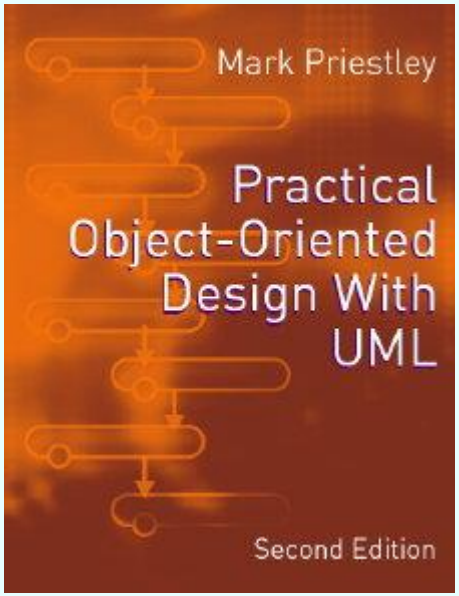
PRACTICAL OBJECT-ORIENTED DESIGN WITH UML 2e



Chapter 5:
Restaurant System: Analysis

Analysis

- What is to be analyzed?
  - the system requirements
- Why?
  - to demonstrate their implementability
- How?
  - by drawing interaction diagrams *realizing* use cases

Analysis

- From Requirements get
  - Use-case descriptions which document external interactions with the putative software, known as external messages between actor and system
  - Domain Model which defines the relationships between important business concepts
- Analysis is about finding out how objects derived from the domain model can be made to cooperate in such a way as to implement behaviour described in use-cases.
- Hence the term ‘use-case realization’

Analysis & UP

- Analysis in the UP mostly occurs in the Elaboration Phase
- Use-case realization leads to a more comprehensive class diagram, usually a reified version of the the domain model
- Besides use-case realizations, another important product of the analysis workflow is the Software Architecture Description (SAD)

Analysis v. Design

- Difficult to draw a boundary
- Traditional informal distinction:
  - analysis models the real-world system
  - design models the software
- Object-oriented methods use the same notation for both activities
  - encourages ‘seamless development’ and iteration

Object Design

- We need to define attributes and operations for each class in the model
- Start from domain model, but:
  - structure of real-world application is not always the optimal structure for a software system
  - domain model does not show operations
- *Realization* identifies operations and confirms that design supports functionality

Object Responsibilities

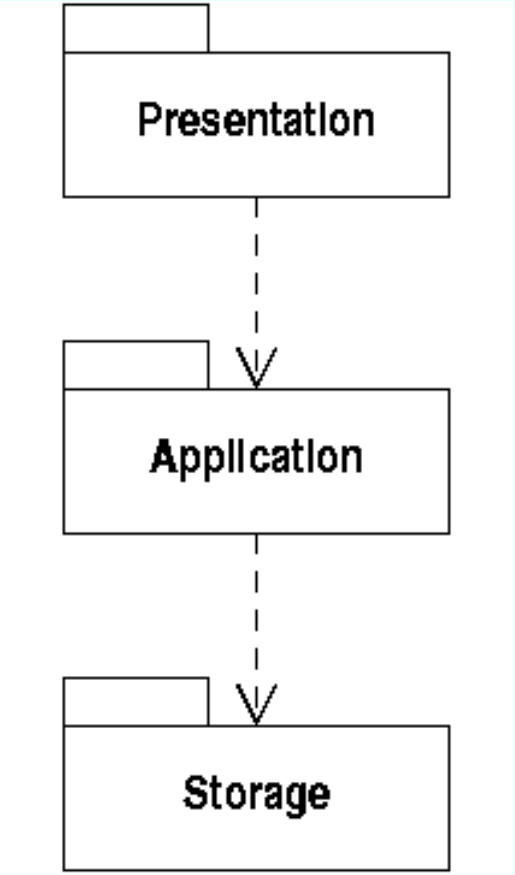
- Each class in a system should have well-defined *responsibilities*
  - to manage a subset of the data in the system
  - to manage some of the processing
- The responsibilities of a class should be *cohesive*
  - they should 'belong together'
  - they should form a sensible whole

Software Architecture

- The UP analysis workflow includes the production of an *architectural description, the SAD*
- This defines:
  - the top-level structure of subsystems
  - the role and interaction of these subsystems
- Typical architectures are codified in *patterns*
  - for example, *layered architectures*

A Layered Architecture

- Subsystems are shown as UML *packages* linked by *dependencies*
- A dependency without a stereotype means *uses*



Separation of Concerns

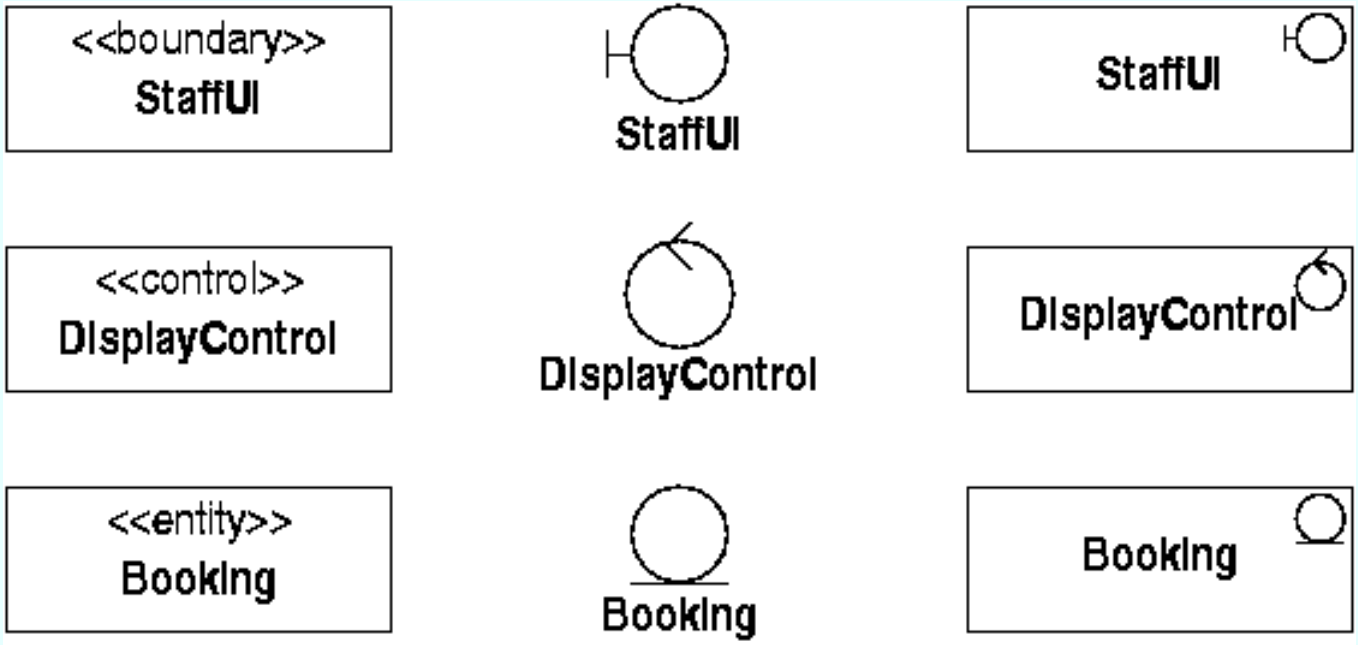
- Layers aim to insulate a system from the effects of change
- For example, user interfaces often change
  - but the application layer does not use the presentation layer
  - so changes to system should be restricted to presentation layer classes
- Similarly, details of persistent data storage are separated from application logic

Analysis Class Stereotypes

- Within this architecture objects can have various typical roles
  - *boundary* objects interact with outside actors
  - *control* objects manage use case behaviour
  - e*ntity* objects maintain data
- These are represented explicitly in UML by using *analysis class stereotypes*

Class Stereotype Notation

- Stereotypes can be text or a graphic icon
- The icon can replace the normal class box



Use Case Realization

- Begin with functionality in application layer
- ‘Display Bookings’: simple dialogue
  - the user provides the required date
  - the system response is to update the display
- Initial realization consists of
  - instance of the ‘Staff’ actor
  - an object representing the system
  - message(s) passed between them

System Messages and Boundary Class

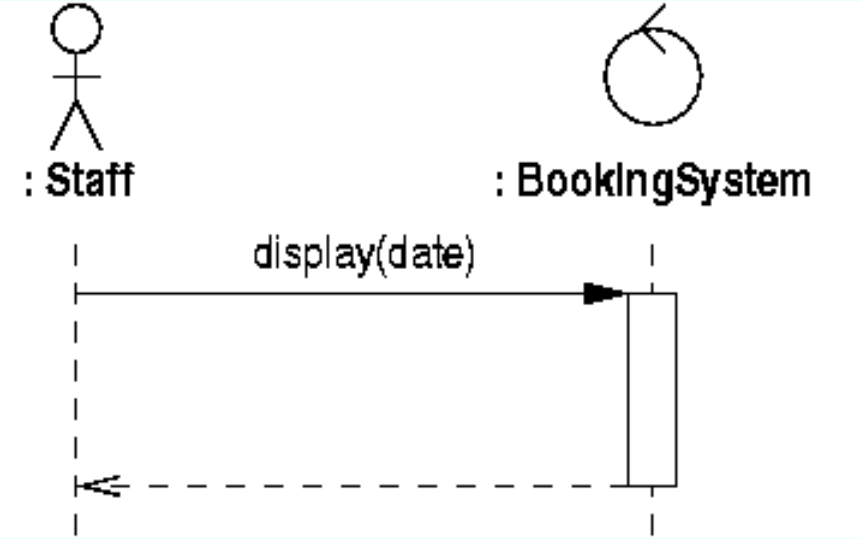
- System Messages – from outside the software to the software, i.e. from actor to software system. Could be clicking on an Ok button or entering a number. Documented in use-case descriptions.
- Internal Messages – from object to object
- The Unified Process (UP) advocates the use of a boundary class between the actor and application classes. It receives system messages.

System Messages and Boundary Class

- However, Analysis Modelling is chiefly concerned with use-case realization within the application layer
- A boundary classes refers to the presentation layer and so we can ignore it.
- In general there are several system messages in a use-case and it is important that they are handled in the correct order and that appropriate objects respond to them.
- This is the role of a control object

System Messages

- *System messages* are sent by an actor
- Represent system by a *controller*
  - initially analysing use case behaviour, not I/O



Sequence Diagrams

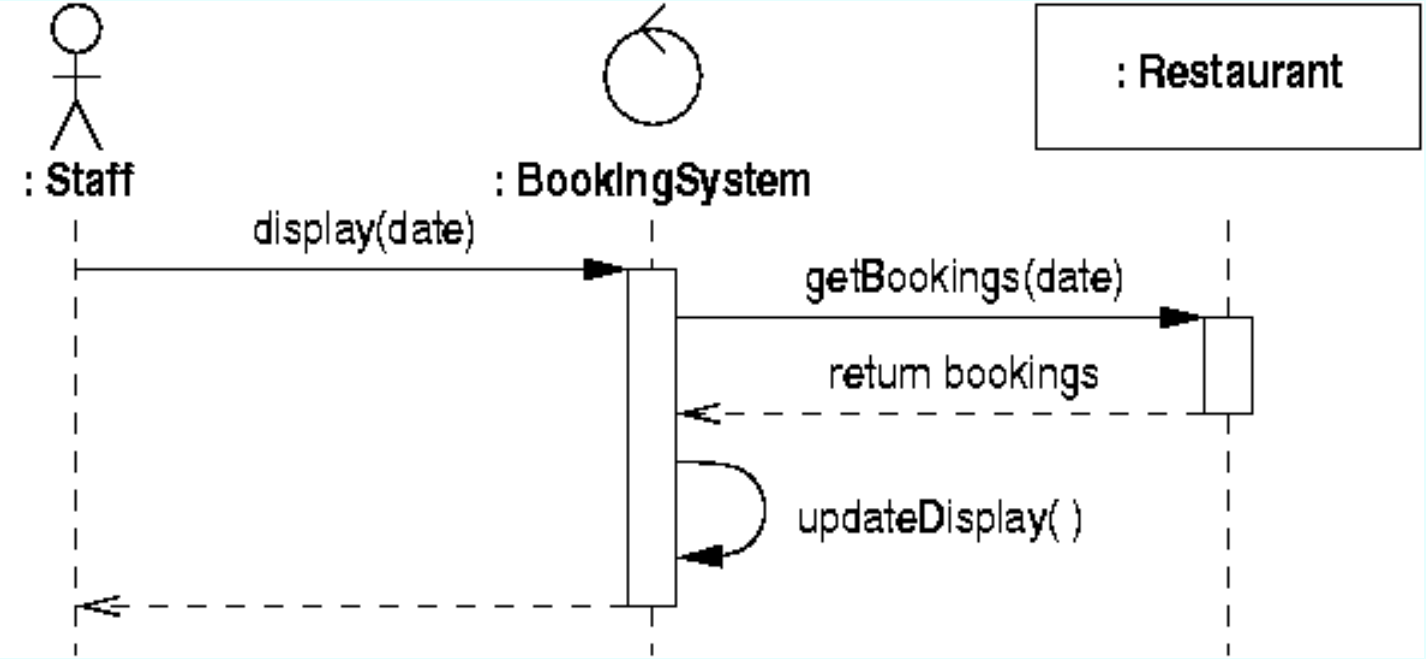
- Time passes from top to bottom
- Instances of classes and actors at top
  - only show those participating in this interaction
  - each instance has a *lifeline*
- Messages shown as arrows between lifelines
  - labelled with operation name and parameters
  - return messages (dashed) show return of control
  - *activations* show when receiver has control

Accessing Bookings

- How does the system retrieve the bookings to display?
- Which object should have the responsibility to keep track of all bookings?
  - if this was an additional responsibility of the 'BookingSystem' control object it would lose *cohesion*
  - better to assign responsibility of keeping track of the booking entities to another object
  - so define a new 'Restaurant' object with the responsibility to manage booking data

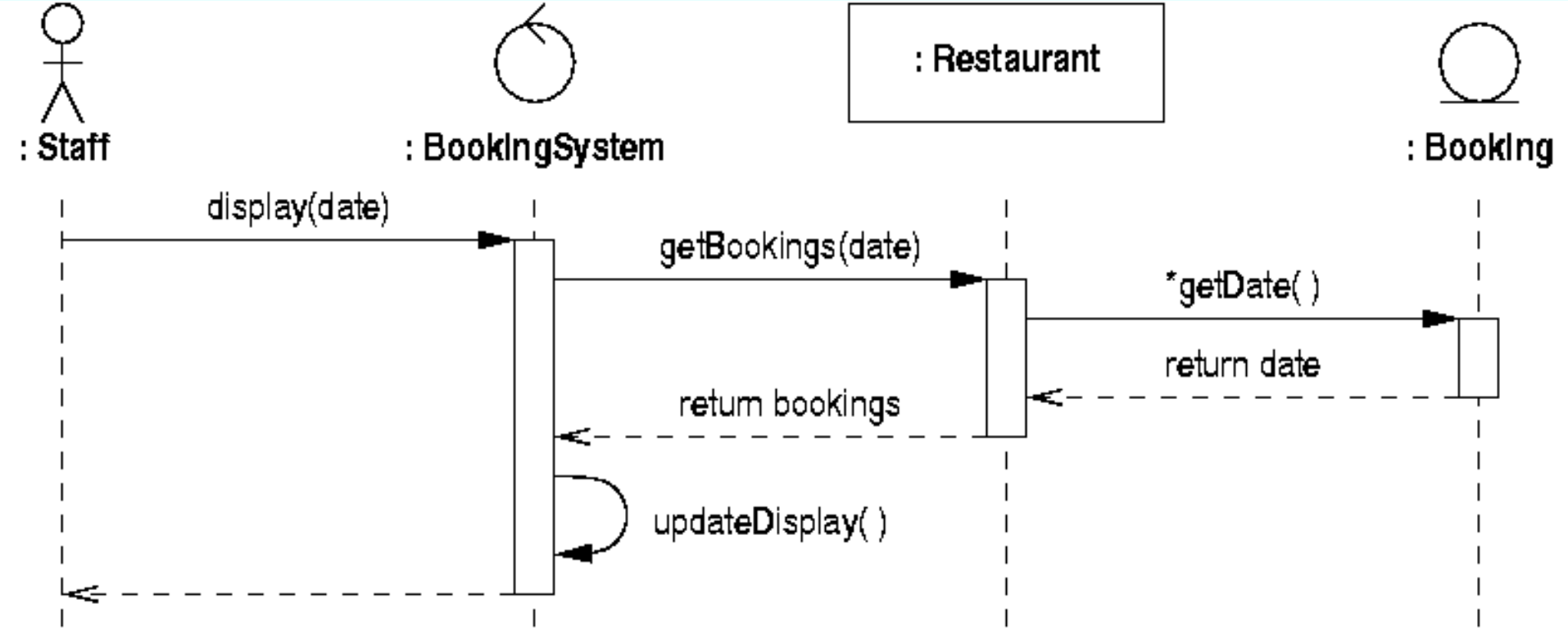
Retrieving Bookings

- Add a message to get relevant bookings
- ‘updateDisplay’ is an *internal* message. In actuality it will be sent to presentation layer object.



Retrieving Booking Details

- Dates of individual bookings will need to be checked by the ‘Restaurant’ object

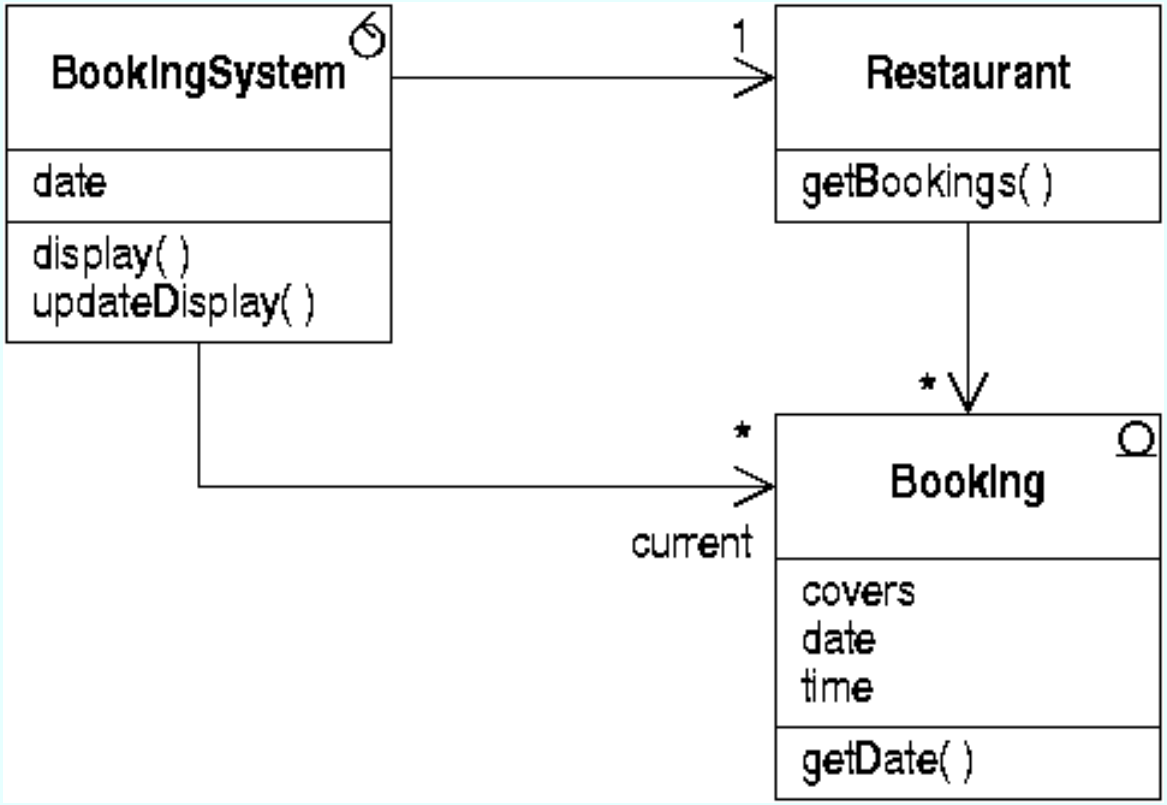


Refining the Domain Model

- This realization has involved:
  - new 'Restaurant' and 'BookingSystem' classes, with an association between them
  - an association from ‘Restaurant’ to ‘Booking’
    - ‘Restaurant’ maintains links to all bookings
    - messages sent from restaurant to bookings
  - an association from ‘BookingSystem’ to ‘Booking’
    - ‘BookingSystem’ maintains links to currently displayed bookings

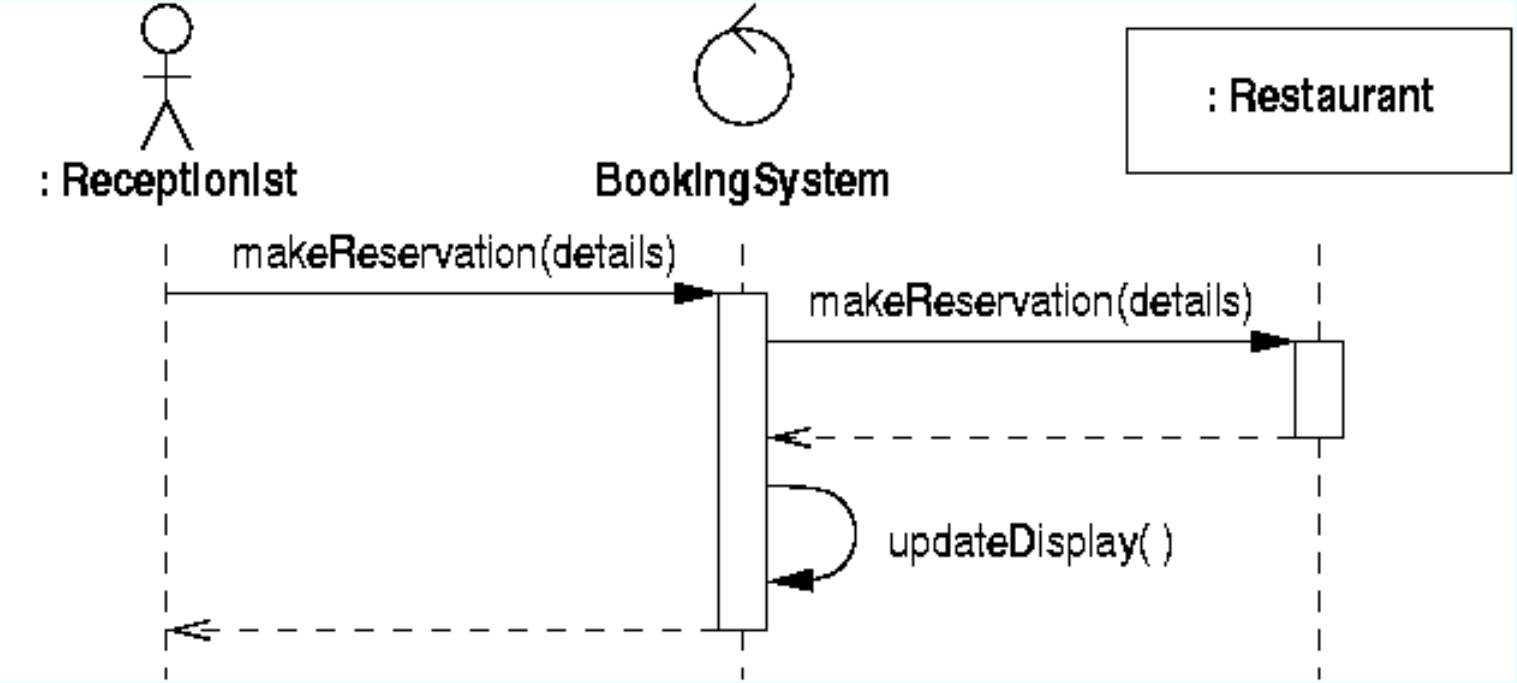
Updated Class Diagram

- Operations are derived from messages sent *to* the instances of a class



Recording New Bookings

- Give ‘Restaurant’ responsibility for creation
  - don’t model details of user input or data yet

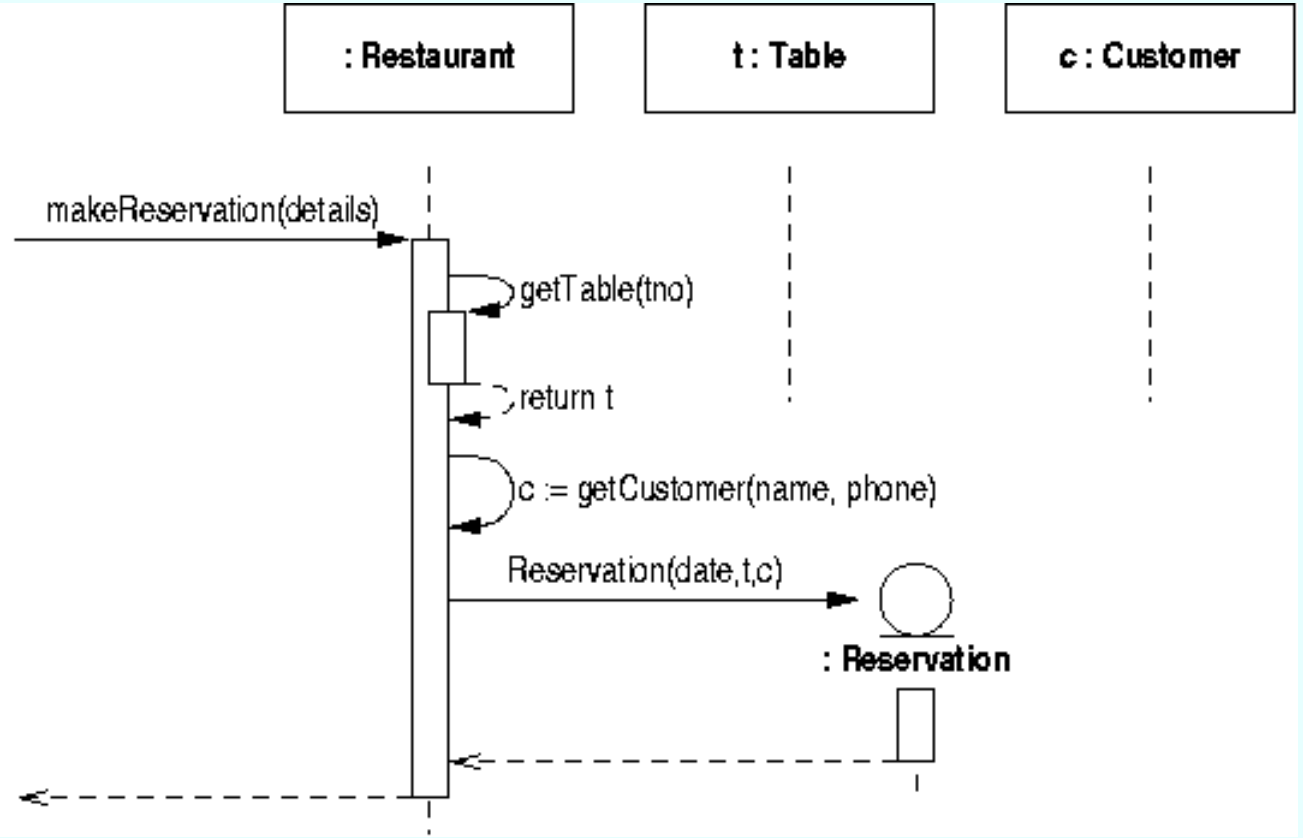


Creating a New Booking

- Bookings must be linked to table and customer objects
  - responsibility of ‘Restaurant’ to retrieve these, given identifying data in booking details
- New objects shown at point of creation
  - lifeline starts from that point
  - objects created by a message arriving at the instance (a *constructor*)

Creating a New Booking

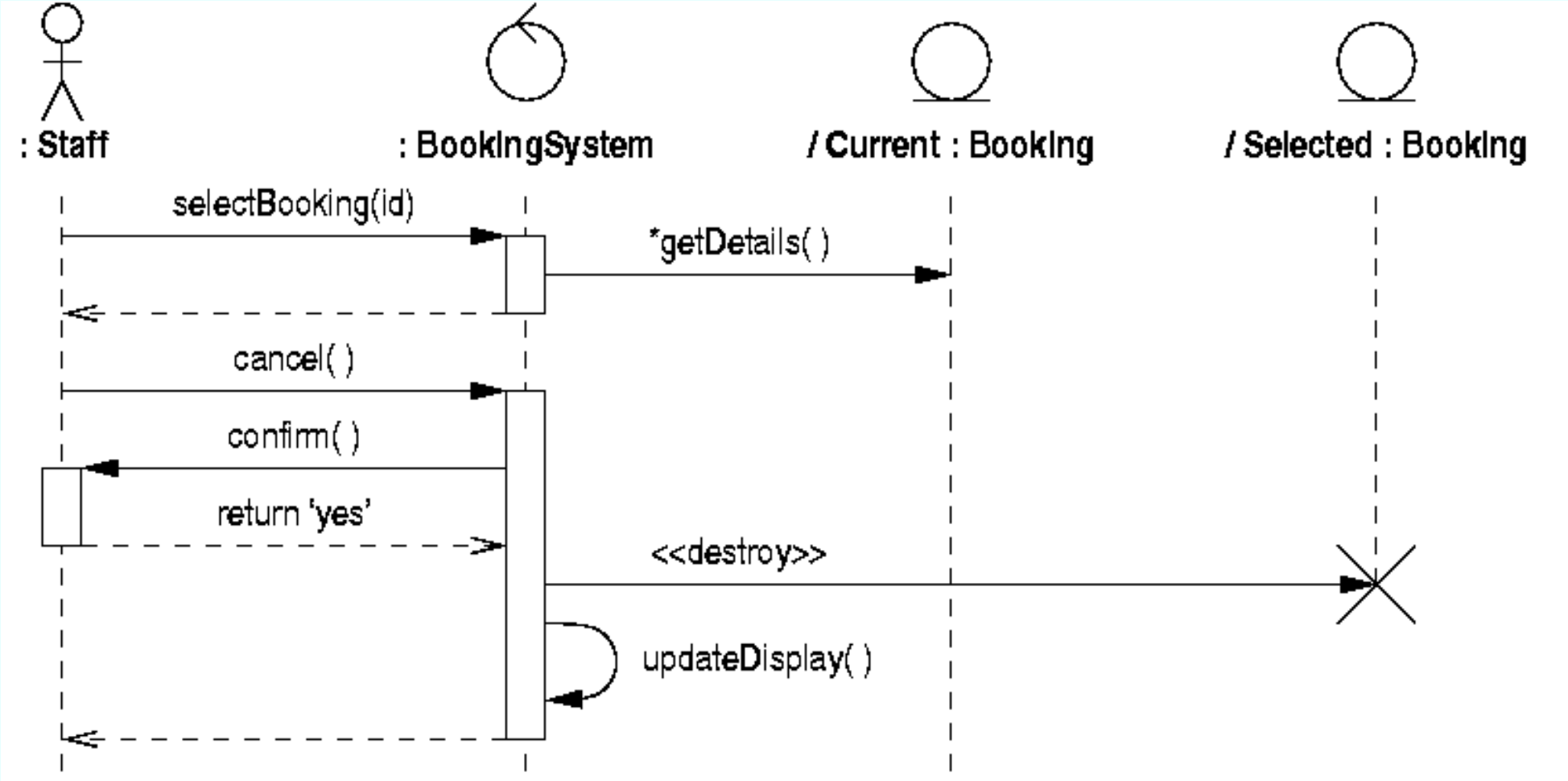
- This completes the previous diagram



Cancelling a Booking

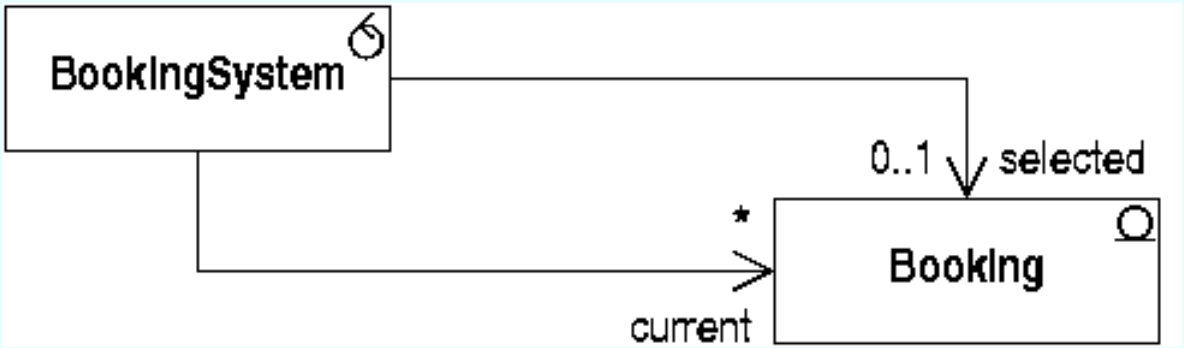
- A three-stage process:
  - select on screen the booking to be cancelled
  - confirm cancellation with user
  - delete the corresponding booking object
- Object deletion represented by a message with a ‘destroy’ stereotype
  - lifeline terminates with an ‘X’
- *Role names* used to distinguish selected object from others displayed

Cancelling a Booking



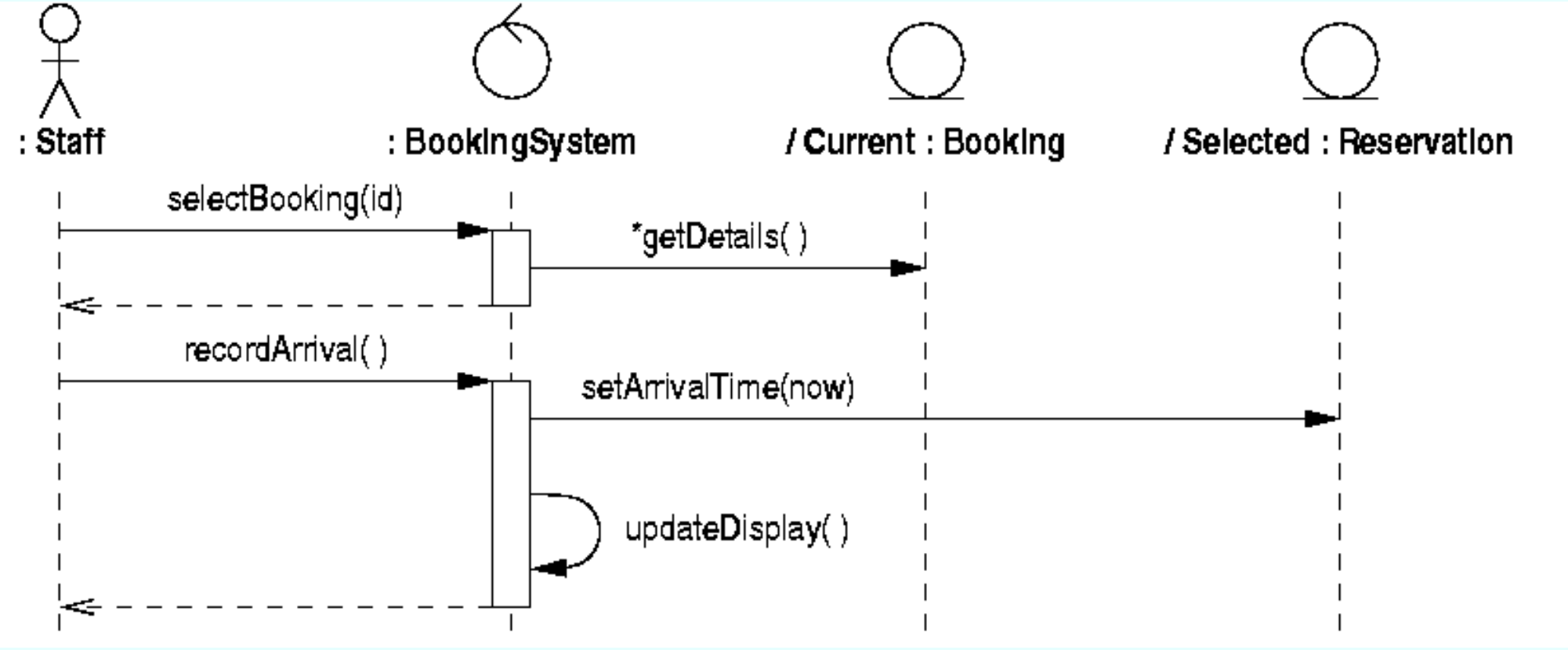
Refining the Domain Model (2)

- 'BookingSystem' has the responsibility to remember which booking is selected
- Add an association to record this



Recording Arrival

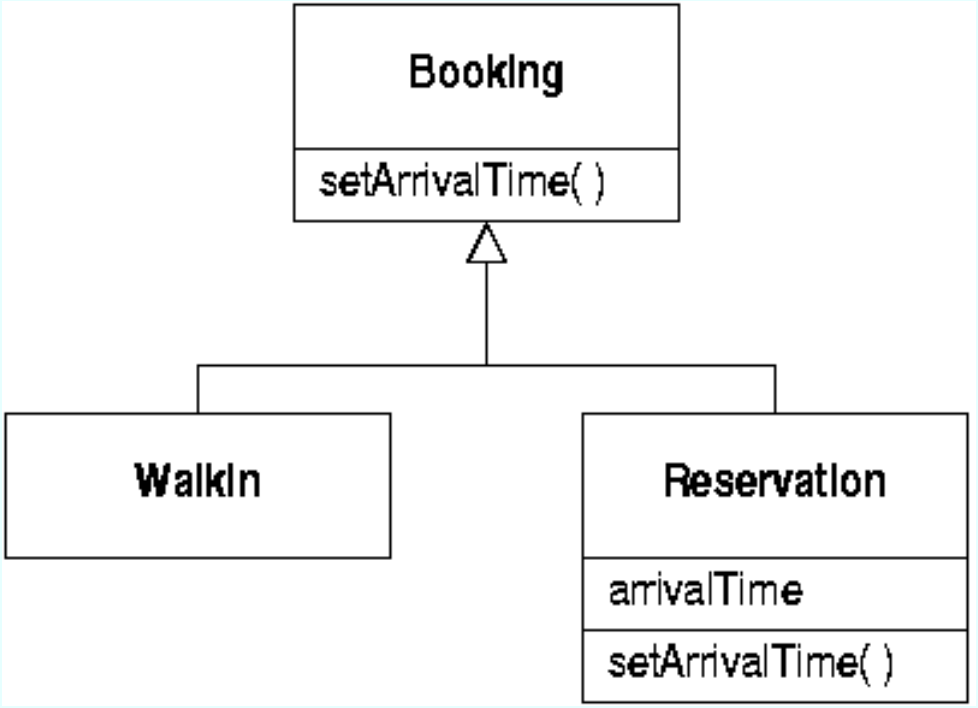
- Selected booking must be a reservation



Class Interface Design

- Should ‘setArrivalTime’ be defined in Booking or Reservation class?
  - on the one hand, it doesn't apply to walk-ins
  - but we want to preserve a common interface to all bookings if possible
- Define operation in ‘Booking’ class
  - default implementation does nothing
  - override in ‘Reservation’ class

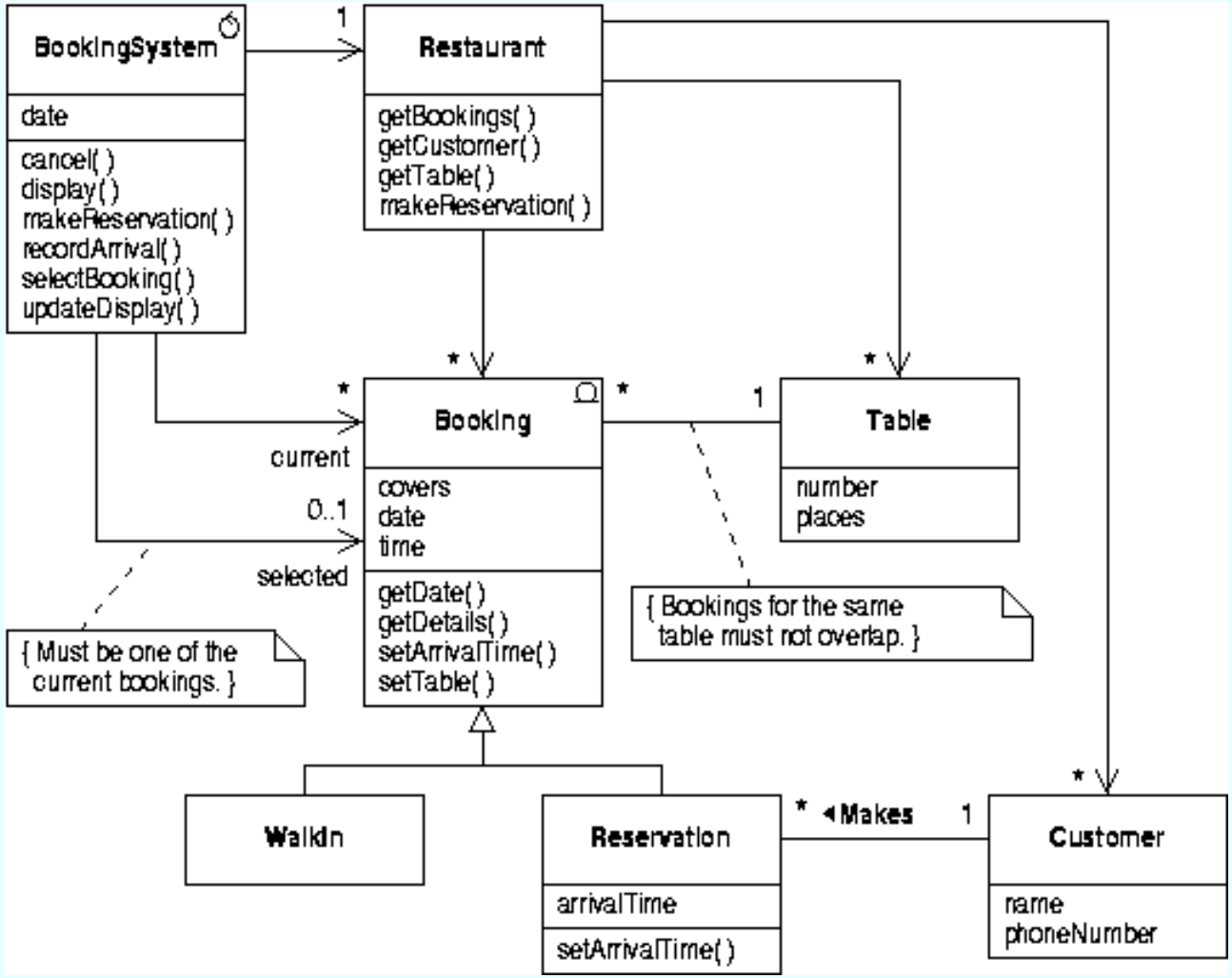
Refined Class Hierarchy



Summary

- Analysis has led to:
  - a set of use case realizations
  - a refined class diagram
- We can see how the class design is going to support the functionality of the use cases
- This gives confidence that the overall design will work

Complete Analysis Class Model



Analysis Exercises

- Draw a squence diagram realising the ‘Record Walk-in’ use case.
- Produce a sequence diagram showing a realisation of the basic flow of events for the ‘Transfer Table’ use case.
- Produce a sequence diagram showing a realisation for making an extended reservation (> 2 hours)
- Produce a sequence diagram showing a realisation for a reservation with multiple tables.
- Do the last realisations lead to any change in the class diagram?