



Unified Modeling Language (UML)



Outline

- What is UML and why we use UML?
- How to use UML diagrams to design software system?
- What UML Modeling tools we use today?



Unified Modeling Language (UML) and Modeling

- UML is a graphical notation useful for OO analysis and design
- Allows representing various aspects of the system
- Various notations are used to build different models for the system
- OOAD methodologies use UML to represent the models they create



Modeling

- Modeling is used in many disciplines – architecture, aircraft building, ...
- A model is a simplification of reality
- “All models are wrong, some are useful”
- A good model includes those elements that have broad effect and omits minor elements
- A model of a system is not the system!



Why build models?

- Models help us visualize a system
- Help specify the system structure
- Gives us a template that can guide the construction
- Document the decisions taken and their rationale



Modeling

- Every complex system requires multiple models, representing different aspects
- These models are related but can be studied in isolation
- Eg. Architecture view, electrical view, plumbing view of a building
- Model can be structural, or behavioral



Why Model ?

- **Analyse the problem-domain**
 - simplify reality
 - capture requirements
 - visualize the system in its entirety
 - specify the structure and/or behaviour of the system
- **Design the solution**
 - document the solution - in terms of its structure, behaviour, etc.



What is UML and Why we use UML?

- **More description about UML:**
 - 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



What is UML and Why we use UML?

- Why we use UML?
 - Use graphical notation: 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.



What is UML ?

- UML - Unified Modeling language
- UML is a modeling language, not a methodology or process
- Fuses the concepts of the Booch, OMT, OOSE methods
- Developed by Grady Booch, James Rumbaugh and Ivar Jacobson at Rational Software.
- Accepted as a standard by the Object Management Group (OMG), in 1997.



More on UML...

UML is a modeling language for **visualising**, **specifying**, **constructing** and **documenting** the artifacts of software systems.



Visualising - *a picture is worth a thousand words; a graphical notation articulates and unambiguously communicates the overall view of the system (problem-domain).*

More on UML...

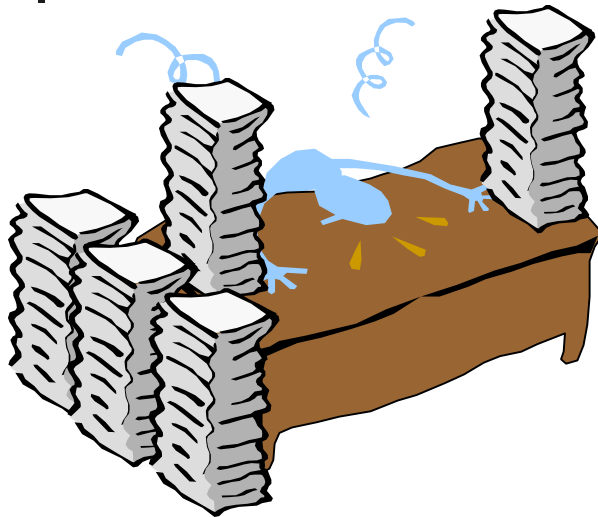


Specifying - *UML provides the means to model precisely, unambiguously and completely, the system in question.*

Constructing - *models built with UML have a "design" dimension to it; these are language independent and can be implemented in any programming language.*



More on UML...



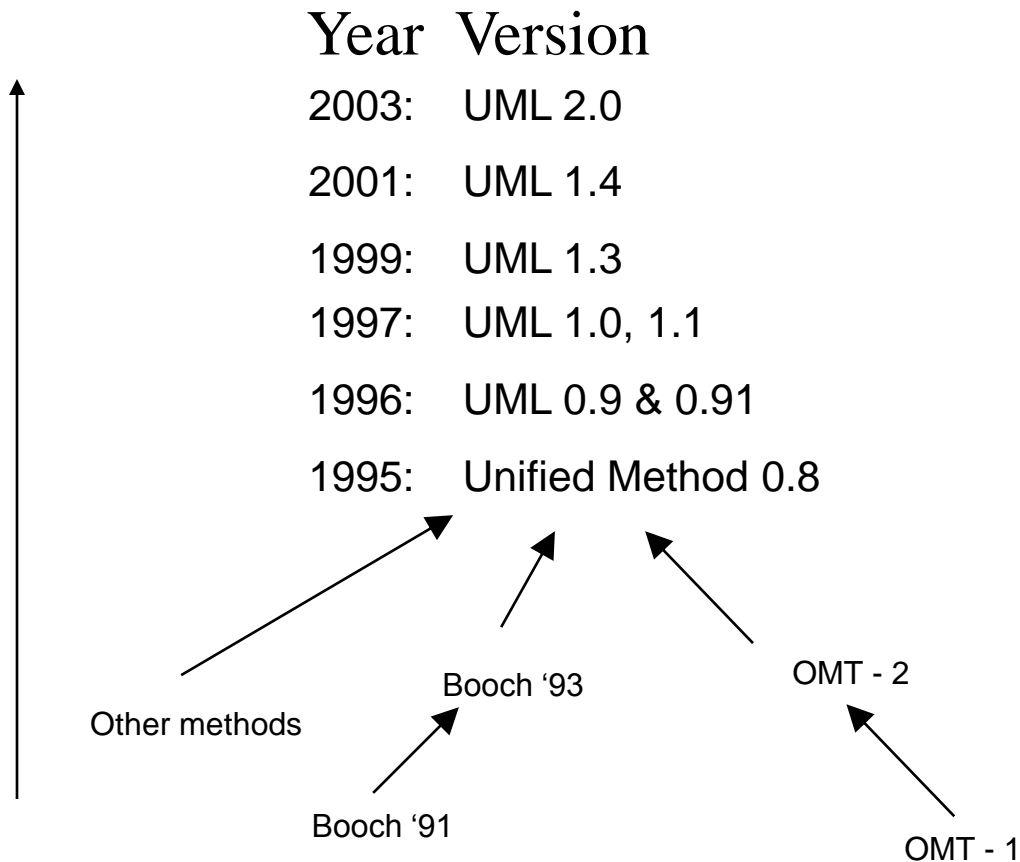
*Documentation is
(among others) for:*

- Requirements
- Design
- Tests

Documenting - *every software project involves a lot of documentation - from the inception phase to the deliverables.*

UML provides the notations for documenting some of these artifacts

What is UML and Why we use UML?





How to use UML diagrams to design software system?

■ Types of UML Diagrams:

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

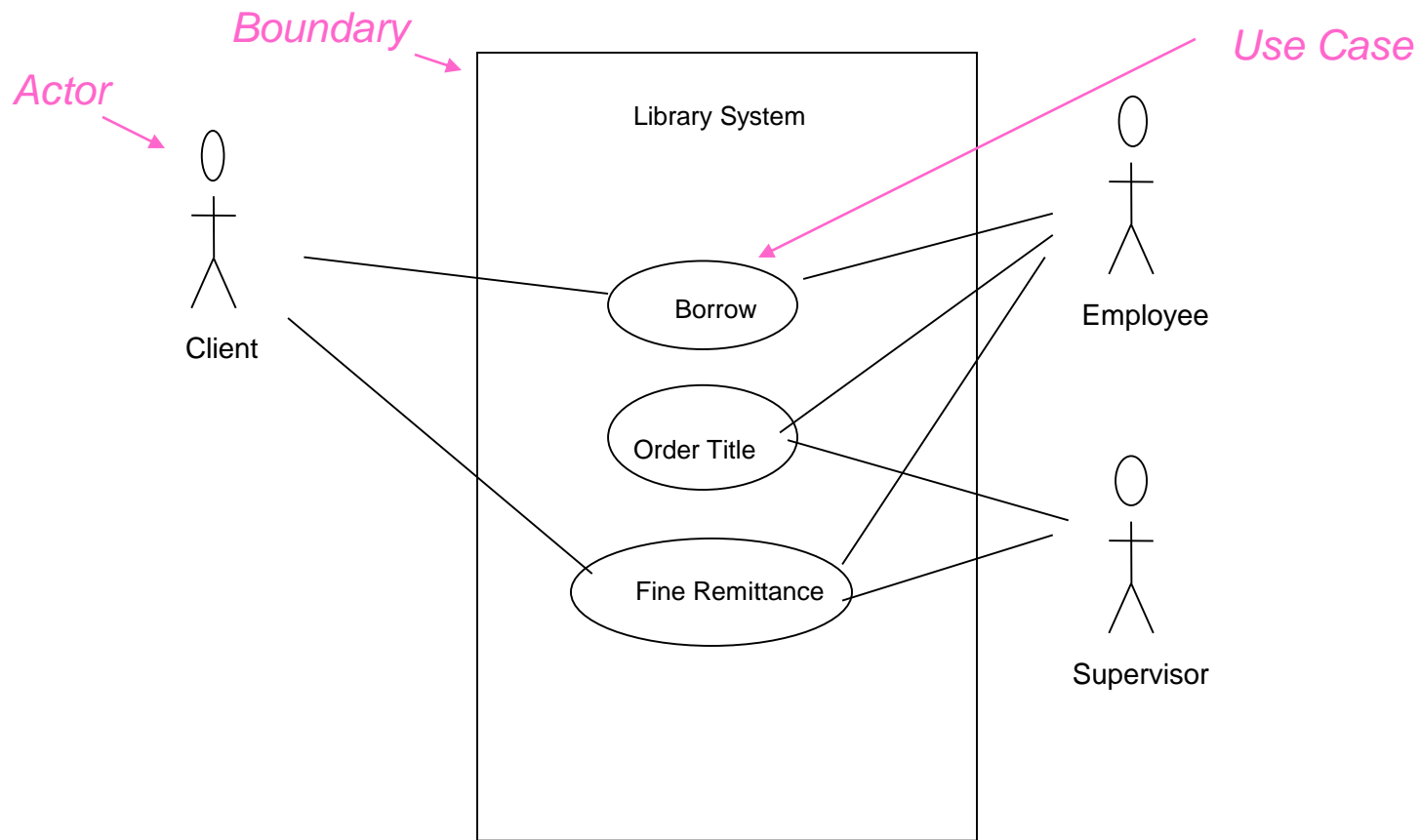
This is only a subset of diagrams ... but are most widely used



Use-Case Diagrams

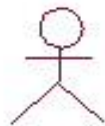
- A use-case diagram is a set of use cases
- A use case is a model of the interaction between
 - External users of a software product (actors) and
 - The software product itself
 - More precisely, an actor is a user playing a specific role
- describing a set of user **scenarios**
- capturing user requirements
- **contract** between end user and software developers

Use-Case Diagrams

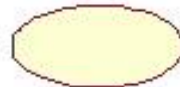


Use-Case Diagrams

- **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.



Actor



Use Case

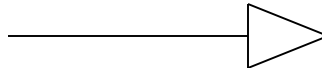


Use-Case Diagrams

- 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.

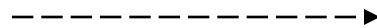




Use-Case Diagrams

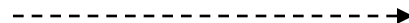
Include: a dotted line labeled <<include>> beginning at base use case and ending with an arrow pointing to the include use case. The include relationship occurs when a chunk of behavior is similar across more than one use case. Use “include” in stead of copying the description of that behavior.

<<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

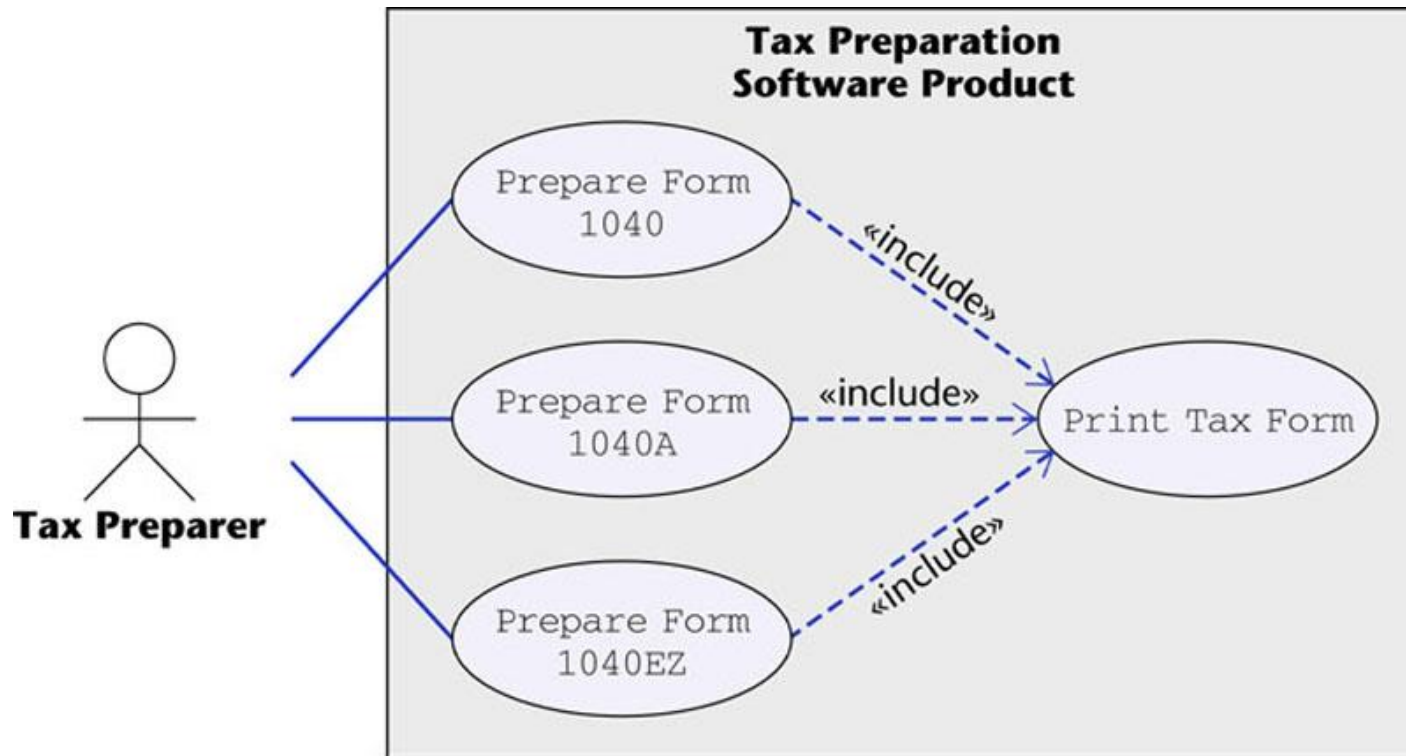
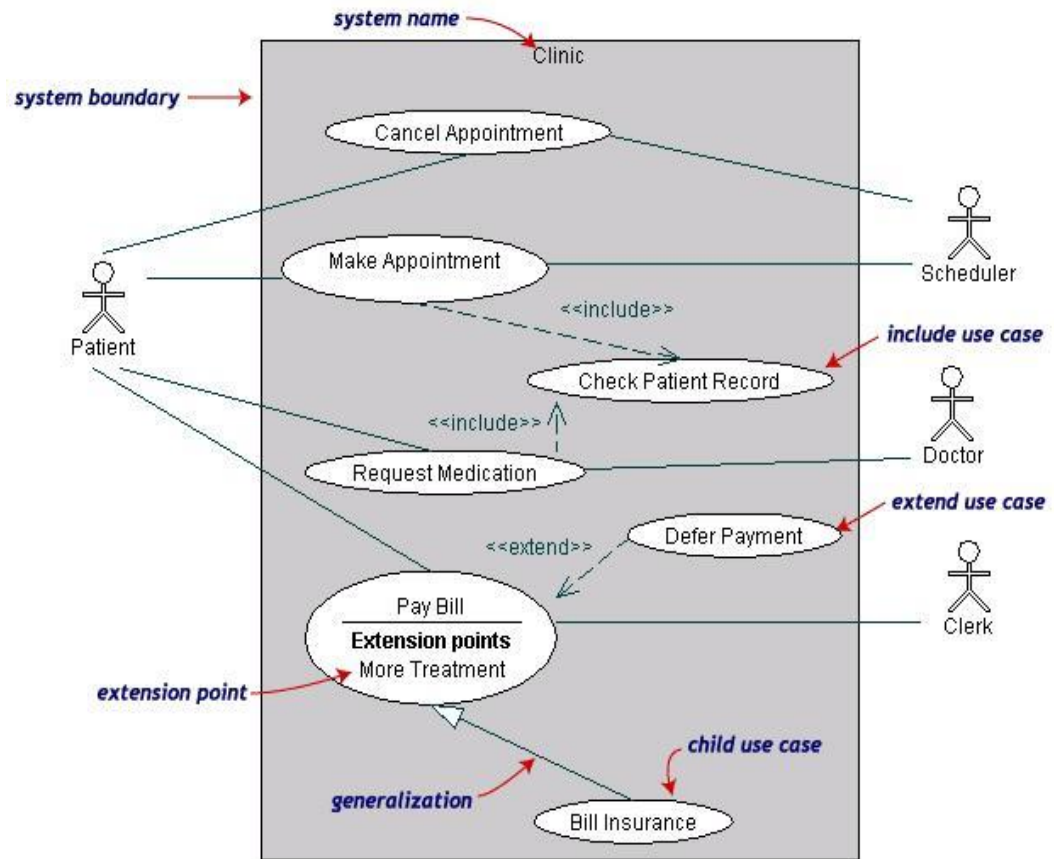


Figure 16.12

Use-Case Diagrams

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



(TogetherSoft, Inc)



Class diagram

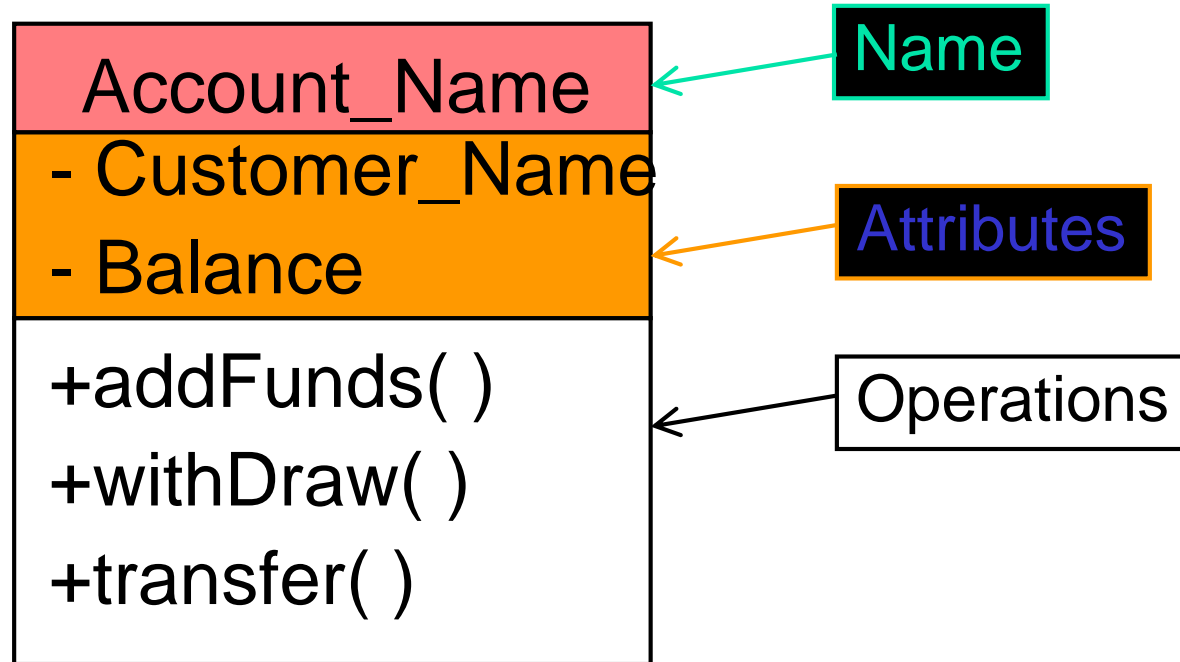
- A class diagram depicts classes and their interrelationships
- Used for describing **structure and behavior** in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used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



Class diagram

- 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.

Class diagram

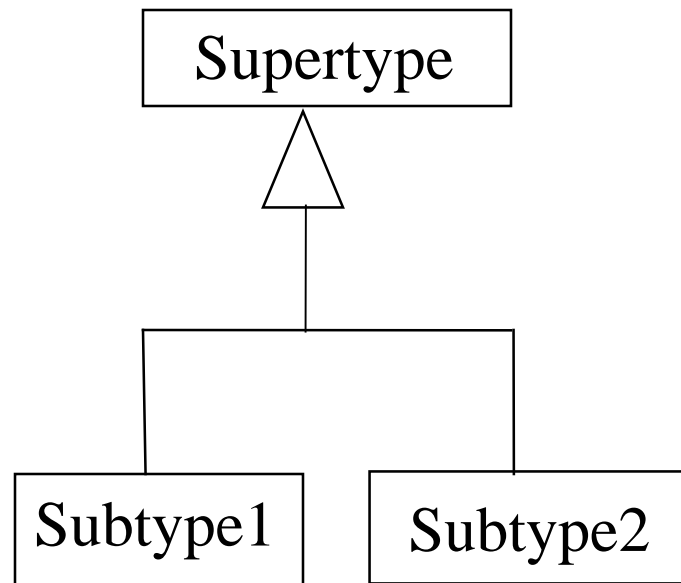




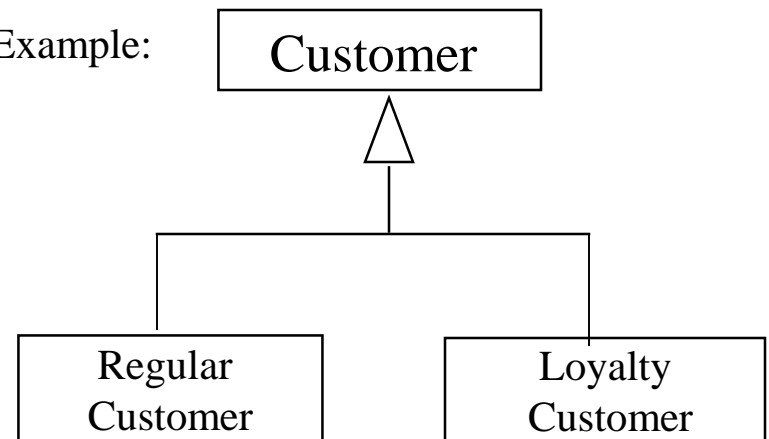
OO Relationships

- 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:



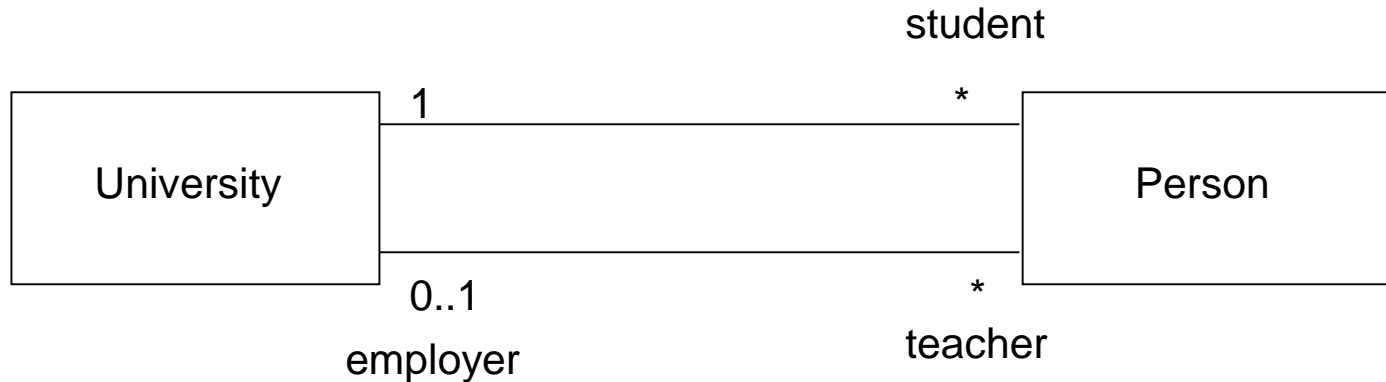
- Inheritance is a required feature of object orientation
- Generalization expresses a parent/child relationship among related classes.
- Used for abstracting details in several layers



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



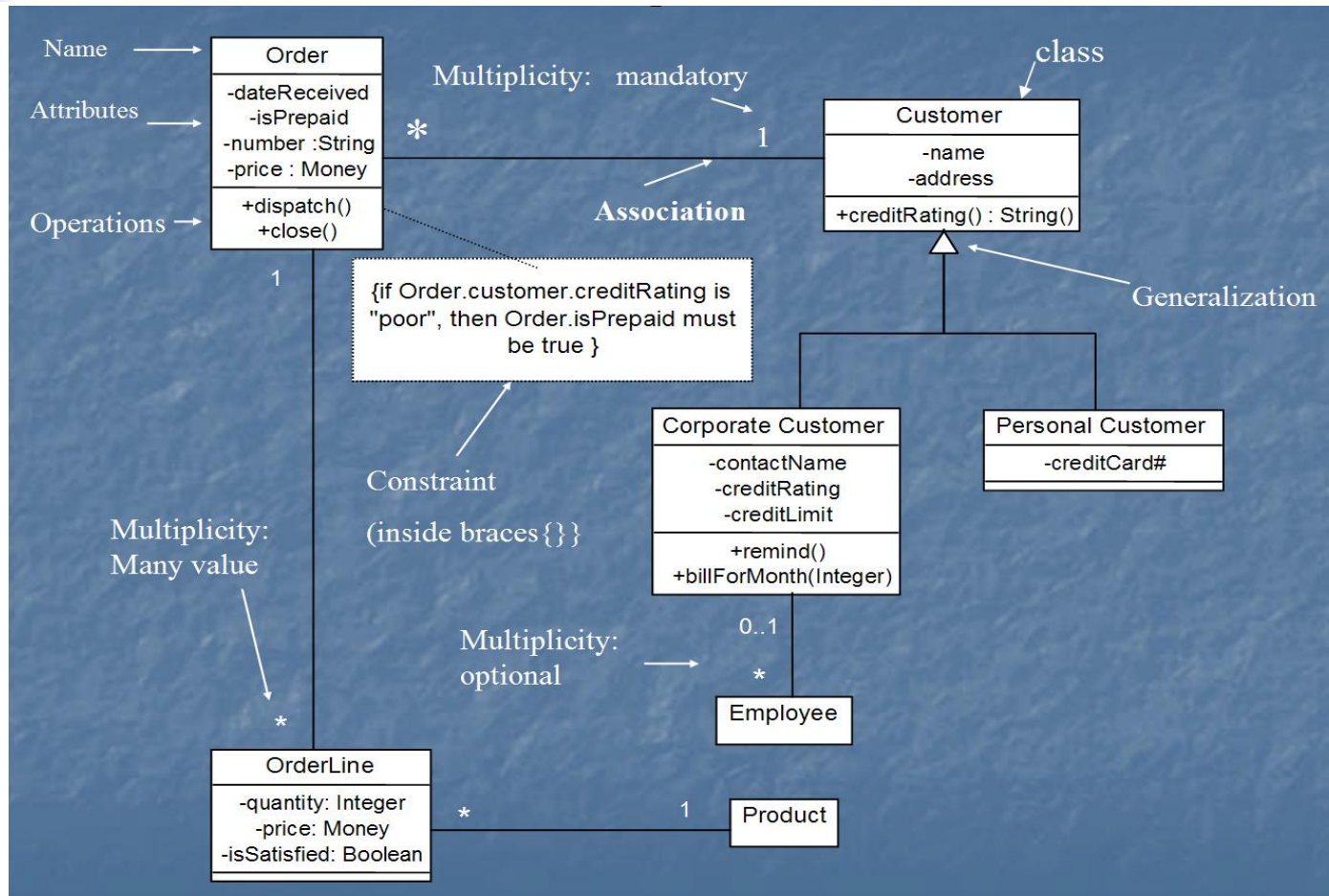
Multiplicity

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

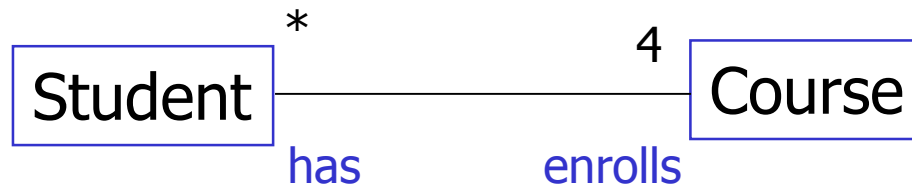
Role

"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



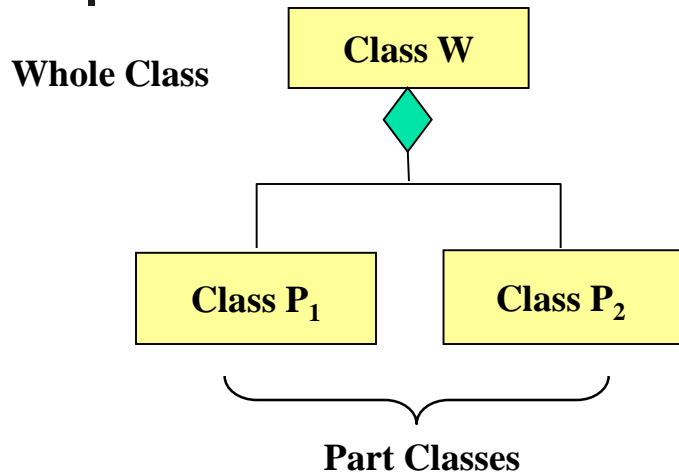
Association: Model to Implementation



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

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

OO Relationships: **Composition**



[From Dr.David A. Workman]

Example

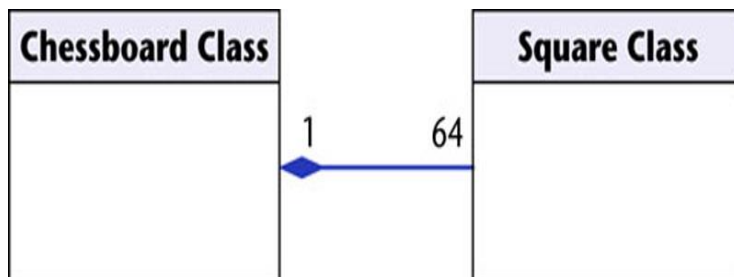


Figure 16.7

Association

Models the part–whole relationship

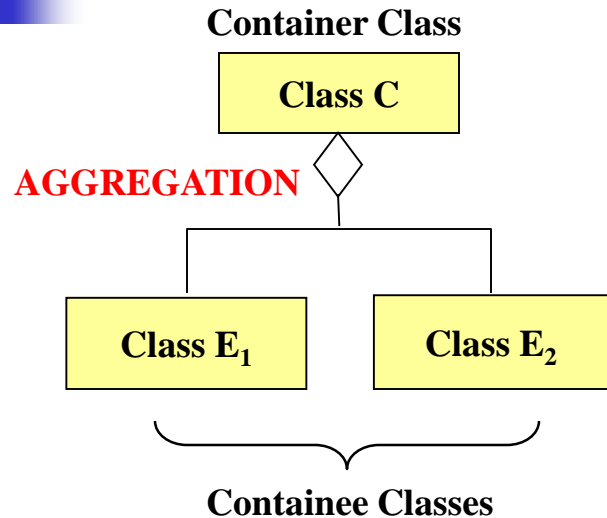
Composition

Also models the part–whole relationship but, in addition, Every part may belong to only one whole, and If the whole is deleted, so are the parts

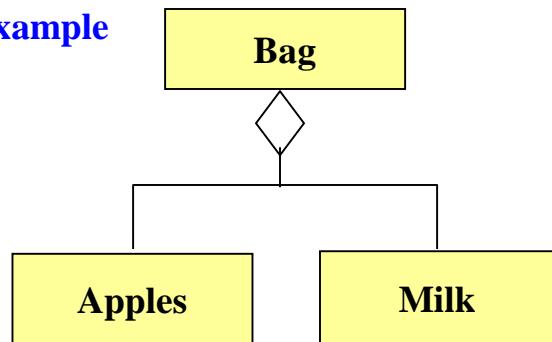
Example:

A number of different chess boards: Each square belongs to only one board. If a chess board is thrown away, all 64 squares on that board go as well.

OO Relationships: Aggregation



Example



Aggregation:

expresses a relationship among instances of related classes. It is a specific kind of Container-Containeer relationship.

express a more informal relationship than composition expresses.

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



Aggregation vs. Composition

■ **Composition** is really a strong form of **association**

- components have only one owner
- components cannot exist independent of their owner
- components live or die with their owner
- e.g. Each car has an engine that can not be shared with other cars.

■ **Aggregations**

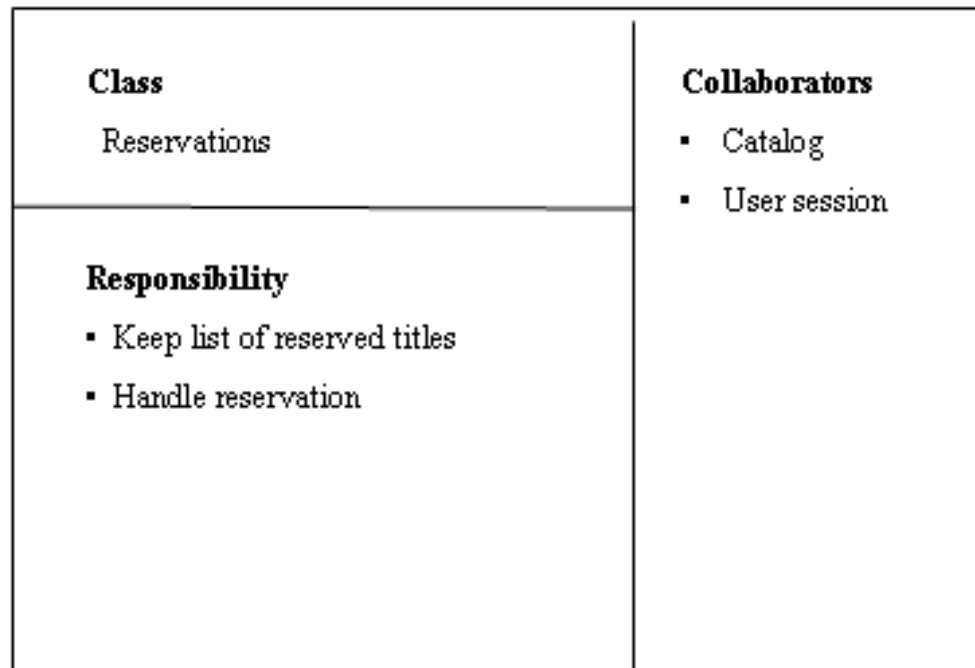
may form "part of" the association, but may not be essential to it. They may also exist independent of the aggregate. e.g. Apples may exist independent of the bag.



Good Practice: CRC Card

Class Responsibility Collaborator

- easy to describe how classes work by moving cards around; allows to quickly consider alternatives.

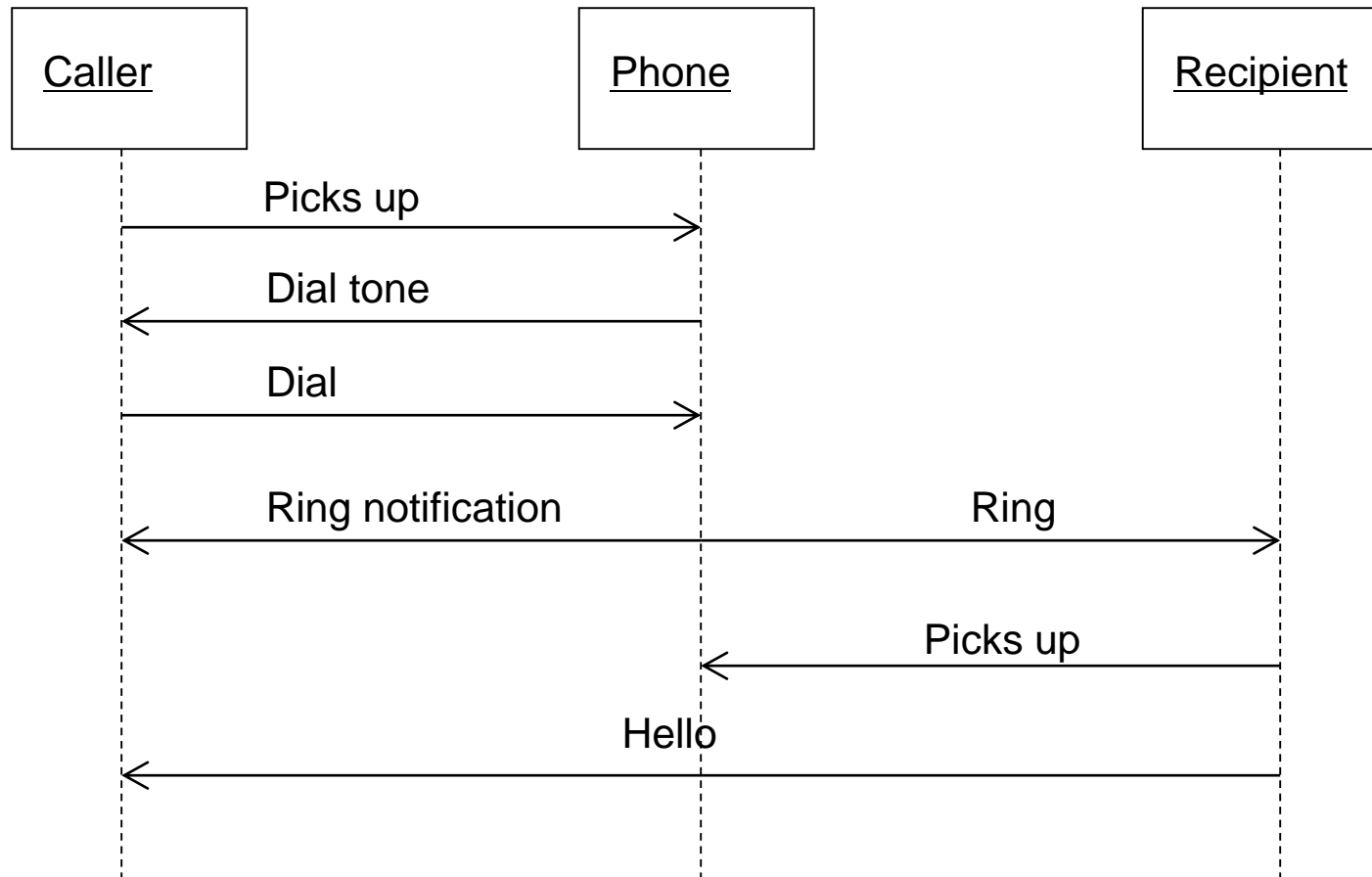




Interaction Diagrams

- show how objects interact with one another
- UML supports two types of interaction diagrams
 - Sequence diagrams
 - Collaboration diagrams

Sequence Diagram(make a phone call)



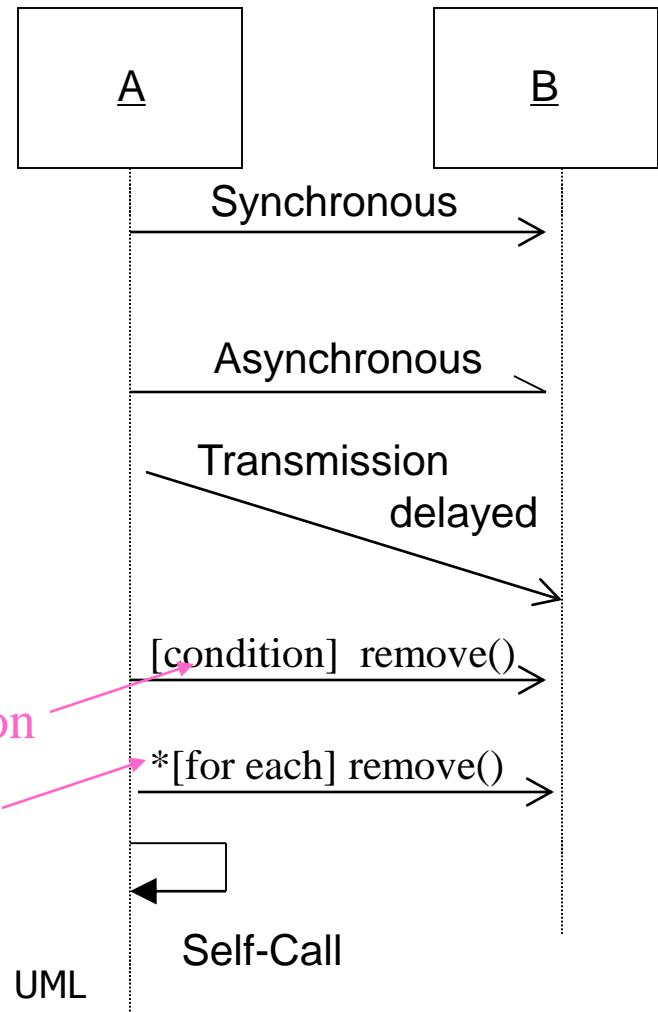
Sequence Diagram: Object interaction

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.

Condition

Iteration



Sequence Diagrams – Object Life Spans

■ Creation

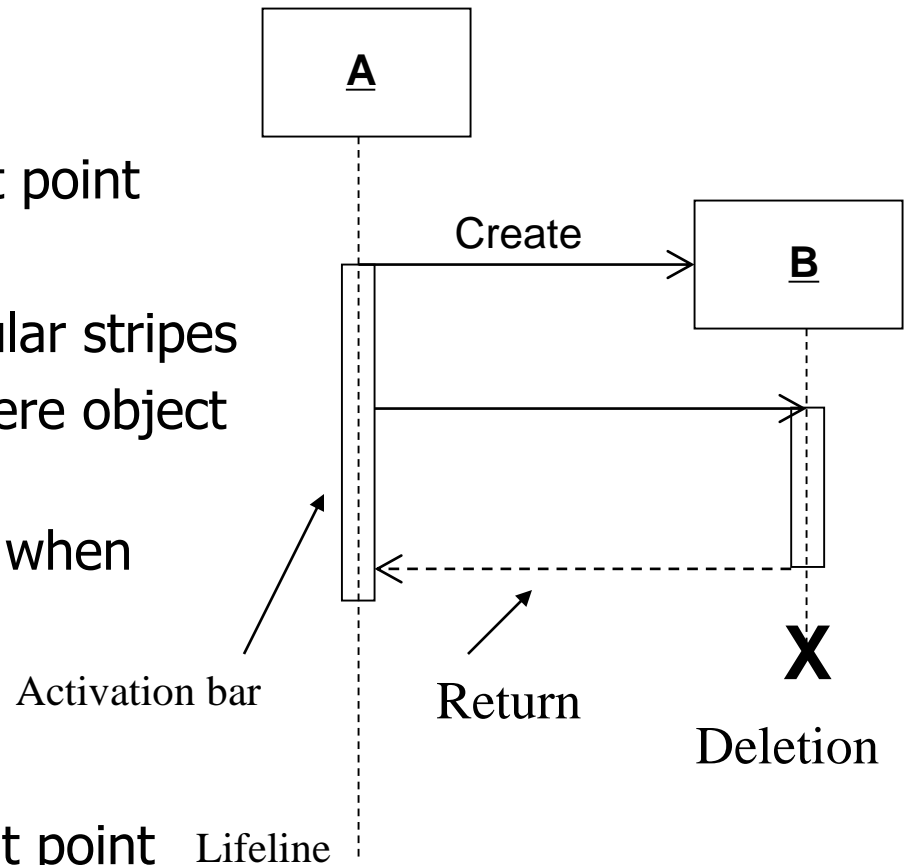
- Create message
- Object life starts at that point

■ Activation

- Symbolized by rectangular stripes
- Place on the lifeline where object is activated.
- Rectangle also denotes when object is deactivated.

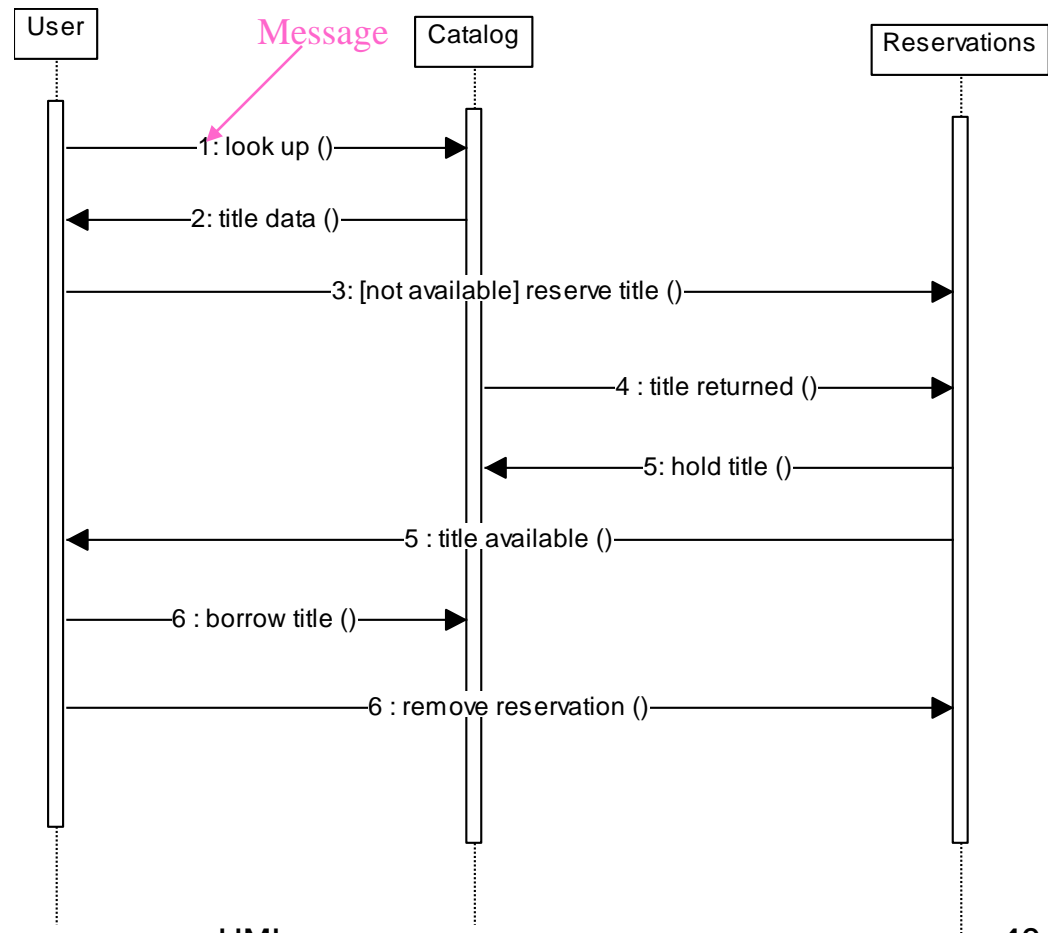
■ Deletion

- Placing an 'X' on lifeline
- Object's life ends at that point

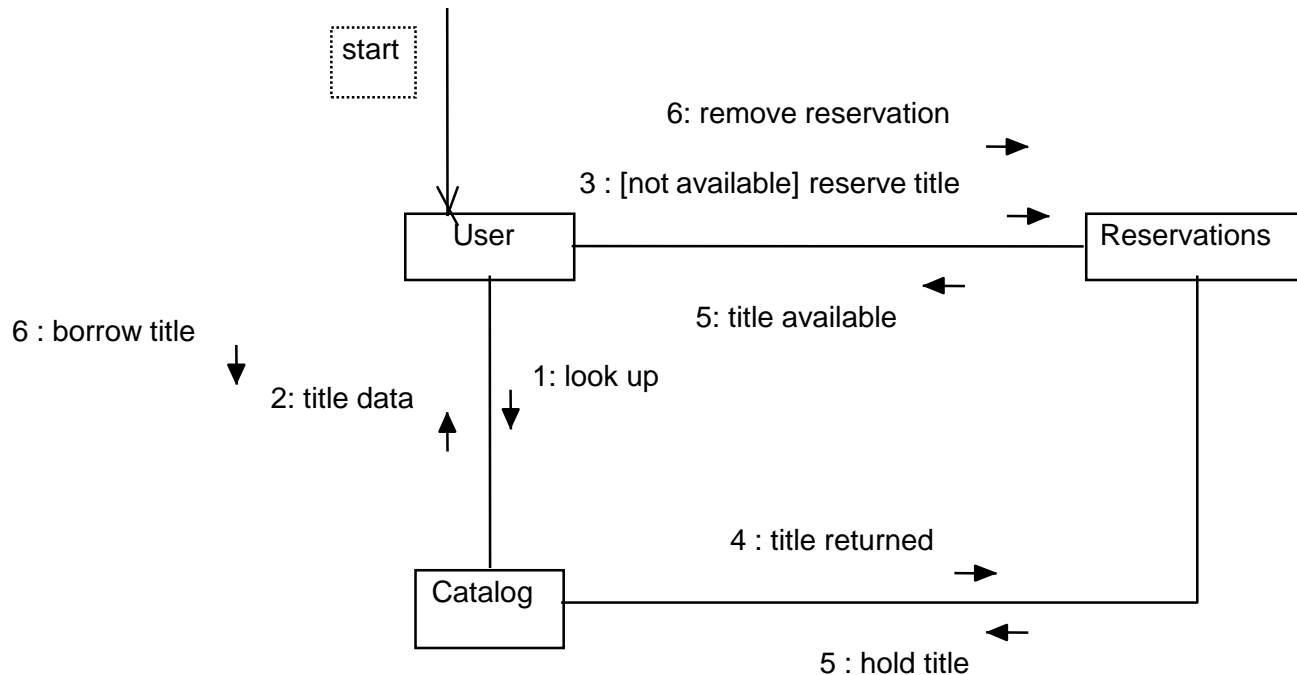


Sequence Diagram

- 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

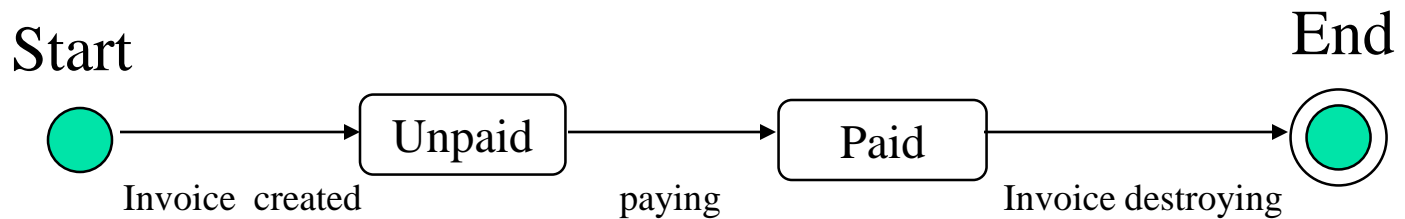


- Collaboration diagrams are equivalent to sequence diagrams. All the features of sequence diagrams are equally applicable to collaboration diagrams
- Use a sequence diagram when the transfer of information is the focus of attention
- Use a collaboration diagram when concentrating on the classes

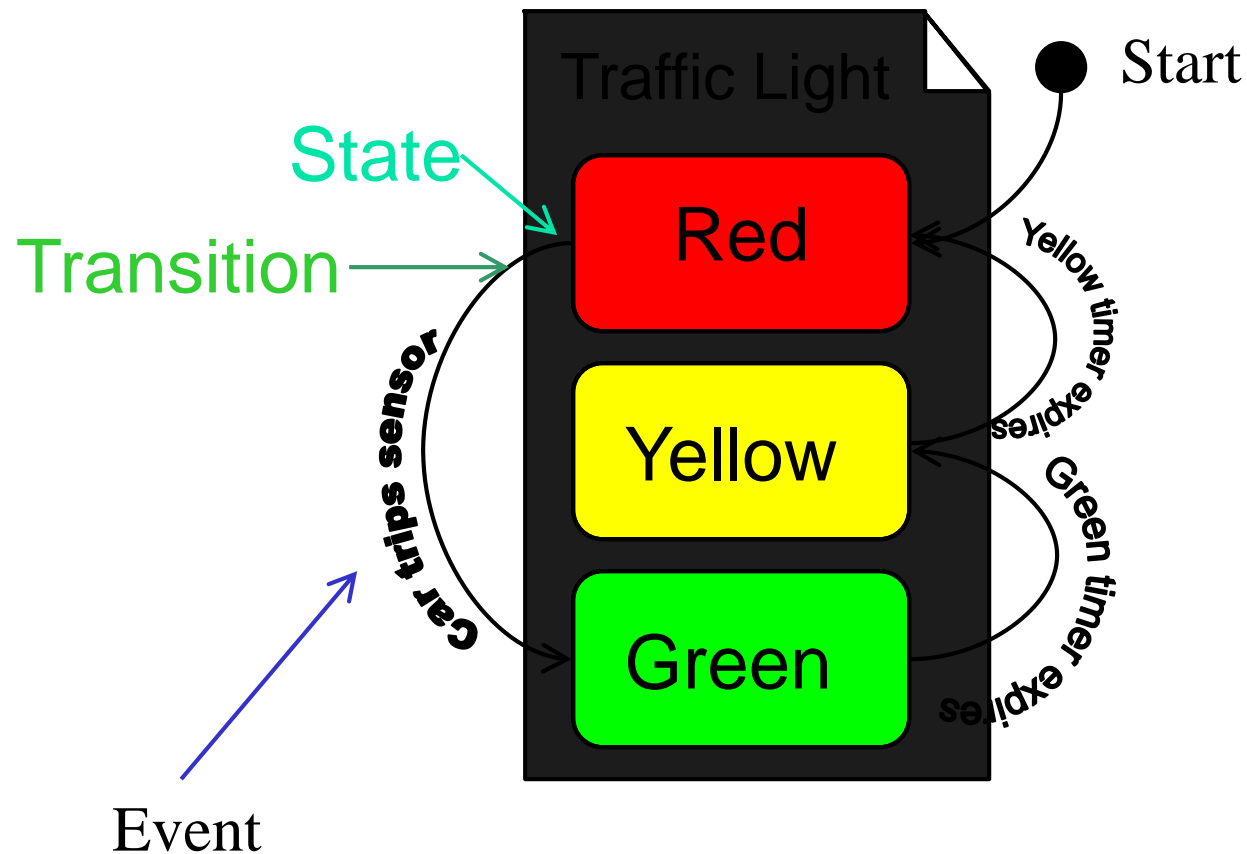


State Diagrams (Billing Example)

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)





What UML Modeling tools we use today?

- List of UML tools http://en.wikipedia.org/wiki/List_of_UML_tools
- ArgoUML: <http://argouml.tigris.org/>
- Rational Rose (www.rational.com) by IBM
- UML Studio 7.1 (<http://www.pragsoft.com/>) by Pragsoft Corporation:
Capable of handling very large models (tens of thousands of classes).
Educational License US\$ 125.00; Freeware version.
- TogetherSoft Control Center; TogetherSoft Solo
(<http://www.borland.com/together/index.html>) by Borland



Design using UML

- Many OOAD methodologies have been proposed
- They provide some guidelines on the steps to be performed
- Basic goal is to identify classes, understand their behavior, and relationships
- Different UML models are used for this
- Often UML is used, methodologies are not followed strictly



Design using UML

- Basic steps
 - Identify classes, attributes, and operations from use cases
 - Define relationships between classes
 - Make dynamic models for key use cases and use them to refine class diagrams
 - Make a functional model and use it to refine the classes
 - Optimize and package
- Class diagrams play the central role; class definition gets refined as we proceed

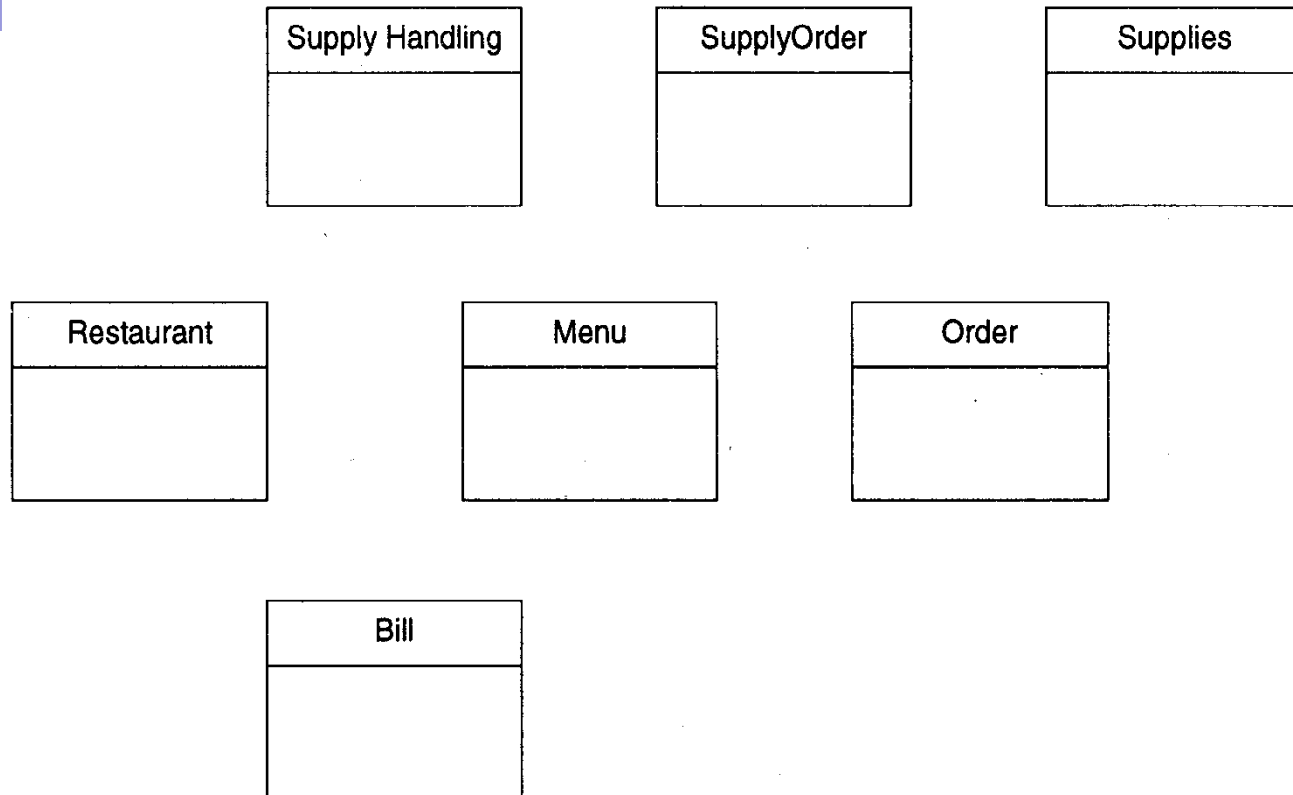


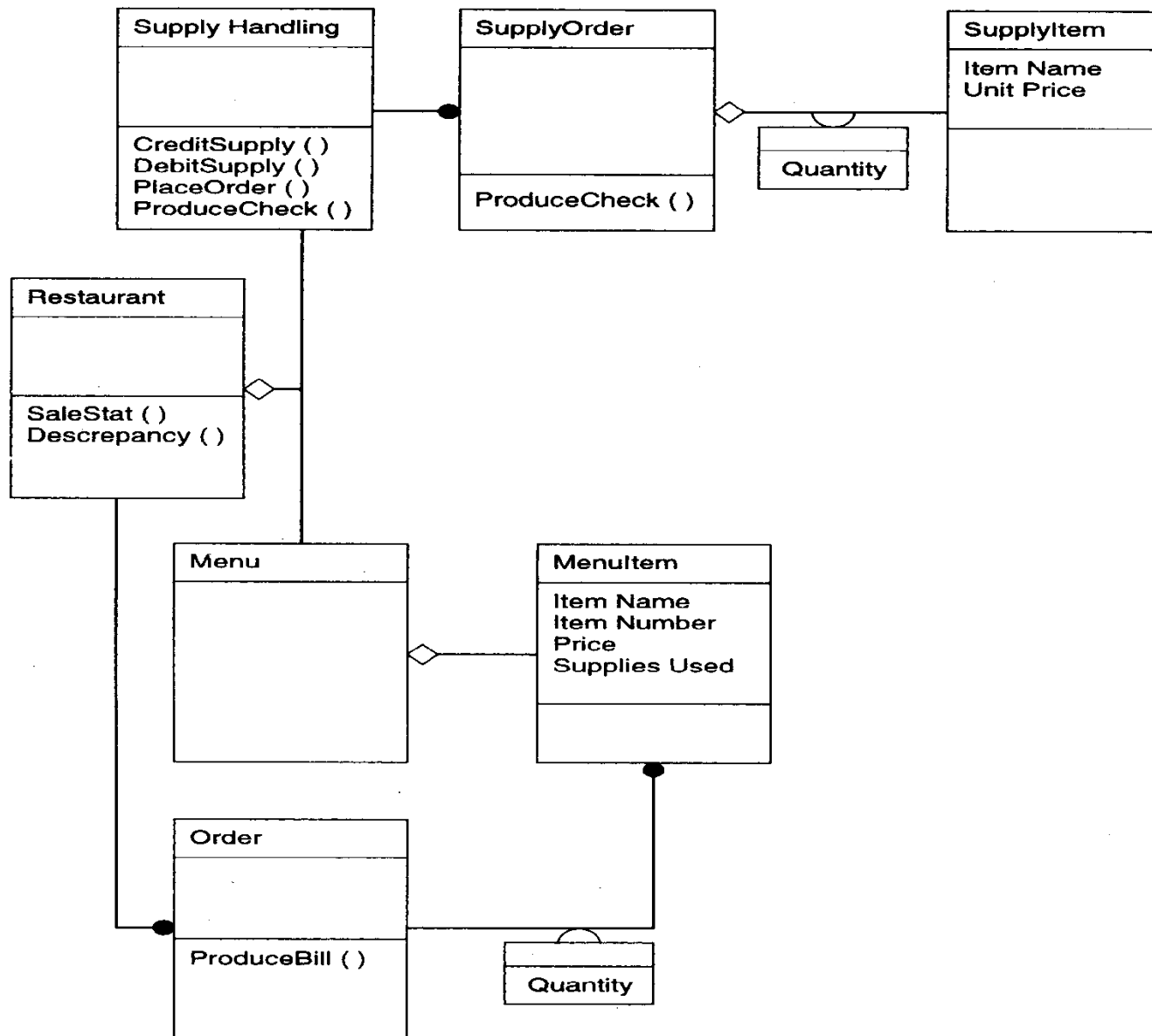
Success Scenario

- Customer read the menu
- Customer places the order
- Order is sent to the kitchen for preparation
- Ordered items are served
- Customer requests for a bill for the order
- Bill is prepared for this order
- Customer is given the bill
- Customer pays the bill

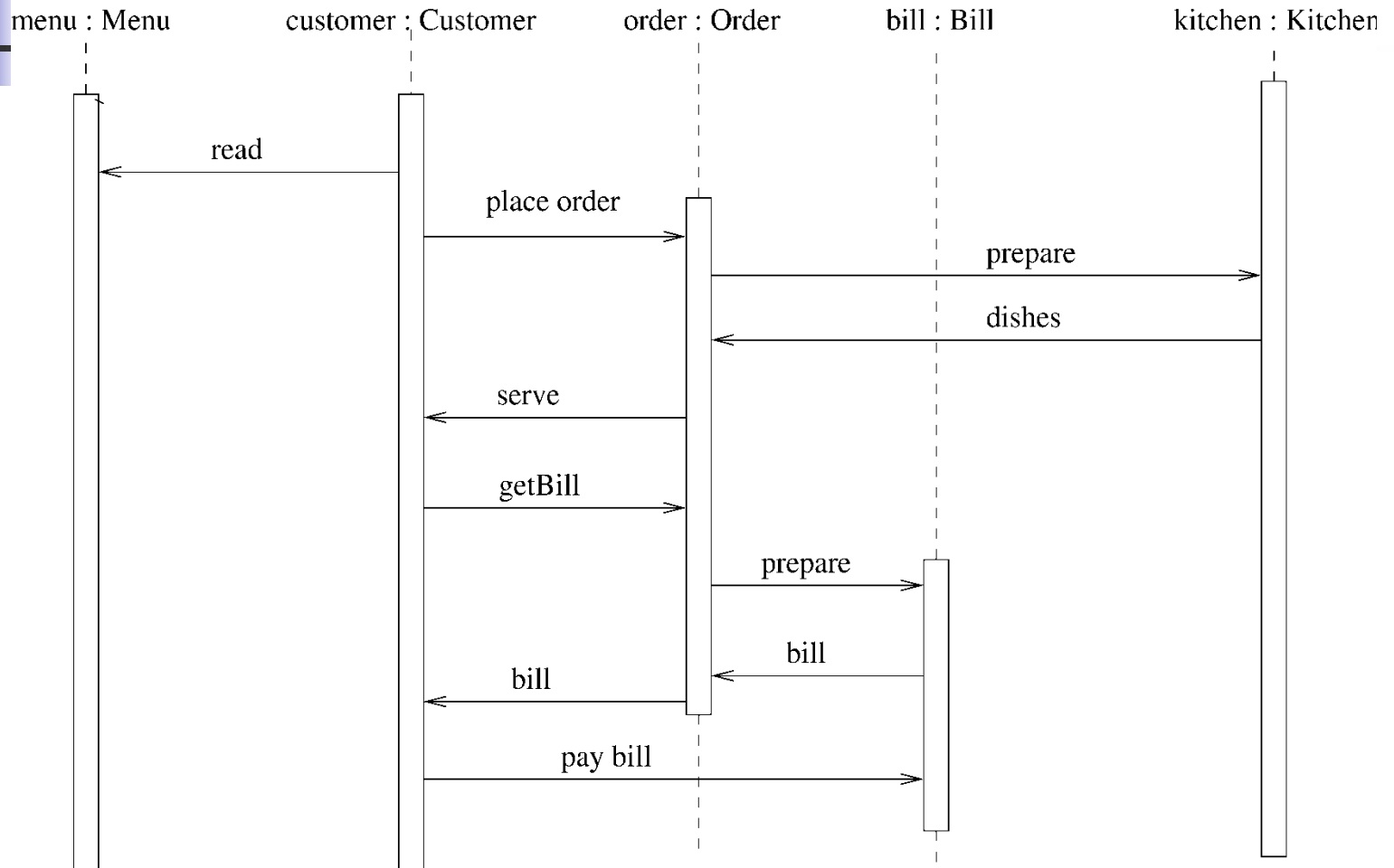


Restaurant example: Initial classes





Restaurant example: a sequence diagram

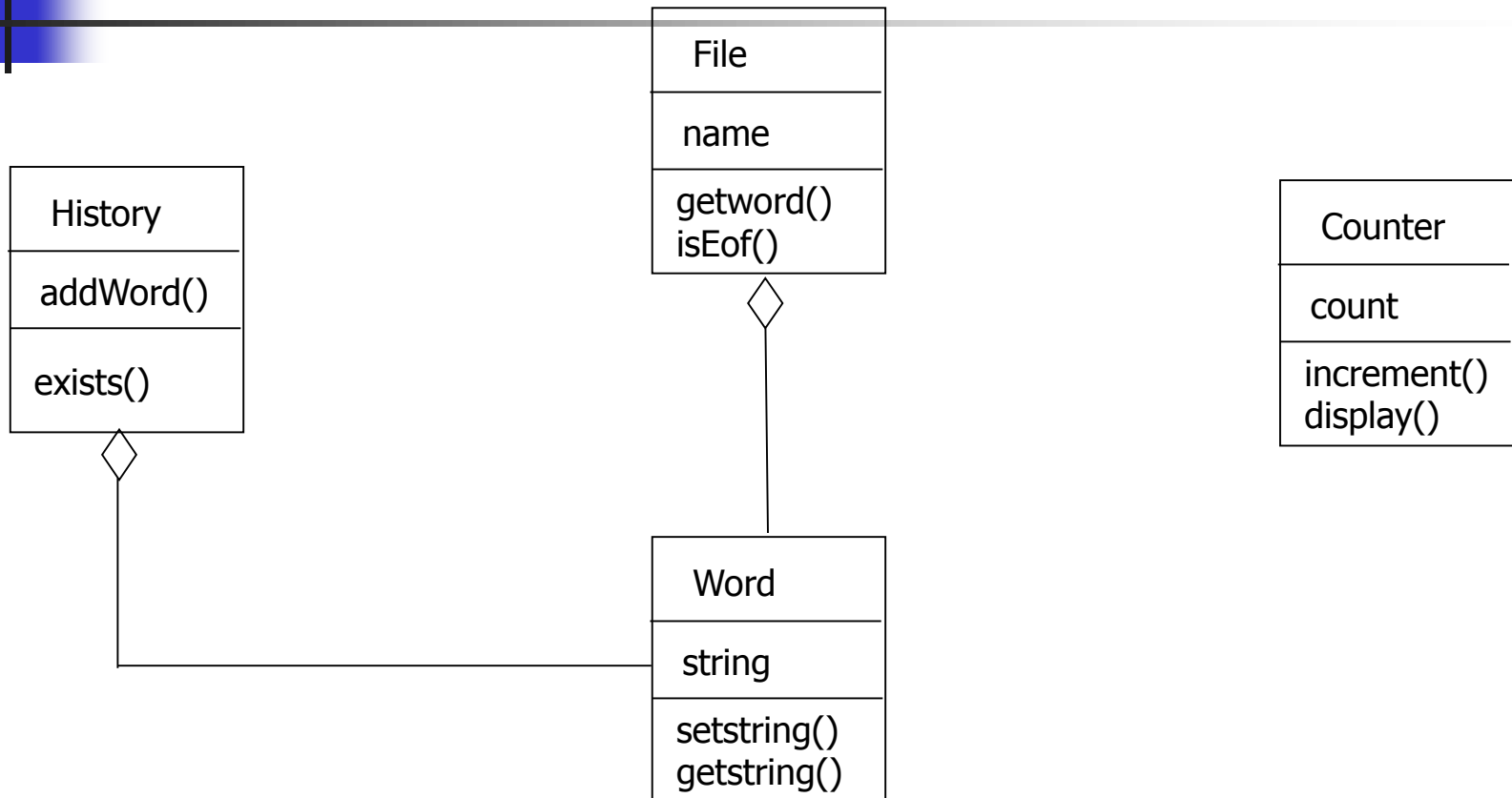




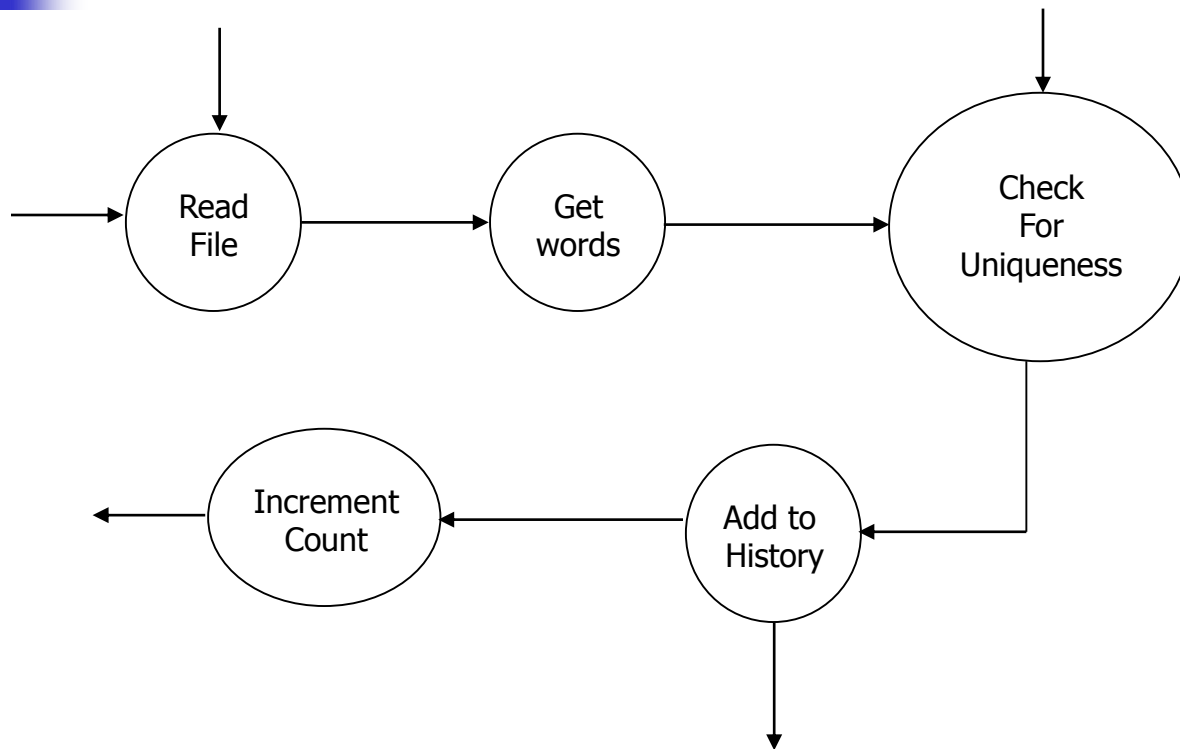
Example: Word Count Problem

- System prompts for the file name; user enters the file name
- System checks for existence of file
- System reads the words from the file
- Systems prints the count

Example: Word Count Problem...



Example: Word Count Problem...





Conclusion

- UML is a standardized specification language for object modeling
- Several UML diagrams:
 - use-case diagram: a number of use cases (use case models the interaction between actors and software)
 - Class diagram: a model of classes showing the static relationships among them including association and generalization.
 - Sequence diagram: shows the way objects interact with one another as messages are passed between them. Dynamic model
 - State diagram: shows states, events that cause transitions between states. Another dynamic model reflecting the behavior of objects and how they react to specific event
- There are several UML tools available



END
