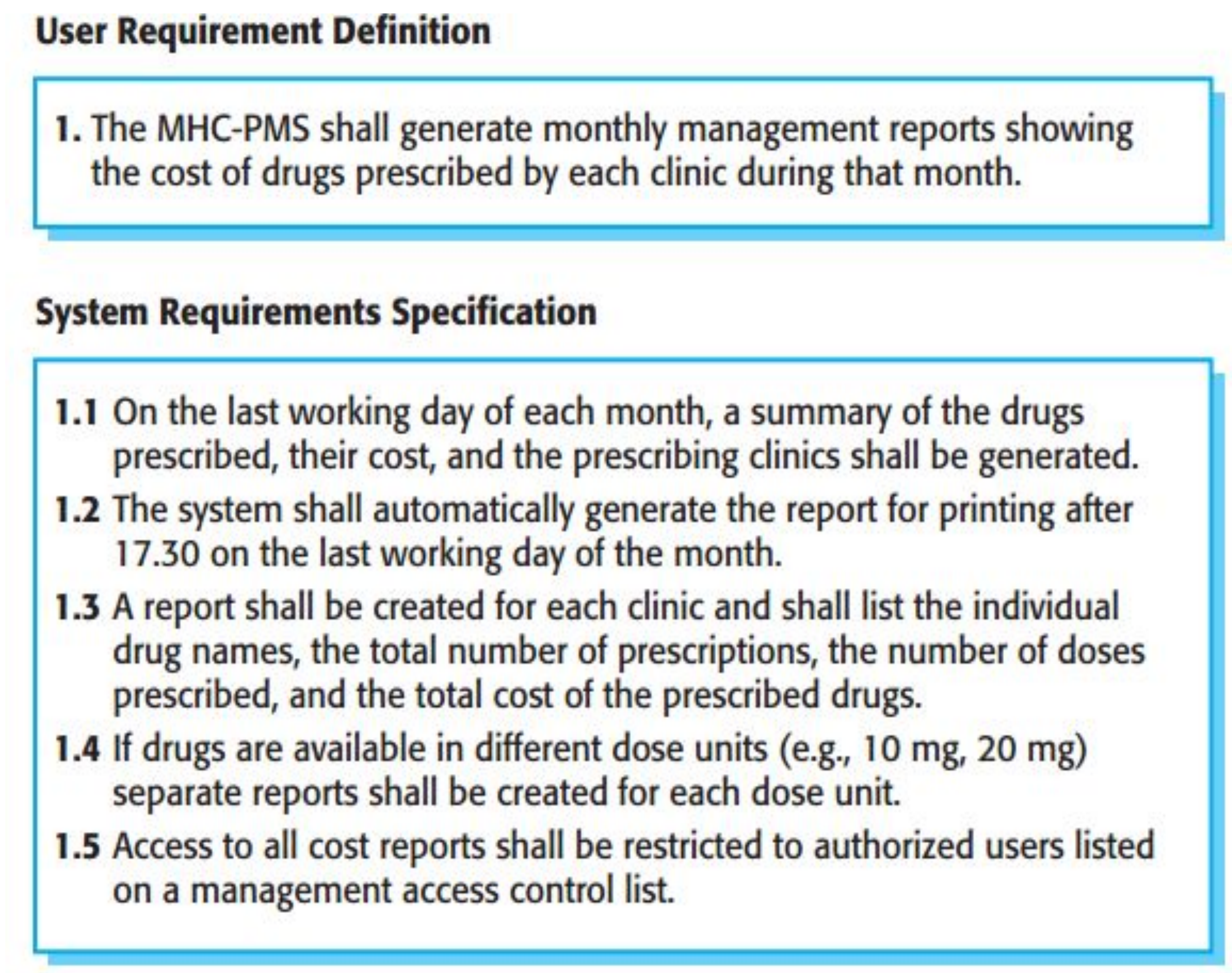
CSE 470
Software Engineering

Requirements | Class Diagram

Imran Zahid
Lecturer
Computer Science and Engineering, BRAC University

Requirements Engineering

What are Requirements - an Example



User Requirement Definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System Requirements Specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.

1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.

1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.

1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Medical Health Care
Patient Management System

What are Requirements?

- It may range from
  - A high-level abstract statement of a service or of a system constraint to
  - A detailed mathematical functional specification.
- This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract – therefore must be open to interpretation;
  - May be the basis for the contract itself – therefore must be defined in detail;
  - Both these statements may be called requirements.

Functional and Non-Functional Requirements

Functional and Non-functional Requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.

Functional Requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

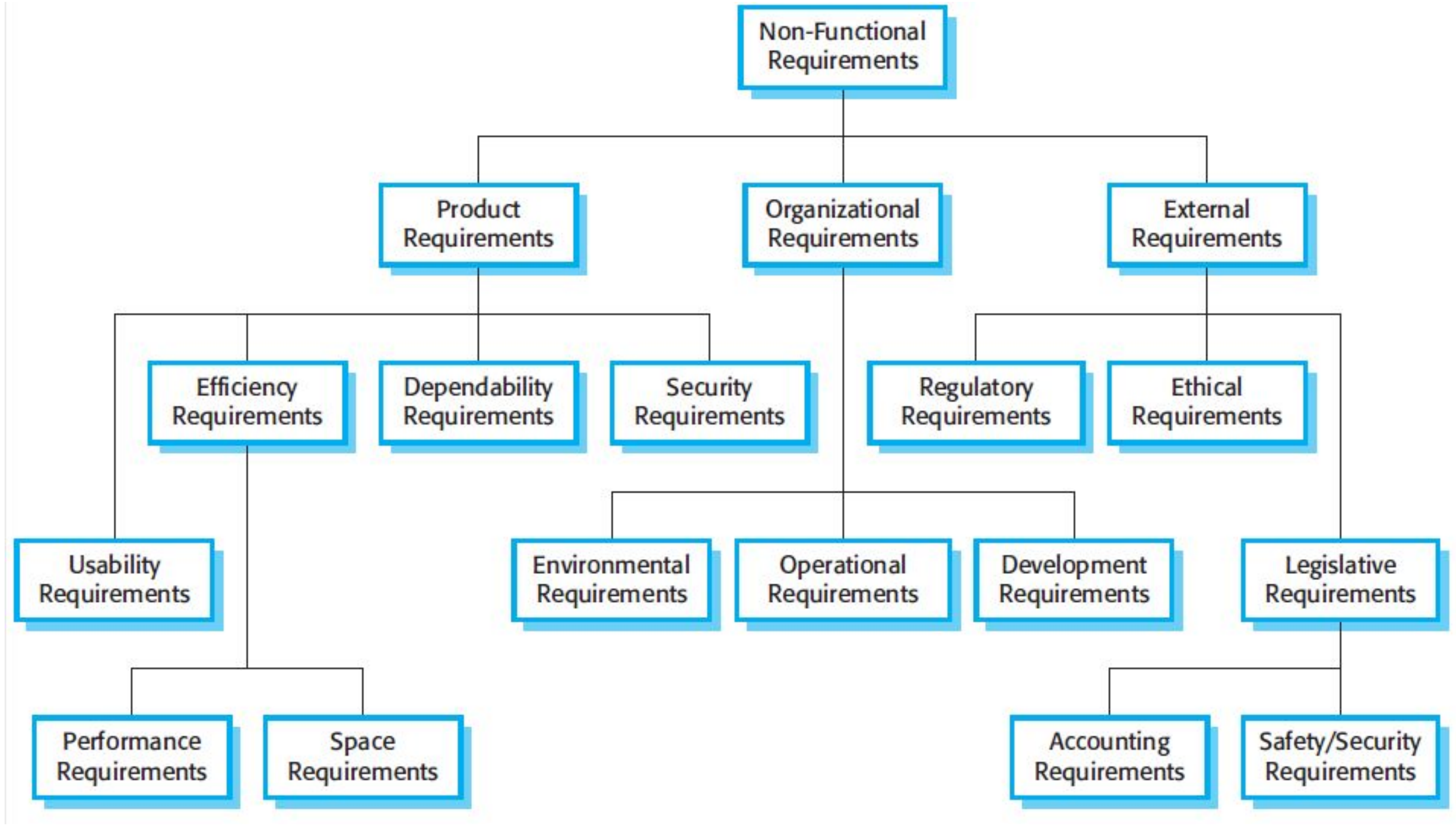
Non-functional Requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Non-functional Requirements - Classification

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Non-functional Requirements - Classification



Functional and Non-functional Requirements

Sl. No.	Functional Requirement	Non-Functional Requirement
1	Defines all the services or functions required by the customer that must be provided by the system	Defines system properties and constraints e.g. reliability, response time and storage requirements Constraints are I/O device capability, system representations, etc
2	It describes what the software should do	It does not describe what the software will do, but how the software will do it
3	Related to business. For example: Calculation of order value by Sales Department or gross pay by the Payroll Department	Related to improving the performance of the business. For example: checking the level of security.
4	Functional requirement are easy to test	Nonfunctional requirements are difficult to test
5	Related to the individual system features	Related to the system as a whole
6	Failure to meet the individual functional requirement may degrade the system	Failure to meet a non-functional requirement may make the whole system unusable
7	Always explicitly mentioned by the client	May be implied by the client

Example - Requirements

Functional requirement

1. A user shall be able to search the appointments lists for all clinics.
2. The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
3. Each staff member using the system shall be uniquely identified by his or her eight-digit employee number.

Product requirement

- The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement

- Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

External requirement

- The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Example - Functional Requirements

- ARENA supports five types of users:
  - The operator should be able to define new games, define new tournament styles (e.g., knock-out tournaments, championships, best of series), define new expert rating formulas, and manage users.
  - League owners should be able to define a new league, organize and announce new tournaments within a league, conduct a tournament, and declare a winner.
  - Players should be able to register in an arena, apply for a league, play the matches that are assigned to the player, or drop out of the tournament.
  - Spectators should be able to monitor any match in progress and check scores and statistics of past matches and players. Spectators do not need to register in an arena.
  - The advertiser should be able to upload new advertisements, select an advertisement scheme (e.g., tournament sponsor, league sponsor), check balance due, and cancel advertisements.

Example - Non-functional Requirements

- Low operating cost. The operator must be able to install and administer an arena without purchasing additional software components and without the help of a full-time system administrator.
- Extensibility. The operator must be able to add new games, new tournament styles, and new expert rating formulas. Such additions may require the system to be temporarily shut down and new modules (e.g., Java classes) to be added to the system. However, no modifications of the existing system should be required.

Example - Non-functional Requirements

- Scalability. The system must support the kick-off of many parallel tournaments (e.g., 10), each involving up to 64 players and several hundreds of simultaneous spectators.
- Low-bandwidth network. Players should boable to play matches via a 56K analog modem or faster

Example - Requirements

VOIP - Conference Call Project

Functional Requirements

- ❑ Add participants
- ❑ Drop participants
- ❑ Count participants
- ❑ Mute microphone
- ❑ Invite/prompt operator

Non-Functional Requirements

- ❑ Voice/Video Quality
- ❑ Reliability
- ❑ Availability
- ❑ Ease of Use
- ❑ Cost
- ❑ Localization

Metrics for Non-functional Requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of Use	Training time Number of help frames

Metrics for Non-functional Requirements

Property	Measure
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Class Diagram

Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Class Diagram - Design

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

Class Diagram - Concerns

- Ensure that the classes are both necessary and sufficient to solve the underlying problem
  - no missing attributes or methods in each class
  - no extra or unused attributes or methods in each class
  - no missing or extra classes

Parts of Speech Analysis

- A common or improper noun implies a class of objects
- A proper noun implies an instance of a class
- A collective noun implies a class of objects made up of groups of instances of another class

Parts of Speech Analysis

- An adjective implies an attribute of an object
- A doing verb implies an operation
- A being verb implies a relationship between an object and its class
- A having verb implies an aggregation or association relationship

Parts of Speech Analysis

- A transitive verb implies an operation
- An intransitive verb implies an exception
- A predicate or descriptive verb phrase implies an operation
- An adverb implies an attribute of a relationship or an operation

Discarding Unnecessary and Incorrect Classes

- Redundant Classes: Some potential classes differ only in name.
- Irrelevant Classes: Classes that have nothing to do with the system. Example: computer connection
- Vague Classes: Classes whose meaning is not clear at all. Examples: system and software

Discarding Unnecessary and Incorrect Classes

- Attributes: Some nouns in the list above are likely to be modeled as attributes rather as classes. Examples: author, title
- Operations: Some nouns are likely to be operations rather than classes. Example: book search.
- Roles: Some nouns are roles of objects involved in associations rather than classes.
- Implementation Constructs: Anything that is not part of the real-world problem.

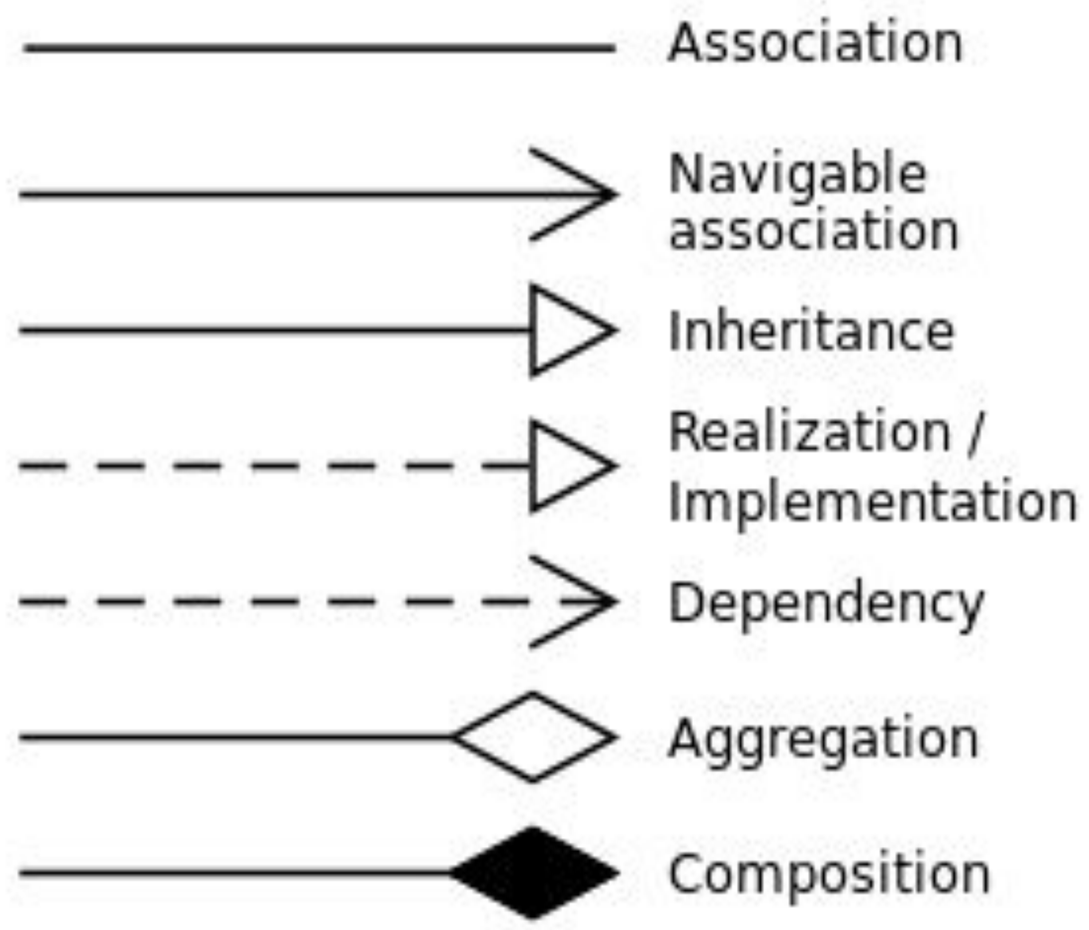
Visibility

To specify the visibility of a class member (i.e. any attribute or method), these notations must be placed before the members' name:

+	Public
-	Private
#	Protected
~	Package

A derived property is a property whose value (or values) is produced or computed from other information, for example, by using values of other properties. A derived property is shown with its name preceded by a forward slash '/'.

Relationships



Relationships

Dependency

Dependency shows that one class depends on another. Change in one class will create change in another class. For example, an employee is dependent on the organization.

Association

Association shows a static relationship between two entities. The association between a student and school is studies. An association can be named, and the ends of an association can be adorned with role names, aggregation indicators, multiplicity, visibility, navigability and other properties.

Relationships

Aggregation

Aggregation is a variant of the "has a" association relationship; aggregation is more specific than association. It is an association that represents a part-whole or part-of relationship.

Composition

The composite aggregation (colloquially called composition) relationship is a stronger form of aggregation where the aggregate controls the lifecycle of the elements it aggregates.

Relationships

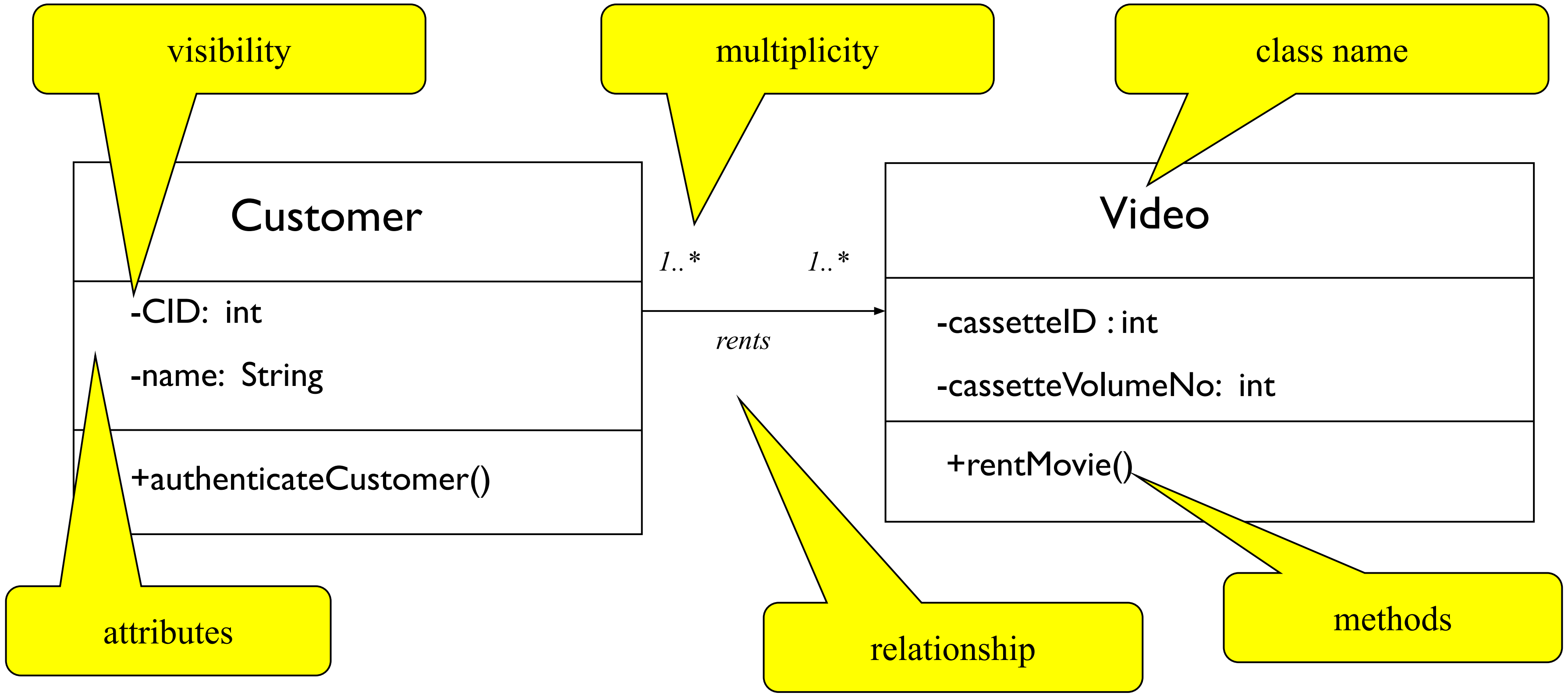
Generalization/Inheritance

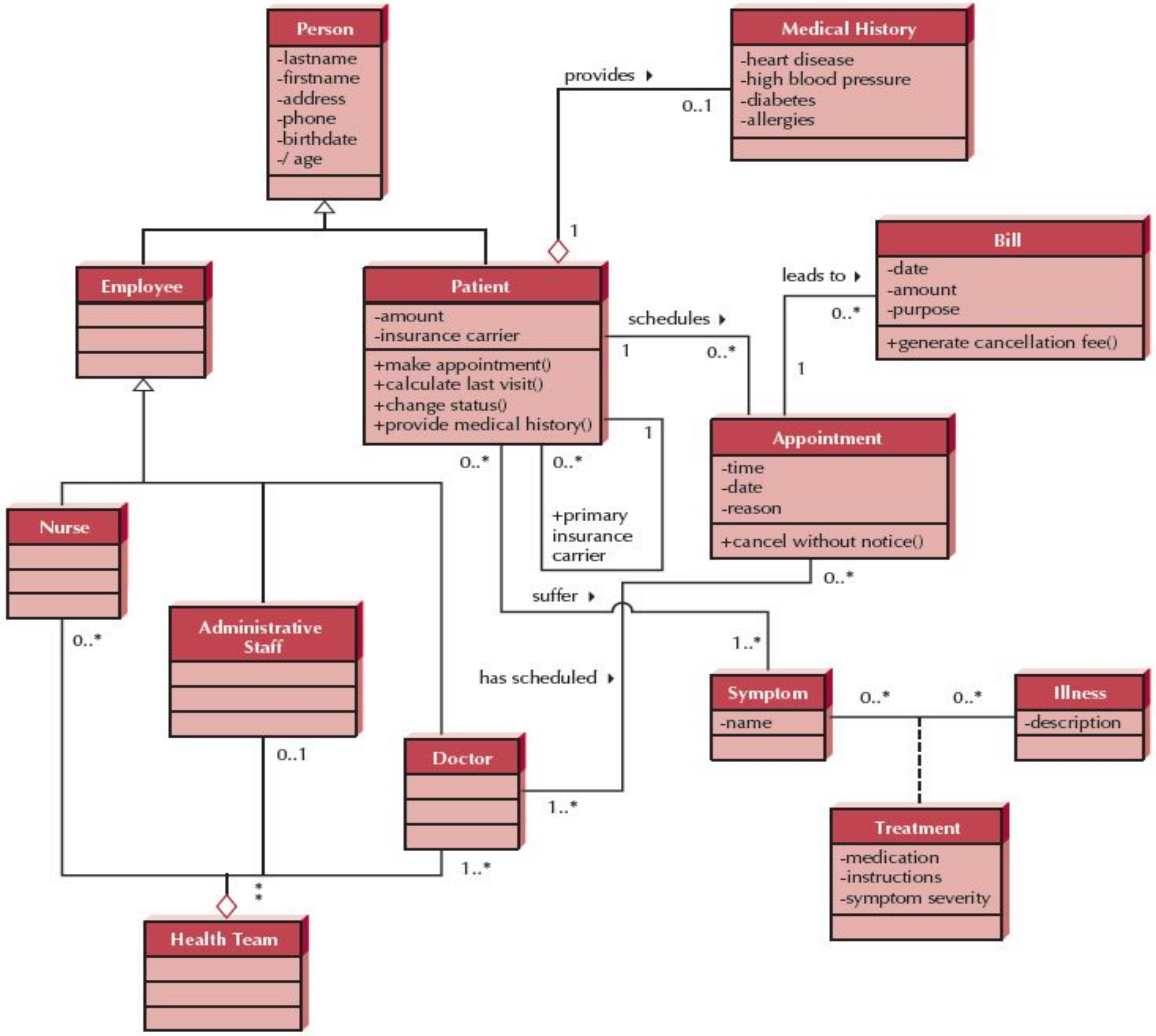
It indicates that one of the two related classes (the subclass) is considered to be a specialized form of the other (the super type) and the superclass is considered a Generalization of the subclass.

Realization/Implementation

In UML modelling, a realization relationship is a relationship between two model elements, in which one model element (the client) realizes (implements or executes) the behavior that the other model element (the supplier) specifies.

Example of a Class Diagram




Person
-lastname
-firstname
-address
-phone
-birthdate
-/ age
Medical History
-heart disease
-high blood pressure
-diabetes
-allergies
provides
0..1
1
Employee
Patient
-amount
-insurance carrier
+make appointment()
+calculate last visit()
+change status()
+provide medical history()
schedules
1
0..*
Bill
-date
-amount
-purpose
+generate cancellation fee()
leads to
0..*
1
Appointment
-time
-date
-reason
+cancel without notice()
1
0..*
0..*
+primary insurance carrier
0..*
Nurse
Administrative Staff
Doctor
suffer
1..*
has scheduled
0..*
Symptom
-name
0..*
0..*
Illness
-description
Treatment
-medication
-instructions
-symptom severity
0..1
1..*
1..*
*
*
Health Team

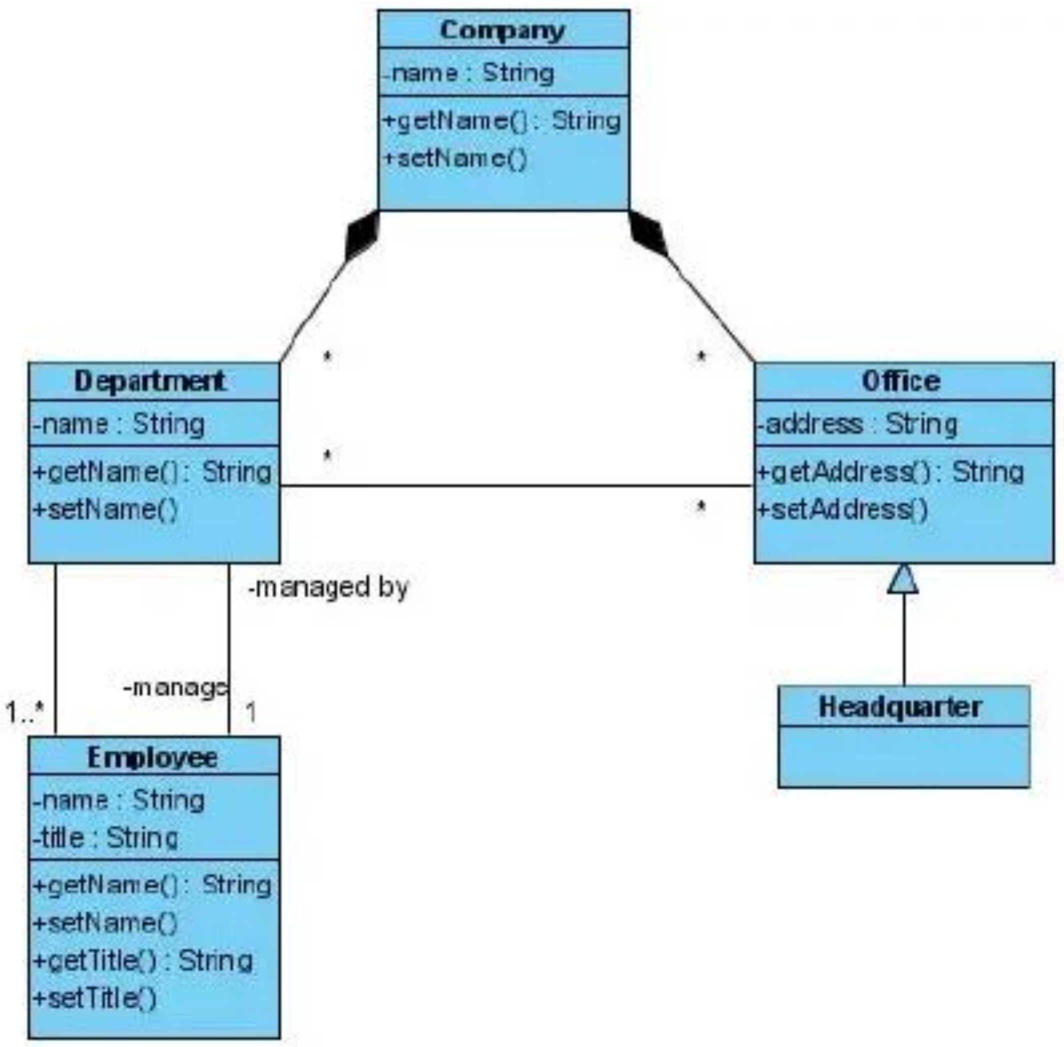
Example - Scenario

A company consists of departments. Departments are located in one or more offices. One office acts as a headquarter. Each department has a manager who is recruited from the set of employees. Your task is to model the system for the company.

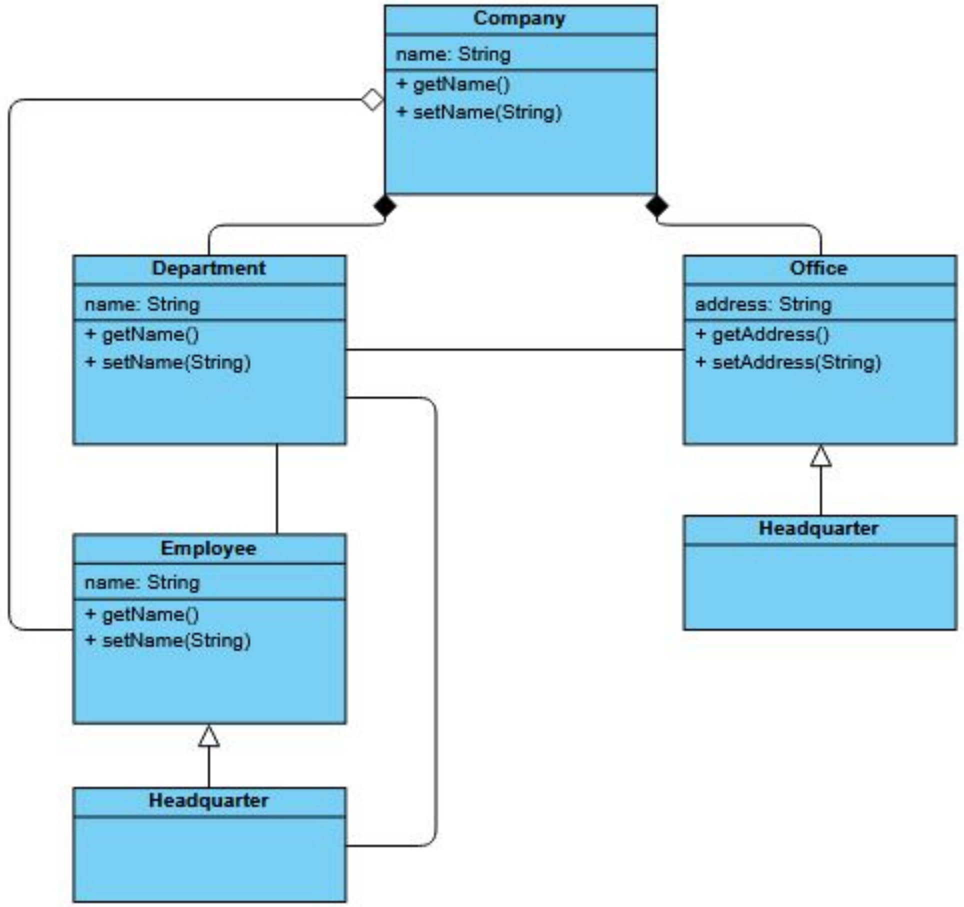
Example - Scenario

A company consists of departments. Departments are located in one or more offices. One office acts as a headquarter. Each department has a manager who is recruited from the set of employees. Your task is to model the system for the company.

Example - Scenario



Example - Scenario



Thank you