

SEVENTH EDITION

# Systems Analysis AND Design

**IN A CHANGING WORLD**

## Chapter 4

JOHN SATZINGER

ROBERT JACKSON

STEPHEN BURD

# Domain Modeling

٤ فصل

Systems Analysis and Design in a  
Changing World 7<sup>th</sup> Ed  
Satzinger, Jackson & Burd

# فصل ۴: چشم انداز

- «اشیاء» (Things) در دامنه مسئله
- نمودار موجودیت-رابطه (Entity-Relationship)
- نمودار کلاس مدل دامنه (The Domain Model Class)
- نمودار ماشین حالت (State Machine Diagram) – تشخیص رفتار اشیا

# انگیزه‌های یادگیری

- شرح اینکه مفهوم «اشیاء» در دامنه مسئله چگونه می‌تواند نیازمندی‌ها را نیز توصیف کند
- تشخیص و تحلیل نهادهای داده (data entities) و کلاس‌های دامنه (domain classes) مورد نیاز در سیستم
- 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

# نگاه اجمالی

- این فصل بر روی یکی دیگر از مفاهیم کلیدی تعریف نیازمندی‌ها تمرکز می‌کند - نهادهای داده یا کلاس‌های دامنه
- برای سیستم نمایشگاه تجاری RMO در فصل ۱، برخی از کلاس‌های دامنه عبارتند از: **Supplier** (تامین‌کننده)، **Product** (محصول) و **Contact** (تماس)

## اشیاء در دامنه مسئله

- دامنه مسئله - بخش خاصی از نیازهای تجاری کاربران که در حوزه‌ی سیستم جدید است.
- «اشیاء» (things) مواردی هستند که کاربر در هنگام انجام وظایف با آنها کار می‌کند و لازم است که حفظ و نگهداری شوند.
- کالاهای، مشتری‌ها، پرداخت‌ها، خریدها و ... مثال‌هایی از اشیاء می‌باشند.
- این «اشیاء» نهایتاً به صورت کلاس‌های دامنه مدل می‌شوند.

## دو تکنیک برای یافتن اشیاء در دامنه‌ی مسئله

### تکنیک طوفان فکری

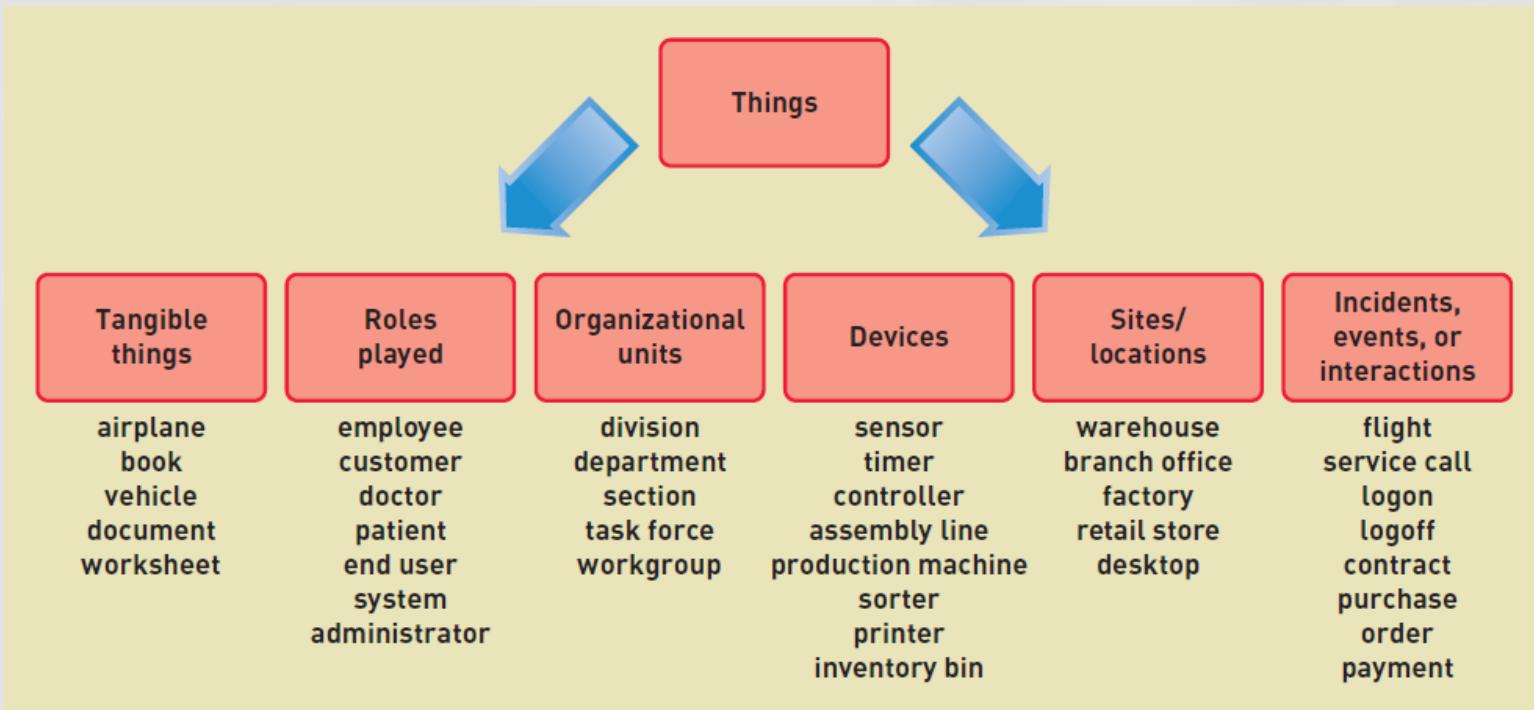
چک لیستی از تمامی انواع اشیائی که در سیستم وجود دارند را تهیه کرده و با استفاده از طوفان فکری کلاس‌های دامنه‌ی هر یک از این نوع‌ها را تشخیص دهید.

### تکنیک اسم

هر یک از اسامی که در هنگام توصیف سیستم ذکر می‌شوند را بررسی نمایید که آیا کلاس دامنه هستند یا خیر (یک ویژگی یا چیزی که اصلاً نیاز به یاد داشتن یا ذخیره ندارد)

# تکنیک طوفان فکری

در مسئله‌مان آیا اشیاء قابل لمس وجود دارند؟ آیا نقشهایی که توسط اشخاص انجام می‌شود وجود دارد؟ آیا واحدهای سازمانی وجود دارند؟ محلها یا مکان‌ها وجود دارند؟ آیا حوادث و رخدادهایی که نیاز به ثبت دارند وجود دارد؟ و ...



# تکنیک طوفان فکری: قدم‌ها (۱ از ۲)

1. یک کاربر و مجموعه‌ای از موارد کاربرد را تشخیص بده
2. با کاربر طوفان فکری انجام بده تا اشیاء به کار رفته در هنگام انجام موارد کاربرد را تشخیص دهید- به عبارتی بهتر اشیائی را که می‌بایست اطلاعاتشان در سیستم حفظ شوند
3. از دسته‌بندی اشیاء استفاده کنید تا بتوانید به طور سیستماتیک در مورد اشیاء بالقوه سوال بپرسید: آیا اشیاء قابل لمسی که نیاز به ذخیره‌ی اطلاعات داشته باشیم وجود دارد؟ آیا مکان‌ها در نظر گرفته می‌شوند؟ آیا نقشه‌ایی توسط اشخاص (مشتری، ادمین، مدیر و ...) انجام می‌شود که نیاز به ذخیره‌سازی داشته باشند؟

## تکنیک طوفان فکری: قدم‌ها (۲ از ۲)

4. کار با تمامی کاربران و ذی‌نفعان را ادامه دهید تا بتوانید لیست طوفان فکری را گسترش داده و کامل نمایید.
5. نتایج را ادغام نمایید، موارد تکراری را حذف کنید و نهایتاً لیست اولیه‌ی اشیاء را بدست بیاورید.

## تکنیک اسم (۱ از ۲)

- تکنیکی برای تشخیص کلاس‌های دامنه‌ی مسئله (اشیاء) با :  
یافتن، دسته‌بندی و تصحیح لیستی از اسامی که در مستندات و مباحثه‌ها استفاده می‌شوند.
- تکنیکی متداول و اصولی  
نهایتا به یک لیست طولانی از اسامی میرسیم که تعداد زیادی از این اسامی شیء نمی‌باشند چون نیاز به ذخیره توسط سیستم ندارند.

## تکنیک اسام (۲ از ۲)

- مشکل در تشخیص اسامی یکسان و متراff و همچنین اشیائی که در حقیقت ویژگی (attribute) هستند نه اسم.
- روش مناسبی برای شروع در مواقعي که هیچ کاربری در دسترس وجود ندارد که بتوان طوفان فکری انجام داد.

## تکنیک اسم: قدم‌ها (۱ از ۳)

۱. با استفاده از موارد کاربرد، اکتورها (کنش‌گرها) و سایر اطلاعات در مورد سیستم- شامل ورودی و خروجی- تمامی اسم‌ها را تشخیص بدهید.

برای پروژه‌ی RMO CSMS اسامی ممکن است شامل مشتری، کالا، فروش، تاییدیه، تراکنش، ارسال، بانک، تقاضای تعویض، خلاصه گزارش، مدیریت، گزارش تراکنش، حسابداری، سفارش دارای تاخیر، برگشت کالا، تایید برگشت و ... باشد.

۲. با استفاده از اطلاعات سیستم‌های موجود، رویه‌های در حال استفاده و فرم‌ها یا گزارش‌های در حال استفاده؛ موارد یا دسته‌بندی‌های اطلاعاتی مورد نیاز را اضافه نمایید.

برای پروژه‌ی RMO CSMS این‌ها می‌تواند شامل قیمت، اندازه، رنگ، مدل، فصل، تعداد موجودی، روش پرداخت و آدرس ارسال باشد.

## تکنیک اسم: قدم‌ها (۲ از ۳)

۳. در حالی که اسامی اضافه می‌شوند سعی کنید تا آن‌ها را پالایش بکنید.  
برای هر اسم جهت کمک به نگه داشتن اسم مربوطه سوال‌های زیر را از  
خوتان بپرسید:

آیا چیز یکتاویست که لازم است سیستم از آن اطلاع داشته باشد؟

آیا درون محدوده‌ی سیستمی که بر روی آن کار می‌کنیم هست یا خیر؟

آیا سیستم نیاز به ذخیره و یاد داشتن بیشتر از یکی از این موارد را دارد یا خیر؟

جهت تصمیم‌گیری برای حذف، سوالات زیر را بپرسید:

آیا این اسم در حقیقت مترادف شیء دیگریست که قبل از تشخیص داده شده است؟

آیا این اسم در حقیقت خروجی سیستم برای اطلاعات دیگریست که قبل از تشخیص داده شده است؟

آیا این اسم در حقیقت یک ورودی است که از ثبت و ضبط سایر اطلاعات تشخیص داده شده نتیجه می‌شوند؟

## تکنیک اسم: قدم‌ها (۳ از ۳)

### ۳. (ادامه)

جهت تصمیم‌گیری برای بررسی بیشتر سوالات زیر را بپرسید:

آیا ممکن است اسم مربوطه اطلاعات خاصی (attribute) در مورد یک شیء دیگری که قبلاً تشخیص داده شده باشد؟

آیا چیزیست که ممکن است در صورت تغییر فرضیات به آن نیاز داشته باشیم؟

۳. یک شاه-لیست از تمامی اسامی تشخیص داده شده تهیه کرده و سپس در مقابل هر اسم یادداشت کنید که اسم مربوطه می‌بایست اضافه، حذف و یا بررسی بیشتر بشود.

۴. لیست مربوطه را با کاربران، ذی‌نفعان و اعضای تیم بررسی نموده و سپس لیست اشیاء موجود در دامنه‌ی مسئله را تعیین نمایید.

# Partial List of Nouns for RMO (1 of 2)

- With notes on whether to include as domain class

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)

<b>Identified noun</b>	<b>Notes on including noun as a thing to store</b>
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.
Management	We know who they are. No need to store.
Marketing	We know who they are. No need to store.
Merchandising	We know who they are. No need to store.

# 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.

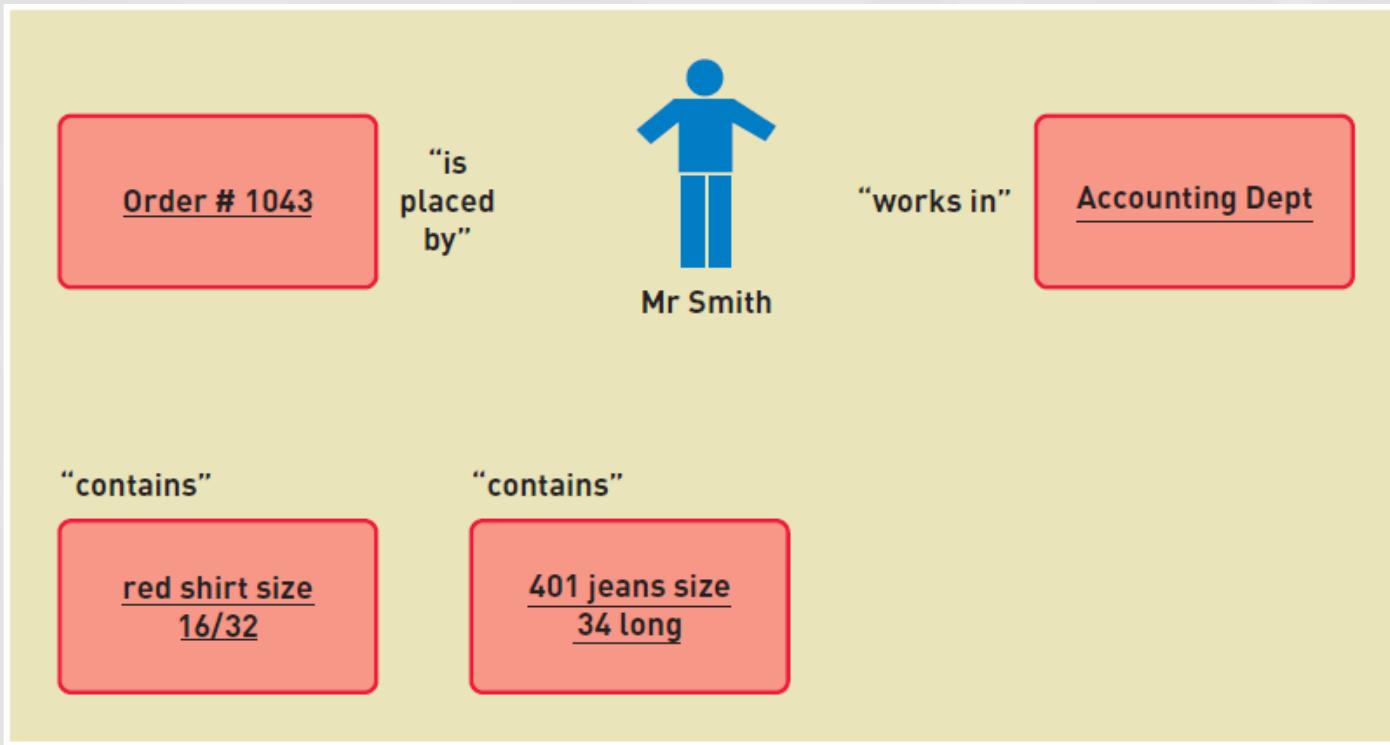
# Attributes and Values

- 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...

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

Mr. Jones has placed no order yet,  
but there might be many placed  
over time.

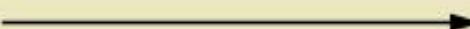
(Direction: Mr. Jones to Order)

A particular order is placed by Mr.  
Smith. There can't be an order  
without stating who the customer is.

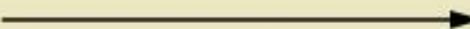
(Reverse direction: Order to Mr. Smith)

An order contains at least one item,  
but it could contain many items.

(Direction: Order to OrderItem)



multiplicity/cardinality  
is zero or more—  
optional relationship



multiplicity/cardinality  
is one and only one—  
mandatory relationship



multiplicity/cardinality  
is one or more—  
mandatory relationship

# Types of Associations

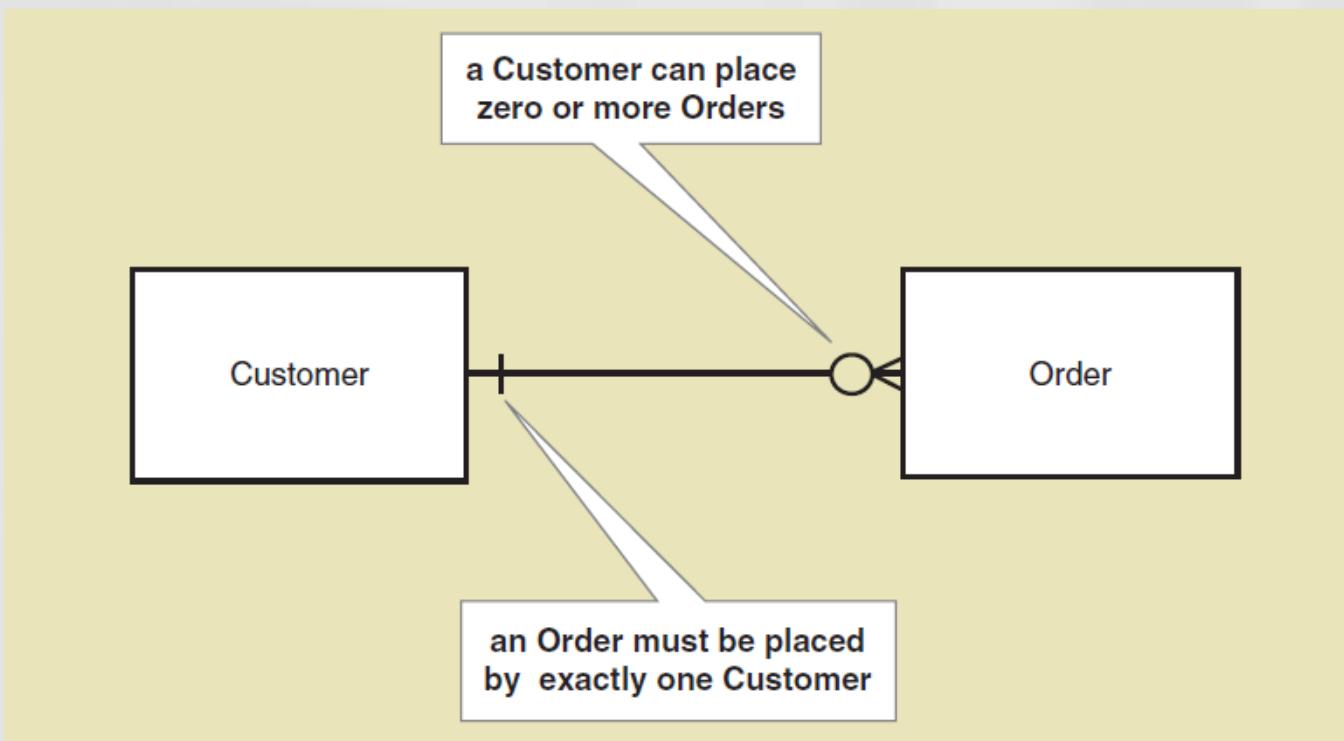
- Binary Association
  - Associations between exactly two different classes
    - Course Section includes Students
    - Members join Club
- Unary Association (recursive)
  - Associations between two instances of the same class
    - Person married to person
    - Part is made using parts
- Ternary Association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

# Entity-Relationship Diagrams: ERD

- ERDs have been used for many years to develop data models that are used in database development
- The term for “things” in ERD models is **data entities**
- ERD models are not UML models and do not use standard UML notation
- ERD models are not as expressive as UML models
  - They do not model generalization/specialization well
  - They do not model whole/part well

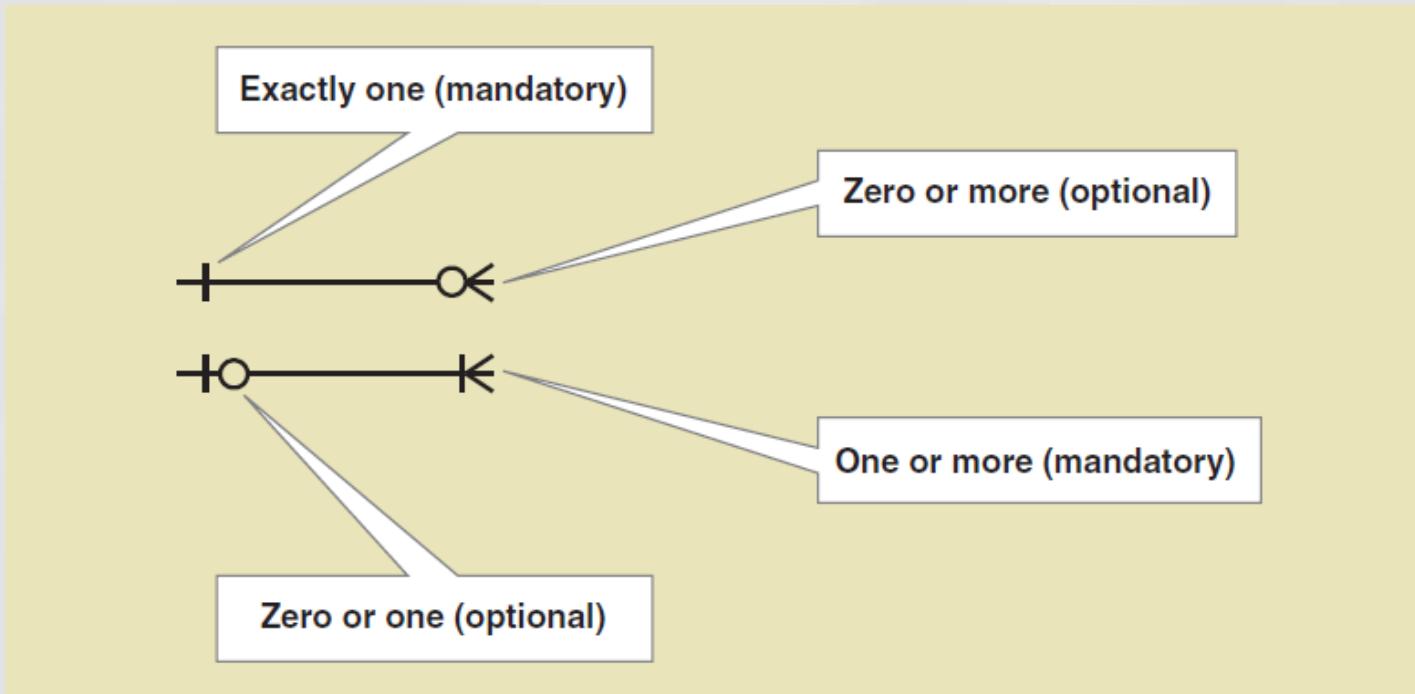
# Example of ERD Notation

- ERD Models normally use “crows feet” notation to show cardinality



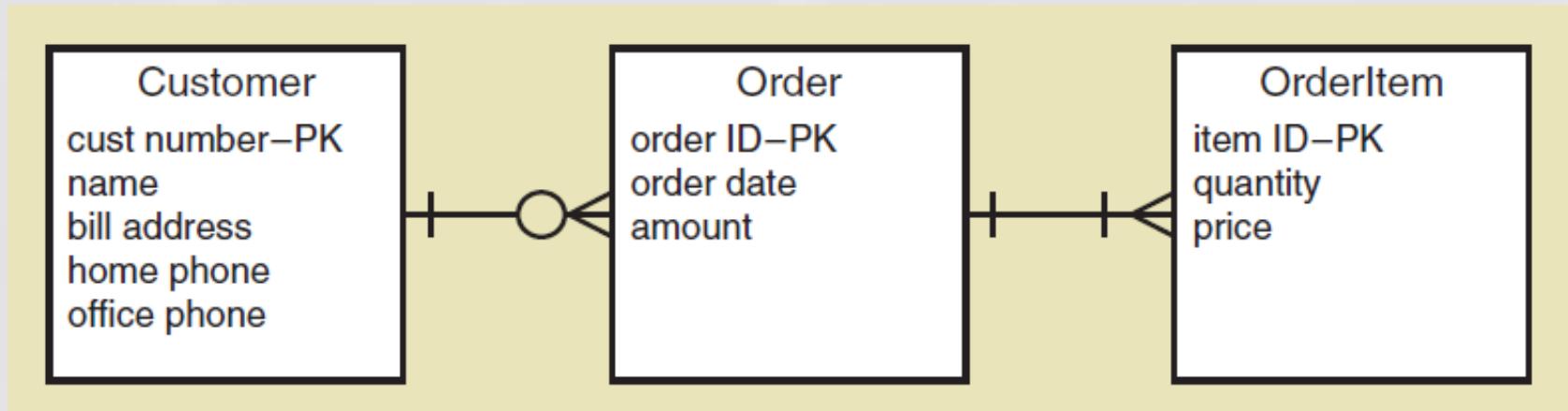
# ERD Cardinality Symbols

- Examples of crows feet notation for various cardinalities



# Expanded ERD with Attributes

- ERD with cardinalities and attributes
- There are several different notation methods for attributes in ERD models
- This notation places attributes within data entities

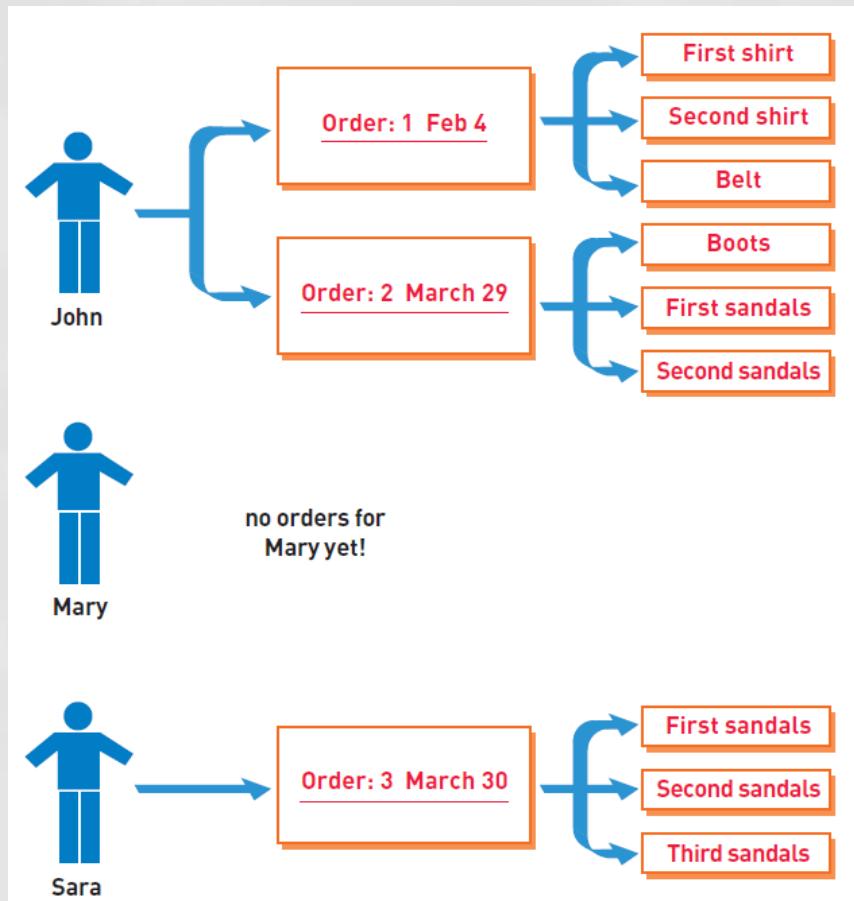


# Semantic Net (1 of 2)

- A **semantic net** is a graphical representation of an individual data entity and its relationship with other individual entities
- It is often used to help understand and then develop an ERD model

# Semantic Net (2 of 2)

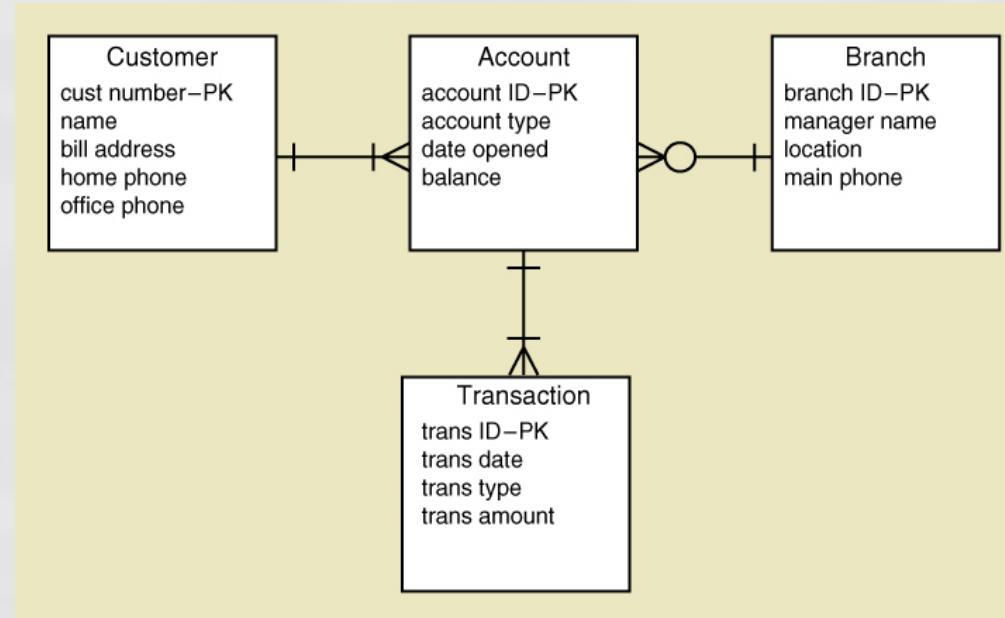
- This example shows three classes.
- Quick quiz
  - What are the classes?
  - How many relationships?
  - What are min and max cardinalities?
  - What type of relationships are they?



# An ERD for a Bank

## Quick Quiz

- What are the key fields?
- How many accounts can a customer have?
- How many branches can a customer be assigned to?
- How many customers can a branch have?

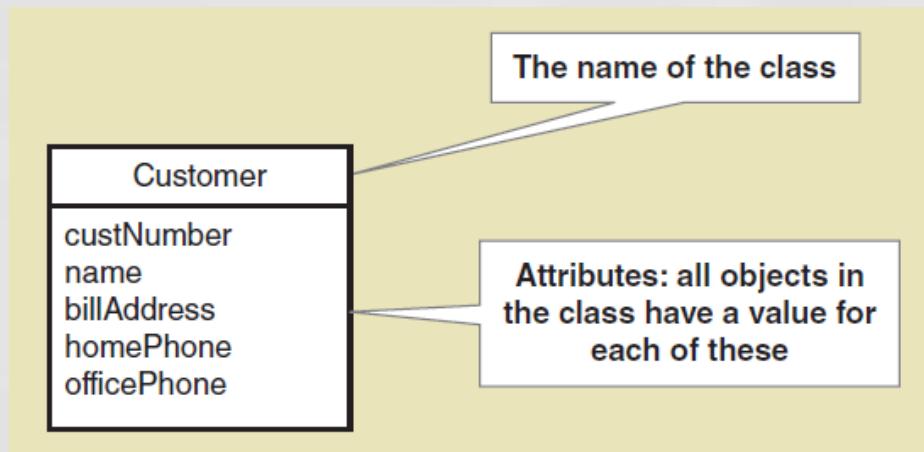


# The Domain Model Class Diagram

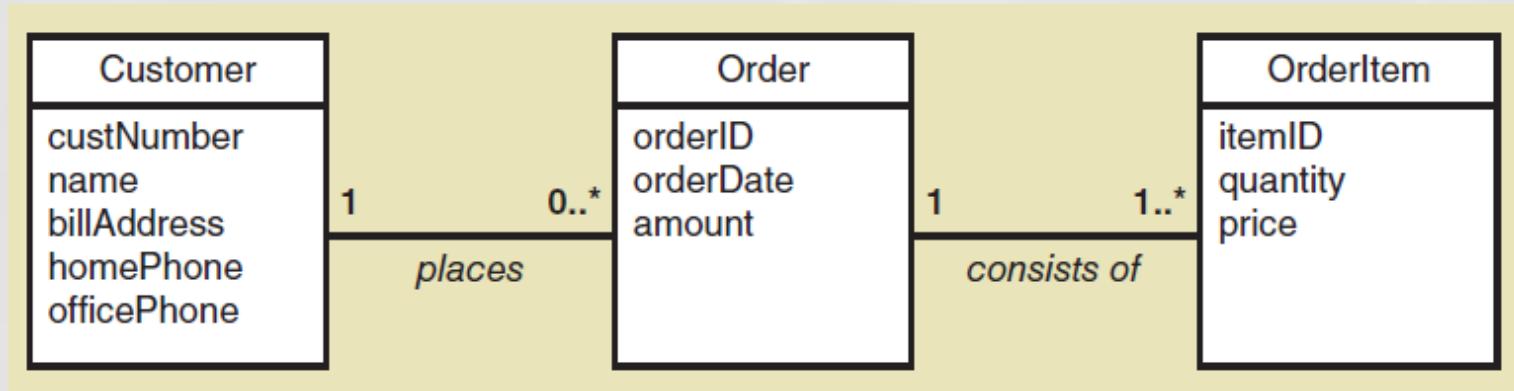
- Class
  - A type of classification used to describe a collection of objects
- Domain Class
  - Classes that describe objects in the problem domain
- Class Diagram
  - A UML diagram that shows classes with attributes and associations (plus methods if it models software classes)
- Domain Model Class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

# UML 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

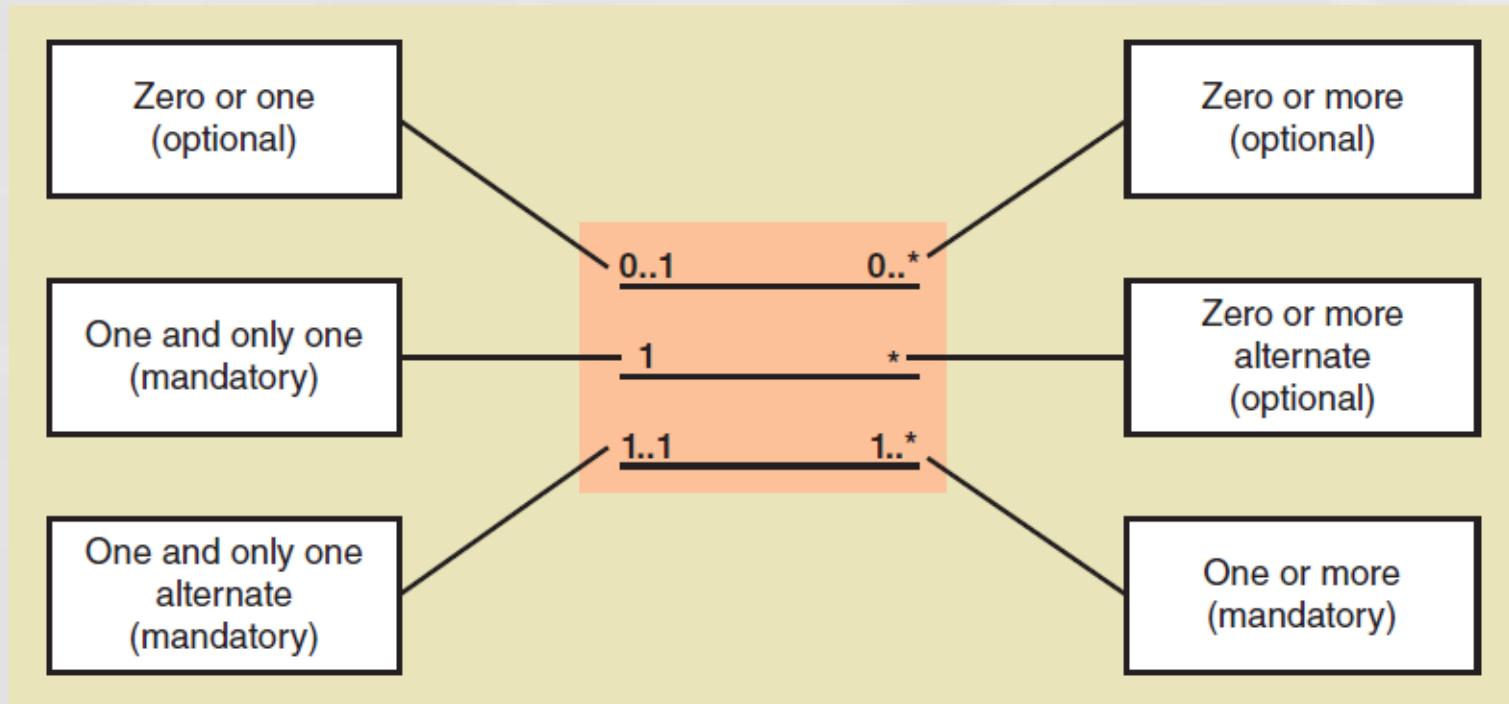


# A Simple Domain Model Class Diagram



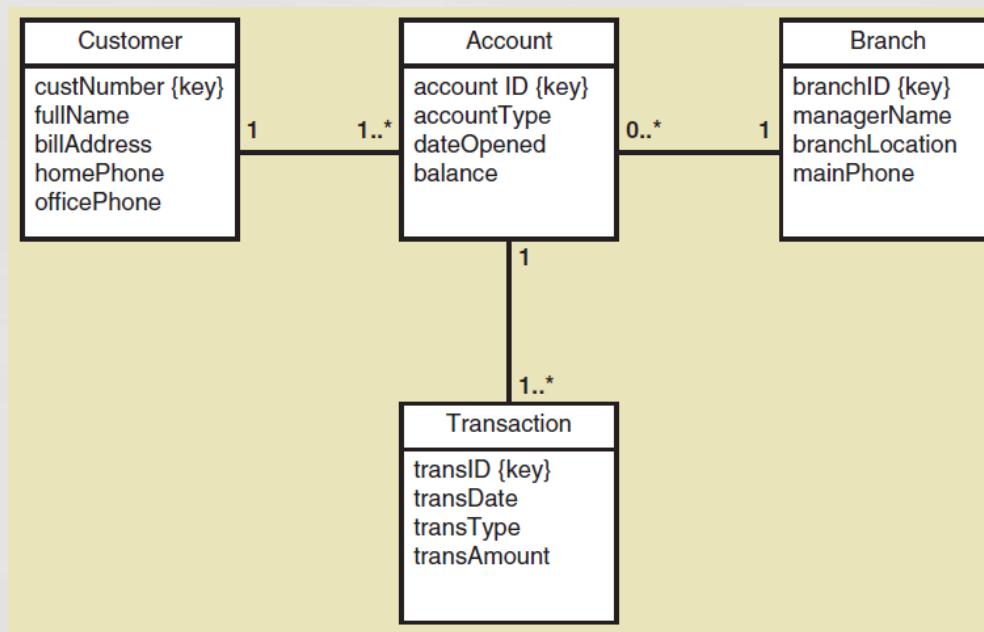
- Note: This diagram matches the semantic net shown previously
  - A customer places zero or more orders
  - An order is placed by exactly one customer
  - An order consists of one or more order items
  - An order item is part of exactly one order

# UML Notation for Multiplicity



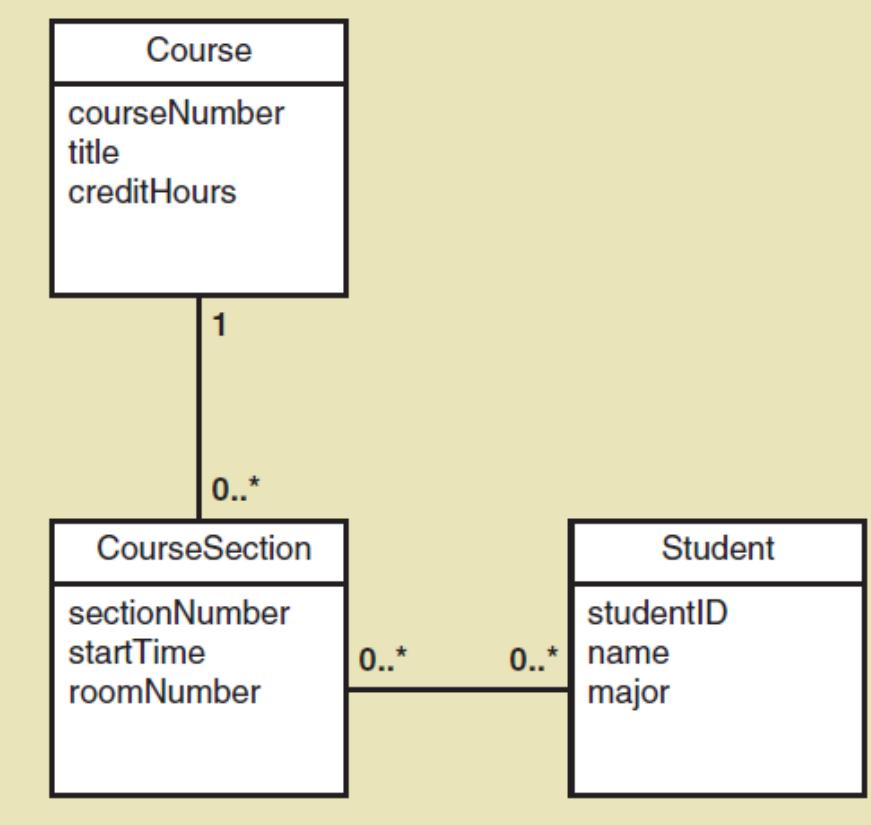
# Domain Model Class Diagram (1 of 2)

- Bank with many branches as show previously in ERD
  - Note notation for the key
  - Note the precise notation for the attributes (camelback)
  - Note the multiplicity notation



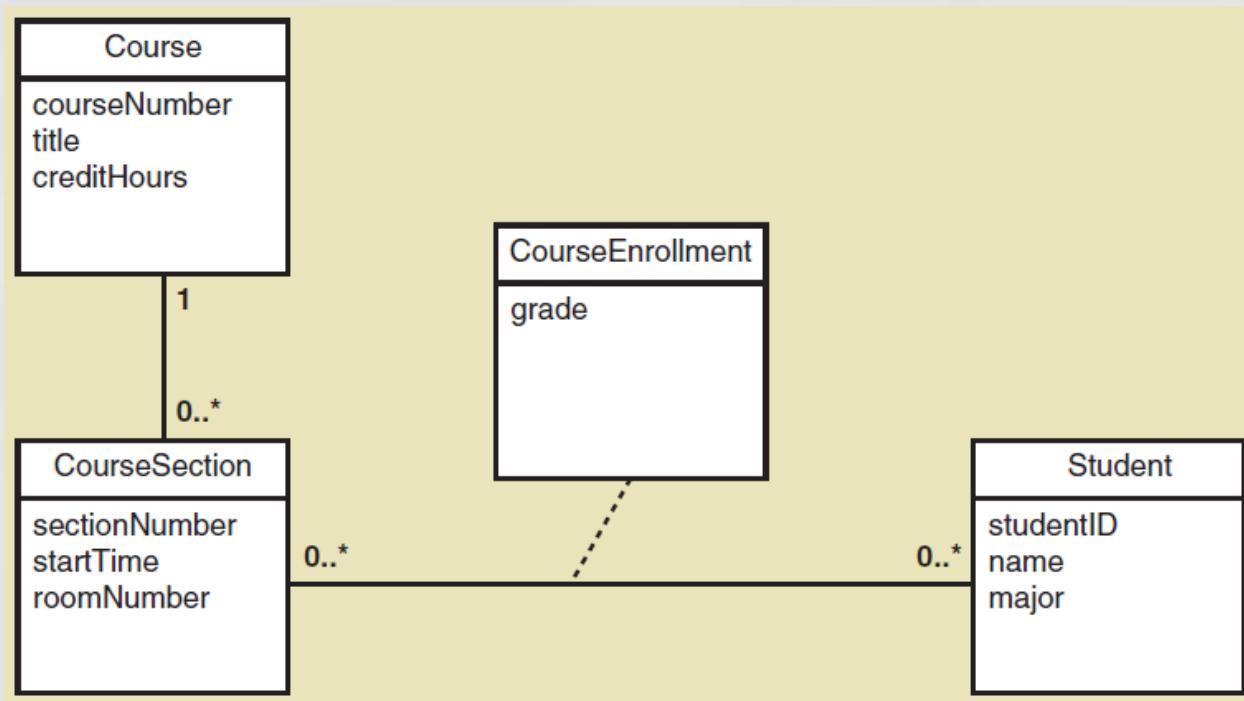
# Domain Model Class Diagram (2 of 2)

- 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)



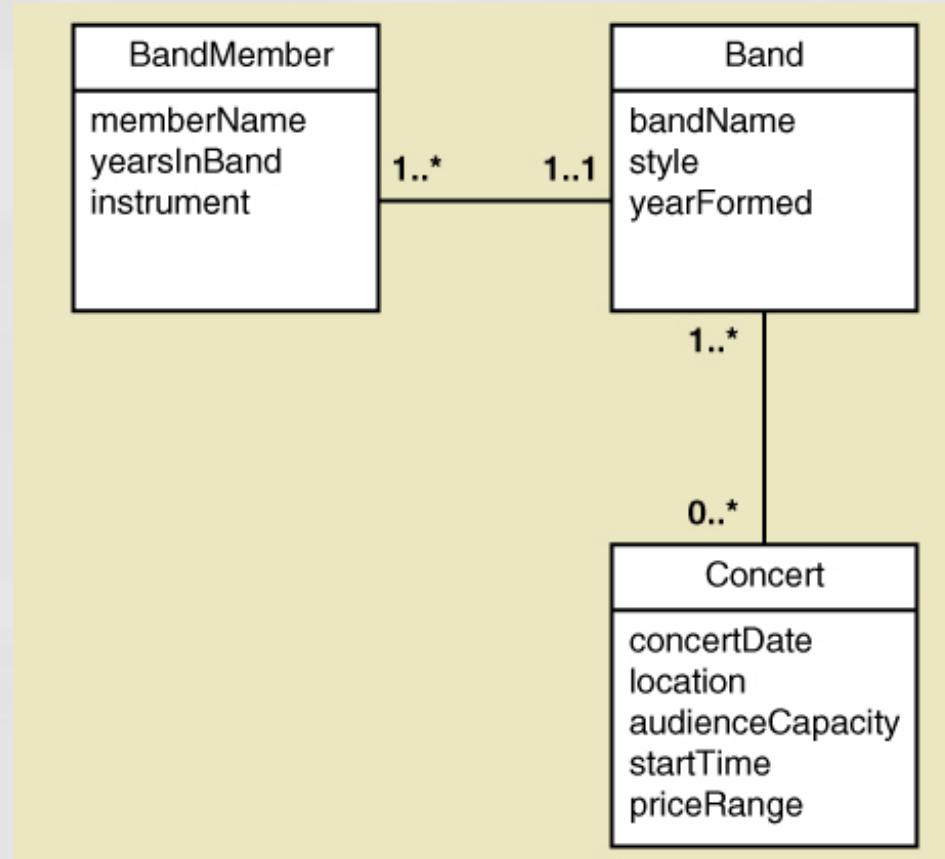
# Association Class Properties

- 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+StudentID
  - 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

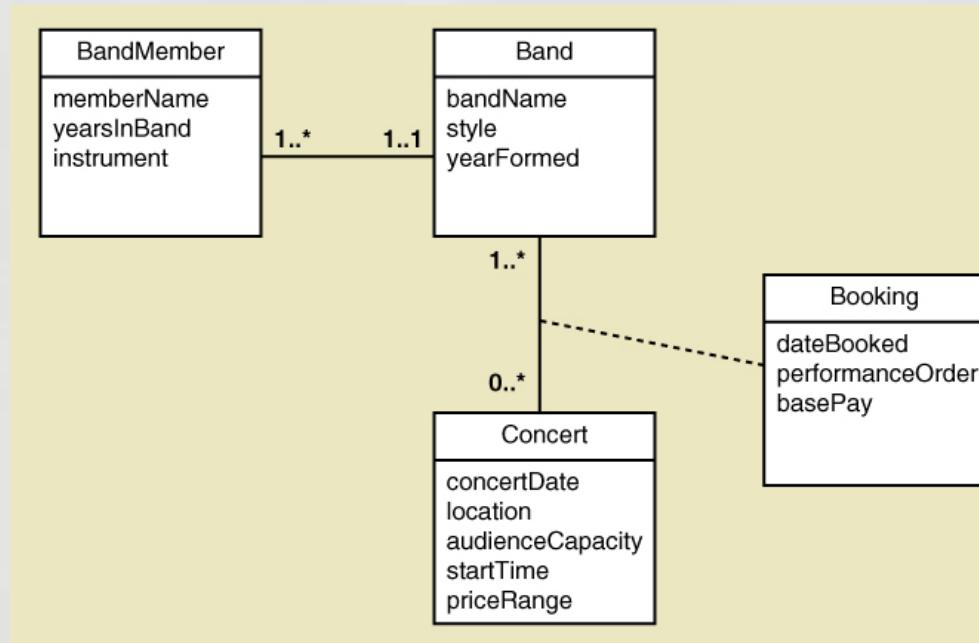
## Quick Quiz

- 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

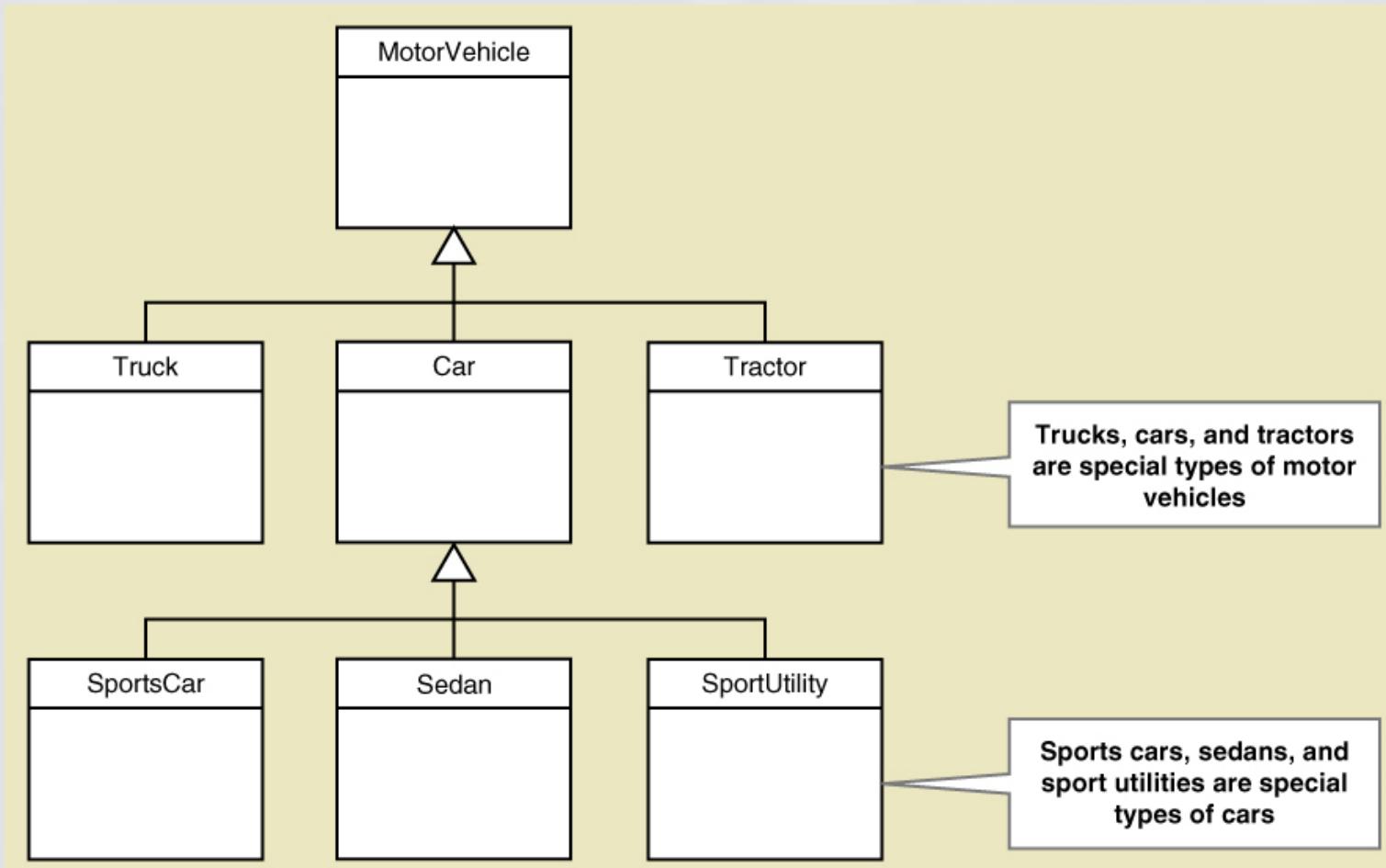
## ● Subclass

- the subordinate or more specialized class in a generalization/specialization hierarchy

## ● Inheritance

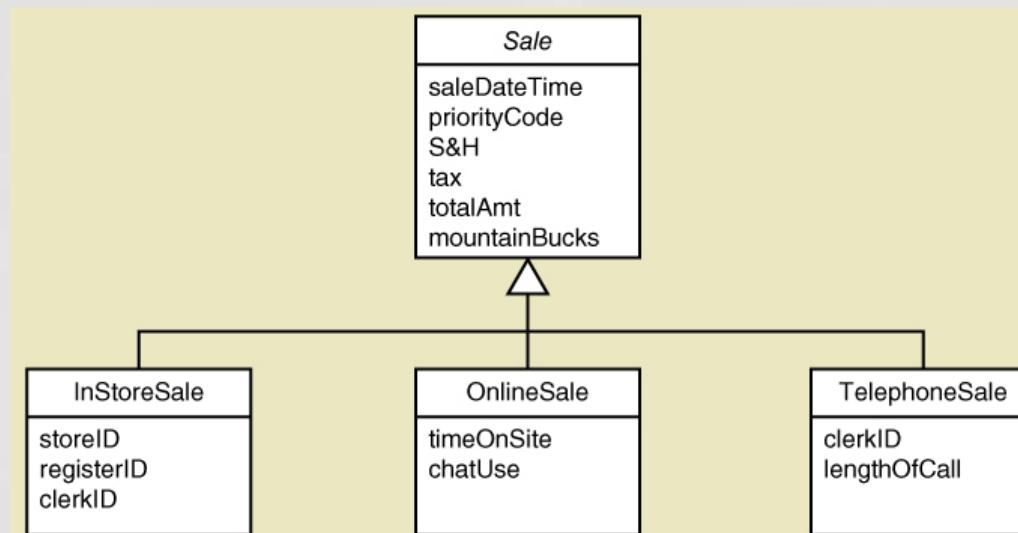
- the concept that subclasses inherit characteristics of the more general superclass

# Generalization/Specialization



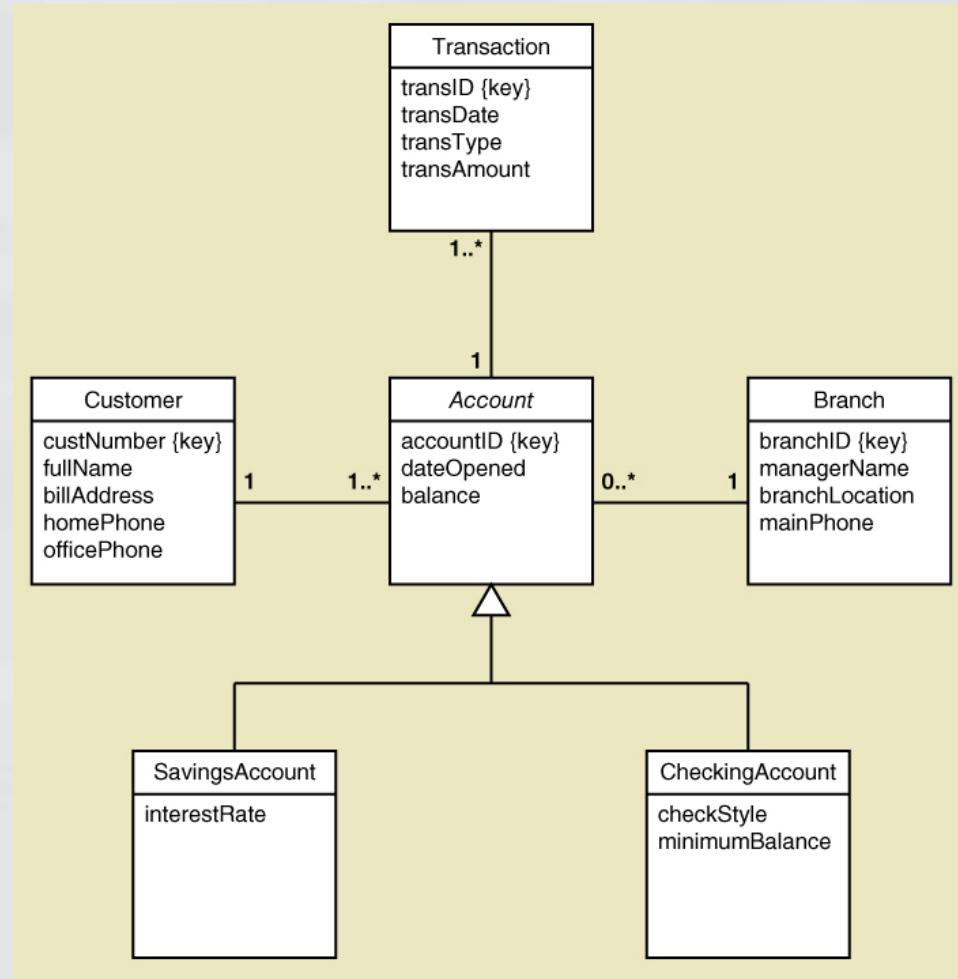
# Generalization/Specialization: Inheritance for RMO Three Types of Sales

- Abstract class— a class that allow 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

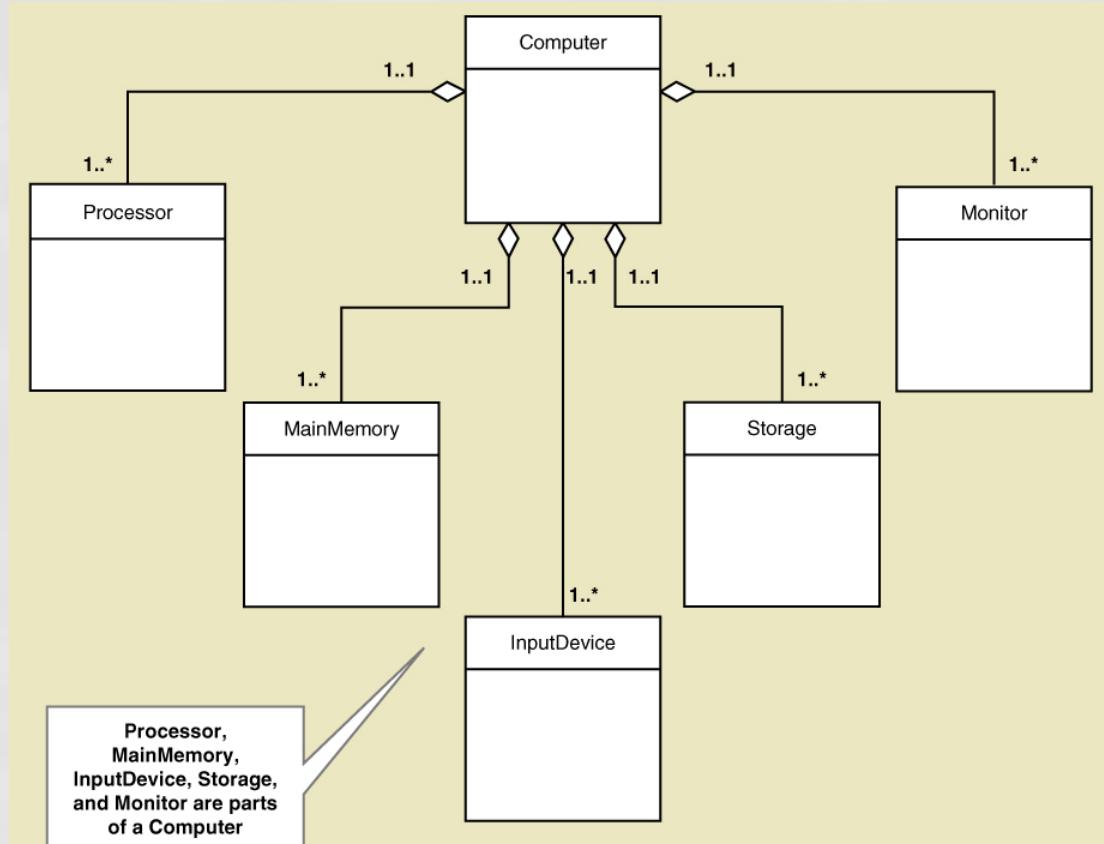


# More Complex Issues about Classes: Whole Part Relationships

- Whole-part relationship— a relationship between classes where one class is part of or a component portion of another class
- 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)
- 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 OrderItems)
  - Chip has circuits (without the chip, there are no circuits)

# Whole Part Relationships: Computer and its Parts

- Note: this is composition, with diamond symbol.
- Whole part can have multiplicity symbols, too (not shown)\



# More on UML Relationships

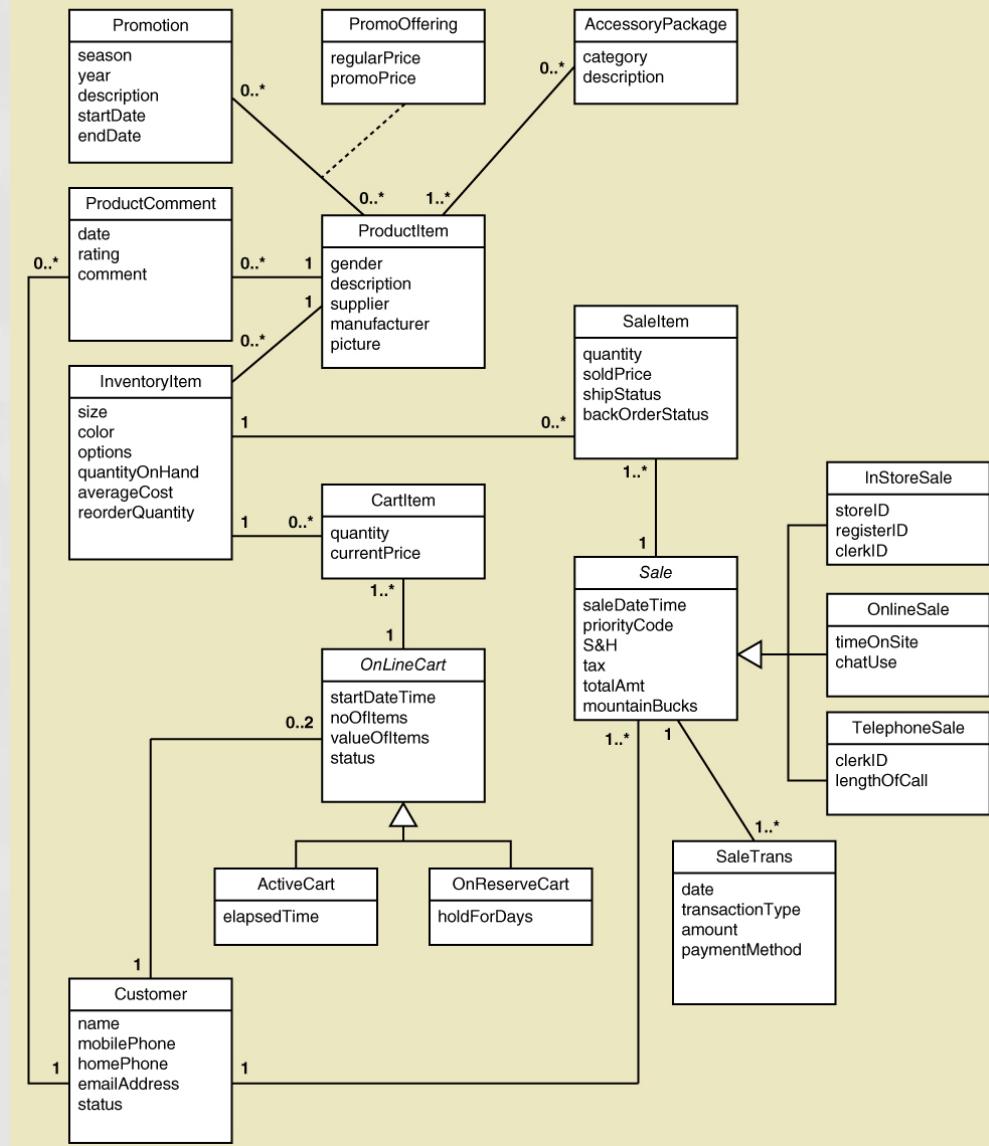
- There are actually three types of *relationships* in class diagrams
  - Association Relationships
    - These are associations discussed previously, just like ERD relationships
  - Whole Part Relationships
    - One class is a component or part of another class
  - Generalizations/Specialization Relationships
    - Inheritance
- Try not to confuse relationship with association

# RMO CSMS Project: Domain Model Class Diagrams

- There are several ways to create the domain model class diagram for a project
- RMO CSMS has 27 domain classes overall
- Can create one domain model class diagram per subsystem for those working on a subsystem
- Can create one overall domain model class diagram to provide an overview of the whole system
- Usually in early iterations, an initial draft of the domain model class diagram is completed and kept up to date. It is used to guide development.

# RMO CSMS

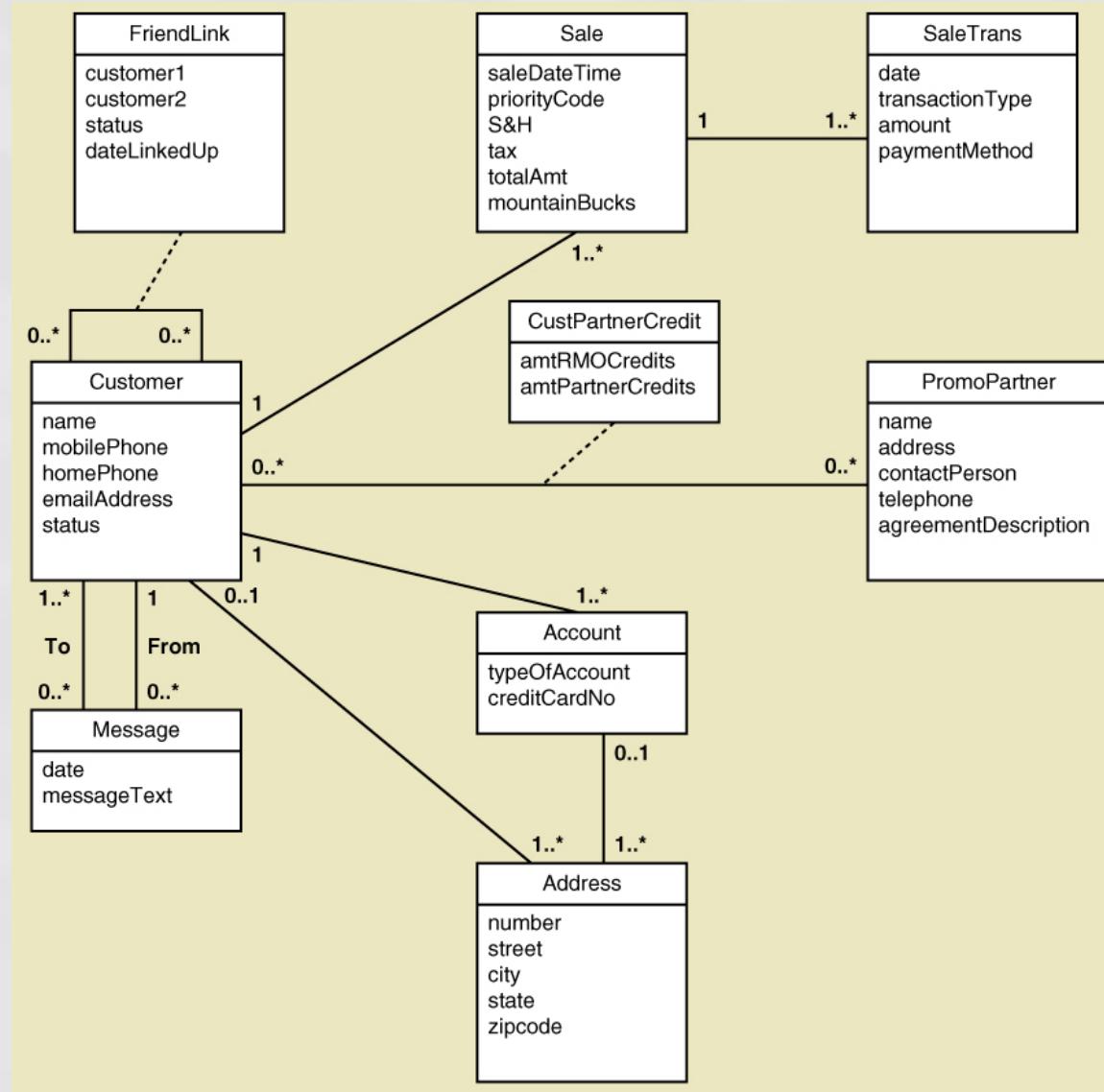
## Project: Sales Subsystem Domain Model Class Diagrams



# RMO CSMS

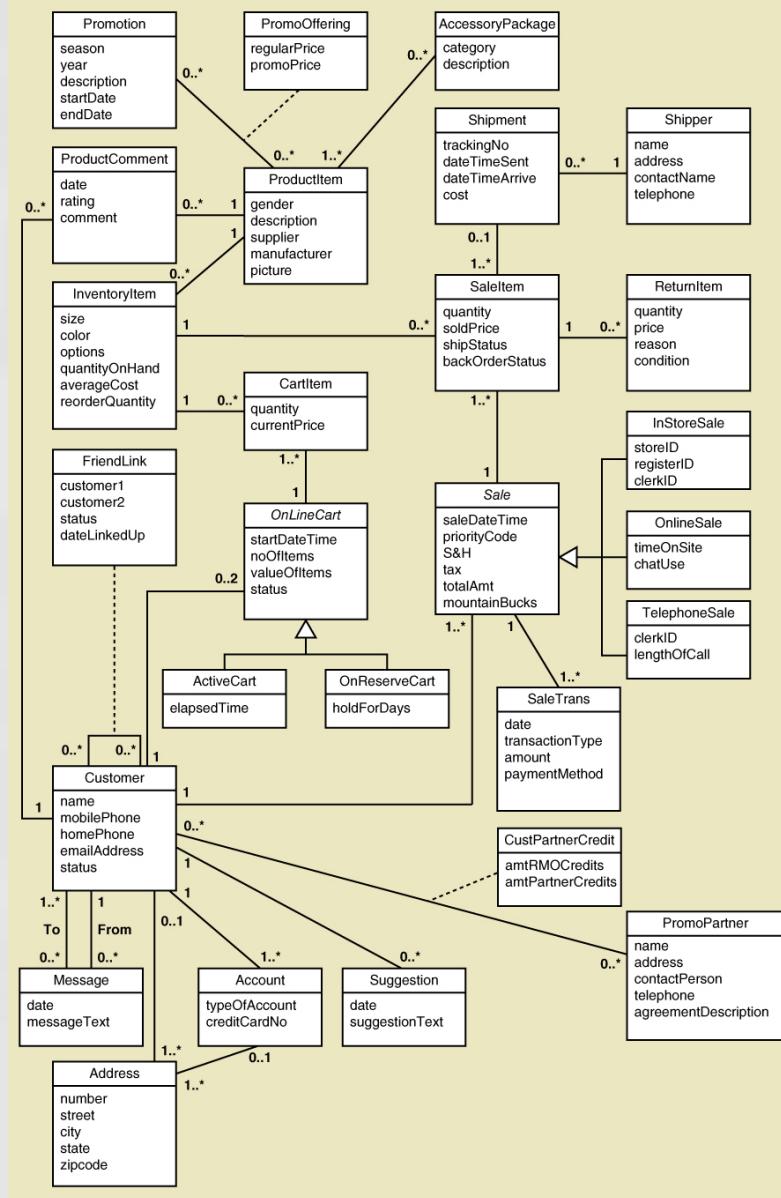
## Project: Customer Account Subsystem

### Domain Model Class Diagram



# RMO CSMS Project:

## Complete Domain Model Class Diagram



# RMO CSMS Project: Domain Model Class Diagrams (2 of 2)

- Given the complete RMO CSMS Domain Model Class Diagram and Sales and Customer Account subsystem examples:
  - Try completing the Order Fulfilment Subsystem Domain Model Class Diagram
  - Try Completing the Marketing Subsystem Domain Model Class Diagram
  - Try Completing the Reporting Subsystem Domain Model Class Diagram
- Review the use cases from Chapter 3 and decide what classes and associations from the complete model are required for each subsystem
  - Classes and associations might be duplicated in more than one subsystem model

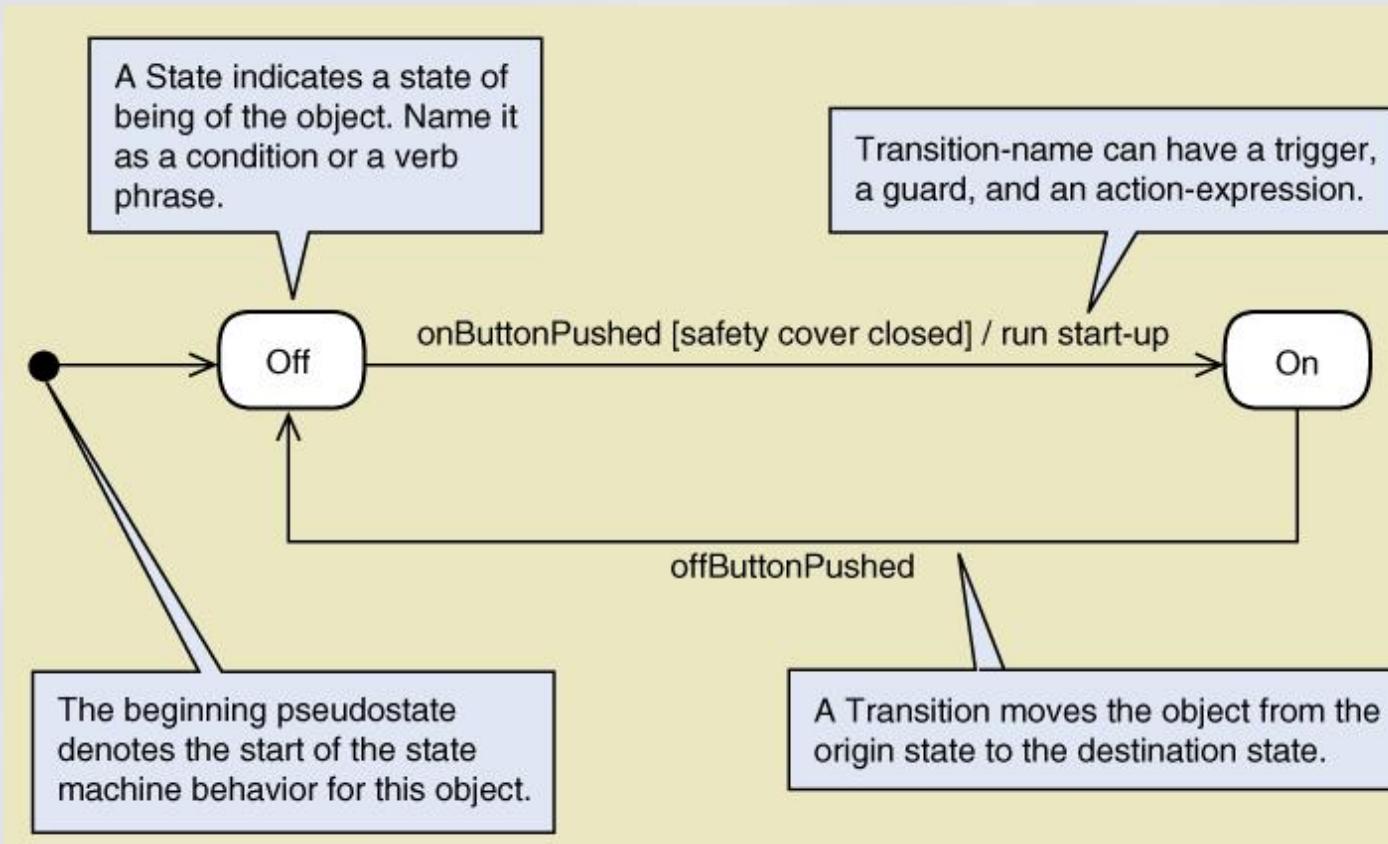
# Object Behavior – State Machine Diagram

- 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



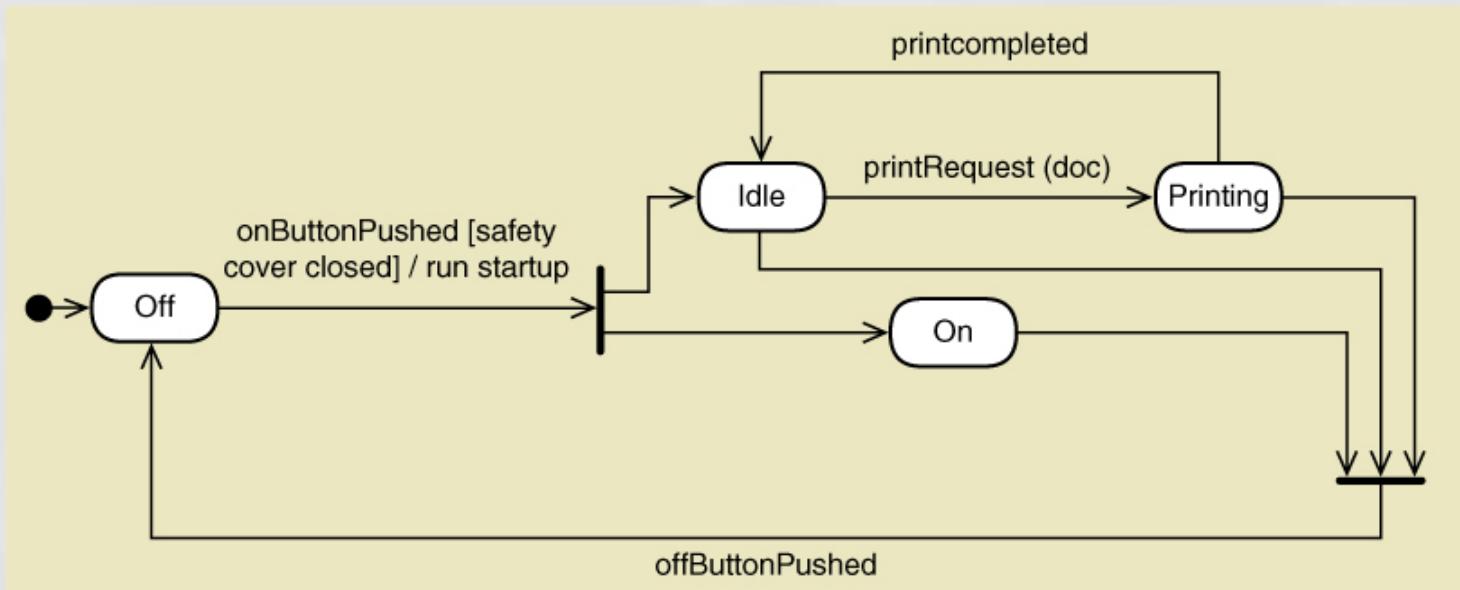
## • Syntax of transition statement

• *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 state are really 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

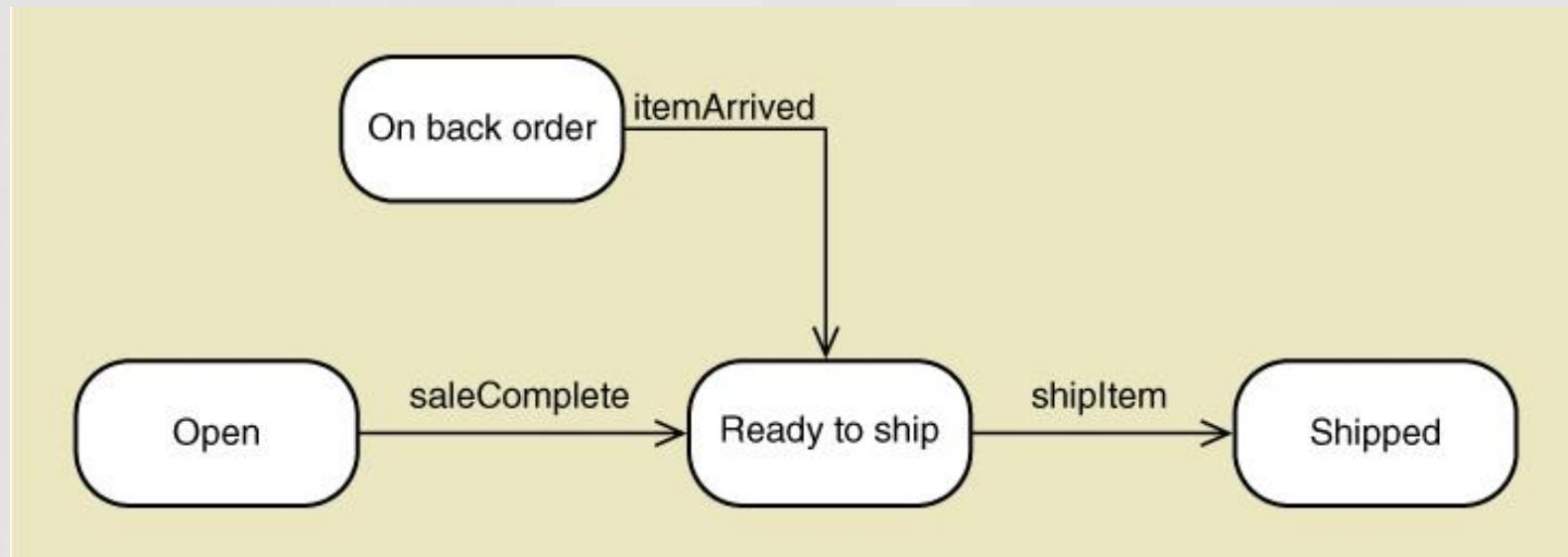
# RMO – Creating a State Machine Diagram: Steps – SaleItem (1 of 3)

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

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

# RMO – Creating a State Machine Diagram: Steps – SaleItem (2 of 3)

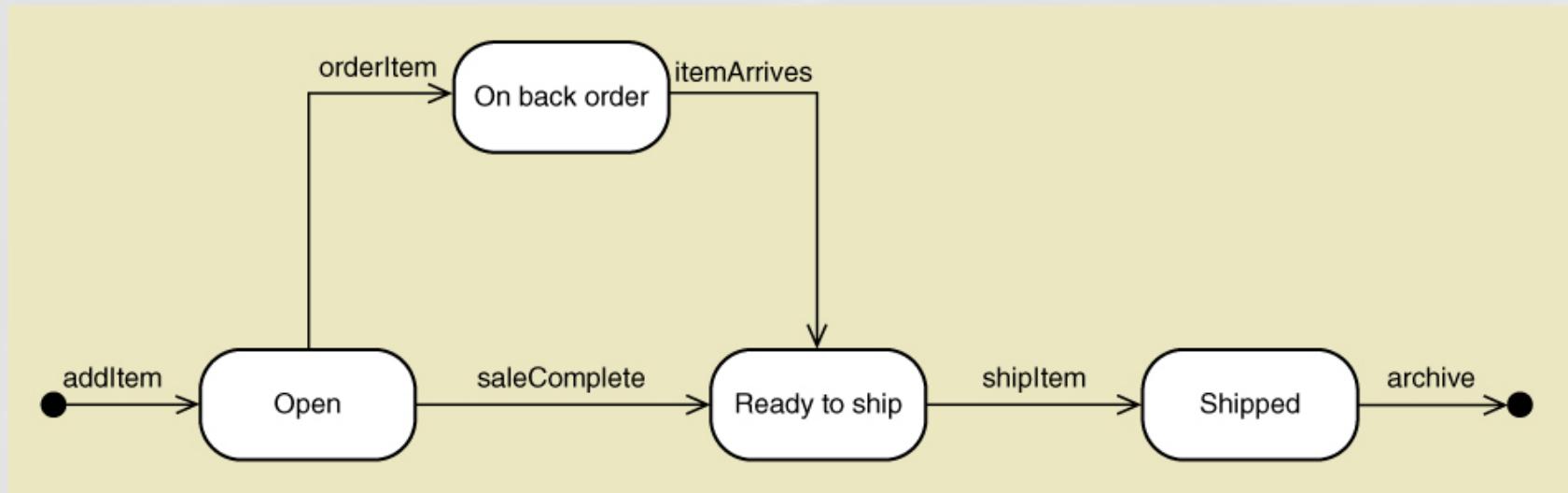
3. Build fragments – see figure below
4. Sequence in correct order – see figure below
5. Look for concurrent paths – none



# RMO – 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



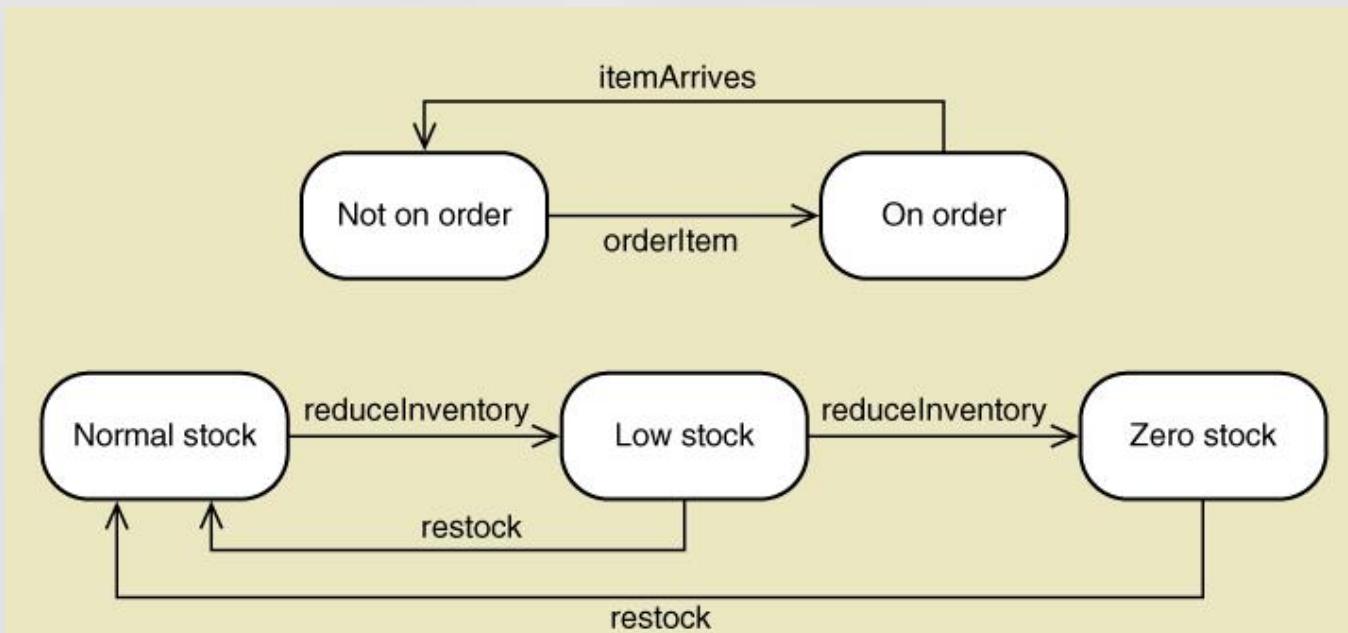
# RMO – Creating a State Machine Diagram: Steps – InventoryItem (1 of 3)

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

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

# RMO – Creating a State Machine Diagram: Steps – InventoryItem (2 of 3)

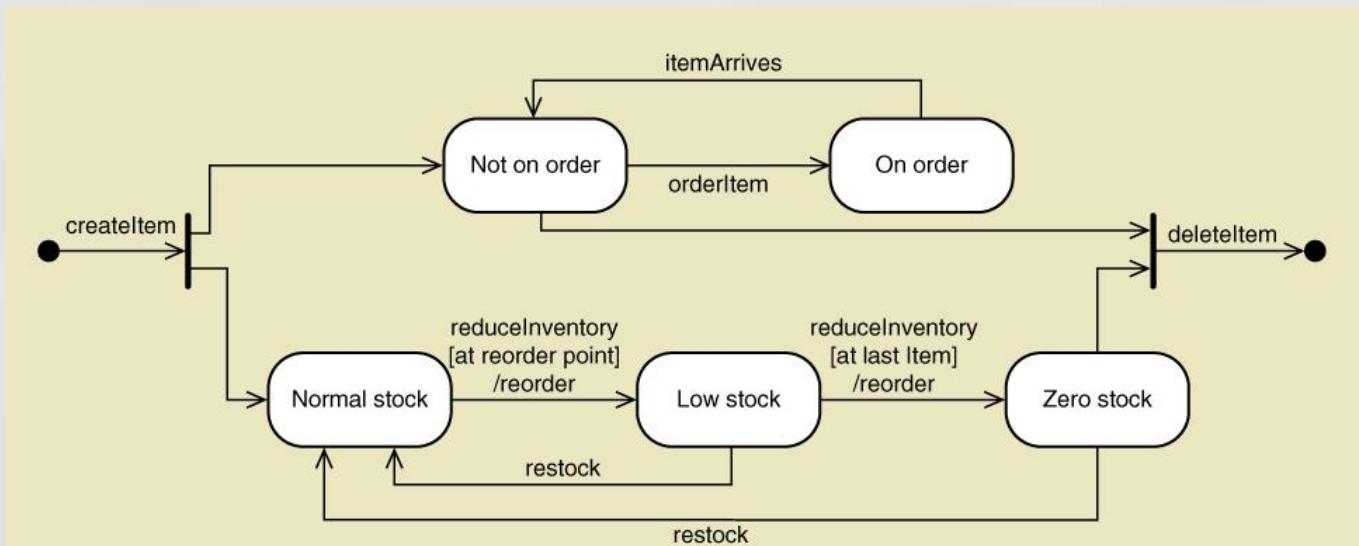
3. Build fragments – see figure below
4. Sequence in correct order – see figure below
5. Look for concurrent paths – see figure below



# RMO – 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