*Edith Cowan University*

*School of Science*

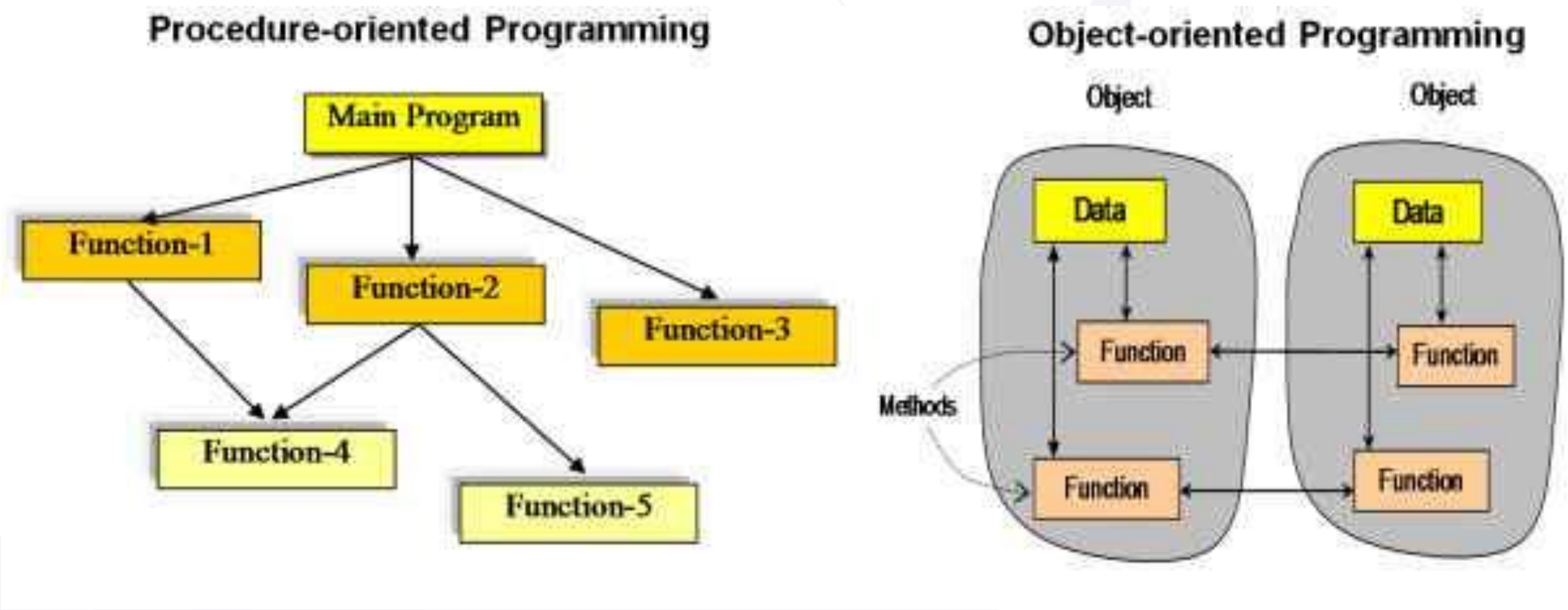
# CSI6207
# Systems Analysis and Database Design

# Object Modeling

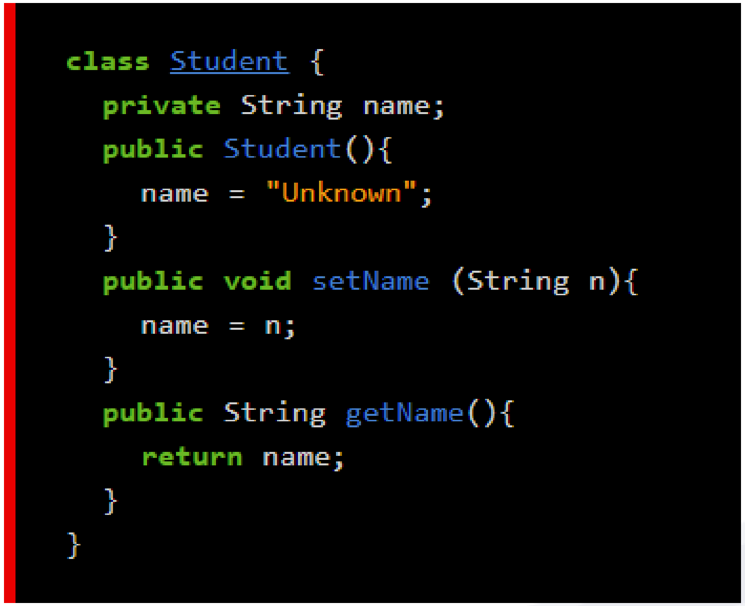
# Evolution of programming techniques

- Programming began in the 1940s, using memory addresses and machine  code directly
- Higher level languages were developed to allow English-like instructions
- Older programs were “monolithic,” and ran from beginning to end
- Newer programs contain modules that can be combined to form programs
- Two major programming techniques:
  - Procedural programming
  - Object-oriented programming
- **Procedural programming**: focuses on the procedures that programmers  create
- **Object-oriented programming**: focuses on objects that represent real-world things and their attributes and behaviors
- Both techniques employ reusable program modules

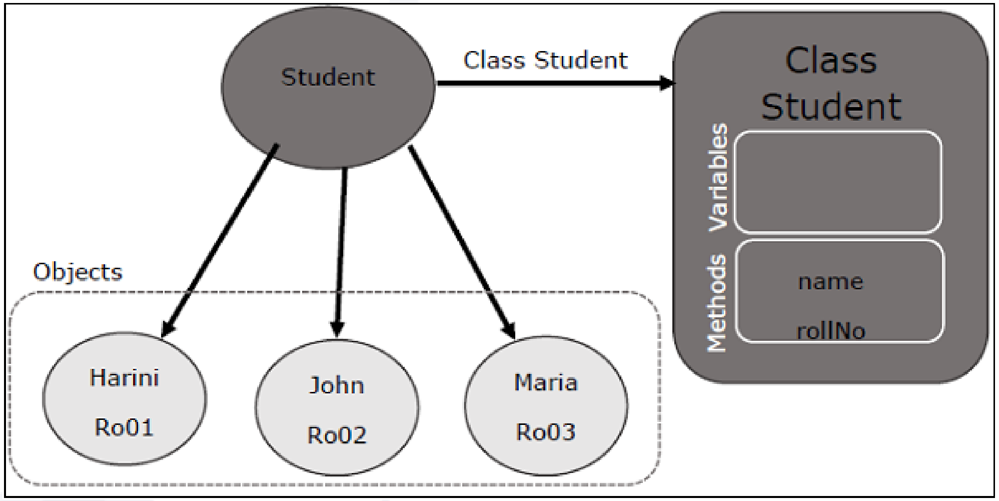
# Evolution of programming techniques



# OOP Example: Class and Objects



```
class Student {
  private String name;
  public Student(){
    name = "Unknown";
  }
  public void setName (String n){
    name = n;
  }
  public String getName(){
    return name;
  }
}
```



[https://www.tutorialspoint.com/scala/scala_classes_objects.htm](https://www.tutorialspoint.com/scala/scala_classes_objects.htm)

```
Student Harini=new Student();
Harini.setName(“HariniRo01”);

Student John= new Student();
John.setName(“JohnRo02”);
```

# Contents

- “Things” in the Problem Domain
- The Domain Model Class Diagram

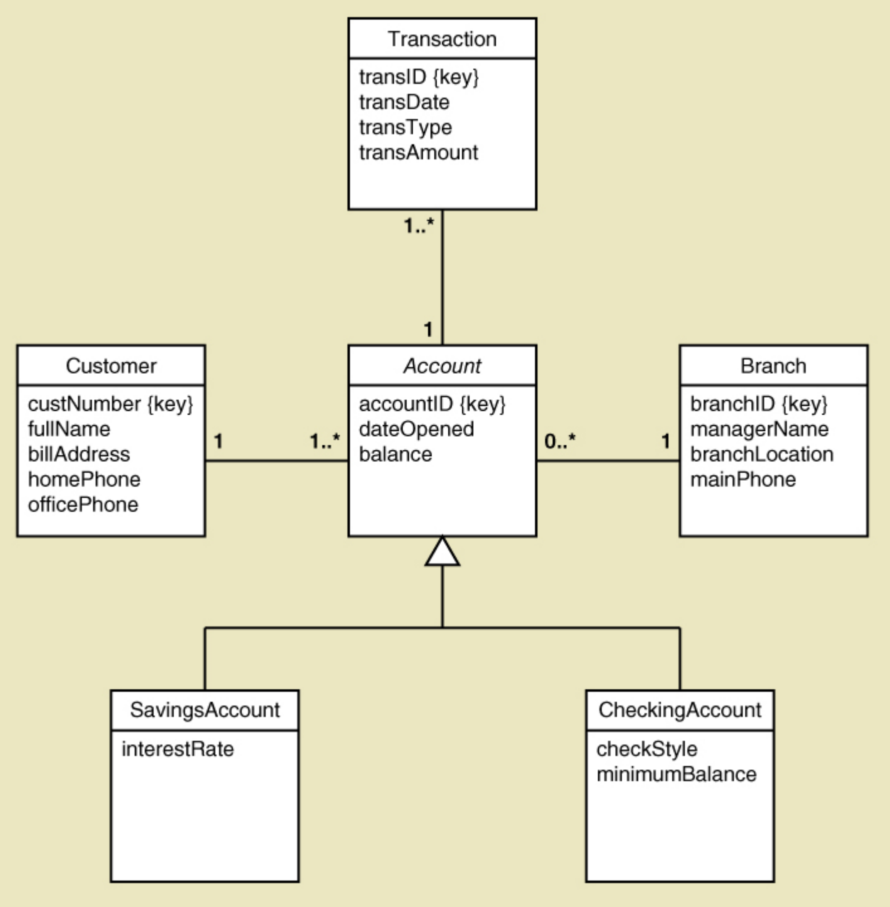


*Fig. A sample class diagram*

# Learning Objectives

- Explain how the concept of “things” in the problem domain also define requirements
- Identify and analyze domain classes needed in the system
- Read, interpret, and create a domain model class diagram
- Understand the domain model class diagram for the RMO Consolidated Sales and Marketing System

# Identifying Data

(from Page 94-98, of Satzinger's Book)

# Things in the Problem Domain

*(Systems Analysis and Design in a Changing World, ©2016. Cengage Learning).*

- 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 referred to as domain model classes within an object-oriented system or entities within a relational system.

# Things in the Problem Domain

## Two Techniques for Identifying Them

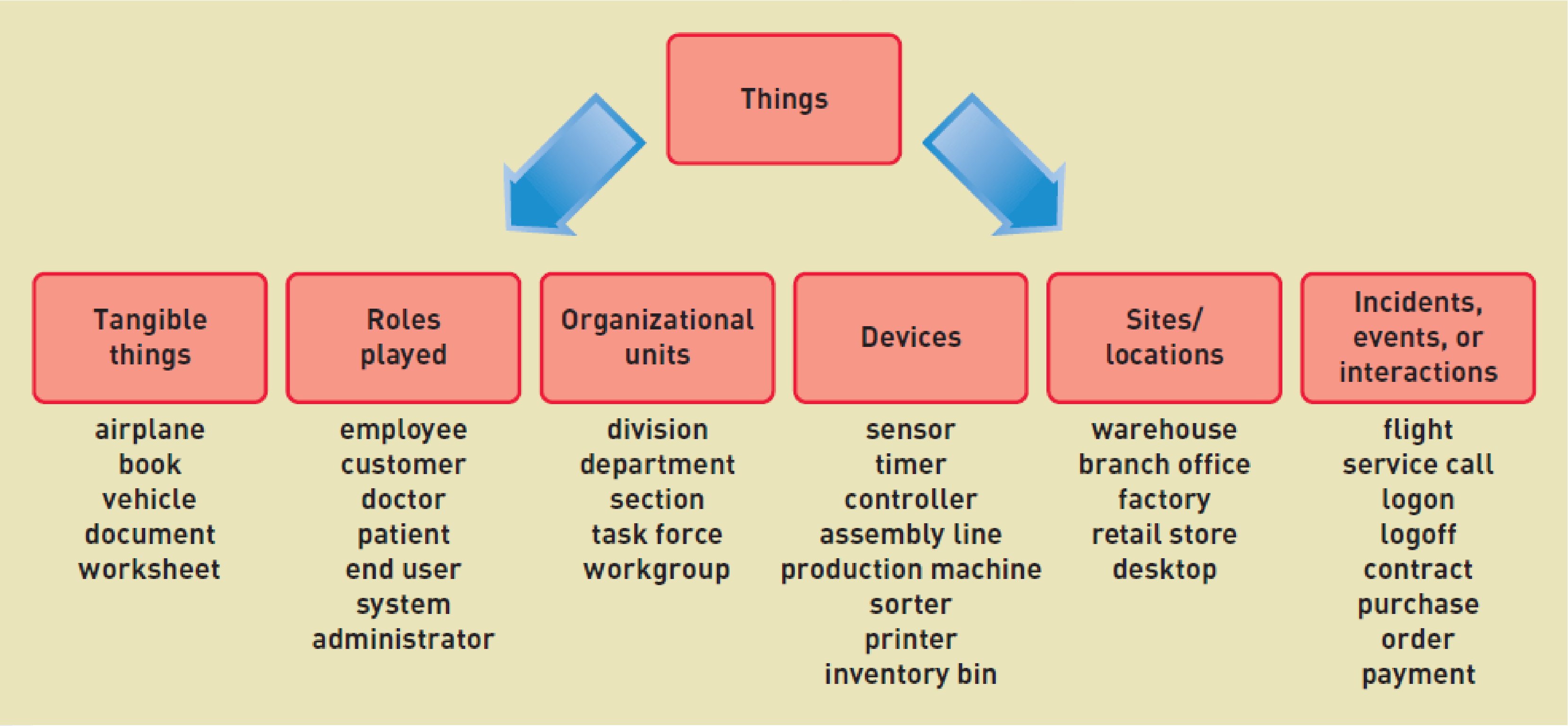
*(Systems Analysis and Design in a Changing World, ©2016. Cengage Learning).*

- Brainstorming Technique
  - Use a checklist of all of the usual types of things typically found and brainstorm to identify domain classes of each type
- Noun Technique
  - Identify all of 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

*(Systems Analysis and Design in a Changing World, ©2016. Cengage Learning)*.

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



Things

| Tangible things | Roles played | Organizational units | Devices | Sites/ locations | Incidents, events, or interactions |
|---|---|---|---|---|---|
| airplane<br>book<br>vehicle<br>document<br>worksheet | employee<br>customer<br>doctor<br>patient<br>end user<br>system administrator | division<br>department<br>section<br>task force<br>workgroup | sensor<br>timer<br>controller<br>assembly line<br>production machine<br>sorter<br>printer<br>inventory bin | warehouse<br>branch office<br>factory<br>retail store<br>desktop | flight<br>service call<br>logon<br>logoff<br>contract<br>purchase order<br>payment |

# Brainstorming Technique: Steps

*(Systems Analysis and Design in a Changing World, ©2016. Cengage Learning)*.

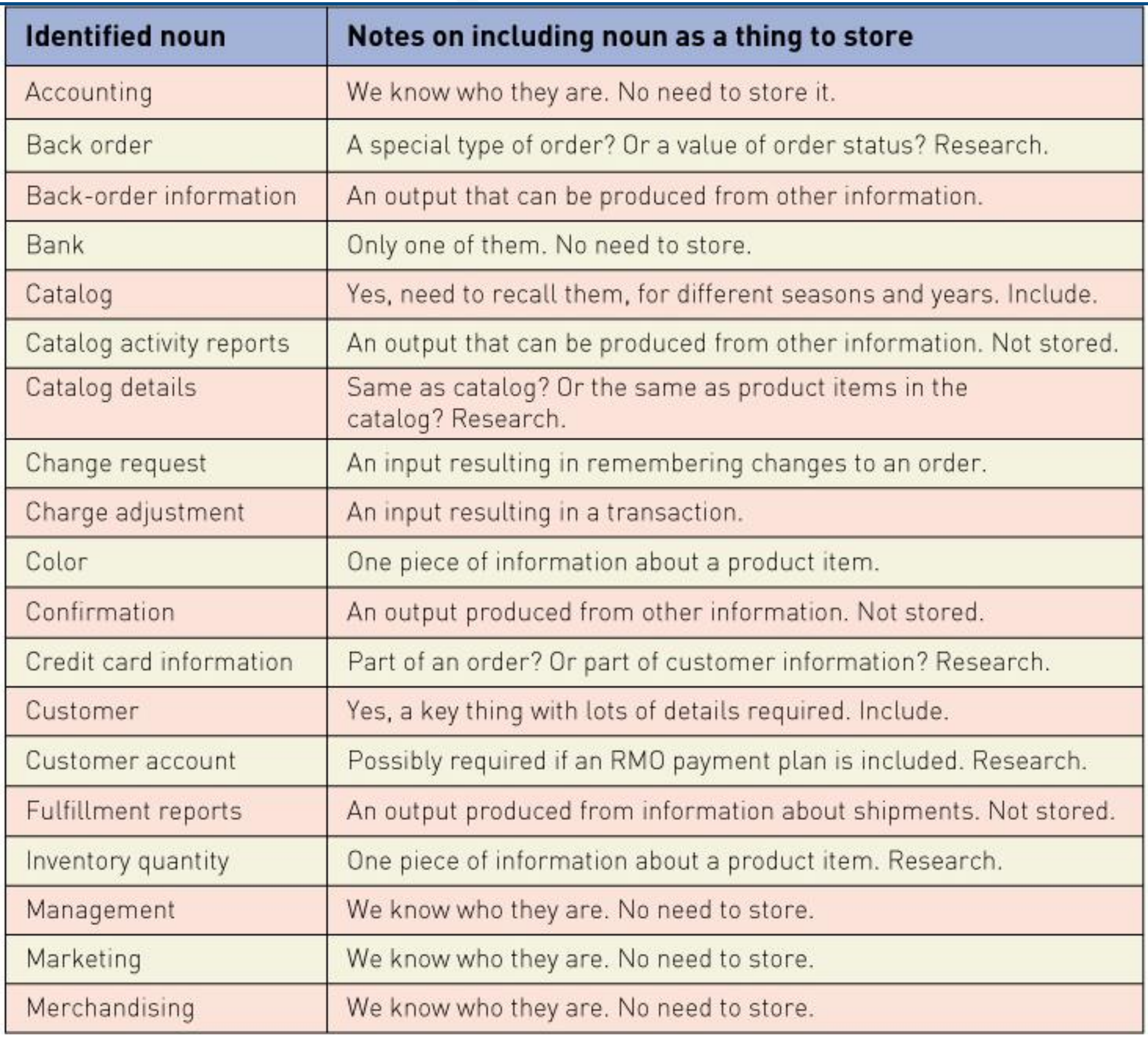
1. Identify a user and a set of use cases
2. Brainstorm with the user to identify things involved when carrying out the use case
   i. that is, things about which information should be captured by the system.
3. Use the types of things (categories) to systematically ask questions about potential things, such as the following:
   i. Are there any tangible things you store information about?
   ii. Are there any locations involved?
   iii. 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

# The Noun Technique

*(Systems Analysis and Design in a Changing World, ©2016. Cengage Learning).*

- A technique to identify problem domain classes (things) by
  - finding, classifying, and refining a list of nouns that come up in 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
  - May need to be trimmed and refined.
- Difficulty identifying synonyms and things that are really attributes
  - Person and student
- Good place to start when there are no users available to help brainstorm

- 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. |
| 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. |

*(Systems Analysis and Design in a Changing World, ©2016. Cengage Learning).*

# Attributes, Associations and Relationship

# 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 to be used in a database
  - Optional for domain classes in an object oriented system. 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.

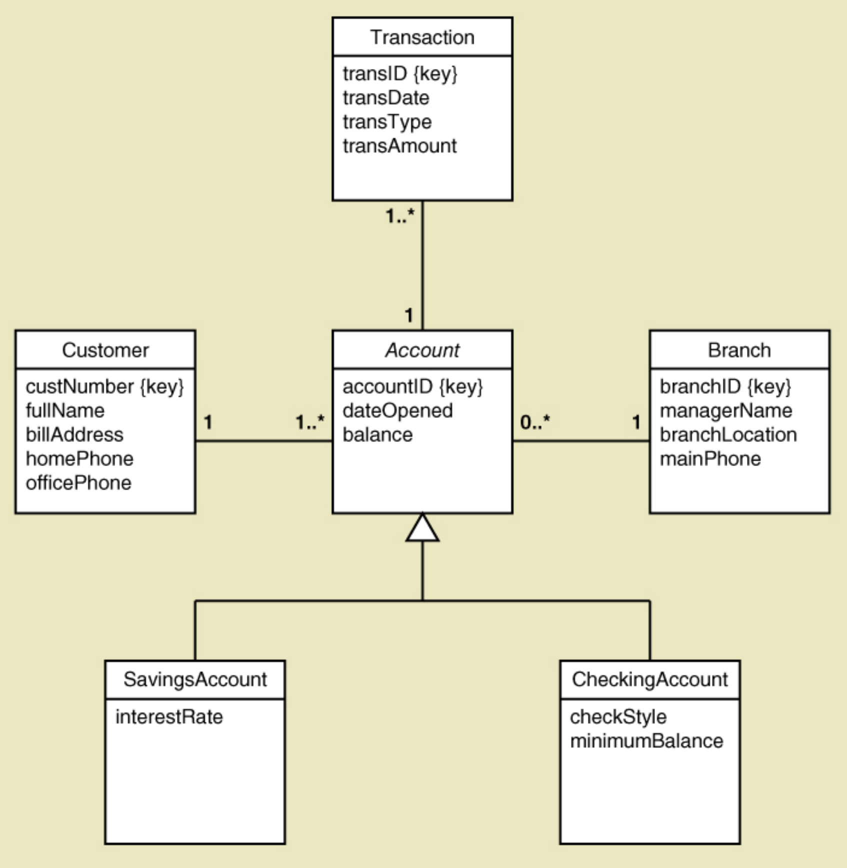
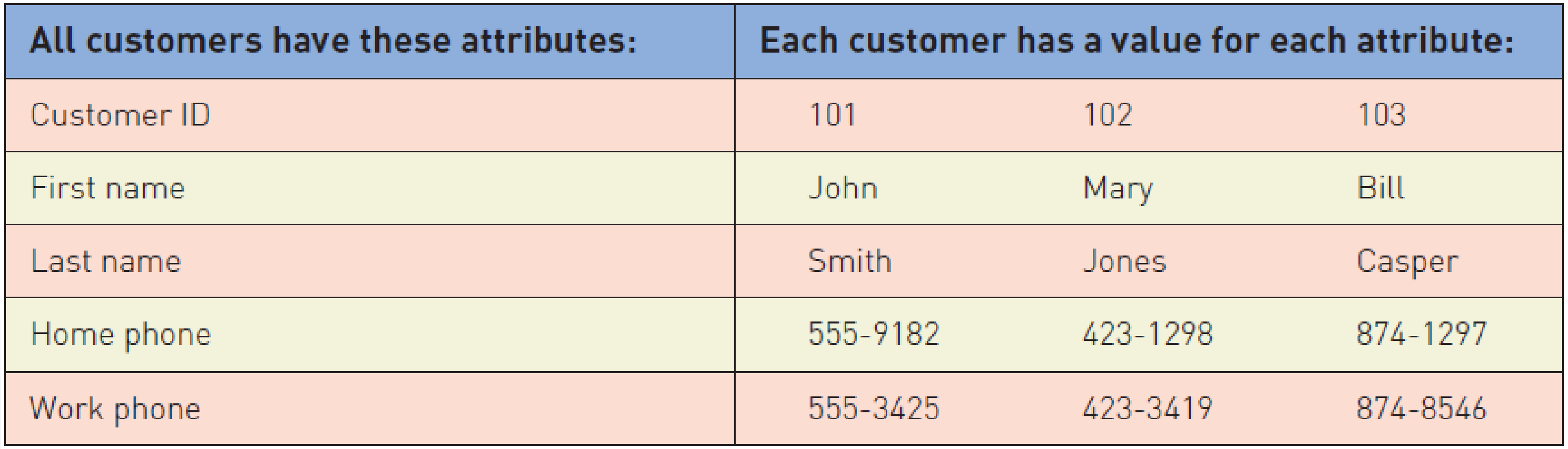


*Fig. A sample class diagram*

# Attributes and Values

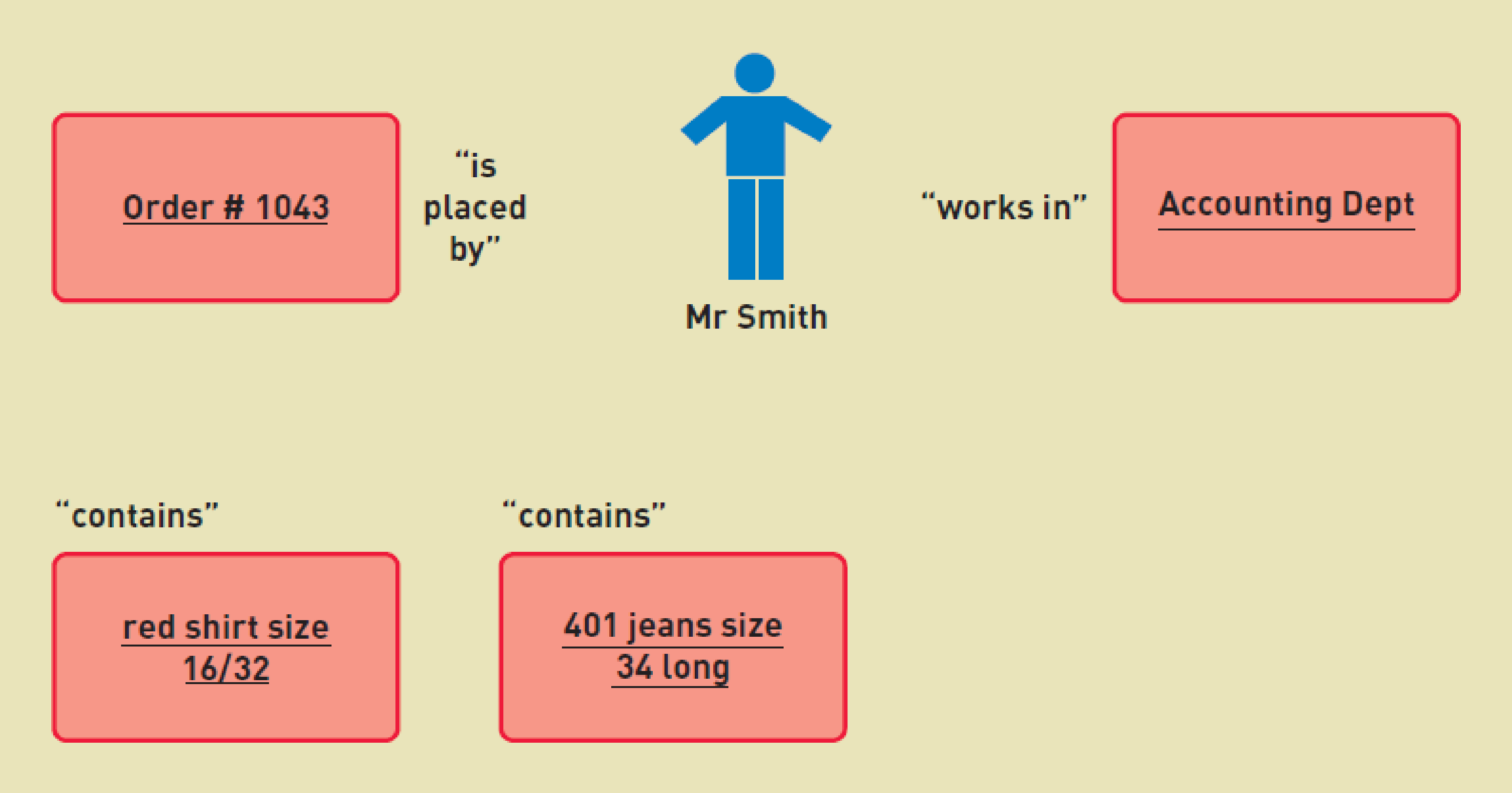
- Class is a type of thing.
- Object is a specific instance of the class.
- Each instance has its own values for an attribute



| 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)

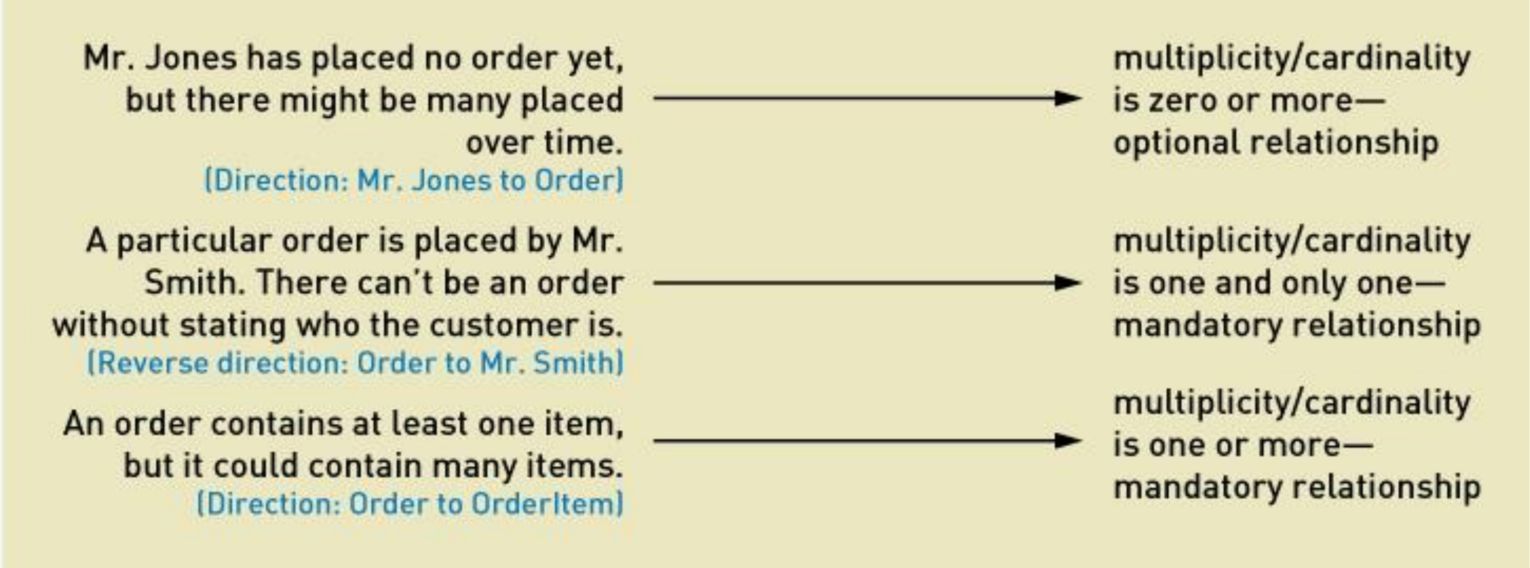


# Just to Clarify…

- 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 for databases (covered in Week 6)
  - **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



Mr. Jones has placed no order yet, but there might be many placed over time.
(Direction: Mr. Jones to Order) → multiplicity/cardinality is zero or more—optional relationship

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) → multiplicity/cardinality is one and only one—mandatory relationship

An order contains at least one item, but it could contain many items.
(Direction: Order to OrderItem) → 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)

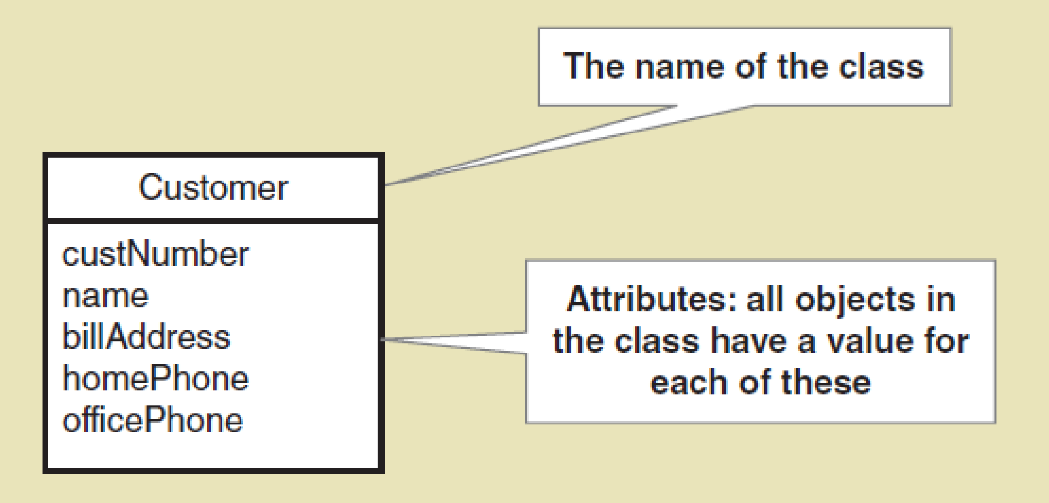
# The Domain Model
# Class Diagram

# The Domain Model Class Diagram

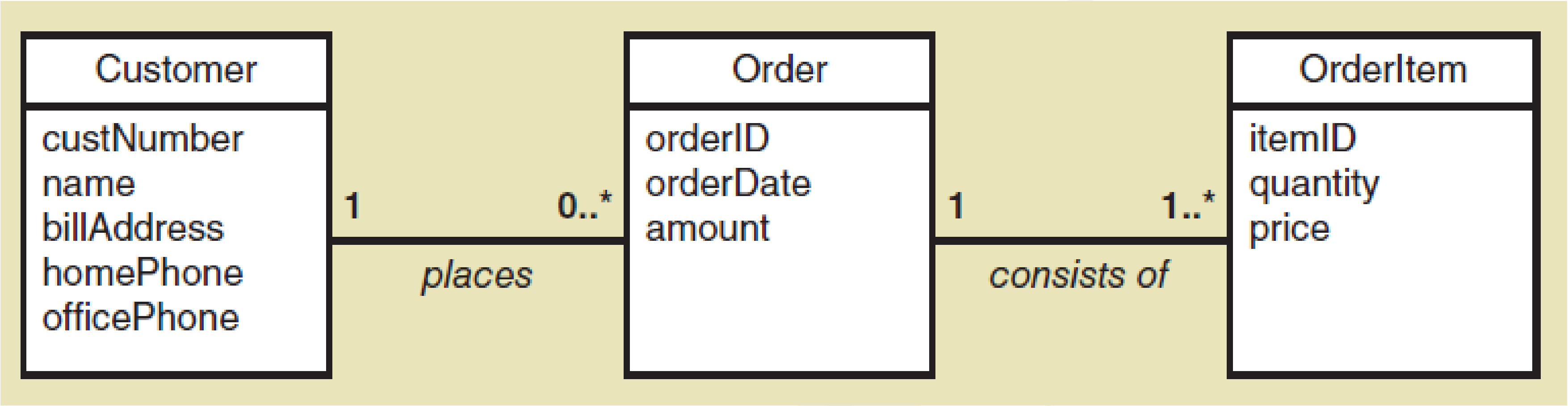
- Class
  - A type of classification used to describe a collection of objects
- Domain Class
  - Classes that describe objects in the problem domain
- Class Diagram
  - A UML diagram that shows classes with attributes and associations (plus methods if it models software classes)
- Domain Model Class 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

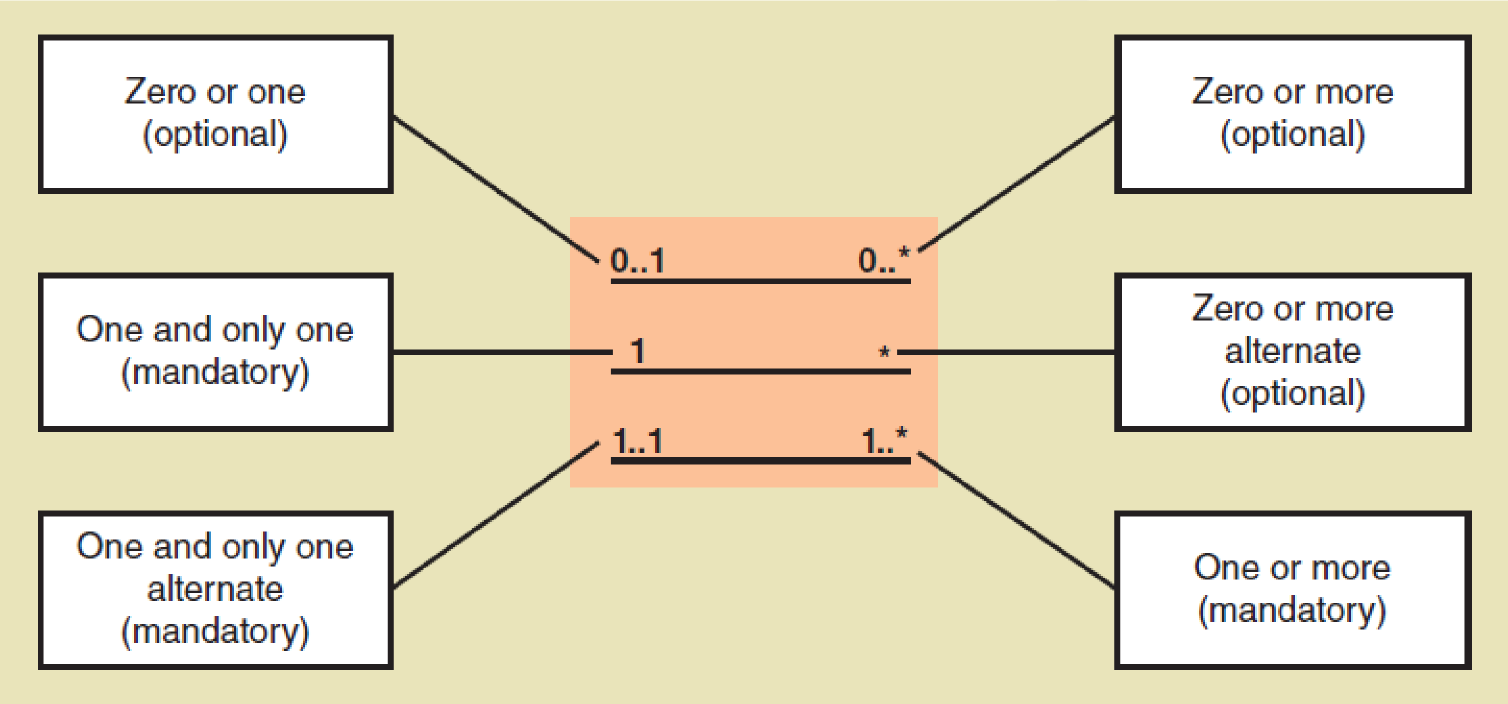


# A Simple Domain Model Class Diagram



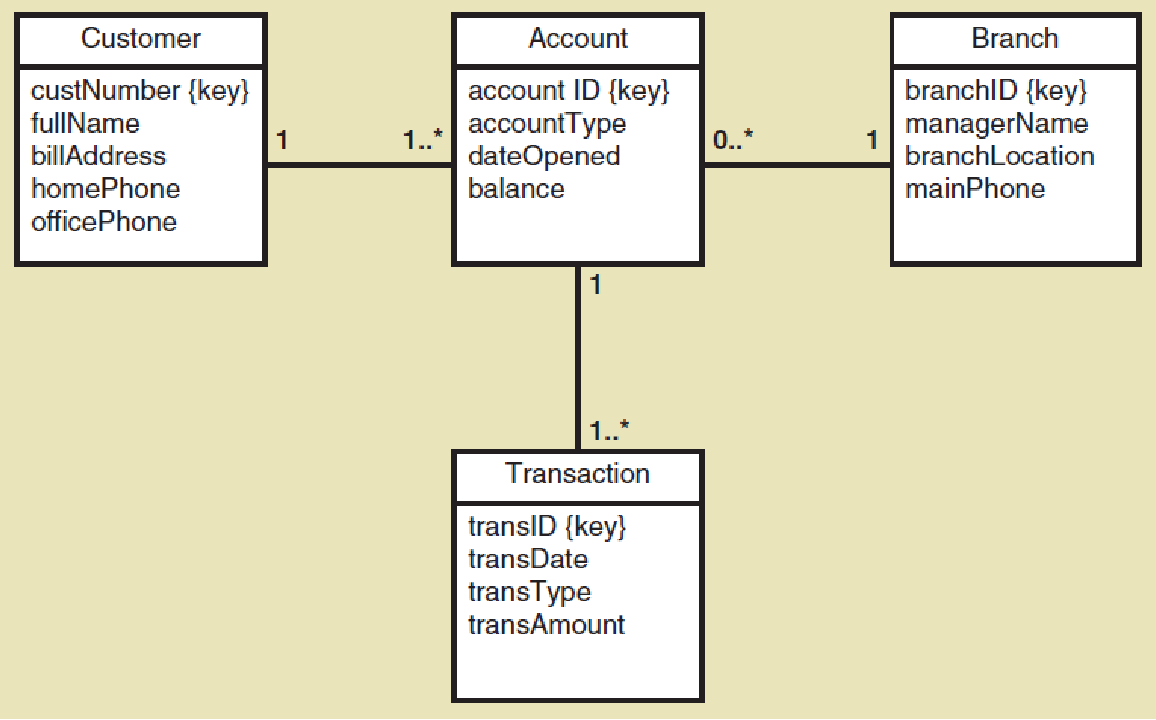
- Note: This diagram matches the semantic net shown previously
  - – A customer places zero or more orders
  - – An order is placed by exactly one customer
  - – An order consists of one or more order items
  - – An order item is part of exactly one order

# UML Notation for Multiplicity


Zero or one (optional)
One and only one (mandatory)
One and only one alternate (mandatory)
0..1
1
1..1
0..*
*
1..*
Zero or more (optional)
Zero or more alternate (optional)
One or more (mandatory)

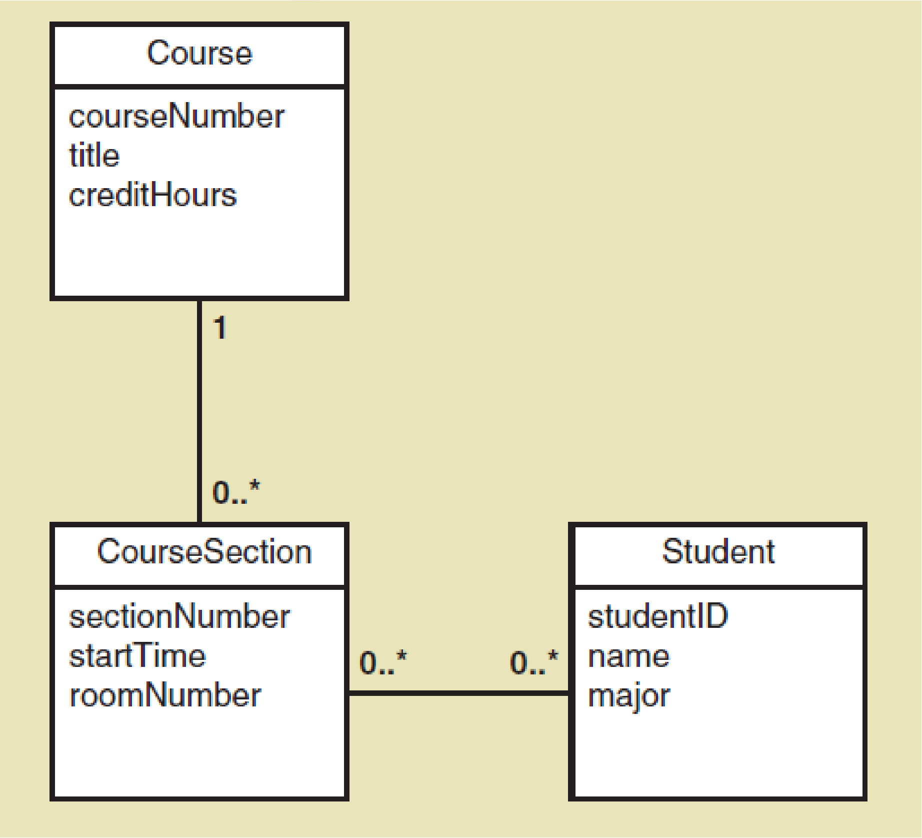
# Domain Model Class Diagram

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



# Domain Model Class Diagram

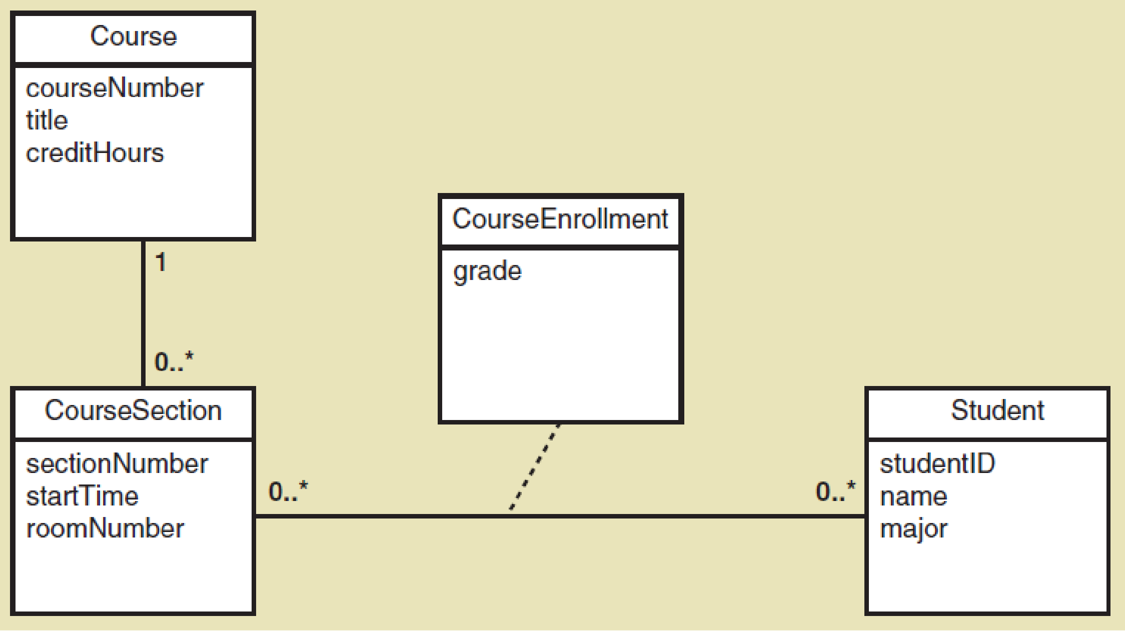
- Course Enrollment at a University
  - A Course has many CourseSections
  - A CourseSection has many Students and a Student is registered in many CourseSections
- Problem
  - How/where to capture student grades?



# Refined Course Enrollment Model

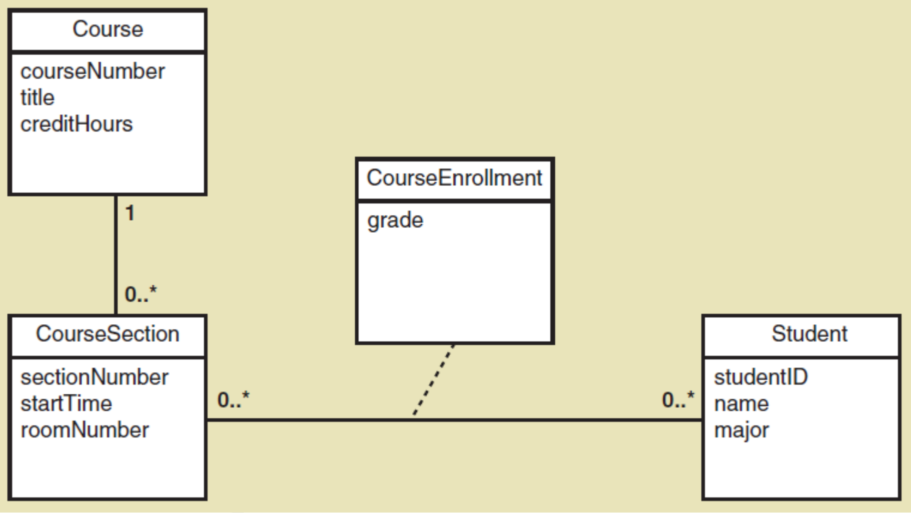
## with an Association Class CourseEnrollment

- **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)



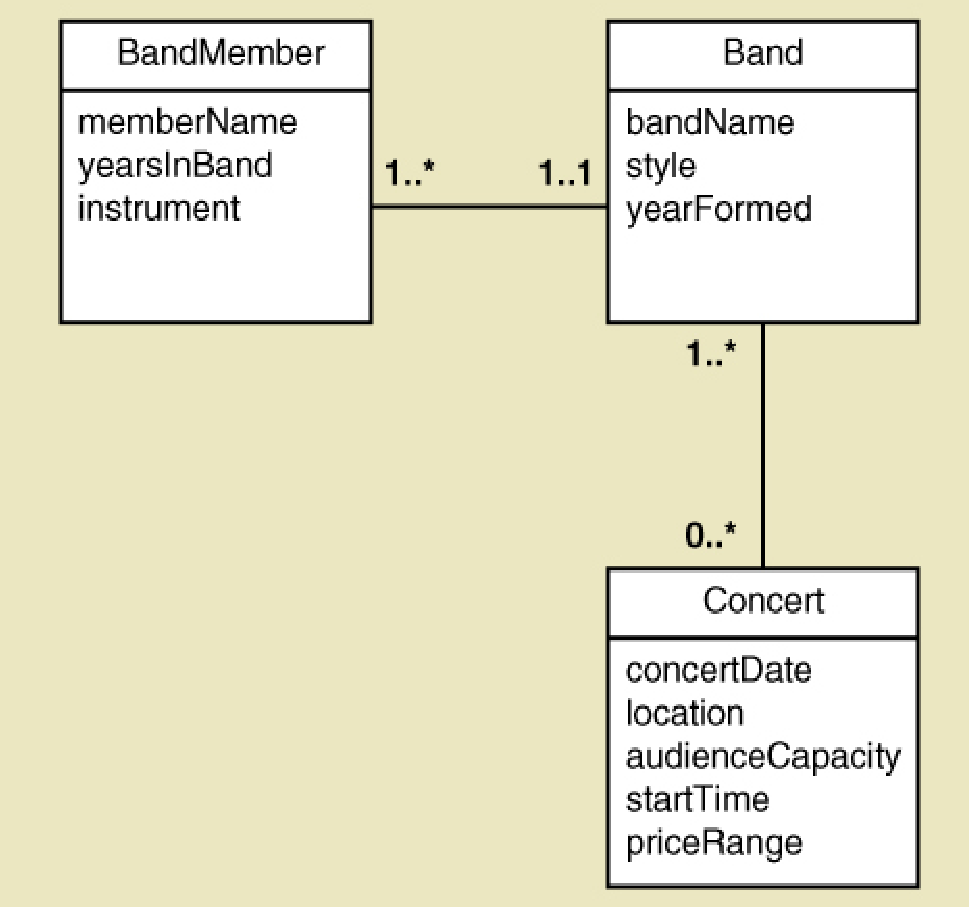
# Association Class Properties

- 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 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.



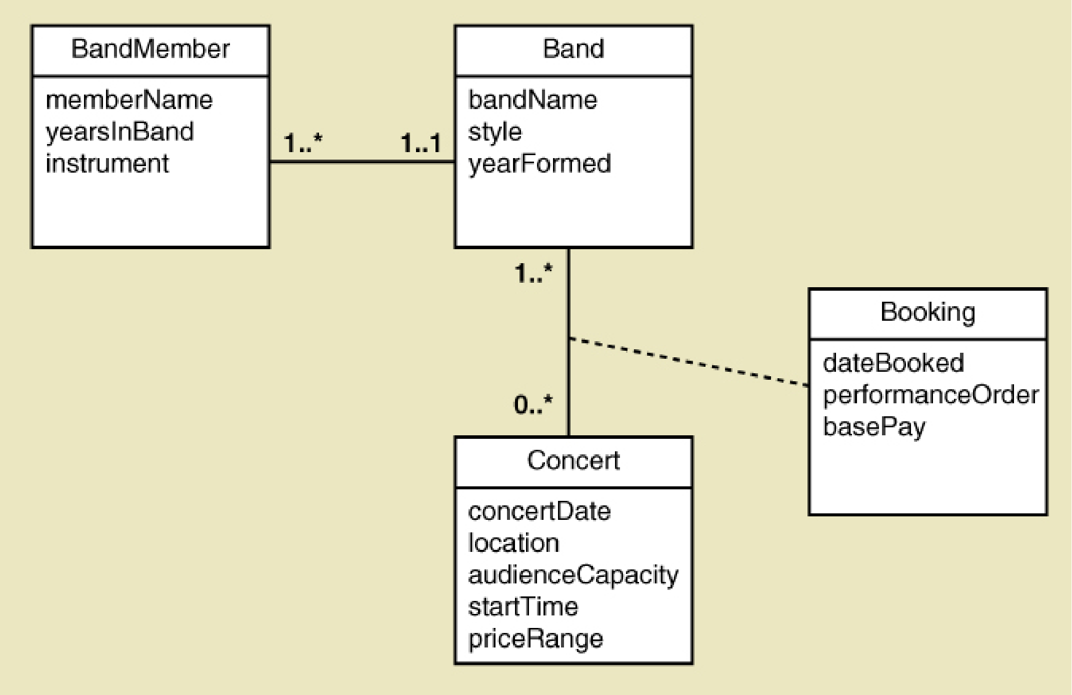
# Band with members and concerts

- 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

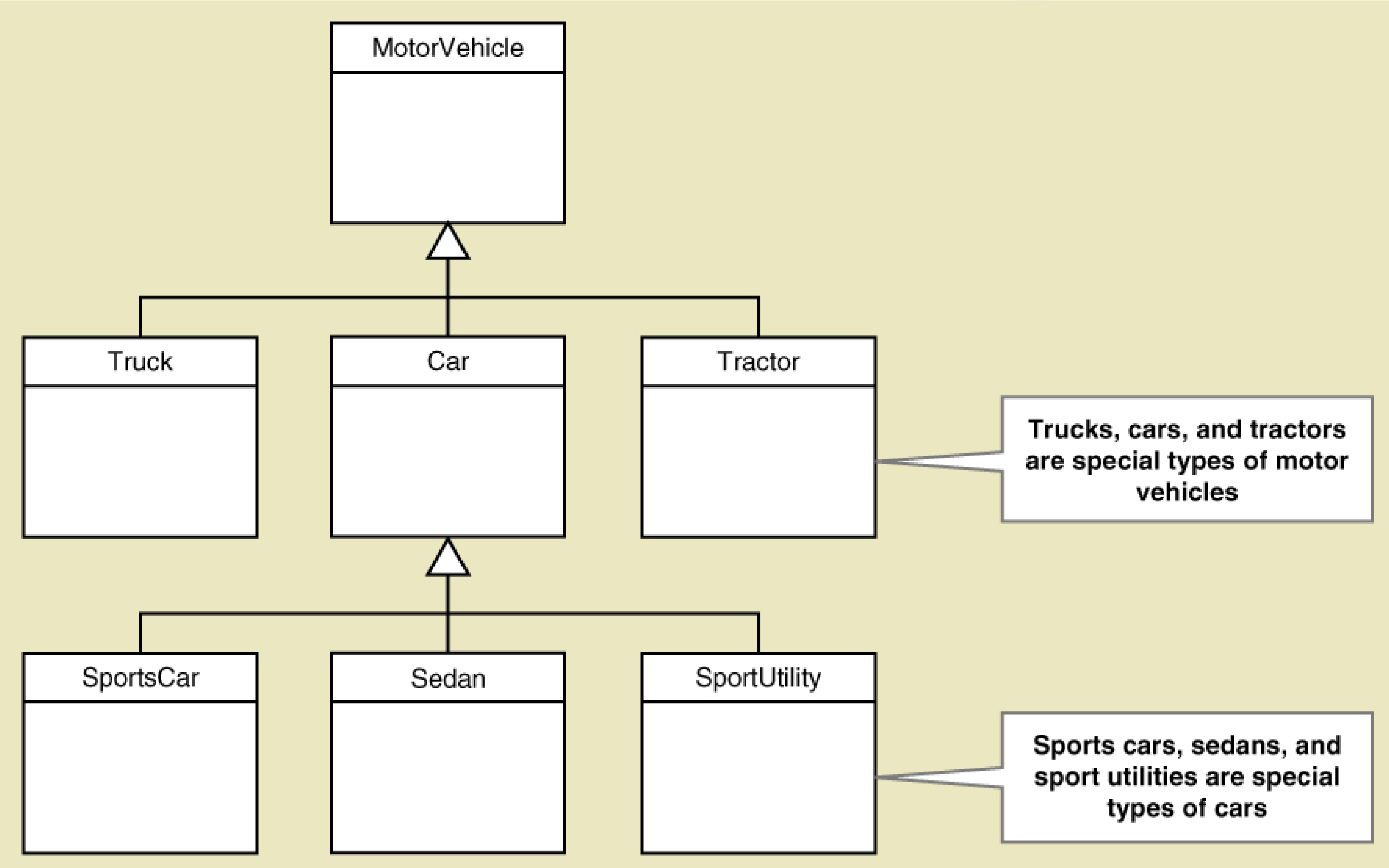
- Note: The association class (Booking) also provides a name and meaning for the association
- Given the keys you identified, what is the key for the Booking class? Does it uniquely identify instances?



# 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 classes inherit characteristics of the more general superclass

# Generalization/Specialization



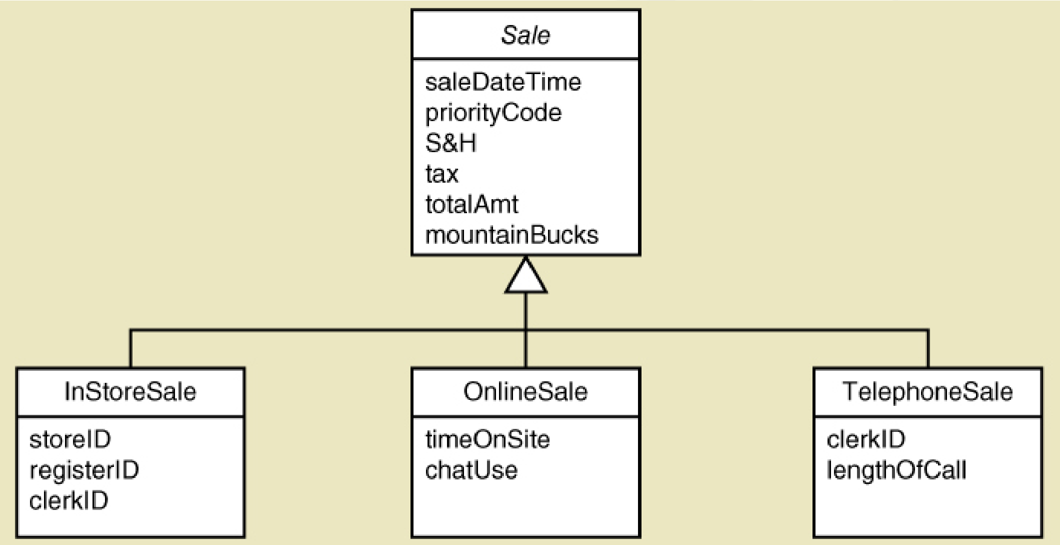
Trucks, cars, and tractors are special types of motor vehicles

Sports cars, sedans, and sport utilities are special types of cars

# Generalization/Specialization

## Inheritance for RMO Three Types of Sales

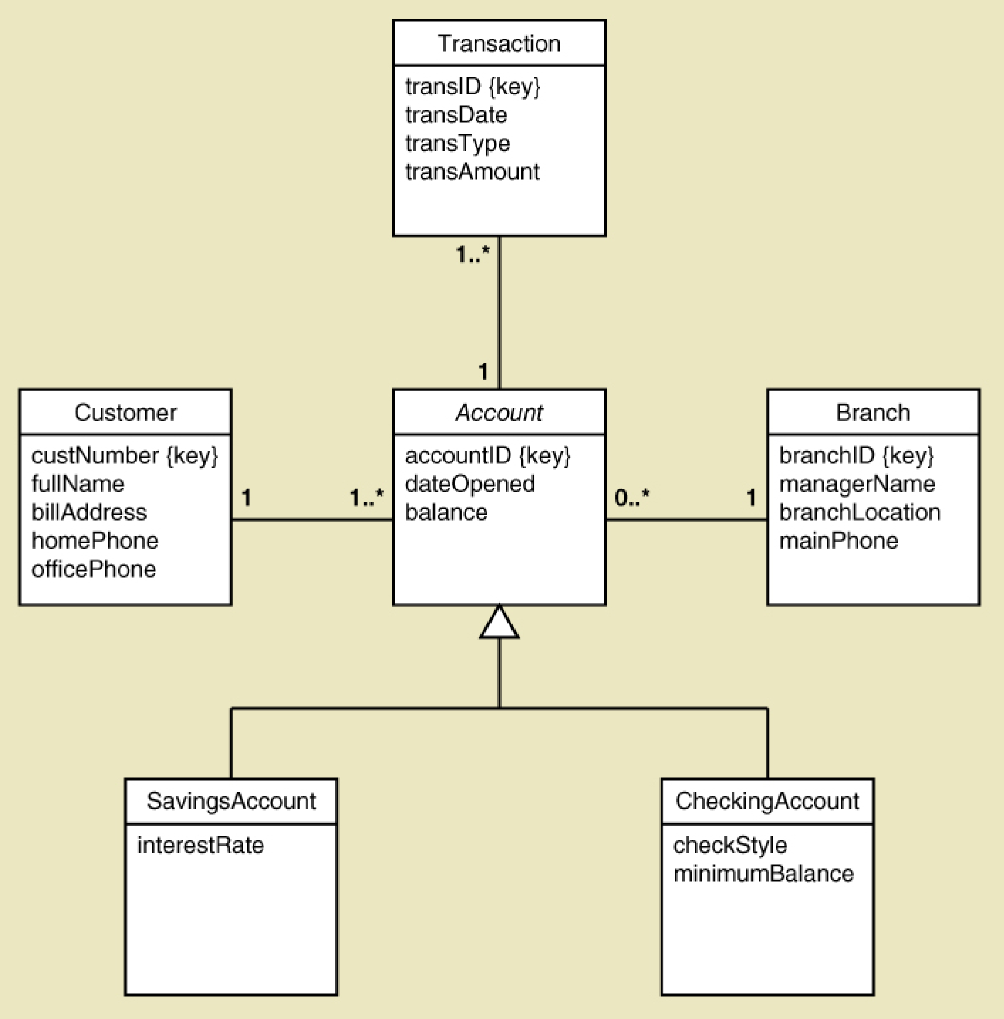
- Abstract class— a class that allow subclasses to inherit characteristics but never gets instantiated. In Italics (*Sale*)
- Concrete class— a class that can have instances
- Inheritance – Attributes of OnlineSale are:
  - timeOnSite, chatUse, saleDateTime, priorityCode, S&H, tax, totalAmt…



# 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 on UML Relationships

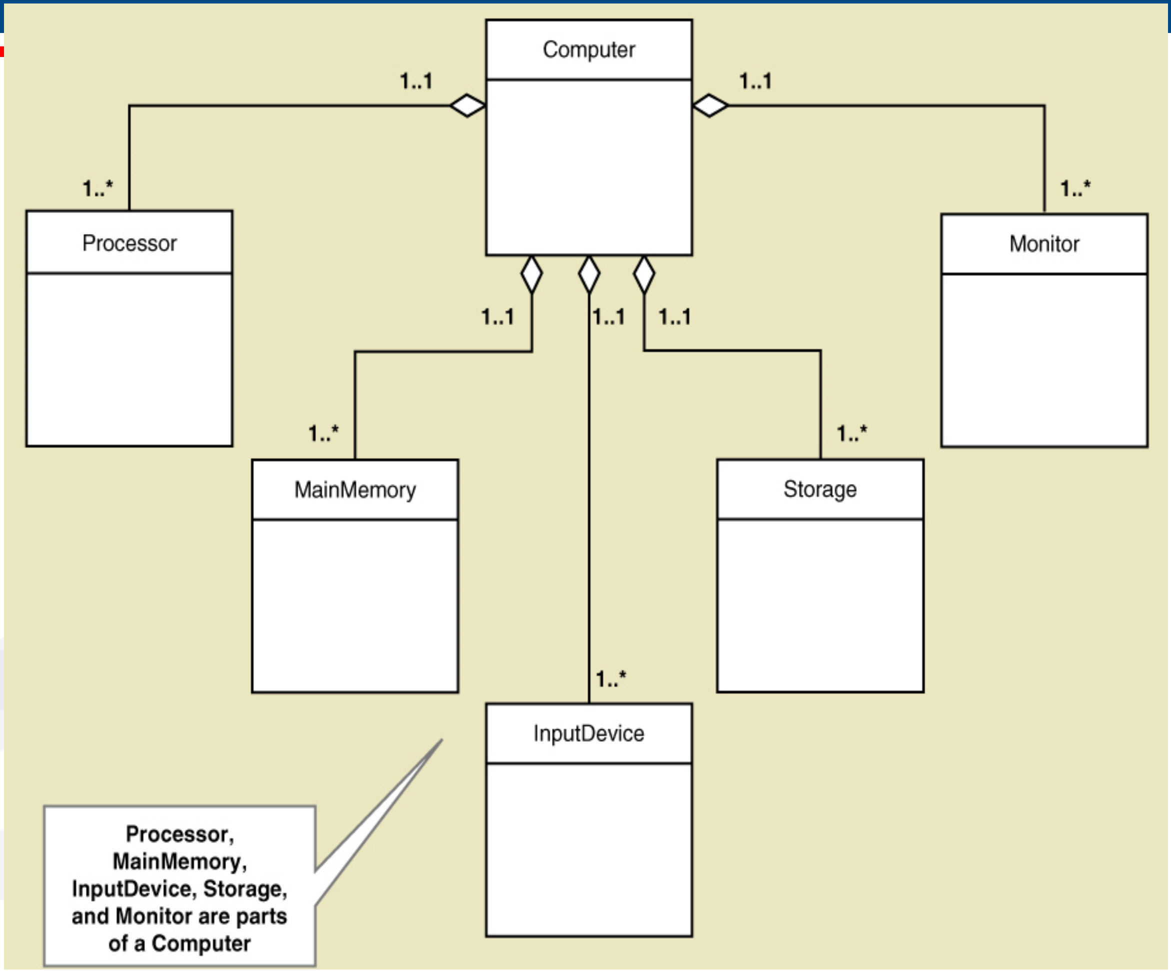
- There are actually three types of ***relationships*** in class diagrams
  - Association Relationships
    - These are associations discussed previously, just like ERD relationships
  - Whole Part Relationships
    - One class is a component or part of another class
  - Generalizations/Specialization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

# 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 OrderIterms)
  - Chip has circuits (without the chip, there are no circuits)

# Whole Part Relationships
## Computer and its Parts

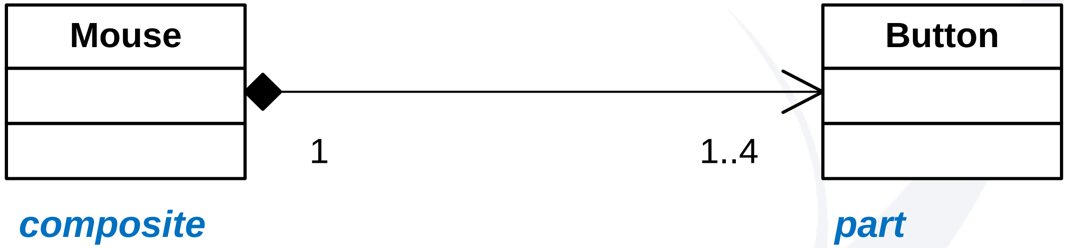
- Note: this is composition, with diamond symbol.
- Whole part can have multiplicity symbols, too (not shown)



Processor, MainMemory, InputDevice, Storage, and Monitor are parts of a Computer

# Composition Semantics

- *Composition example*



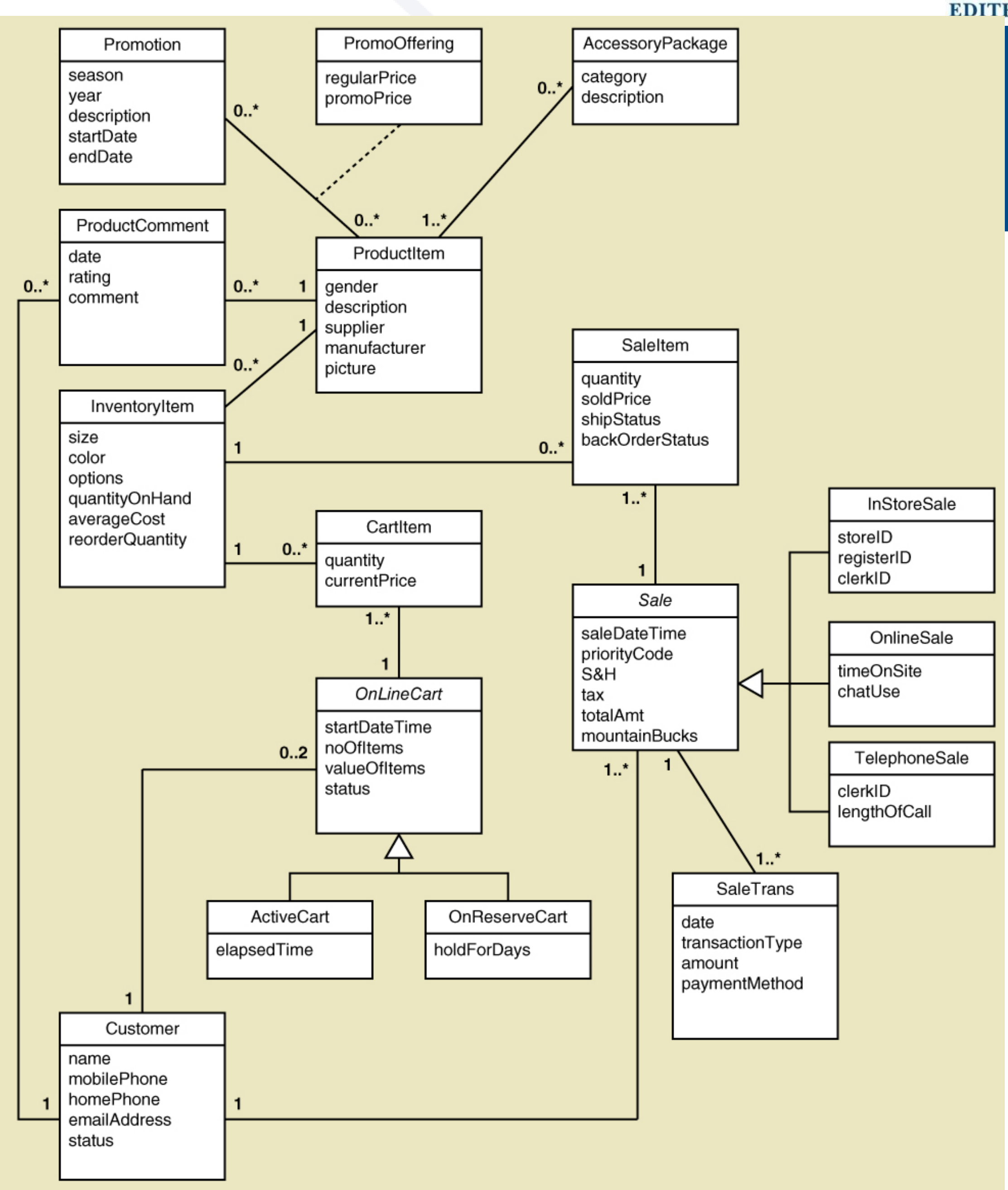
  - *Button object – no independent existence from Mouse object.*
  - *If the Mouse object destroyed, Button object also destroyed.*
  - *Each button object can belong to exactly one Mouse object.*

# 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



Promotion
- season
- year
- description
- startDate
- endDate

PromoOffering
- regularPrice
- promoPrice

AccessoryPackage
- category
- description

ProductComment
- date
- rating
- comment

ProductItem
- gender
- description
- supplier
- manufacturer
- picture

SaleItem
- quantity
- soldPrice
- shipStatus
- backOrderStatus

InventoryItem
- size
- color
- options
- quantityOnHand
- averageCost
- reorderQuantity

CartItem
- quantity
- currentPrice

InStoreSale
- storeID
- registerID
- clerkID

*Sale*
- saleDateTime
- priorityCode
- S&H
- tax
- totalAmt
- mountainBucks

OnlineSale
- timeOnSite
- chatUse

*OnLineCart*
- startDateTime
- noOfItems
- valueOfItems
- status

TelephoneSale
- clerkID
- lengthOfCall

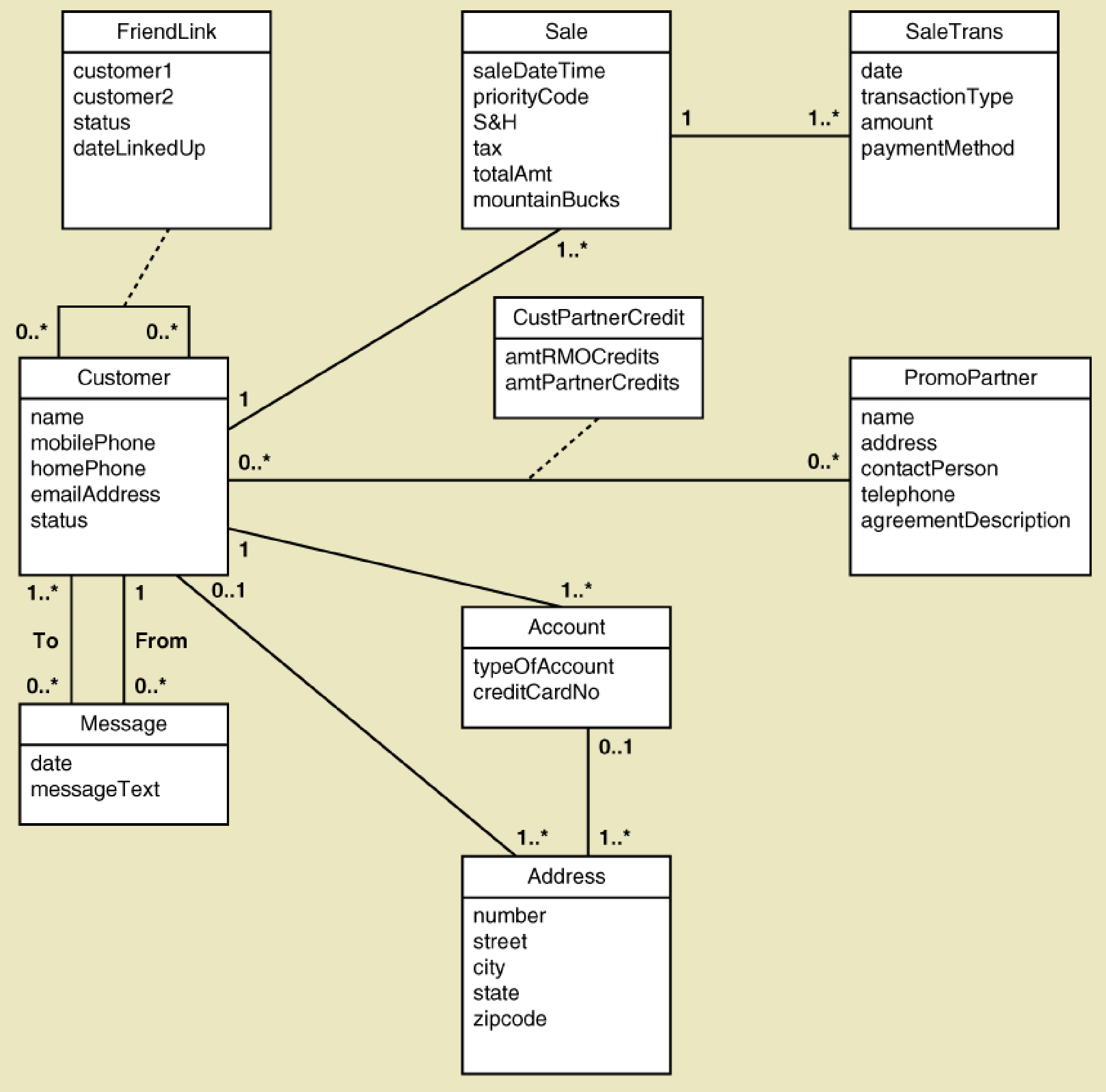
ActiveCart
- elapsedTime

OnReserveCart
- holdForDays

SaleTrans
- date
- transactionType
- amount
- paymentMethod

Customer
- name
- mobilePhone
- homePhone
- emailAddress
- status

# RMO CSMS Project
## Customer Account Subsystem
## Domain Model Class Diagram

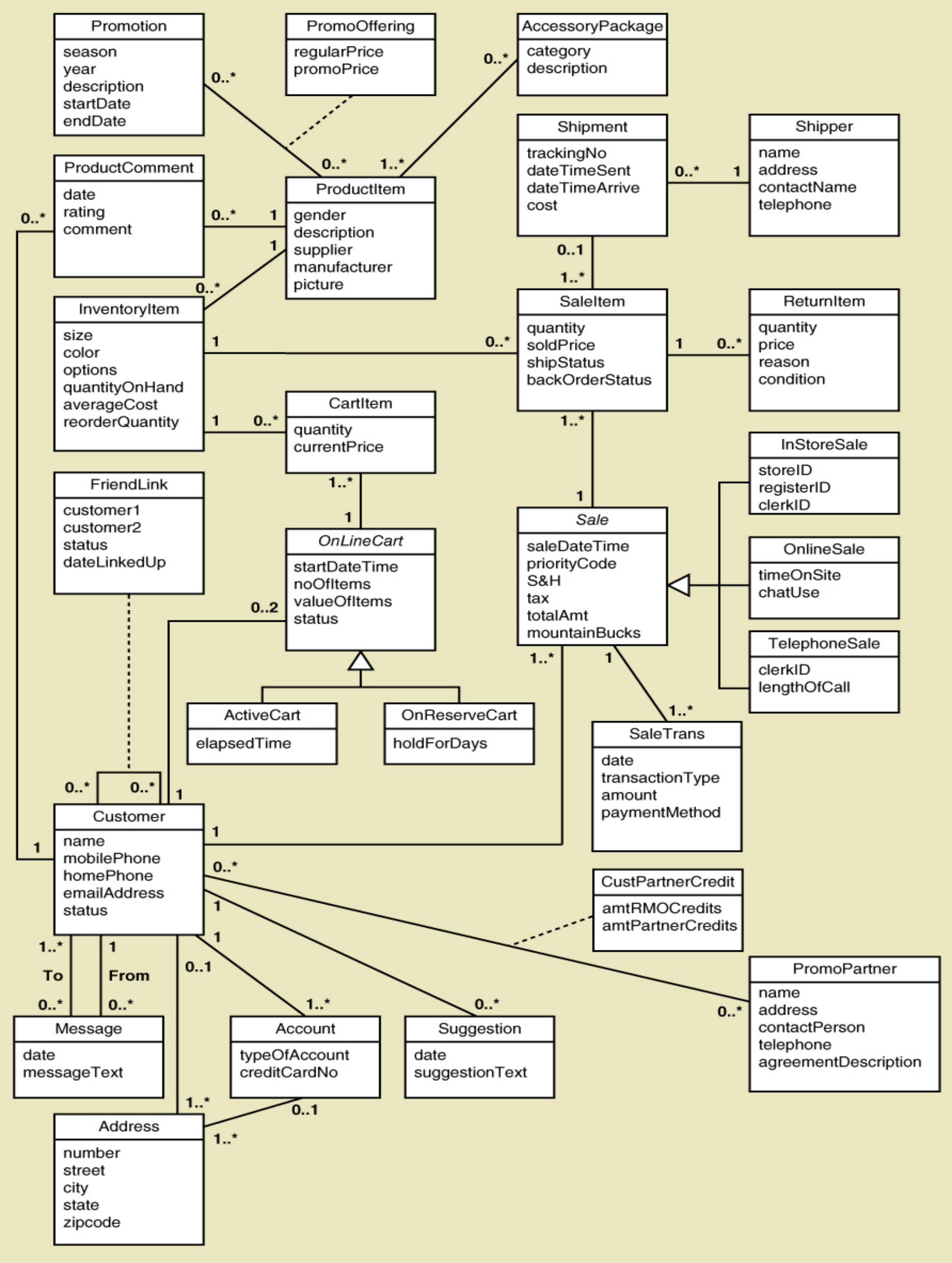


*Edith Cowan University*

*School of Science*

# RMO CSMS Project

## Complete Domain Model Class Diagram



Promotion
season
year
description
startDate
endDate

PromoOffering
regularPrice
promoPrice

AccessoryPackage
category
description

ProductComment
date
rating
comment

ProductItem
gender
description
supplier
manufacturer
picture

Shipment
trackingNo
dateTimeSent
dateTimeArrive
cost

Shipper
name
address
contactName
telephone

InventoryItem
size
color
options
quantityOnHand
averageCost
reorderQuantity

SaleItem
quantity
soldPrice
shipStatus
backOrderStatus

ReturnItem
quantity
price
reason
condition

CartItem
quantity
currentPrice

FriendLink
customer1
customer2
status
dateLinkedUp

*OnLineCart*
startDateTime
noOfItems
valueOfItems
status

ActiveCart
elapsedTime

OnReserveCart
holdForDays

*Sale*
saleDateTime
priorityCode
S&H
tax
totalAmt
mountainBucks

InStoreSale
storeID
registerID
clerkID

OnlineSale
timeOnSite
chatUse

TelephoneSale
clerkID
lengthOfCall

SaleTrans
date
transactionType
amount
paymentMethod

Customer
name
mobilePhone
homePhone
emailAddress
status

CustPartnerCredit
amtRMOCredits
amtPartnerCredits

PromoPartner
name
address
contactPerson
telephone
agreementDescription

Message
date
messageText

Account
typeOfAccount
creditCardNo

Suggestion
date
suggestionText

Address
number
street
city
state
zipcode

# RMO CSMS Project
## Domain Model Class Diagrams

- 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

# Summary

# Summary

- 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

- 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

# Questions?