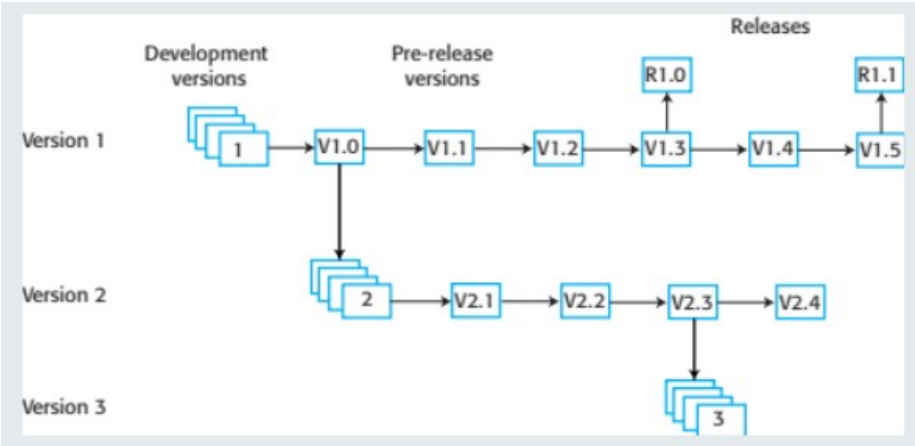
DEVELOPMENT PHASES

- A development phase is where the development team that's responsible for managing the software configuration add new functionality to the software.
- A system testing phase is where a version of the system is released internally for testing.
  - No new system functionality is added. Changes made are bug fixes, performance improvements and security vulnerability repairs.
- A release phase is where the software is released to customers for use.
  - New versions of the release system are developed to repair bugs and vulnerabilities and to include new features.

MULTI-VERSION SYSTEMS

For large systems, there is never just one 'working' version of a system. There are several versions of the system at different stages of development.

There may be several teams involved in the development of different system versions.



TERMINOLOGY

Term	Explanation
Baseline	A baseline is a collection of component versions that make up a system. Baselines are controlled, which means that the versions of the components making up the system cannot be changed. This means that it is always possible to recreate a baseline from its constituent components.
Branching	The creation of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently.
Codeline	A codeline is a set of versions of a software component and other configuration items on which that component depends.
Configuration (version) control	The process of ensuring that versions of systems and components are recorded and maintained so that changes are managed and all versions of components are identified and stored for the lifetime of the system.
Configuration item or software configuration item (SCI)	Anything associated with a software project (design, code, test data, document, etc.) that has been placed under configuration control. There are often different versions of a configuration item. Configuration items have a unique name.
Mainline	A sequence of baselines representing different versions of a system.

Term	Explanation
Merging	The creation of a new version of a software component by merging separate versions in different codelines. These codelines may have been created by a previous branch of one of the codelines involved.
Release	A version of a system that has been released to customers (or other users in an organization) for use.
Repository	A shared database of versions of software components and meta-information about changes to these components.
System building	The creation of an executable system version by compiling and linking the appropriate versions of the components and libraries making up the system.
Version	An instance of a configuration item that differs, in some way, from other instances of that item. Versions always have a unique identifier.
Workspace	A private work area where software can be modified without affecting other developers who may be using or modifying that software.