SOFTWARE ENGINEERING

Lecture 6: Use Cases Modeling (Continuation)

Prof. Mohamad Kassab   m.kassab@nyu.edu

LESSON OUTLINE

- What is a Use Case?
- Use Cases Modeling Elements.
- Use Cases Guidelines:
  - Complete Single Goal
  - Leveled Steps
  - Technology Neutral
  - Scenario Plus Fragments
  - Exhaustive Alternatives
  - Common Sub Behavior
  - Interrupts as Extension

What is a Use Case?

- The use case technique is used in software and systems engineering to capture the functional requirements of a system.
- A use case is a text story that is essentially a group of related **scenarios** that provide value to the user of a system.
- A **scenario** is one of many paths or a specific sequence within a use case.
- For example, an ATM system can have different functionalities (use cases) (e.g., withdrawal, deposit, etc.)
- One use case (e.g., withdrawal) can have multiple **scenarios**:
  - **Standard Withdrawal**
  - **Insufficient Funds**
  - **Incorrect PIN**
  - **Maximum Daily Withdrawal Limit Reached**

Use Case Example: Withdraw from ATM

Scenario 1 (Main Success Scenario): Standard Withdrawal

1. User inserts their ATM card into the machine.
2. ATM prompts the user to enter their PIN.
3. User enters the correct PIN.
4. ATM displays the main menu, and the user selects "Withdrawal" option.
5. User specifies the amount to withdraw.
6. ATM checks the account balance and ensures sufficient funds are available.
7. If funds are available, ATM dispenses the requested amount in cash.
8. ATM updates the account balance and prints a receipt.
9. User retrieves the cash and the receipt.

Use Case Example: Withdraw from ATM

Scenario 2 (Alternative Scenario): Insufficient Funds

1. User inserts their ATM card into the machine.
2. ATM prompts the user to enter their PIN.
3. User enters the correct PIN.
4. ATM displays the main menu, and the user selects "Withdrawal" option.
5. User specifies the amount to withdraw.
6. ATM checks the account balance and detects insufficient funds.
7. ATM displays an error message indicating insufficient funds.
8. User is prompted to either enter a lower amount or cancel the transaction.
9. If canceled, the user retrieves their card.

Use Case Example: Withdraw from ATM

Scenario 3 (Alternative Scenario): Incorrect PIN

1. User inserts their ATM card into the machine.
2. ATM prompts the user to enter their PIN.
3. User enters an incorrect PIN.
4. ATM displays an error message indicating an incorrect PIN.
5. User is given multiple attempts to enter the correct PIN.
6. After multiple failed attempts, the ATM retains the card.
7. User is prompted to contact their bank for assistance.

Use Case Example: Withdraw from ATM

Scenario 4 (Alternative Scenario): Maximum Daily Withdrawal Limit Reached

1. User inserts their ATM card into the machine.
2. ATM prompts the user to enter their PIN.
3. User enters the correct PIN.
4. ATM displays the main menu, and the user selects "Withdrawal" option.
5. User specifies the amount to withdraw, which exceeds the daily limit.
6. ATM displays an error message indicating the daily limit is reached.
7. User is prompted to enter a lower amount or cancel the transaction.

Why do we need use cases?

- **Gather requirements**
  - Establishes system boundary, a contractual view of all capabilities expected of the system
- **Communicate requirements**
  - Bridge gap between the business team and the development team
- **Manage requirements**
  - Project planning, scheduling and tracking
  - Controlled delivery of product
  - Planned Evolution
- **Verify requirements**
  - Complete list of verifiable functional requirements
  - Test plans are closely tied to use case scenarios

Use Case Modeling Elements

- The elements of use case modeling are:
  - Actors
  - Use case diagrams
  - Use case specifications

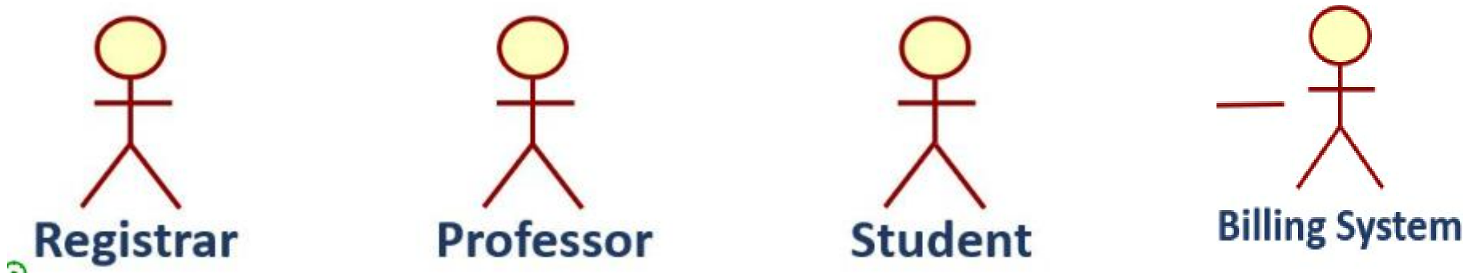
Example: University Registration System

A university wants to create an online registration system.

The university registrar sets up the curriculum, students register for courses and the professor requests course rosters.

Actors

- Actors are external entities that interact with a system.
- They define a role played by entities when they interact with a system.
- In our example we have the following actors:



Types of Actors

- In the context of use cases modeling, there are 3 types of actors:

- Primary Actors
- Supporting Actors
- Off-stage Actors

Primary Actors

- Primary actors are the main external entities that directly interact with the system to achieve a specific goal or perform a use case.
- They are essential for the success of the use case and have a direct interest in the outcomes.
- Primary actors are typically the users or entities initiating the use cases.

Supporting Actors

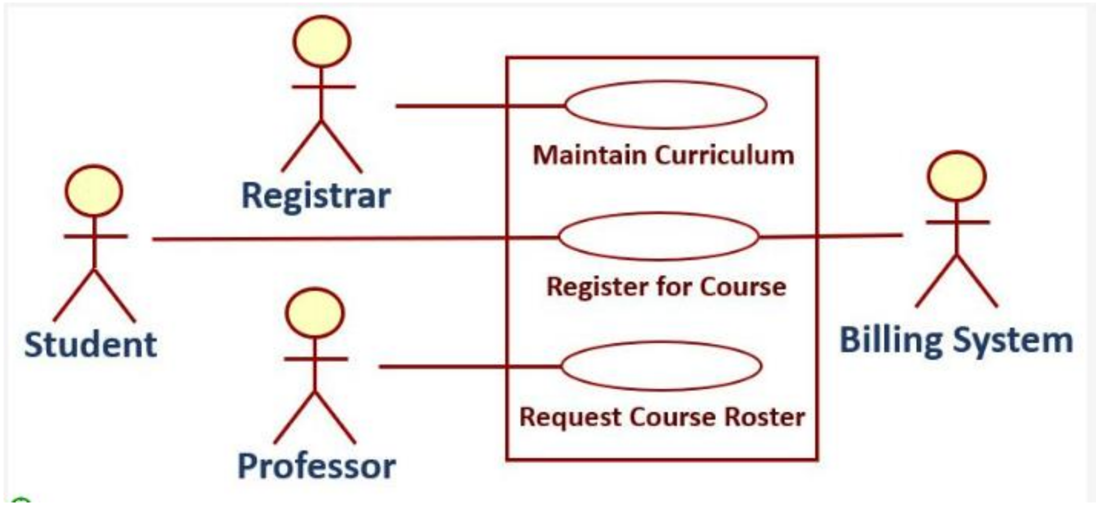
- Supporting actors are entities that provide a service to the system or assist in achieving the goals of a use case.
- While not the primary initiators, supporting actors play a supportive role and may interact with the system to facilitate certain processes.
- They are not the main focus of the use case but contribute to its execution.

Off-stage Actors

- Off-stage actors are external entities that have an interest in the system but do not directly interact with it in the context of a particular use case.
- They are relevant for understanding the broader context or external factors that may influence the system.
- Off-stage actors are often represented as icons outside the system boundary in a use case diagram.

Use Case Diagram

- Begin by depicting a system boundary using a rectangle. This boundary encapsulates the entire system.
- Place **primary** actors on the left side of the diagram and **supporting** actors on the right side of the diagram.
- Represent each use case as an oval within the system boundary.
- Ensure that the name of each use case starts with a verb, highlighting the action it describes.
- Connect the actors to the corresponding use cases using lines, indicating the interactions and relationships between actors and use cases.



Fully Dressed Specification: Use Case Specification

Use Case Specifications include the following items:

1. **Use Case Name**
2. **Scope**
3. **Level**
4. **Primary Actor**
5. **Stakeholders and Interests**
6. **Pre-Conditions**
7. **Success Guarantee (success end condition)**
8. **Main Success Scenario**
9. **Extensions**
10. **Special Requirements**
11. **Technology & Data Variation List**
12. **Frequency of Occurrence**
13. **Open Issues/miscellaneous**

Fully Dressed Specification: Use Case Specification

1. Use Case Name

- Start with a verb
- Example “Buy Item”

2. Scope

- The system under design

3. Level

- Choose either “user goal” or “subfunction”

Fully Dressed Specification: Use Case Specification

4. Primary Actor

- The principal actor that calls upon a system service to fulfill a goal

5. Stakeholders and Interests

- :Interests of the actors a use case must satisfy

6. Preconditions

- Conditions that must be true before a use case can be executed. Example
  - Registrar is identified and authenticated

Fully Dressed Specification: Use Case Specification

7. Success Guarantee

- Conditions that must be true after the use is executed and successful’ Example
  - For example; Curriculum for a semester is saved

Fully Dressed Specification: Use Case Specification

8. Main Success Scenario

- Focus on what not on the how
- Do not repeat pre-conditions
- **Example:**
  - 1. Registrar enters the semester
  - 2. System presents course catalog
  - 3. Registrar selects a course
  - 4. System prompts for course offerings
  - 5. Registrar assigns sections and times
  - Registrar repeats steps 3-5 until done
  - 6. System saves the semester curriculum

Fully Dressed Specification: Use Case Specification

9. Extensions

- Alternate scenario paths such as
    - Exception conditions (failure conditions)
    - Alternate scenarios of success
- Example
    - *a. At any time, system fails
        - 1. Registrar starts again from the point of failure
    - 2a. Incorrect semester
        - 1. System signals error and rejects entry

Fully Dressed Specification: Use Case Specification

10. Special Requirements

- Non-Functional requirements, quality attributes or constraints such as
  - Static and dynamic performance requirements
  - Usability
  - Design constraints
- Example
  - Registrar must be able to create curriculum offline and upload it into the system when ready

Fully Dressed Specification: Use Case Specification

11. **Technology & Data Variation List**

- Variations in technology
- Variations in data

12. **Frequency of Occurrence**

- When & how often a use case will be used
- Min,max and average
- Number of times per actor per day
- Example: This activity occurs for a week 4 months prior to the beginning of a semester. 10 concurrent users create curriculum consisting of 900 course sections

13. **Open Issues/Miscellaneous**

- Dynamic list of current unresolved issues

Partially Dressed Specification: Use Case Specification

What is a Partially dressed use case?

- A partially dressed use case is often seen as a simplified or abbreviated version of a fully dressed use case.
- It may only contain the **use case title**, **main success scenario**, and **extensions** sections only.

Useful Use Case Guidelines

1. Complete Single Goal
2. Leveled Steps
3. Technology Neutral
4. Scenario Plus Fragments
5. Exhaustive Alternatives
6. Common Sub Behavior
7. Interrupts As Extensions

1. Complete Single Goal

- Improper goals will leave the writers deciding where one use case ends and another begins.

- Write each use case to address one complete and well-defined goal.

- That goal may be at any level in an Ever Unfolding Story

2. Leveled Steps

- Excessively large or excessively small use case steps obscure the goal and make the use case difficult to read and comprehend.
- Each scenario should have 3 to 9 steps. Ideally, they are all at similar levels, a level of abstraction just below the use case goal.

Use Case Horror: Small steps

- **USE CASE:** *Purchase Goods*
- **Primary Actor:** *User* – Customer wanting to make purchase.
- **Main success scenario:**
- 1. System asks user for first name.
- 2. User enters first name.
- 3. System asks for middle initial.
- 4. User enters middle initial.
- 5. System asks for last name.
- 6. User enters last name.
- 7. System asks for first line of street address.
- ... (etc....!!!)

Too long and tedious!

Use Case Horror: Large steps

- **USE CASE:** *Purchase Goods*
- **Primary Actor:** *Visitor* – Customer wanting to make purchase.
- **Main success scenario:**
- 1. Visitor enters all the customer, product and purchase information.
- 2. System presents visitor with final sums, charges credit card and delivers packing list to the shipping department.

The scenario doesn't tell us anymore than we can deduce from the name!

Use Case Horror: Mixed steps

USE CASE: *Purchase Goods*

Primary Actor: *User* – Customer

Main success scenario:

1. The system displays the Login screen.
2. *User* enters a username and password.
3. The system verifies the information.
4. The system sets access permissions.
5. The system displays the Main screen.
6. *User* does one or more of the following: *Place Order, Return Product, Cancel Order, Send Catalog, Register Complaint*.
7. The system terminates the session and reset the screen.

Step 6 is at a vastly different scale than steps 1-5

Solution: Aim for 3-9 steps

USE CASE **:** *Purchase Goods*

Primary Actor: *Visitor* – Customer wanting to make purchase.

Main success scenario:

1. Visitor enters customer information (name, address, etc.).
2. System retrieves customer's profile information, and presents product search and selection mechanisms.
3. Visitor selects products until satisfied. After each selection, system adds the selected product to the customer's "shopping cart" and presents the ongoing running total of products selected.
4. Visitor selects to purchase the selected items.
5. System presents contents of shopping cart, requests customer's payment information.
6. Customer enters manner of payment and other payment details.
7. System presents visitor with final sums, charges credit card and delivers packing list to the shipping department.

3. Technology Neutral

- Including technology constraints and implementation details in a use case description increases the complexity and obscures the goal of the use case.
- Write each use case in a technology neutral manner.

Use Case Horror: Too Much Technology

- **USE CASE:** *FILE ACCIDENT CLAIM*
- **Primary Actor: Claimant** – policyholder reporting accident claim
- **Main success scenario:**
- 1. **Claimant** accesses Accident Reporting System
- 2. **Claimant** identifies self to system using name and policy number
- 3. **System** verifies claimant owns a valid policy against policy database.
- 4. **Claimant** submits claim with substantiating data including name of both parties, investigating officer, and citation number, if applicable
- 5. **System** logs claim file and policy database, and sends email acknowledgement to the claimant

This example contains at least three flaws:

1. It presupposes the system will store policy information in a database.

2. It presupposes the system will store claim information in a data file.

3. It presupposes the system will send email to acknowledge a claim.

Solution: Technology Neutral

USE CASE: *FILE ACCIDENT CLAIM*

Primary Actor: Claimant – policyholder reporting accident claim

Main success scenario:

1. **Claimant** accesses Accident Reporting System
2. **Claimant** identifies self to system using name and policy number
3. **System** verifies claimant owns a valid policy.
4. **Claimant** submits claim with substantiating data including name of both parties, investigating officer, and citation number, if applicable
5. **System** logs claim and acknowledges its receipt to the claimant
6. **Requestor**: mark request delivered.

4. Scenario Plus Fragments

- The reader must be able to easily follow the path through the specific scenario or story that they are interested in, otherwise they are likely to become frustrated or miss important information.

- Write the success storyline as a simple scenario without any consideration to possible failures. Below it, place story fragments that show what alternative condition might have occurred.

5. Exhaustive Alternatives

- A use case may have many alternatives. Missing some means the developers will misunderstand the system's behavior and the system will be deficient.

- Capture all failure and alternatives that must be handled in the use case.

A comprehensive use case

Use Case: *Access Email*

Main success scenario

1. User initiates use case by logging on to the system and requesting email.
2. Server displays user's email screen.
3. User and server repeat the following sequence indefinitely.
4. User selects email.
5. Server display email, and marks it read.
6. User selects "quit" option.
7. Server updates user's email account information and logs off user.

Alternate Courses

<u>User saves email</u>

4. User selects "Save Email" option.
5. Server marks email as unread.

<u>User replies to email</u>

4. User selects "reply" option.

5a. Server prompts user for a reply.
5b. User enters reply and instructs server to send email.
5c. Server sends email to address of message's originator.

<u>User forwards email to another user</u>

4. User selects "forward Email" option.

5a. Server requests address.
5b. User supplies information and instructs server to send email.
5c. Server forwards email to specified address.

Error Courses

<u>Invalid Login</u>

1. User supplies incorrect login id or password.
2. Server displays error message and initiates a new login sequence.

6. Common Sub Behavior

- Writing the same steps for different use cases is wasteful and makes it more difficult to see common sub-processes within a use case model.
- Express shared courses of actions with <<included>> use cases.

Common Sub Behavior (Example)

Use Case 1: Book a Flight

Description:

1. User selects "Book a Flight" option.
2. User provides travel details including departure and destination airports, travel dates, and passenger information.
3. System checks for available flights based on user input.
4. User selects a preferred flight option.
5. System prompts the user for payment information.
6. **Call *Collect Payment Information***

Common Sub Behavior (Example)

Use Case 2: Book a Cruise

Description:

1. User selects "Book a Cruise" option.
2. User provides cruise details such as destination, departure date, cabin preferences, and passenger information.
3. System displays available cruise options.
4. User selects a preferred cruise package.
5. System prompts the user for payment information.
6. **Call *Collect Payment Information***

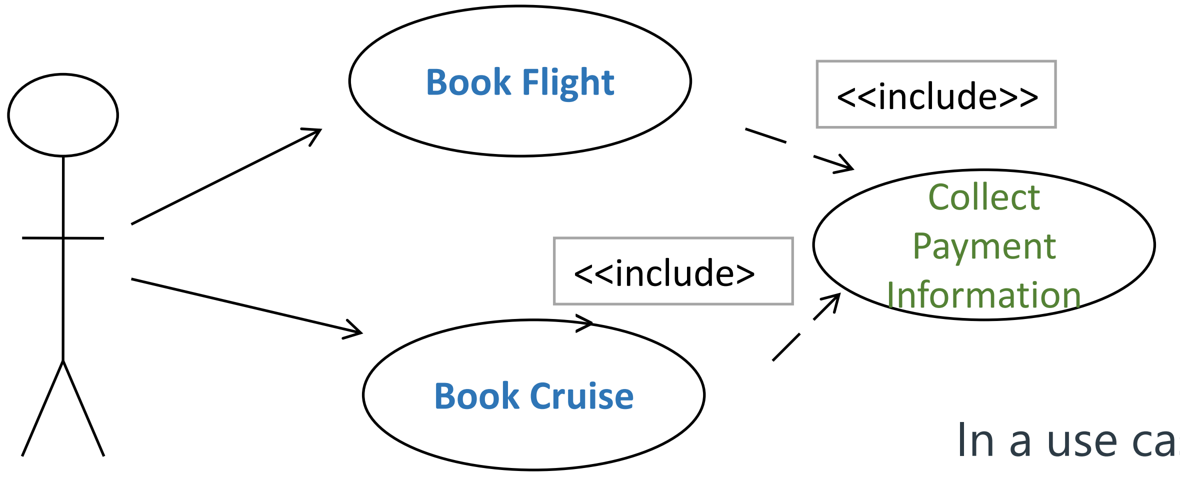
Common Sub Behavior (Example)

Use Case 3: Collect Payment Information

Description:

1. User initiates a payment process (e.g., during flight or cruise booking).
2. System prompts the user for payment information.
3. User provides payment details such as credit card information.
4. Payment Gateway validates and processes the payment.
5. System updates the booking status based on the payment result.
6. If payment is successful, the system confirms the booking; otherwise, it provides an error message.

Common Sub Behavior



In a use case diagram, the "**<<include>>**" stereotype is used to indicate that one use case includes the functionality of another use case.

It signifies a relationship between the base use case (the included one) and the inclusion use case (the one doing the including).

Don’t Do This!

Use Case: Book Flight:

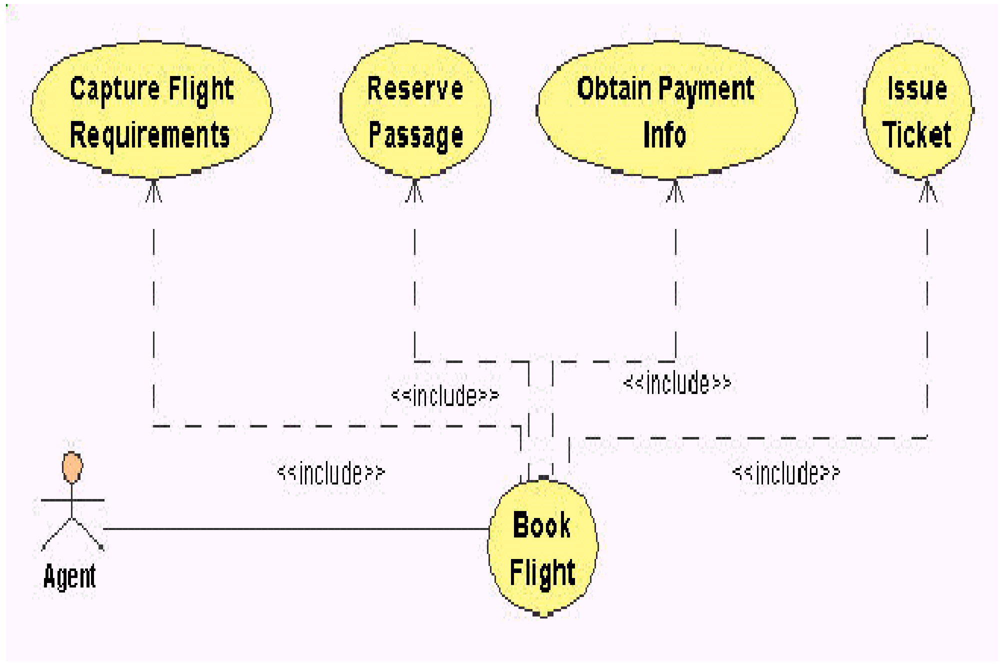
1. The use case begins when the agent selects set travel itinerary. Call *Capture Flight Information* to get the travel itinerary.

2. The agent chooses select flight. Call *Reserve Passage* to reserve a flight segment.

3. The agent chooses book flight, Call *Obtain Payment Information* to get payment choice.

4. Call *Issue Ticket* to print ticket.

Misuse of <<include>> to create a director style use case.



7. Interrupts As Extensions

- An alternative that affects more than one step in a scenario can scatter related details throughout a use case, causing the reader to become confused or lose track of important information.

- Create an extension use case when an alternative course of action interrupts a number of steps in a scenario.

Use case horror: Confusing alternatives

***Use Case: Book Flight*:**

1. The use case begins when the agent specifies a travel itinerary for a client.
2. The system searches a set of appropriate flights and presents them to the agent.
3. The agent chooses select flight.
4. The system verifies space is available on the flight and reserves a seat on the flight.
5. The agent finalizes the booking by supplying payment information.
6. The system books the seats, and issues the ticket.

Alternatives:

2a: Client is frequent flier:

2a1 The system retrieves the client's profile and displays the flights sorted by client's airline preference.

4a: Client is a frequent flier:

4a1 The system offers premium seats for the client.

4b: Seat is not available in ticket category

4b1: The system informs the agent that no seats are available in the client's chosen price category.

4b2: The agent specifies another price preference.

4c: Seat is not available (flight fully booked)

4c1: The system informs agent that no seats are available at all.

4c2: The agent specifies another set of departure time preferences.

4d: Client is a frequent flier and seats are not available.

4d1 The agent puts the client on a priority wait listed for the seats.

5a Client is a frequent flier

5a1: The system verifies client has upgrade coupons in their account.

5a2: The system wait lists the client for upgrade on the flight.

Solution: Use extension use cases

Use Case: Book Flight for Frequent Flier

Extends *Book Flight*

2a1 The system retrieves the client's profile and displays the flights sorted by client's airline preference.

4a1 The system offers premium seats for the client.

5a1: The system verifies client has upgrade coupons in their account.

5a2: The system wait lists the client for upgrade on the flight.

Alternatives:

4d: Client is a frequent flier and seats are not available.

4d1 The agent puts the client on a priority wait listed for the seats.

7. Interrupts As Extensions

- In a use case diagram , the **"<<extends>>"** stereotype is used to indicate that one use case can extend the behavior of another use case under certain conditions.

- It signifies a relationship between the base use case (the one being extended) and the extending use case.

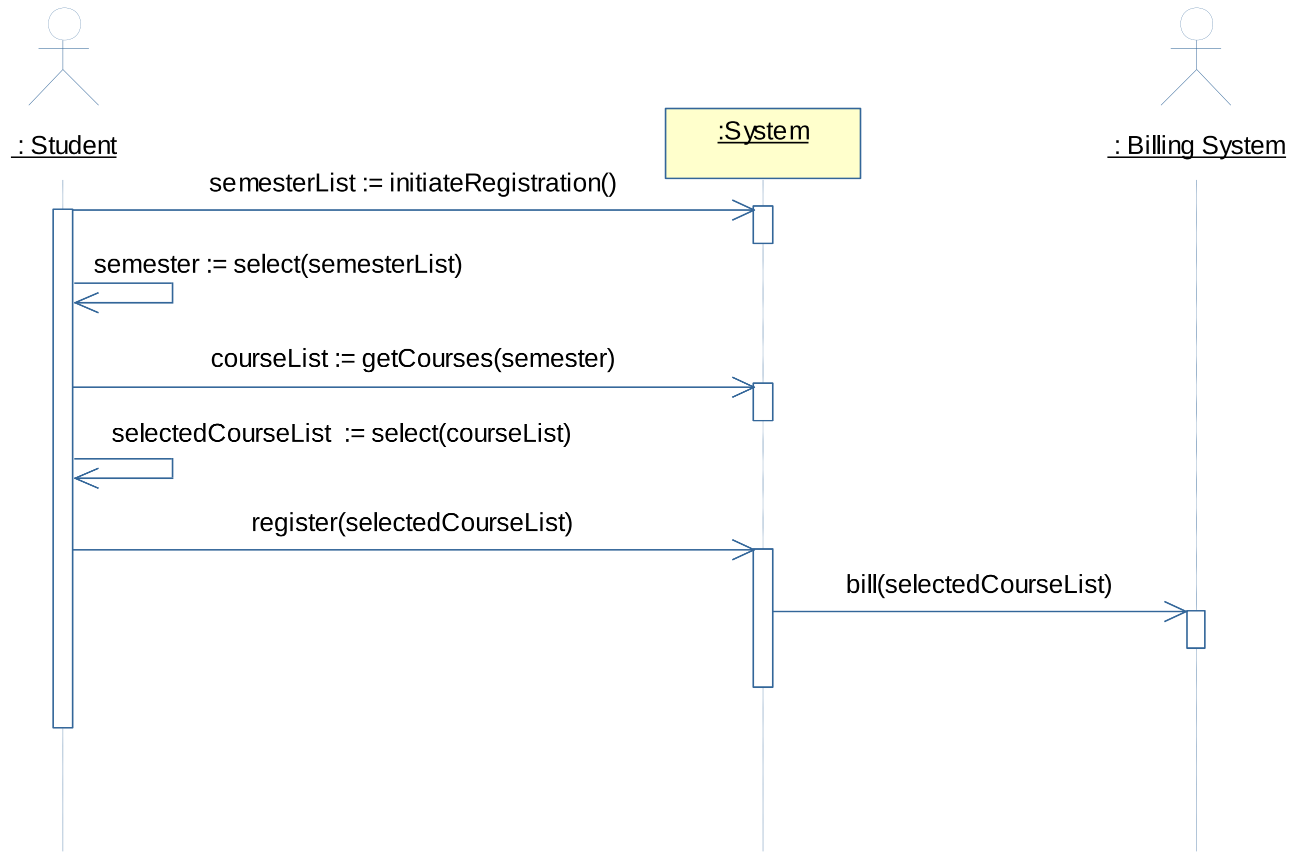
System Sequence Diagrams

- Identify behavior whose implementation is to be specified
- Details:
  - Model roles
  - Model lifelines
  - Model activations
  - Model messages

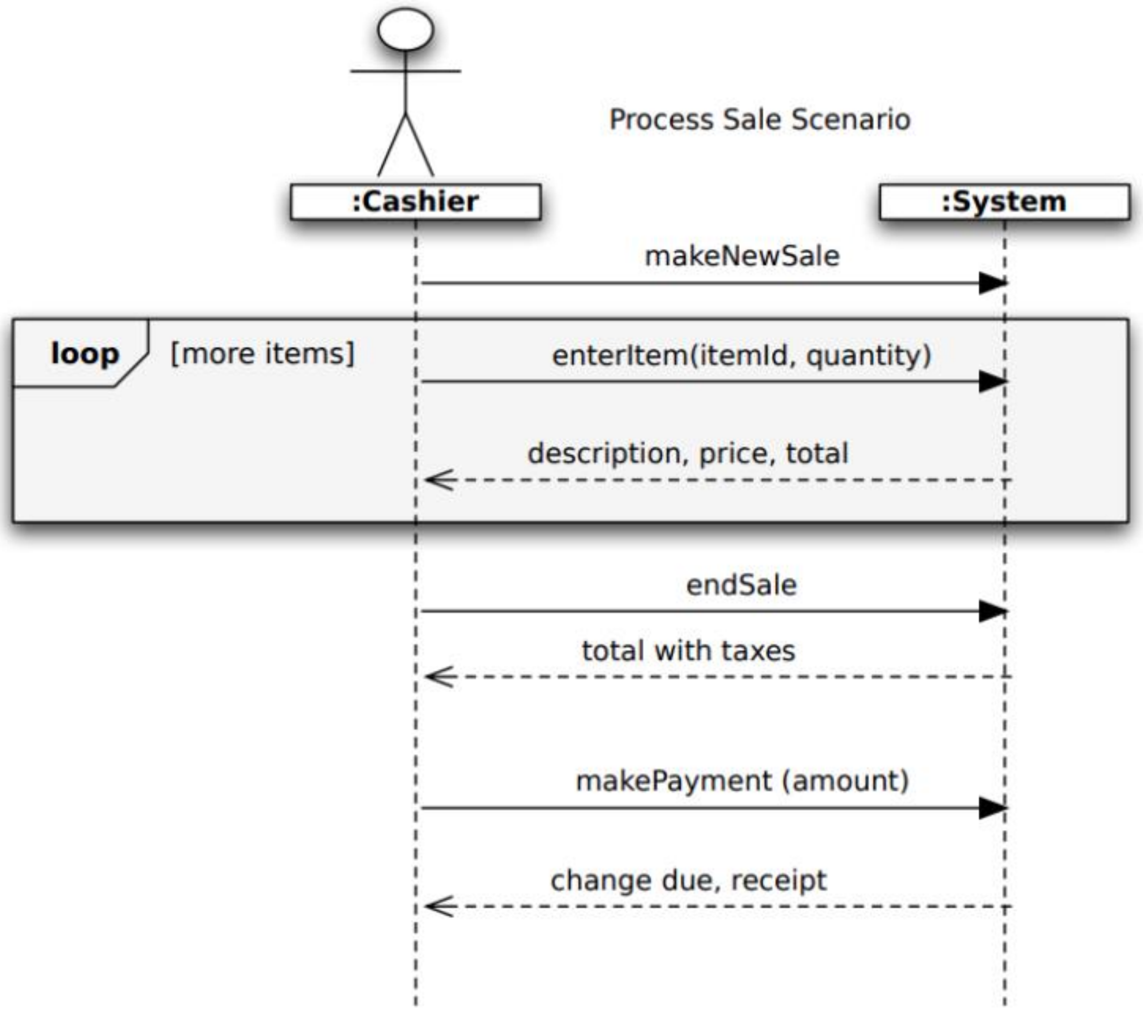
System Sequence Diagram

- Displays the lifeline of a participating object as they exchange messages in a single scenario
- Participating object:
  - Does some action
  - Puts results of action in a message
  - Sends message to another object
- The message sequence is emphasized – easy to understand interactions!!

Example: Register for Courses


: Student
:System
: Billing System
semesterList := initiateRegistration()
semester := select(semesterList)
courseList := getCourses(semester)
selectedCourseList := select(courseList)
register(selectedCourseList)
bill(selectedCourseList)

Example: Process Sale Scenario

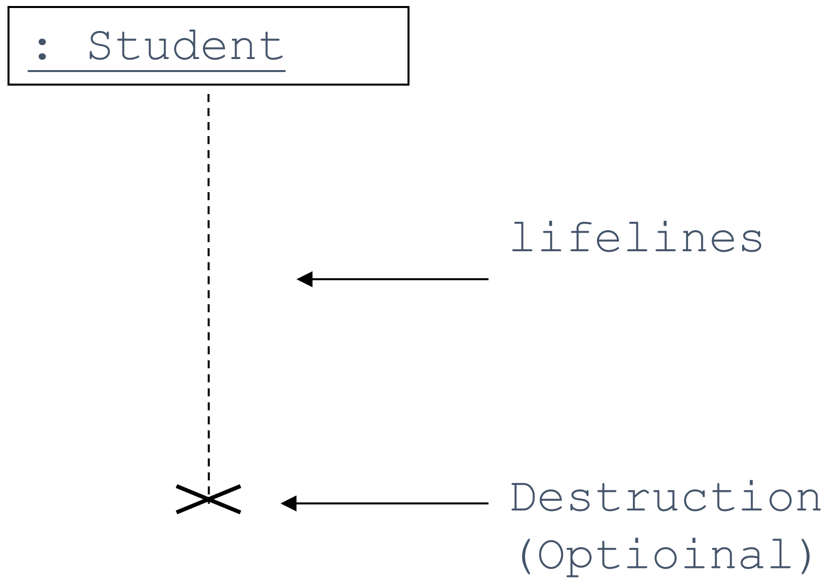


Roles

- The role name (objects) instanced is usually anonymous.
- The class name follows after the colon.

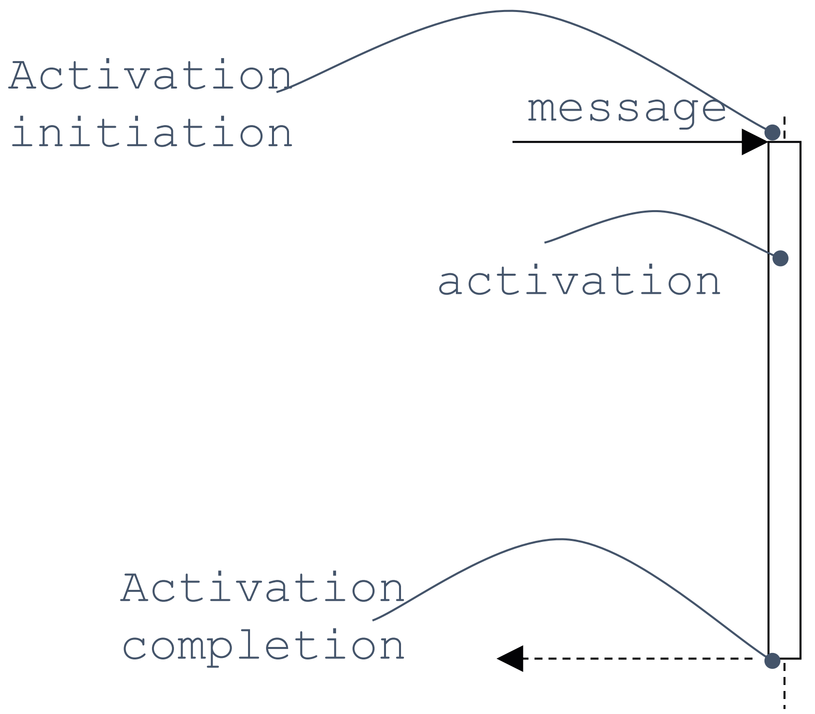
Lifelines

- Represent existence of objects over a period of time



Activations

- Represent time during which an object performs an action



Readings , Assignments, Project Tasks

Read:

- Chapters 7, 8 of the textbook.

Assignments & Project Tasks:

- Assignment 1 is open (Due on Friday 7/2). This is an Individual Submission.

Questions?