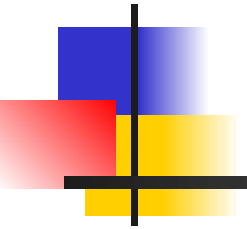
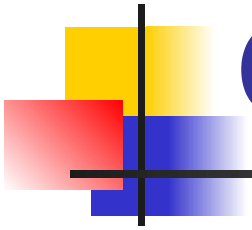
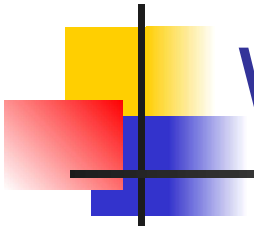


UML Lecture



Outline

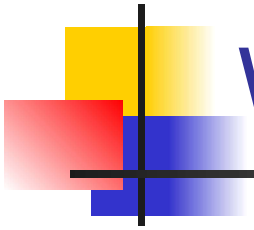
- What is UML and why we use UML?
- How to use UML diagrams to design software system?
- What UML Modeling tools we use today?



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:

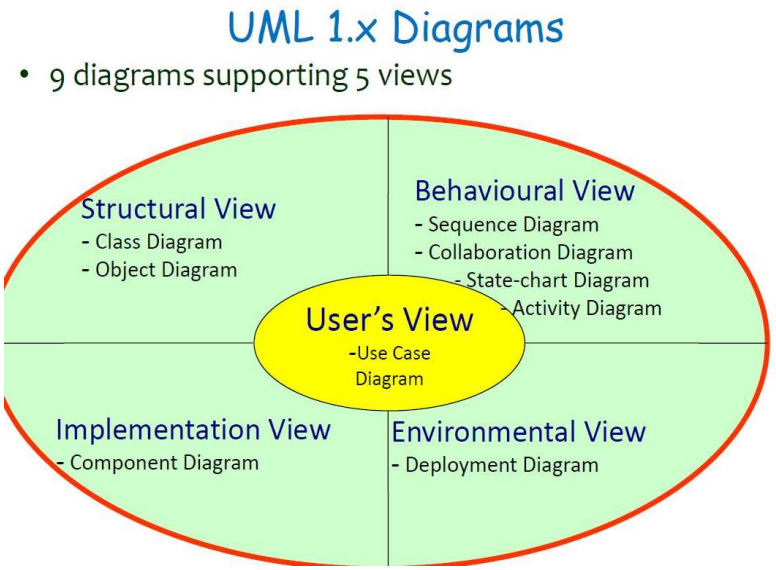
- It is a industry-standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems

- The UML uses mostly graphical notations to express the OO analysis and design of software projects.

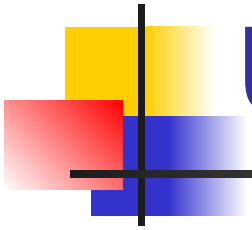
- Simplifies the complex process of software design

How to use UML diagrams to design software system?

- Types of UML Diagrams:
  - Use Case Diagram
  - Class Diagram
  - Sequence Diagram
  - Collaboration Diagram
  - State Diagram
  - Activity Diagram



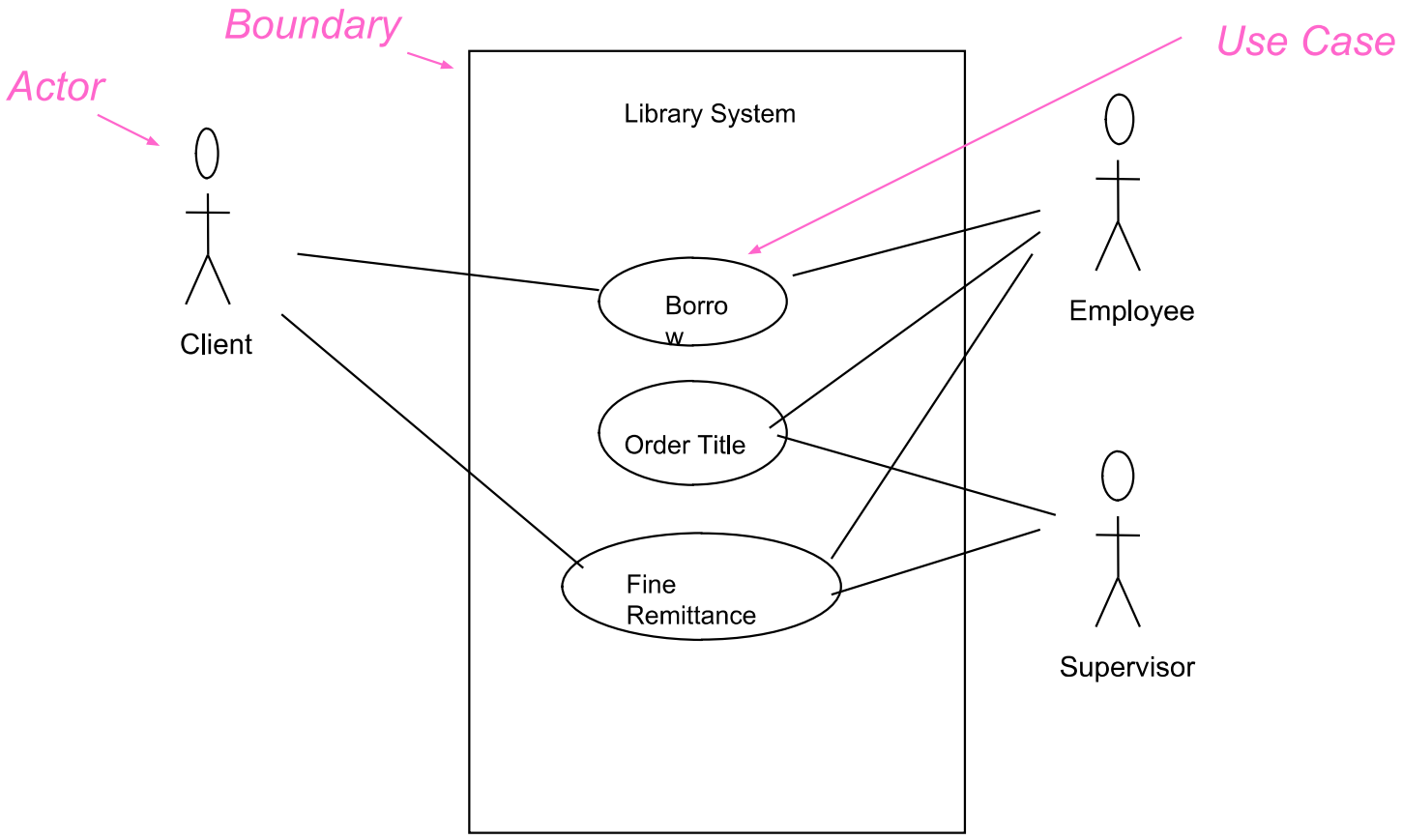
This is only a subset of diagrams … but are most widely used



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

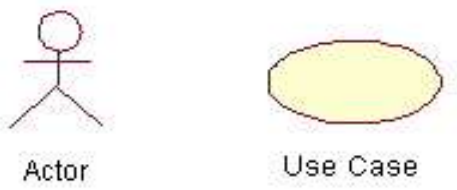


Use-Case Diagrams

- **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.

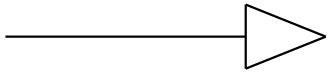


Use-Case Diagrams

- Association:

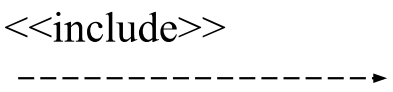
communication between an actor and a use case; Represented by a solid line.

- 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.

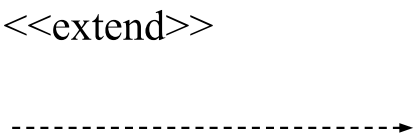


Use-Case Diagrams

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.



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".

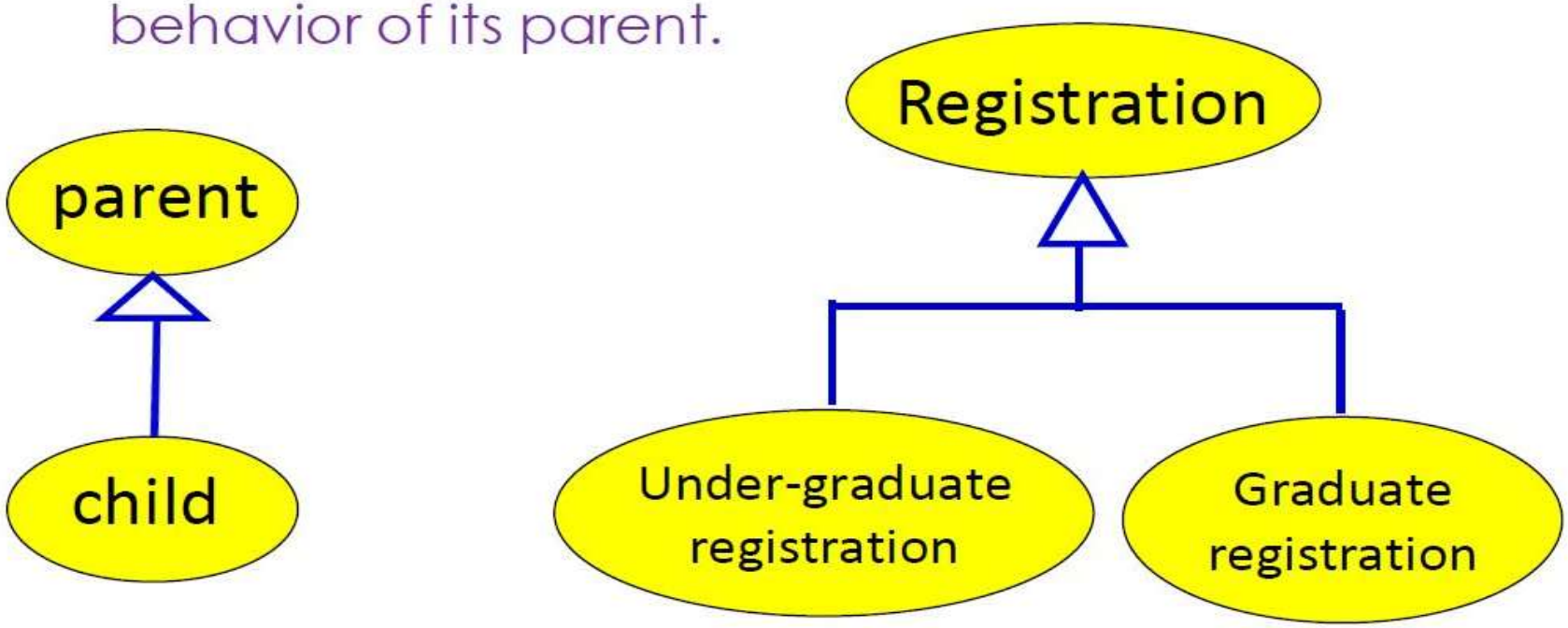


Factoring Use Cases

- Two main reasons for factoring:
  - Complex use cases need to be factored into simpler use cases
  - To represent common behaviour across different use cases
- Three ways of factoring:
  - Generalization
  - Include
  - Extend

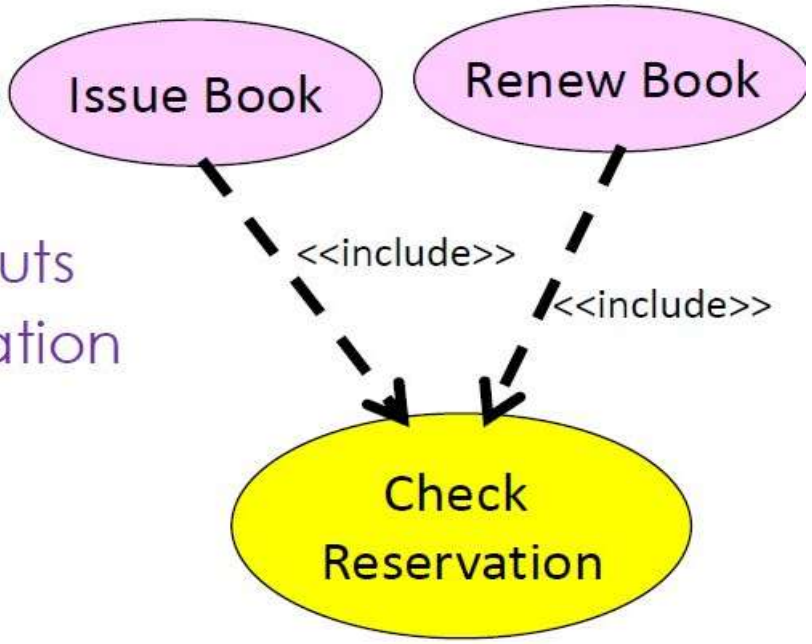
Generalization

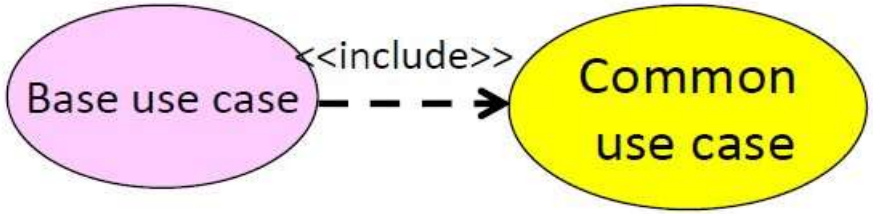
- The child use case inherits the behaviour of the parent use case.
  - The child may add to or override some of the behavior of its parent.



Include

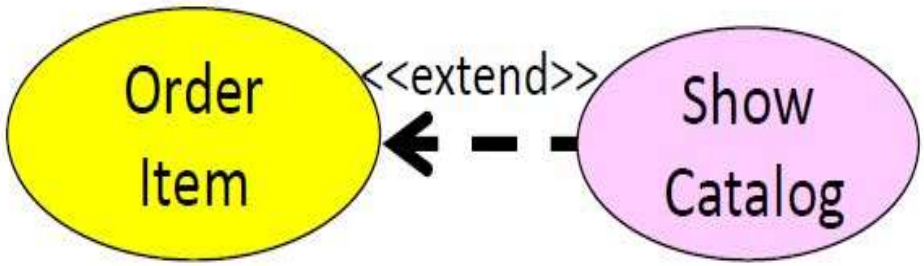
- When you have a piece of behaviour that is similar across many use cases
  - Break this out as a separate use-case and let the other ones "include" it
- Examples of use case include
  - Validate user interaction
  - Sanity check on sensor inputs
  - Check for proper authorization

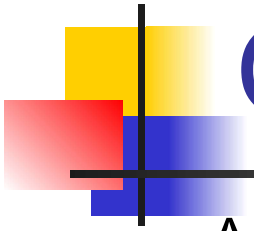




Extends

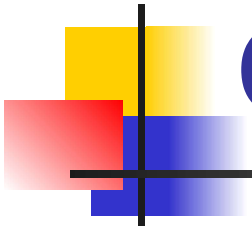
- Use when a use-case optionally can do a little bit more:
  - Capture the normal behaviour
  - Capture the extra behaviour in a separate use-case
  - Create extends dependency
- Makes it a lot easier to understand





Class diagram

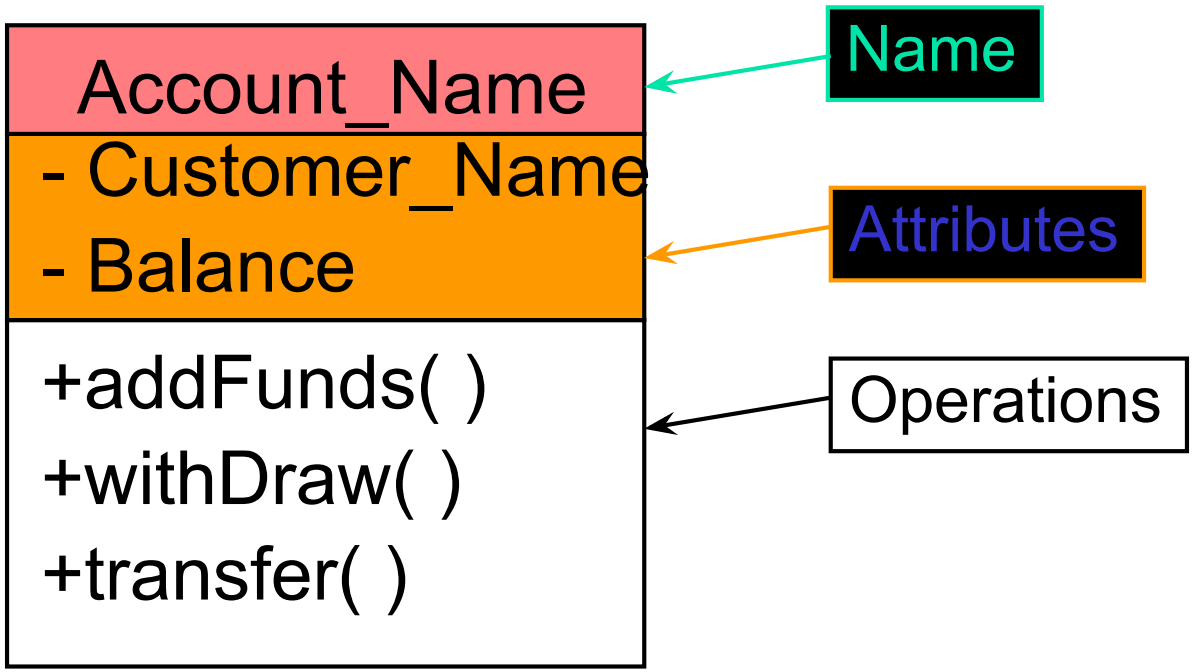
- A class diagram depicts **classes** and their **interrelationships**
- Used for describing structure and behavior in the use cases
- Provide a **conceptual model** of the system in terms of entities and their relationships
- Used for **requirement capture**, end-user interaction
- Detailed class diagrams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



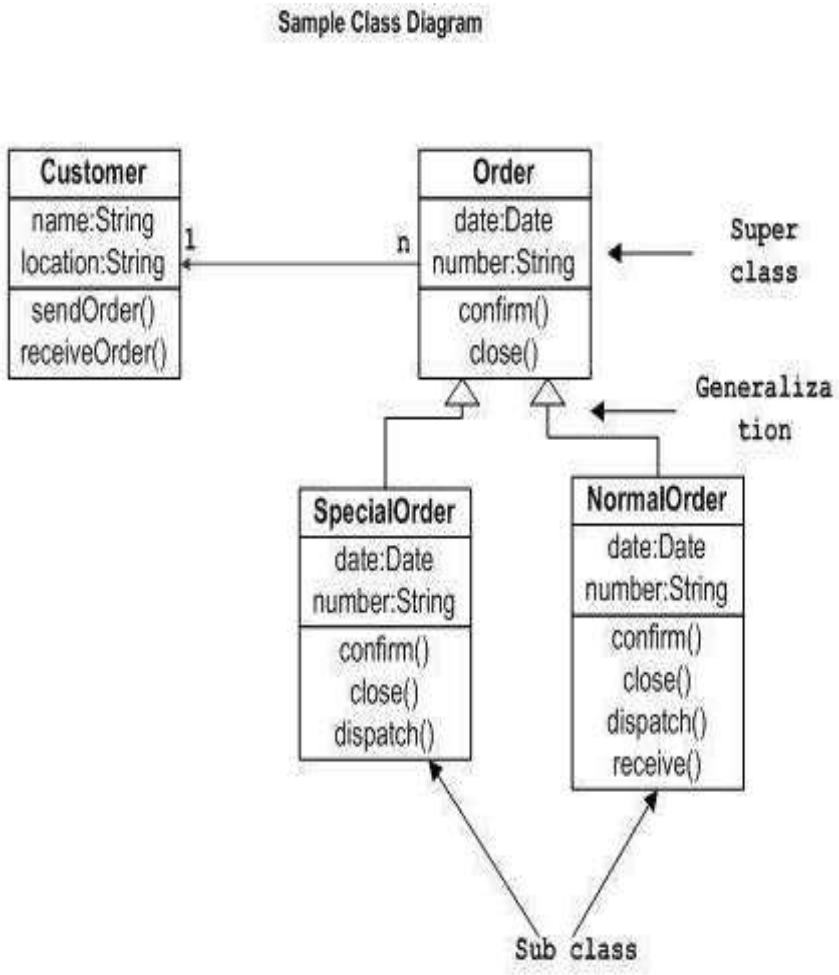
The following diagram is an example of an Order System of an application.
It describes a particular aspect of the entire application.

First of all, Order and Customer are identified : the two elements of the system.

They have a one-to-many relationship because a customer can have multiple orders.

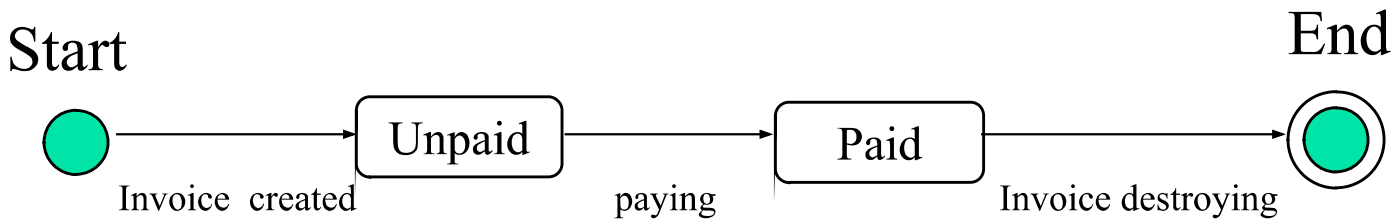
Order class is an **abstract class** and it has two concrete classes (inheritance relationship) **SpecialOrder** and **NormalOrder**.

The two inherited classes have all the properties as the **Order class**.
In addition, they have additional functions like **dispatch () and receive ().**

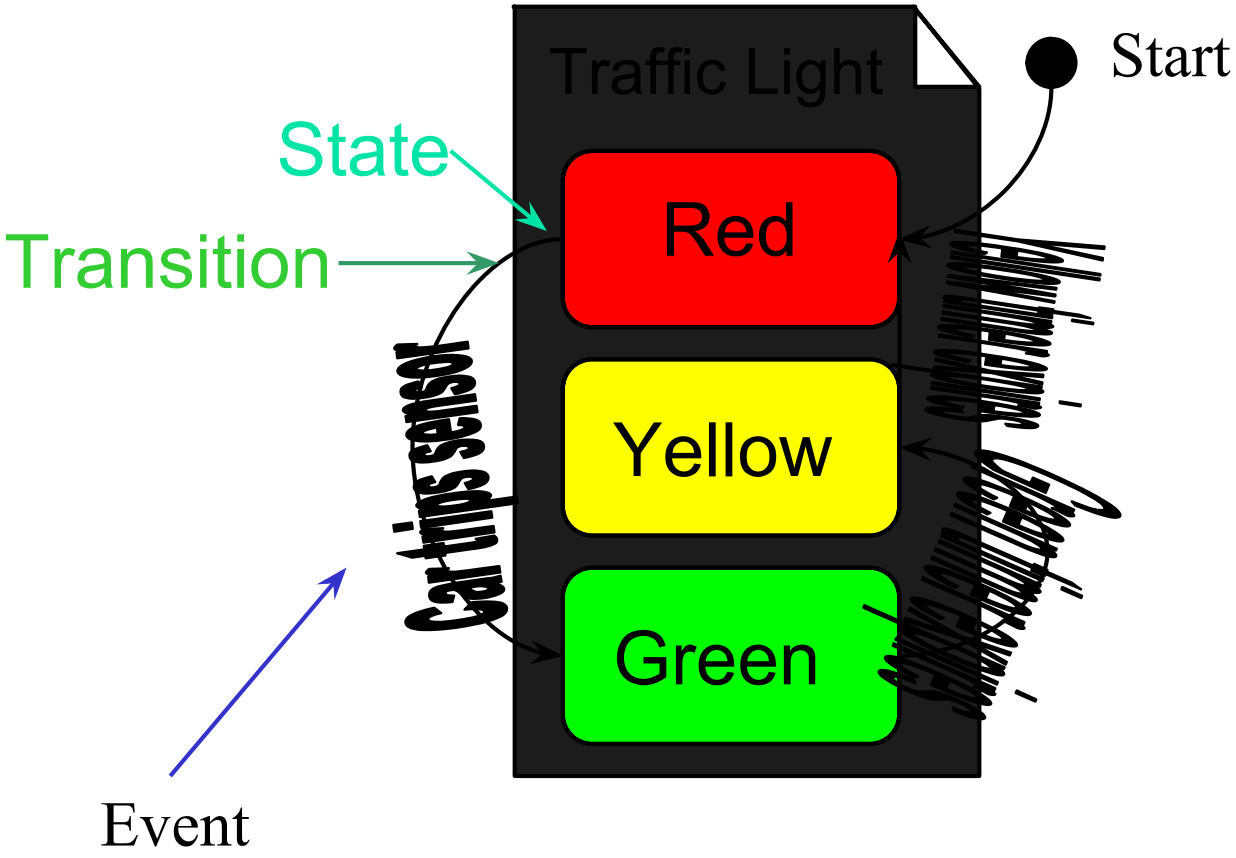


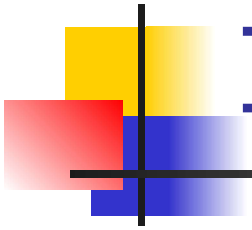
State Diagrams (Billing Example)

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)

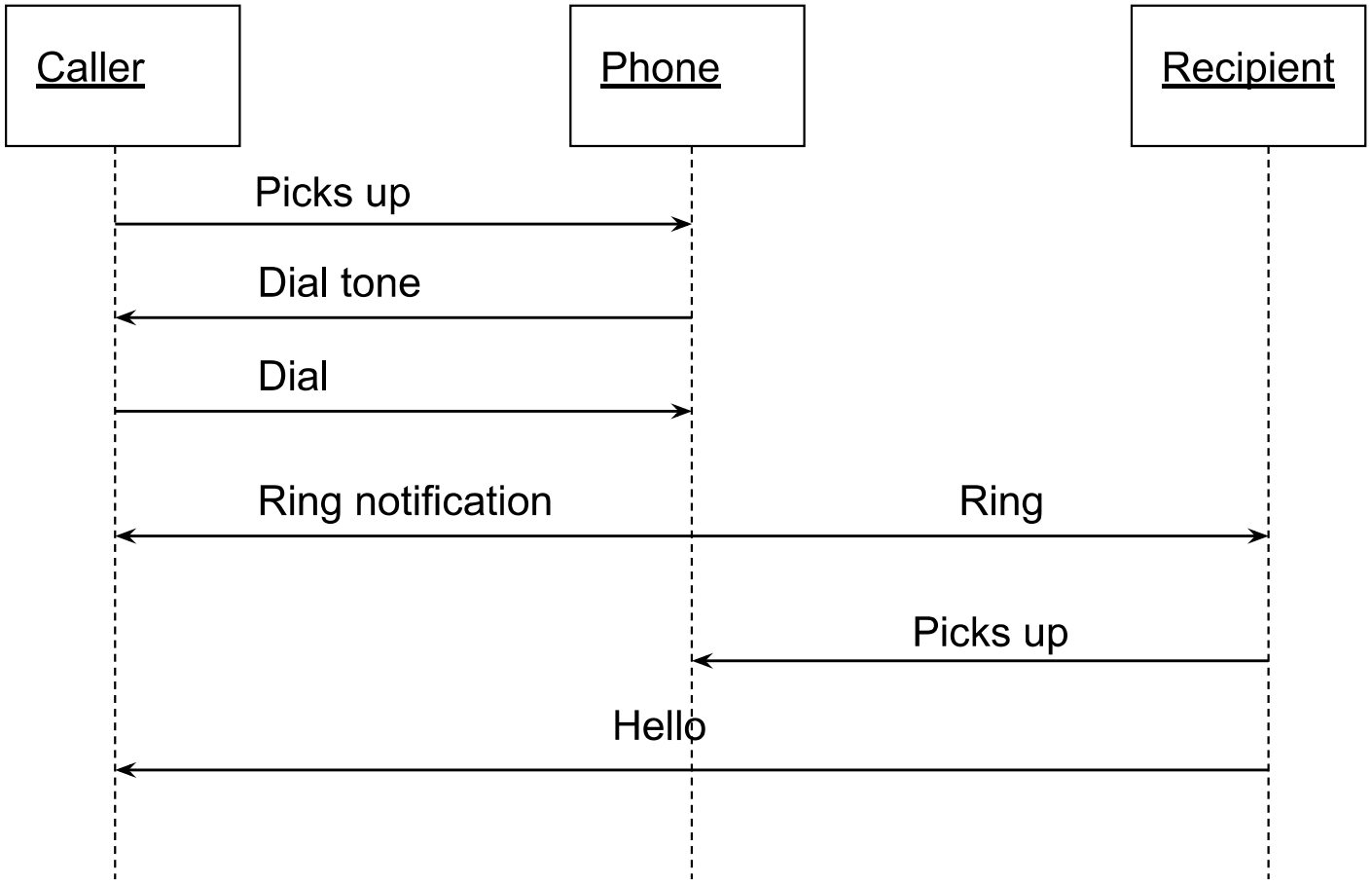




Interaction Diagrams

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

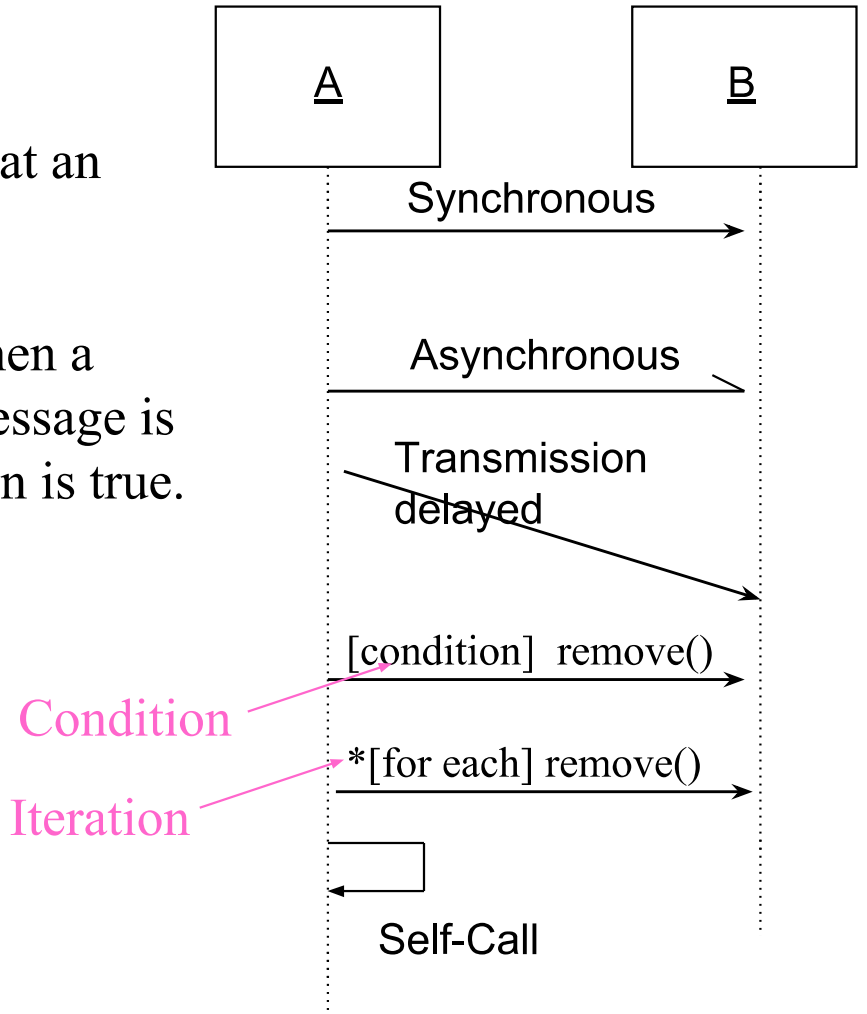
Sequence Diagram(make a phone call)



Sequence Diagram:Object interaction

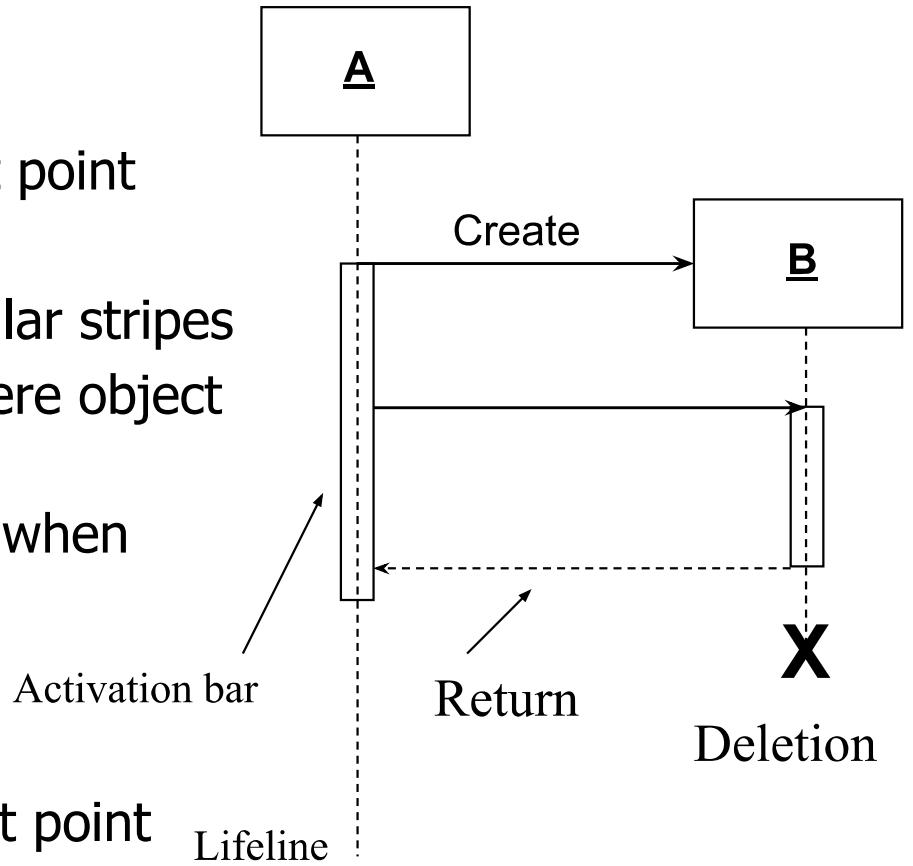
Self-Call: A message that an Object sends to itself.

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



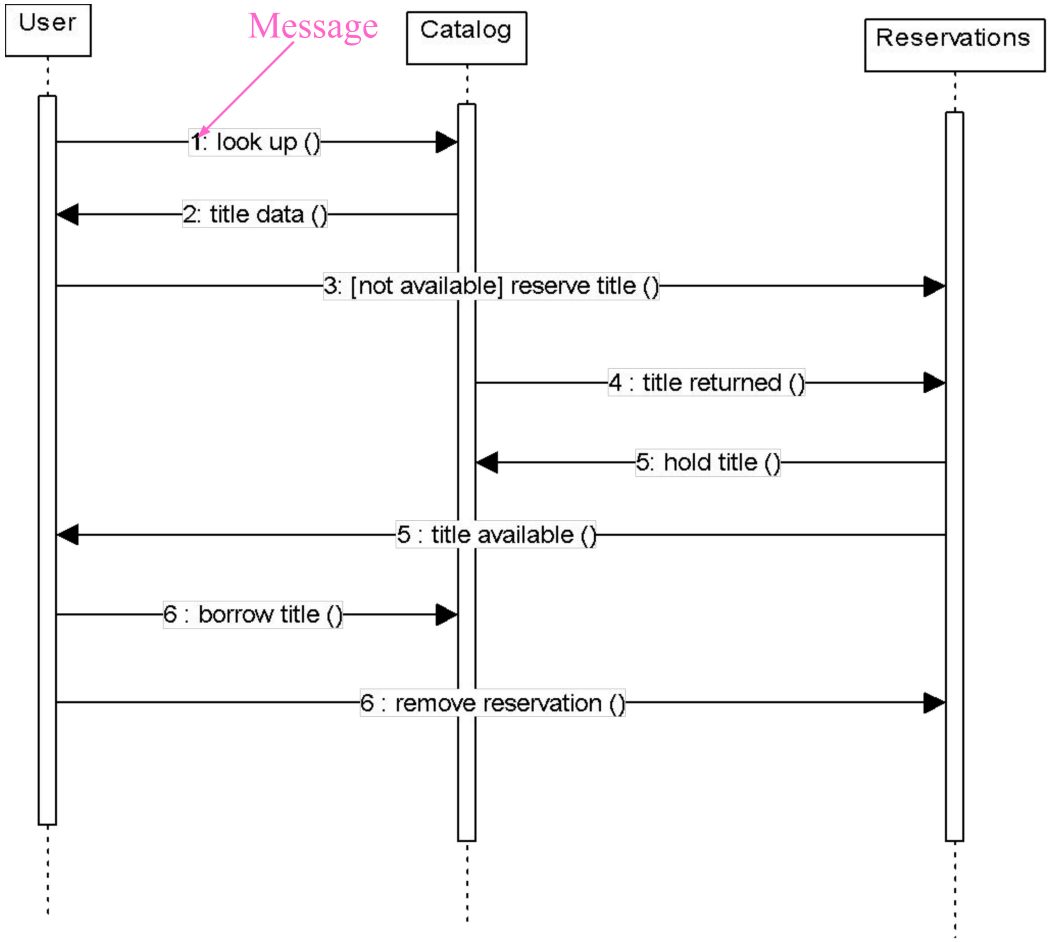
Sequence Diagrams – Object Life Spans

- 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

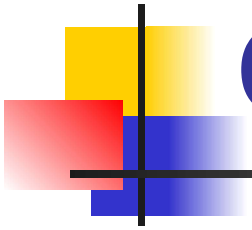


Sequence Diagram

- 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.

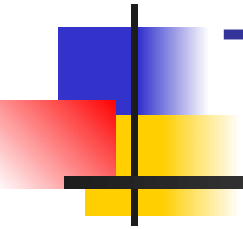


Classification	Types	Features
Structure Diagrams	Class Diagram	Structure of each class; relationships between classes
	Component Diagram	Components that make up the software and the dependencies between them
	Deployment Diagram	Physical layout of the system
	Package Diagram	Grouping of model elements such as classes and relationships between groups (packages)
Behavioral Diagrams	Use Case Diagram	Functions provided by the system, and relationships with external users and other systems
	Sequence Diagram	Interaction of objects along the time axis
	Collaboration Diagram	Objects interacting to implement some behavior within a context
	Statechart Diagram	Model life time of an object from creation to termination
	Activity Diagram	System operation flow



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?