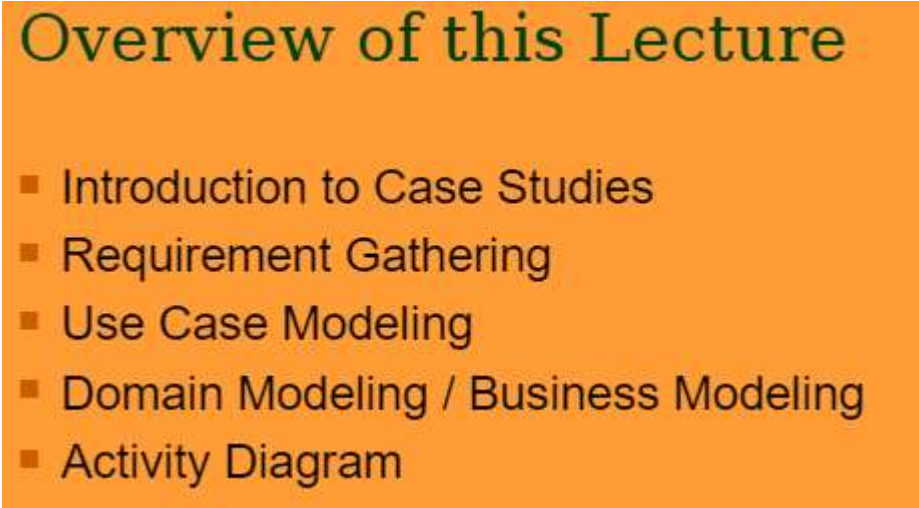
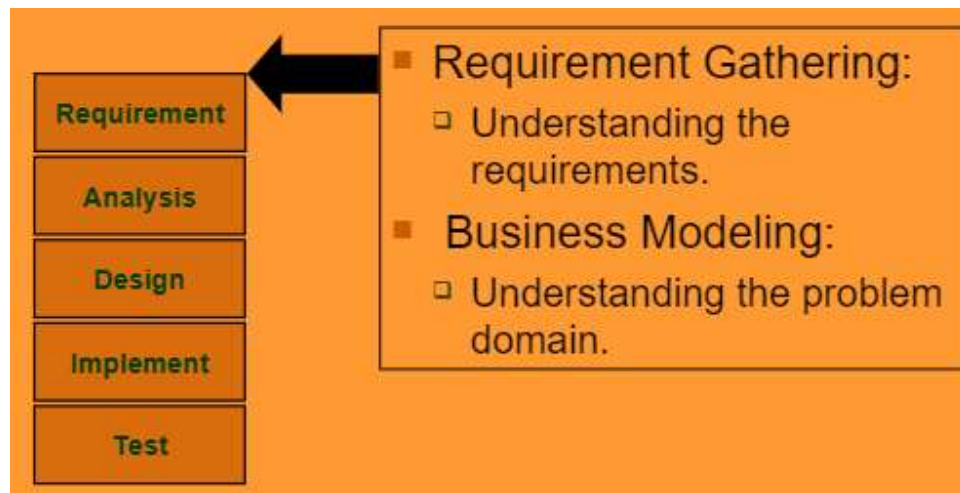


Lecture 3

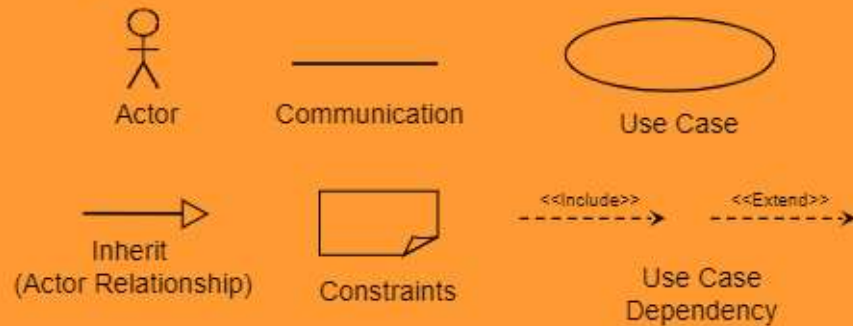
Thursday, February 2, 2023 6:08 PM

- - 
- Case Studies
 - Case Study 1: The Restaurant
 - Day to day ops include:
 - making reservations
 - allocating tables
 - Current sys use manual booking sys- hand written forms in large folder
 - 3 sittings (slots) in an evening- booking can span more than 1 slot
 - Each booking records contact name and phone number
 - Annotation made for various events:
 - Arrival of customer (cross out record)
 - Table switch (arrow to new table)
 - Cancellation
 - Time to vacate
 - "walk in" customer
 - Probs w/ manual sys:
 - Slow
 - Confusing and difficult to read
 - No backup copies
 - Hard to get useful management data
 - Dev comp version
 - Case Study 2: Monopoly Game
 - Familiar, played in many countries
 - Software version run as simulation:
 - User start simulation by indicating # of simulated players
 - All turns are simulated w/ result (trace) printed
- Requirement Gathering
 - Where we at now?
 - Requirement Gathering: understand the req
 - Business Modeling: understand the problem domain



- Requirements analysis
 - Biggest importance of software development
 - Unified process, big emphasis on capture of sys req using use cases
 - Req analysis has main goal to define set of use case scenarios/descriptions
- Req: Overview
 - Early phase of dev
 - Inputs:
 - Informal specification
 - Activities:
 - Make use case model
 - Define use cases
 - Make domain model
 - Make glossary
 - Make activity diagram
- Use Case Modeling
 - Use Case View
 - Give structured view of sys functionality
 - Description of how users interact w/ the sys
 - Supported by UML use case diagrams
 - Serves as starting pt for all subsequent devs
 - 3 important def:
 - Use case- diff tasks users perform while interacting w/ the sys
 - Scenario- certain instances of the use case:
 - Basic course of Events: normal flow
 - Alternative Course of Events: optional flow
 - Exceptional Course of Events: erroneous flow
 - Actor- roles played by users when interacting w/ a sys (like receptionists or head waiter), individual user may play 1/more roles @ diff times, roles usually go w/ certain level of access
 - Use Case Diagram

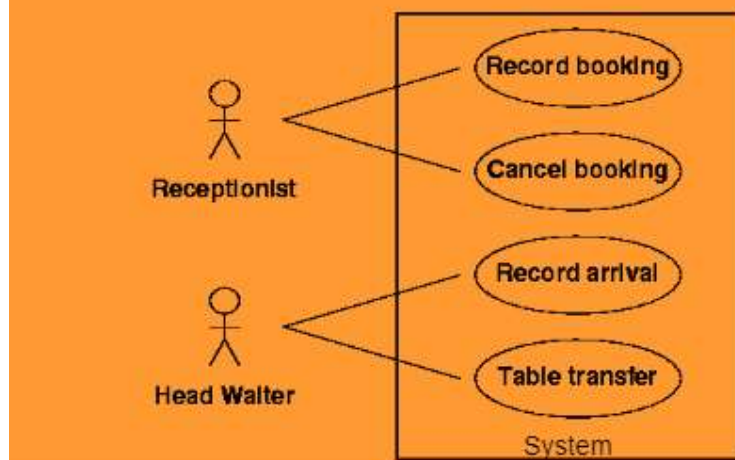
- UML diagram to *summarize* the relationship between actors and use cases.
- Diagram Element:



○ Case Study 1: Use Cases

- Preliminary list of use cases:
 - Record new booking
 - Cancel a booking
 - Record customer arrival
 - Transfer a customer from 1 table to another
- Diagram

It shows use cases, actors, and who does what.



○ Use Cases Description

- Has all possible interactions that user can have when doing a given task
- Usually dialogue btwn sys and user
- Described courses of events (aka scenarios)
- Full description of use case:
 - Basic course of events
 - Numb of alternative and exceptional courses

○ Basic course of Events- describes what happens in the 'normal' case

- For example, for 'Record Booking':

Record Booking: Basic Course of Events

- Receptionist enters date of requested reservation;
- System displays bookings for that date;
- There is a suitable table available: Receptionist enters details (customer's name, phone number, time of booking, number of covers, table number);
- System records and displays new booking.

○ Use Case Templates

- UML doesn't define standard format for use case descriptions, many template defined to structure descriptions
- Basically a list of subheadings like:
 - Name
 - Actors
 - Courses of events
- Other does emphasis on exchange btwn user and sys

<i>Actor</i>	<i>System</i>
1. Receptionist enters date of requested reservation;	2. System displays bookings for that date.
3. there is a suitable table available: Receptionist enters details (customer's name, phone number, time of booking, number of covers, table number);	4. System records and displays new booking.

- User Interface Prototype- During use case modeling activity, good to have idea of UI
- Alternative Course of Events- Describe predicted alternative flows

- For example, if no table is available:

Record Booking – No Table Available: Alternative Course of Event

- 1 Receptionist enters date;
- 2 System displays bookings;
- 3 no table available: end of use case.

- Exceptional Course of Events- when mistake made

- E.g., allocate a booking to a small table:

Record Booking – table too small: Exceptional course of events

- - 1 receptionist enters date;
 - 2 system displays bookings;
 - 3 receptionist enters details;
 - 4 system asks for confirmation of oversize booking;
 - 5 if “no”, use case terminates with no booking made;
 - 6 if “yes”, booking recorded with warning flag.

- Shared functionality- diff use cases overlap

- E.g., ‘Record Arrival’:

Record Arrival: Basic Course of Events

- - head waiter enters date;
 - system displays bookings;
 - head waiter confirms arrival for booking;
 - system records this and updates display.

- Use case inclusion- move shared functionality to sep use case, and include some in other use cases

- Move shared functionality to a separate use case:

Display Bookings: Basic Course of Events

- - staff enters a date;
 - system displays bookings for that date.

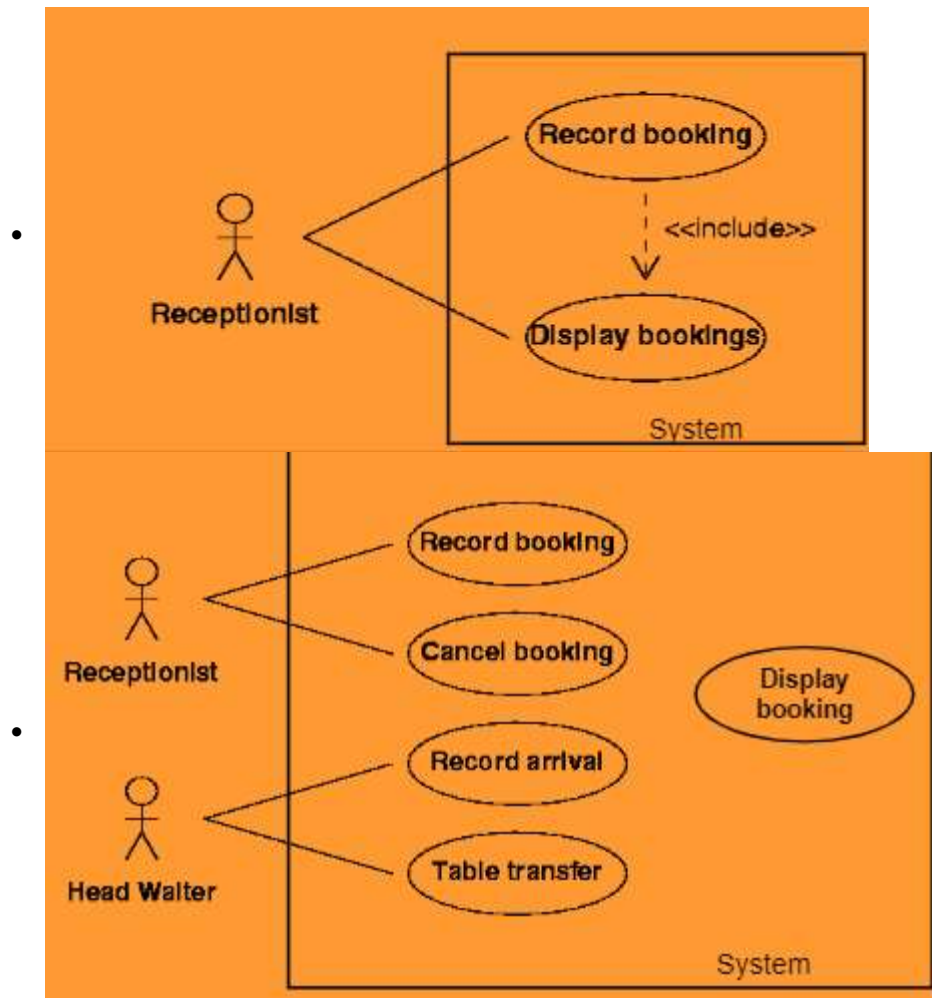
- - *Include* this in other use cases, e.g.:

Record Booking: Basic Course of Events (revised)

- - receptionist performs *Display Bookings*;
 - receptionist enters details;
 - system records and displays new booking.

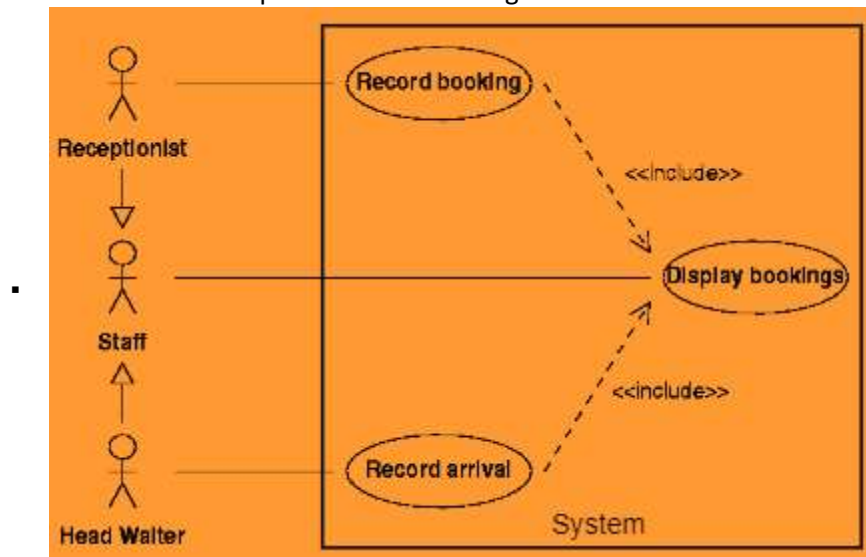
- The <<include>> Dependency

- UML shows inclusion as dependency btwn use cases, labelled w/ stereotype <<include>> on the dashed arrow



○ Actor Generalization

- Head waiter can also do display booking use case
- Intro more general actor Staff to show what other 2 actors have in common
- Initial actors are specializations of the general actor



○ Use Case Extension

- Recording a walk-in as exceptional course of events: person arrives but no booking recorded
- Can be separate use case: customer arrives & asks if free table
- Extend 'Record Arrival': w/out a booking, customer stays to eat

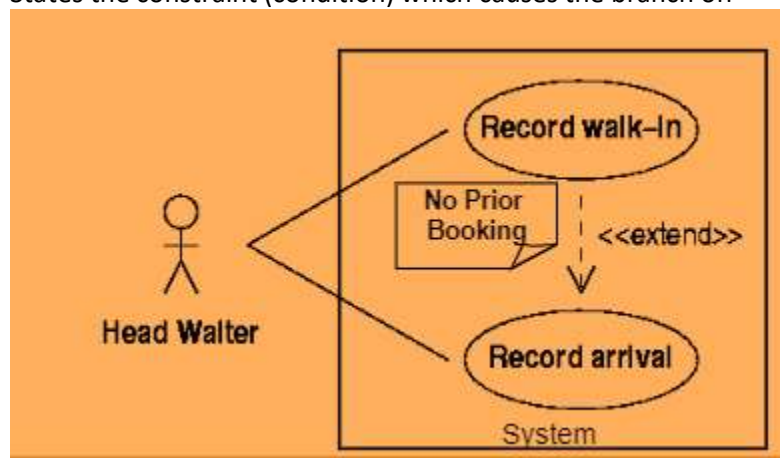
Record Walk-in: Basic Course of Events:

- 1 Head waiter performs *Display Bookings* use case;
- 2 Head waiter enters details (time, number of covers and the table allocated to the customer);
- 3 System records and displays new booking.

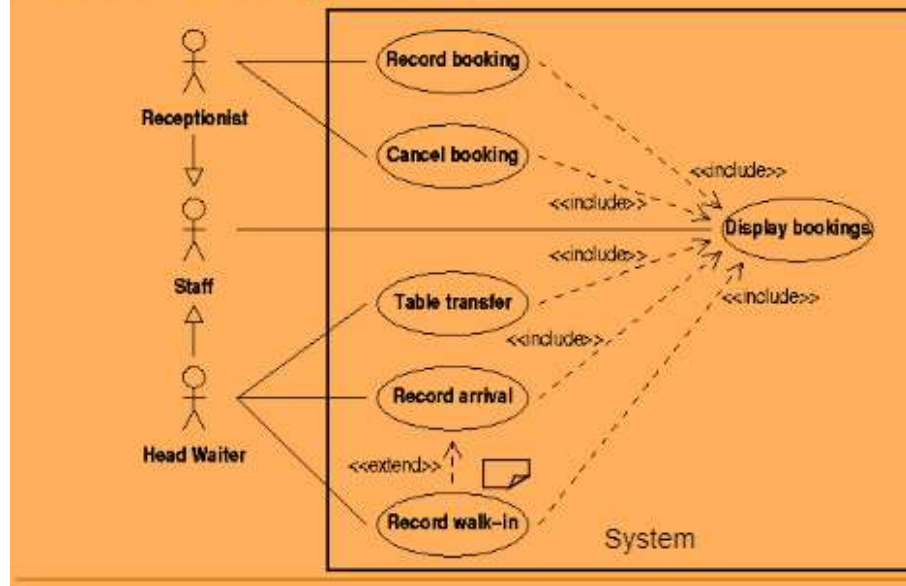
- Similar to record arrival use case, simplify?
- <<include>> is inappropriate
 - Bc interactions specified in 'Record walk-in' use case not performed every time 'Record arrival' use case is performed
 - 'Record walk-in' use case dun in some cases of 'Record Arrival' when there no booking for customer, but table free and customer still wants to eat there
 - 'Record walk-in' use case has more details than 'Record Arrival' use case
- Record Walk-in is performed in some cases of Record Arrival, when no booking for customer (but table available & customer still wants to eat there)

○ The <<extend>> Dependency

- Use case extension show w/ <<extend>> dependency
- Record walk-in optionally extends functionality of Record arrival
- States the constraint (condition) which causes the branch off

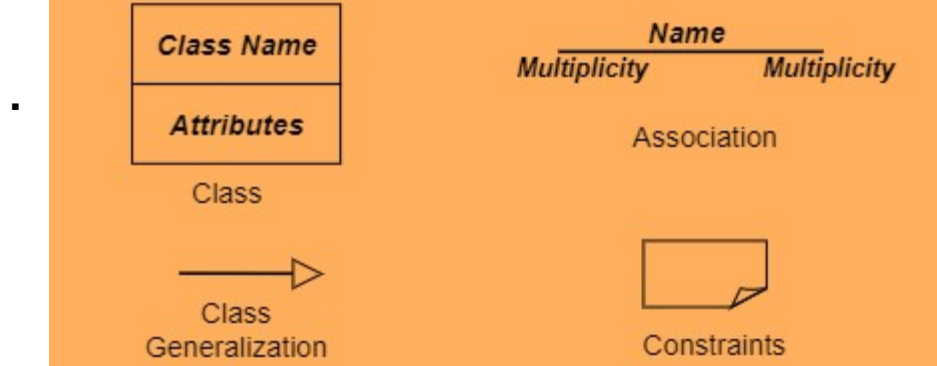


Case Study 1: Complete Use Case Diagram



- Guidelines
 - Use Case:
 - Should cover the full sequence of steps from the beginning of a task till end
 - Should describe user's interaction w/ sys, not actual computations
 - Should be as independent as possible from any UI
 - Only include actions where actor interacts w/ comp
 - Diagram should be used to supplement use case description, not main artifact
- Use Cases
 - Strengths:
 - Simple & familiar, easy to understand, customer early involve in dev
 - Emphasis user goals & perspective
 - Scale in term of sophistication & formality
 - Problems:
 - All uses must be complete & unambiguous
 - Only actor initiated activities recorded
 - Software req come from use cases usually mimic manual sys too closely
 - Stops innovating/more efficient way to dev
- Domain Modelling/ Business Modelling
 - Domain Modelling
 - Using UML diagram to make model of real-world sys-understand problem domain
 - Model recorded as simplified class diagram
 - Dev is seamless: same notation for analysis & design, design can evolve from initial domain model
 - Business Modelling – make use case model, define use cases, make domain model, & make glossary
 - Notation

A subset of *class diagram* model elements are used.



- Classes – rep real-world entities
- Attributes- rep data held abt entities
- Association- relationships btwn entities
- Generalization- used to simplify structure of model
- Constraints- indicate conditions

○ Case Study 1: Customers & Reservations

• Basic business fact: customers make reservations.

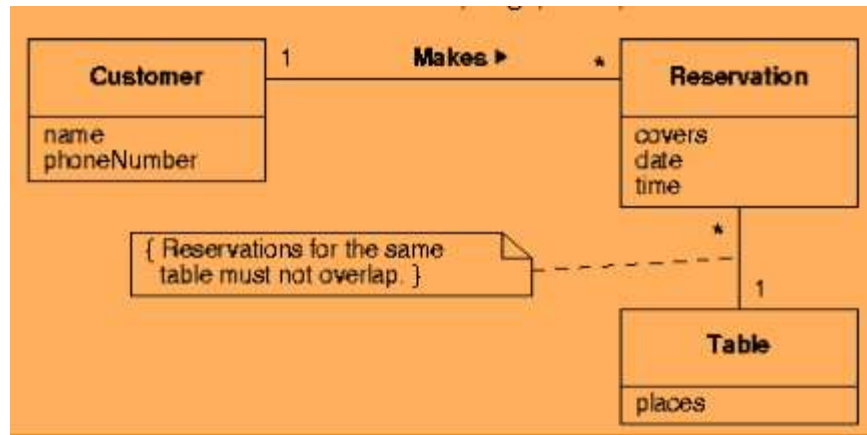


○ Define relationship

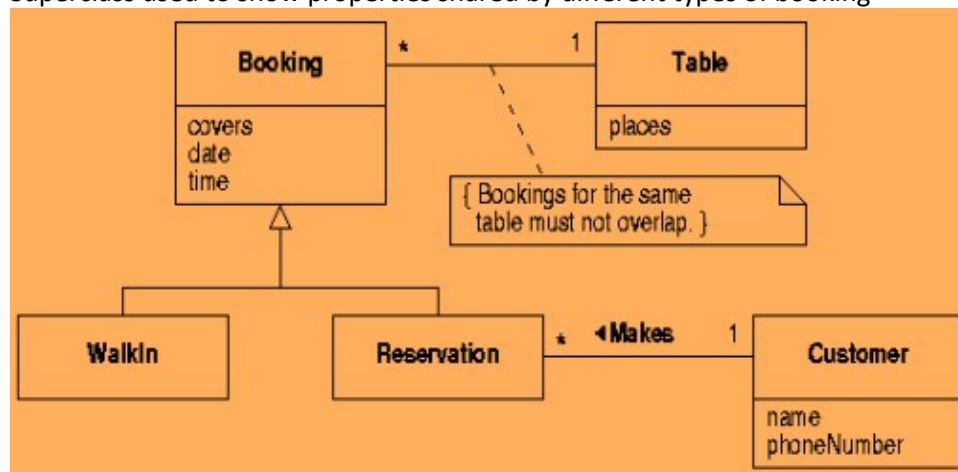
- Name to relationship, verb (so can be read as sentence)
- How many ppl make reservation?
 - One principle contact, details held
 - Principle contact make more than 1 reservation
 - Expected # of diners modelled as attribute of reservation

○ Case Study 1: Tables & Reservation

- Is table number attribute of 'reservation'?
- Better modelled as sep class: table exist even if no reservations, other attributes of tables can be stored (like size)



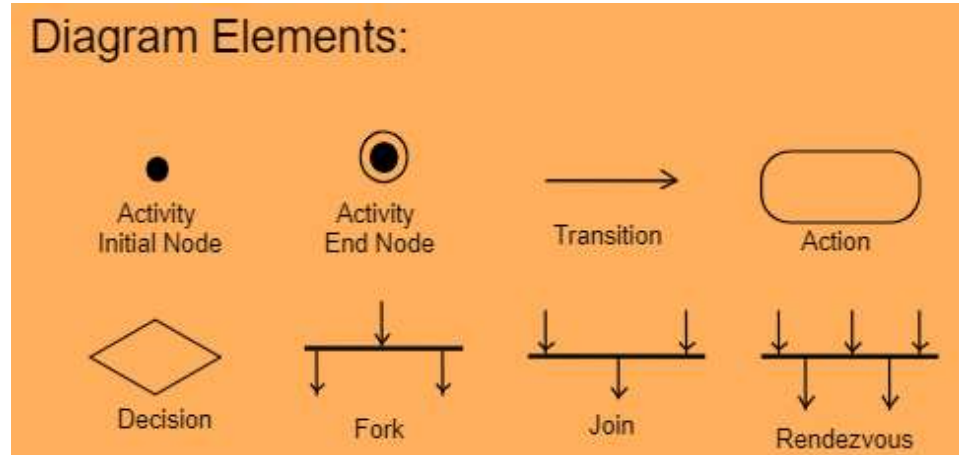
- Constraints
 - Not all domain properties can be shown
 - Constraints add info to models (written in note connected to model element btwn constrained)
- Use of Generalization
 - Superclass used to show properties shared by different types of booking



- Correctness
 - How know when model complete (we don't)
 - Not end in itself, guide to further dev tho
 - Realizing use case tests the domain model, usually lead to refinements
- Supplementary Docs
 - Common supplementary docs: glossary, activity diagram
 - Glossary: mini dictionary, captures concepts & vocab in problem domain, avoid misunderstanding & facilitates comm
 - Activity diagram is a UML diagram that describe activities
- Case Study 1: Partial Glossary
 - Booking- assignment of diners to a table
 - Covers- number of diners for a booking
 - Customer- person making reservation
 - Reservation- booking made in advance
 - Walk-in- booking not made in advance
- Activity Diagrams
 - ^
 - Similar to flow chart that describes sequence of activities
 - Useful in business modelling, use cases, design
 - Associated w/ several classes

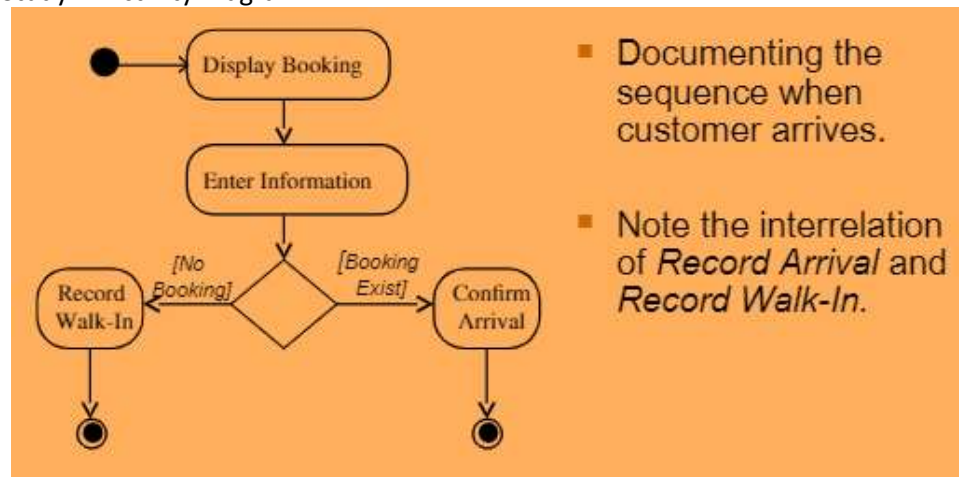
- Strengths of activity diagrams is rep of concurrent activities

○ Diagram Elements



- Action: basic block in activity diagram, rep unit of work, auto transition when complete
- Transition: rep ctrl flow: simply mvmt btwn actions
- Initial & End Node: show start and end pts in activity diagram

○ Case Study 1: Activity Diagram

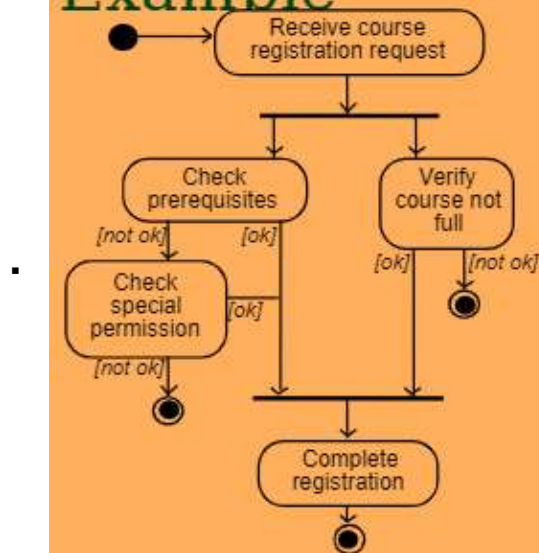


○ Concurrent Actions

- Independent actions which can be carried out @ same time (parallel)
- Shown w/ forks, joins, and rendezvous:
 - Fork: 1 incoming transition, multiple outgoing transition, execution splits into multiple concurrent threads
 - Rendezvous: multiple incoming & multiple outgoing transitions, all incoming transitions occur b4 outgoing transitions
 - Joins: multiple incoming transitions & 1 outgoing transition, outgoing transition will be taken when all incoming transitions occurred

○ Activity Diagram: Another Ex

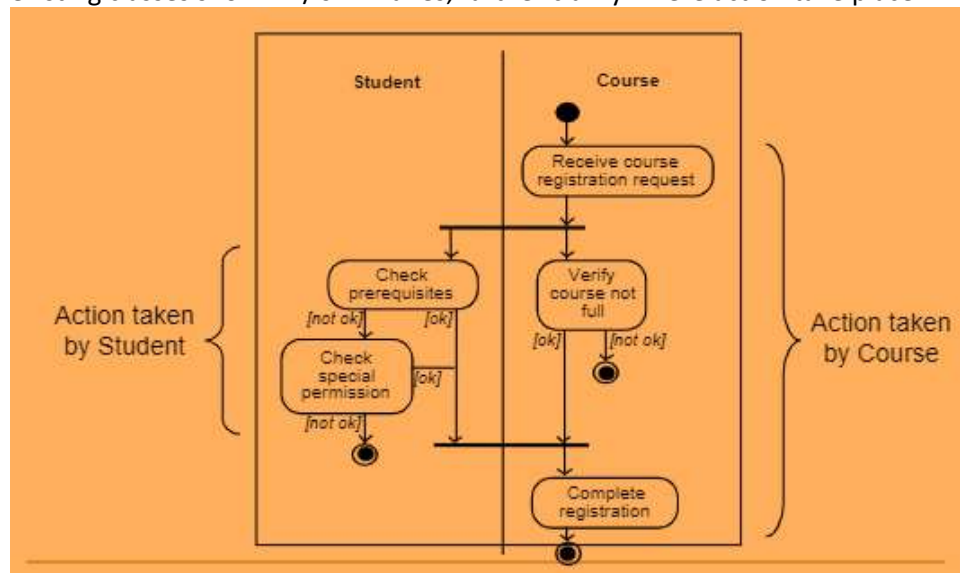
Example



- Use Cases:
 - ▣ Submit registration request.
- Register student:
 - ▣ Verify prerequisites.
 - ▣ Verify course enrolment.
 - ▣ Verify special cases.

○ Swimlanes

- Activity diagrams most associated w/ many classes- partition of activities among existing classes shown w/ swimlanes, further clarify where action take place



- Where we @ rn



■ Typical Artifacts:

- ▣ Requirement Specification.
- ▣ Use Cases:
 - Use Case Description.
 - Use Case Diagrams.
 - Activity Diagrams.
 - Glossary.
- ▣ Domain Model.

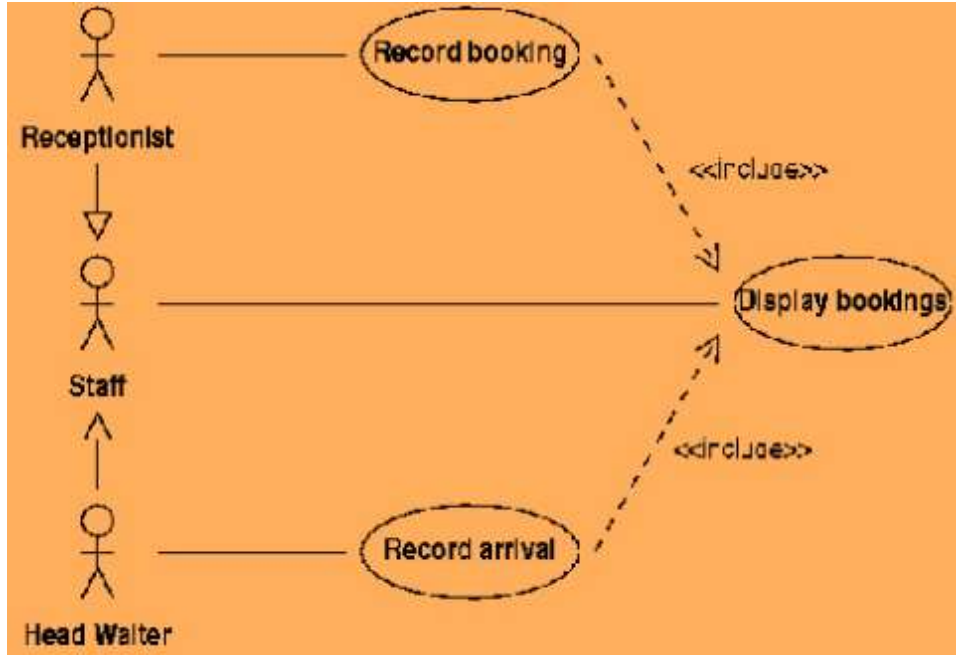
•

L3 Assessment

Thursday, February 2, 2023 6:08 PM

Q1:

Explain how you can redraw the use case diagram of 'Restaurant Booking' system project without the 'Staff' actor and



making no use of actor generalization. Do the two versions of the diagram describe the same facts, from the point of view of the restaurant's operations? Which diagram do you find clearer and easier to understand?

A:

The diagram can be drawn without the 'Staff' actor by letting them both perform the Display bookings use case independently from the Record Booking use case. From the view of the restaurant's operation's, this does not describe the same facts, as the Head Waiter and the Receptionist both seem to carry additional use cases. I find the diagram shown above easier and clearer because it is concise and thorough.

Q2:

Extend the description of the 'Record booking' use case described in Lesson 3 to cover the situation where the receptionist tries to double-book a particular table. Would this be an alternative or an exceptional course of events?

Record Booking: Basic Course of Events (revised)

1. receptionist performs Display Bookings;
2. receptionist enters details;
3. system records and displays new booking.

A:

Double-Booking - Exceptional course of Events

1. Receptionists performs Record Bookings
2. Receptionists performs Record Bookings again
3. System asks for confirmation of double-booking

4. If 'no', use case terminates with just primary booking
 5. If 'yes', both bookings recorded with warning flag
- This would be an exceptional course of events.

Q3:

How detailed a use case description should be?

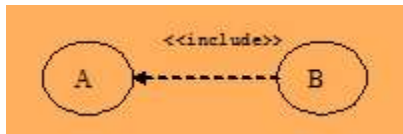
A:

There is no clear answer for that as it depends on what it is modeling. The use case could be as detailed as a list of subheadings, or as complex as an exchange between the user and the system. However it is, it should most likely include a basic course of events and a number of alternative and exceptional courses.

The use case should cover the full sequence of steps from the beginning of a task till the end, describe user's interactions with the system, be as independent as possible from any user interface, and only include actions where actor interacts with the computer. It should not show any of the actual computations.

Q4:

What is the correct meaning of the below diagram?



- A. The use case A may not necessarily be performed every time B is performed;
- B. The use case A has to be performed every time B is performed;
- C. The use case B has to be performed every time A is performed;
- D. None of the above.

Q5:

The use case view is NOT used for:

- A. describing the internal behavior and the actual computations of the system;
- B. describing the externally visible behavior of the system;
- C. describing how users interact with the system ;
- D. comprising all the possible interactions that a user can have when performing a given task;
- E. being often a dialogue between system and user.