

INFO6030

System Analysis and Design

Welcome

Week 4

Detailed Use Case & Domain Class

“Detailing System Requirements”

“Functional Requirements”

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been copied and communicated to you by or on behalf of the University of Newcastle pursuant to Part VB of the *Copyright Act 1968* (the Act)

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Lecture Slide Content

- These lecture slides are the basis of the lecture.
- In other words, the actual lecture may cover more than what is contained in these slides.
- **Why**
 - A question may be asked that is related to an assessment
 - A point may be raised and then we chat about that idea
 - I may ask questions in the lecture and chat about the responses
 - I may related the content to assessments or current events
 - Example/s may be discussed due to the above actions

Indicative Course Schedule

Week	Week Begins	Topic	Learning Activity	Assessment Due
1	13 May	Introduction This is an indicative course schedule	Lab: Review Questions Team formation	
2	20 May	Unified Process (UP) and Models	Review Questions Case Study Team formation	
3	27 May	UP Information Gathering and Modelling	Review Questions Case Study	
4	3 June	Modelling Use Cases and Class Diagrams	Review Questions Case Study	
5	10 June	Feasibility	Review Questions	Quiz 1 (in lab)
6	17 June	State Diagrams	Review Questions	Assessment 1: Friday 11.59pm
7	26 June	Design Class Diagrams	Short Presentation (each team) Case Study	
8	1 July	Design Sequence Diagrams	Review Questions Case Study	
9	8 July	Human Computer Interaction	Review Questions	Quiz 2 (in lab)
10	15 July	Testing Controls and Revision	Review Questions Case Study	
11	22 July	Implementation and Deployment	Review Questions	Assignment 2: Friday 11.59pm
12	29 July	Revision	Short Presentation (each Team)	

Lecture Overview

- **Detailing System Requirements**
- **Functional Requirements**
 - Detailed Use Case
 - Domain Class Diagram
 - System Sequence Diagram (Separate File)

Contact Information

Lecturer: eugene Lutton

Email: eugene.lutton@newcastle.edu.au

Email Subject header:

- INFO6030 / Location / Reason
 - For example: INFO6030 / Online / Activity 4 question
 - For example: INFO6030 / Callaghan / Video week 3
-
- **Consultation:** *Please email for appointment*



Links

- **Detailed Use Case**

- What actor interacts with the system and functionality?

- **Domain Class Diagram**

- What data is required to enable this functionality of the system and the actor?

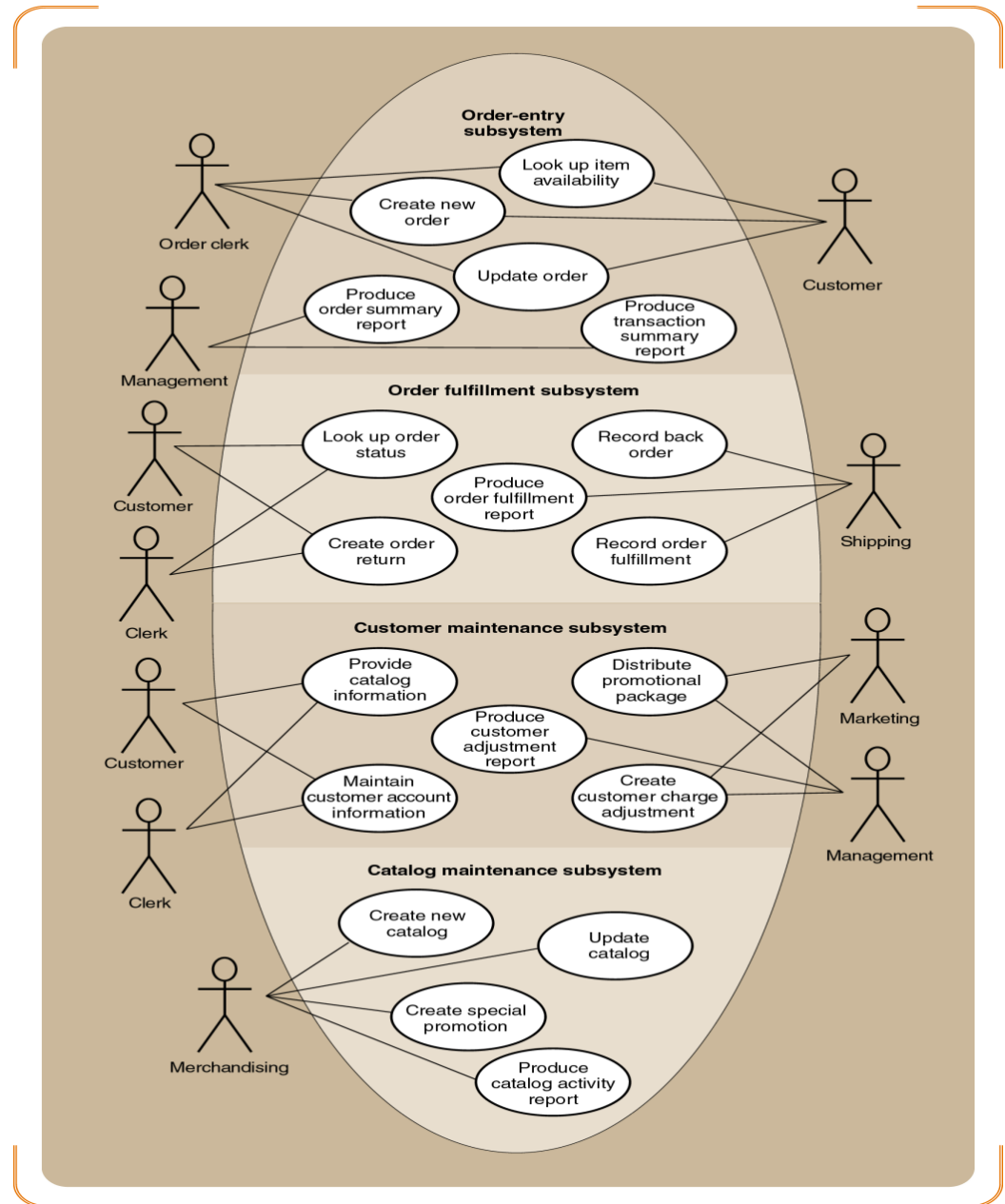
- **System Sequence Diagram**

- What is the data that is flowing between actor and system?

Use Case Diagrams

Use Case Modelling

Figure 6-4:
A Use Case Diagram of the
Customer Support System
(by Subsystem)



Steps in Use Case Modeling

- Establish the system boundary. Identify the actors that use the system.
- For each actor, consider what functions the system provides for each Actor.
- Represent each function as a Use case.
- Connect the Actor to all Use Cases initiated by the Actor.
- If the Use Case requires participation by other Actors, connect these actors also to the Use Case.

Developing a Use Case Diagram

Two ways to identify additional use cases

- Divide one large use case into two
- Define another use case based on a common subroutine <<*includes*>>

Conduct iterations to translate business events into use cases

- [1] Identify the actors and *roles* for each use case
- [2] Extract *system response(s)* to *business events*
- Look for use case completion: the data of the system *stabilises* after completion of the goal (of a use case)

Distinguish between *temporal* and *state* events

Types of Events

External Events

- Occur outside the system
- Usually caused by external agent

Temporal Events

- Occurs when system reaches a point (deadline) in time

State Events

- Asynchronous events responding to system triggers
 - Eg, a “picking list” or a “library loan” has received a message indicating all items are included all items and its **state** moves from *pending* to *completed*

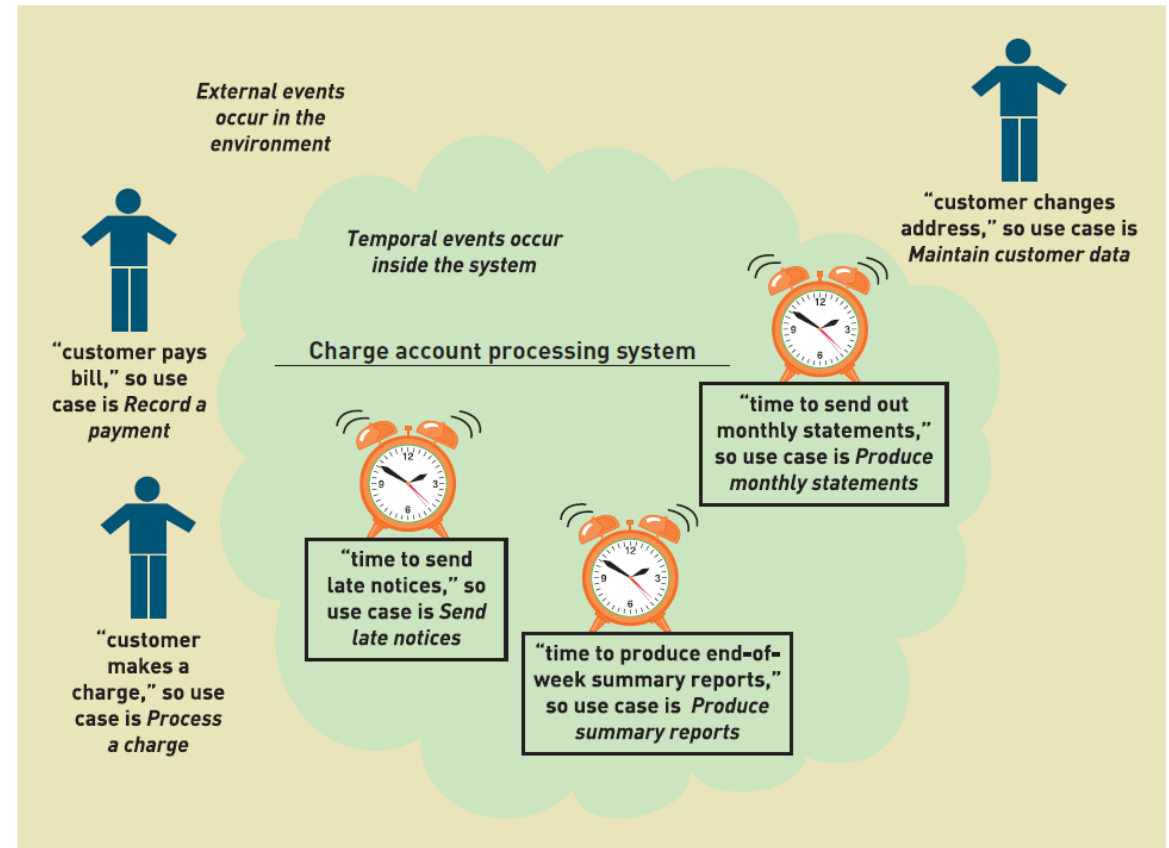


Figure 3-3
Events Affecting a Charge Account Processing System that Lead to Use Cases

Identifying Events

Distinguish events from prior conditions and system responses

- Is something an event or part of a sequence making up the event?
 - **Can the transaction complete without interruption?**
 - **Is the system waiting for the next transaction?**
 - Example: Is paying the credit card bill at the end of the month part of an event associated with purchasing an item?
 - No => They are separate events



Customer thinks about getting a new shirt



Customer drives to the mall



Customer tries on a shirt at Sears



Customer goes to Wal-Mart



Customer tries on a shirt at Wal-Mart



Customer buys a shirt
(the event that directly affects the system!)

Figure 5-5

Identifying Events

Trace the ***sequence of events*** initiated by external agents (actors)

- Investigate all transactions carried out by external agents that may affect the system
- Isolate events that actually **touch** the system

Figure 5-6

The Sequence of “Transactions” for One Specific Customer Resulting in Many Events



Identifying Events

Identify *technology dependent* events

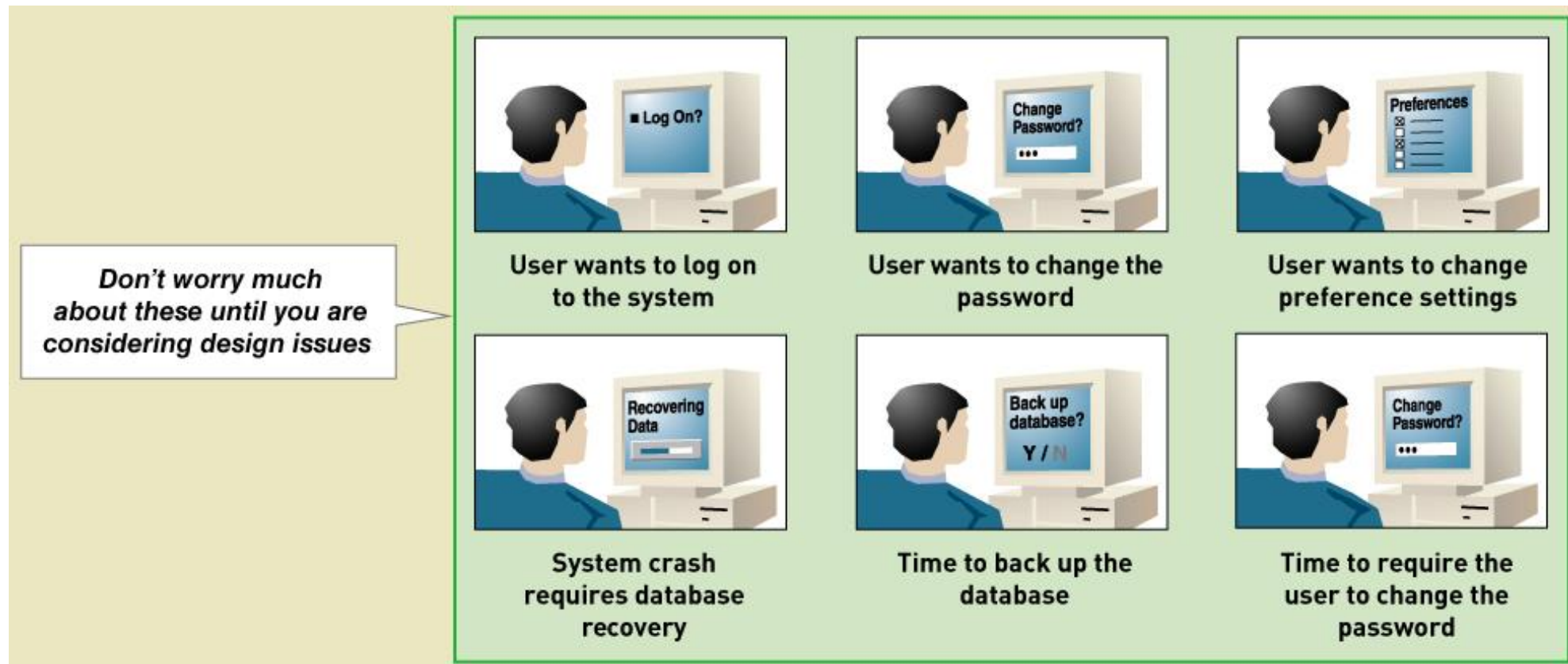
- These typically involve design choices or system controls necessary for implementation but not relevant until detailed design
 - Example: logging on to the system (a system *control*)

Defer specifying technology dependent events and use the *Perfect Technology Assumption*:

- Separates technology dependent events from functional requirements
 - Unlimited processing and storage capacity
 - Equipment does not malfunction
 - Users have ideal skill sets

Identifying Events

Figure 3-8
Events Deferred until Later Iterations



Finding Use Cases

Event	Trigger	Source	Use case	Response	Destination
Reservation clerk wants to make a booking	Booking requirement	Clerk	Make a booking	Flight seat	Clerk customer
Manager wants to make an entry for a new flight	New entry	Manager	Make an entry for a new flight	A new entry	Manager
Event	Trigger	Source	Use Case	Response	Destination
Customer wants to check item availability	Item inquiry	Customer	Look up item availability	Item availability details	Customer

Finding Use Cases

- What functions do the system perform?
- What functions do the “actors” require?
- What input/output does the system need?
- What **verbs** are used to describe the system?
 - The Reservation Clerk **makes a booking** using the system, based on the...
 - The Airport Manager can **make an entry for a new flight**. They can also **modify flight details**, provided...

« Includes » Relationships

«includes» relationship

- Use case calling services of common subroutine
- A Common subroutine becomes an additional use case
- Examples
 - Validate customer account and
 - Look Up Item Availability

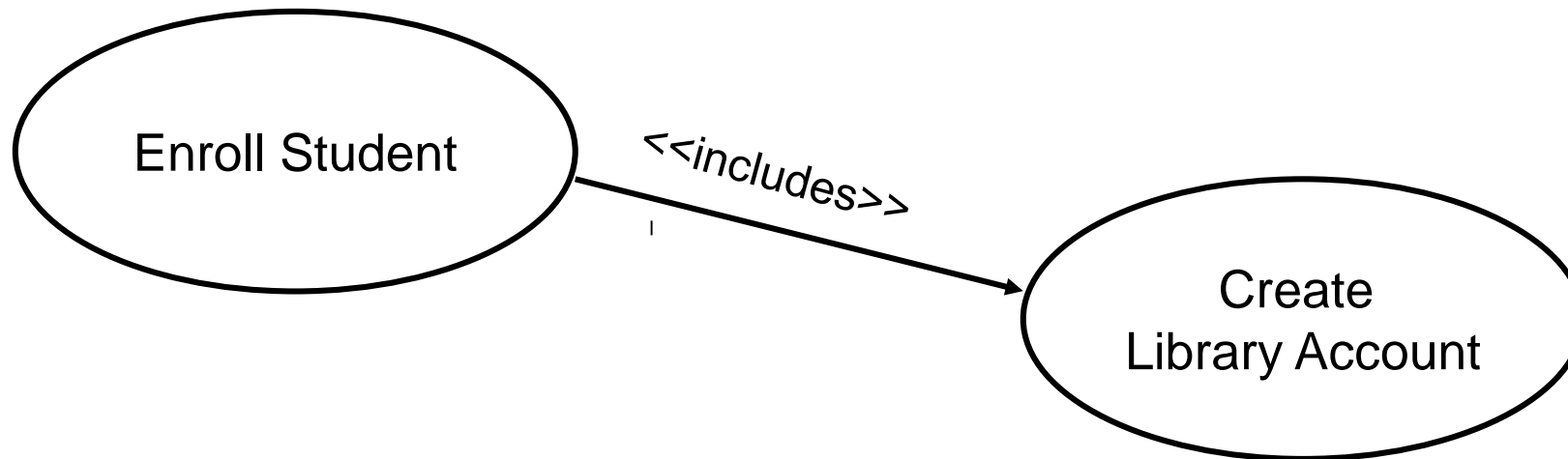
Notation

- Relationship denoted by connecting line with arrow
- Direction of the arrow indicates major/minor cases

Review / Extension

A <<*includes*>> B

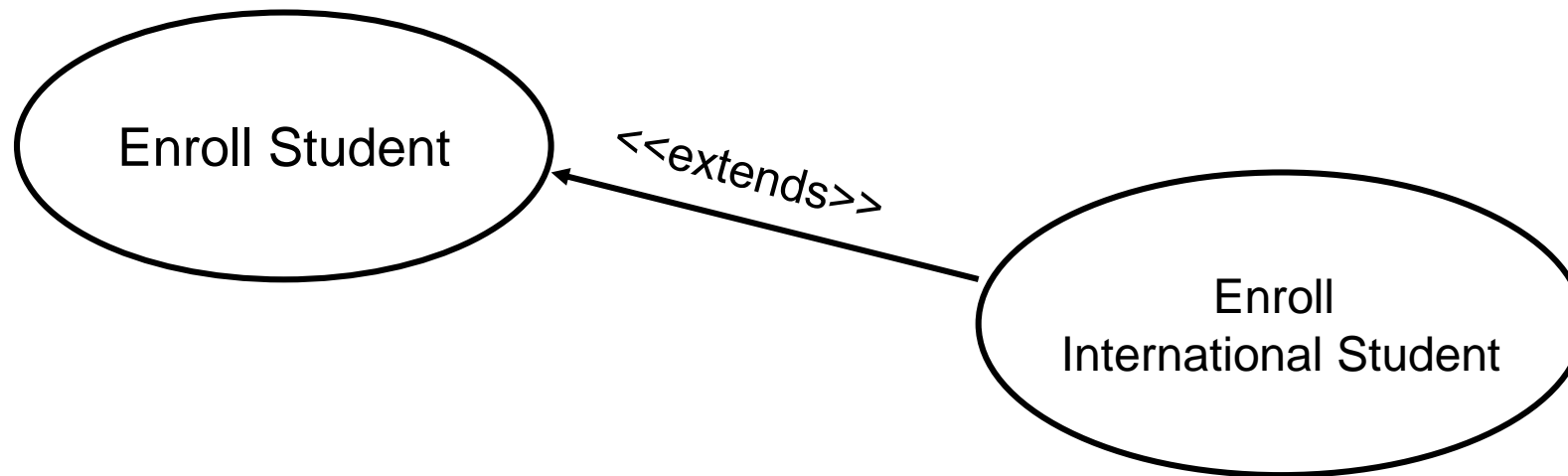
- A uses B to complete a task
 - Example: Enrolling a student requires the creation of a *library account*, as shown in the Use-Cases below



Review / Extension

A <<*extends*>> B

- B can extend the functionality of A
 - Example: Enrolling an international student may have specialised tasks (such as check passport) and therefore extends the general tasks involved in the *Enroll Student* use-case



Steps in Use Case Modeling

Factor optional or *exceptional behavior* in a Use Case, into a separate Use Case and connect this new Use Case with the main Use Case with the «*extend*» relationship.

Analyze all Use Cases and identify *common functions* across groups of Use Cases. Factor each such common function into a separate Use Case and connect this common Use Case with the other Use Cases that use it, with the «*include*» relationship.

Use Case Realisation

The use case diagram is an ***external*** view of the system

A use case is ***realised*** in a collaboration

A collaboration shows an ***internal implementation- dependent solution*** of a use case in terms of:

- classes/objects
- their relationships
- their interaction

Use Cases; Domain Model; Iteration Planning

Select use cases for further development

- Identify *risks* to determine *priority*
- Core architecture typically selected/implemented first

Needed for transition into the *elaboration* phase

- Converting use cases into software
- Iteratively integrate software components into more complex systems
- Begin testing of software

Use-Cases: Points to Ponder

Does every actor become a class?

- There is no such relationship. If some data/state of the actor needs to be stored by the system, then the actor finds expression as a class.

Is there any mapping between a Use-Case and a class or set of classes?

- There is no direct mapping that can be established. However, associated with **each** Use-Case is a Sequence Diagram (and therefore also a Collaboration Diagram). These model the classes internally (inside the automated boundary of the system), and the interactions between them.
- But
- The Use Case needs data that is shown in the Sequence Diagram and the diagrams needs instances of a class
- So there is no true mapping but it can be deduced

Requirements Elicitation

- Use Case: Guidelines for Formulation
- Name
 - Use a verb phrase to name the use case
 - The name should indicate what the user is trying to accomplish
 - Examples:
 - “Request Meeting”, “Schedule Meeting”, “Propose Alternate Date”
- Length
 - A use case description should not exceed 1-2 pages. If longer, use include relationships
 - A use case should describe a complete set of interactions.

Requirements Elicitation

- Use Case: Guidelines for Formulation

Flow of events:

- Use the active voice. Steps should start either with “The Actor” or “The System ...”
- The causal relationship between the steps should be clear
- All flow of events should be described (not only the main flow of event)
- The boundaries of the system should be clear. Components external to the system should be described as such
- Define important terms in the glossary.

Use Case Descriptions

Use cases have internal complexity

- Sequence of steps to execute business process
- Several variations may exist within a single use case
 - Such a valid variation is known as a *scenario*
 - Example: “Create new order” varies from *phone* to *Internet* order

Use Case Descriptions

Use case descriptions are written at (3) levels of detail

- Brief description
 - Summary statement conjoined to activity diagram
- Intermediate description
 - Expands brief description with an internal flow of activities.
- Fully Developed Description (**expected for this course and assignments**)
 - Expands intermediate description for richer view

Simple Use Case Diagram

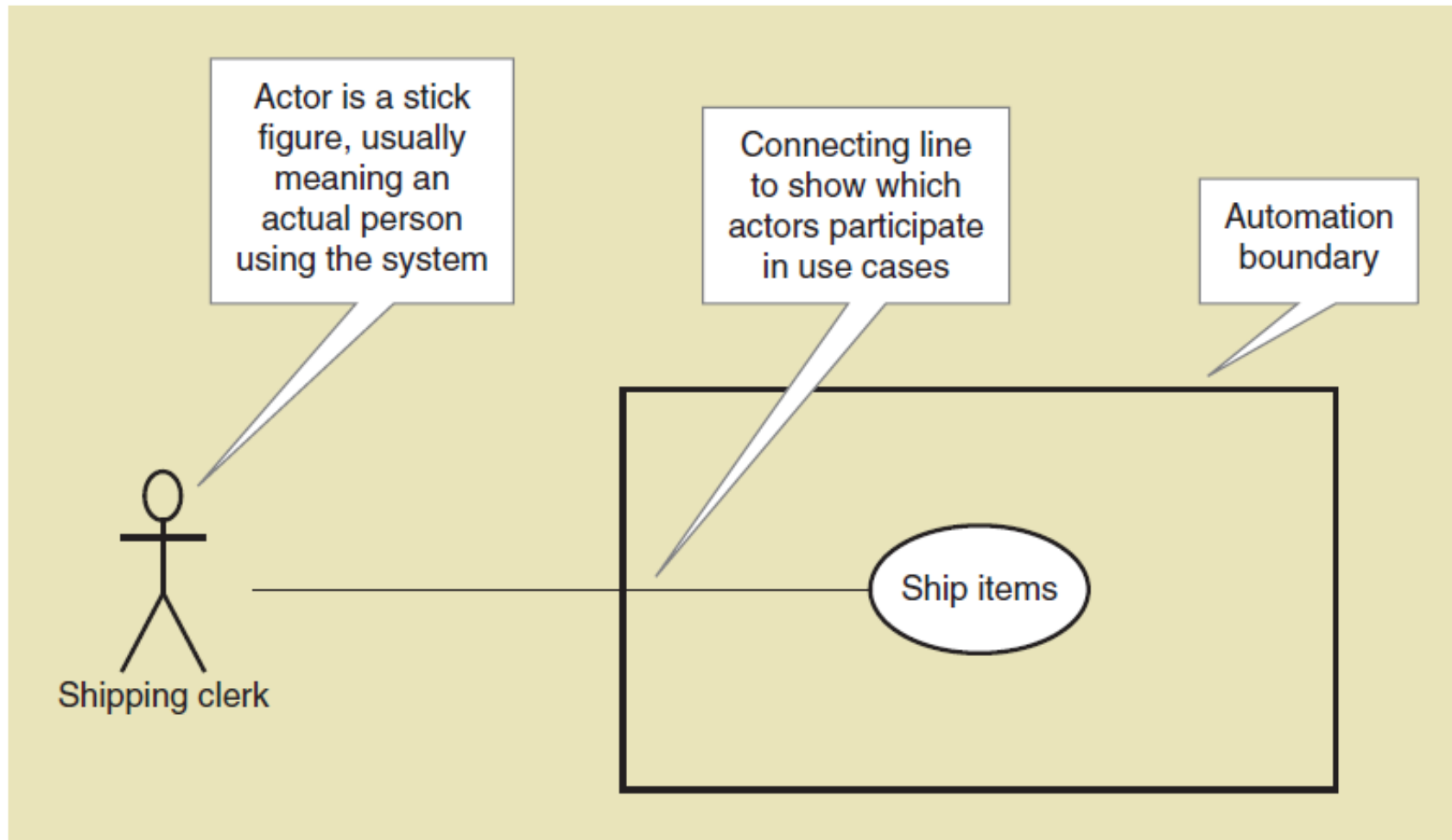
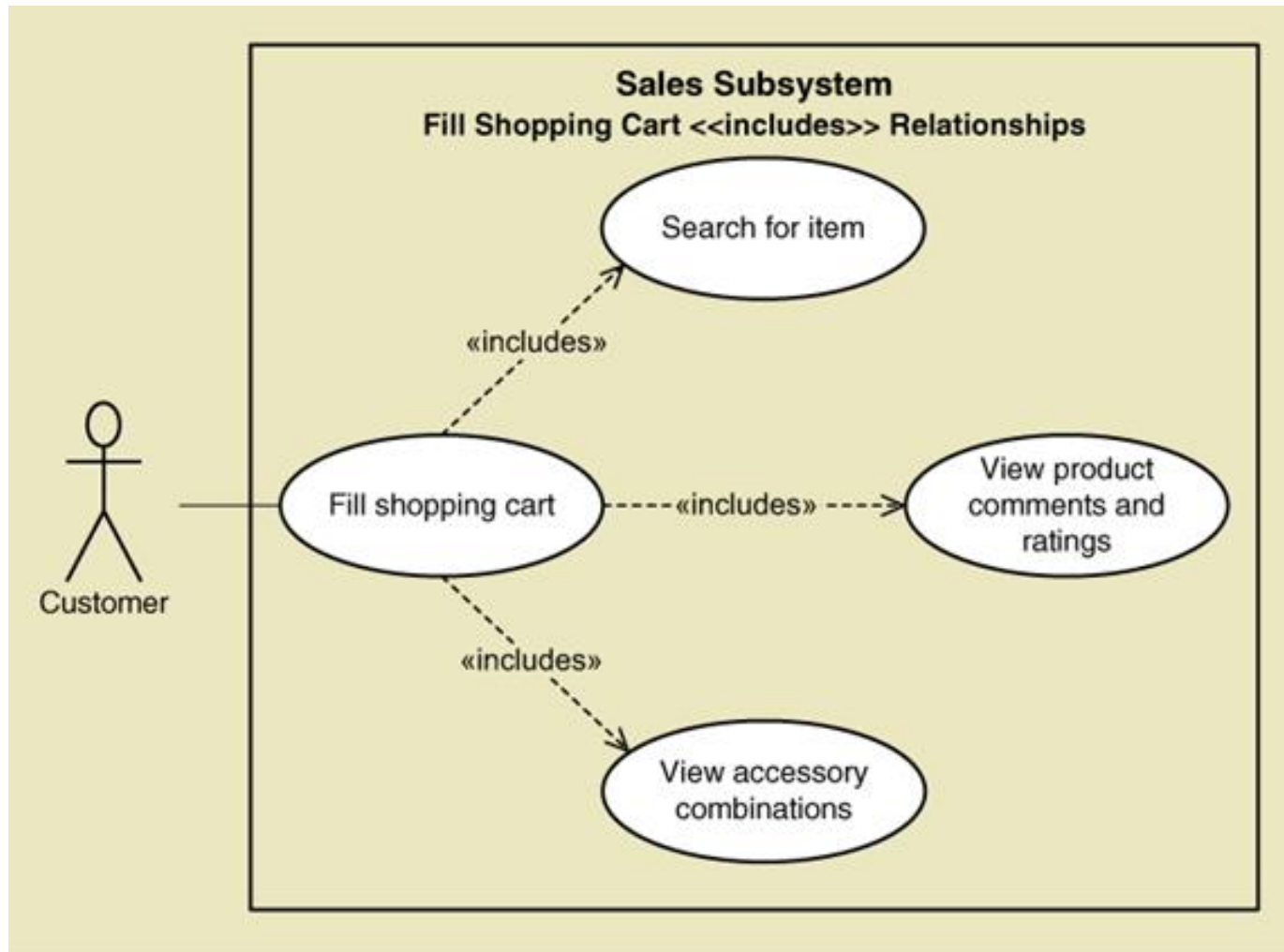


Figure 3-11
Simple Use Case

Sales Subsystem

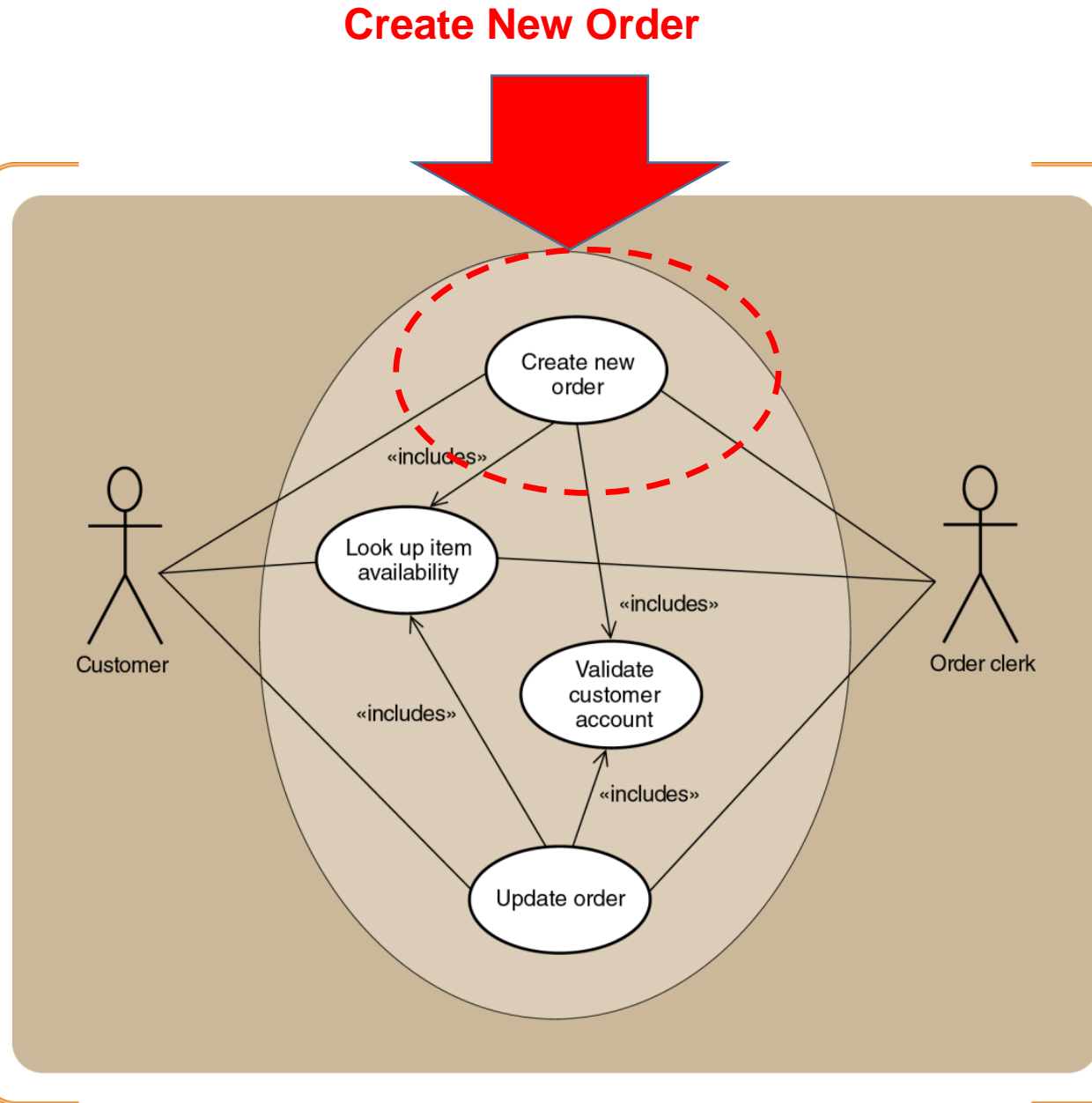
Figure 3-15
Sales Subsystem
with «*Includes*» Use
Cases



Sales Subsystem

Figure 6-6

Sales Subsystem with «Includes» Use Cases



Brief Use Case Description

Use case	Brief use case description
<i>Create customer account</i>	User/actor enters new customer account data, and the system assigns account number, creates a customer record, and creates an account record.
<i>Look up customer</i>	User/actor enters customer account number, and the system retrieves and displays customer and account data.
<i>Process account adjustment</i>	User/actor enters order number, and the system retrieves customer and order data; actor enters adjustment amount, and the system creates a transaction record for the adjustment.

Fig 5-1

Brief use case description is often a one sentence description showing the main steps in a use case

Use Case: Guidelines for Formulation

- Use Case: Guidelines for Formulation

Flow of events:

- Use the active voice. Steps should start either with “The Actor” or “The System ...”
- The causal relationship between the steps should be clear
- All flow of events should be described (not only the main flow of event)
- The boundaries of the system should be clear. Components external to the system should be described as such
- Define important terms in the glossary.
- Must show any extends or includes eg Invoke Includes/extends Use case (name)
- Exceptions must state where the system goes: See Vending machine example

Use Case: Guidelines for Formulation

- Example: Badly written Use Case

“The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through.”



(Greim, 2015)

Question: Can you identify the issues? What is wrong with this use case?

Use Case: Guidelines for Formulation

- Example: Badly written Use Case

“The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through.”

Issues:

- It is not clear which action triggers the ticket being issued.
- Because of the passive form, it is not clear who opens the gate
 - The driver?
 - The computer?
 - A gate keeper?

It is not a complete transaction

A complete transaction would also describe the driver paying for the parking and driving out of the parking lot.

Figure 5-2:
Fully Developed Description
of *Create Customer Account*
Use Case

Use case name:	<i>Create customer account.</i>	
Scenario:	Create online customer account.	
Triggering event:	New customer wants to set up account online.	
Brief description:	Online customer creates customer account by entering basic information and then following up with one or more addresses and a credit or debit card.	
Actors:	Customer.	
Related use cases:	Might be invoked by the <i>Check out shopping cart</i> use case.	
Stakeholders:	Accounting, Marketing, Sales.	
Preconditions:	Customer Account subsystem must be available. Credit/debit authorization services must be available.	
Postconditions:	Customer must be created and saved. One or more Addresses must be created and saved. Credit/debit card information must be validated. Account must be created and saved. Address and Account must be associated with Customer.	
Flow of activities:	Actor	System
	1. Customer indicates desire to create customer account and enters basic customer information. 2. Customer enters one or more addresses. 3. Customer enters credit/debit card information.	1.1 System creates a new customer. 1.2 System prompts for customer addresses. 2.1 System creates addresses. 2.2 System prompts for credit/debit card. 3.1 System creates account. 3.2 System verifies authorization for credit/debit card. 3.3 System associates customer, address, and account. 3.4 System returns valid customer account details.
Exception conditions:	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.	

11 Parts

**Has a few
issues
What are
they?**

Use Case Description Assignment Template

**Required for
Assignment**

Use Case Name:	Scan Item	
Scenario:		
Triggering Event:		
Brief Description:		
Actors:		
Stakeholders:		
Preconditions:		
Postconditions:		
Flow of Activities:	Actor	System
	1	1.1
	2.	2.1
	3.	3.1
	4.	4.1
Alternative Flow	##?	
Exception Conditions:	## ?	
	## ?	

Use Case Diagrams

- Use Case Descriptions: Fully Developed
- Flow of Activities for Scenario of “Buy A Beverage”

Use Case Name	Buy a Beverage
Brief Description	The vending machine delivers a beverage after a customer selects and pays for it
Trigger	customer inserts money to buy a drink
Actors	customer
Related Use cases	Include, extend, or generalization
Stakeholders	Owner of machine (we need more info) (not an actor)
Pre- conditions	the waiting state in which it displays the message “Enter coins”
Post-conditions	the item is dispensed. Returns to waiting state

Use Case Diagrams

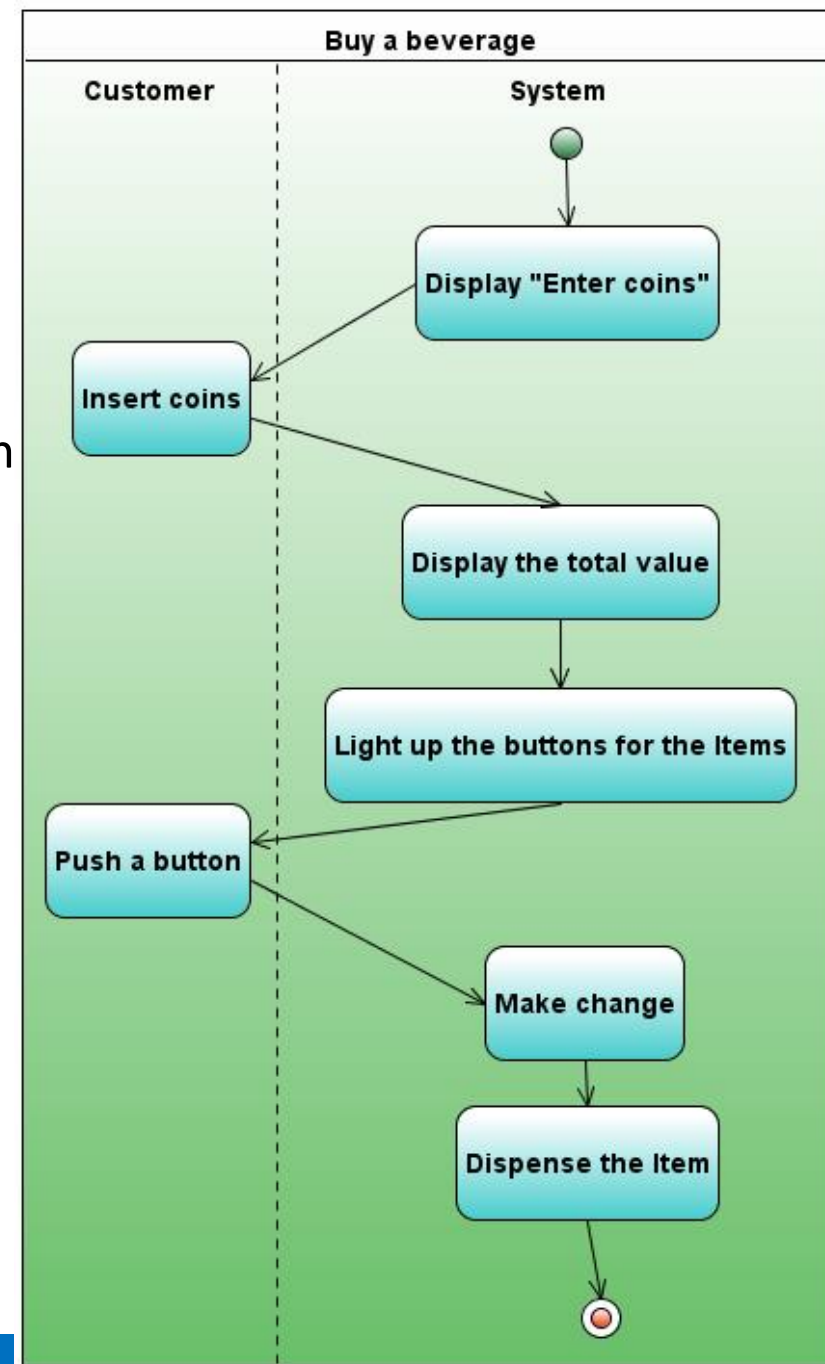
- Use Case Descriptions: Fully Developed
- Flow of Activities for Scenario of “Buy A Beverage”

Use Case Name	Buy a Beverage	
Flow of Events	Actors	System
	<ol style="list-style-type: none">1. Customer inserts coins into the machine.2. Customer pushes a button.	<ol style="list-style-type: none">1.1 Displays the total value of money entered1.2 Machine lights up the buttons for the items that can be purchased for the money inserted.2.1 Dispenses the corresponding item and make changes. Returns to waiting state.
Alternative Flow	<ol style="list-style-type: none">2.2 If the customer presses a button for an item that costs more than the money inserted,<ol style="list-style-type: none">a. the message “You must insert \$nn.nn more for that item” is displayed, where nn.nn is the amount of additional money needed, andb. the machine continues to accept coins or a selection2.3 If the customer has inserted enough money to buy the item but the machine cannot make the correct change<ol style="list-style-type: none">a. the message “Cannot make correct change” is displayed, andb. the machine continues to accept coins or a selection.	
Exception Conditions	<ol style="list-style-type: none">2.4. If the customer presses the cancel button before an item has been selected,<ol style="list-style-type: none">a. the customer’s money is returned andb. the machine resets to the waiting state.	

Activity Diagrams

Activity Diagrams

- Use Case Description
 1. Use case name: Buy a beverage
 2. Pre condition: the waiting state in which it displays the message “Enter coins”
 3. Customer: Customer inserts coins into the machine.
 4. System: Displays the total value of money entered and
 - Machine lights up the buttons for the items that can be purchased for the money inserted
 5. Customer: Customer pushes a button
 6. System: Dispenses the corresponding item and gives change.



Activity Diagram

- Alternative Flow: Use Case Description

- 6.1 If the customer presses a button for an item that costs more, than the money inserted:
 - a. the message *"You must insert \$nn.nn more for that item"* is displayed, where *nn.nn* is the amount of additional money needed, and
 - b. the machine continues to accept coins or a selection
- 6.2 If the customer has inserted enough money to buy the item but the machine cannot make the correct change
 - a. the message *"Cannot make correct change"* is displayed, and
 - b. the machine continues to accept coins or a selection.

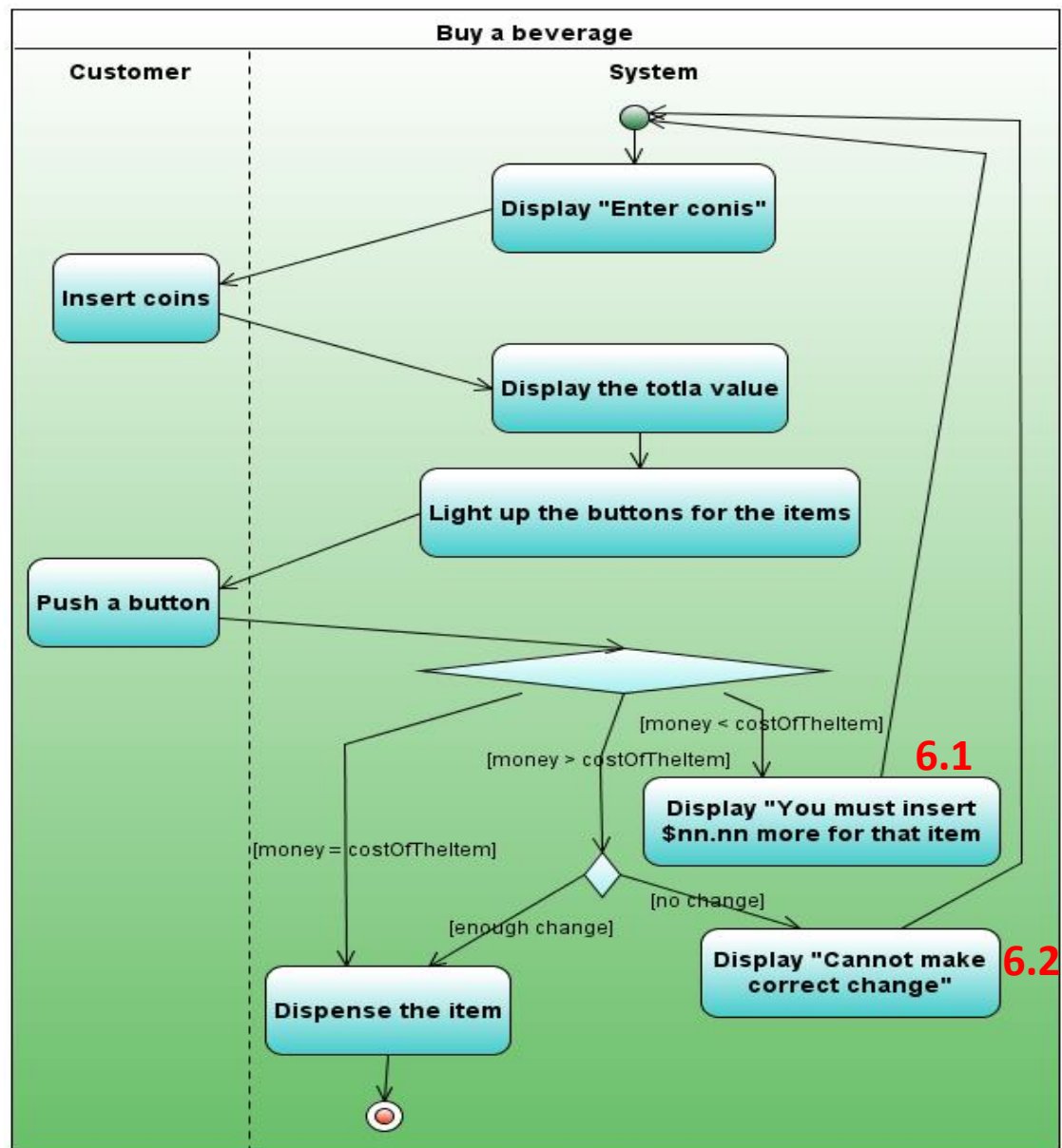
Exception Conditions:

- 6.3 If the customer presses the cancel button before an item has been selected,
 - a. the customer's money is returned and
 - b. the machine resets to the waiting state.

↖ DIY: add this to the diagram

Activity Diagrams

- Alternative Flow: Use Case Description



Activity Diagrams

- Activity diagrams are used for business process modelling, for modelling the logic captured by a single use case.
- It is essentially a flowchart:
 - Showing flow of control from activity to activity
 - Records the dependencies between activities, such as which things can happen in parallel and what must be finished before something else can start.

Activity Diagrams

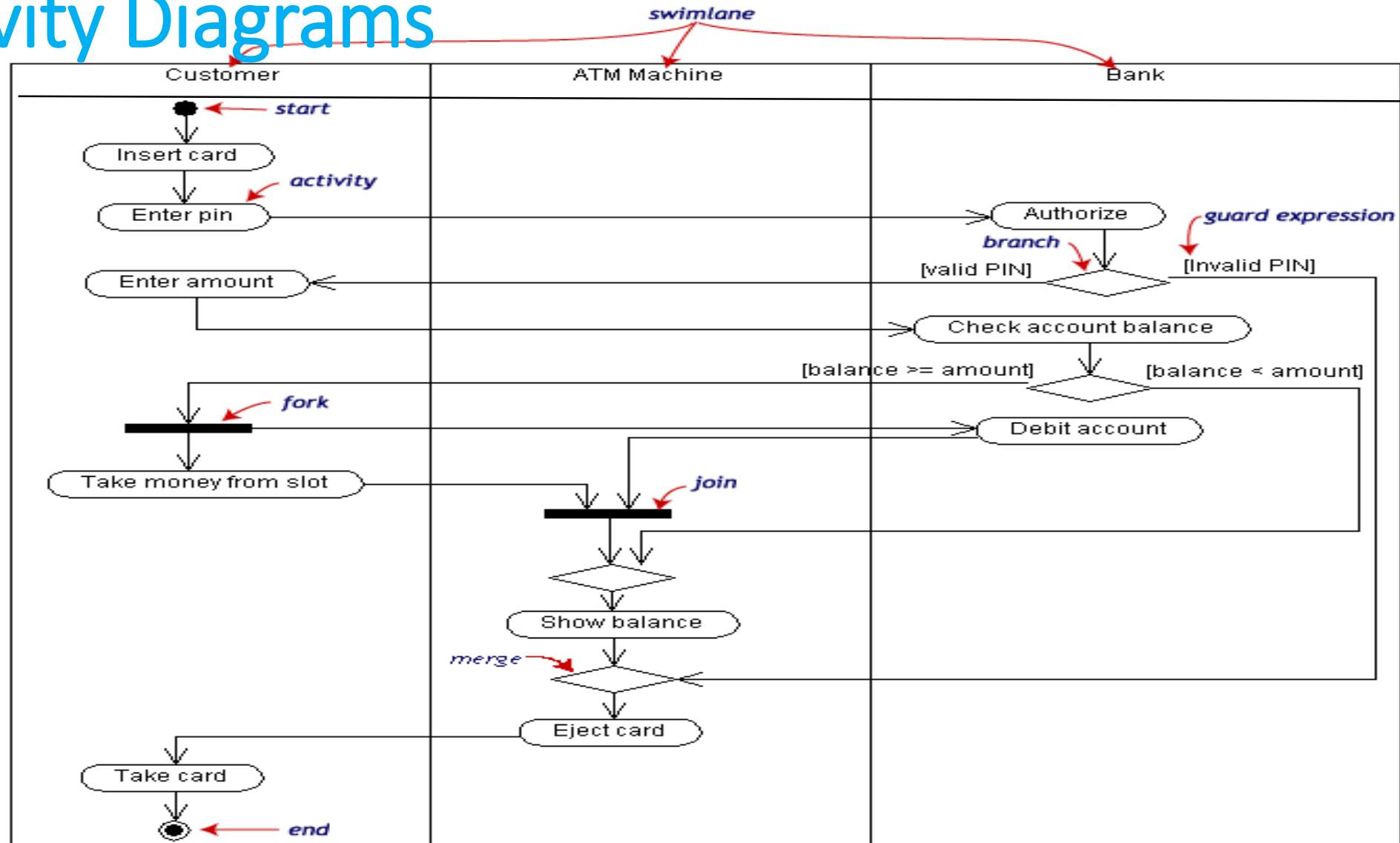


Figure 5-4
Activity Diagram of
the *Create customer
account Use Case*

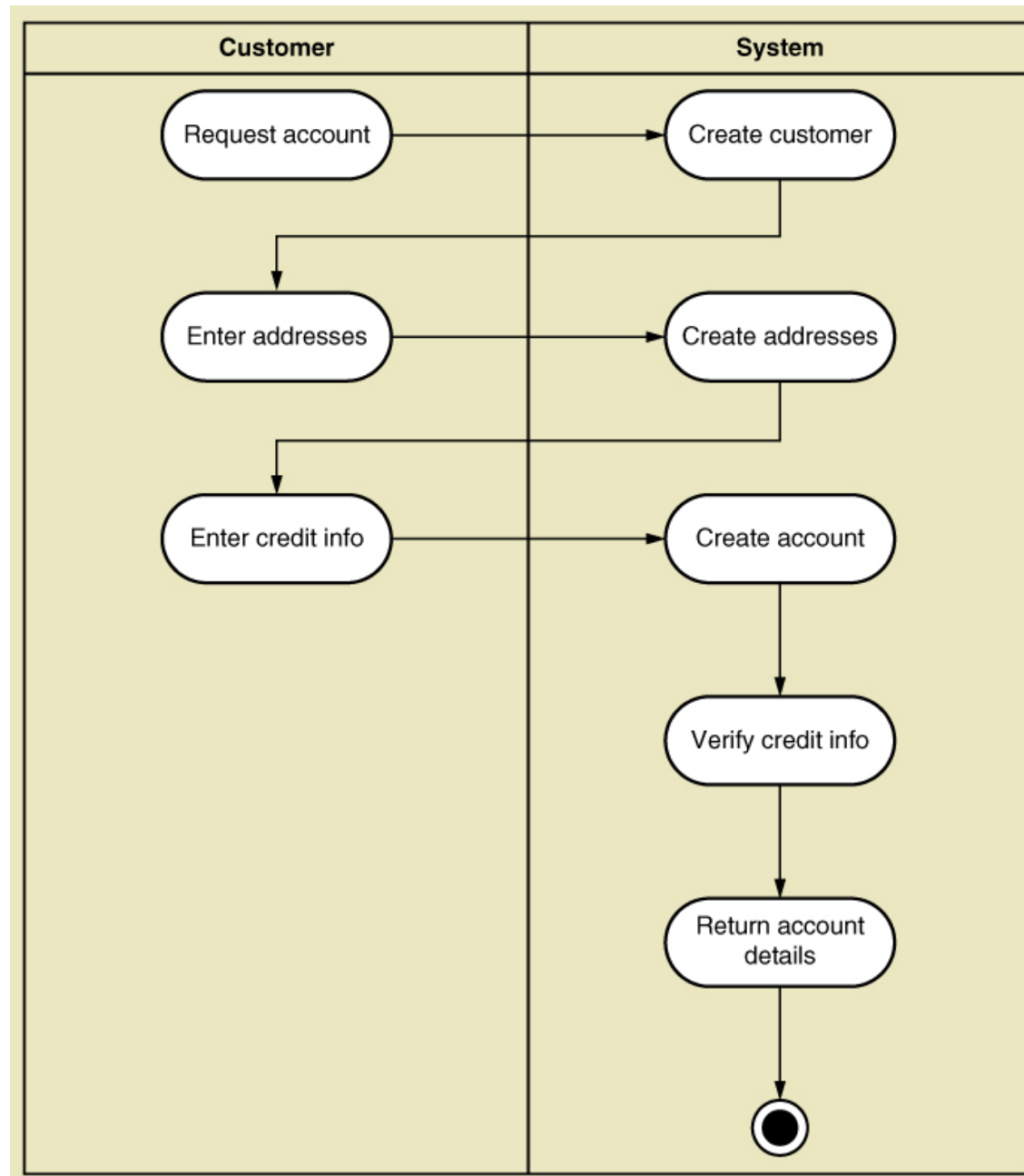
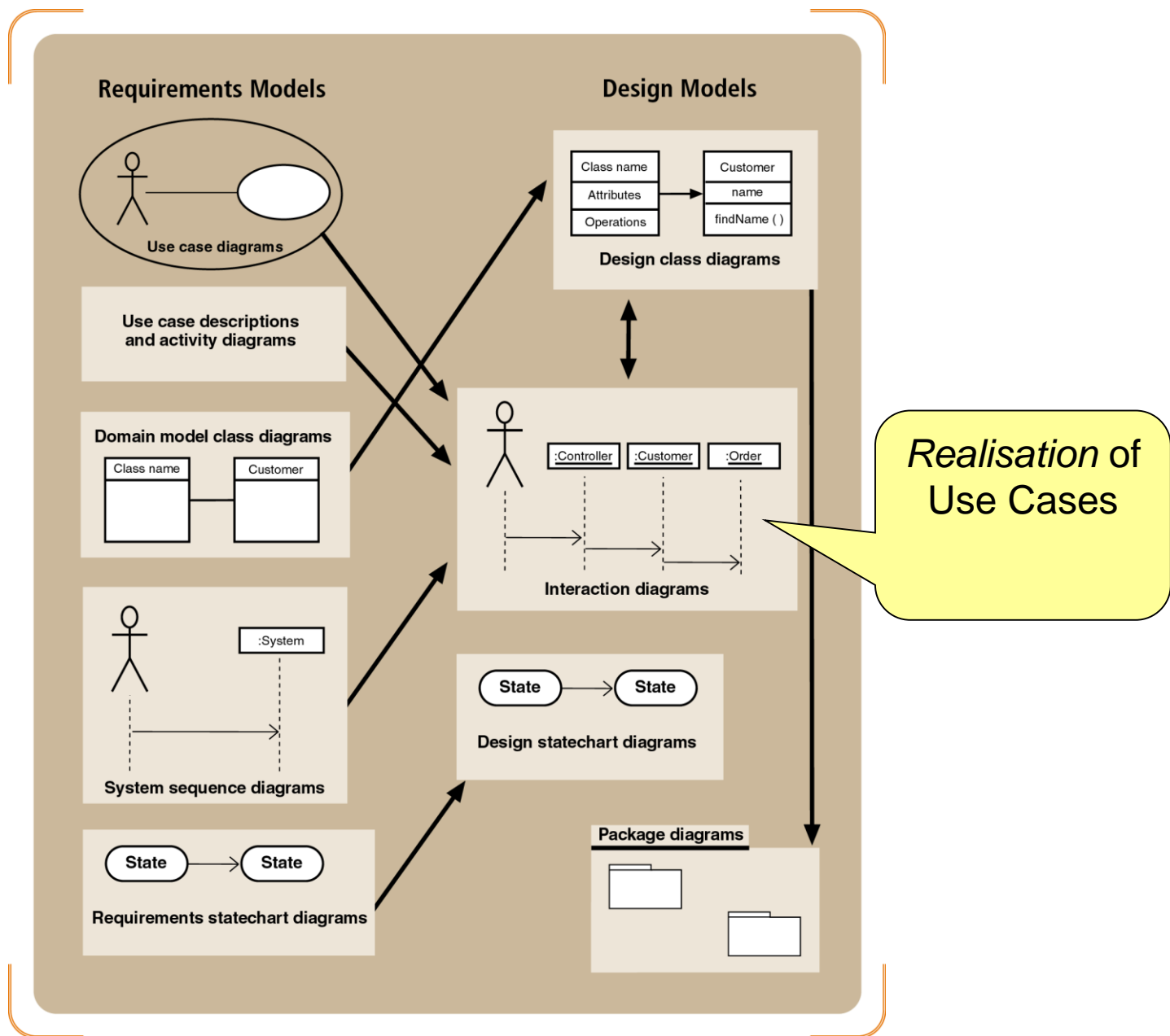


Figure 8-4:
Design models
with their
respective
input models



Problem Domain Classes

Problem Domain Classes

Problem Domain Classes are a set of work-related “things” in the system component

- These *Things* have data representation within the system
- Examples: products, orders, invoices, customers

OO approach to things in the problem domain

- Objects that ***interact*** within the system

A key initial step in defining requirements is to identify and understand *things* in the problem domain

Problem Domain Classes

At this stage we are only interested in the domain classes and the attributes or data that is required.

So we are not interested in Database Modelling

- So no PKs or FKs

Actual Class behaviour

- Methods

Identify the Types of Things

Separate the *tangible* from the *intangible*

- Example: an *item* or an *order* are both tangible and would be representative of a class, as opposed to *quality* which is intangible but might be an attribute of a class

Include information from all types of users

Ask important questions about the nature of each event

- E.g., What actions upon things should be acknowledged and recorded by the system?
- What needs to be stored? Why???

Identify the Types of Things

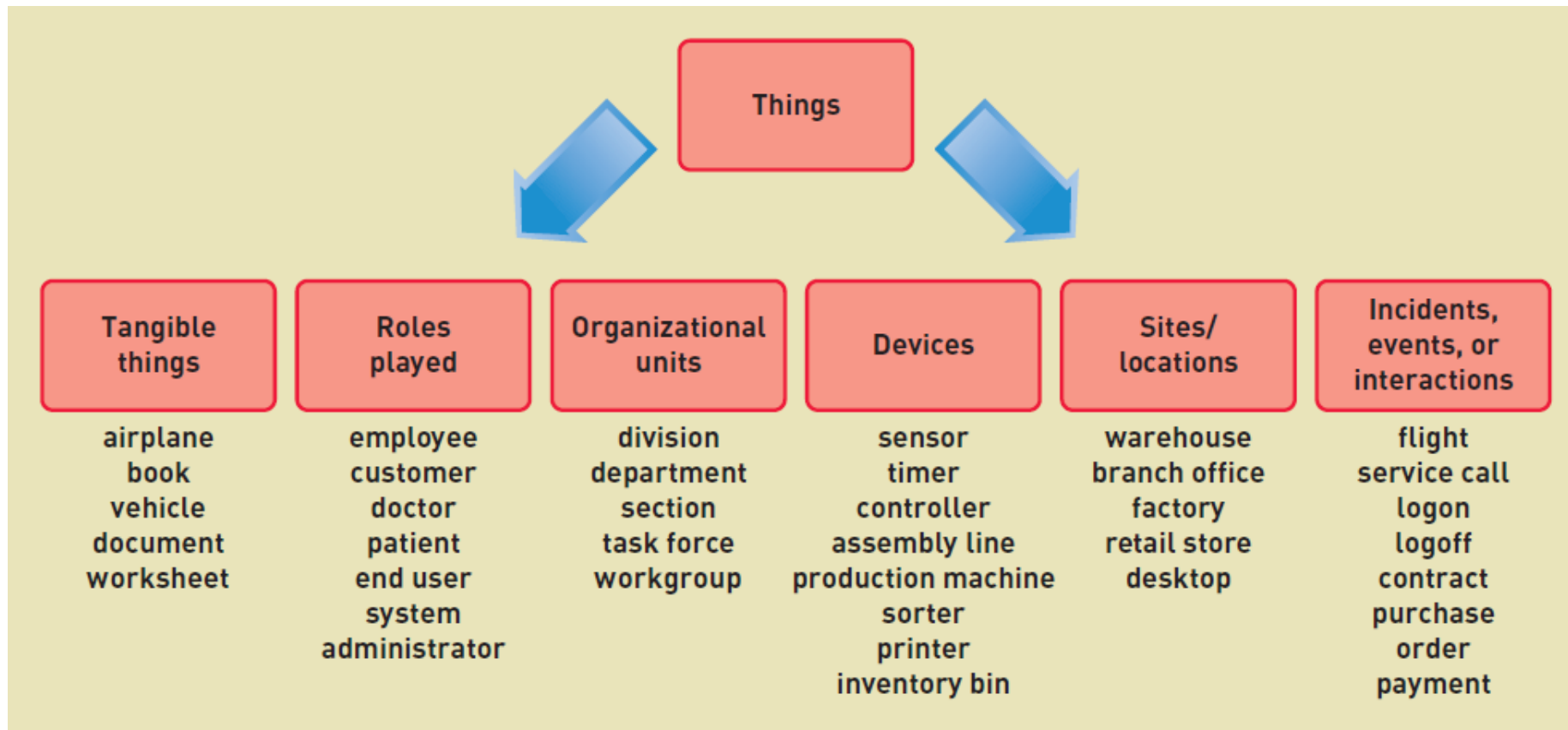


Figure 4-1
Types of Things

Procedure for Developing A List of Things




- List *nouns* that users mention when discussing the system
- Use the Event table as a source of potential things
 - Use cases, external agents, triggers, response
- Select *nouns* with questions concerning their relevance to the system
- Record assumptions about things
- Look for Hierarchies (Sub-Classes, attributes, compositions or aggregations in the list of things)
- Identify attributes for each thing to define it
 - For each thing, identify a set of unique attributes (a key) that can identify one instance of the thing from another instance
 - This key is related to the domain eg boat license

Class Identification Procedure

- When you are interacting with the client, take note of the following or when reading a scenario
- Look for **Nouns**
 - Not all will be a class
- Look for **Adjectives**
 - These may be the attributes/data for each class
- Look for **Verbs**
 - This may be the associations between classes


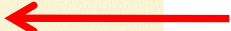
Class Identification Procedure

Figure 5-13a:
Partial List of “Things”
Based on Nouns for
RMO

IDENTIFIED NOUN	NOTES ON INCLUDING NOUN AS A THING TO STORE
Accounting	We know who they are. No need to store it.
Back order	A special type of order? Or a value of order status? Research. 
Back-order information	An output that can be produced from other information.
Bank	Only one of them. No need to store. 
Catalog	Yes, need to recall them, for different seasons and years. Include. 
Catalog activity reports	An output that can be produced from other information. Not stored.
Catalog details	Same as catalog? Or the same as product items in the catalog? Research.
Change request	An input resulting in remembering changes to an order.
Charge adjustment	An input resulting in a transaction.

Class Identification Procedure

Figure 5-13b:
Partial List of
“Things” Based on
Nouns for RMO

IDENTIFIED NOUN	NOTES ON INCLUDING NOUN AS A THING TO STORE
Color	One piece of information about a product item.
Confirmation	An output produced from other information. Not stored. 
Credit card information	Part of an order? Or part of customer information? Research. 
Customer	Yes, a key thing with lots of details required. Include.
Customer account	Possibly required if an RMO payment plan is included. Research.
Fulfillment reports	An output produced from information about shipments. Not stored.
Inventory quantity	One piece of information about a product item. Research.
Product item	Yes, what RMO includes in a catalog and sells. Include.
Management	We know who they are. No need to store.
Marketing	We know who they are. No need to store.

Class Identification Procedure

Figure 5-13c:
Partial List of
“Things” Based on
Nouns for RMO

IDENTIFIED NOUN	NOTES ON INCLUDING NOUN AS A THING TO STORE
Merchandising	We know who they are. No need to store. ←
Order	Yes, a key system responsibility. Include.
Payment method	Part of an order. Research.
Price	Part of a product item. Research.
Promotional materials	An output? Or documents stored outside the scope? Research. ←
Prospective customer	Possibly same as customer. Research. ←
Return	Yes, the opposite of an order. Include.
Return confirmation	An output produced from information about a return. Not stored.
RMO	There is only one of these! No need to store.
Season	Part of a catalog? Or is there more to it? Research.
Shipment	Yes, a key thing to track. Include. ← (continued)

Class Identification Procedure

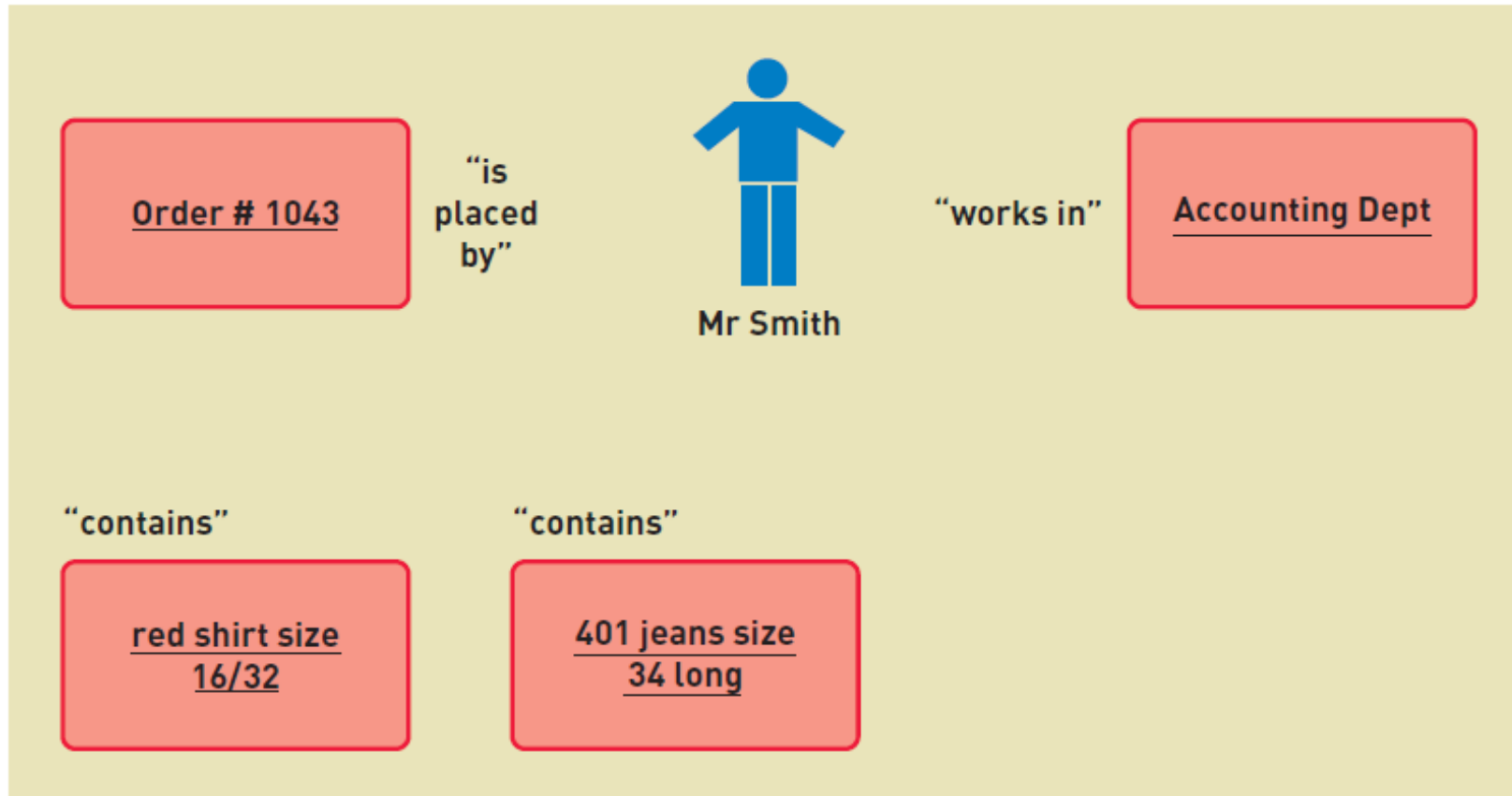


Figure 4-4
Associations Naturally Occur between Things

Associations Between Things

Please Watch This Video:

Title: UML Class Diagrams - Association and Multiplicity

Link:

<https://www.youtube.com/watch?v=BhEoV57nj0Q>

Duration: 9:22 minutes



(MargretPosch, 2015)

Associations Between Things

Analysts document entity **associations** (relationships)

- Example: “Is placed by” and “works in”

Associations

- apply in two directions
 - Customer *places* an order
 - An order *is placed by* a customer
- Have **Multiplicity**: the number of associations between instances (of the Domain Classes) that are taking part in the association between the Domain Classes
 - **One to one** or **one to many** (*many to many associations will be decomposed*)
- Can be **Mandatory** or **Optional**
- Can be **Unary** (recursive), **binary** (associations between two different classes of things), **n-ary** (associations between *n* different classes of things)

Associations Between Things

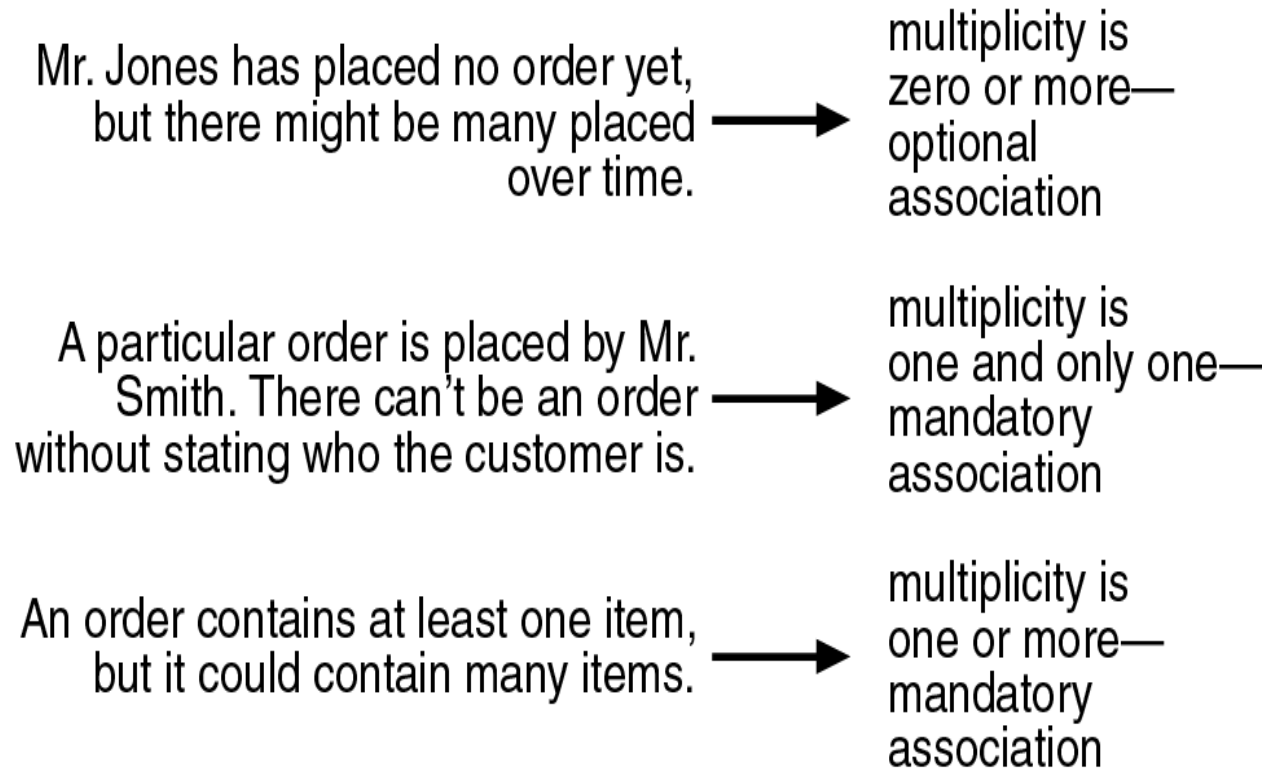


Figure 5-15
Multiplicity of Relationships

Attributes of Things

Specific details of things are called *attributes*

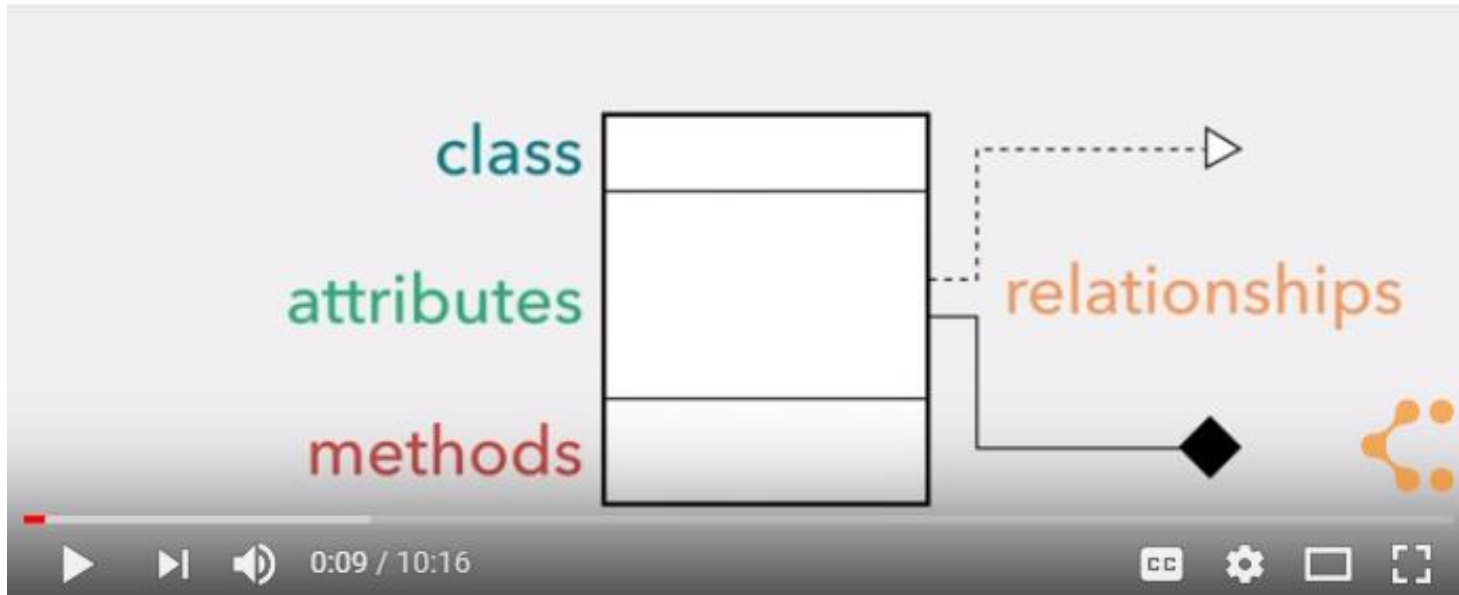
Identifier (*key*): is an attribute capable of uniquely identifying a specific instance of a thing (from another instance of a thing)

- Examples: Medicare number, vehicle VIN number, or product ID number
- A key is not included in a domain class diagram (ass1)

Compound attribute is a set of related attributes

- Example: multiple names for the same customer

Class Diagrams



Please watch this video:

Title: UML Class Diagram Tutorial (not a domain class diagram)

Link: <https://www.youtube.com/watch?v=UI6lqHOVHic>

Duration: 10:15 minutes

(Lucidchart, 2017)

Class Diagrams

How to create a class diagram from scratch:

- Identify a first set of candidate **classes**
- Add **associations** and **attributes**
- Find **generalizations**
- List the main **responsibilities** of each class
- Decide on specific **operations**
- **Iterate** over the entire process until the model is satisfactory
 - Add or delete classes, associations, attributes, generalizations, responsibilities or operations
 - Identify interfaces

Don't be too disorganized. Don't be too rigid either.

Please watch this video:

Title: UML Class Diagram Tutorial (not a domain class diagram)

Link: <https://www.youtube.com/watch?v=UI6lqHOVHic>

Duration: 10:15 minutes

Class Diagrams

Discovering classes:

- Look at a source material such as a description of requirements
- Extract the *nouns* and *noun phrases*
- Eliminate nouns that:
 - are redundant
 - represent instances
 - are vague or highly general
 - not needed in the application
- Pay attention to classes in a domain model that represent *types of users* or other actors

Class Diagrams

Identifying associations and attributes:

- Start with classes you think are most central and important (Week 3 Lab University scenario)
- Decide on the clear and obvious data it must contain and its relationships to other classes.
- Work outwards towards the classes that are less important.
- Avoid adding many associations and attributes to a class (however, add important ones)
 - A system is simpler if it manipulates less information

Class Diagrams

Tips about identifying and specifying valid associations:

- An association should exist if a class
 - *possesses*
 - *controls*
 - *is connected to*
 - *is related to*
 - *is a part of (aggregation or composition)*
 - *has as parts (aggregation or composition)*
 - *is a member of, or (generalisation)*
 - *has as members (generalisation)*some other class in your model
- Specify the multiplicity at both ends
- Label it clearly.

Class Diagrams

Identifying attributes:

- Look for information that must be *maintained* about each class
- Several nouns rejected as classes, may now become attributes
- An attribute should generally contain a simple value
 - E.g. string, number

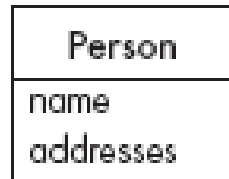
Class Diagrams

Tips about identifying and specifying valid attributes:

- It is not good to have many duplicate attributes
- If a subset of a class's attributes form a coherent group, then create a distinct class containing these attributes

Class Diagrams

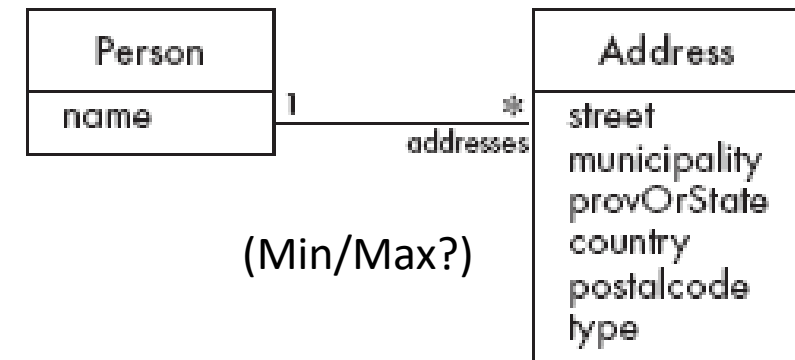
Note: What do you notice, that your team would change?



Bad, due to
a plural attribute



Bad, due to too many
attributes, and the
inability to add more
addresses



Good solution. The type indicates whether it
is a home address, business address etc.

Class Diagrams

Difficulties and Risks when creating class diagrams:

Modelling is particularly difficult skill

- *Even excellent programmers have difficulty thinking at the appropriate level of abstraction*
- *Education traditionally focus more on design and programming than modelling*
- *It is a perception process*

Resolution:

- *Ensure that group members have adequate training*
- *Have experienced modeller as part of the team*
- *Review all models thoroughly*

Classes and Objects

Domain model class diagram as UML class

- OOA applies domain model class diagram to things

Problem domain objects have *attributes*

Software objects encapsulate attributes and behaviors

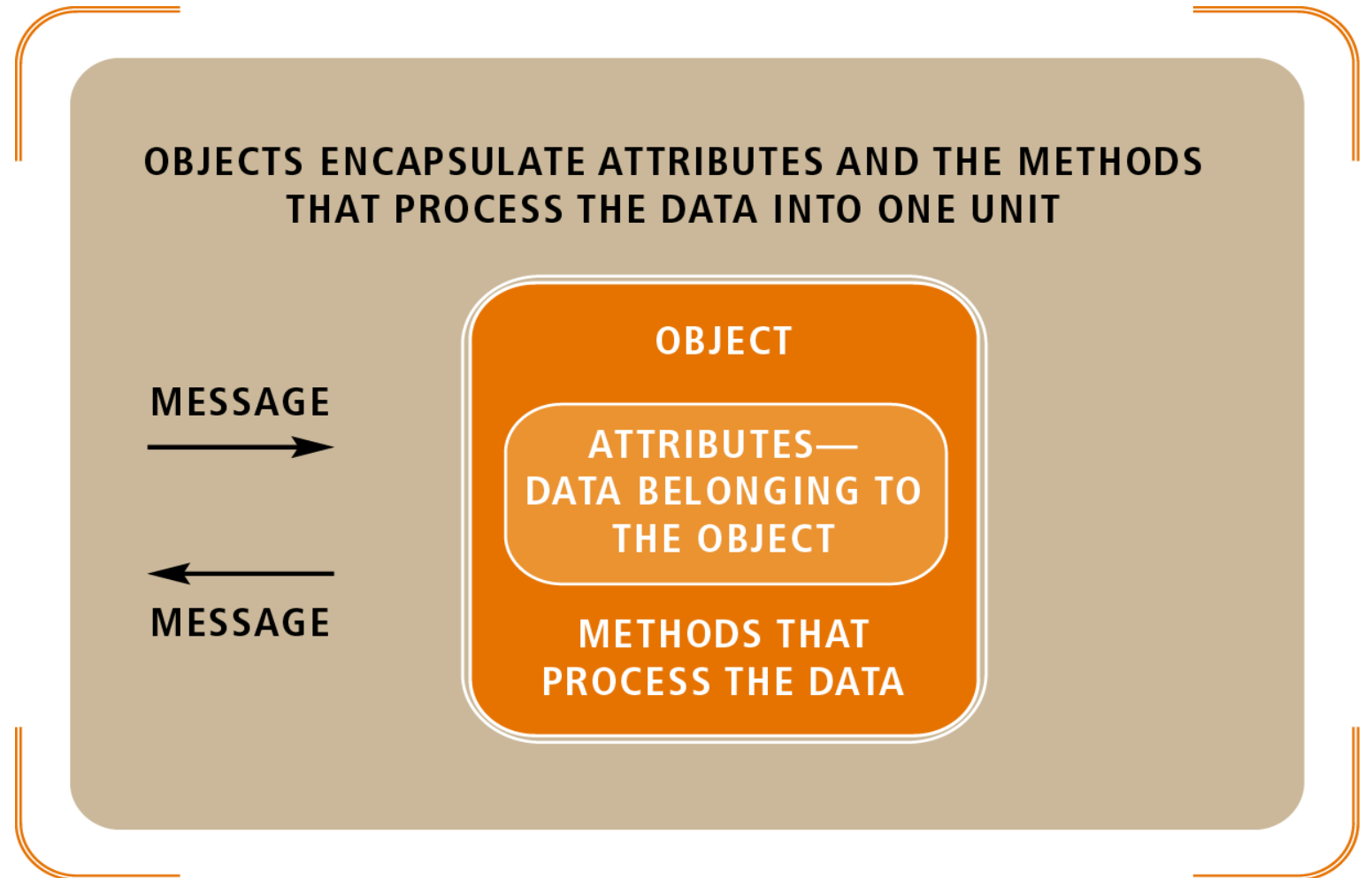
- Behavior: actions that the object processes itself

Software objects communicate with *messages*

The OO approach is to treat an Information system as a set of *interacting objects*

Classes and Objects

Figure 5-17
Objects Encapsulate
Attributes and Methods



Domain Model: Class Diagram Notation

Class diagram

- General class symbol: rectangle with three sections
- Sections convey name, attributes, and behaviors
- Methods (behaviours) not shown in a domain model class diagram
- Lines connecting rectangles show associations
- Multiplicity reflected above connecting lines

Domain class objects reflect *business concerns, policies, and constraints*

Abstract Classes have no **instances** (e.g., Bank Account) and is denoted in class diagrams by an **Italicised** Name

Domain Model: Class Diagram Notation

Note: What do you notice, that your team would change?

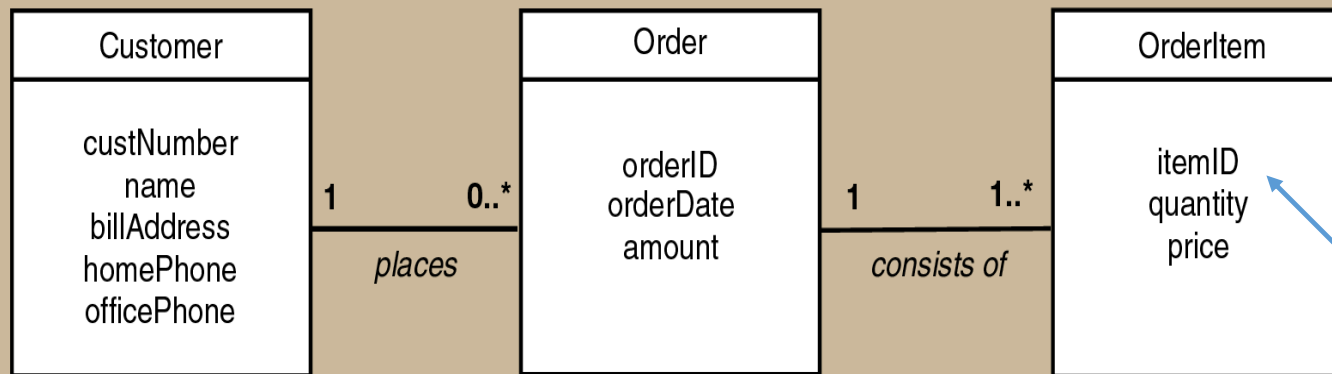


Figure 5-21
An Expanded Domain
Model Class Diagram
Showing Attributes

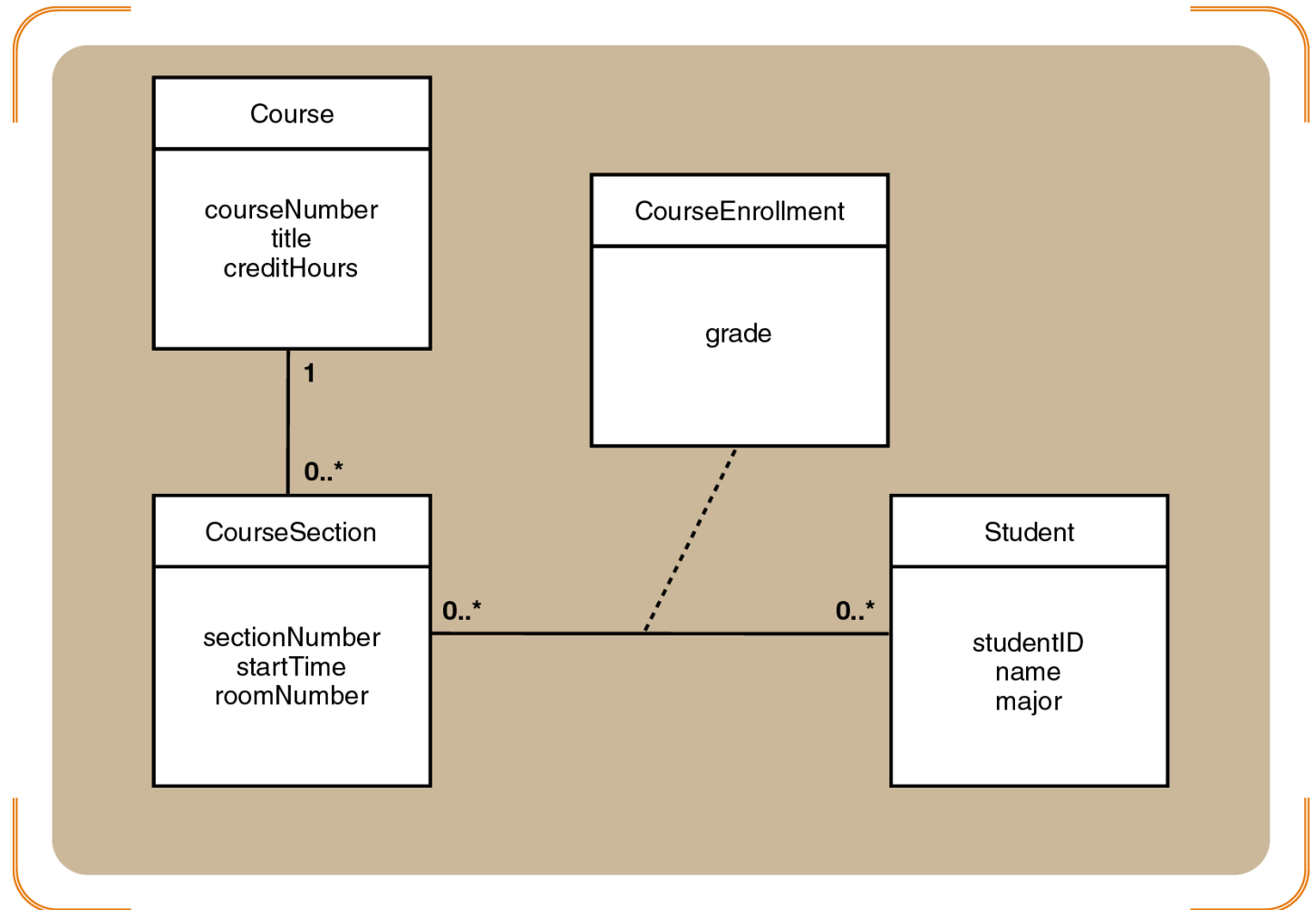
custNumber,
orderID and
itemID

Is this domain?

Domain Model: Class Diagram Notation

Figure 5-24

A Refined University Course Enrollment Domain Model Class Diagram With an *Association Class*



Hierarchies

Generalisation/specialisation notation:

- ***Inheritance*** hierarchy
- Rank things from the more general to the more specialised
 - Motor vehicle class includes trucks, cars, buses

Classification: means of defining classes of things:

- Superclass: ***generalisation*** of a class
- Subclass: ***specialisation*** of a class

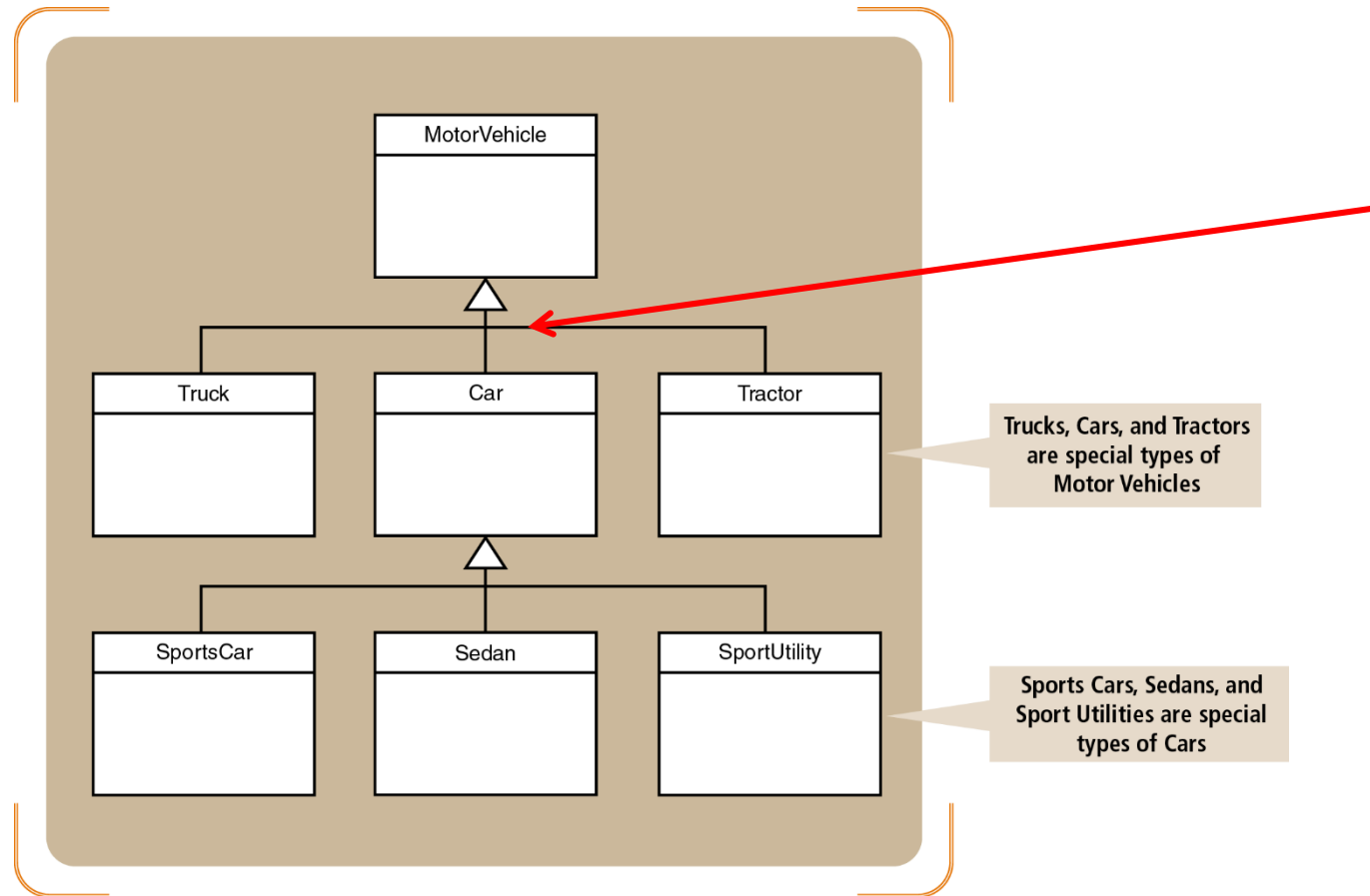


Figure 5-25

A Generalisation/Specialisation Hierarchy Notation for Motor Vehicles

Note: Inheritance applies from Parent to Child Classes

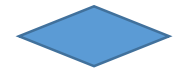
Hierarchies

Whole-part Hierarchy Notation

- “The whole is equal to the sum of the parts”

Two types of whole-part hierarchies

- Aggregation: association with independent parts Indicated by a hollow diamond
 - Example: keyboard is part of computer system
- Composition: association with dependent part Indicated by a solid diamond
 - Example: CRT and monitor

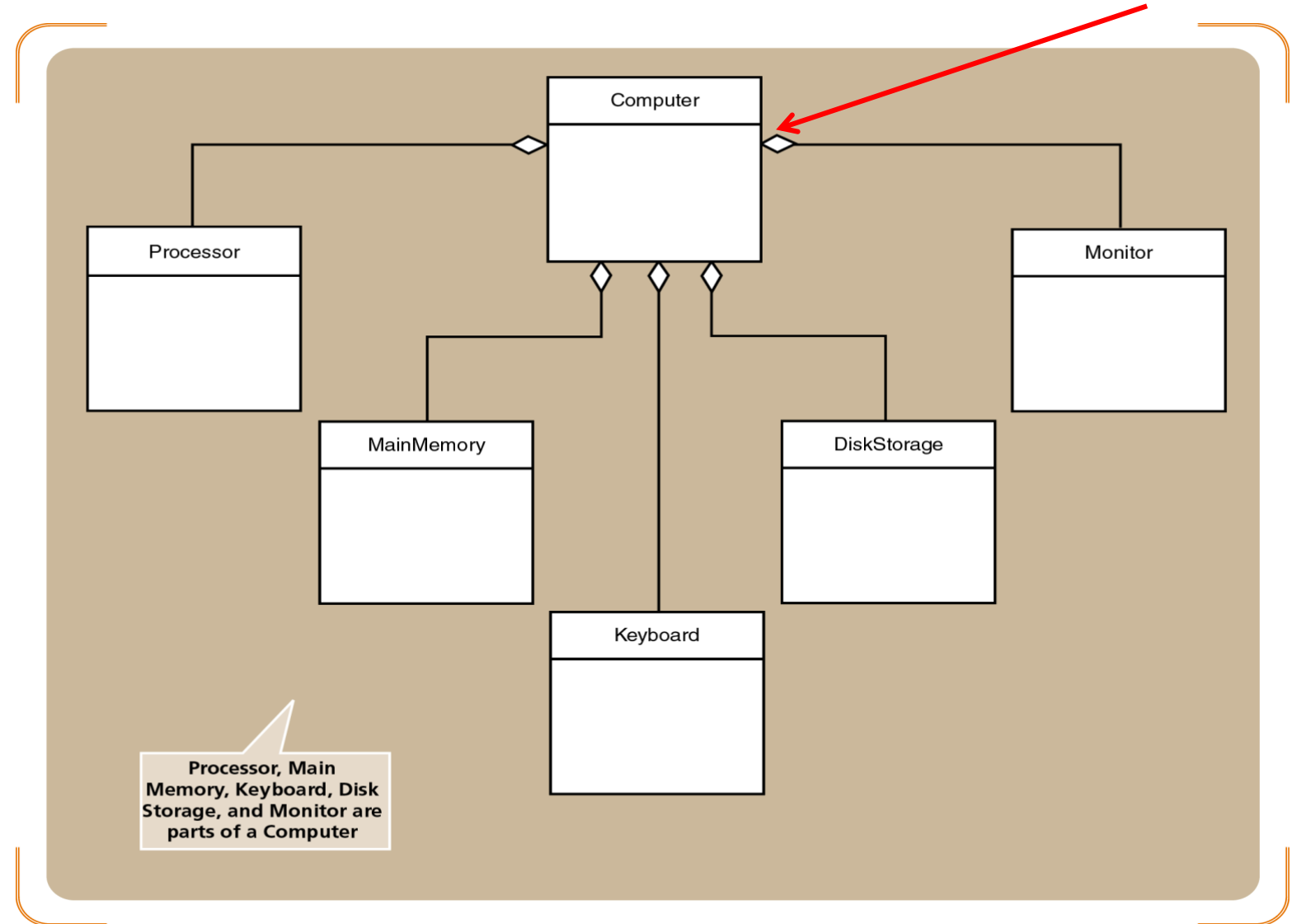


Multiplicity applies to whole-part relationships

Hierarchies

Figure 5-27

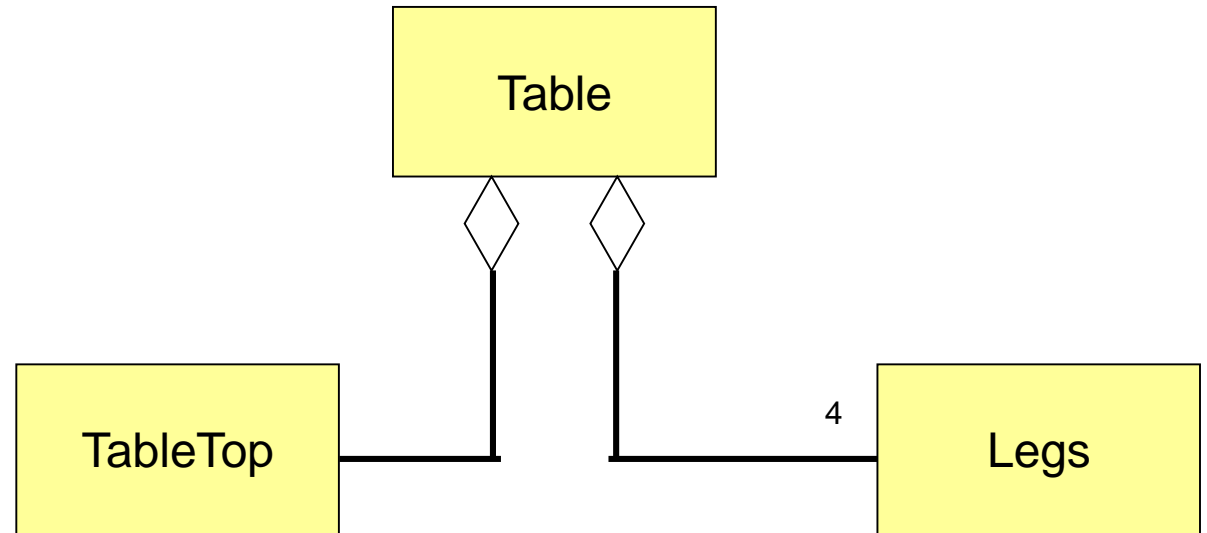
Whole-part (Aggregation) Associations Between a Computer and Its Parts (each of which can exist separately)



Review / Extension

Two types of whole-part hierarchies:

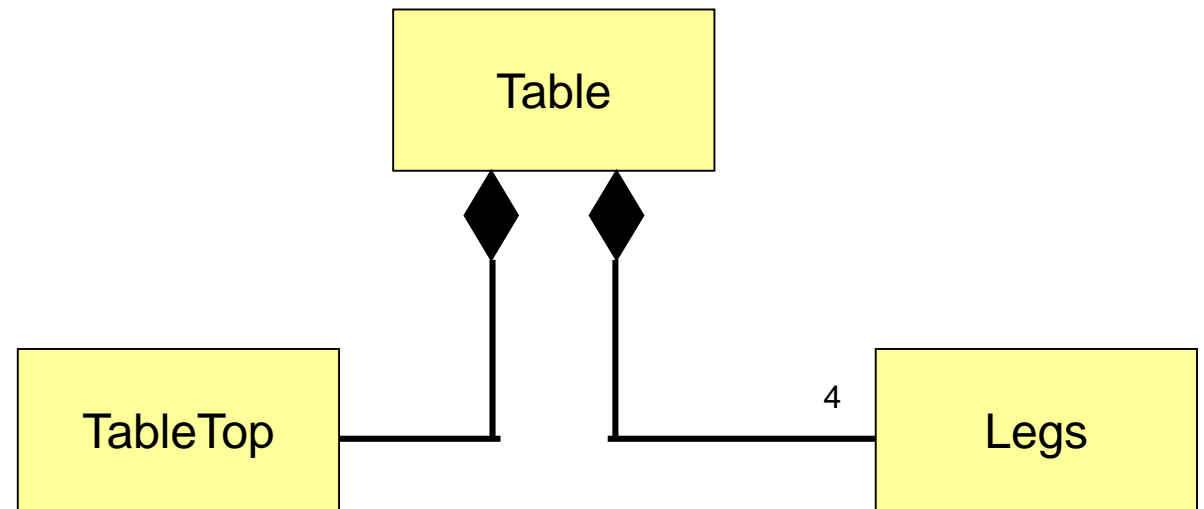
- **Aggregation:**
 - association with independent parts (the whole is only the sum of its parts: if one ceases to exist the whole is still OK)
- Depicted with an *Empty* Diamond
 - Example: a **cabinetmakers** view of a table could be that:
 - A table is made up of legs and a tabletop
 - If the table is disassembled, the components survive and can be reused.



Review / Extension

Two types of whole-part hierarchies

- **Composition:**
 - association with dependent part
- Depicted with a **Filled** Diamond
 - Example: A **Table Owners** view of a table could be that:
 - The whole is more than the sum of its parts
 - If any part is damaged then the whole is damaged, for example if the tabletop is damaged then the table is useless



Questions ?



(pexels.com, n.d.)

Hierarchies

A short Introduction into Design Class Diagrams

Design Class Diagrams

- Models classes into precise software analogs
- Includes domain class information *plus* methods
- Triangle symbol between classes indicates *inheritance*
- *Properties* of attributes are shown with *curly braces*
 - *eg {key}*

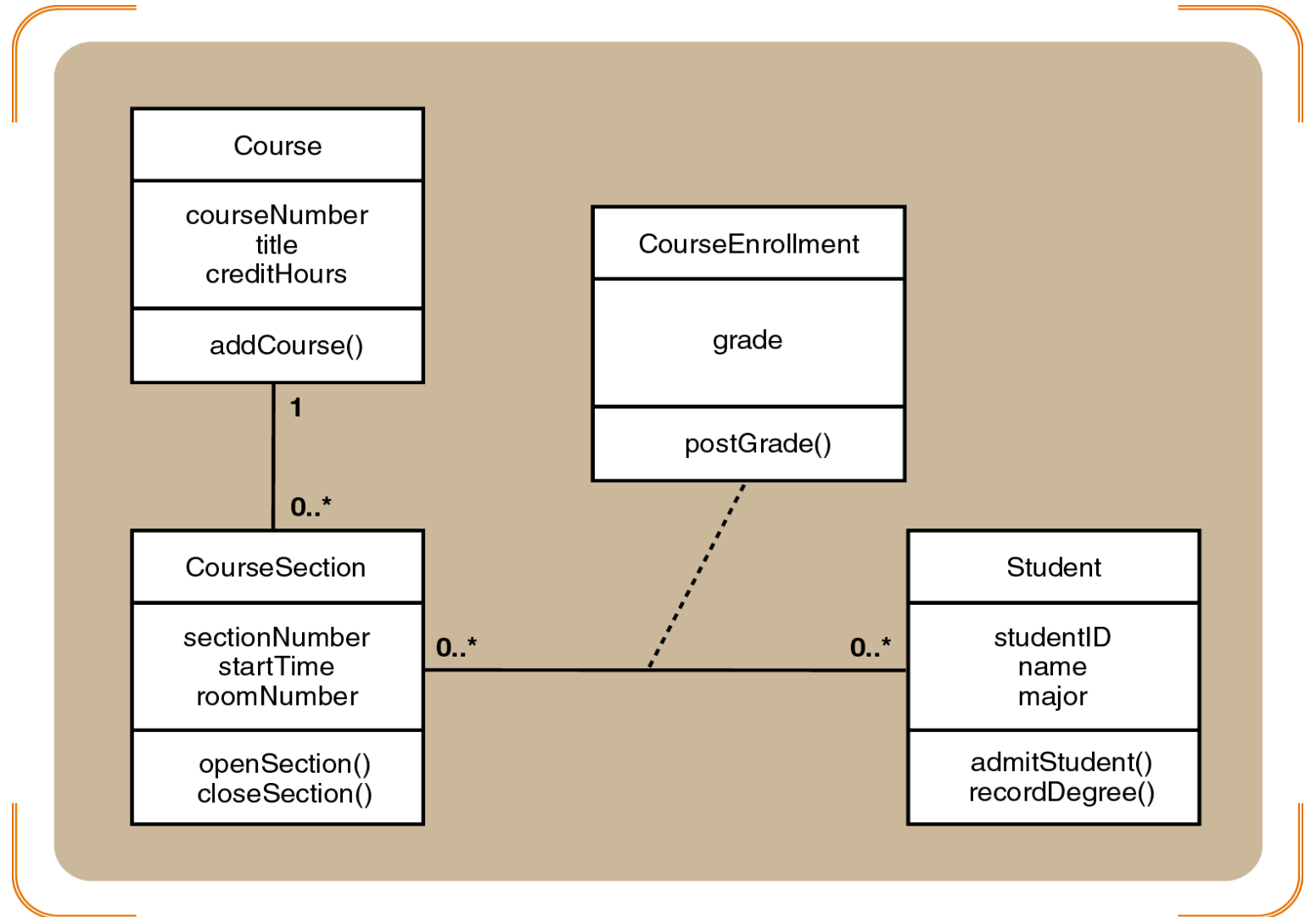
Class fundamentals

- Instances of a class (objects) manage their own data
- **Abstract** classes are not instantiated (created)
- Subclasses *inherit* attributes/behaviors from superclass

Hierarchies

Figure 5-29

University Course
Enrollment Design
Class Diagram (With
Methods)

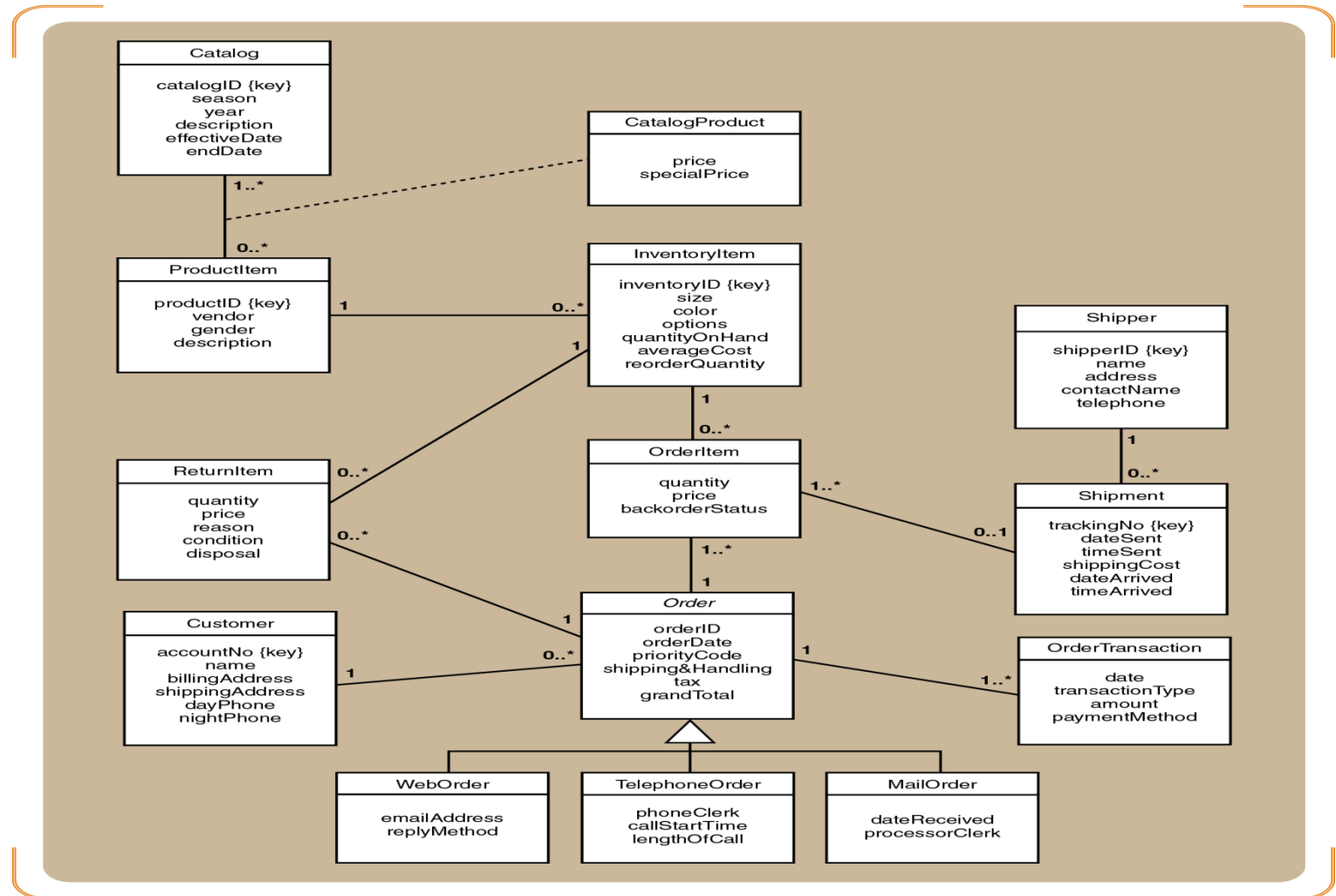


RMO's Domain Class Diagram

- Derived from noun list developed in Figure 5-31
- RMO domain class diagram shows attributes
- Domain Class Models do not show methods as the Problem domain classes reflect high-level view of RMO's use cases

RMO's Domain Class Diagram

Figure 5-31:
Rocky Mountain
Outfitters Domain
Model Class Diagram



Summary

- The Unified Process is Use Case Driven
- Actors who are they?
- What functionality is required?
- What data is moving into and out of the system?
- Use Case
- Class Diagram
- Notes: System Sequence Diagram

Weekly Activity

- Watch the videos and have a look at the additional reading identified in the lecture slides
- There is a forum for week 4
- Lab questions
- Assessment Item 1
 - Further analysis
 - Work on the assignment

Reminder:

The online chapters, videos etc for the 2016 text can be accessed via: http://www.cengage.com/cgi-wadsworth/course_products_wp.pl?fid=M20b&product_isbn_issn=9781305117204&token

References

- Athuraliya . (2022, December 12). What is Sequence Diagram? Complete Guide with Examples. Retrieved 31 May 2024 from <https://createlly.com/blog/diagrams/sequence-diagram-tutorial/>
- LucidChart (2017, July 21) UML Class Diagram Tutorial. [video] Retrieved 31 May 2024 from <https://www.youtube.com/watch?v=UI6lqHOVHic>
- MargretPosch (2015 UML Class Diagrams - Association and Multiplicity) [video] Retrieved 31 May 2024 from <https://www.youtube.com/watch?v=BhEoV57nj0Q>
- Metal Deploye Resistor (n.d) Contact Us [image] Retrieved 1 June 2020 from <http://www.mdresistor.com/en/harmonic-filter-resistor/contact-us-email-meta-deploye-resistor-4/>
- Satzinger, Jackson and Burd. (2005) *Object-Oriented Analysis & Design with the Unified Process*, Thomson Publishing Co.
- Satzinger, Jackson and Burd. (2016) *Systems Analysis and Design in a Changing World*, Cengage Learning.

Confused? Do You Have Questions ?



(pexels.com, n.d.)

Thank you