# Table Booking System: Domain Model

Comp 47480: Object Oriented Design

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

# Table Booking System: Where are we?

- We have built the Use Case Model for this iteration.
  - so we know **what** the system is to do

- We now look at the essential classes in the system and their relationships.
  - this is called the **domain model**
  - = a first-cut class model

- In this phase we try to determine the **essential domain abstractions**, their key attributes, and how they are related to each other.

# Domain Modelling

- Use UML to construct a model of the real-world system
  - First-cut class model
  - Similar to Entity-Relationship (ER) Modelling in database
  - Not thinking of software implementation yet

- Model recorded as a **UML class diagram**

- Encourages a "seamless" approach to development
  - same notation used for analysis and design
  - design can evolve from initial domain model
  - mapping to real software is fairly direct
    - (may see more on this topic in later lectures)

# Domain Model Notation

- Subset of class diagram notation
- **classes** represent real-world entities
- **associations** represent relationships between the entities
- **attributes** represent the data held about entities
- **Generalisation/specialisation** can be used to simplify the structure of the model
- Don't worry about **methods** at this stage; they'll come in later in the modelling process

# Getting Started...

- First step is to identify a set of candidate classes
- Try the **noun-identification technique**
  - Nouns in the problem description suggest candidate classes
  - Many of the nouns will not be useful classes of course.
  - There may even be key abstractions that are hidden in the description

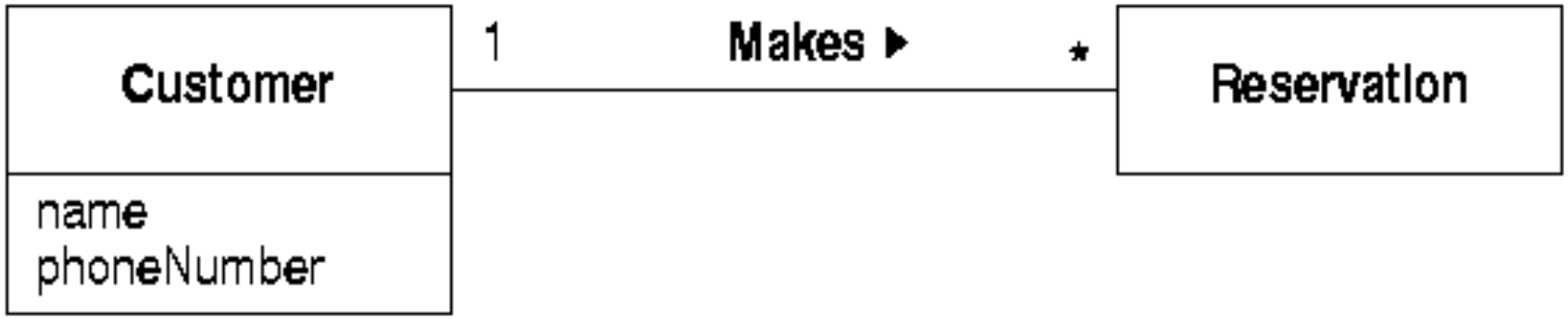
# Possible Classes

- This list of nouns was extracted using an automated tool from the problem description. Which of them would make good classes?
  - Restaurant
  - Table
  - Booking
  - Diner
  - Booking Form
  - Time Period
  - Contact Name
  - Reservation
  - Customer
  - Cover
  - Party
  - Walk-In
  - Sheet

If in doubt, probably best to leave it *in*.

# Modelling Customers and Reservations

- Customers make reservations



# Defining an Association

- If useful, give a name to the association
  - use a verb so that the relationship can be read as a sentence

- A customer can make many reservations
  - what consequences if only one was permitted?
  - should zero be permitted?

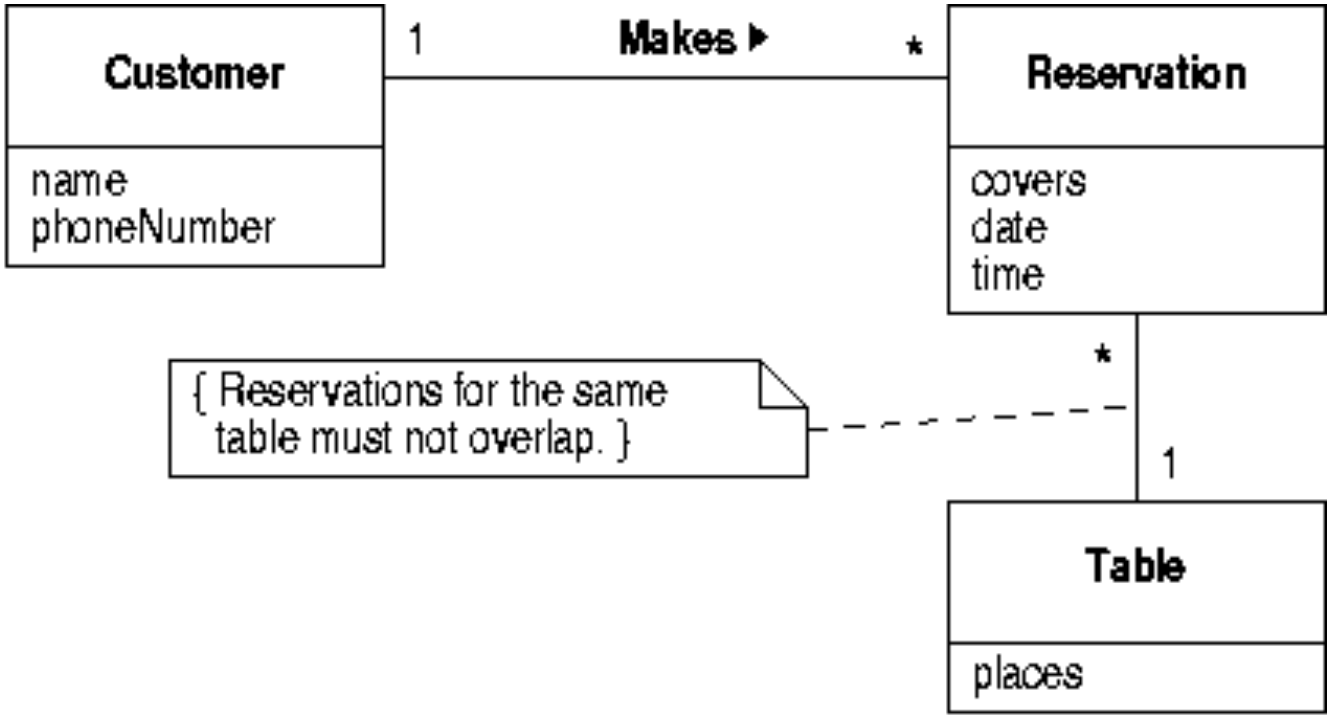
- How many people make a reservation?
  - one principal contact whose details are held
  - how about the other diners?

# How to model Table?

- We *could* model Table as an attribute of Reservation
- So for example, Reservation would contain an attribute `no_of_places` to store the size of the table that has been reserved.
- Why is this a bad idea?

# A better solution for Table

- Table is better modelled as a separate class
  - tables exist even if there are no reservations
  - other attributes of tables, e.g. size, location in restaurant, can be stored



# Constraining the model

- Not all domain properties can be shown graphically
  - e.g. it should be impossible to double-book a table
  - other hidden constraints on the previous diagram?

- Constraints add information to models
  - written in a note connected to the model element being constrained
  - OCL (**Object Constraint Language**) may also be used

# Object Constraint Language

- OCL is a declarative language based on first-order logic and set theory that is used to add constraints to UML diagrams
  - some constraints cannot be expressed (easily) in UML notation
  - OCL is a formal language with a precise semantics

- An example: "Every Reservation must be associated with a Customer who has a phone number" could be expressed as:

```
context Reservation
inv self.customer.phoneNumber != nil
```

- You should know what OCL is, though it's not widely used.

- Use clear English for constraints in this module.

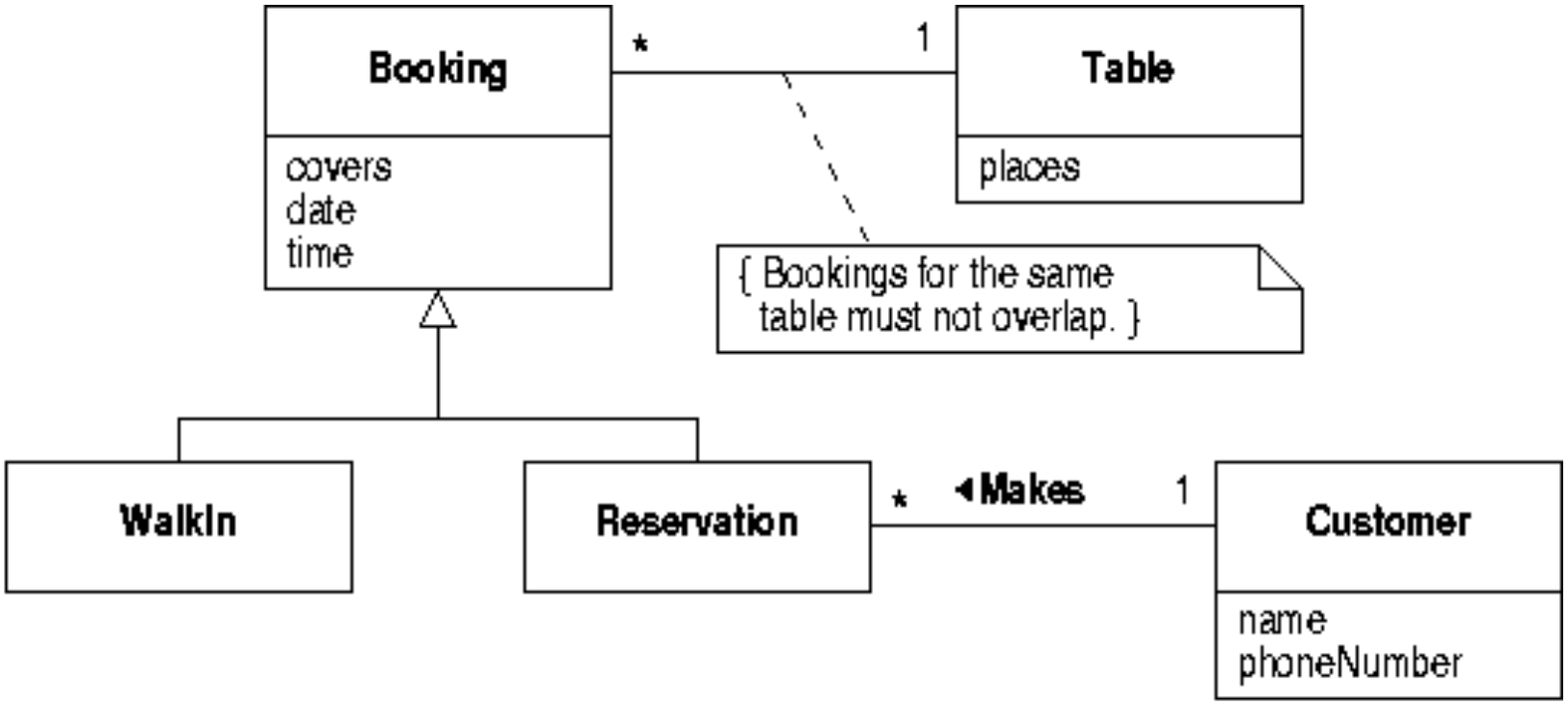
# How to model a “walk in” reservation?

- A Walk-in is where a customer arrives and wants a table but has no reservation.

- How do we fit this into our class model elegantly?

# Using Generalisation

- We introduce a new class, **Booking**, and use specialisation to model the properties shared by different types of Booking.




- Note that Booking and Reservation are normally synonyms

# Correctness and Completeness

- How do we know if a domain model is **correct**?
  - there are many plausible models, none is "correct"
- How do we know when a domain model is **complete**?
  - we don't: we stop when we think we've done enough
- Domain modelling is not an end in itself, but a guide to further development
- Realising use cases tests the domain model, and will usually lead to refinements
  - We'll see this later on when we look at **sequence diagrams**

# Partial Restaurant Glossary

- **Booking**: an assignment of diners to a table
  - contrast Reservation
- **Covers**: the number of diners for a booking
- **Customer**: a person who makes a reservation
- **Reservation**: a booking made in advance
- **Walk-in**: a booking that is made only when the diners arrive to eat

# Summary

- We have completed the requirements workflow for one iteration in the development of a restaurant table booking system.

- The two UML models that were introduced are:
  - Use Case model
  - Class Model

- We now proceed by
  - looking at the UML Class Model in more detail
  - building sequence diagrams for the identified use cases