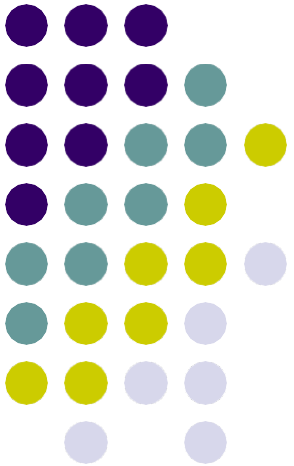


Course Codes: IS332
Course: Analysis and Design of Information Systems



By

Dr. Sherif Zahran

Information Systems Department
Faculty of Computers and Artificial Intelligence
Cairo University, Egypt

Lecture Rules







Project phases delivery form



- Please fill out the following form when submitting any phase: https://forms.gle/5tsks6cZURf1cwNUA
- This form is used to submit your different **deliverables** in course IS332 in the academic year (2022-2023)
- Only filled by the **team leader**
- Please submit your file named as:
  - T<Team number>-D<Deliverable number>.docx or .pdf.

Example: T12-D1.docx

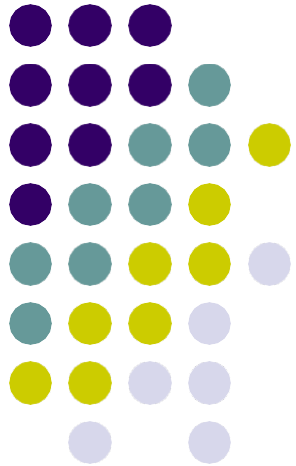
  - Do not add spaces in the filename.
  - Avoid zipping the file unless it is larger than 10 MB.

Domain Modeling

Chapter 4

Systems Analysis and Design
in a Changing World 7th Ed

Satzinger, Jackson & Burd



Chapter 4 Outline



- “Things” in the Problem Domain
- The Entity-Relationship Diagram
- The Domain Model Class Diagram
- The State Machine Diagram – Identifying Object Behavior

Learning Objectives

- Explain how the concept of “things” in the problem domain also define requirements
- Identify and analyze data entities and domain classes needed in the system
- Read, interpret, and create an entity-relationship diagram
- Read, interpret, and create a domain model class diagram
- Understand the domain model class diagram for the RMO Consolidated Sales and Marketing System
- Read, interpret, and create a state machine diagram that models object behavior

Overview





This chapter focuses on another key concept for defining requirements— **data entities** or **domain classes**



In the R M O Tradeshow System from Chapter 1, some domain classes are Supplier, Product, and Contact



In this chapter's opening case Waiters on Call, examples of domain classes are Restaurants, Menu Items, Customers, Orders, Drivers, Routes, and Payments

Things in the Problem Domain

- **Problem domain**—the specific area (or domain) of the users' business need that is within the **scope** of the new system.
- "Things" are those **items** users work with when accomplishing tasks that need to be remembered
- Examples of "Things" are products, sales, shippers, customers, invoices, payments, etc.
- These "Things" are modeled as **domain classes** or **data entities**
- In this course, we will call them domain classes.
- In database class you may call them data entities

Things in the Problem Domain: Two Techniques for Identifying Them

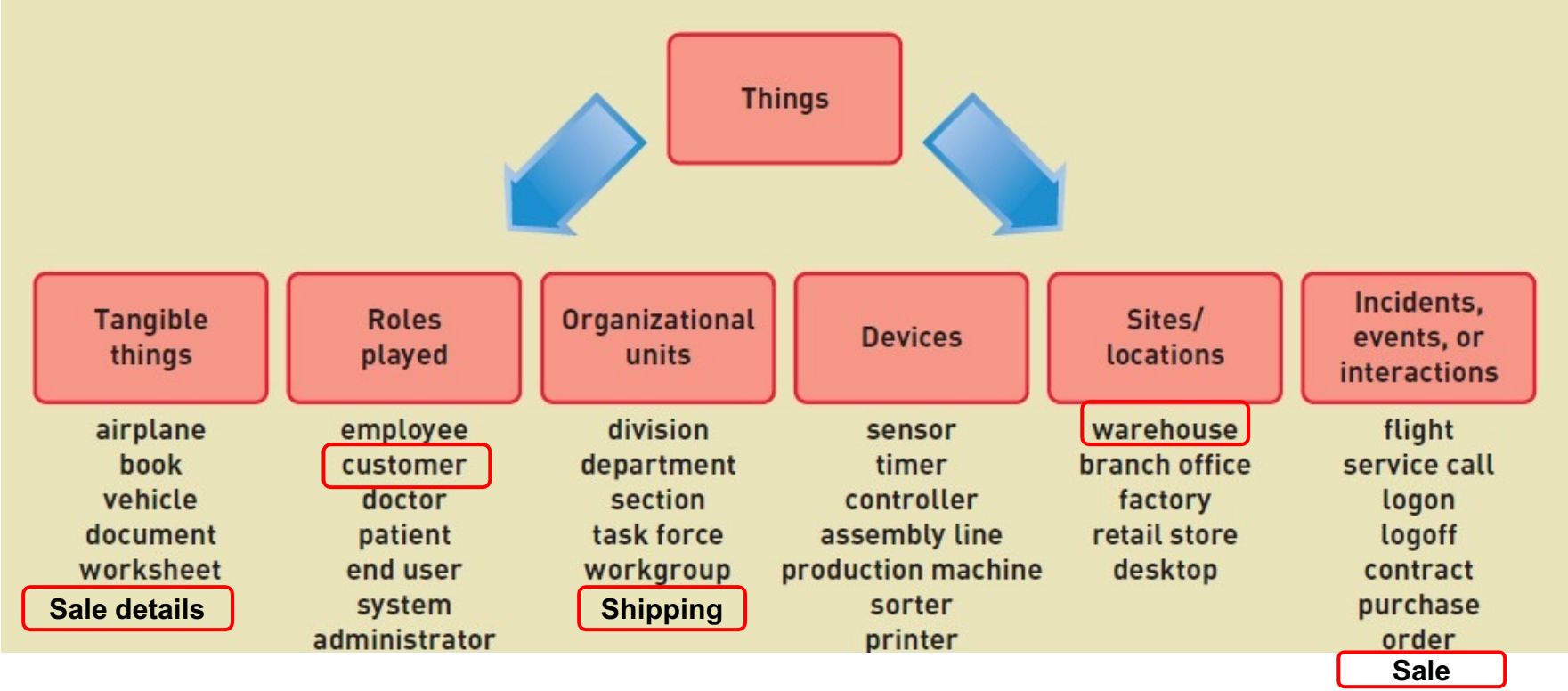
- **Brainstorming Technique**
  - Use a **checklist** of all the usual types of things typically found and brainstorm to identify **domain classes** of each type
- **Noun Technique**
  - Identify all the nouns that come up when the system is described and determine if each is a **domain class**, an **attribute**, or not something we need to remember



Brainstorming technique

Brainstorming Technique

- Are there any **tangible** things?
- Are there any **organizational units**? **Sites/locations**?
- Are there **incidents** or **events** that need to be recorded?



Brainstorming Technique: Steps

1. Identify a **user** and a **set of use cases**
2. Brainstorm with the user to identify things involved when carrying out the use case—that is, **things** about which information should be captured by the system.
3. Use the **types of things** (categories) to systematically ask questions about potential things, such as the following:
    1. Are there any **tangible** things you store information about?
    2. Are there any **locations** involved?
    3. Are there **roles** played by people that you need to remember?
4. Continue to work with **all types of users** and **stakeholders** to expand the brainstorming list
5. **Merge** the results, **eliminate** any duplicates, and **compile** an initial list



Noun technique

The Noun Technique



- A technique to **identify** problem domain classes (things) by finding, classifying, and refining a list of nouns that come up in discussions or documents
- **Popular** technique. **Systematic**.
- Does end up with long lists and many nouns that are not things that need to be stored by the system
- Difficulty identifying synonyms and things that are really attributes
- Good place to start when there are no users available to help brainstorm

The Noun Technique: Steps (1 of 3)

1. Using the **use cases**, **actors**, and other information about the system— including **inputs** and **outputs**—identify all nouns.
   - For the RMO CSMS, the nouns might include customer, product item, sale, confirmation, transaction, shipping, bank, change request, summary report, management, transaction report, accounting, back order, back order notification, return, return confirmation…
2. Using other information from **existing systems**, **current procedures**, and **current reports or forms**, add items or categories of information needed.
   - For the RMO CSMS, these might include price, size, color, style, season, inventory quantity, payment method, and shipping address.

The Noun Technique: Steps (2 of 3)

3. **As this list of nouns builds, refine it.**

- Ask these questions about each noun to help you decide whether you should **include** it:
  - Is it **a unique thing** the system needs to know about?
  - Is it **inside the scope** of the system I am working on?
  - Does the system **need to remember** more than one of these items?

The Noun Technique: Steps (2 of 3)

3. **As this list of nouns builds, refine it.**

- Ask these questions to decide to **exclude** it:
  - Is it really a **synonym** for some other thing I have identified?
  - Is it really **just an output** of the system produced from other information I have identified?
    - E.g., Date of Birth (DoB) and age
  - Is it really **just an input** that results in recording some other information I have identified?
    - E.g., Address details, zip, … , Address

The Noun Technique: Steps (2 of 3)

3. **As this list of nouns builds, refine it.**

- Ask these questions to **research** it:
  - Is it likely to be a specific piece of information (**attribute**) about some other thing I have identified?
  - Is it something I might need **if assumptions change**?

The Noun Technique: Steps (3 of 3)

4. Create a **master list** of all nouns identified and then note whether each one should be included, excluded, or researched further.
5. **Review** the list with users, stakeholders, and team members and then **define** the list of things in the problem domain.

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.

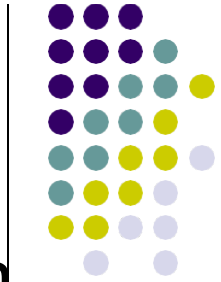
Partial List of Nouns for RMO (2 of 2)

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 R M O 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.
Management	We know who they are. No need to store.
Marketing	We know who they are. No need to store.
Merchandising	

Domain Classes

23

Details about Domain Classes



- Attribute— describes one piece of information about each instance of the class
  - Customer has first name, last name, phone number
- Identifier or key
  - One attribute uniquely identifies an instance of the class. Required for data entities, optional for domain classes. Customer ID identifies a customer
- Compound attribute
  - Two or more attributes combined into one structure to simplify the model. (E.g., address rather than including number, street, city, state, zip separately). Sometimes an identifier or key is a compound attribute.

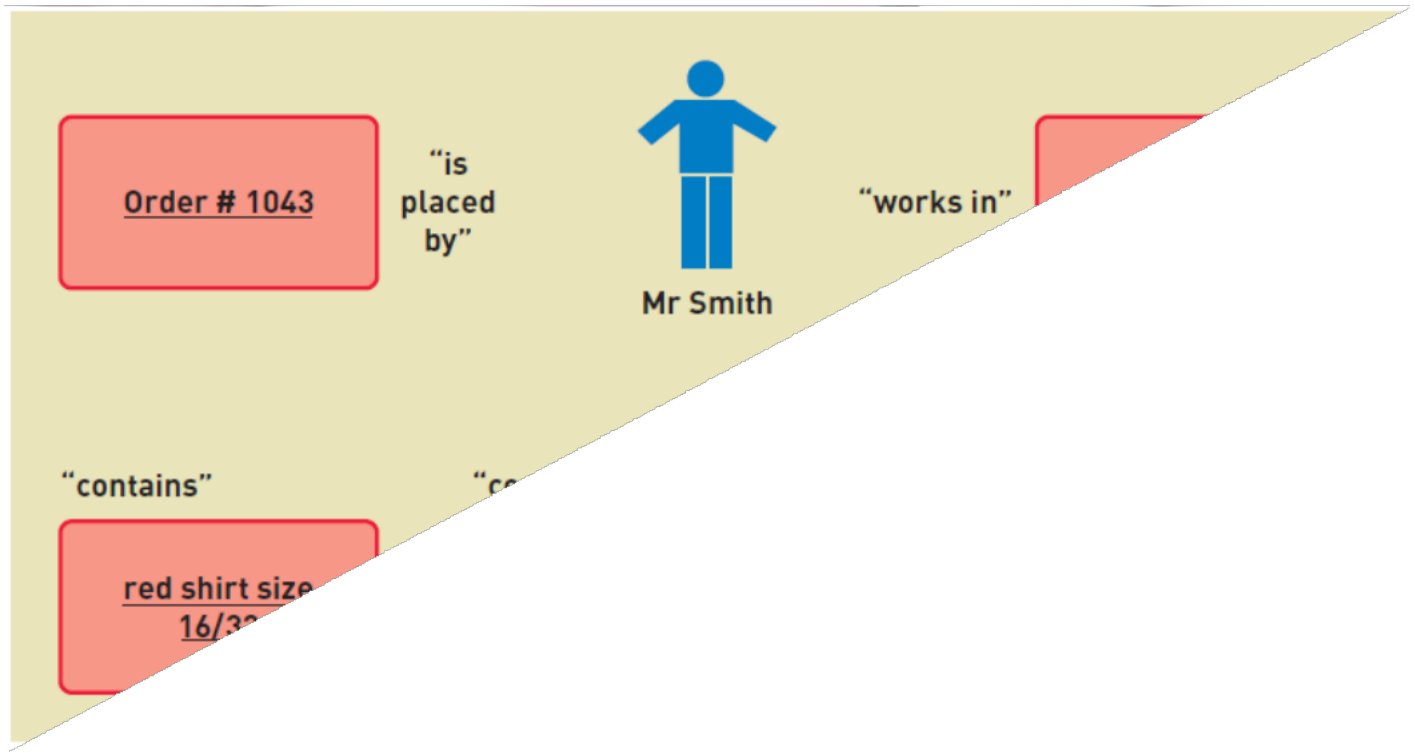
Attributes and Values

- Class is a type of thing. Object is a specific instance of the class. Each instance has its own values for an attribute

All customers have these attributes:	**Each customer has a value for each attribute:**		
Customer ID	101	102	103
First name	John	Mary	Bill
Last name	Smith	Jones	Casper
Home phone	555-9182	423-1298	874-1297
Work phone	555-3425	423-3419	874-8546

Associations Among Things

- Association— a naturally occurring relationship between classes (UML term)



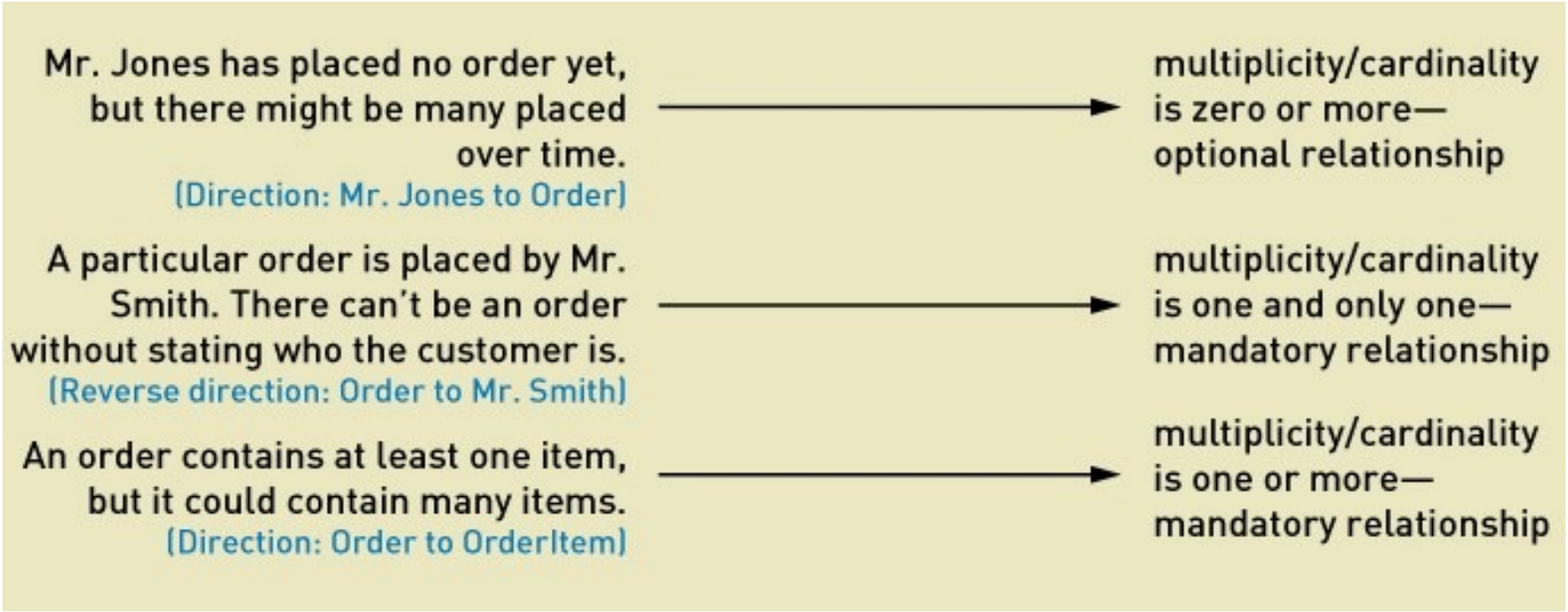
Just to Clarify…



- Called ***association*** on class diagram in UML
  - **Multiplicity** is term for the number of associations between classes: 1 to 1 or 1 to many (synonym to cardinality)
  - UML is the primary emphasis of this text
- Called ***relationship*** on ERD in database class
  - **Cardinality** is term for number of relationships in entity relationship diagrams: 1 to 1 or 1 to many (synonym to multiplicity)
- Associations and Relationships apply in two directions
  - Read them separately each way
  - A customer places an order
  - An order is placed by a customer

Minimum and Maximum Multiplicity

- Associations have minimum and maximum constraints
  - minimum is zero, the association is optional
  - If minimum is at least one, the association is mandatory



Types of Associations



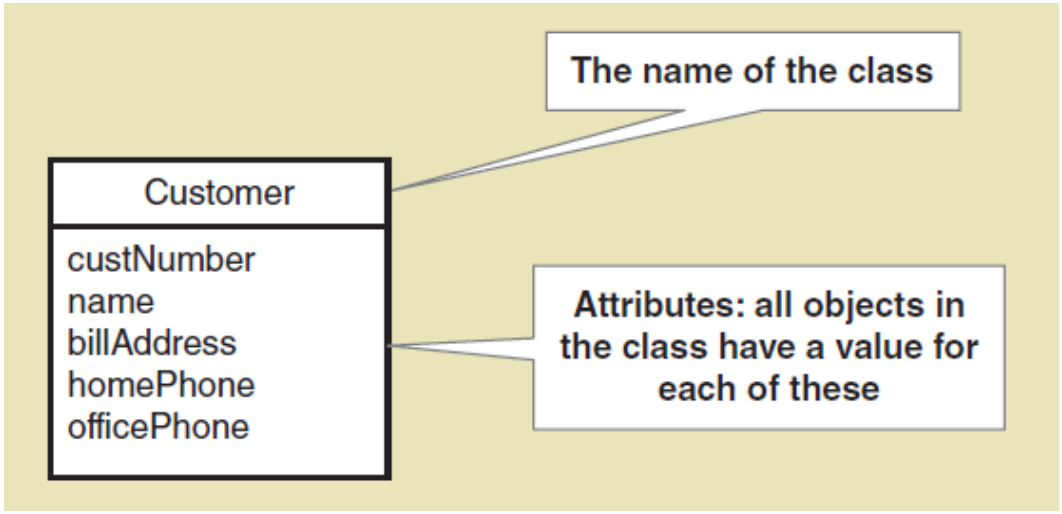
- Binary Association
  - Associations between exactly two different classes
    - Course Section includes Students
    - Members join Club
- Unary Association (recursive)
  - Associations between two instances of the same class
    - Person married to person
    - Part is made using parts
- Ternary Association (three)
- N-ary Association (between n)

The Domain Model Class Diagram



Class	A type of classification used to describe a collection of objects
Domain Class	Classes that describe objects in the problem domain
Class Diagram	A UML diagram that shows classes with attributes and associations (plus methods if it models software classes)
Domain Model Class Diagram	A class diagram that only includes classes from the problem domain, not software classes so **no** methods

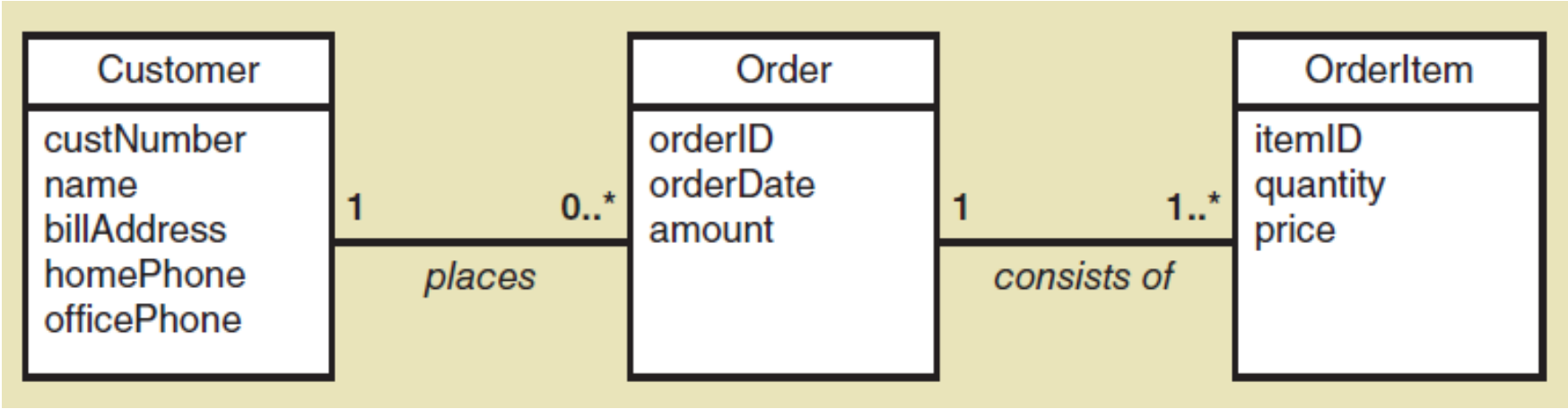
U M L Domain Class Notation



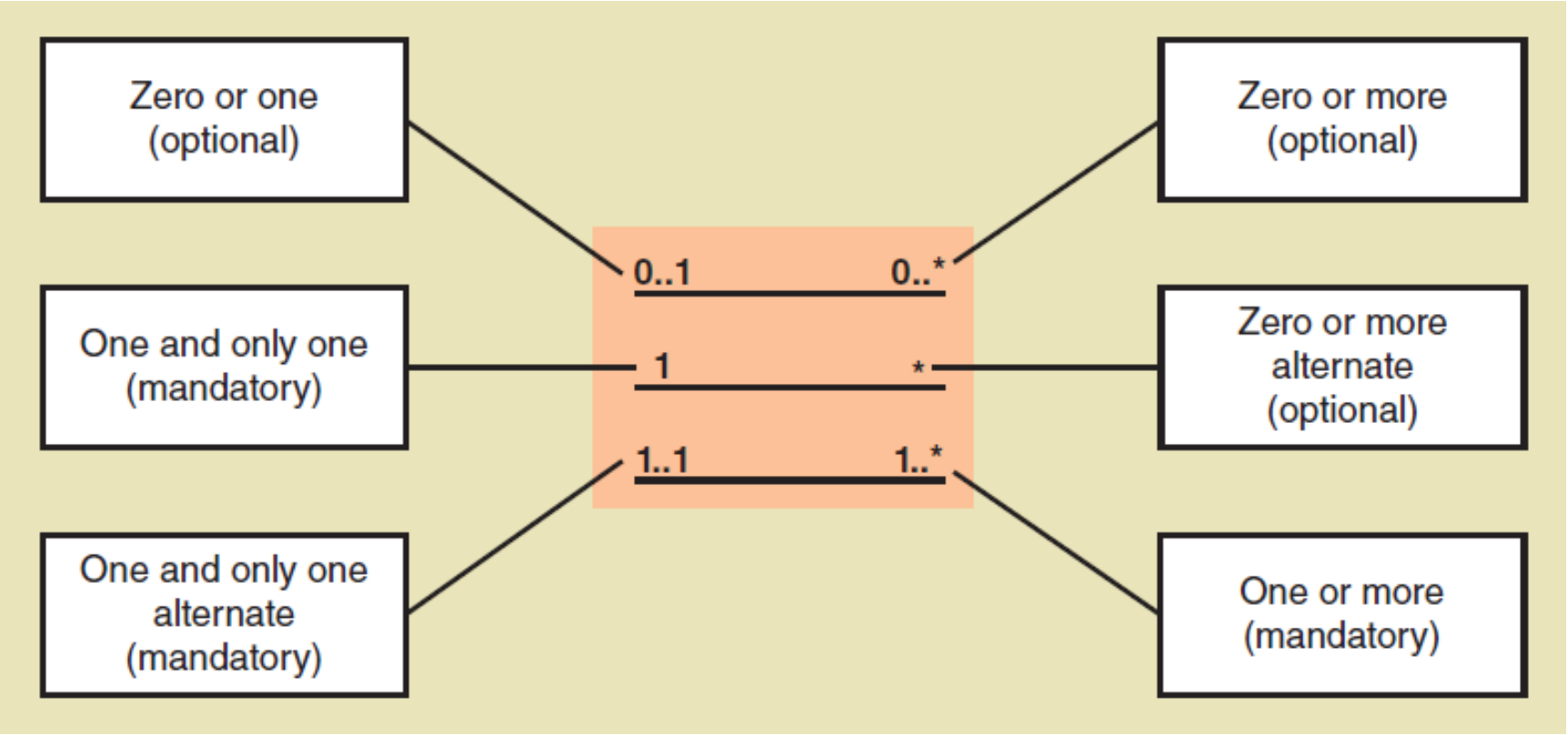
- **Domain class** a name and attributes (**no methods**)
- **Class name** is always **Capitalized**
- **Attribute** names are **not** capitalized and use **camelback notation** (words run together and second word is capitalized)
- **Compound class** names also use camelback notation

A Simple Domain Model Class Diagram

- A customer places zero or more orders
- An order is placed by exactly one customer
- An order consists of one or more order items
- An order item is part of exactly one order

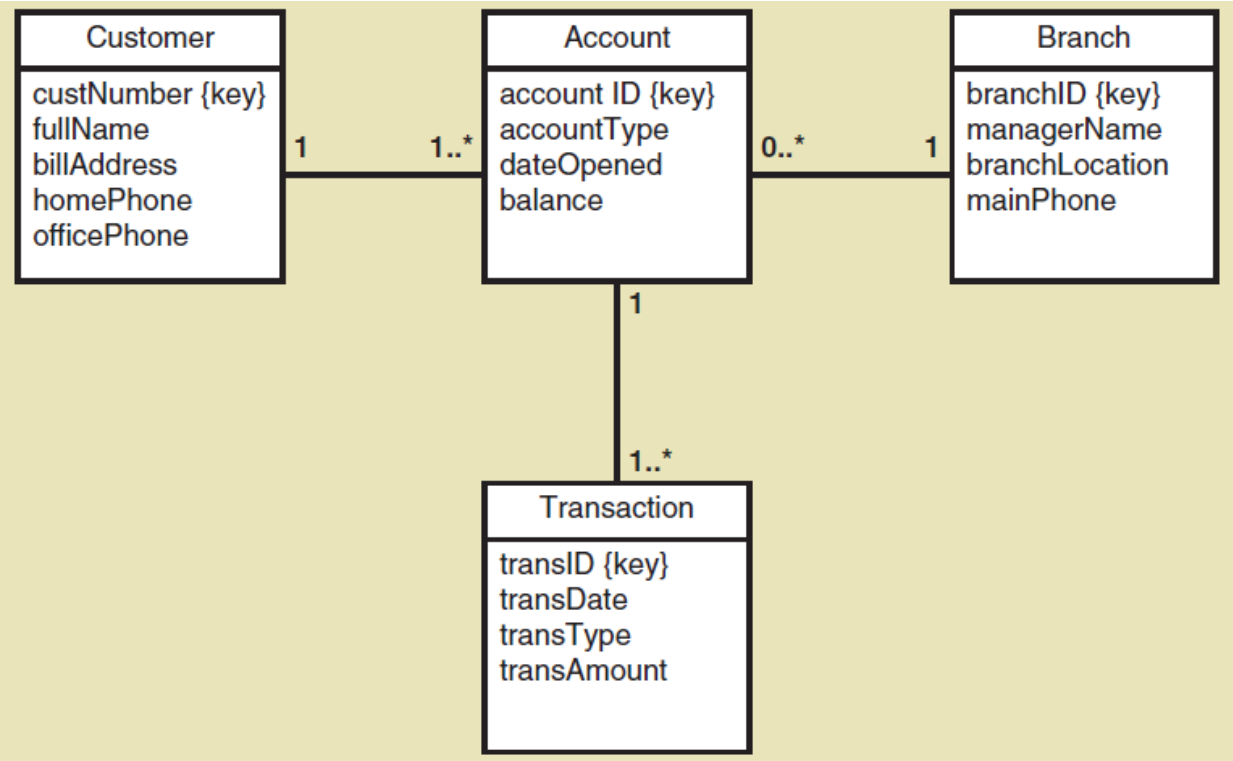


UML Notation for Multiplicity



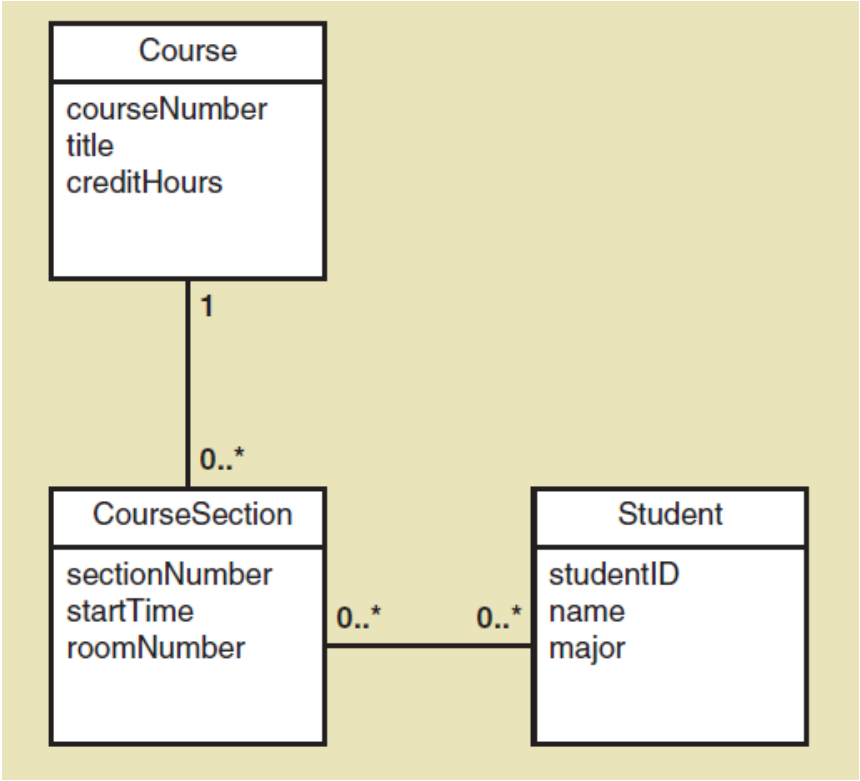
Domain Model Class Diagram (1 of 2)

- Bank with many branches
  - Note notation for the key
  - Note the precise notation for the attributes (camelback)
  - Note the multiplicity notation



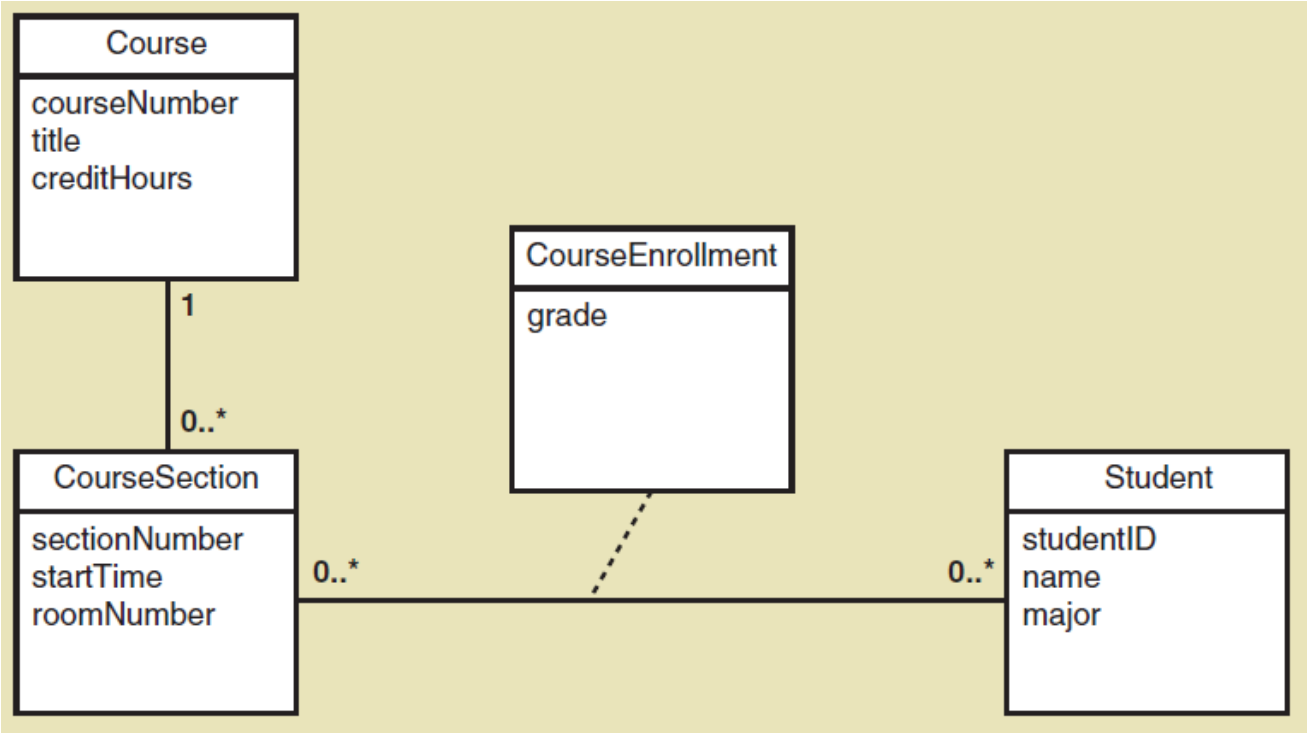
Domain Model Class Diagram (2 of 2)

- Course Enrollment at a University
- A Course has many CourseSections
- A CourseSection has many Students and a Student is registered in many CourseSections
- **Problem**
  - How/where to capture student grades?



Refined Course Enrollment Model with an Association Class CourseEnrollment

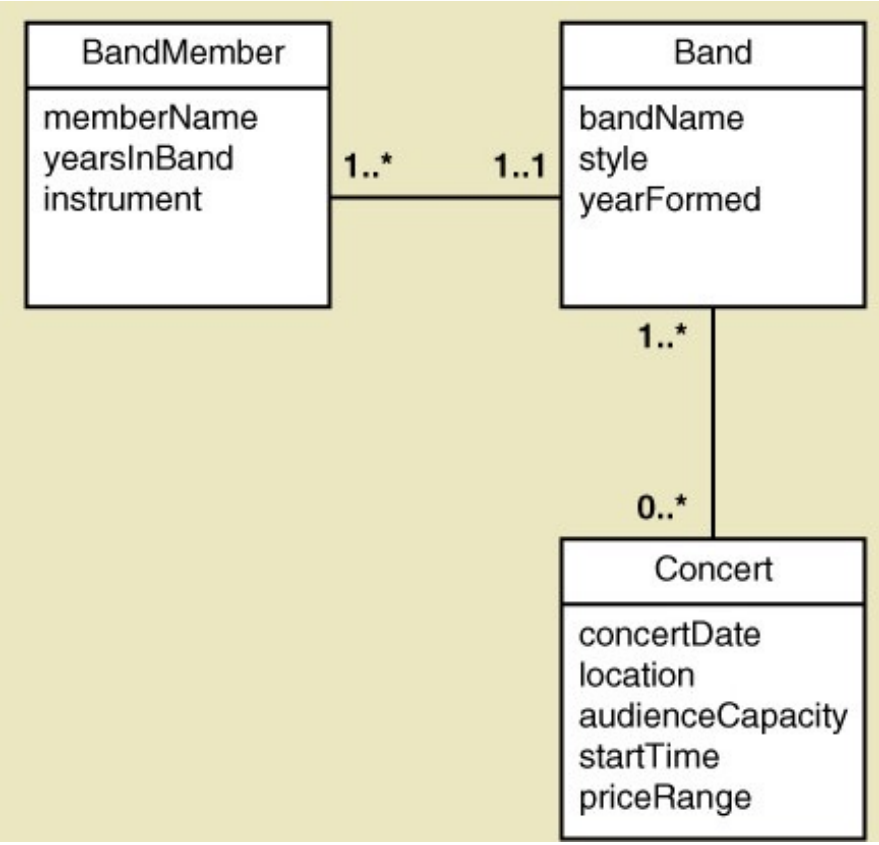
- **Association class**— an association that is treated as a class in a many to many association because it has attributes that need to be remembered (such as grade)



Association Class Properties

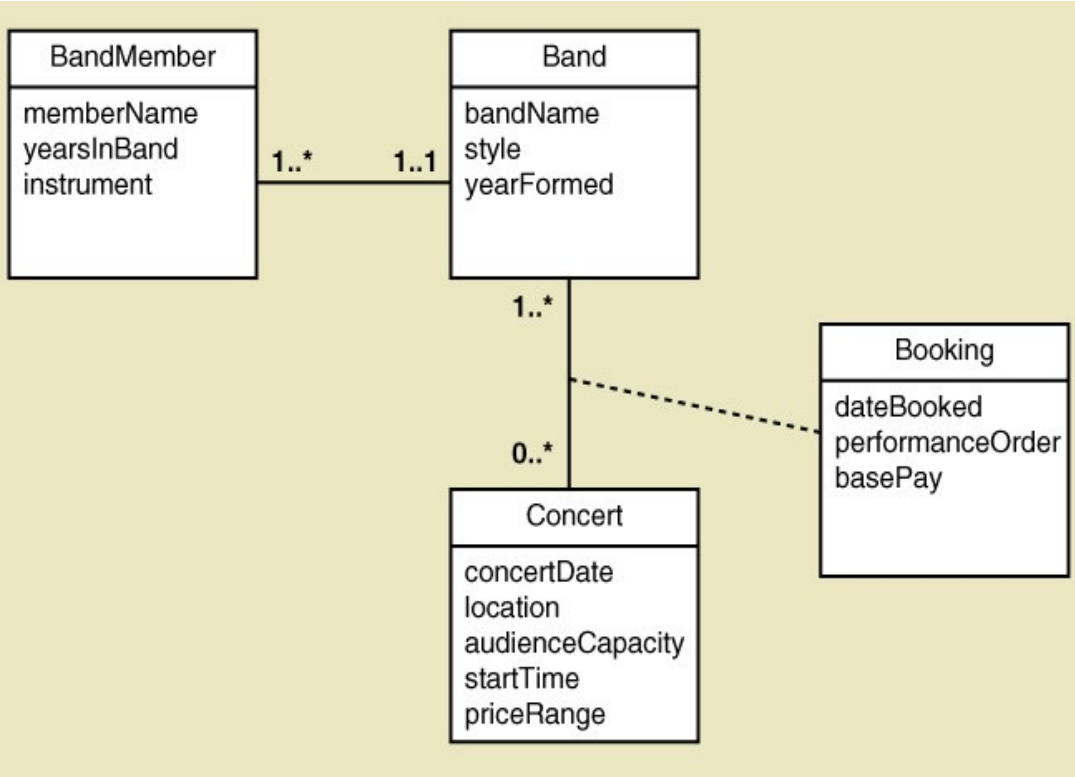
- The association class **is** the same "thing" as the association itself
- The unique identifier (key) for the association class is the concatenation of the keys of the attached classes
  - In the previous example the key for CourseSection is **CourseNumber+SectionNumber**
  - Hence the key for CourseEnrollment is **CourseNumber+SectionNumber+Student ID**
- Note: If more information is required to uniquely identify instances of the association class, then the model is incorrect, i.e., if the key cannot be formed by the concatenation of the endpoint keys, it is in error.

Band with members and concerts



- How many bands can a person play in?
- For a band, how many concerts can it play in?
- For a concert, how many bands may be playing?
- What attributes can you use for keys? Do you need to add “key” attributes?

Band with Concert Booking Information



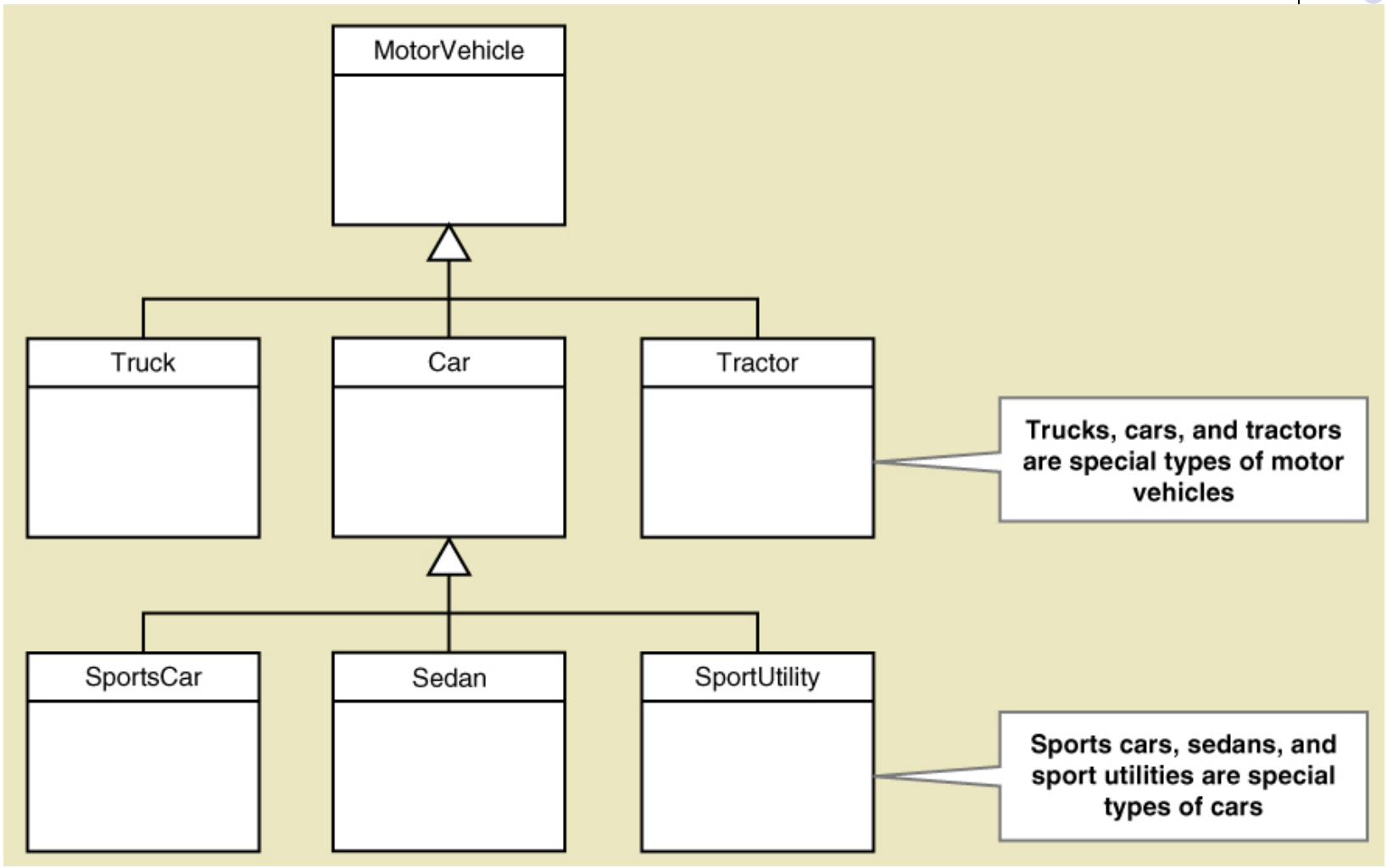
- Note: The association class (Booking) also provides a name and meaning for the association
- Given the keys you identified, what is the key for the Booking class? Does it uniquely identify instances?

More Complex Issues about Classes:
Generalization/Specialization Relationships



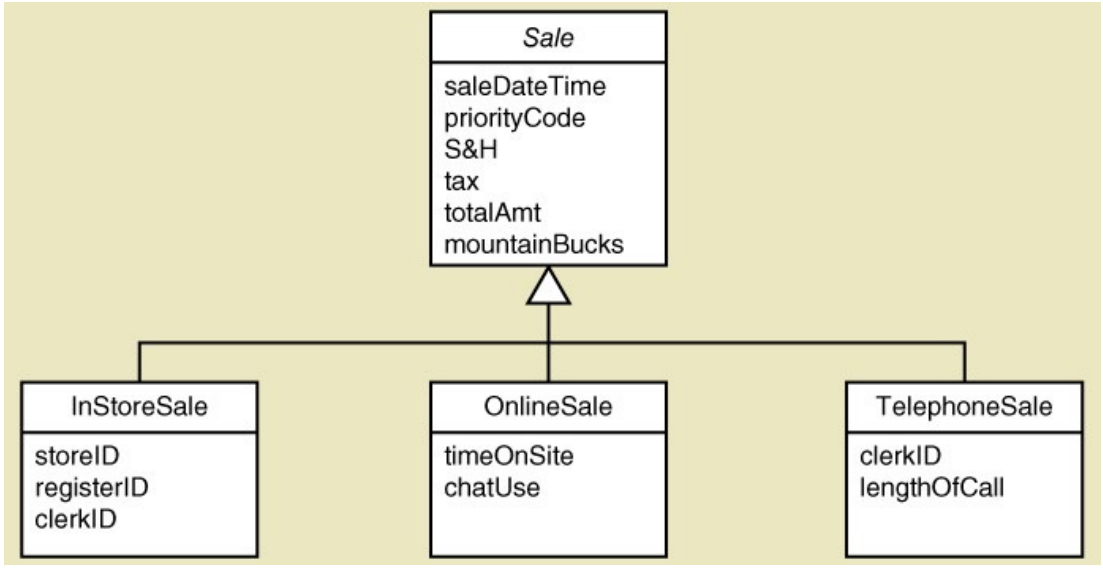
Generalization/ Specialization	• A hierarchical relationship where subordinate classes are special types of the superior classes. • Often called an Inheritance Hierarchy
Superclass	• the superior or more general class in a generalization/specialization hierarchy
Subclass	• the subordinate or more specialized class in a generalization/specialization hierarchy
Inheritance	• the concept that subclasses classes inherit characteristics of the more general superclass

Generalization/Specialization



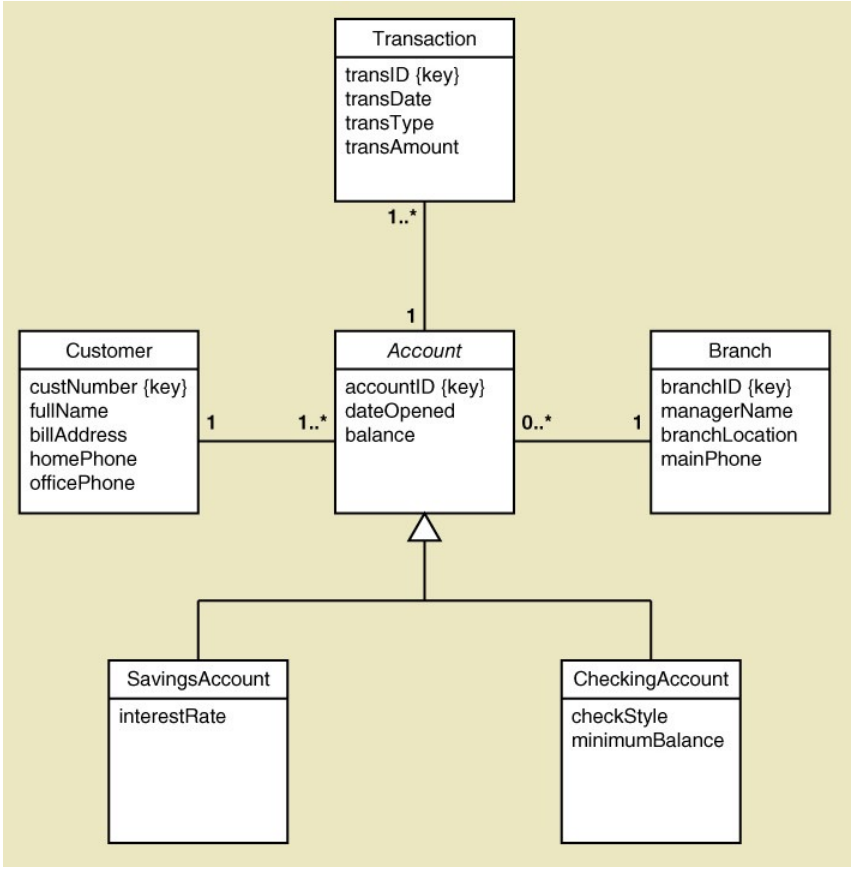
Generalization/Specialization:
Inheritance for RMO Three Types of Sales

- **Abstract class**— a class that allows subclasses to inherit characteristics but never gets instantiated. In Italics (*Sale*)
- **Concrete class**— a class that can have instances
- **Inheritance** – Attributes of OnlineSale are:
  - timeOnSite, chatUse, saleDateTime, priorityCode, S&H, tax, totalAmt…



Generalization/Specialization:
Inheritance for the Bank with Special Types of Accounts

- A SavingsAccount has 4 attributes
- A CheckingAccount has 5 attributes
- Note: the subclasses inherit the associations too



More Complex Issues about Classes:
Whole Part Relationships

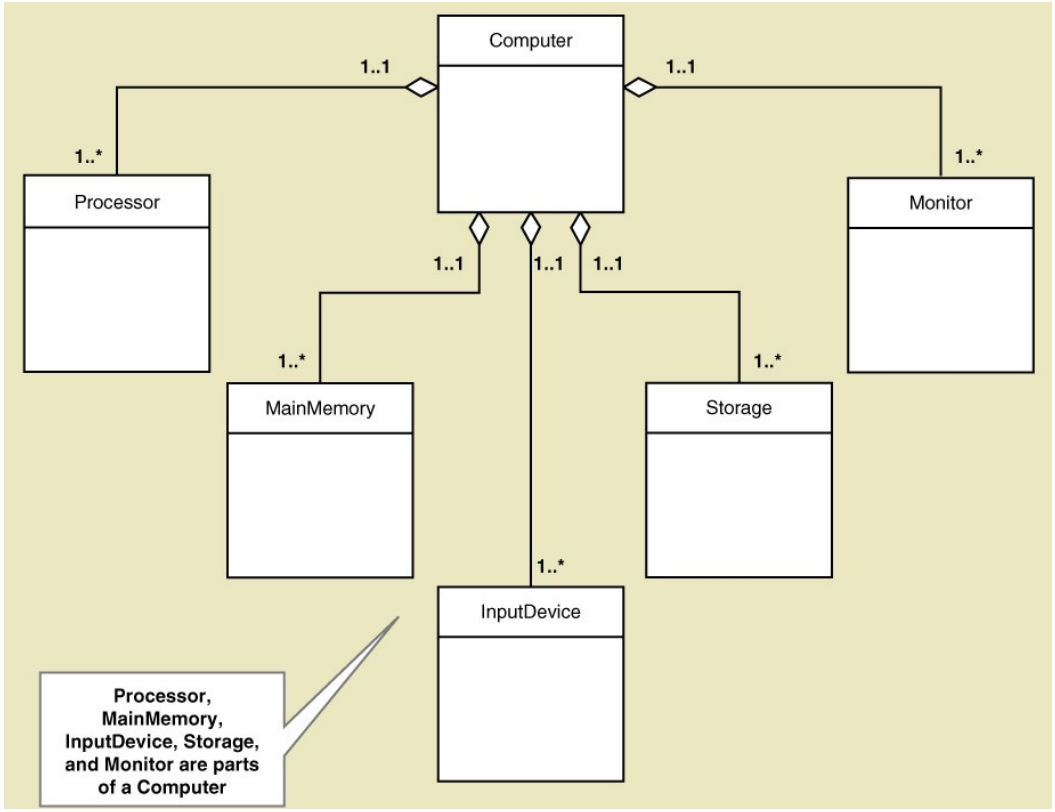


- **Whole-part relationship**— a relationship between classes where one class is part of or a component portion of another class

1. **Aggregation**— a whole part relationship where the component part exists separately and can be removed and replaced (UML diamond symbol on next slide)
   - Computer has disk storage devices (storage devices exist apart from computer)
   - Car has wheels (wheels can be removed and still be wheels)
2. **Composition**— a whole part relationship where the parts cannot be removed (filled in diamond symbol)
   - OrderItem on an Order (without the Order, there are no OrderIterms)
   - Chip has circuits (without the chip, there are no circuits)

Whole Part Relationships:
Computer and its Parts

- Note: this is **aggregation**, with diamond symbol.
- **Composition** is a filled diamond symbol.



More on UML Relationships



- There are actually three types of ***relationships*** in class diagrams
  1. Association Relationships
     - These are associations discussed previously, just like ERD relationships
  2. Whole Part Relationships (aggregation, composition)
     - One class is a component or part of another class
  3. Generalizations/Specialization Relationships
     - Inheritance



State Machine Diagram

Object Behavior – State Machine Diagram

- Each class has objects that may have status conditions or "**states**"
- Object behavior consists of the various states and the movement between these states
- **State** – a condition during an object's life when it satisfies some criterion, performs an action, or waits for an event
- **Transition** – the movement of an object from one state to another

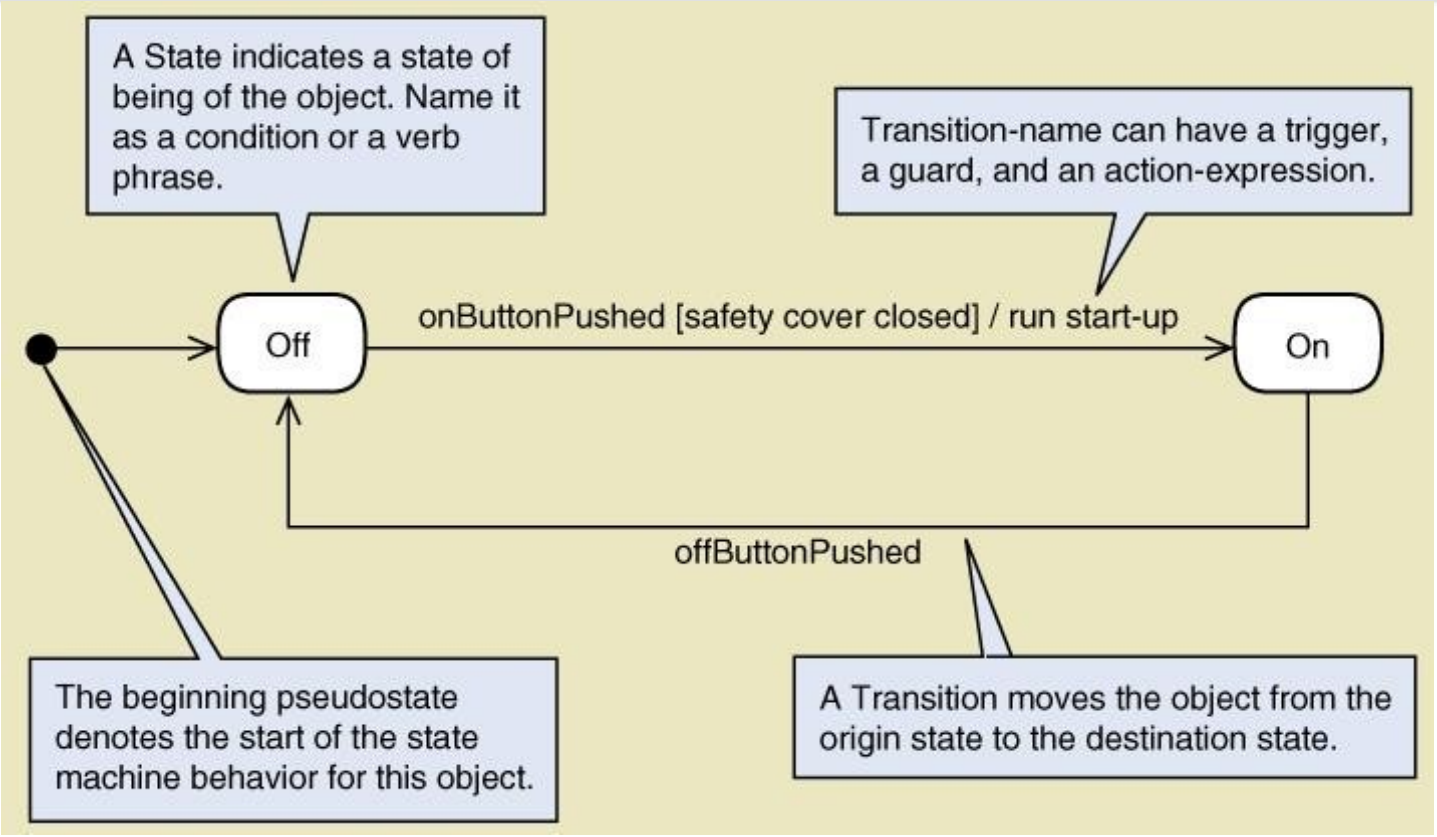
State Machine Diagram

- **State Machine Diagram** – a diagram which shows the life of an object in states and transitions
- **Origin state** – the original state of an object before it begins a transition
- **Destination state** – the state to which an object moves after completing a transition
- **pseudostate** – the starting point in a state machine diagram. Noted by a black circle.
- **action-expression** – some activity that must be completed as part of a transition
- **guard-condition** – a true/false test to see whether a transition can fire

State Machine for a Printer

- Syntax of transition statement

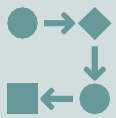
transition-name (parameters, …) [guard-condition] / action-expression



Concurrency in a State Machine Diagram



Concurrent states – when an object is in one or more states at the same time



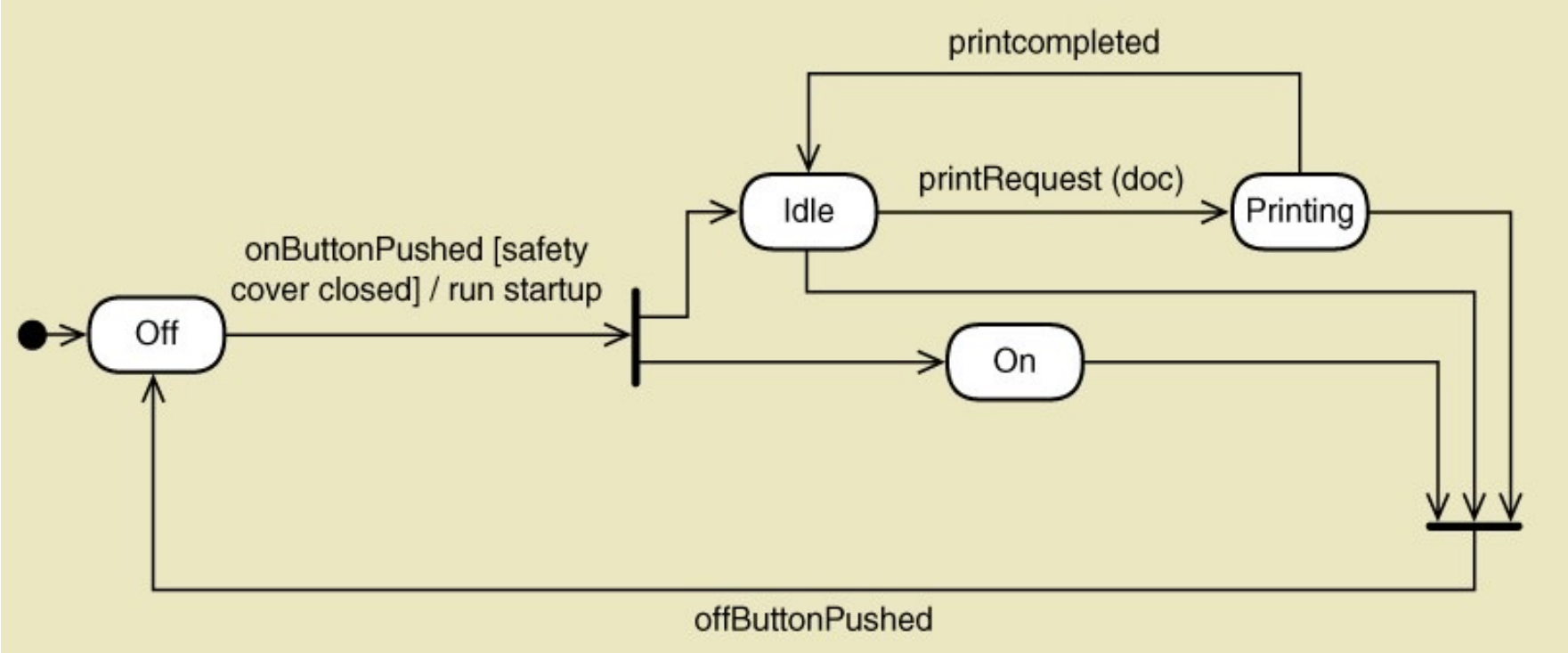
Path – a sequential set of connected states and transitions



Concurrent paths – when multiple paths are being followed concurrently, i.e. when one or more states in one path are parallel to states in another path

Printer with Concurrent Paths

- Concurrent paths often shown by synchronization bars (same as Activity Diagram)
- Multiple exits from a state is an "**OR**" condition.
- Multiple exits from a synchronization bar is an "**AND**" condition.



Creating a State Machine Diagram: Steps (1 of 2)

1. **Review** the class diagram and **select** classes that might require state machine diagrams
2. For each class, **make** a list of status conditions (states) you can identify
3. **Begin** building diagram fragments by identifying transitions that cause an object to leave the identified state
4. **Sequence** these states in the correct order and **aggregate** combinations into larger fragments
5. **Review** paths and **look** for independent, concurrent paths

Creating a State Machine Diagram: Steps (2 of 2)

6. **Look** for additional transitions and **test** both directions
7. **Expand** each transition with appropriate message event, guard condition, and action expression
8. **Review** and **test** the state machine diagram for the class
   - Make sure the names of the state is really a state for the object in the class
   - Follow the life cycle of an object coming into existence and being deleted
   - Be sure the diagram covers all exception condition
   - Look again for concurrent paths and composite states

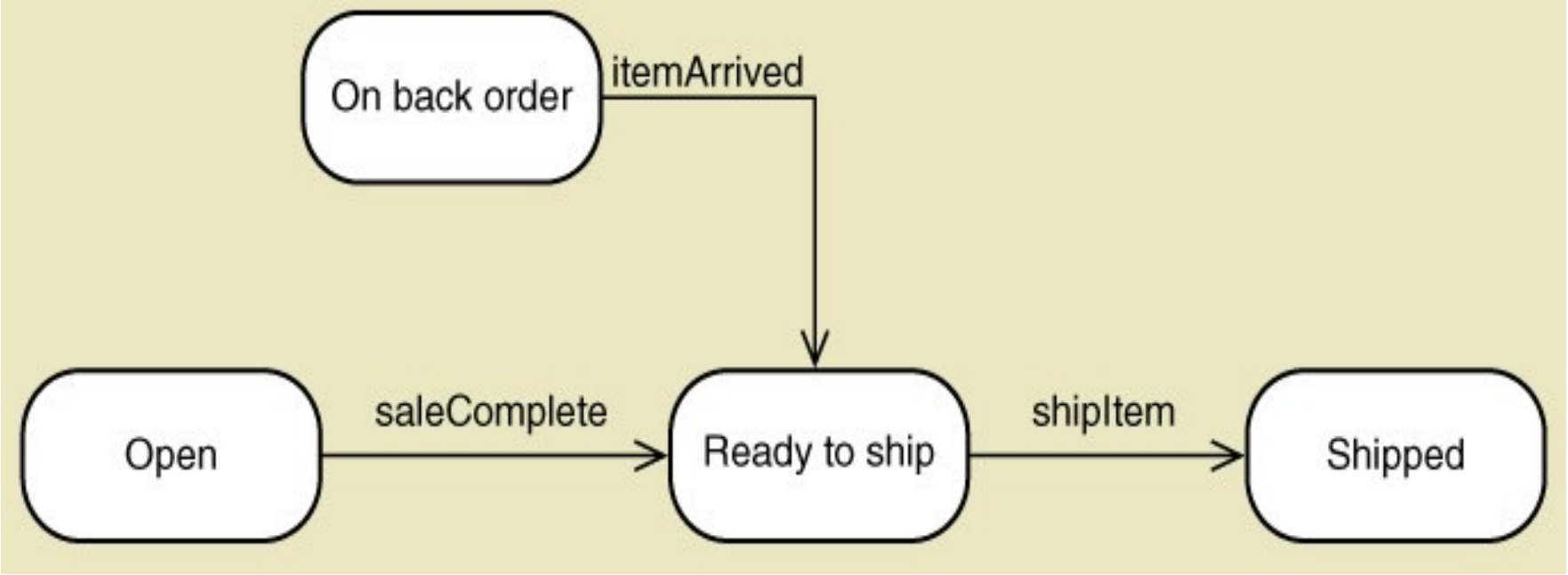
RMO – Creating a State Machine Diagram: Steps – SaleItem (1 of 3)

1. Choose SaleItem. It has status conditions that need to be tracked
2. List the states and exit transitions

State	Transition causing exit
Open	saleComplete
Ready to Ship	shipItem
On back order	itemArrived
Shipped	No exit transition defined

R M O – Creating a State Machine Diagram: Steps – SaleItem (2 of 3)

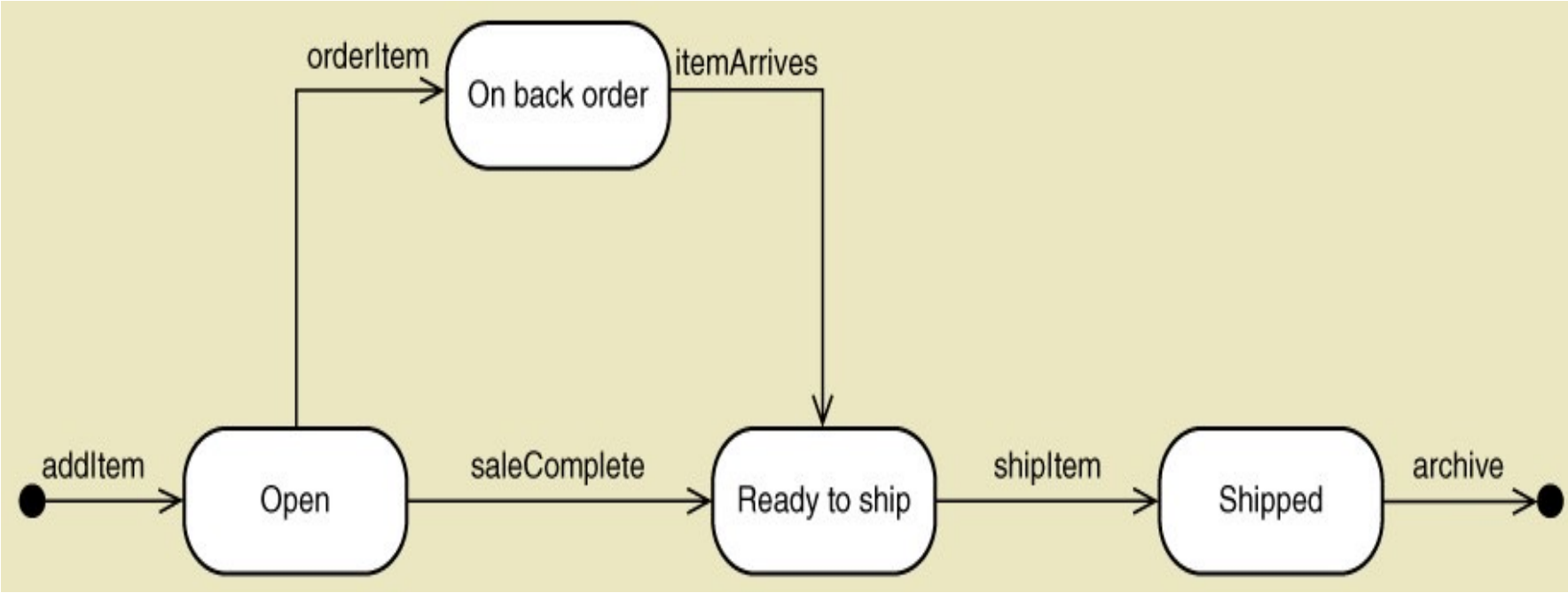
3. Build fragments – see figure below
4. Sequence in correct order – see figure below
5. Look for concurrent paths – none



R M O – Creating a State Machine Diagram: Steps – SaleItem (3 of 3)

6. Add other required transitions
7. Expand with guard, action-expressions etc.
8. Review and test

Below is the final State Machine Diagram



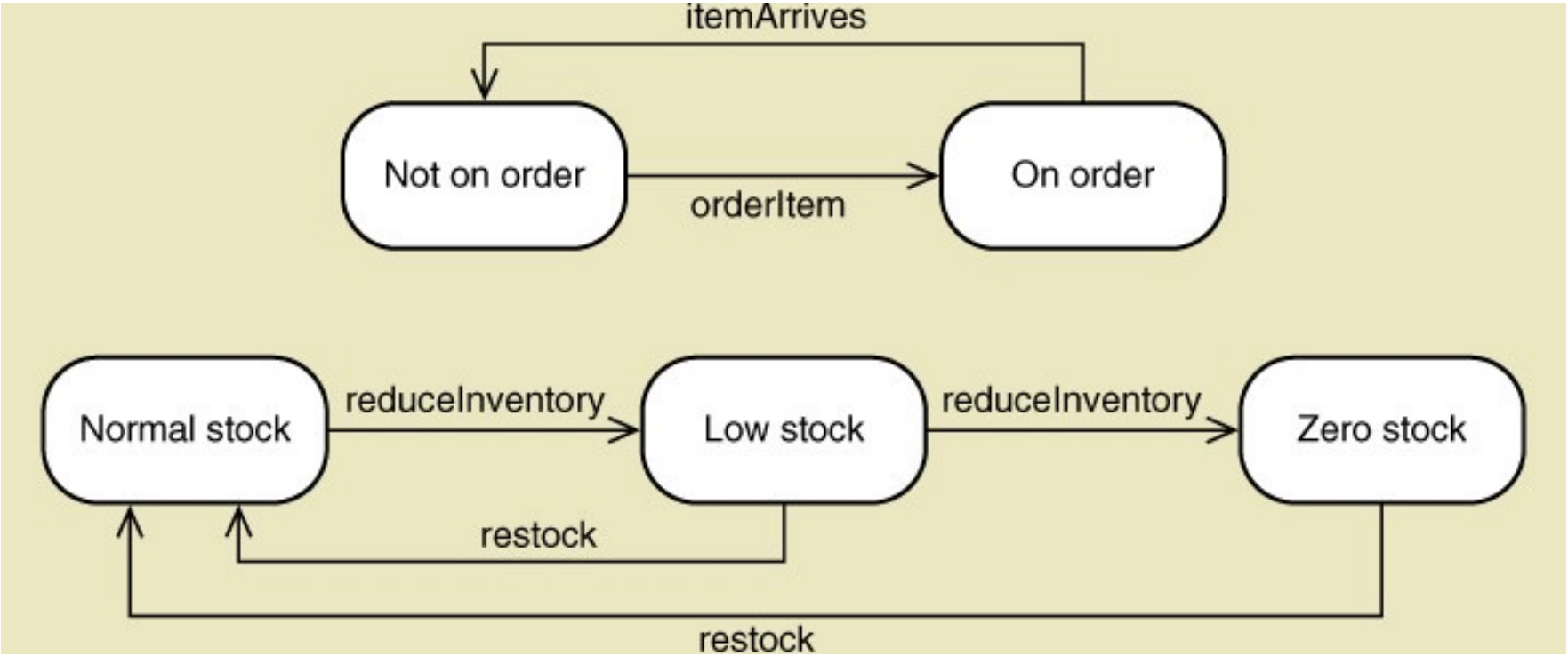
R M O – Creating a State Machine Diagram: Steps – InventoryItem (1 of 3)

1. Choose InventoryItem. It has status conditions that need to be tracked
2. List the states and exit transitions

State	Transition causing exit
Normal stock	reduceInventory
Low stock	reduceInventory OR restock
Zero stock	removeItem OR restock
On order	itemArrives
Not on order	orderItem

R M O – Creating a State Machine Diagram: Steps – InventoryItem (2 of 3)

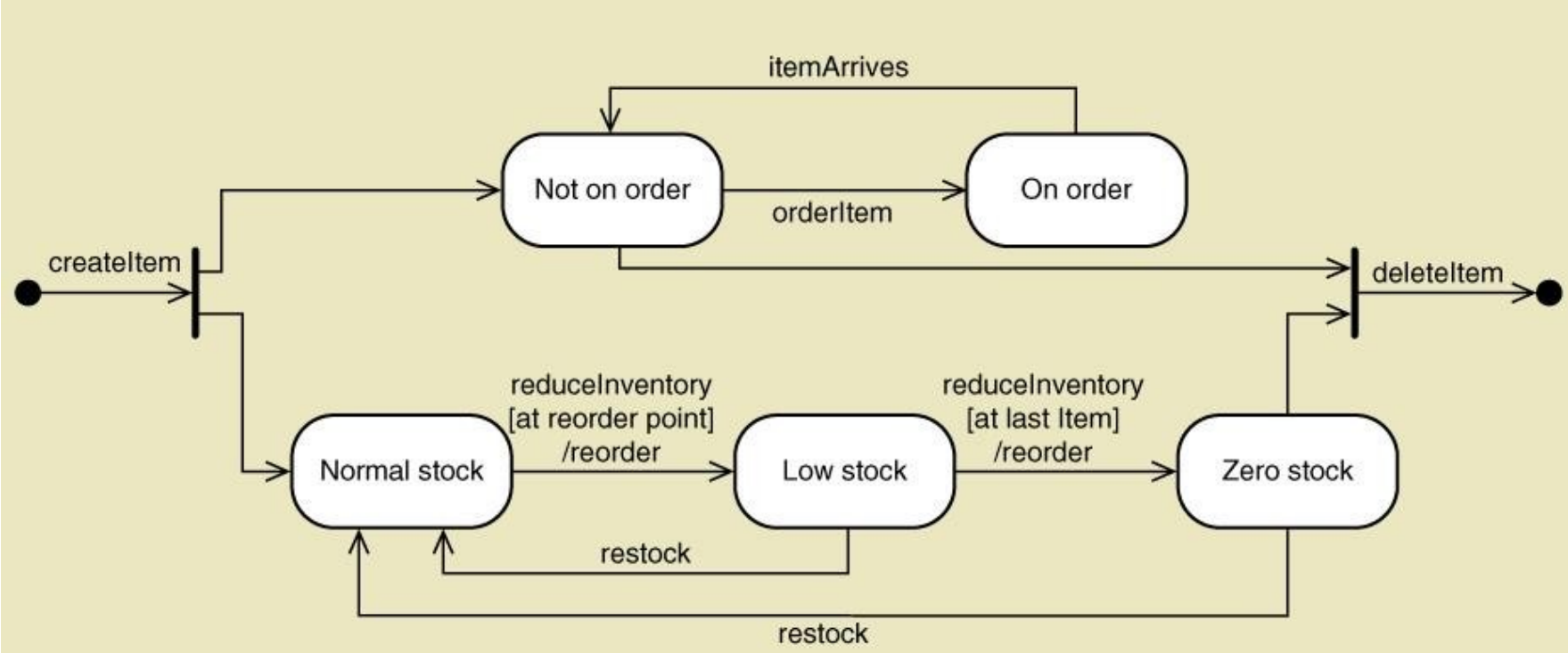
3. Build fragments – see figure below
4. Sequence in correct order – see figure below
5. Look for concurrent paths – see figure below



R M O – Creating a State Machine Diagram: Steps – InventoryItem (3 of 3)

6. Add other required transitions
7. Expand with guard, action-expressions etc.
8. Review and test

Below is the final State Machine Diagram



Summary (1 of 4)

- This chapter focuses on modeling functional requirements as a part of systems analysis
- “Things” in the problem domain are identified and modeled, called domain classes or data entities
- Two techniques for identifying domain classes/data entities are the brainstorming technique and the noun technique
- Domain classes have attributes and associations
- Associations are naturally occurring relationships among classes, and associations have minimum and maximum multiplicity

Summary (2 of 4)

- Entity-relationship diagrams (ERDs) show the information about data entities
- ERDs are often preferred by database analysts and are widely used
- ERDs are not UML diagrams, and an association is called a relationship, multiplicity is called cardinality, and generalization/specialization (inheritance) and whole part relationships are usually not shown

Summary (3 of 4)

- The UML class diagram notation is used to create a domain model class diagram for a system. The domain model classes do not have methods because they are not yet software classes.
- There are actually three UML class diagram relationships: association relationships, generalization/specialization (inheritance) relationships, and whole part relationships
- Other class diagram concepts are abstract versus concrete classes, compound attributes, composition and aggregation, association classes, super classes and subclasses

Summary (4 of 4)

- Some objects have a life cycle with status conditions that change and should be tracked
- A State Machine Diagram tracks the behavior of these objects with states and transitions
- To develop a State Machine Diagram
  - Choose a single object class.
  - Identify the states and exit transitions
  - Identify concurrent paths
  - Identify additional paths
  - Build the State Machine Diagram

Course Code: IS332
Course: Analysis and Design of Information Systems

Thank you

s.zahran@fci-cu.edu.eg