Introduction to UML

Acknowledgment: These slides are adopted with some minor mofidifications from a presentation by Majid Ali Khan from University of Central Florida.

Acknowledgements

- Slides material are taken from different sources including:
  - Prashanth Aedunuthula UML presentation, Fall 2004
  - Lecture slides from Software Engineering course at UC Berkeley (Professor Necula – Fall 2004)
  - Lecture slides from a course on web at:
    - www.sts.tu-harburg.de/ teaching/ws-98.99/OOA+D/3-0-UML.pdf

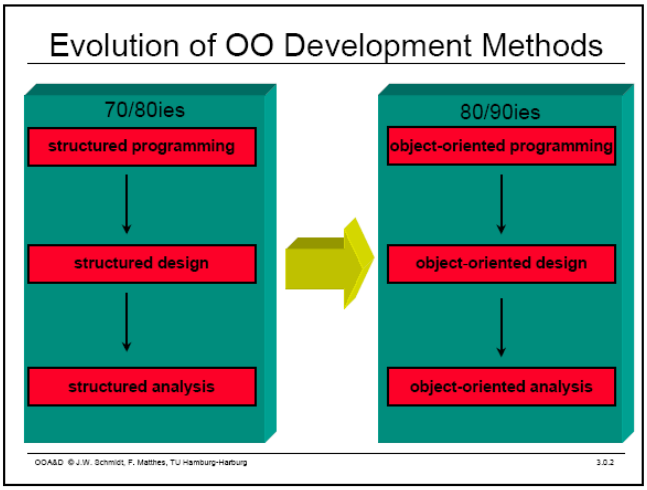
Overview

- What is Modeling?
- What is UML?
- A brief history of UML
- Understanding the basics of UML
- UML diagrams
- UML Modeling tools

Modeling

- Describing a system at a high level of abstraction
  - A model of the system
  - Used for requirements and specifications
- Is it necessary to model software systems?

Object Oriented Modeling



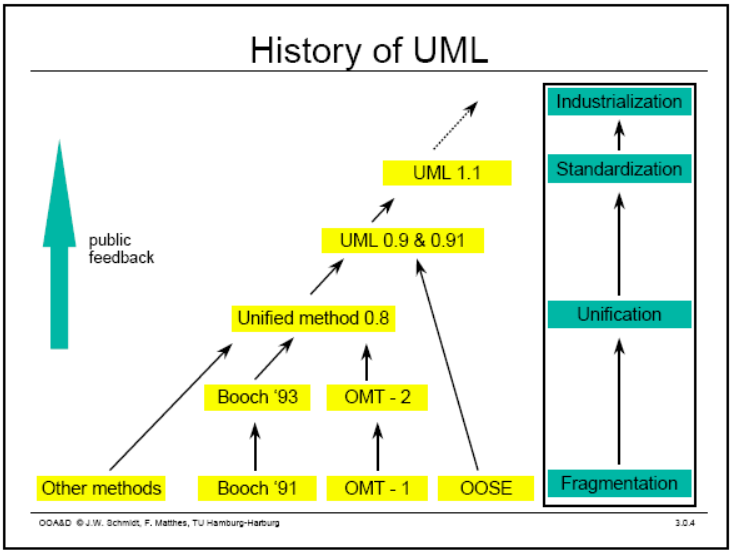
What is UML?

- UML stands for “Unified Modeling Language”
- It is a industry-standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- The UML uses mostly graphical notations to express the OO analysis and design of software projects.
- Simplifies the complex process of software design

Why UML for Modeling

- Use graphical notation to communicate more clearly than natural language (imprecise) and code(too detailed).
- Help acquire an overall view of a system.
- UML is *not* dependent on any one language or technology.
- UML moves us from fragmentation to standardization*.*

History of UML



Types of UML Diagrams

- **Use Case Diagram**
- **Class Diagram**
- **Sequence Diagram**
- **Collaboration Diagram**
- **State Diagram**

This is only a subset of diagrams … but are most widely used

Use Case Diagram

- Used for describing a set of user **scenarios**
- Mainly used for capturing user requirements
- Work like a **contract** between end user and software developers

Use Case Diagram (core components)

Actors: A role that a user plays with respect to the system,including human users and other systems. e.g.,inanimate physical objects (e.g. robot); an external system that needs some information from the current system.

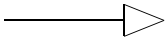
Use case: A set of scenarios that describing an interaction between a user and a system, including alternatives.

System boundary: rectangle diagram representing the boundary between the actors and the system.

Use Case Diagram(core relationship)

Association: communication between an actor and a use case; Represented by a solid line.

Generalization: relationship between one general use case and a special use case (used for defining special alternatives)
Represented by a line with a triangular arrow head toward the parent use case.



Use Case Diagram(core relationship)

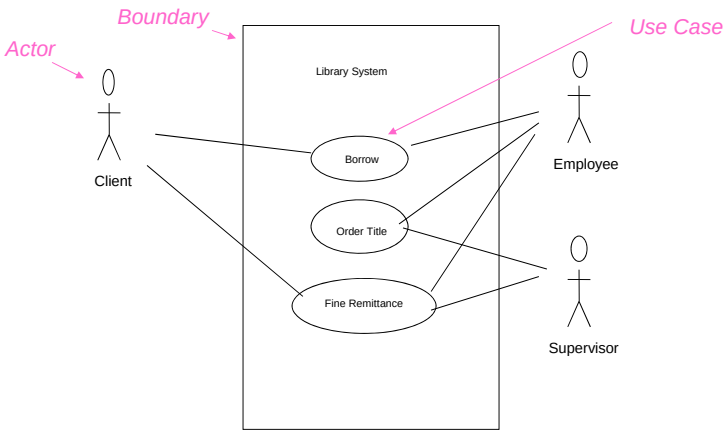
Include: a dotted line labeled <<include>> beginning at base use case and ending with an arrows pointing to the include use case. The include relationship occurs when a chunk of behavior is similar across more than one use case. Use "include" in stead of copying the description of that behavior.

<<include>>
----------►

Extend: a dotted line labeled <<extend>> with an arrow toward the base case. The extending use case may add behavior to the base use case. The base class declares "extension points".

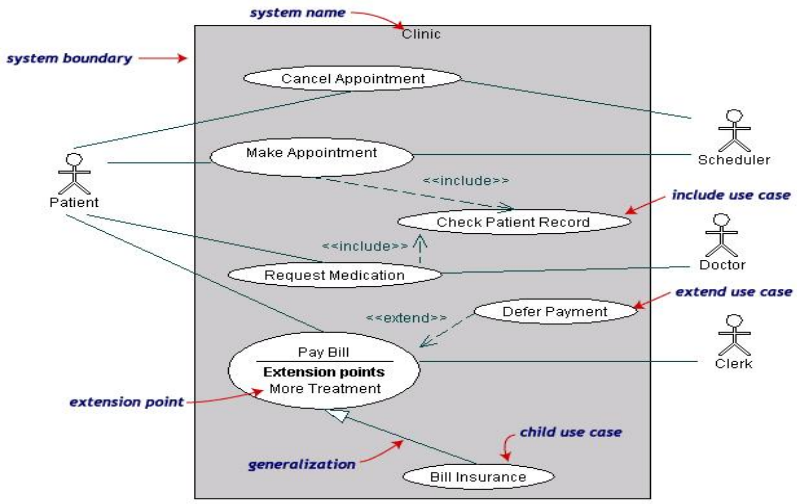
<<extend>>
----------►

Use Case Diagrams



- A generalized description of how a system will be used.
- Provides an overview of the intended functionality of the system

Use Case Diagrams(cont.)



(TogetherSoft, Inc)

Use Case Diagrams(cont.)

•**Pay Bill** is a parent use case and **Bill Insurance** is the child use case. (generalization)

•Both **Make Appointment** and **Request Medication** include **Check Patient Record** as a subtask.(include)

•The **extension point** is written inside the base case **Pay bill**; the extending class **Defer payment** adds the behavior of this extension point. (extend)

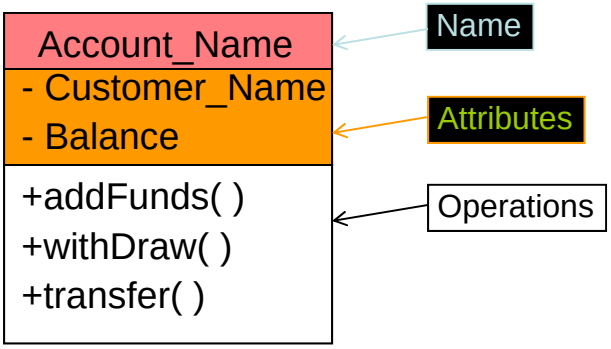
Class diagram

- Used for describing structure and behavior in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used for developers

Class representation

- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations
- Modifiers are used to indicate visibility of attributes and operations.
  - '+' is used to denote *Public* visibility (everyone)
  - '#' is used to denote *Protected* visibility (friends and derived)
  - '-' is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.

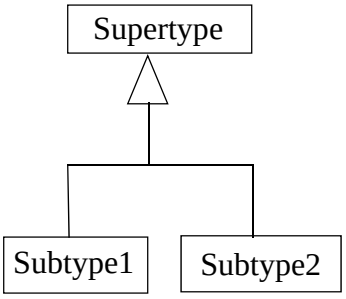
An example of Class



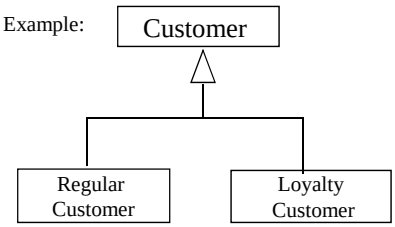
OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)
- Associations can be further classified as
  - Aggregation
  - Composition

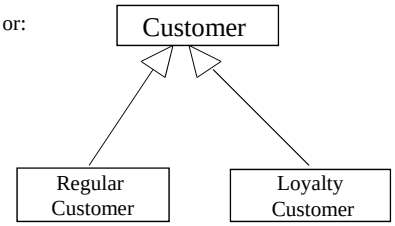
OO Relationships: **Generalization**



Example:



or:



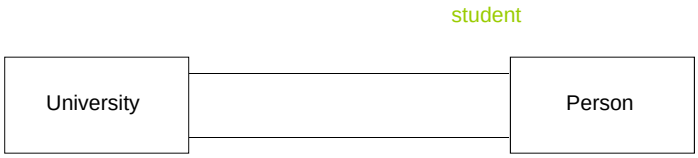
- Generalization expresses a parent/child relationship among related classes.

- Used for abstracting details in several layers

OO Relationships: **Association**

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.
- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)
  - Navigability (unidirectional, bidirectional)

Association: Multiplicity and Roles



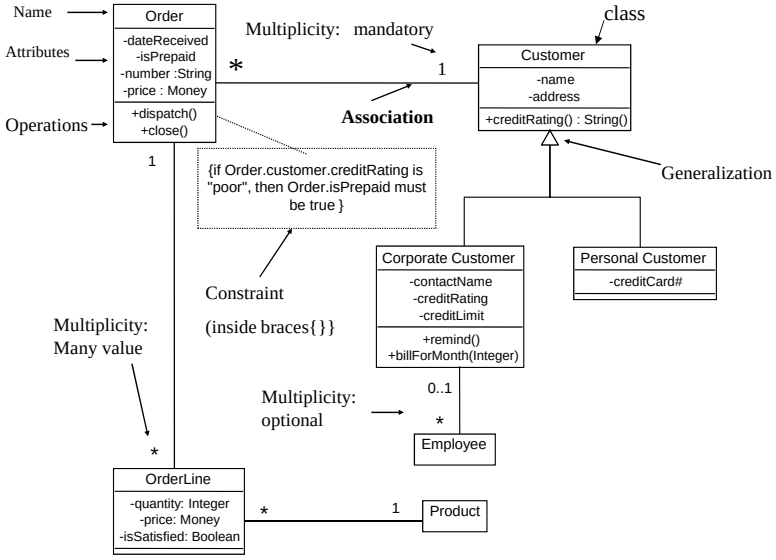
Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

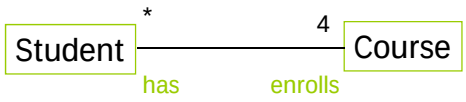
"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

Class Diagram



[from *UML Distilled Third Edition*]

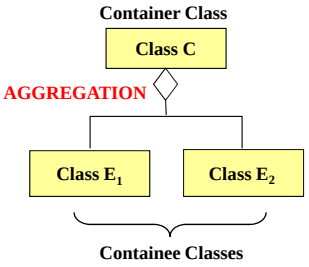
Association: Model to Implementation




```
Class Student {
   Course enrolls[4];
}

Class Course {
   Student have[];
}
```

OO Relationships: **Aggregation**

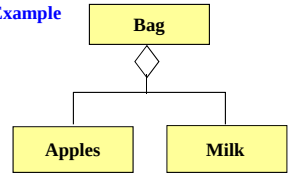


Aggregation: expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

It expresses a relationship where an instance of the Container-class has the responsibility to hold and maintain instances of each Containee-class that have been created outside the auspices of the Container-class.

Aggregation should be used to express a more informal relationship than composition expresses. That is, it is an appropriate relationship where the Container and its Containees

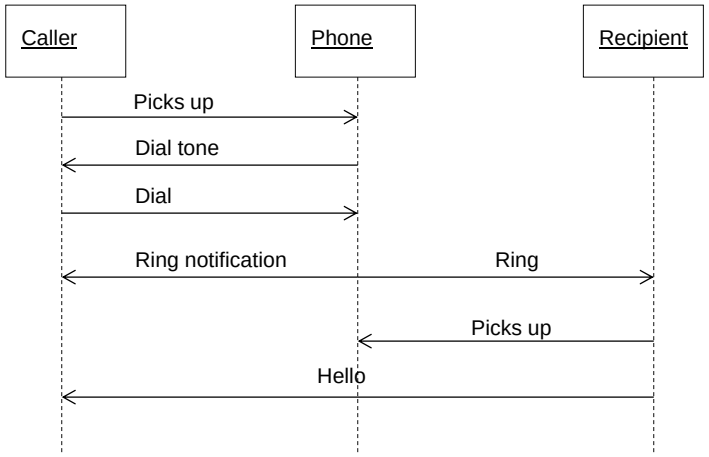
Example



Aggregation is appropriate when Container and Containees have no special access privileges to each other.

[From Dr.David A. Workman]

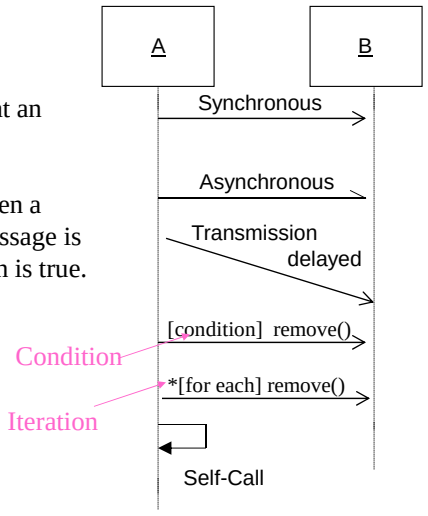
Sequence Diagram(make a phone call)



Sequence Diagram:Object interaction

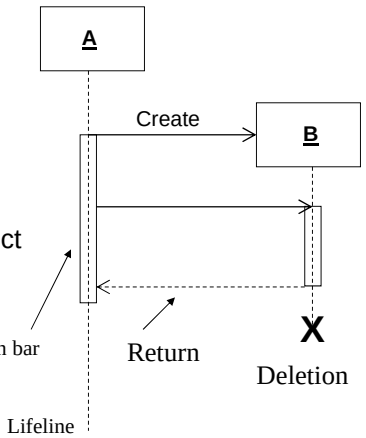
Self-Call: A message that an Object sends to itself.

Condition: indicates when a message is sent. The message is sent only if the condition is true.

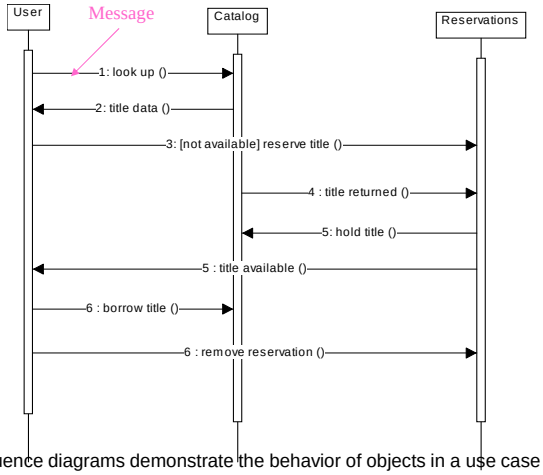


Sequence Diagrams – Object Life Spans

- Creation
  - Create message
  - Object life starts at that point
- Activation
  - Symbolized by rectangular stripes
  - Place on the lifeline where object is activated.
  - Rectangle also denotes when object is deactivated.
- Deletion
  - Placing an 'X' on lifeline
  - Object's life ends at that point



Sequence Diagram



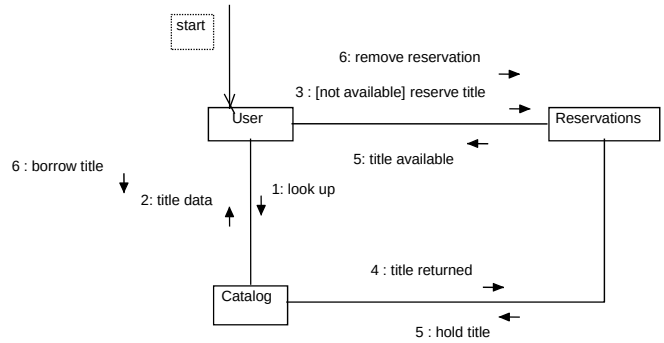
•Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass.

•The horizontal dimension shows the objects participating in the interaction.

•The vertical arrangement of messages indicates their order.

•The labels may contain the seq. # to indicate concurrency.

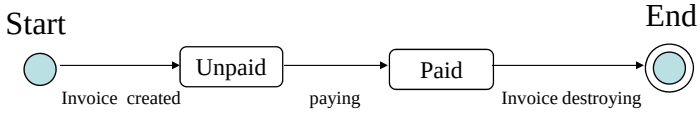
Interaction Diagrams: Collaboration diagrams



- Shows the relationship between objects and the order of messages passed between them. between them.
- The objects are listed as rectangles and arrows indicate the messages being passed
- The numbers next to the messages are called sequence numbers. They show the sequence of the messages as they are passed between the objects.
- convey the same information as sequence diagrams, but focus on object roles instead of the time sequence.

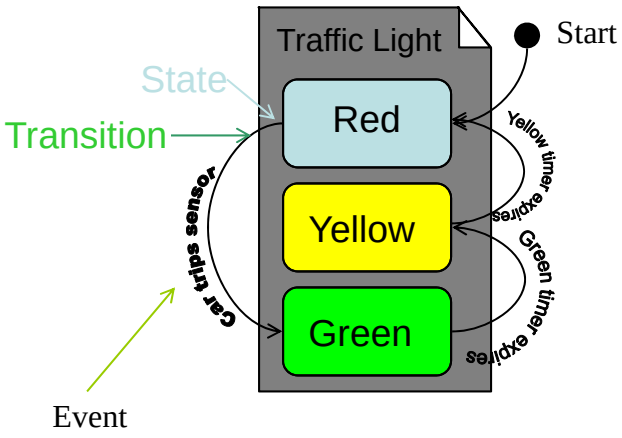
State Diagrams
(Billing Example)

State Diagrams show the sequences of states an object goes through during its life cycle in response to stimuli, together with its responses and actions; an abstraction of all possible behaviors.



State Diagrams

(Traffic light example)



UML Modeling Tools

- Rational Rose (www.rational.com) by IBM
- TogetherSoft Control Center, Borland (http://www.borland.com/together/index.html)
- **ArgoUML** (free software) (http://argouml.tigris.org/)
  OpenSource; written in java
- Others (http://www.objectsbydesign.com/tools/umltools_byCompany.html)

Reference

1. **UML Distilled:** A Brief Guide to the Standard Object Modeling Language
   Martin Fowler, Kendall Scott

2. IBM Rational
   http://www-306.ibm.com/software/rational/uml/

3. Practical UML --- A Hands-On Introduction for Developers
   http://www.togethersoft.com/services/practical_guides/umlonlinecourse/

4. Software Engineering Principles and Practice. Second Edition; Hans van Vliet.

5. http://www-inst.eecs.berkeley.edu/~cs169/