Case Study: Restaurant Table Booking System

Comp 30160: Object Oriented Design

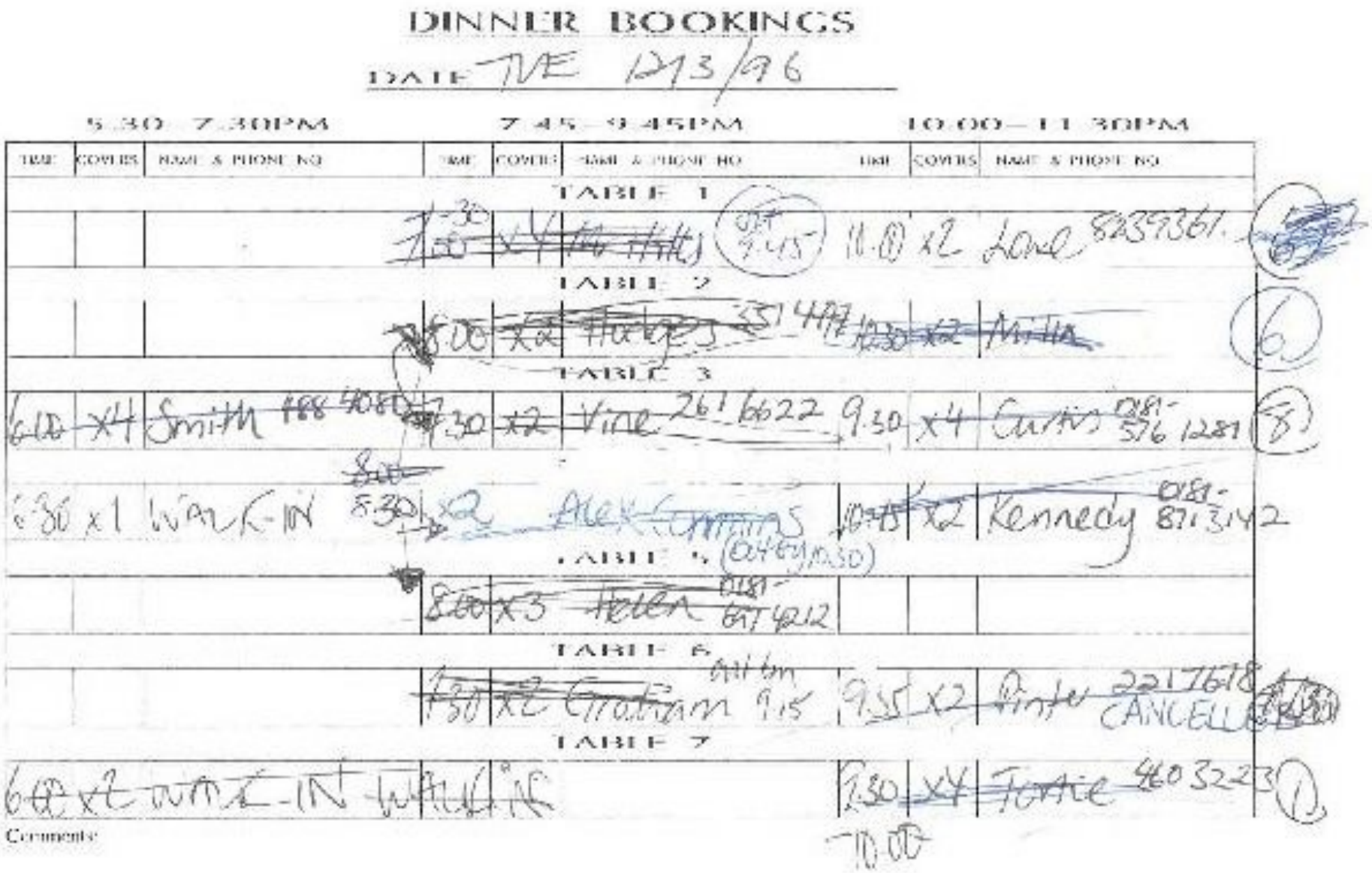
(Slides based on Chapter 4 of *Practical Object-Oriented Design with UML* by Mark Priestly)

Case Study: Restaurant Modelling

- We introduce several UML diagrams by applying one iteration of the Unified Process to a Restaurant Table Booking system.
- Our focus will be on the analysis and design models; we won't look much at implementation and deployment.
- In this section we look at building the requirements model for the system:
  - create use case model
  - create first-cut class model (aka **domain model**)
- (Most software developers would be thinking of building a web application providing services accessed by web and mobile clients. We'll ignore that side of things and do this as a pure UML exercise.)

Restaurant System

- Current system uses manual booking sheets



Current Functionality

- Advance bookings recorded on sheet
  - name and phone number of contact
  - number of diners (covers)
- 'Walk-ins' also recorded
  - number of covers only
- Arrivals noted by crossing out booking
- Cancellations, table changes etc. are recorded physically on the booking sheet

What extra functionality might an automated system provide?

Note the new terminology from the restaurant domain: **walk-in** and **cover**.

Define First Iteration

- First iteration should implement a minimal useful system, based on conversation with customer
- Proposed functionality for first iteration:
  - record bookings
  - update booking sheet information (note arrivals, cancellations etc.)
- Automated system could then already replace manual sheets

Getting Started: Use Case View

- This view is intended to provide a structured view of the system's functionality
  - **what** the system does for the user, not **how** it does it
- Based on a description of how users interact with the system
- Supported by UML use case diagrams
- Serves as the starting point for all subsequent development
  - "use case-driven development"

Use Cases

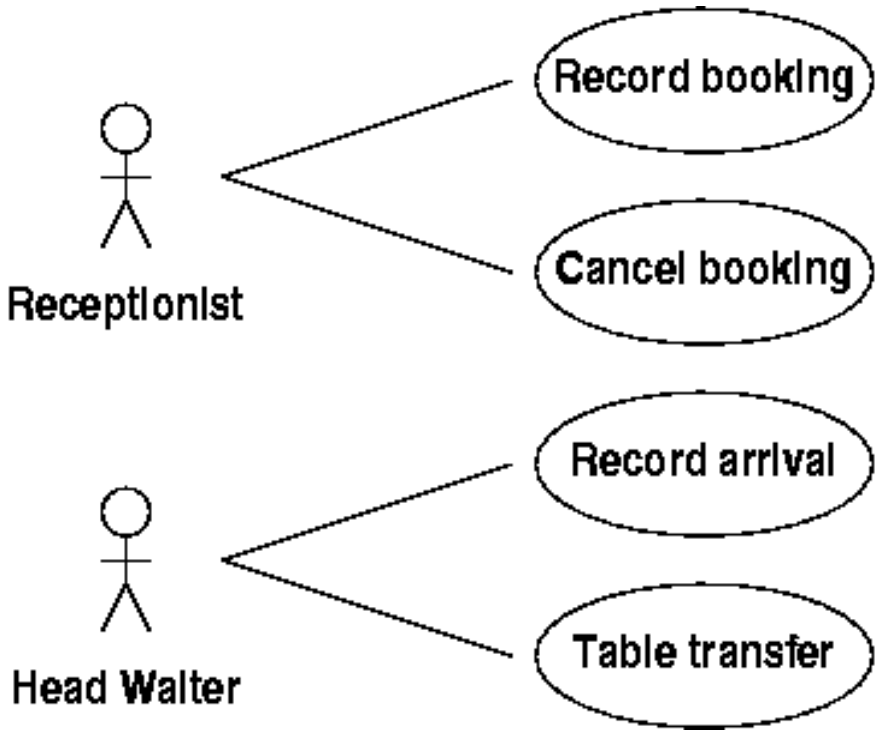
- Represent the different tasks that users can perform while interacting with the system
  - **tasks**; more than a mouseclick
- A use case isn't just "what the system does"
  - should represent some benefit for the user.
- Preliminary use case list for first iteration booking system:
  - record a new booking
  - cancel a booking
  - record the arrival of a customer
  - move a customer from one table to another

Actors

- Actors are the roles users play when interacting with a system, e.g.:
  - Receptionist (makes bookings)
  - Head waiter (assigns tables etc)
- Individual users may play one or more roles at different times
- Customers are not users of this booking system
  - => not recorded as actors
- All actors are **stakeholders**, but not all stakeholders are actors!

Use Case Diagrams

- Shows use cases, actors and who does what



Describing Use Cases

- A use case comprises the possible interactions that a user can have when performing a given task
- These are described as courses of events, or **scenarios**
- A full description of a use case includes:
  - a basic course of events
  - an number of **alternative** and **exceptional** courses

Basic Course of Events

- This describes what happens in the 'normal' case
- For example, for 'Record Booking':
  - receptionist enters date
  - system displays bookings
  - receptionist enters details
    - name, phone number, #covers.
  - system records and displays new booking
- Usually involves dialogue between system and actor

Are there reservation types not well supported by this use case?

Alternative Courses of Events

- Describe predicted alternative flows that the use case may take
- For example, if no table is available:
  - receptionist enters date
  - system displays bookings
  - no table available: end of use case

Is this a realistic use case?

Exceptional Courses of Events

- Situations where a mistake has been made or some error has occurred
- E.g. allocate a booking to a small table
  - receptionist enters date
  - system displays bookings
  - receptionist enters details
  - system asks for confirmation of oversize booking
  - if "no", use case terminates with no booking made
  - if "yes", booking recorded with warning flag
- Whether a scenario is simply alternative or exceptional is often a matter of taste.

Use Case Templates

- UML does not define a standard format for use case descriptions
  - they're intended for customer communication
- Various templates have been defined to structure descriptions, including headings such as:
  - name
  - actors
  - courses of events
  - ... + 20 other headings

Sharing Functionality between Use Cases

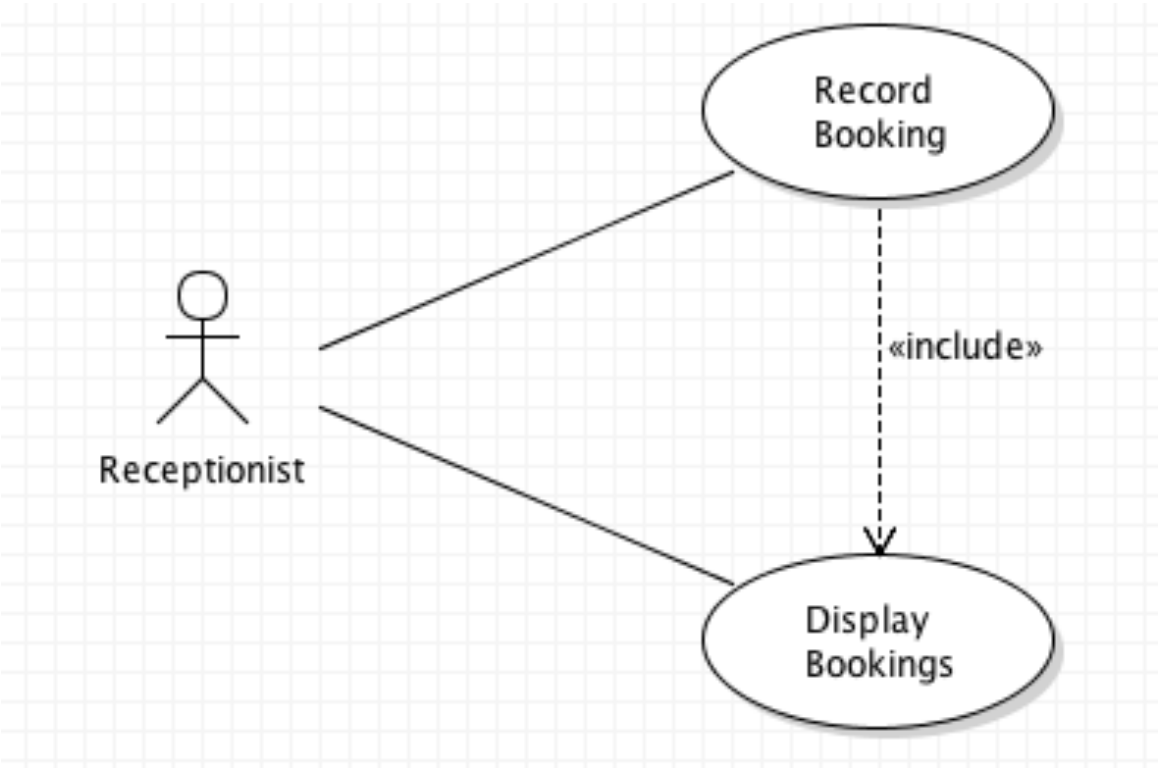
- Different use cases can overlap
- E.g. **Record Arrival**:
  - head waiter enters date
  - system displays bookings
  - head waiter confirms arrival for booking
  - system records this and updates display
- First two steps shared with ‘Record Booking’ (even though different actor)
  - We’d like to model this without duplication

Use Case Inclusion

- Move shared functionality to a separate use case, eg ‘Display Bookings’:
  - user enters a date
  - system displays bookings for that date
- Include this in other use cases:
  - receptionist performs ‘Display Bookings’
  - receptionist enters details
  - system records and displays new booking

The ‘include’ Dependency

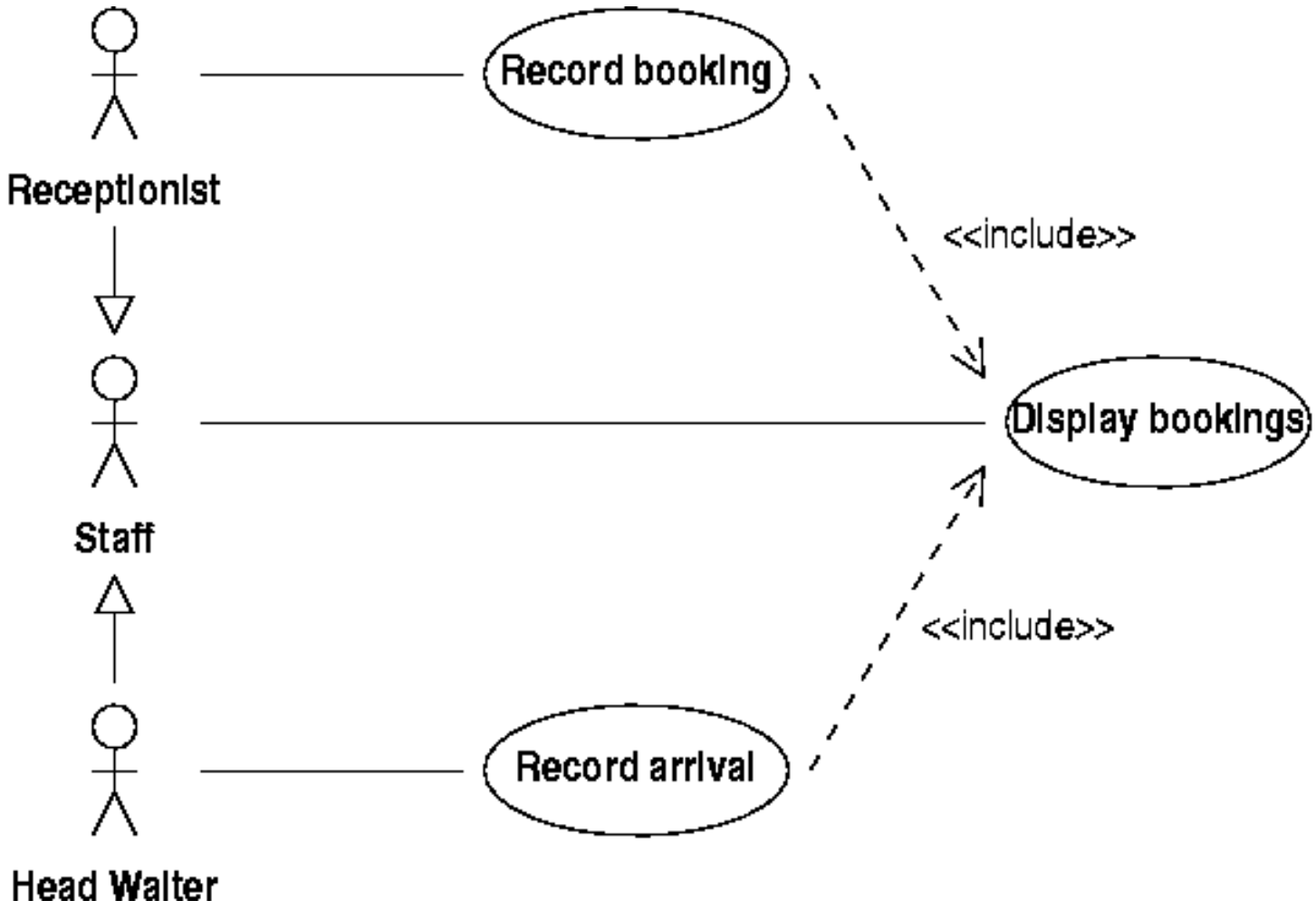
- UML shows inclusion as a dependency between use cases, labelled with the **stereotype** include:



Actor Generalisation

- This diagram shows that the receptionist can display bookings without performing the including use case 'Record Booking'
  - a useful function in itself
- However, head waiters can also display bookings...
  - Again, we'd like to model this without duplication
- Solution: Introduce a more general actor called Staff to model what Head Waiters and Receptionists have in common
- The initial actors are specialisations of the general actor. See next slide.

Actor Generalisation Notation

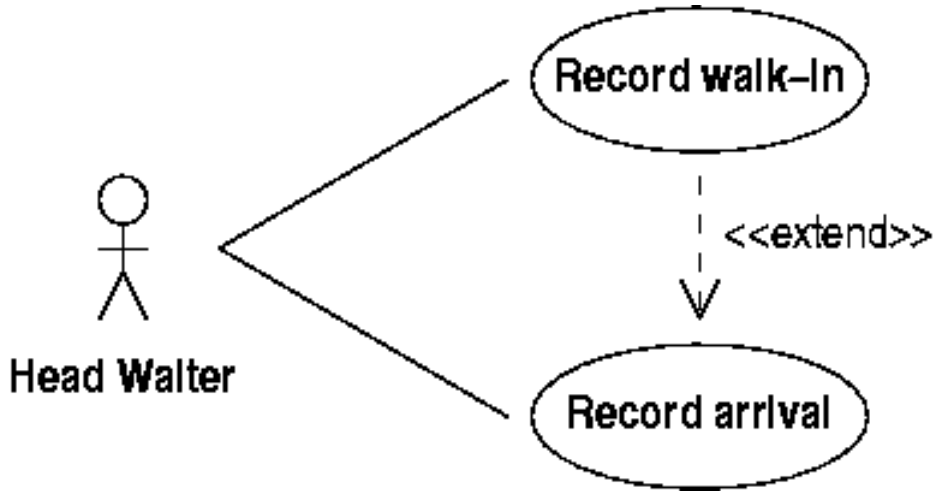


Use Case Extension

- Recording a walk-in can be modelled a separate use case
  - a customer arrives and asks if there's a free table
- It could also be described as an exceptional source of events in the 'Record Arrival' use case.
  - Someone arrives, but there's no booking recorded.
- Then it can be modelled as an **extension** of 'Record Arrival'
  - even without a booking, the customer stays to eat
  - See next slide.

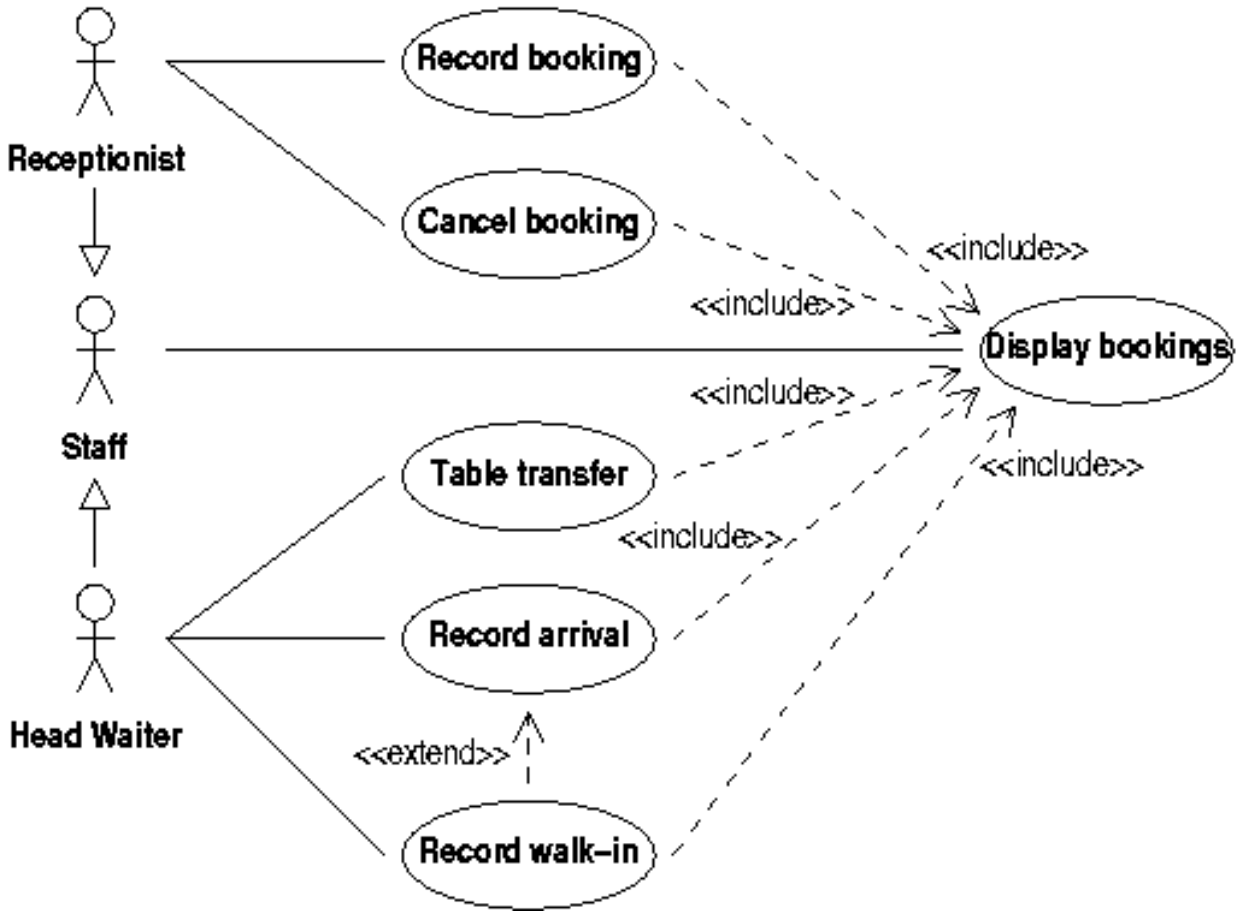
The ‘extend’ Dependency

- Use case extension is shown with a dependency thus:



- Extensions are special cases of the base use case they extend. You might decide to implement the base case in one release, but leave the exceptional case to a later release.

Complete Use Case Diagram



Use Case Summary

- Use case describe the functionality of the system, i.e. **what** it does.
- It's a very simple model, one that customers can easily read. It is built on the basis of discussions with the customer.
- It can be made more sophisticated by using *include* and *extends* relationships
  - However, this makes the model harder for customers to read
  - Overuse of *include* and *extends* relationships may mean that the software system is being designed -- avoid this!
- Use Cases are core in UML and are often used to drive the subsequent software development, termed *use case driven development*.