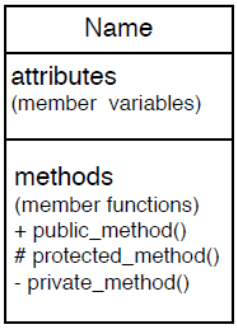
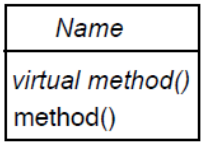
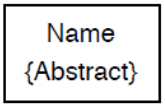
Basic UML Class Diagram Notation

Class

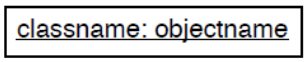


Abstract class

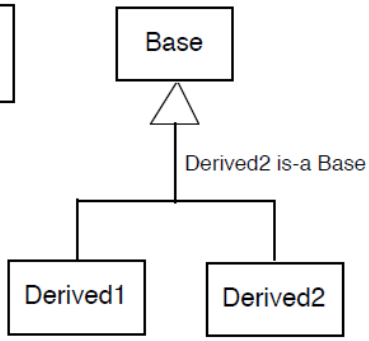




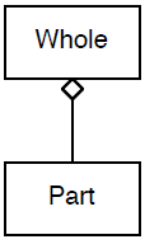
Object



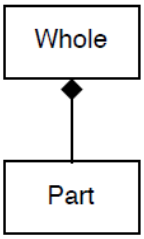
Inheritance (is-a) relationship



Aggregation and Composition (has-a) relationship

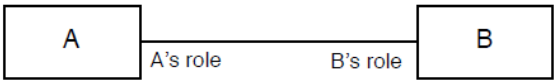


Whole has Part as a part;
lifetimes might be different;
Part might be shared with other Wholes.
(aggregation)

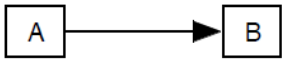


Whole has Part as a part;
lifetime of Part controlled by Whole,
Part objects are contained in one Whole object.
(composition)

Association (uses, interacts-with) relationship

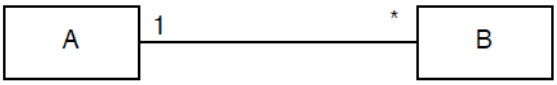


Navigability - can reach B starting from A



Multiplicity in Aggregation, Composition, or Association

* - any number	0..1 - zero or one	Follow line from start class to end class,
1 - exactly 1	1..* - 1 or more	note the multiplicity at the end.
n - exactly *n*	*n .. m* - *n* through *m*	Say "Each <start> is associated with <multiplicity> <ends>"



Each A is associated with any number of B's.
Each B is associated with exactly one A.

By Nate Higgers

Extends

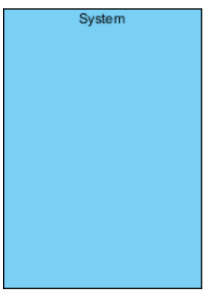
- Indicates that an **"Invalid Password"** use case may include (subject to specified in the extension) the behavior specified by base use case **"Login Account"**.
- Depict with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow.
- The stereotype "<<extends>>" identifies as an extend relationship

Generalization

- A generalization relationship is a parent-child relationship between use cases.
- The child use case is an enhancement of the parent use case.
- Generalization is shown as a directed arrow with a triangle arrowhead.
- The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.

Boundary of system

- The system boundary is potentially the entire system as defined in the requirements document.
- For large and complex systems, each module may be the system boundary.
- For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc.
- can form a system boundary for use cases specific to each of these business functions.
- The entire system can span all of these modules depicting the overall system boundary



Actor

- Someone interacts with use case (system function).
- Named by noun.
- Actor plays a role in the business
- Similar to the concept of user, but a user can play different roles
- For example:
  - A prof. can be instructor and also researcher
  - plays 2 roles with two systems
- Actor triggers use case(s).
- Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).

Use Case

- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
- i.e. Do something
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

Include

- When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.
- A use case includes the functionality described in another use case as a part of its business process flow.
- A uses relationship from base use case to child use case indicates that an instance of the base use case will include the behavior as specified in the child use case.
- An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.
- The stereotype "<<include>>" identifies the relationship as an include relationship.

Communication Link

- The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
- Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.

FUNCTIONAL vs NONFUNCTIONAL REQUIREMENTS

	Functional requirements	Nonfunctional requirements
Objective	Describe what the product does	Describe how the product works
End result	Define product features	Define product properties
Focus	Focus on user requirements	Focus on user expectations
Essentiality	They are mandatory	They are not mandatory but desirable
Origin type	Usually defined by the user	Usually defined by developers or other tech experts
Testing	Component, API, UI testing, etc. Tested before nonfunctional testing	Performance, usability, security testing, etc. Tested after functional testing
Types	Authentication, authorization levels, data processing, reporting, etc.	Usability, reliability, scalability, performance, etc.

Constraint Requirements: These outline any limitations or restrictions that must be adhered to during the development or operation of the system. They could include regulatory constraints, budget constraints, or technological limitations.