# INFORMATION SYSTEMS ANALYSIS AND DESIGN

Mr. JACKSON ALUNGA

jalungar@gmail.com

1

# 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?

# What is UML and Why we use UML?

- UML → "Unified Modeling Language"
- Language: express idea, not a methodology

- Modeling: Describing a software system at a high level of abstraction

- Unified: UML has become a world standard
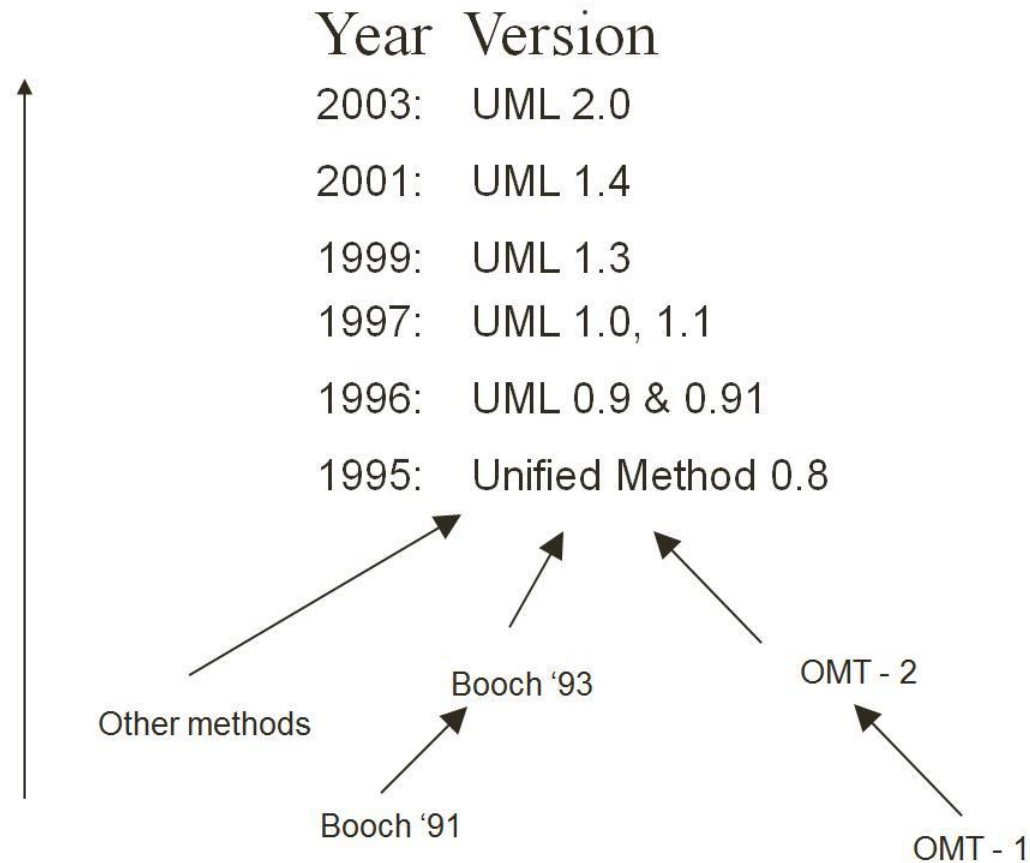  - Object Management Group (OMG): www.omg.org

# 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 and Why we use UML?

Year  Version

2003:  UML 2.0

2001:  UML 1.4

1999:  UML 1.3

1997:  UML 1.0, 1.1

1996:  UML 0.9 & 0.91

1995:  Unified Method 0.8

Other methods

Booch '93

Booch '91

OMT - 2

OMT - 1

6

# 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



9
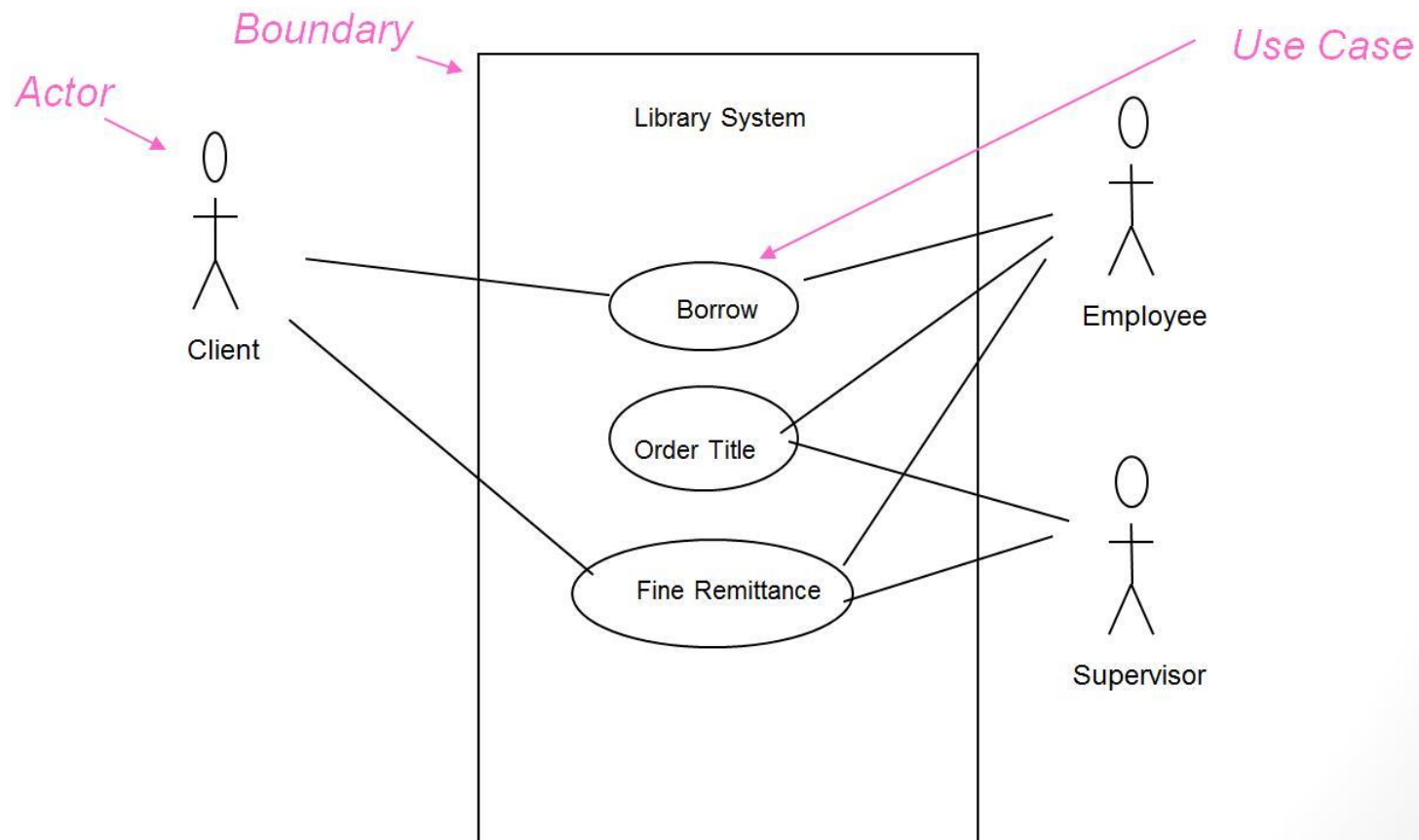
# Use-Case Diagrams

- **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.
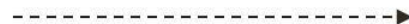
# Use-Case Diagrams

Include: a dotted line labeled <<include>> beginning at base use case and ending with an 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



Figure 16.12

The McGraw-Hill Companies, 2005
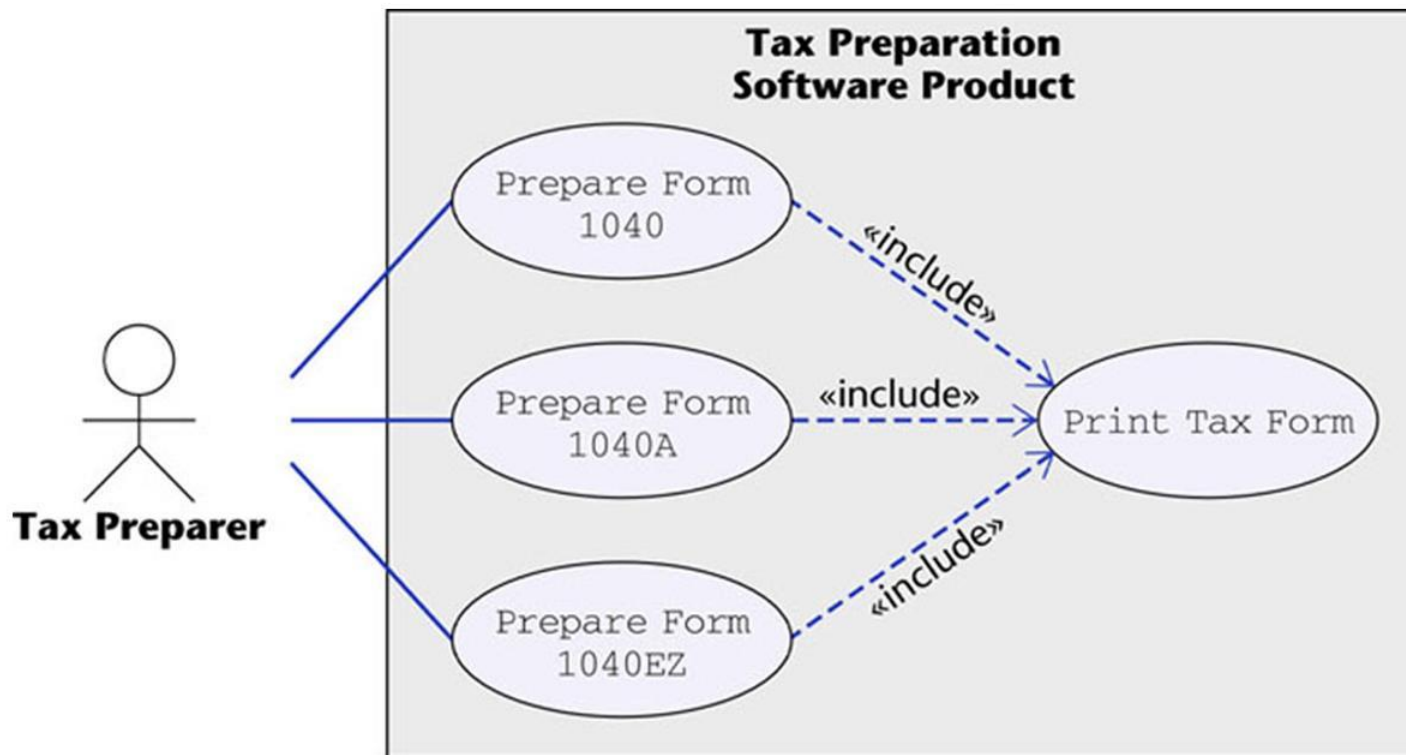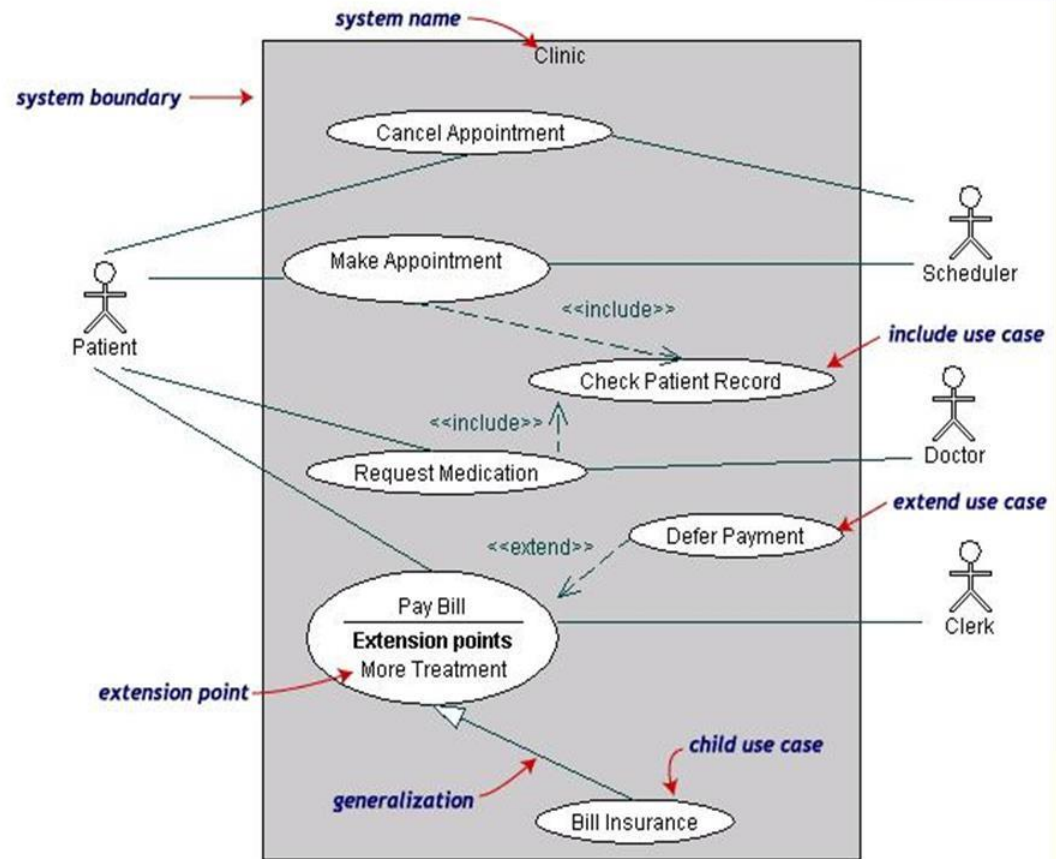
# Use-Case Diagrams

- 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)

14

# Class diagram
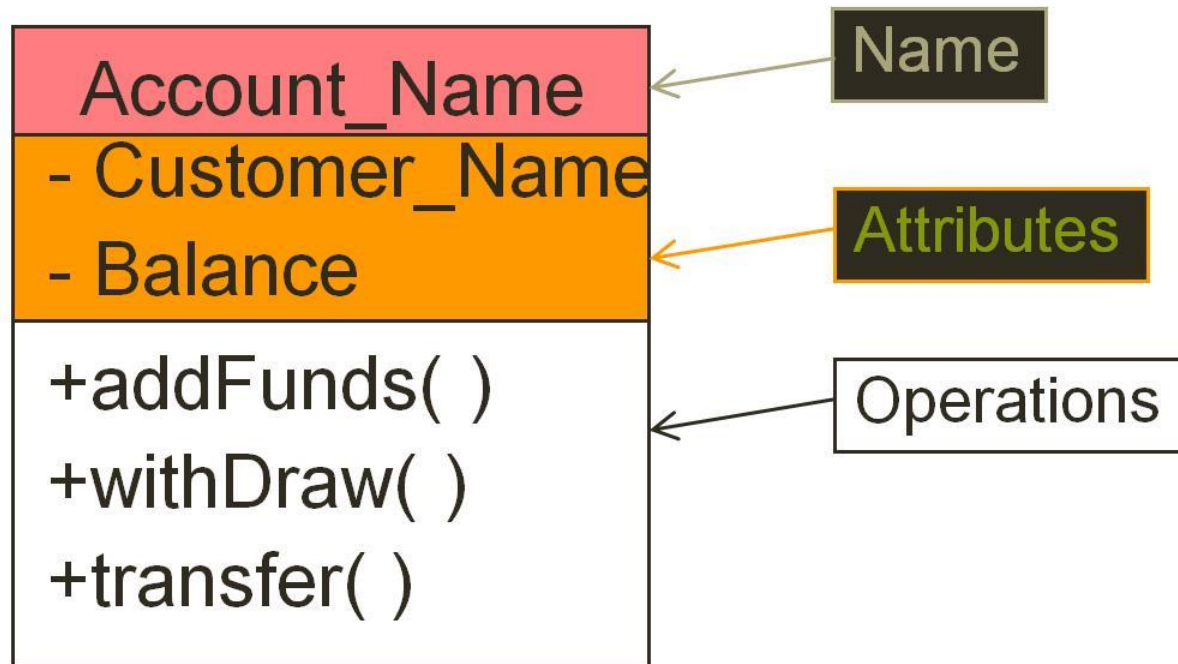
- A class diagram depicts classes and their interrelationships

- 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)

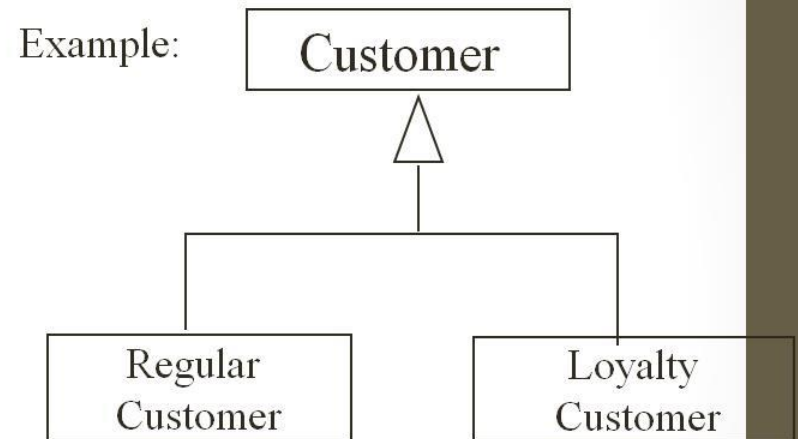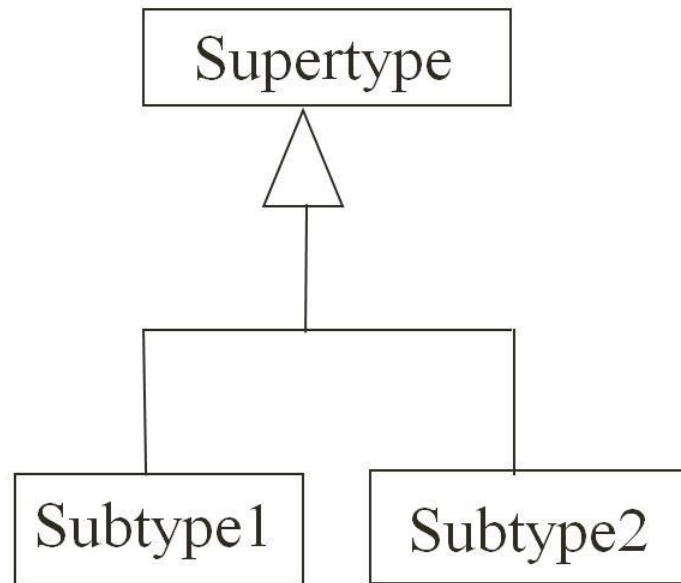- By default, attributes are hidden and operations are visible.

# Class diagram

| Account_Name |
|---|
| - Customer_Name |
| - Balance |
| +addFunds( ) |
| +withDraw( ) |
| +transfer( ) |

Name

Attributes

Operations

17

# 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

18

# OO Relationships: **Generalization**

Supertype

Subtype1    Subtype2

Example: Customer

Regular Customer    Loyalty Customer

-Inheritance is a required feature of object orientation

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

-Used for abstracting details in several layers

19

# 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
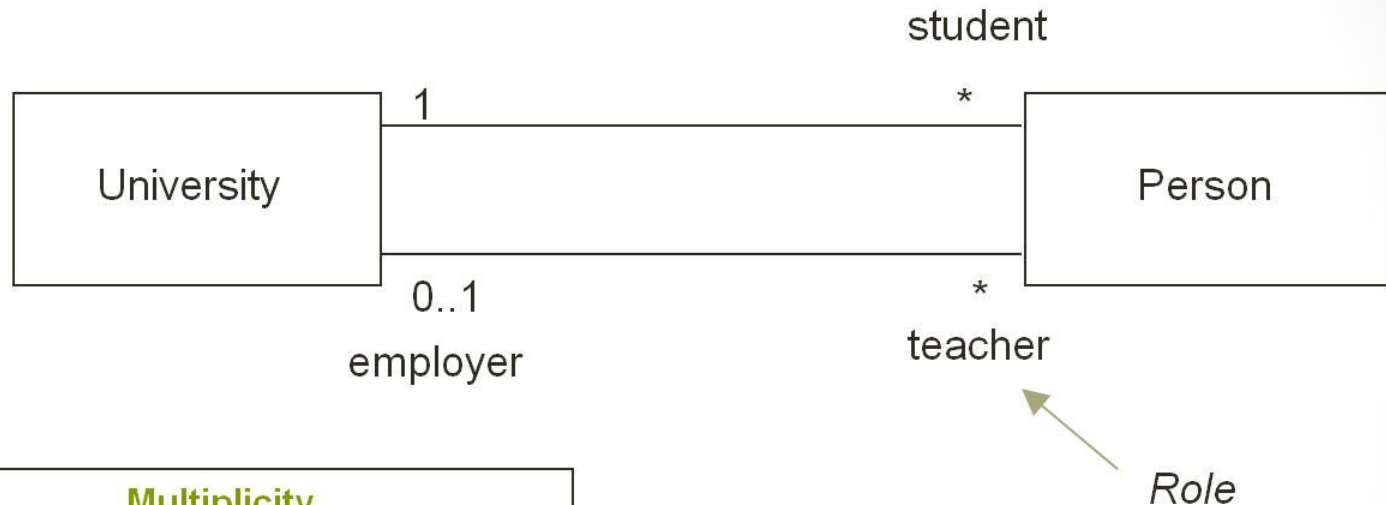


**Multiplicity**

| 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
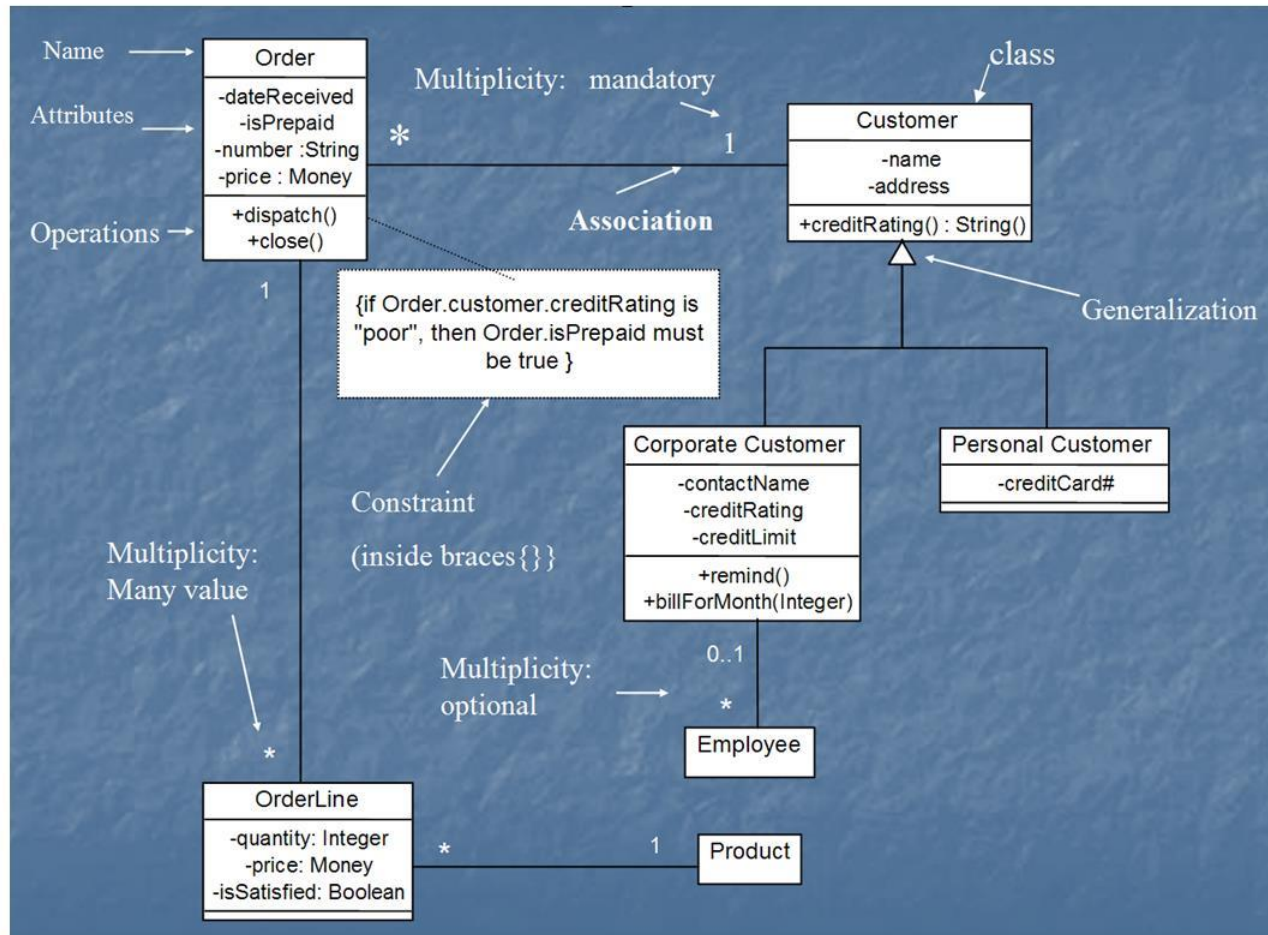

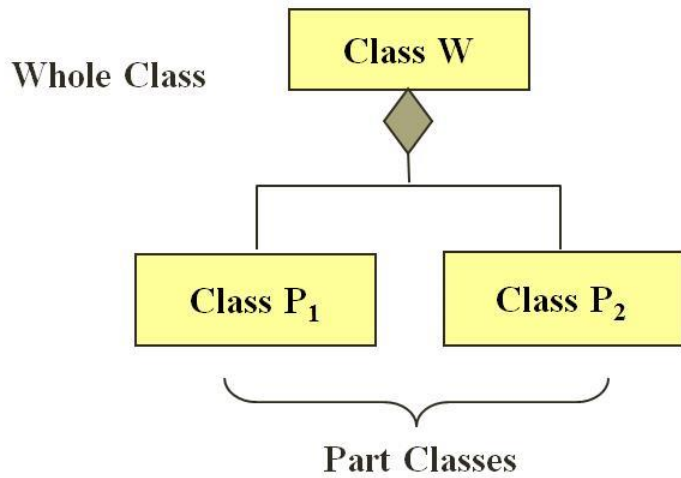
[from *UML Distilled Third Edition*]

22

# Association: Model to Implementation



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


Class Course {
    Student have[];
}
```
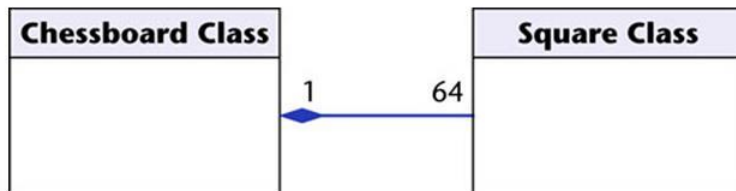
# OO Relationships: **Composition**

Whole Class



Part Classes

[From Dr.David A. Workman]

**Example**



Figure 16.7

The McGraw-Hill Companies, 2005

## Association
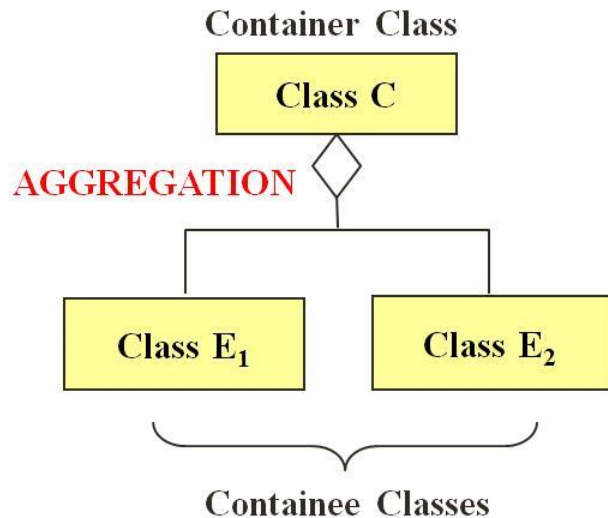Models the part–whole relationship

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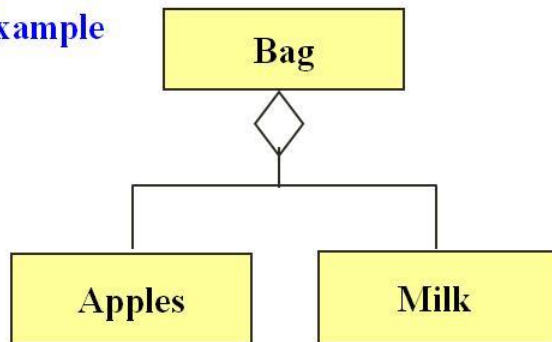
## Example:
A number of different chess boards: Each square belongs to only one board. If a chess board is thrown away, all 64 squares on that board go as well.

24

# OO Relationships: **Aggregation**

**Container Class**

```
        Class C
```

**AGGREGATION**

```
   Class E₁       Class E₂
```

**Containee Classes**

**Example**

```
        Bag

   Apples       Milk
```

[From Dr.David A. Workman]

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

express a more informal relationship than composition expresses.

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

25

# 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.

| Class | Collaborators |
|---|---|
| Reservations | - Catalog<br>- User session |
| **Responsibility** | |
| - Keep list of reserved titles<br>- Handle reservation | |

# Interaction Diagrams

- show how objects interact with one another

- UML supports two types of interaction diagrams
  - ➢Sequence diagrams
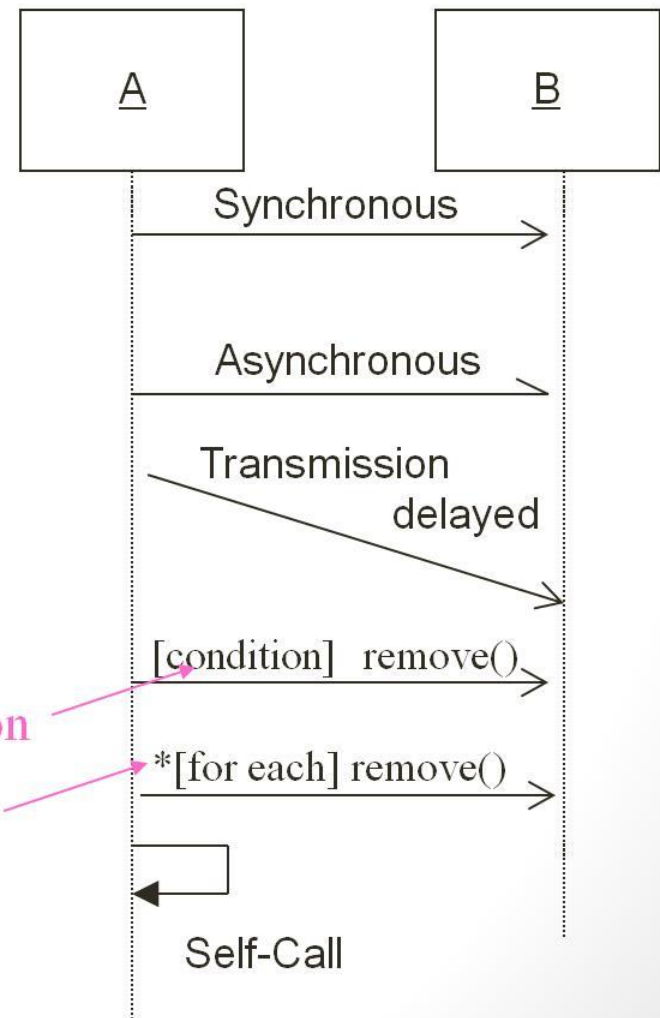  - ➢Collaboration diagrams

# Sequence Diagram(make a phone call)
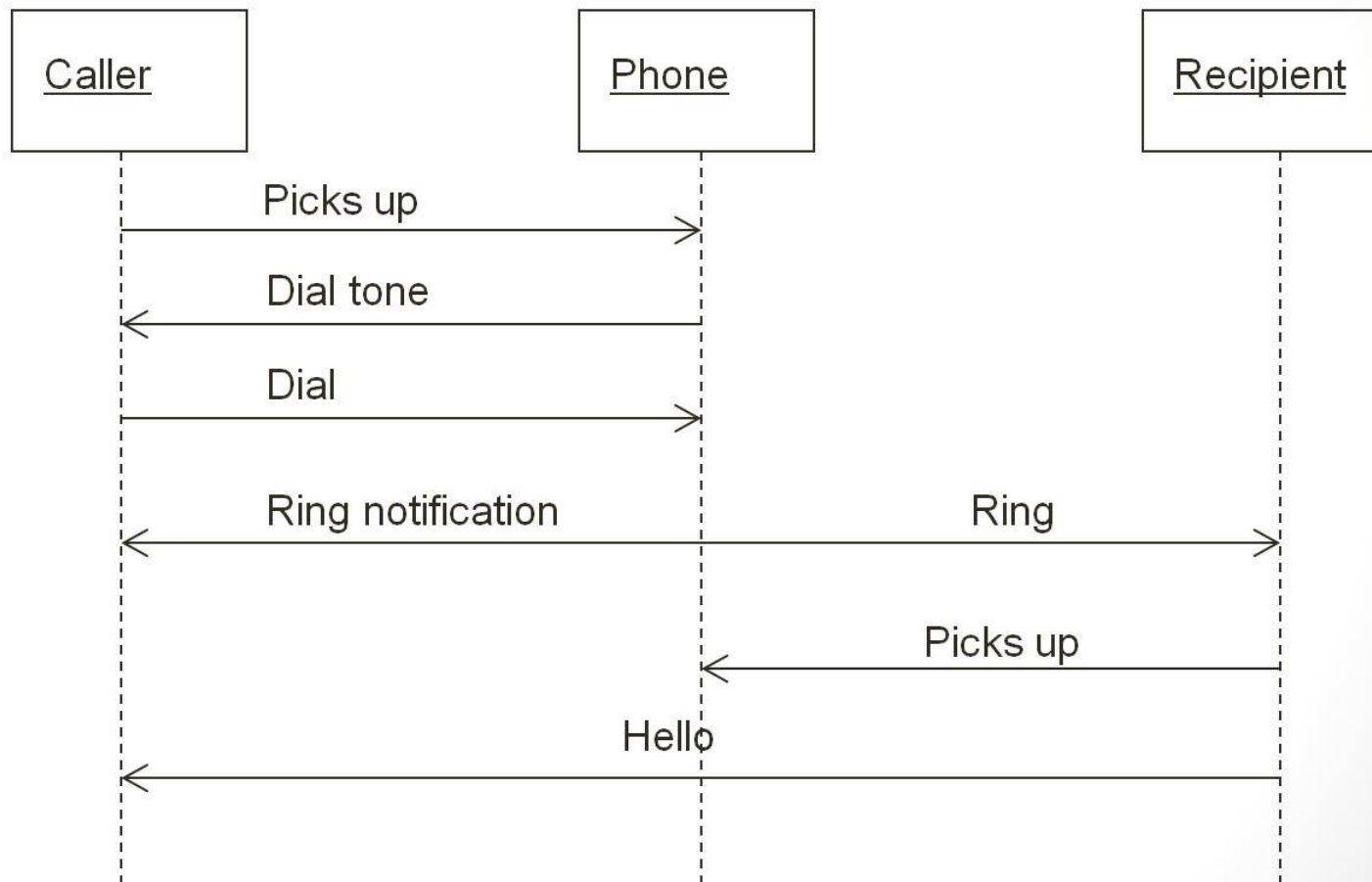
# Sequence Diagram:Object interaction

*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.

A

B

Synchronous

Asynchronous

Transmission
delayed

[condition]   remove()

Condition

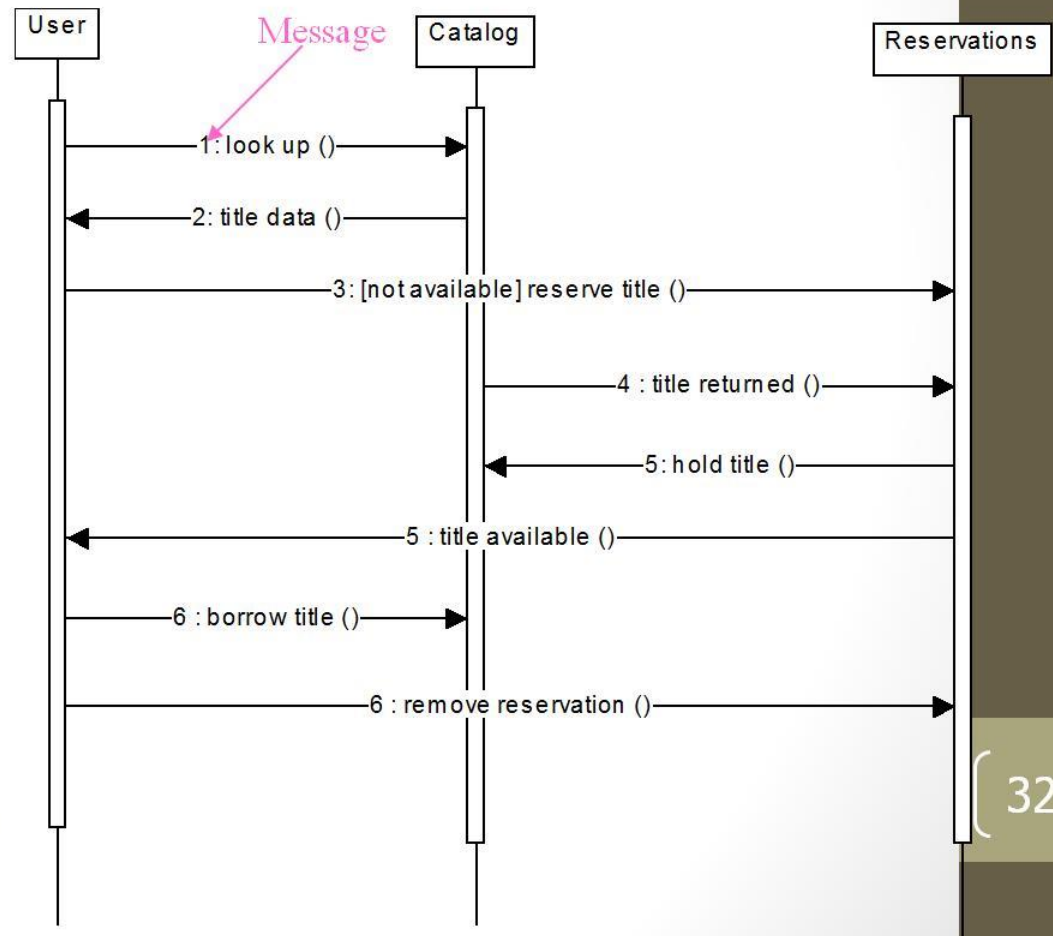*[for each] remove()

Iteration

Self-Call

# 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
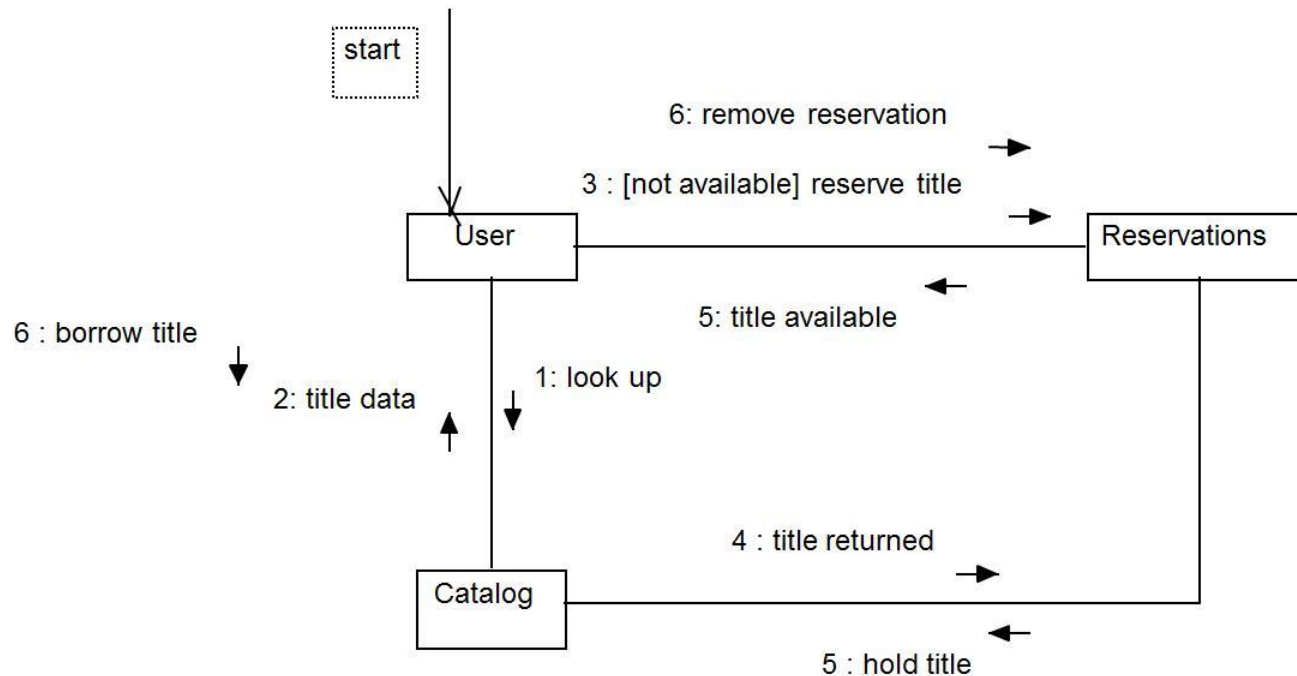  - Object's life ends at that point

A

Create

B

Activation bar

Return

Deletion

X

Lifeline

31
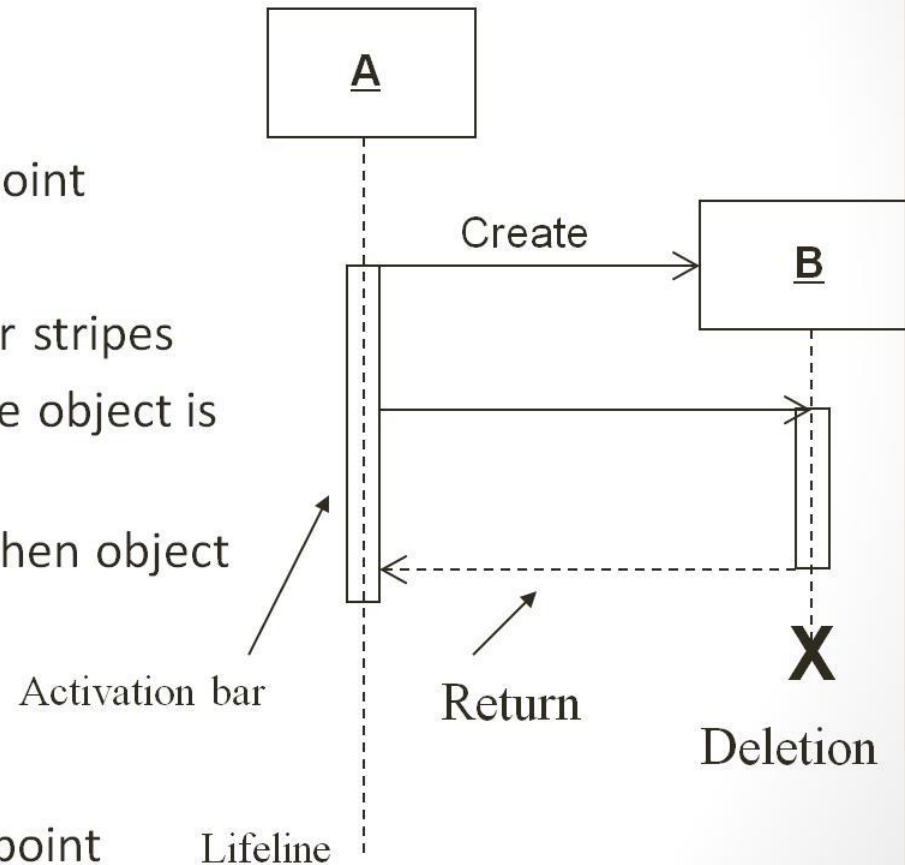
# Sequence Diagram

• 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.

| User | Message | Catalog | | Reservations |
|---|---|---|---|---|

1 : look up ()

2 : title data ()

3 : [not available] reserve title ()

4 : title returned ()

5 : hold title ()

5 : title available ()
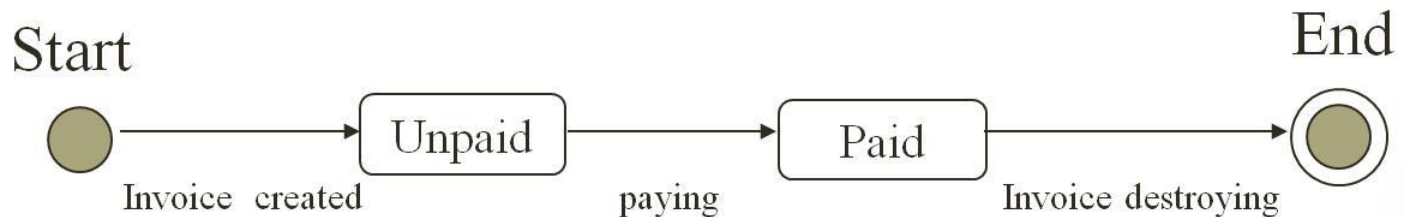
6 : borrow title ()

6 : remove reservation ()
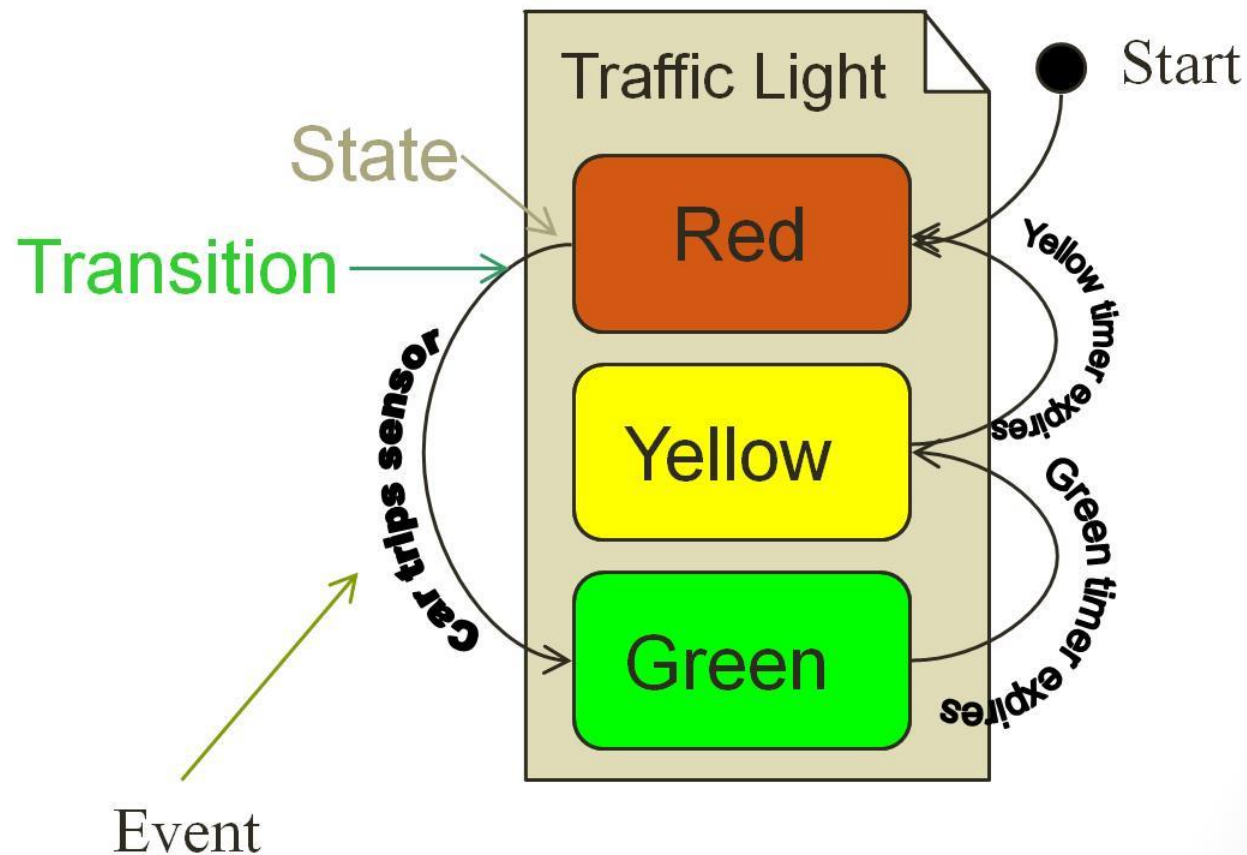
32

# 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

33

# 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.

Start                                                                        End

● ———— Invoice  created ——→ [ Unpaid ] ——— paying ——→ [ Paid ] ——— Invoice destroying ——→ ◉

# State Diagrams (Traffic light example)

# 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

# Thank you

Questions?