Unified Modeling Language (UML)
Introduction and Use Case Diagram

Miaoqing Huang
University of Arkansas
Spring 2011

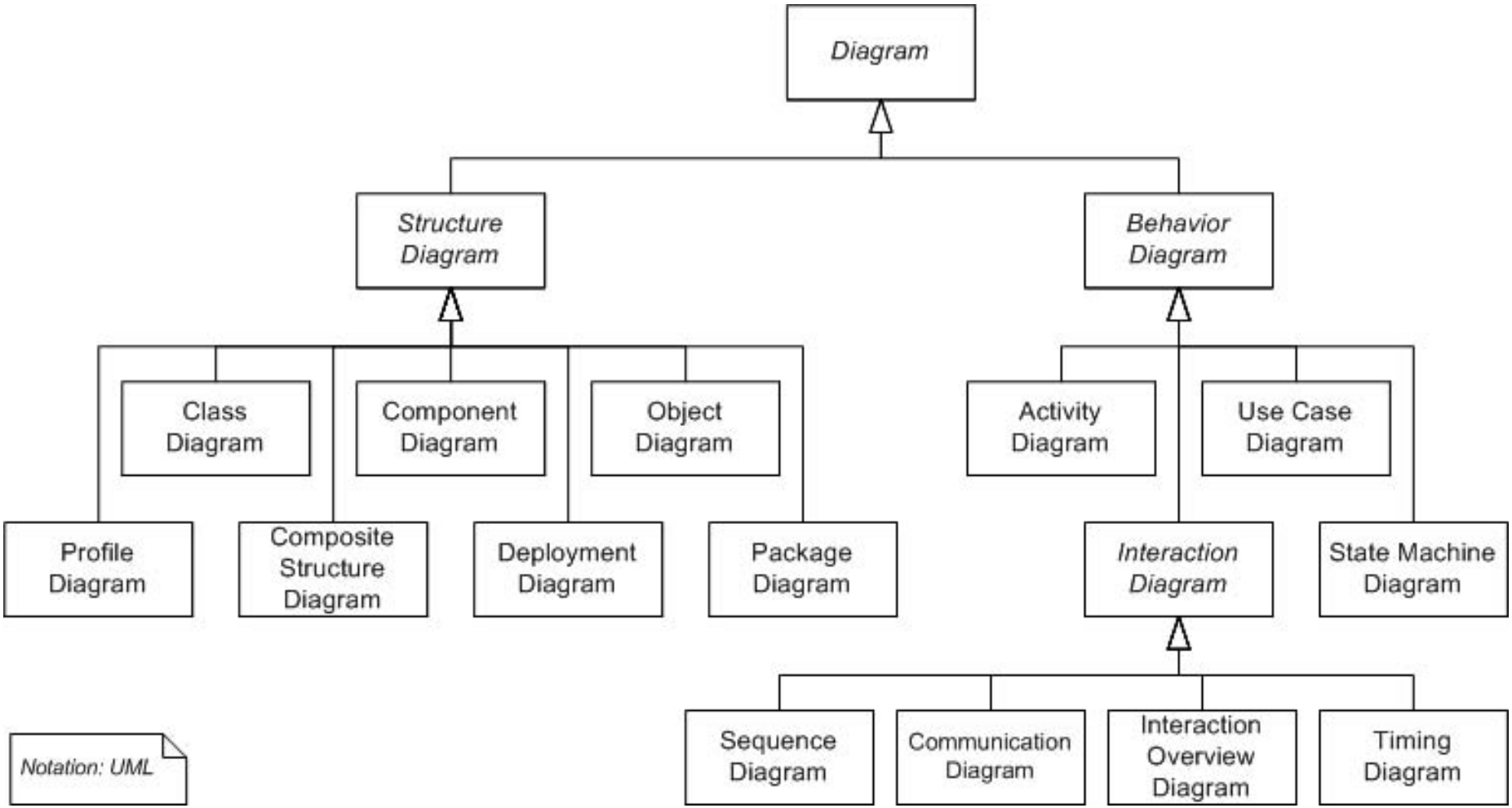
Outline

Outline

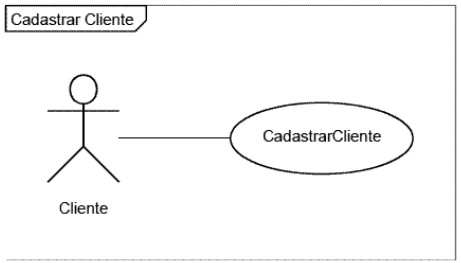
What is UML?

- UML is a modeling language
  - A model is an abstraction describing a system
  - Modeling language is used to express design
    - Use notation for depicting models
- UML is the de facto modeling language in software engineering
  - Created and managed by the Object Management Group, now at version 2.3 (May 2010)
  - Key persons: (three amigos) James Rumbaugh, Grady Booch, Ivar Jacobson
- UML includes a set of graphical notation techniques to create visual models of software-intensive systems
  - 14 diagrams
    - 7 structural diagrams
    - 7 behavior diagrams

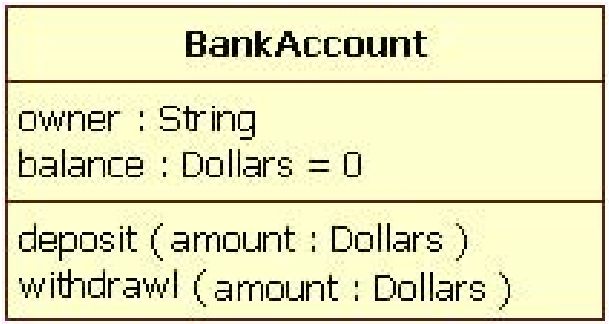
UML Diagrams



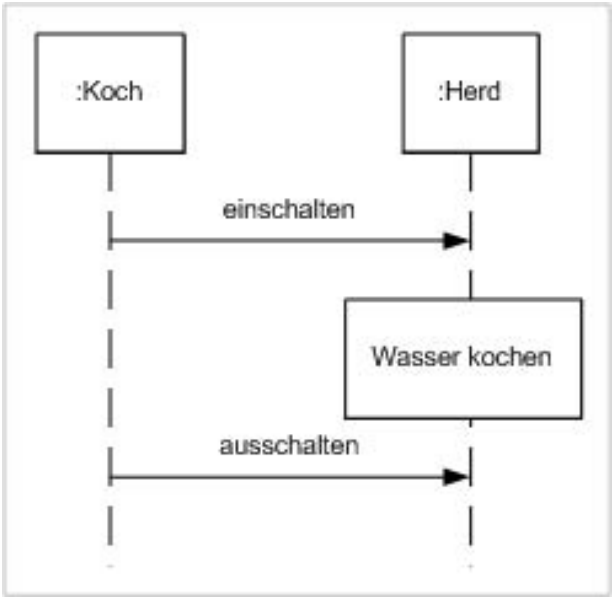
UML Diagrams – Example



use case diagram



class diagram



sequence diagram

UML Diagrams – Example

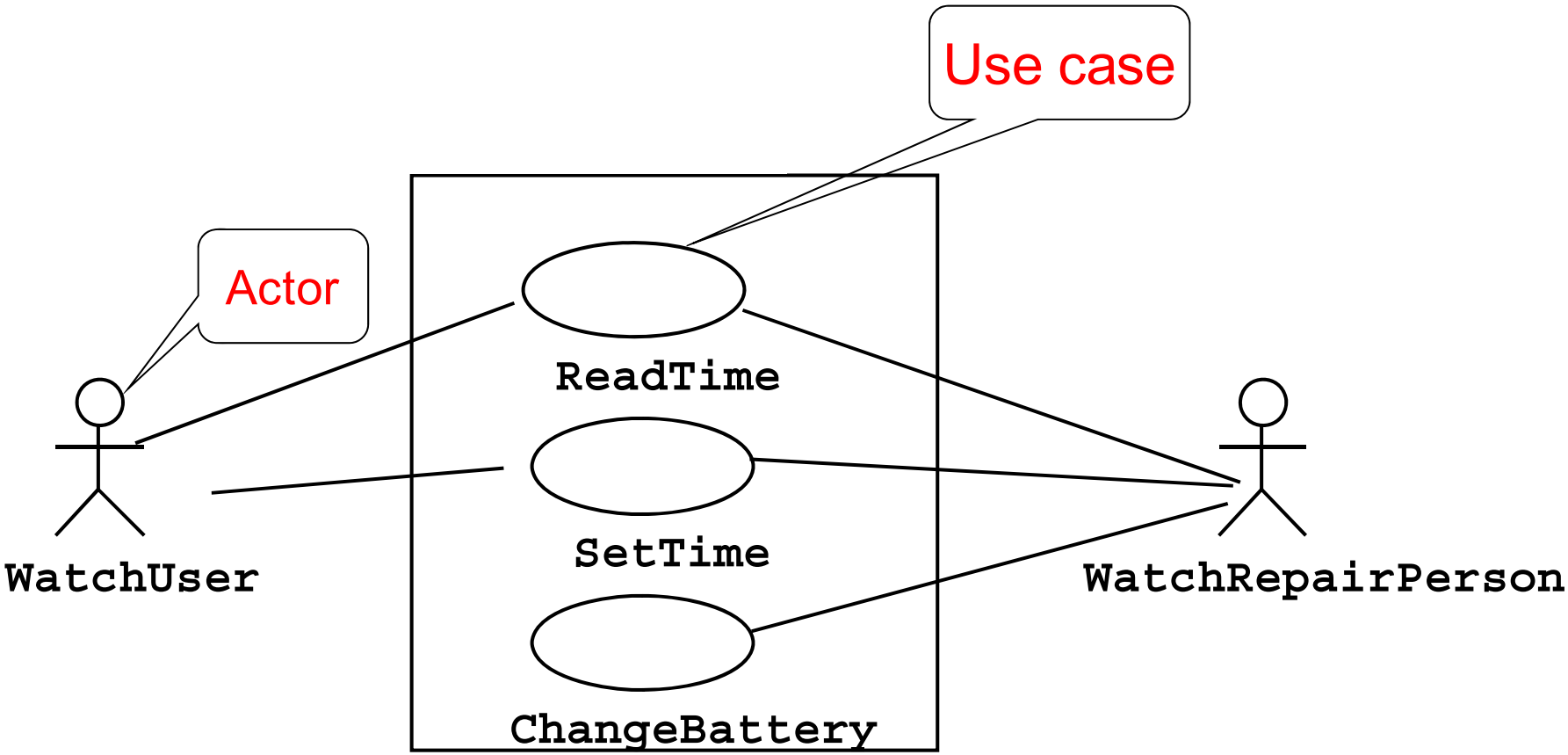
- Use case diagram
  - Describe the functional behavior of the system as seen by the user
- Class diagram
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagram
  - Describe the dynamic behavior between objects of the system
- State machine diagram
  - Describe the dynamic behavior of an individual object

UML Core Conventions

- All UML Diagrams denote graphs of nodes and edges
  - Nodes are entities and drawn as rectangles or ovals
  - Rectangles denote classes or instances
  - Ovals denote functions
- Names of Classes are not underlined
  - `SimpleWatch`
  - `Firefighter`
- Names of Instances are underlined
  - `myWatch:SimpleWatch`
  - `Joe:Firefighter`
- An edge between two nodes denotes a relationship between the corresponding entities

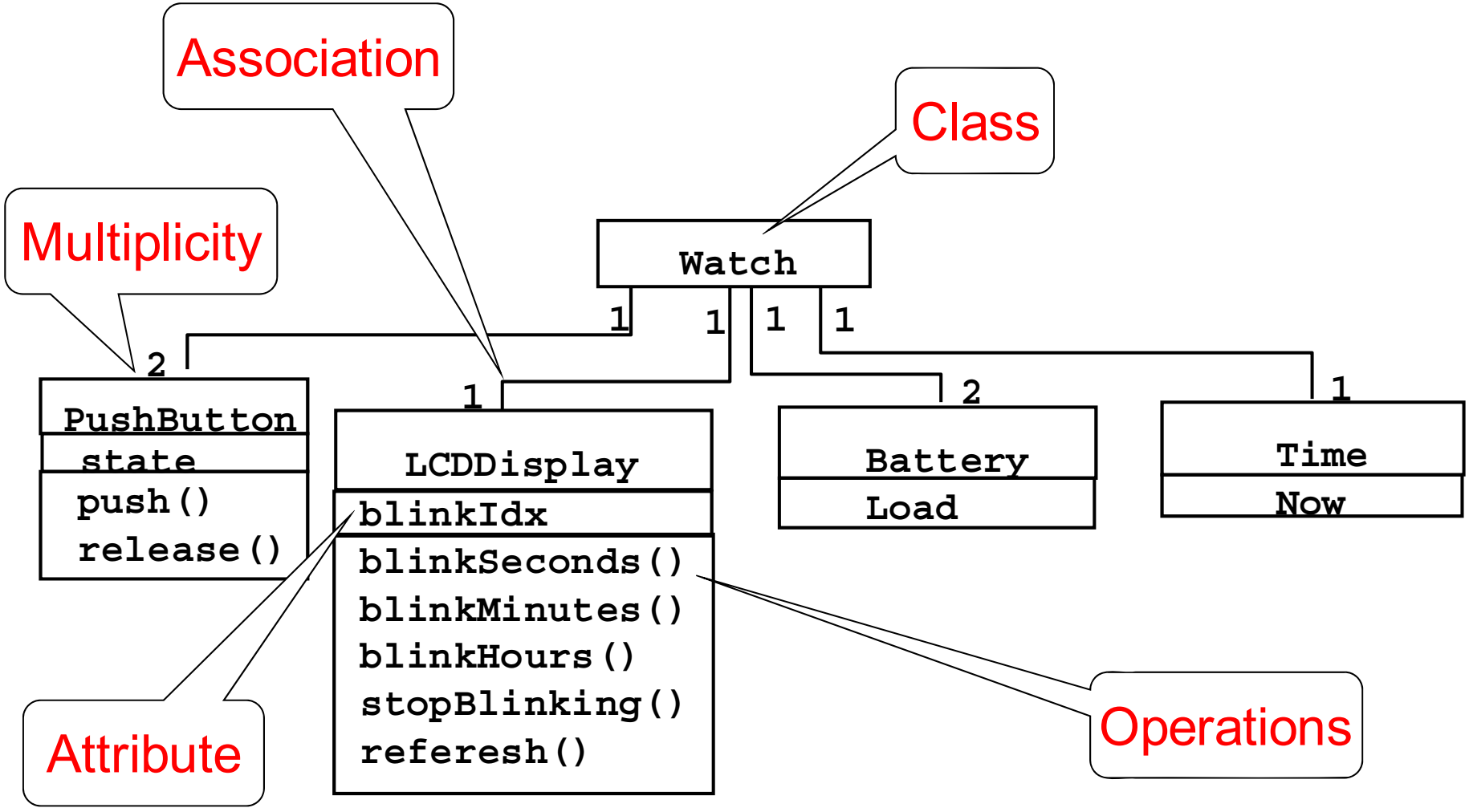
Use Case Diagram

- Use case diagrams represent the functionality of the system from user's point of view



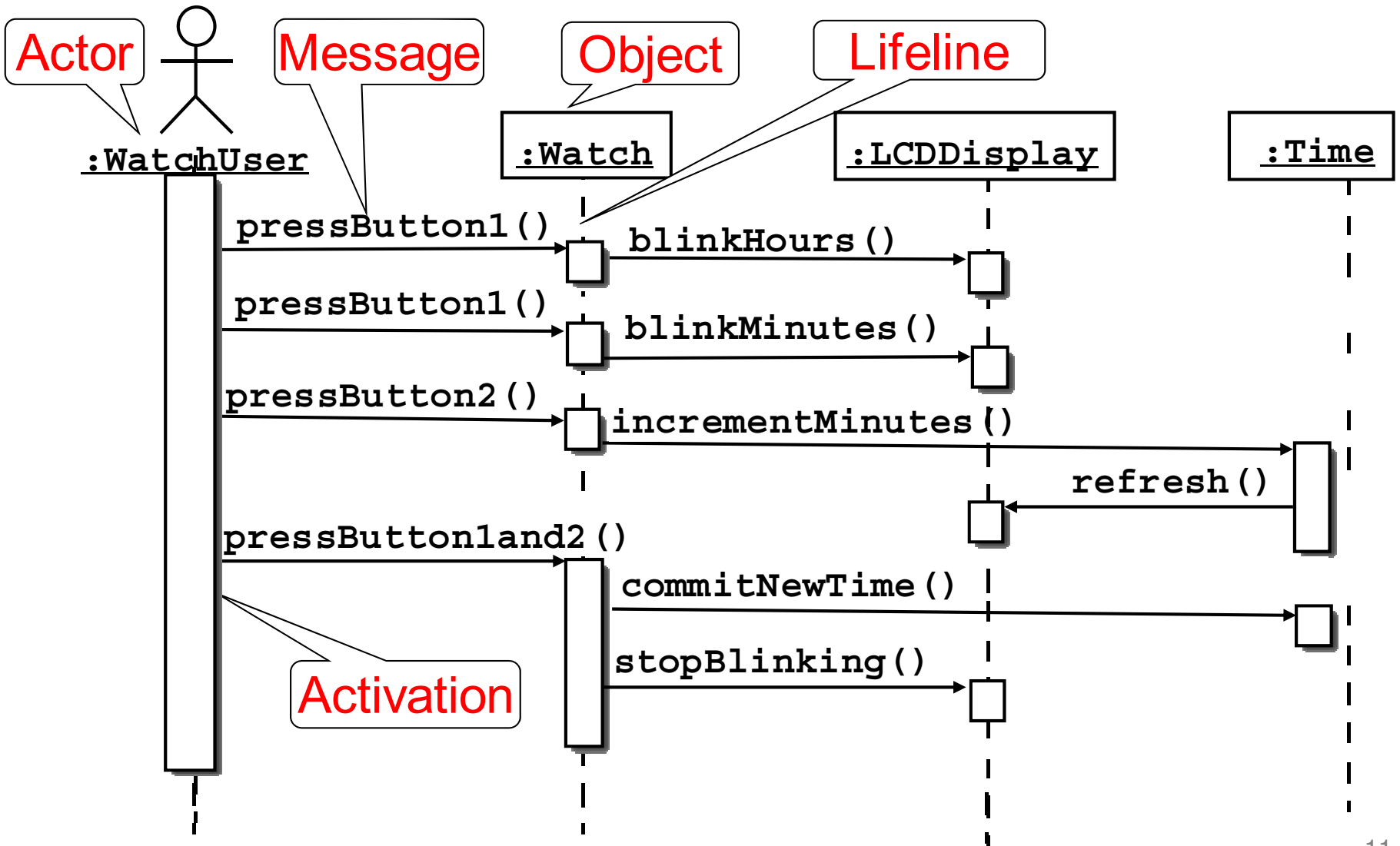
Class Diagram

- Class diagrams represent the structure of the system



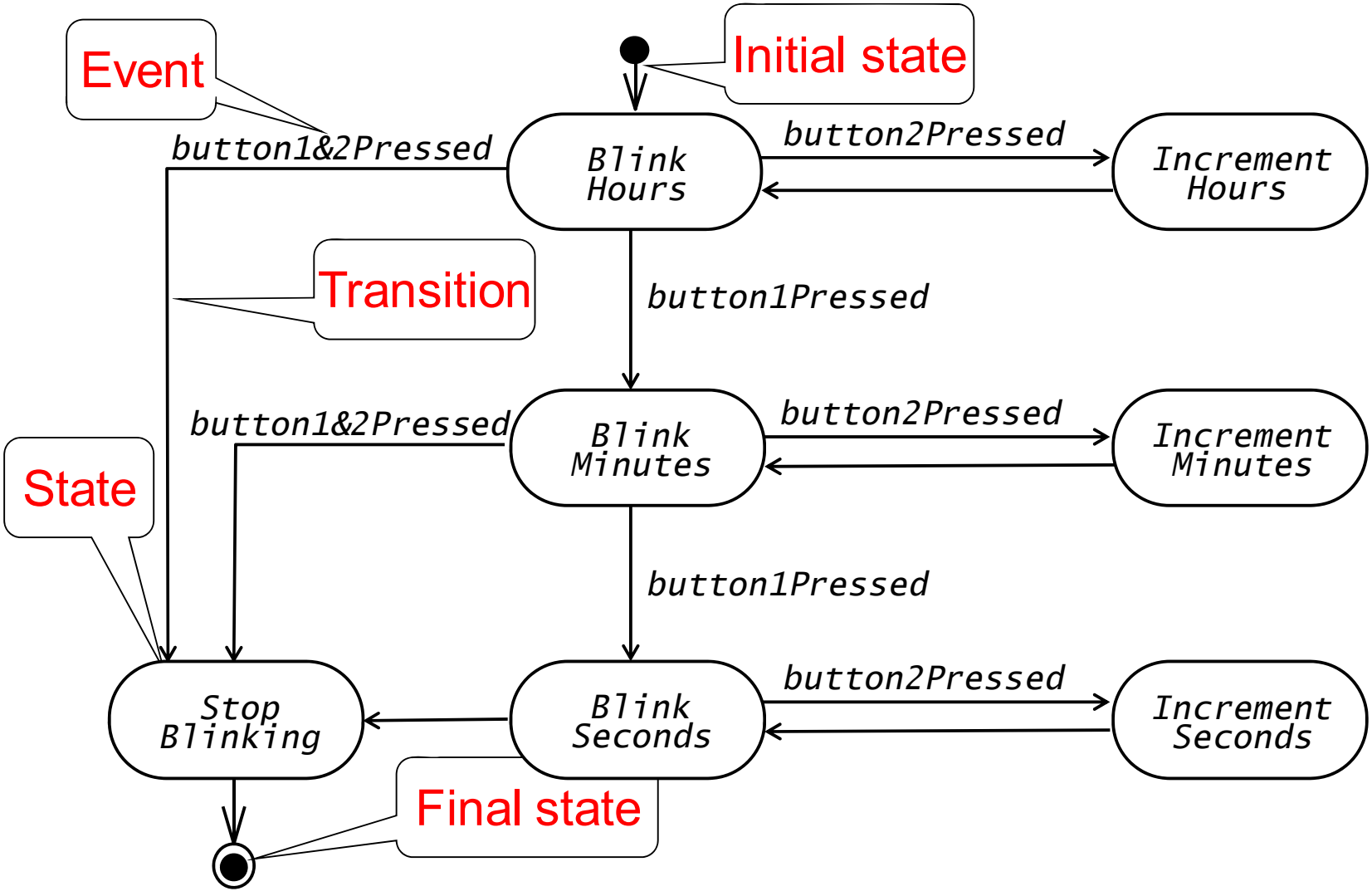
Sequence Diagram

- Sequence diagrams represent the behavior of a system as messages ("interactions") between *different objects*



State Machine Diagram

- State machine diagrams represents behavior of a *single object* with interesting dynamic behavior



Outline

What is a use case?

- Scenario
  - A sequence of steps describing an interaction between a user and a system

The customer browse the catalog and adds desired items to the shopping basket. When the customer wishes to pay, the customer describes the shipping and credit information and confirms the sale. The system checks the authorization on the credit card and confirms the sale both immediately and with a follow-up email.

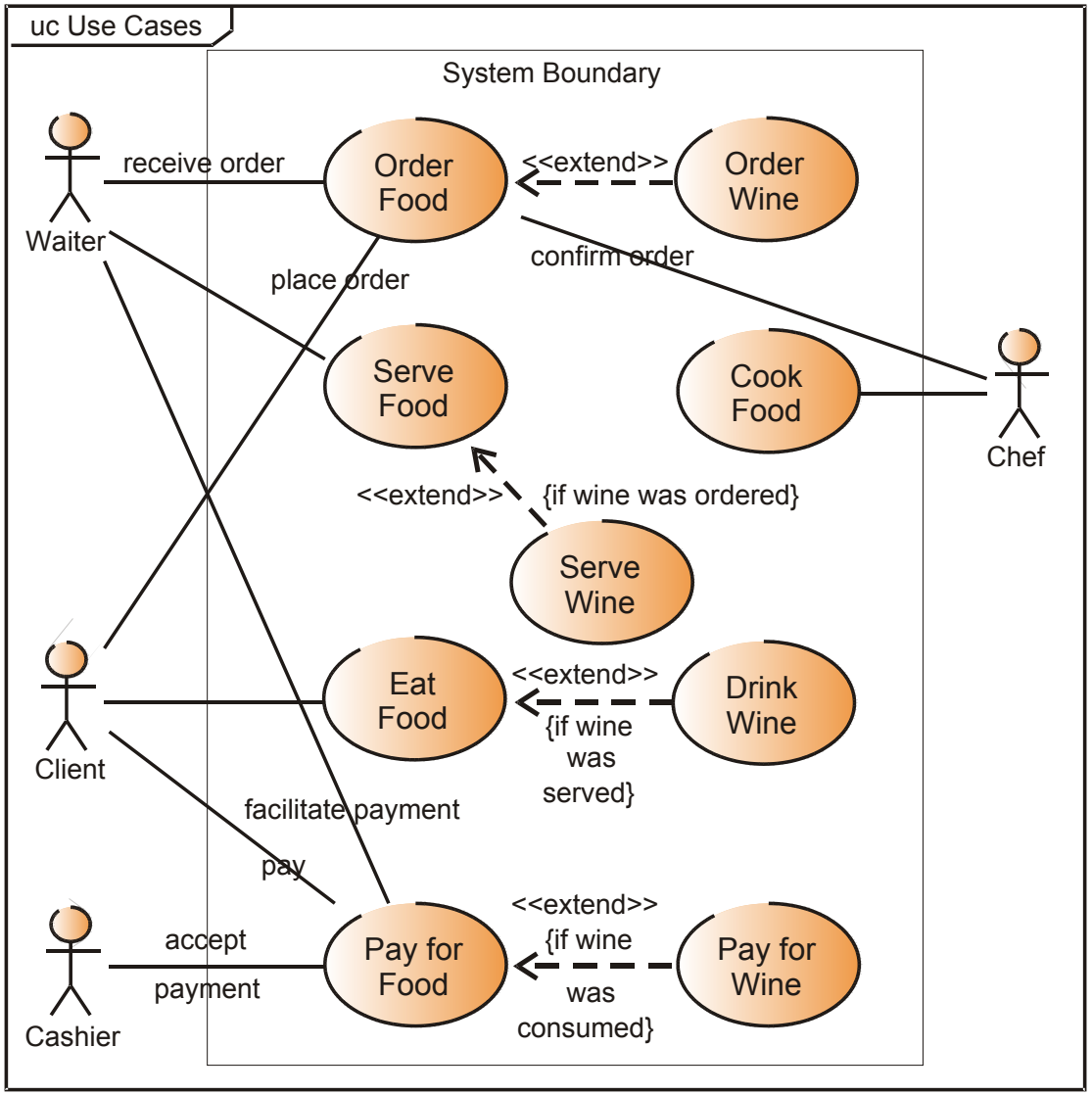
- Use case
  - A use case is a set of scenarios tied together by a common user goal
    - Buy a Product use case: a successful purchase or authorization failure

Describe a use case

A set of scenarios in the Use Case "Buy a product online":

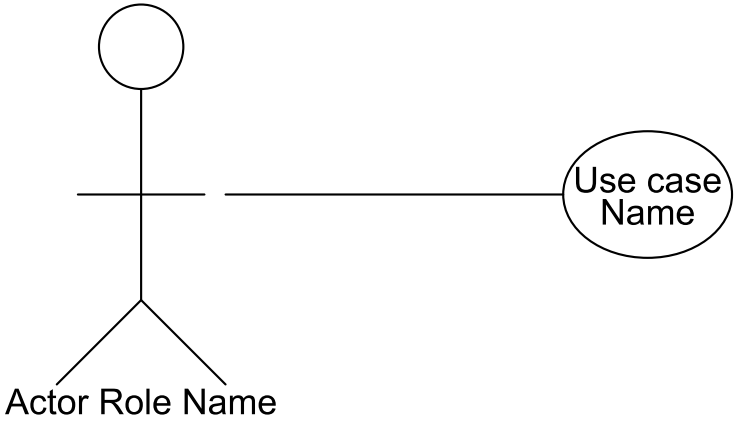
1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

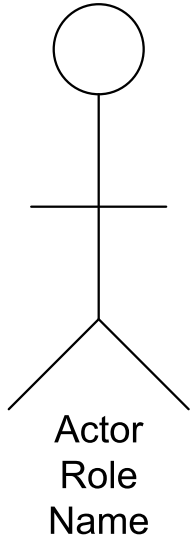
An Example


uc Use Cases
System Boundary
Waiter
receive order
Order Food
<<extend>>
Order Wine
confirm order
place order
Serve Food
Cook Food
Chef
<<extend>>
{if wine was ordered}
Serve Wine
Client
Eat Food
<<extend>>
{if wine was served}
Drink Wine
facilitate payment
pay
Cashier
accept payment
Pay for Food
<<extend>>
{if wine was consumed}
Pay for Wine

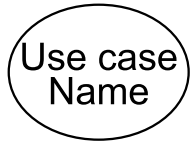
Actor and Use Case

- An actor is a role that a user play with respect to the system
- Actors are connected to the use cases by a line
  - A single actor may perform many use cases
  - A use case may have several actors performing it

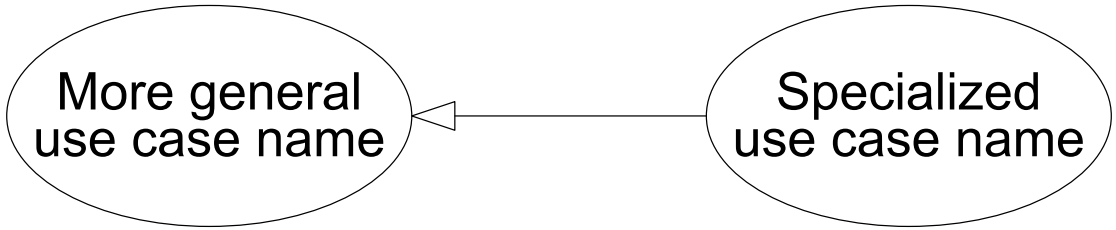




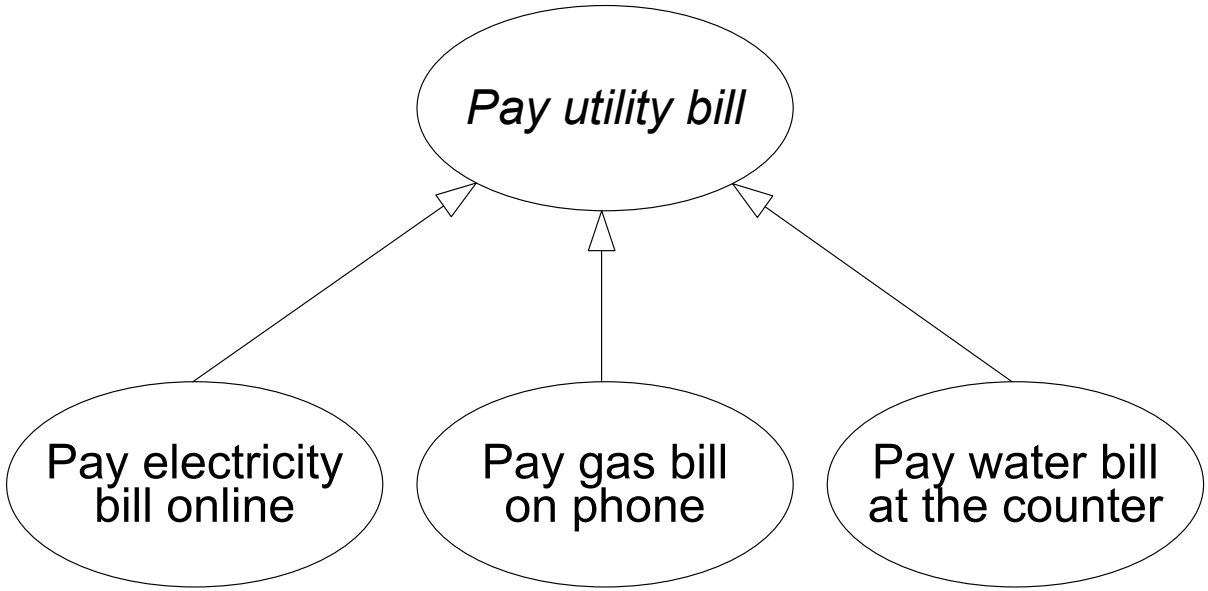
- Four types of relationships in use case diagram
  - Generalization between use cases
  - Generalization between actors
  - Include relationship between use cases
  - Extend relationship between use cases



Generalization between use cases

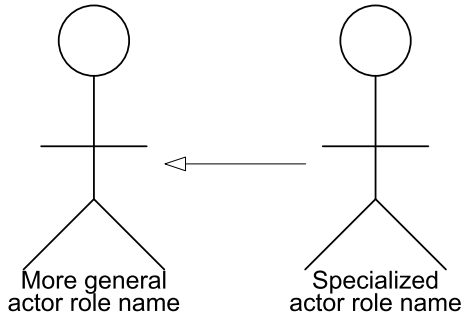


- Different versions of a use case share some actions in common and have some that are unique to each one
  - Generalized use case: *abstract use case*
    - It may never exist in a real system
  - Specialized use case: *concrete use case*



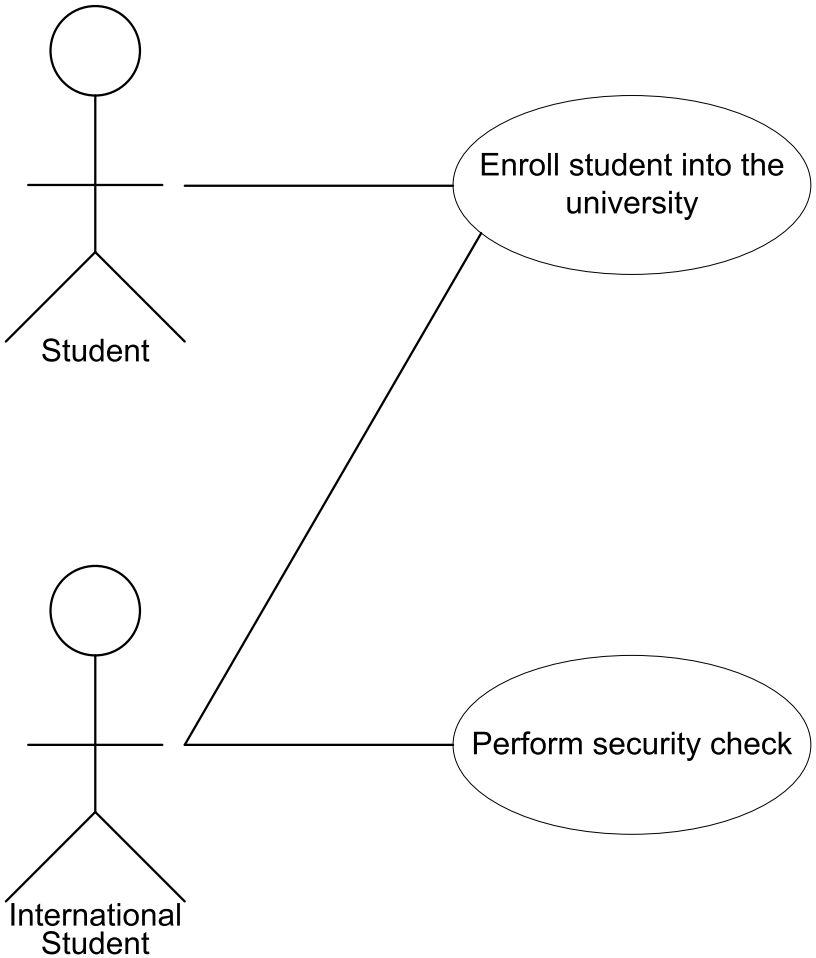
Generalization between Actors

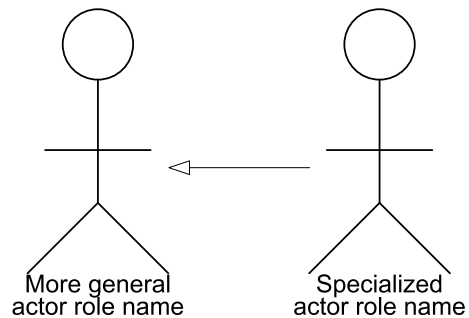
- Specialized actor can do everything the general actor can do, and *more*



Generalization between Actors

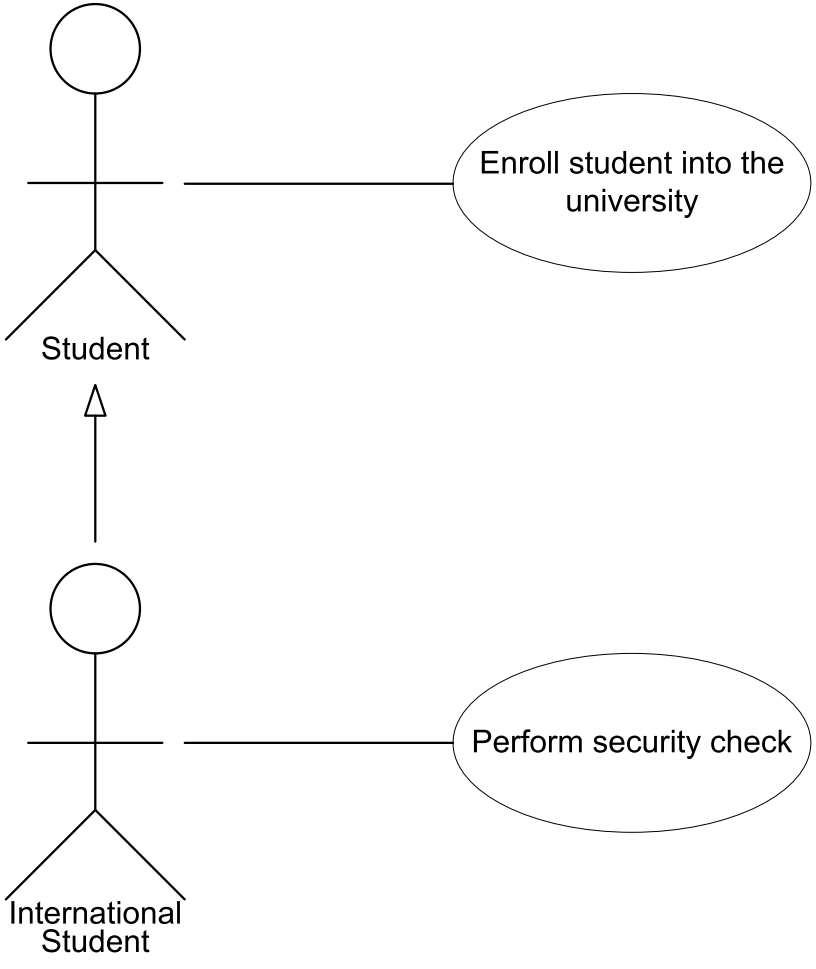
- Specialized actor can do everything the general actor can do, and *more*

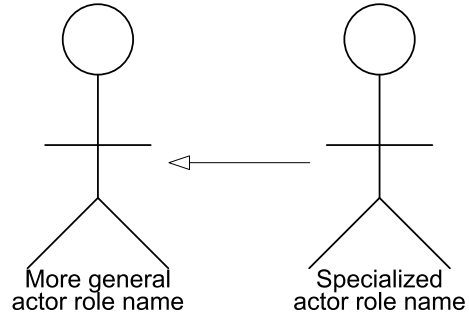




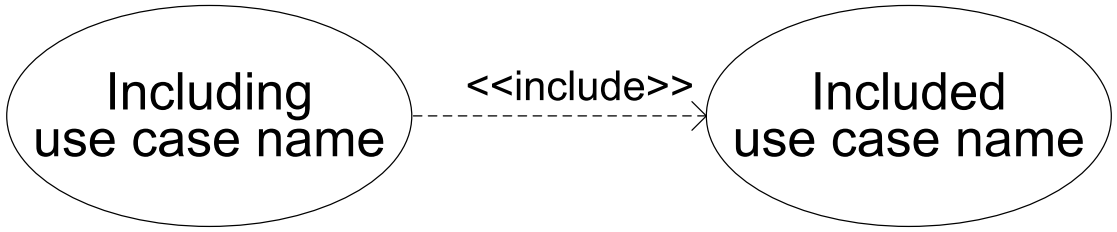
Generalization between Actors

- Specialized actor can do everything the general actor can do, and *more*

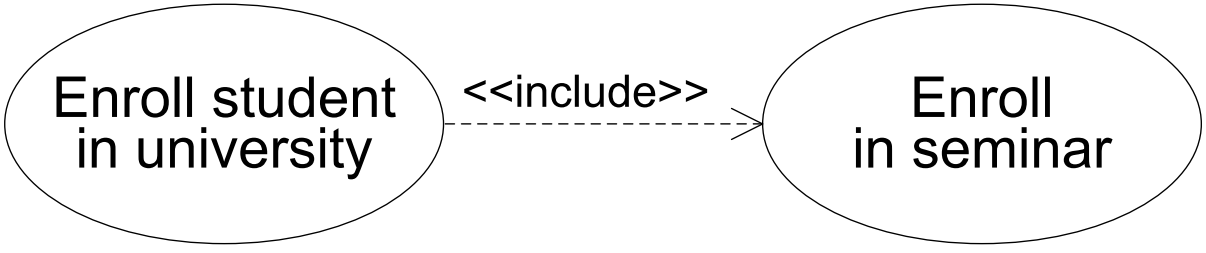




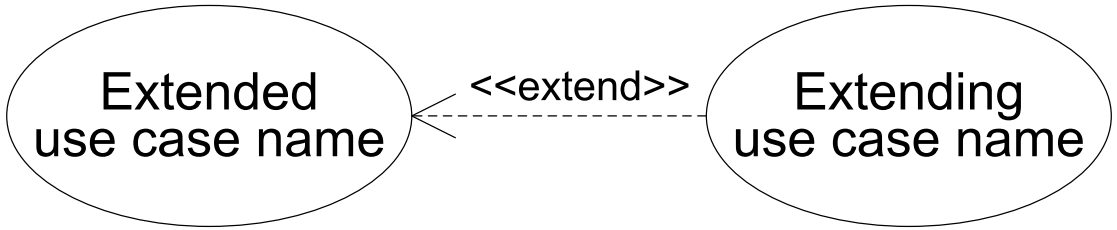
Include Relationship between Use Cases



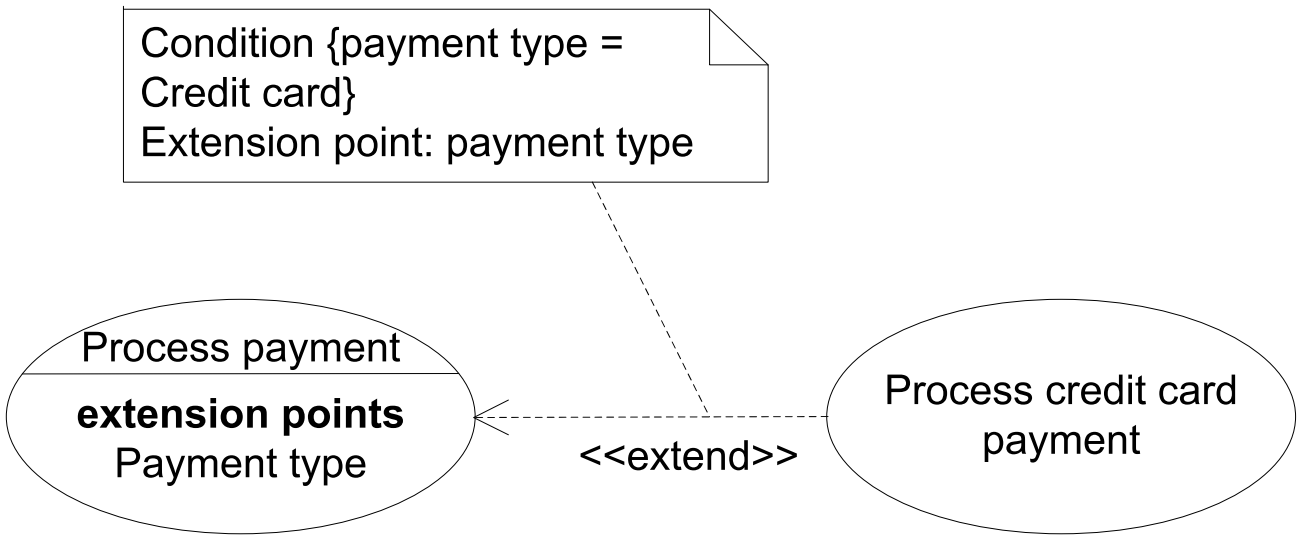
- One use case includes the functionality of another use case
  - Invocation of one use case by another one
    - e.g., calling a function or invoking an operation within source code



Extend Relationship between Use Cases



- One use case may be extended by the functionality in another use case



Include, Generalization, Extend

- Include
  - Use include when you are repeating yourself in two or more separate use cases and you want to avoid repetition
- Generalization
  - Use Generalization when you are describing a variation on normal behavior and you wish to describe it casually
- Extend
  - Use extend when you are describing a variation on normal behavior and you wish to use the more controlled form, declaring your extension points in your base use case