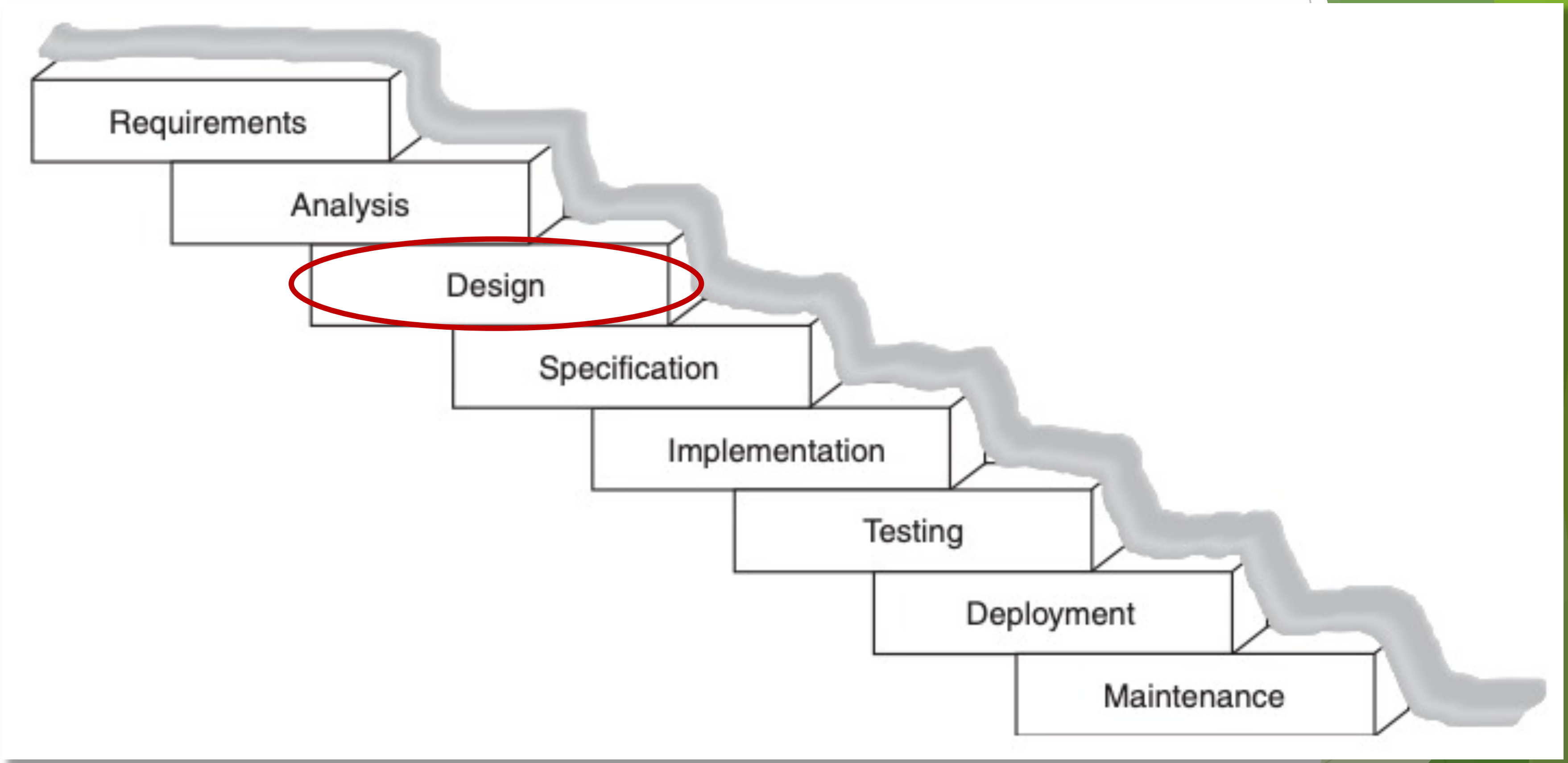
12 System Design (2)

Introduction to OOA OOD and UML

2022 Spring

College of Information Science and Engineering
Ritsumeikan University
Yu YAN

Where Are We Now?



Outline

- Introduction
- Mapping the Analysis Class Model into the Design Class Model
- Handling Persistence with A Relational Database
- Finalizing the User Interfaces
- Using Patterns, Frameworks and Libraries
- Handling Multiple Activities
- Exercise 12

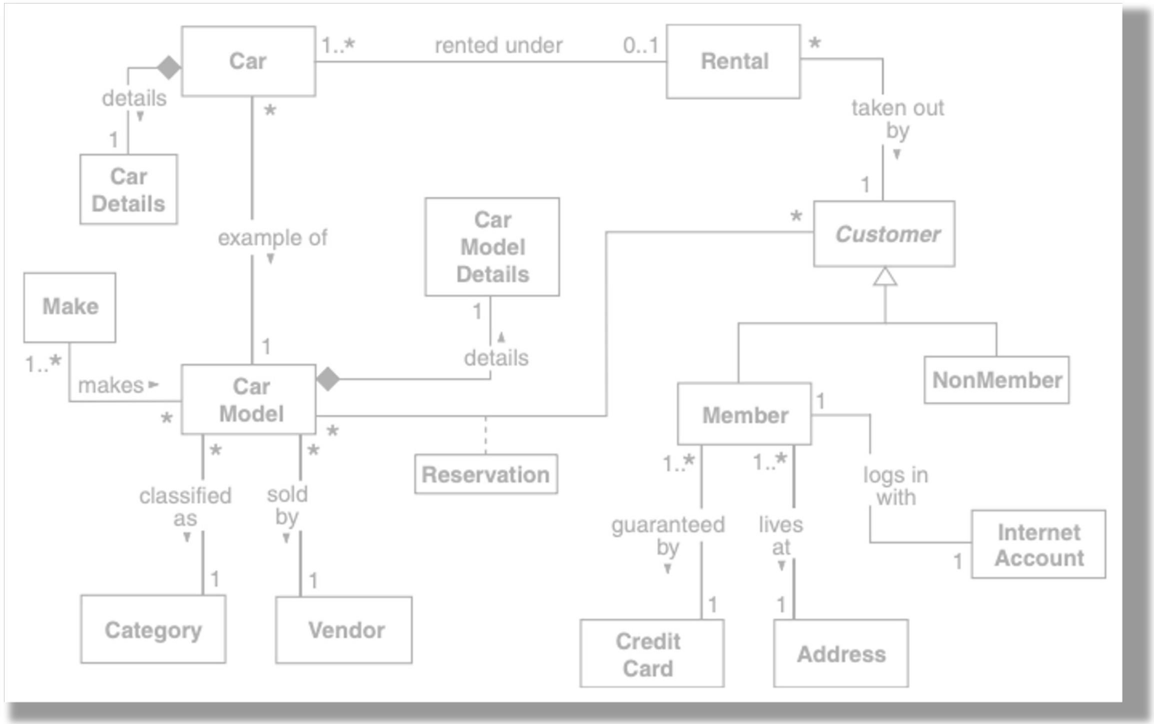
- Subsystem design involves transforming a conceptual analysis model into implementable classes
- The subsystem design can proceed as follows:
  - Design the classes and fields of the business layer, using the analysis class model as a guide
  - Decide how any persistent data will be stored and design the storage layout
  - Finalize the look and feel of the user interface, with reference to the sketches that were produced during the analysis phase
  - Walk through the system use cases, with reference to the user interface design, noting the business services that must be supported by the middle tier. Business services are questions and commends that a client can send to the server
  - Develop the business services into server objects, whose messages are available over the network. Server objects implement the business services using the business layer
  - Finalize the measures that are needed to ensure concurrency control and thread safety

Outline

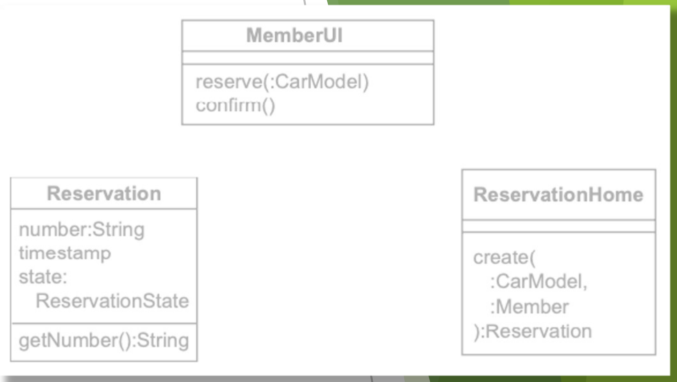
- Introduction
- Mapping the Analysis Class Model into the Design Class Model
- Handling Persistence with A Relational Database
- Finalizing the User Interfaces
- Using Patterns, Frameworks and Libraries
- Handling Multiple Activities
- Exercise 12

Mapping Operations

- Now that we're switching to design, we can stick to the programming terms 'message' and 'method', rather than the UML term 'operation'
- Messages will be added for one or more because of the following reasons:
  - To allow client objects to read or change the values of fields
  - To allow client objects to access derived data
  - Because our experience or intuition tell us that a particular message might be useful
  - Because some framework or pattern that we have decided to use requires certain messages to be present



Car
1..*
rented under
0..1
Rental
*
taken out by
1
Customer
*
details
1
Car Details
*
example of
1
Car Model Details
1
details
Make
1..*
makes
*
Car Model
*
*
*
classified as
1
Category
sold by
1
Vendor
Reservation
NonMember
Member
1
1..*
guaranteed by
1
Credit Card
1..*
lives at
1
Address
logs in with
1
Internet Account



MemberUI

reserve(:CarModel)
confirm()

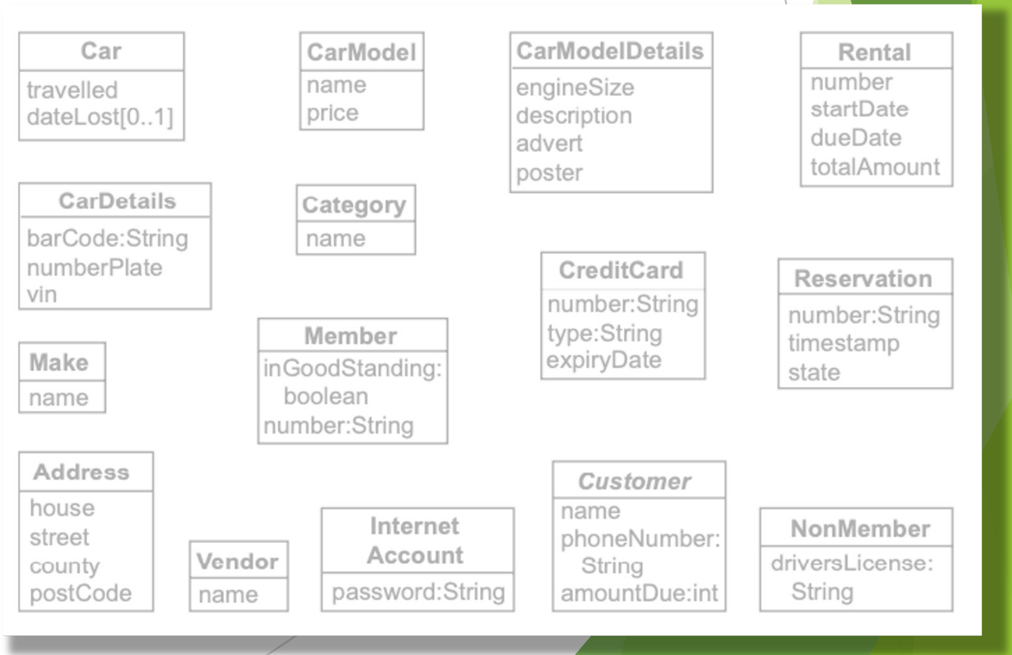
Reservation

number:String
timestamp
state:
ReservationState

getNumber():String

ReservationHome

create(
:CarModel,
:Member
):Reservation



Car
travelled
dateLost[0..1]

CarModel
name
price

CarModelDetails
engineSize
description
advert
poster

Rental
number
startDate
dueDate
totalAmount

CarDetails
barCode:String
numberPlate
vin

Category
name

CreditCard
number:String
type:String
expiryDate

Reservation
number:String
timestamp
state

Make
name

Member
inGoodStanding:
boolean
number:String

Address
house
street
county
postCode

Vendor
name

Internet Account
password:String

Customer
name
phoneNumber:
String
amountDue:int

NonMember
driversLicense:
String

Variable Types - Review

- When we introduce a field, we need to decide what type it is: a primitive or a class:
  - The primitive could include:
    - boolean {true, false},
    - byte {0~255, can be stored in 8-bit byte},
    - char {value from 0 to 255, interpreted as characters}
    - double {4.9E-324},
    - float {1.4E-45, 3.4028235E38, 1.7976931348623157E308},
    - int {-2147483648, 2147483647, stored in 4 bytes},
    - long {-9223372036854775808, 9223372036854775807, 8 bytes},
    - short {-32768, 32767, stored in 2 bytes}
  - Classes that we ourselves are designing
  - Classes from the pattern and frameworks that we have chosen to use

Visibility of Fields - Review

- As well as providing the names and types of fields, we must declare their visibility
- The visibility of a field specifies which pieces of code are allowed to read or modify the value
  - Private (shown by "–" in UML): Only visible with in the defining class
  - Package (shown by "~" in UML): Visible within the defining class and to all classes in same package
  - Protected (shown by "#" in UML): Visible within the defining class, to all classes in the same package, and to all subclasses of the defining class (whether inside or outside the package)
  - Public (shown by "+" in UML): Visible everywhere

Visibility of Messages - Review

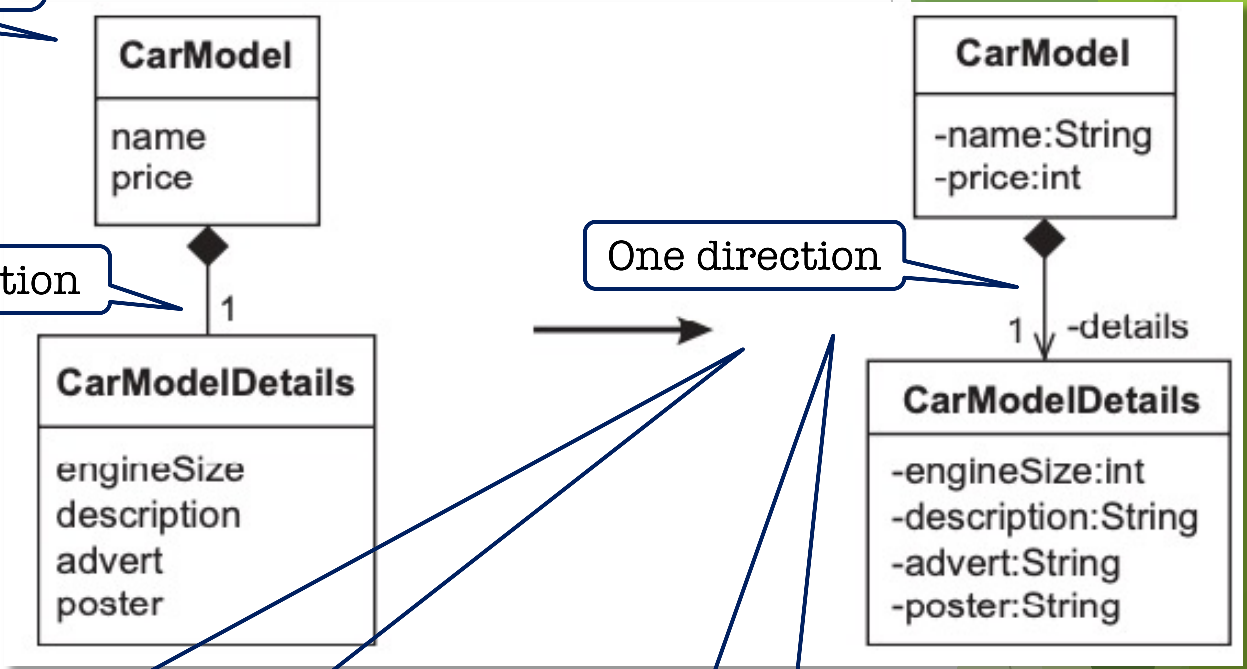
- Visibilities can be applied to messages too:
  - Public: if it's part of the interface of the package
  - Package: if it's implementation code to be used by the class itself and by classes in the same package
  - Protected: if it's implementation code to be used by the class itself, by its subclasses and by classes in the same package
  - Private: if it's implementation code for use by this class only (which decreases coupling and allows the compiler to do more optimizations, as with fields)

Accessors - Review

- Accessor messages come in two varieties:
  - getters: it returns the value of a field
  - setters: it sets the field to a new value

Mapping Classes, Attributes and Compositions

Mapping attributes and compositions from analysis



Bi-direction

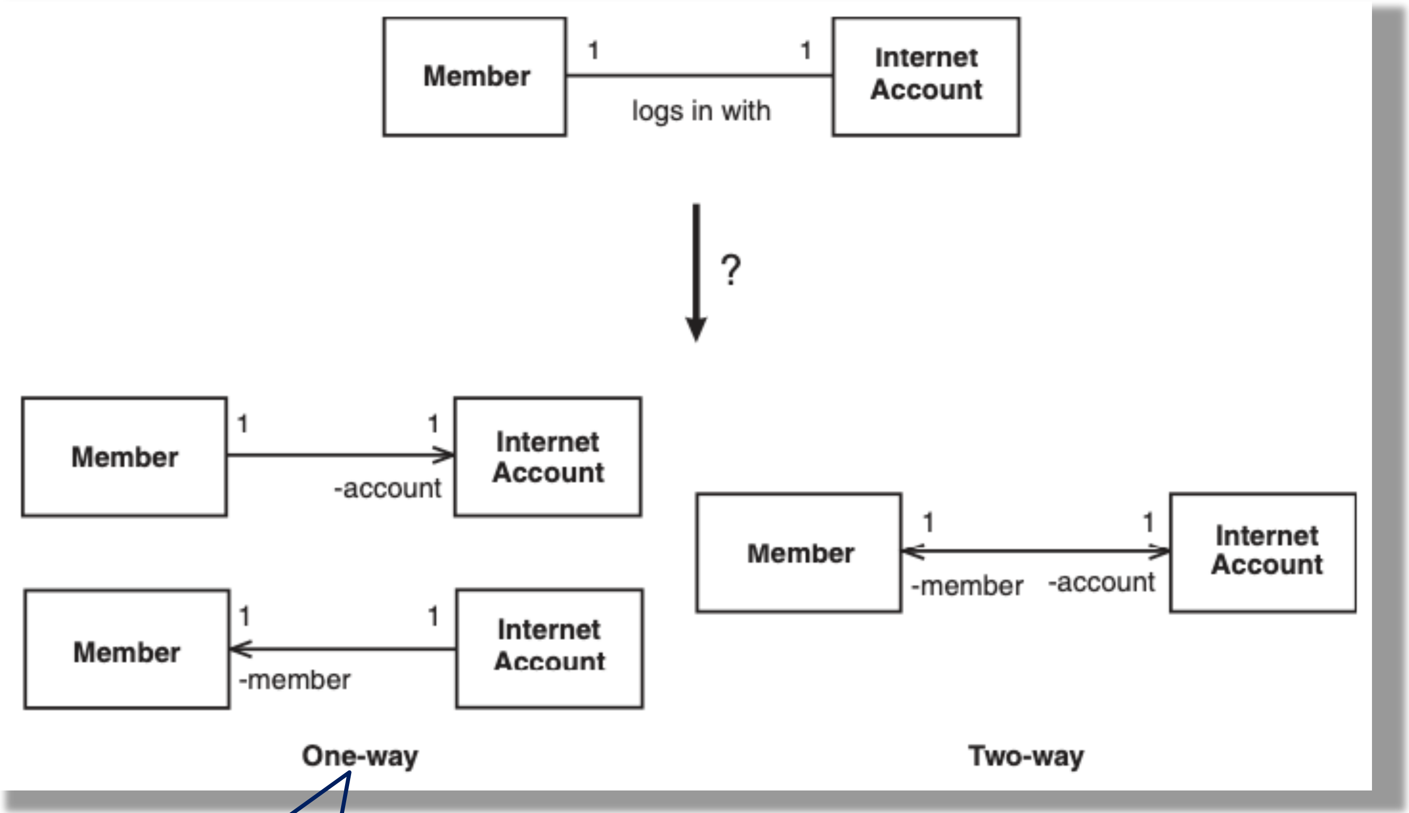
One direction

Once we have decided the direction of a composition, we can add an arrow-head to the class diagram to show it, along with a role and a visibility to indicate how the composition maps to a field

Messages can only flow into the direction of arrow

Mapping Other Types of Association (1)

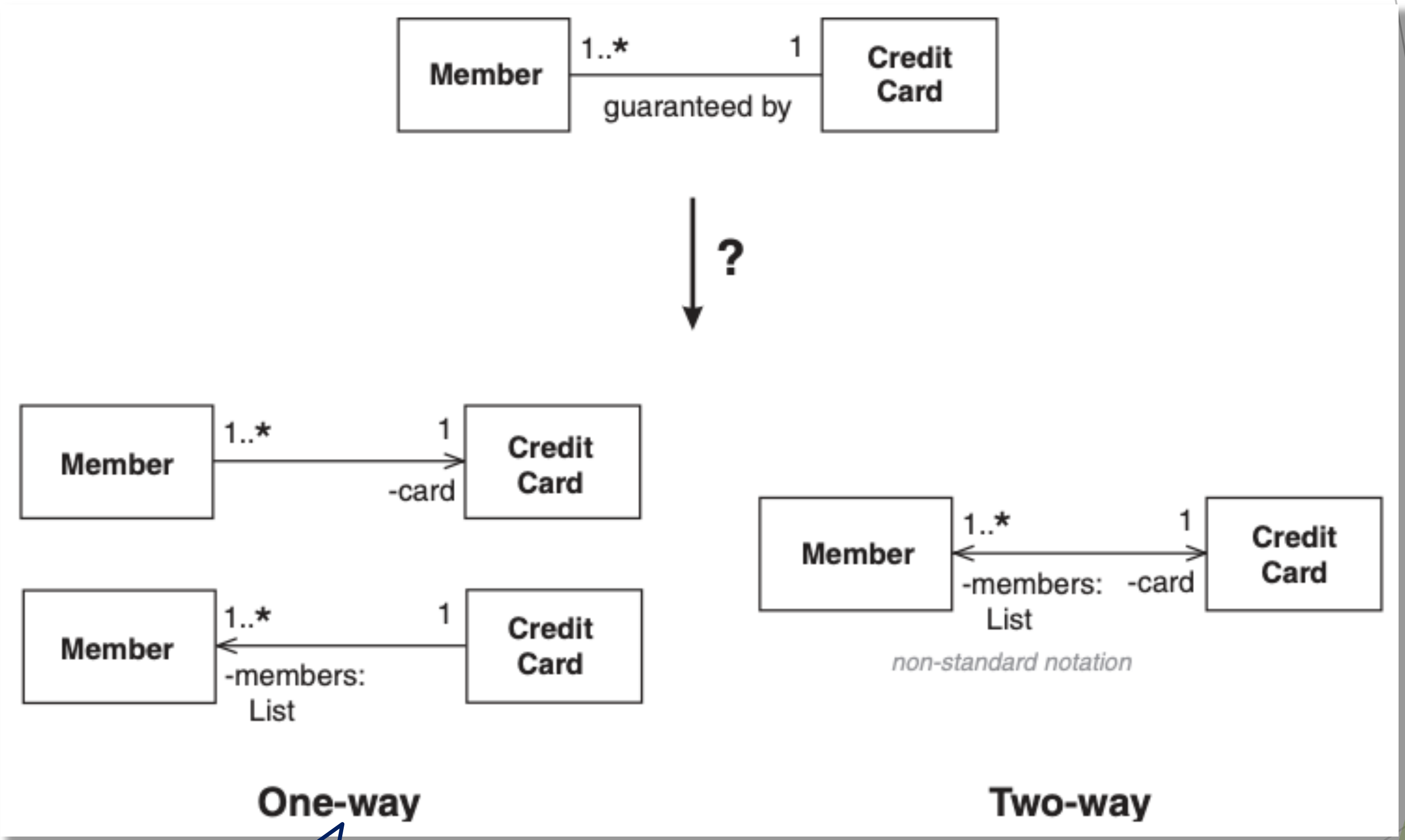
- One-to-One Association



One-way is preferred

Mapping Other Types of Association (2)

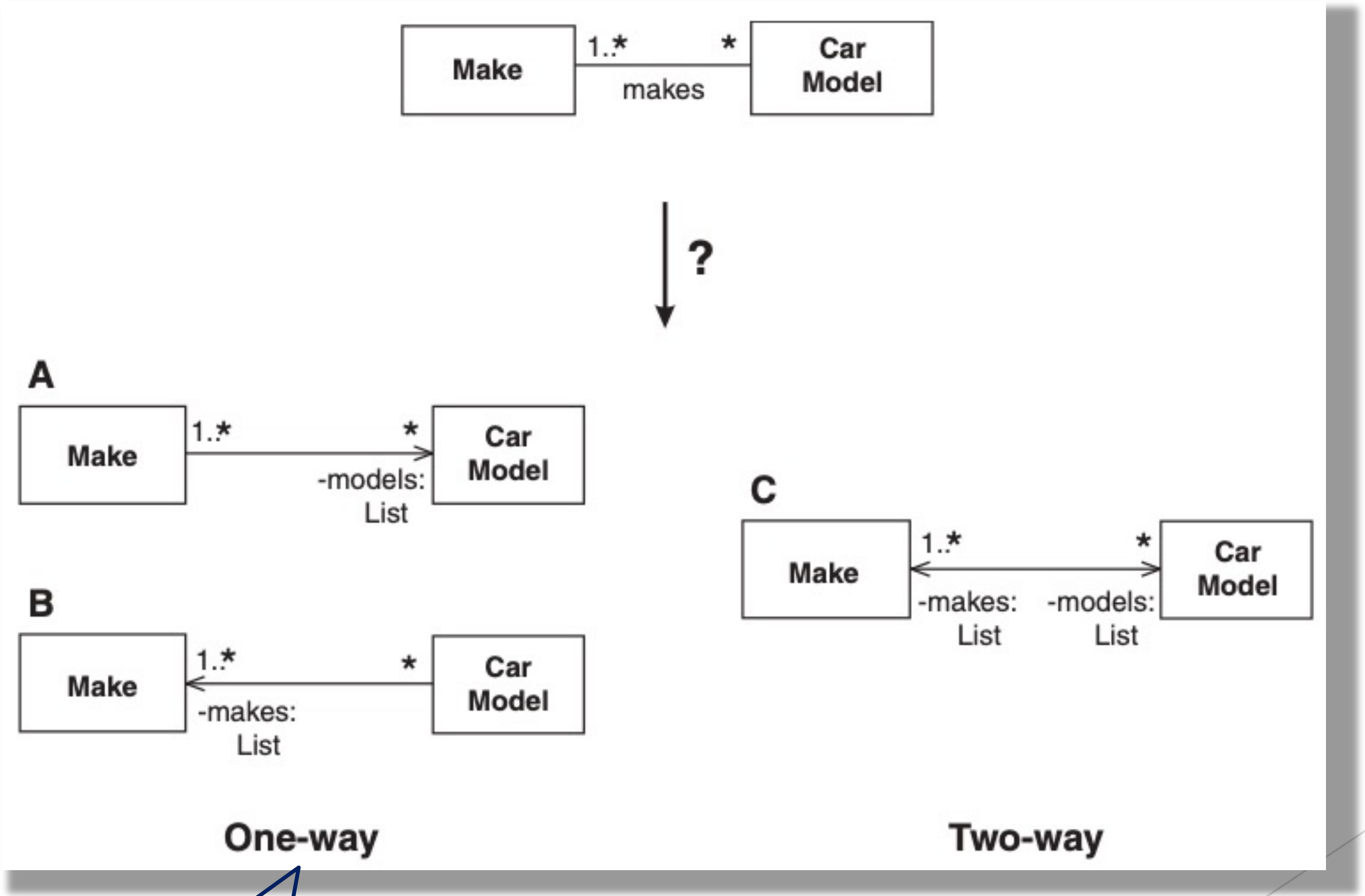
- One-to-Many Association



One-way is preferred

Mapping Other Types of Association (3)

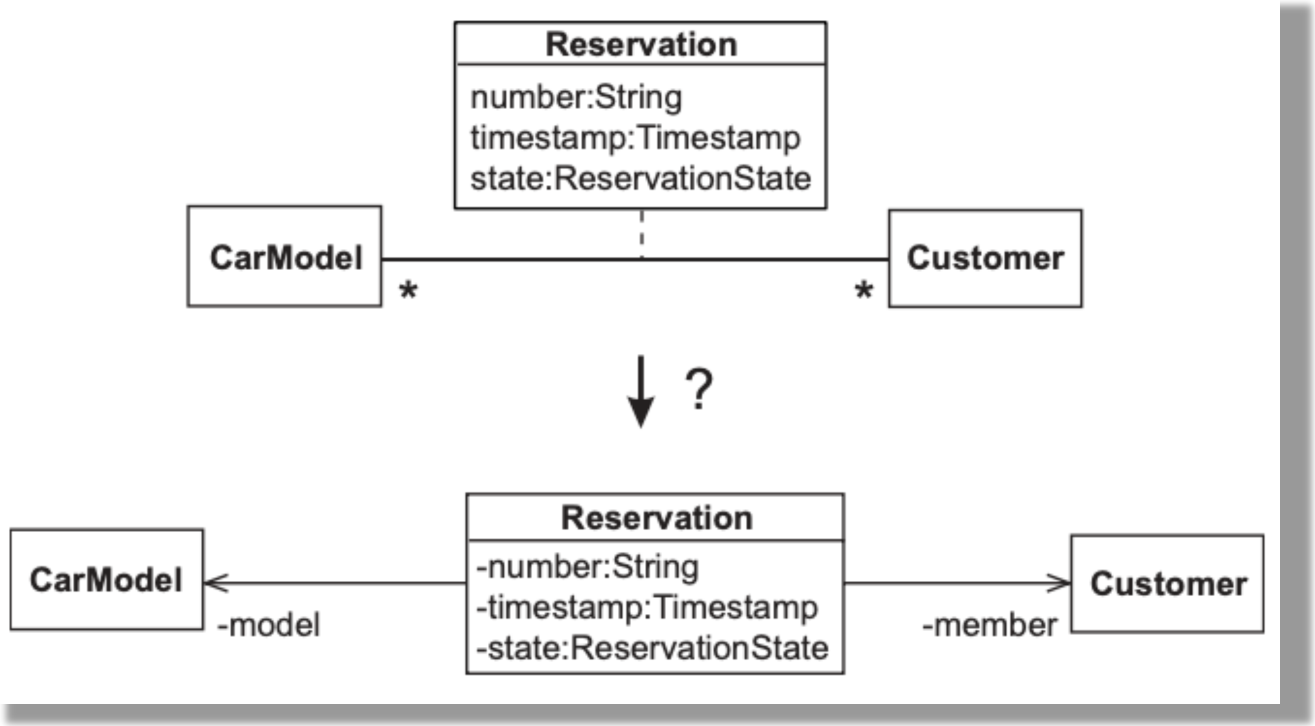
- Many-to-Many Association



One-way is preferred

Mapping Other Types of Association (4)

- Association Classes



Outline

- Introduction
- Mapping the Analysis Class Model into the Design Class Model
- Handling Persistence with A Relational Database
- Finalizing the User Interfaces
- Using Patterns, Frameworks and Libraries
- Handling Multiple Activities
- Exercise 12

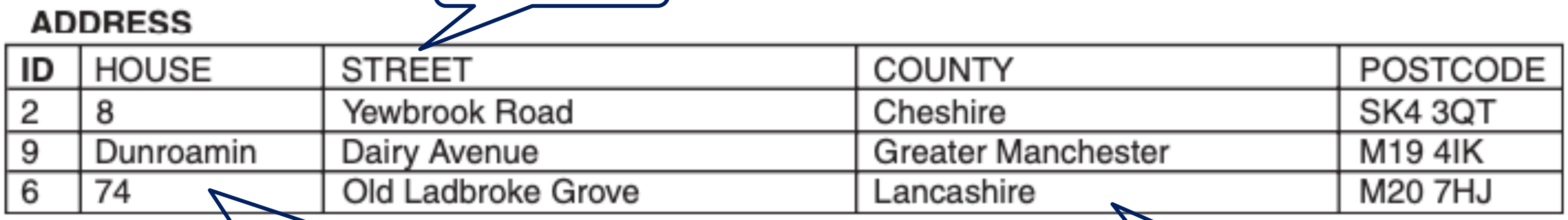
Database Management Systems

- A **database management system (DBMS)** manages arbitrary amounts of data in separate databases
- We use DBMS to:
  - Store the data using **Data Definition Language (DDL)**
  - Add, remove and update the data in our database using a **Data Manipulation Language (DML)**
  - Retrieve the data from our database using a **Data Query Language (DQL)**
- There are many varieties of DBMS: **indexed file**, **hierarchical**, **network**, **relational** and **object-oriented**
- A **relational database** is often used to store the data in a multi-tier Internet system
  - **The Structured Query Language (SQL)**

The Relational Model (1)

- The relational model is based on **tables (relations)** of data which contain **columns** and **rows**

Attribute



ADDRESS

ID	HOUSE	STREET	COUNTY	POSTCODE
2	8	Yewbrook Road	Cheshire	SK4 3QT
9	Dunroamin	Dairy Avenue	Greater Manchester	M19 4IK
6	74	Old Ladbroke Grove	Lancashire	M20 7HJ

Each column stores the values of each attribute

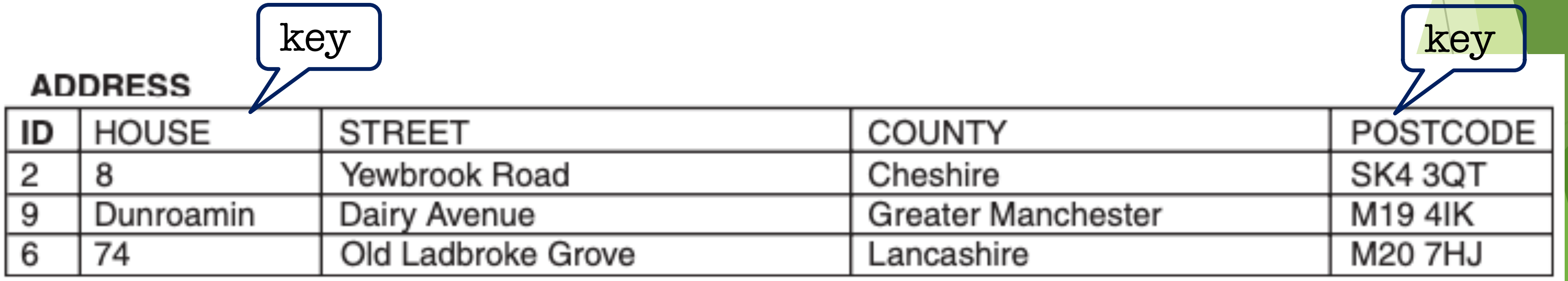
Figure 10.8: An ADDRESS table

Each row stores each data

- SQL standard defines a couple of dozen types:
  - **VARCHAR(X):** A string, up to a maximum of X characters
  - **INTEGER:** A whole number which is often, but not always, 32 bits
  - **DATA:** A day in the Gregorian calendar
  - **TIMESTAMP:** A combination of data and time of day
  - **BOOLEAN:** True or false

The Relational Model (2)

- With the relational model, every row must be unique
- A **key** is a value, or a combination of values, that uniquely identifies a row
- A **compound key** is a key that combines several values
- Some tables have more than one **candidate key**. In such cases, we have to choose one candidate to act as the **primary key**



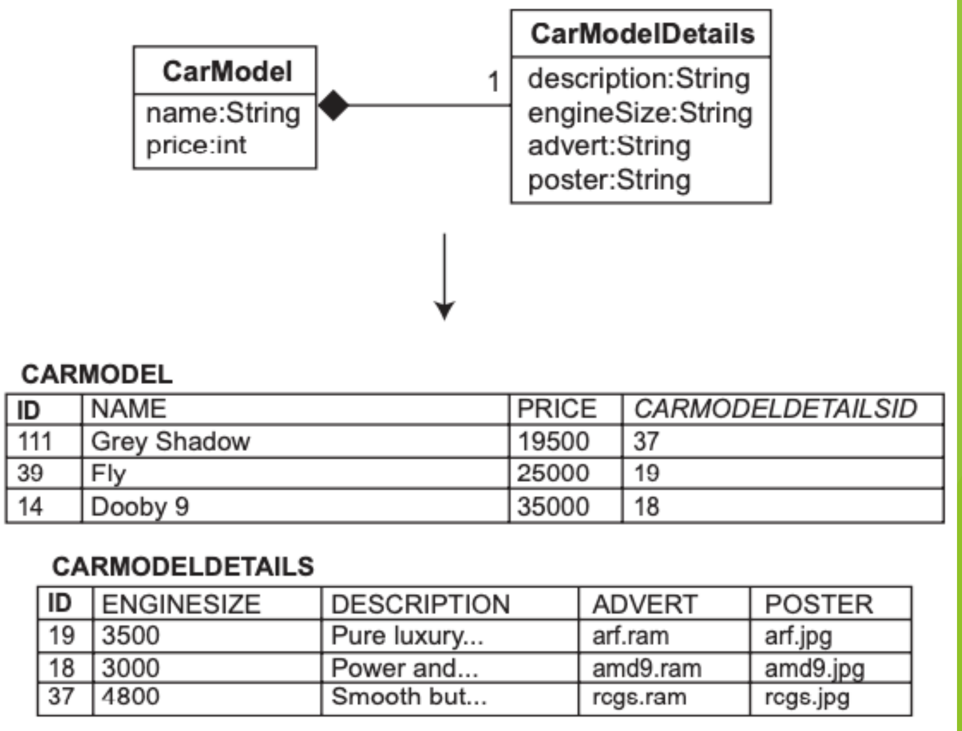
ADDRESS

ID	HOUSE	STREET	COUNTY	POSTCODE
2	8	Yewbrook Road	Cheshire	SK4 3QT
9	Dunroamin	Dairy Avenue	Greater Manchester	M19 4IK
6	74	Old Ladbroke Grove	Lancashire	M20 7HJ

Figure 10.8: An ADDRESS table

Mapping an Object Model to a Relational Model (1)

- We usually start with either the analysis class diagram or the design class diagram
- We need to introduce a table with the same name as the entity's class
- Each row in an entity table represents a unique object from the business domain
- Each field is mapped as an attribute in the entity table



CARMODEL

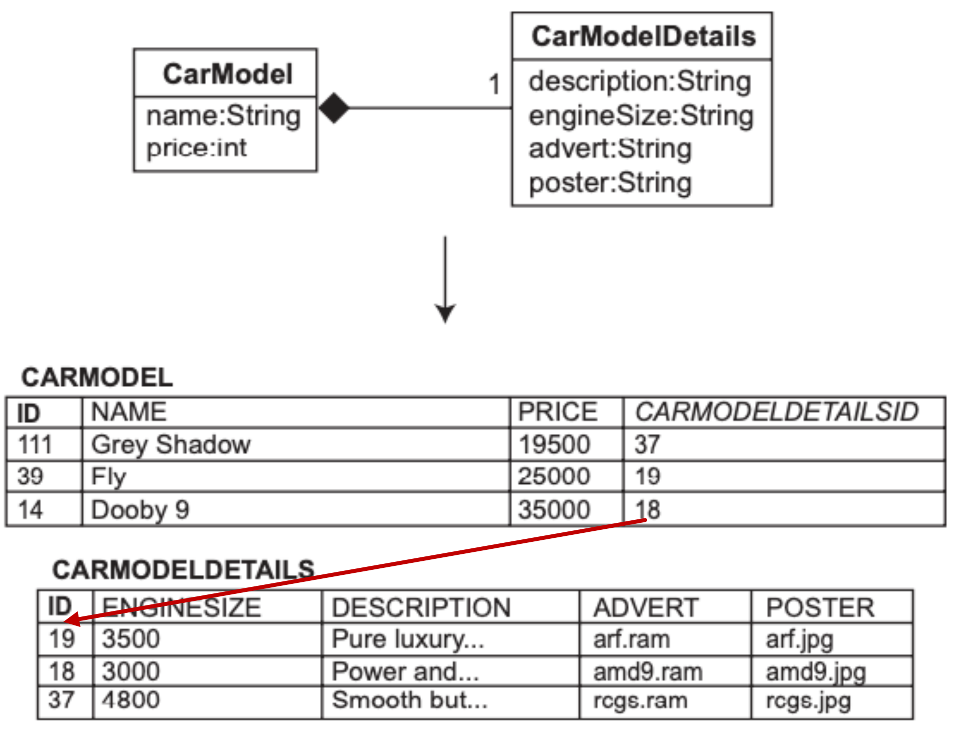
ID	NAME	PRICE	*CARMODELDETAILSID*
111	Grey Shadow	19500	37
39	Fly	25000	19
14	Dooby 9	35000	18

CARMODELDETAILS

ID	ENGINESIZE	DESCRIPTION	ADVERT	POSTER
19	3500	Pure luxury...	arf.ram	arf.jpg
18	3000	Power and...	amd9.ram	amd9.jpg
37	4800	Smooth but...	rcgs.ram	rcgs.jpg

One-to-One Associations

- For a one-to-one association, we can add a **foreign key** to one of the entity tables
- A foreign key is an entry in one table that refers to a primary key in another table
- A foreign key is a reference from a row in one table to a row in another table
- To emphasize which column represent foreign keys, the column names are shown in italics



CARMODEL

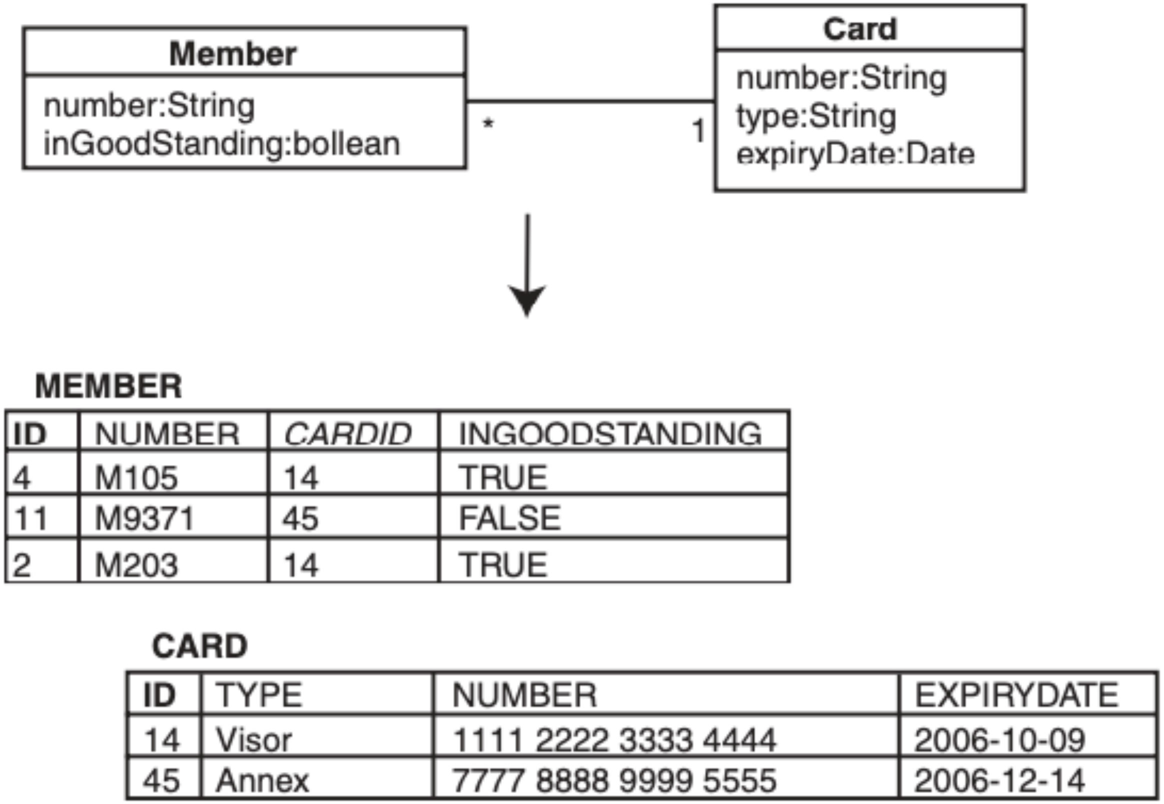
ID	NAME	PRICE	*CARMODELDETAILSID*
111	Grey Shadow	19500	37
39	Fly	25000	19
14	Dooby 9	35000	18

CARMODELDETAILS

ID	ENGINESIZE	DESCRIPTION	ADVERT	POSTER
19	3500	Pure luxury...	arf.ram	arf.jpg
18	3000	Power and...	amd9.ram	amd9.jpg
37	4800	Smooth but...	rcgs.ram	rcgs.jpg

One-to-Many Associations

- For a one-to-many association, we can add a **foreign key** to the "many" table



MEMBER

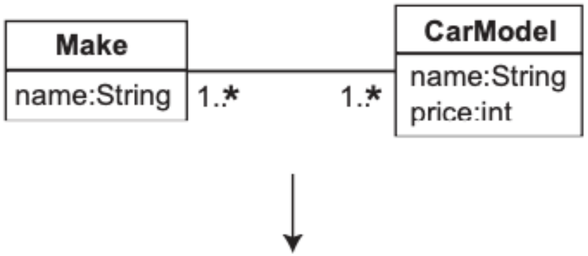
ID	NUMBER	*CARDID*	INGOODSTANDING
4	M105	14	TRUE
11	M9371	45	FALSE
2	M203	14	TRUE

CARD

ID	TYPE	NUMBER	EXPIRYDATE
14	Visor	1111 2222 3333 4444	2006-10-09
45	Annex	7777 8888 9999 5555	2006-12-14

Many-to-Many Associations

- A link table is necessary to represent a link between an entity in one table and an entity in another table
- Association classes, since they have data of their own, must be mapped to link tables



CARMODEL

ID	NAME	PRICE	*CARMODELDETAILSID*
111	Grey Shadow	19500	37
39	Lacrosse	25000	19
14	Dooby 9	35000	18

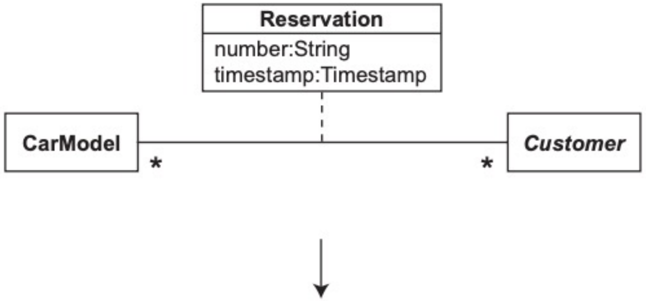
MAKECARMODEL

MAKEID	*CARMODELID*
8	111
65	14
9	39
8	39

Link table

MAKE

ID	NAME
65	Astra Marten
9	Alpha Rodeo
8	Rolls Choice

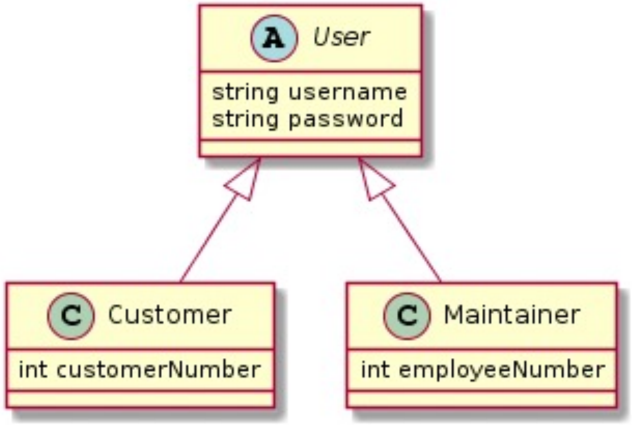


RESERVATION

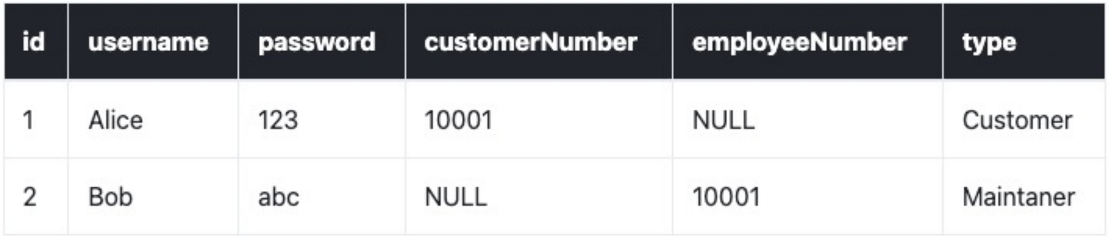
ID	*CARMODELID*	*CUSTOMERID*	NUMBER	TIMESTAMP
7	33	4	R187a	2004-12-06 14:23:16.543
1	22	2	R7b	2004-12-03 00:03:21.872
99	72	4	R459b	2004-12-05 09:45:07.210

Mapping Inheritance

- There are three types of table to map

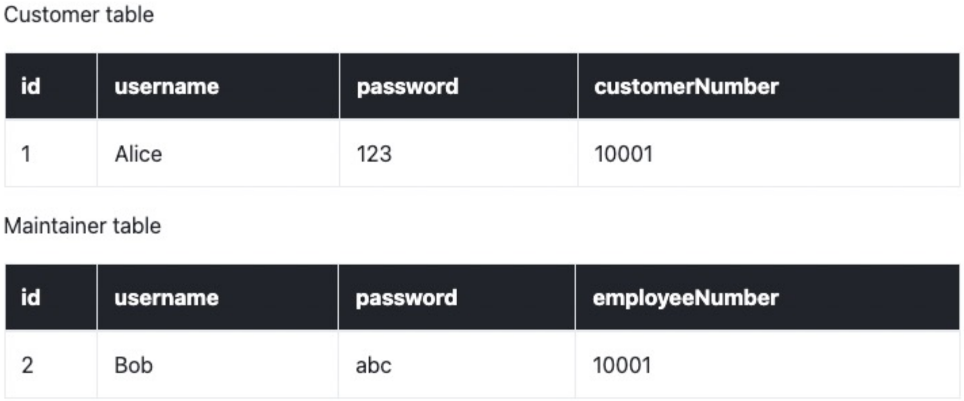


- Single Table



id	username	password	customerNumber	employeeNumber	type
1	Alice	123	10001	NULL	Customer
2	Bob	abc	NULL	10001	Maintaner

- Concrete Table Inheritance



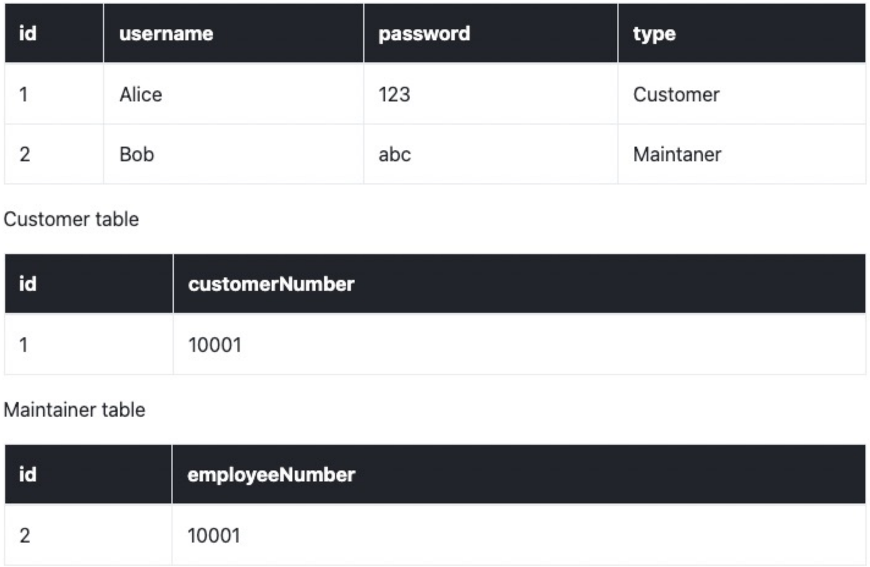
Customer table

id	username	password	customerNumber
1	Alice	123	10001

Maintainer table

id	username	password	employeeNumber
2	Bob	abc	10001

- Class Table Inheritance



id	username	password	type
1	Alice	123	Customer
2	Bob	abc	Maintaner

Customer table

id	customerNumber
1	10001

Maintainer table

id	employeeNumber
2	10001

Outline

- Introduction
- Mapping the Analysis Class Model into the Design Class Model
- Handling Persistence with A Relational Database
- Finalizing the User Interfaces
- Using Patterns, Frameworks and Libraries
- Handling Multiple Activities
- Exercise 12

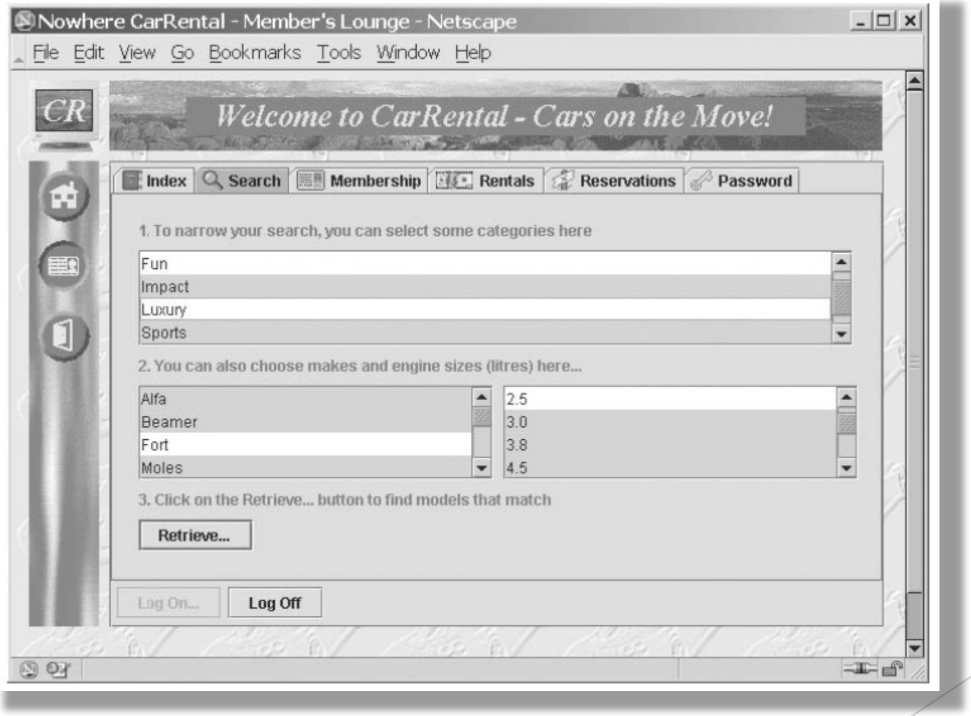
- We need to **design** the user interfaces:
  - We have to take the coarse boundary objects, the vague user interface sketches and the precise use cases, and transform them into user interface descriptions that can be implemented directly

Basic Principles for Good UI Design (1)

- Be guided by use cases
  - Although we would expect each user interface to represent a number of use cases, the interfaces themselves can still be simple
  - We would expect the grouping of related use cases to be reflected in the structure of the user interface
  - We should also avoid splitting a single use case, or a chain of related use cases, into more than one interface
- Keep it simple
  - Many of our users will be novices, especially those users accessing our system over the Internet, so simplicity is very important.
  - There's a strong argument for keeping user interfaces simple and uncluttered for expert users too: we don't want the overwhelming functionality of our user interface to hamper their productivity.

Basic Principles for Good UI Design (2)

- Use notebooks
  - A notebook is a set of pages that shows only one page at a time – the other pages are available via tabs along the edge
  - The advantage of a notebook is that, because its size and location stays the same, the user can focus on a single area of the screen, even for long interactions that involve more than one use case. This improves the **user experience** and also **user productivity**



1. To narrow your search, you can select some categories here

Fun
Impact
Luxury
Sports

2. You can also choose makes and engine sizes (litres) here...

Alfa
Beamer
Fort
Moles

2.5
3.0
3.8
4.5

3. Click on the Retrieve... button to find models that match

Basic Principles for Good UI Design (3)

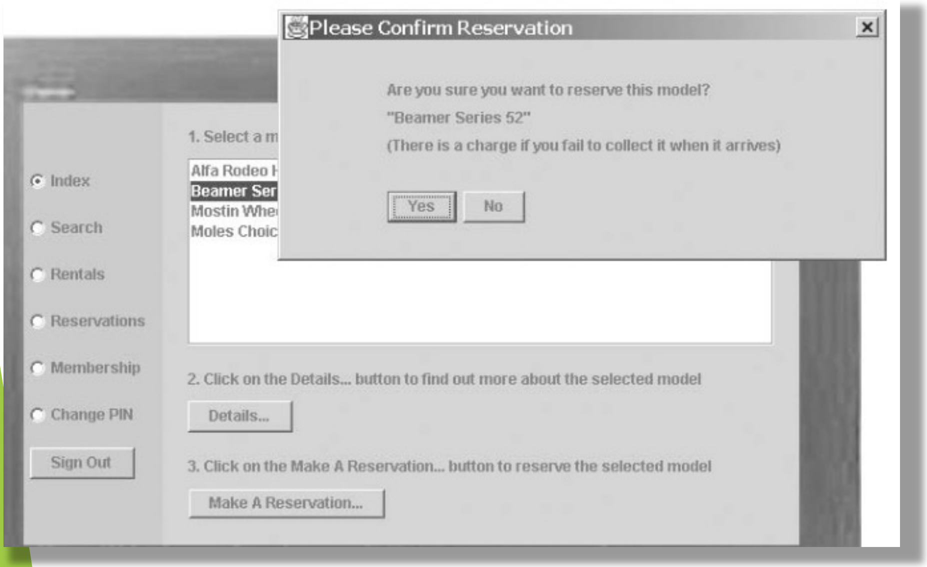
- Use wizards
  - wizard is a sequence of pages that guides a user, step by step, through a complex activity

When the user selects an activity, they're presented with a page representing "step 1"; the user completes "step 1" and clicks the "Next" button to go on to "step 2"; and so on. Before completing an activity, a user can revisit the earlier steps, using the "Back" button.

  - Each page of the wizard contains fields, lists and other widgets that the user employs to enter the data needed for the current step
  - The step-by-step nature of the wizard and the simple instructions allow the user to perform a complex activity without having to remember how to do it

Basic Principles for Good UI Design (4)

- Avoid multiple windows
  - Multiple, overlapping windows were invented for the benefit of computer experts rather than novice users. Even if our user is sufficiently computer-literate to open a Web browser, we should not expect them to be able to cope with multiple browser windows or pop-up dialogs



Please Confirm Reservation

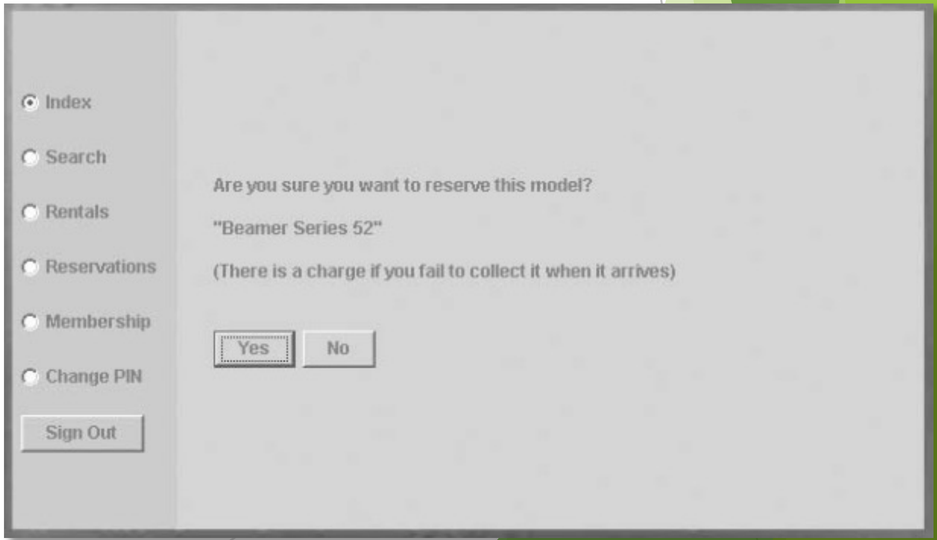
Are you sure you want to reserve this model?
"Beamer Series 52"
(There is a charge if you fail to collect it when it arrives)

1. Select a m

Alfa Rodeo I
Beamer Ser
Mostin Whe
Moles Choic

2. Click on the Details... button to find out more about the selected model

3. Click on the Make A Reservation... button to reserve the selected model



Are you sure you want to reserve this model?

"Beamer Series 52"

(There is a charge if you fail to collect it when it arrives)

Outline

- Introduction
- Mapping the Analysis Class Model into the Design Class Model
- Handling Persistence with A Relational Database
- Finalizing the User Interfaces
- Using Patterns, Frameworks and Libraries
- Handling Multiple Activities
- Exercise 12

- A pattern is a portable solution to a small programming problem
  - Patterns allow developers to work faster and produce better code by not duplicating effort
  - Some fundamental patterns: singletons (where a class has only one instance); factories (for creating objects); homes (for creating objects and finding existing ones); and states (representing an object's life cycle), etc.
- A framework is a way of putting together part of a system
  - Frameworks tend to be much larger
  - Some of the code is already written for you (this may take the form of partially implemented classes or code-generation tools)
- A library is a collection of prewritten classes that can be used off-the-shelf
  - For example: Numerical Recipes in C, VTK, The Java 2 Platform library

Outline

- Introduction
- Mapping the Analysis Class Model into the Design Class Model
- Handling Persistence with A Relational Database
- Finalizing the User Interfaces
- Using Patterns, Frameworks and Libraries
- Handling Multiple Activities
- Exercise 12

- Multitasking means: each program runs as an independent process with its own protected area of code (program instructions) and data (program variables)
- Multithreading means: each thread represents an activity within a program, running alongside other activities
  - Some programming languages allow us to execute multiple tasks within a single process. These tasks are usually referred to as threads of execution or just threads

Controlling Multiple Tasks

- Each process managed by the operating system can be idle (waiting for user input perhaps) or active (performing some computation)
- Because we usually have more processes than CPUs, the operating system must share the CPU time between the active process
  - The operating system allows each process to run for a small amount of time and then moves on to the next. This time-slicing is controlled by a piece of software called a scheduler
- As an extra facility, most operating systems allow us to assign a priority to each process, so that some processes get more of the available time than others

Controlling Multiple Threads

- Threads are different from processes in that they all share the same data area within their process
- As well as protecting external resources, the programmer has to protect internal data (of the same program)

Threads Safety

- Multi-threading causes problems, because threads can be interrupted before they're finished (to allow other threads to run). For example, consider the following scenario:
  - Two threads A and B are accessing an object O.

“Thread A starts to read on of O’s fields, F, using a getter method. When A has read half of the value, the scheduler interrupts it so that B can run for a while. B starts to modify F, via its setter. The scheduler allows B to finish its modification before it wakes up A. When A wakes up, it reads the rest of F”

  - Thread A has now read half of the old value and half of the new value, which is clearly nonsense. This kind of data corruption applies to external resources too (imagine if A were reading a text file and B were modifying it)

Thread synchronization

- We can solve most multi-threading problems by encapsulating each shared resource inside a single object. It is then the object's responsibility to make sure that only one thread is allowed in at a time. Preferably, the programming language should support this mutual exclusion.
  - ✓ For example, in Java, a method can be marked as synchronized: the run-time system guarantees that only one thread at a time can be active inside any of an object's synchronized

Outline

- Introduction
- Mapping the Analysis Class Model into the Design Class Model
- Handling Persistence with A Relational Database
- Finalizing the User Interfaces
- Using Patterns, Frameworks and Libraries
- Handling Multiple Activities
- Exercise 12

Exercise 12

- Please submit your answer file "UML_Ex12a_Your group names.pdf" to "Exercise 12a" and your answer file "UML_Ex12b_Your group names.pdf" to "Exercise 12b" under "Assignments" tab in Manaba +R
- The deadline for Ex12a is **2022/07/07 (Thur.) 9:00**
- The deadline for Ex12b is **2022/07/14 (Thur.) 9:00**
- The maximum points for "Exercise 12" will be **10p**
- If you put a wrong file name or wrong file format, your assignment will not be evaluated. Please be careful!

Ex12 (Group work)

- In the last two week, each group selected their own business and did the requirement analysis and the problem analysis for the business.
- This week, there are two tasks: (5p)

1. Please choose at least 4 pairs of classes in your analysis class diagram and map them into relational models
   - Since we have not learned any DBMS software, you can create a simple table in MS Word to make the entity tables
   - Please show each pair of classes and the corresponding entity tables in the report
2. Please design your own UI for your business (5p)
   - The UI should cover all of the use cases in the requirement analysis diagram
   - It is possible to group use cases into functions
   - There are some UI design software, please search online to choose one software to design
   - Please also clarify the UI design software that you used in the report