**UML Quick Guide**

Contents

[Use Case Diagram: 2](#_Toc323537745)

[Class Diagram 2](#_Toc323537746)

[Composition & Aggregation 4](#_Toc323537747)

[Aggregation: 4](#_Toc323537748)

[Composition: 4](#_Toc323537749)

[Dependency 5](#_Toc323537750)

[Multiplicity 6](#_Toc323537751)

[Class Notaion: 3 pieces of info – name, fields, methods 7](#_Toc323537752)

[Object Diagram: 8](#_Toc323537753)

[Interaction Diagram 9](#_Toc323537754)

[Sequence Diagram 9](#_Toc323537755)

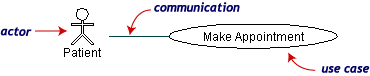
[Activity Diagram: 10](#_Toc323537756)

[Deployment Diagram 11](#_Toc323537757)

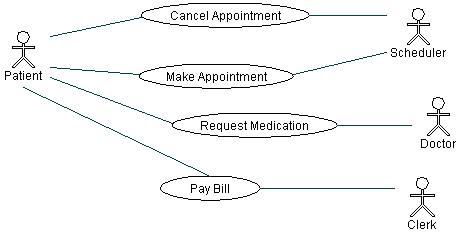
[State Diagram 12](#_Toc323537758)

# Use Case Diagram:

Emphasis on WHAT a system does rather than HOW.



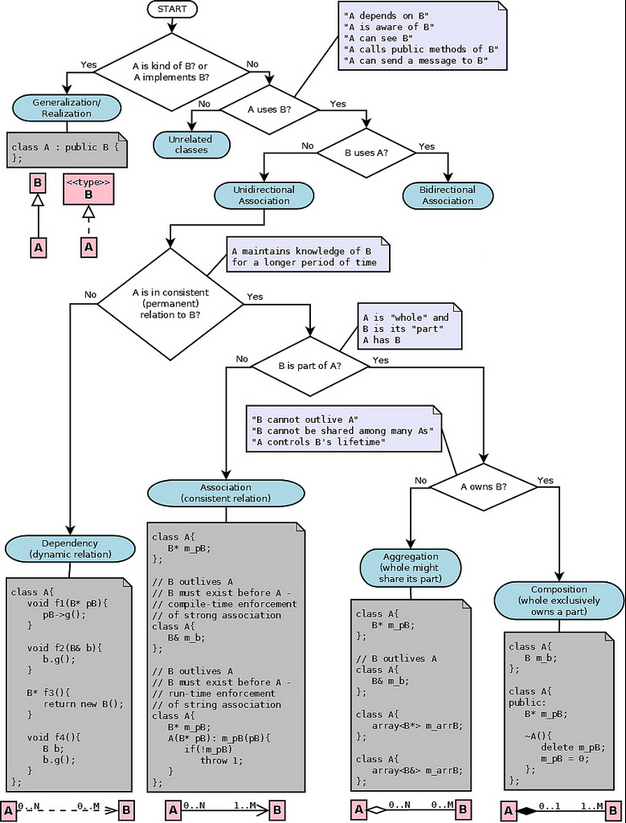
***A use case is a summary of a single task. An actor is who or what initiates the task.***

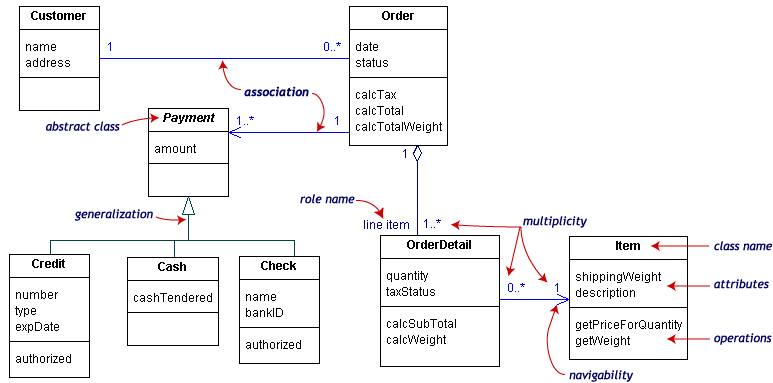


A use case diagram is useful in 1. determining requirements 2. communicating with clients 3. generating test case

# Class Diagram

UML Class Diagrams display relationships between classes in a given model. Here is a list of various types of class relationships and statements that can describe them:

* **generalization** ("A is *a kind of* B")
* **realization** ("A*implements* B")
* **dependency** ("A *uses* B and *forgets* about it", "A can send message to B")
* **association** ("A *uses* B and *keeps* it", "A *has* B")
* **aggregation** ("B can be a *part* of A (*whole*)", "B can outlive A", "B can be *shared* between many As")
* **composition** ("B is a *part* of A (*whole*)", "A *owns* B", "B cannot outlive A", "B belongs only to one A")
* 



Relationships between classes:

* + association: straight line. No **navigability arrows** means bi-directional. 1 arrorw means one-way only.
  + generalization: a **triangle** pointing to the **base class**
  + aggregation: an association in which one class belongs to a collection. Has a **diamond end** which pointing to the **container class**.

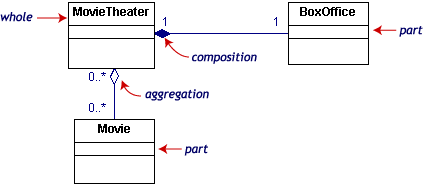
# Composition & Aggregation

## Aggregation:

is a whole/part relationship.

## Composition:

A **strong** form of **Aggregation**! Item(s) can belong to **ONLY 1 container.** Destroy the container, the item(s) cease to exist! Strong in the sense of it indicates the “**lifetime**” of the **part** is **dependent** upon the **whole**.



Aggregation denotes whole/part relationships whereas associations do not. However, there

is not likely to be much difference in the way that the two relationships are implemented. That is,

it would be very difﬁcult to look at the code and determine whether a particular relationship ought

to be aggregation or association.

## Dependency

Sometimes the relationship between a two classes is very weak. They are not implemented with member variables at all. Rather they might be implemented as member function arguments. Consider, for example, the Draw function of the Shape class. Suppose that this function takes an argument of type DrawingContext.

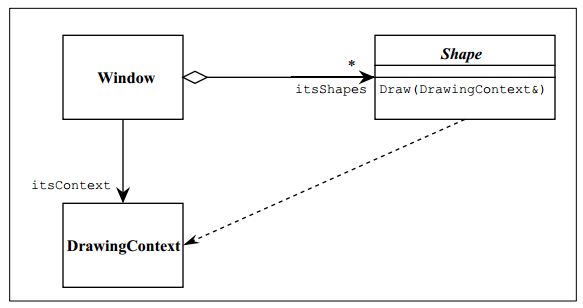
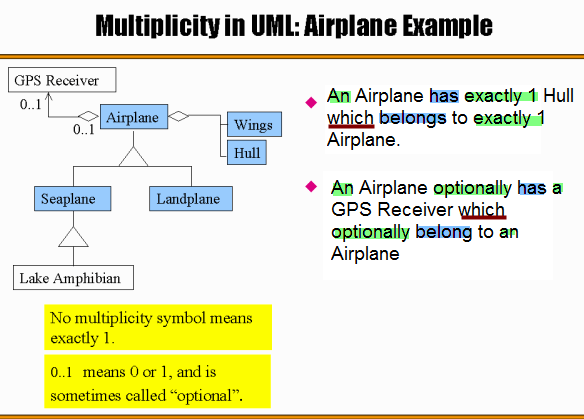
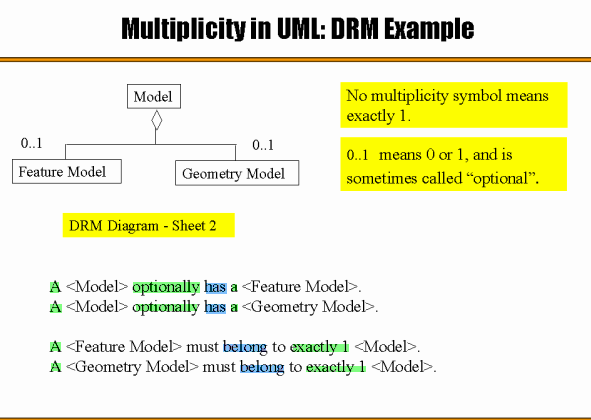
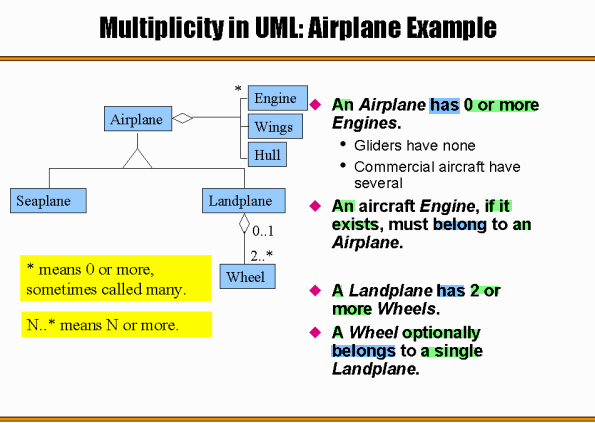


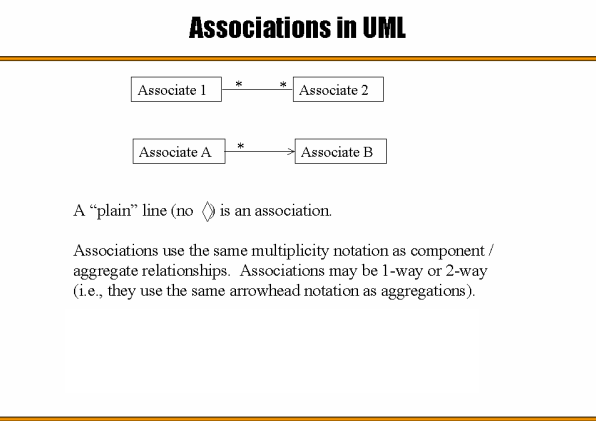
Figure shows a dashed arrow between the Shape class and the DrawingContext class. This is the dependency relationship. In Booch94 this was called a ‘using’ relationship. This relationship simply means that Shape somehow depends upon DrawingContext.

# Multiplicity

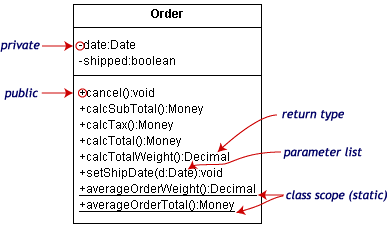








# Class Notaion: 3 pieces of info – name, fields, methods

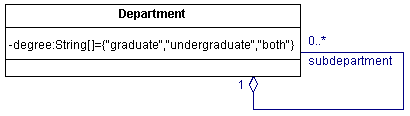


* method: <access modifier(**+**:*public*, #:*protected*, -:*private*)><name>(<param\_name>:<param\_type>, …):<return type>

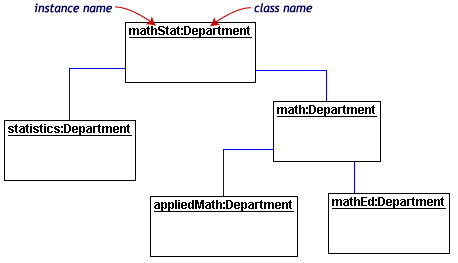
# Object Diagram:

* show instances of class. Useful for explaining **small pieces** with **complicated** relationships, especially **recursive relationships**.

*e.g. this small* ***class diagram*** *shows a university department can contain 0 to many sub-departments. But it doesn’t tell the* ***names*** *of the sub-departments or the* ***relationship*** *between the sub-departments*

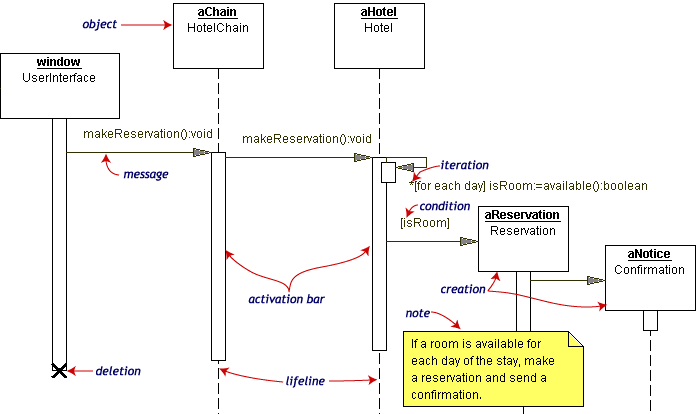


The following **object diagram instantiates the class diagram** and shows the instances(**names**) of sub-departments and **relationships** between the sub-department.

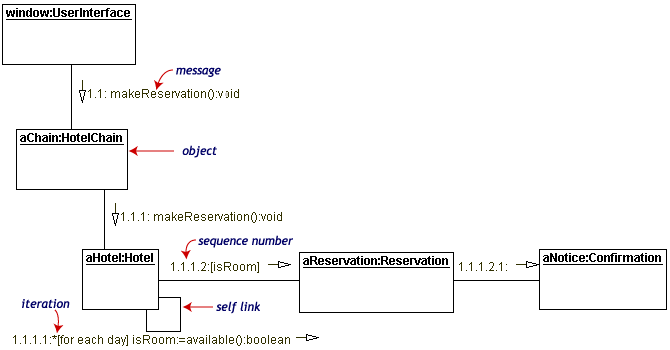


# Interaction Diagram

## Sequence Diagram



* + **Collaboration Diagram:** communicate the same info as the sequence diagram but focus on object roles

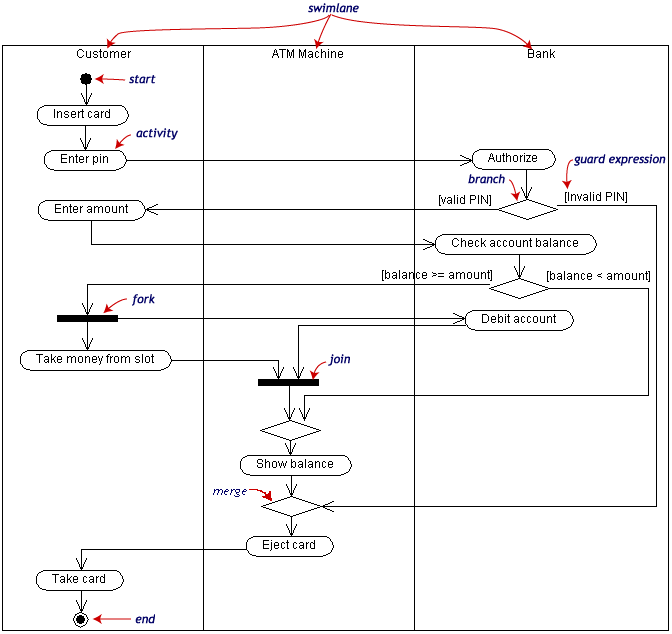


Each message has a sequence number. The top-level message is numbered 1. Message at the same level have the same prefix but suffixes with 1, 2, etc… according to when they occur.

# Activity Diagram:

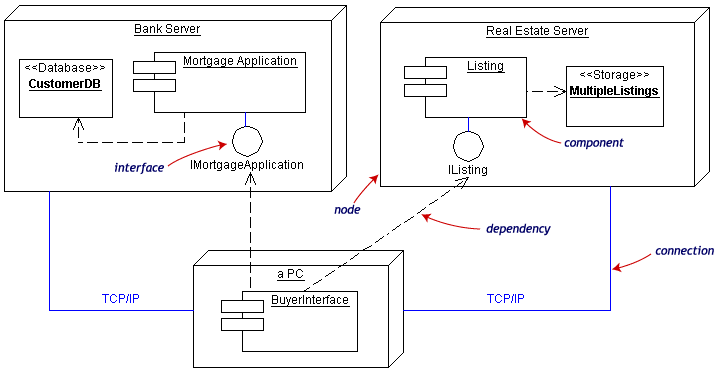
a fancy flowchart. Focus on the flow of activities involved in a process

* + **Activities** are round rectangles.
  + A single **transition** comes out of each activity.
  + A transition may **branch** (represented by a hollow **diamond**) into 2+ **mutually exclusive** transitions.
  + **Guard expressions** (inside **[]**) label the transiction **coming out** of the branch
  + A transition may **fork**(solid **bar**) into 2+ parallel transitions. 2+ parallel transitions may **join**(solid **bar**) into a transition.



# Deployment Diagram

* + - **Component** are **physical** analogs of **class** diagram. Deployment Diagram shows the **physical relationship of software and hardware**.
    - **Component**: rectangles with 2 tabs on upper left.
    - **Nodes** represent physical **hardware**.
    - **Each component belongs to 1 and only 1 node**.

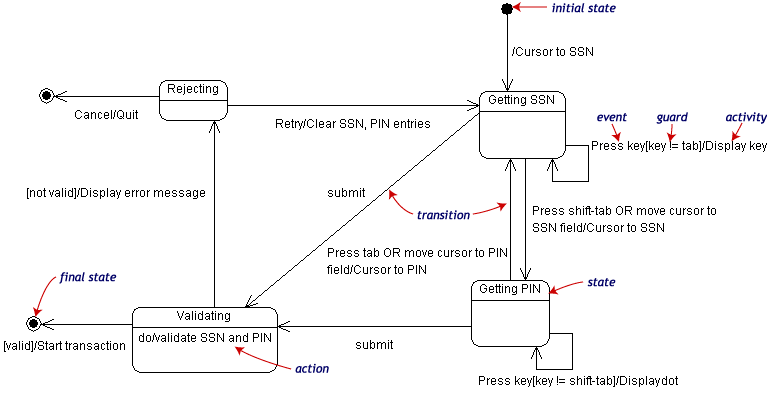


# State Diagram

Objects have behaviors and state. The state of an object depends on its current activity or condition. A statechart diagram shows the possible states of the object and the transitions that cause a change in state.

Our example diagram models the login part of an online banking system. Logging in consists of entering a valid social security number and personal id number, then submitting the information for validation.

Logging in can be factored into four non-overlapping states: Getting SSN, Getting PIN, Validating, and Rejecting. From each state comes a complete set of transitions that determine the subsequent state.



States are rounded rectangles. Transitions are arrows from one state to another. Events or conditions that trigger transitions are written beside the arrows. Our diagram has two self-transition, one on Getting SSN and another on Getting PIN.

The initial state (black circle) is a dummy to start the action. Final states are also dummy states that terminate the action.

The action that occurs as a result of an event or condition is expressed in the form /action. While in its Validating state, the object does not wait for an outside event to trigger a transition. Instead, it performs an activity. The result of that activity determines its subsequent state.