Chapter 10
Object-Oriented Design Principles

Dr. Supakit Nootyaskool
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Outline

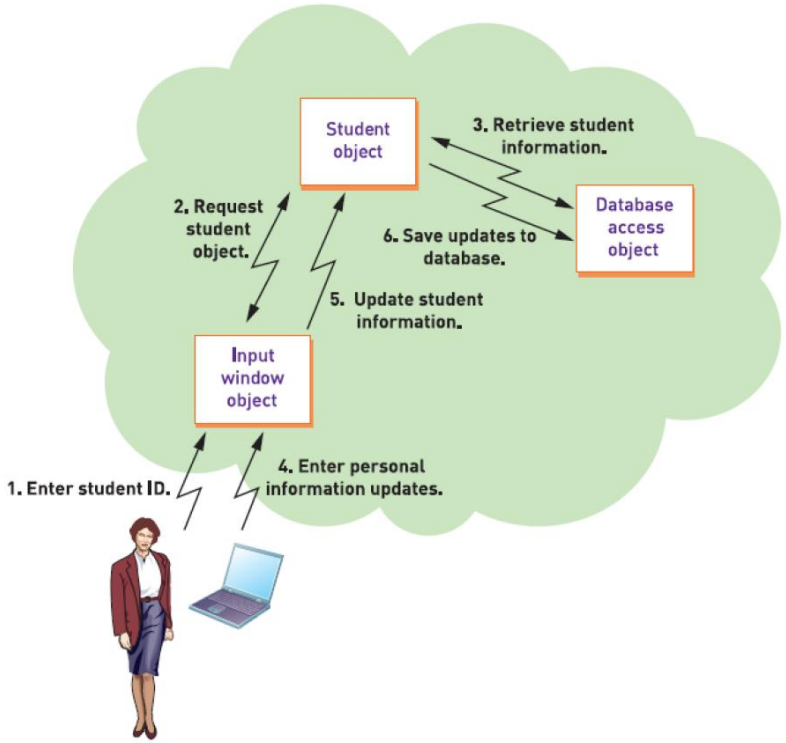
- Object-oriented design: bridging from analysis to implementation
- Object-oriented architectural design
- Fundamental principles of object-oriented detailed design
- Design classes and the design class diagram
- Detailed design with CRC cards
- Fundamental detailed design principles

Learning objective

- Explain
  - the purpose and objective of object oriented design
  - some of the fundamental principles of object-oriented design
- Develop
  - UML component diagrams
  - design class diagrams
- Use CRC cards to define class responsibilities and collaborations

10.1 Object-oriented design: Bridging from analysis to implementation

- OOD is a process by which a set of detailed OOD models are build and then used by the programmer to write and test the new system.
- An object-oriented programming (OOP) is a set of the program by using the concept of object representation.

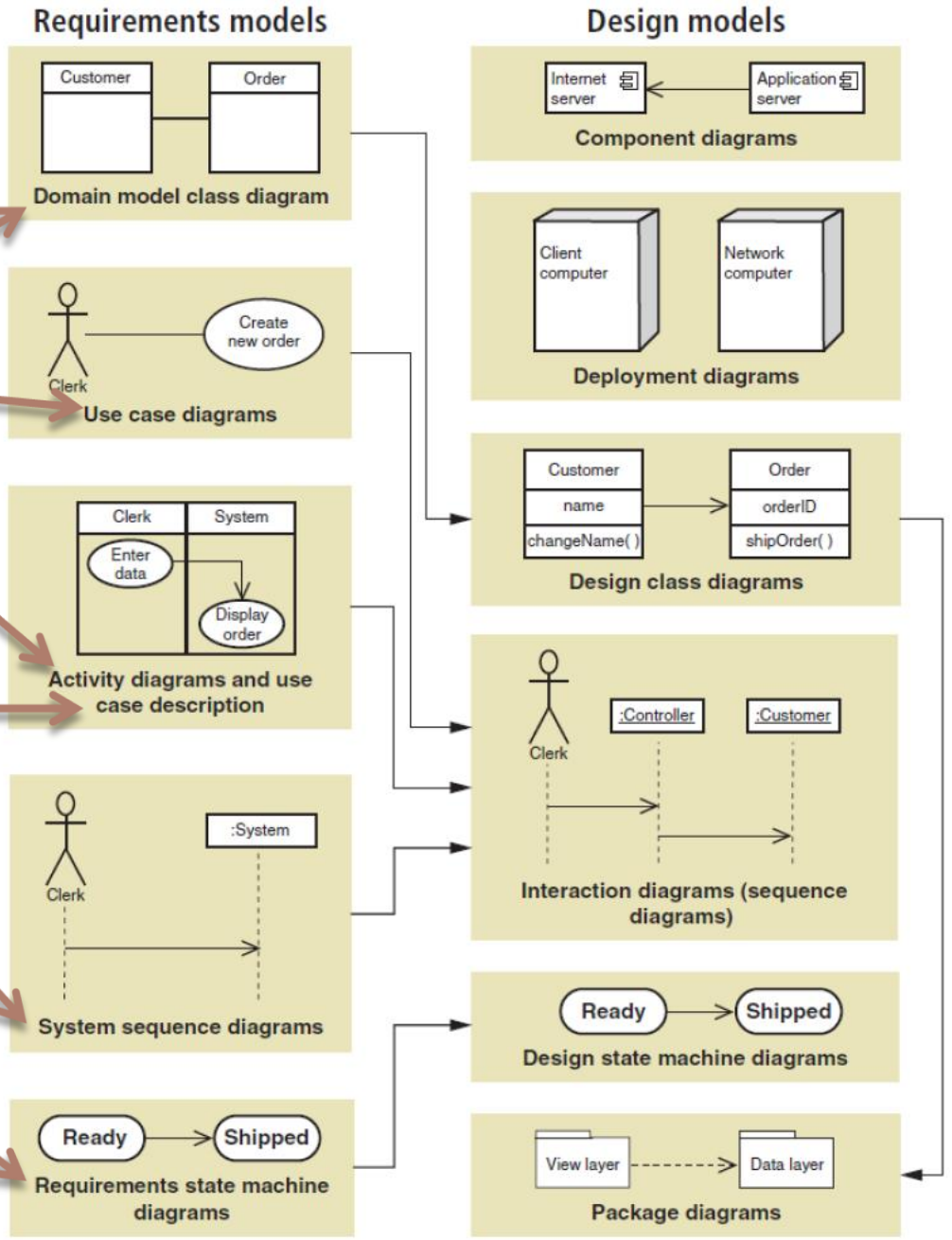


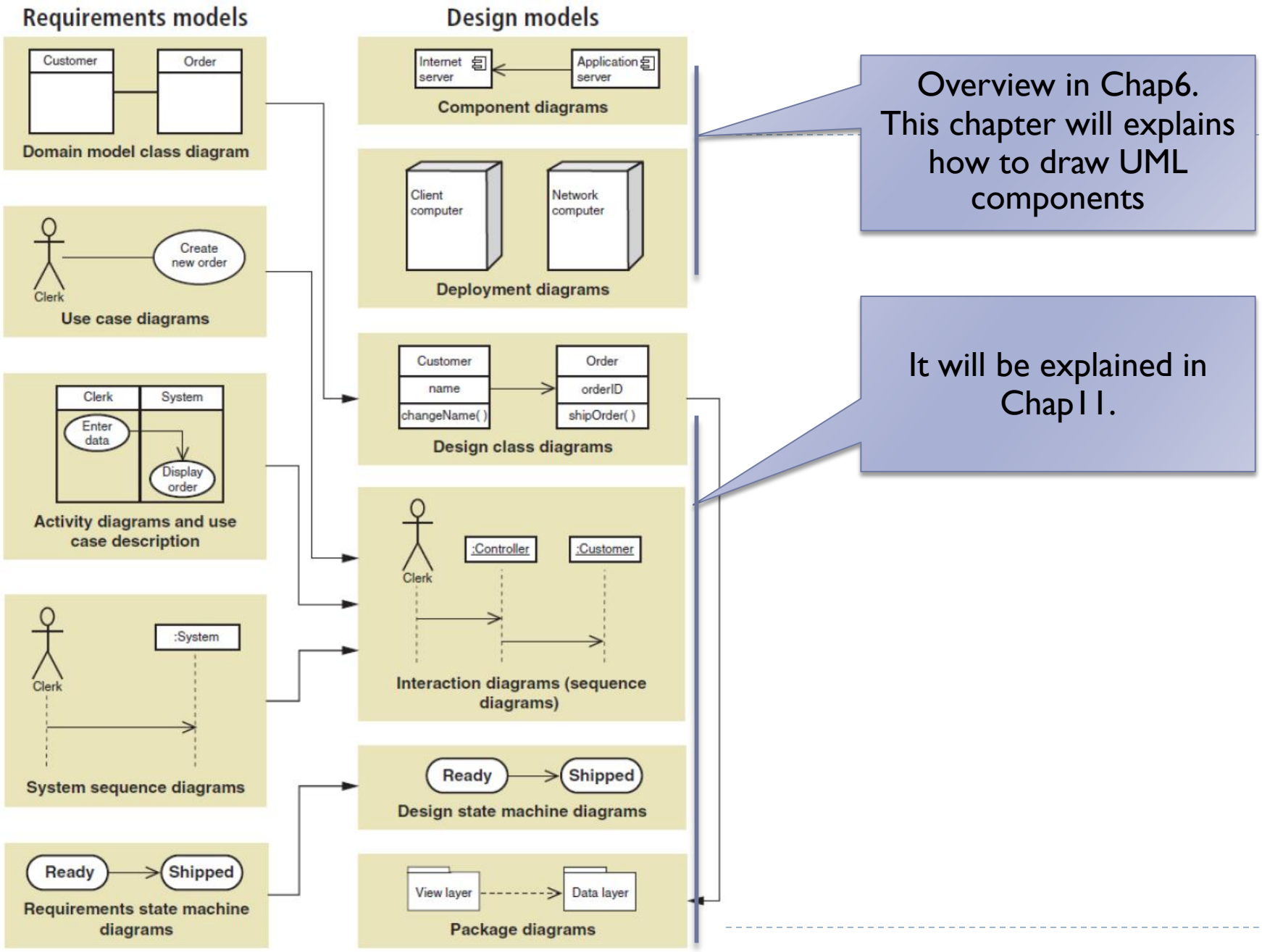
-Three objects
- User interface object
- Problem domain object
- Data access object

-A multilayer architecture by The object don't need to exist on the same system

UML requirement VS. Design models

- Identify
  - All the classes or things
  - Elementary business process
  - Necessary step to carry out a use case
  - Describe document the internal workflow of each use case
  - Related activity diagram show message or data between user and system
  - Track all status of all condition requirement for a class.




Requirements models
Design models
Customer
Order
Domain model class diagram
Internet server
Application server
Component diagrams
Client computer
Network computer
Deployment diagrams
Clerk
Create new order
Use case diagrams
Customer
name
changeName( )
Order
orderID
shipOrder( )
Design class diagrams
Clerk
System
Enter data
Display order
Activity diagrams and use case description
Clerk
:Controller
:Customer
Interaction diagrams (sequence diagrams)
Clerk
:System
System sequence diagrams
Ready
Shipped
Design state machine diagrams
Ready
Shipped
Requirements state machine diagrams
View layer
Data layer
Package diagrams
Overview in Chap6.
This chapter will explains how to draw UML components
It will be explained in Chap11.

10.2 Object-oriented architectural design

- The first step in system design is architectural design
  - The system will be deployed
    - Web app.
    - Win app.
  - Software system
    - Single user system – example spreadsheet, sample accounting .,..
    - Enterprise-level system
      - Client server architectures
        - Network based system
        - Internet based system
      - ….

10.2 Object-oriented architectural design (2)

- Enterprise-level system
  - A system that has shared resources among multiple people or groups in an organization
- Option are client-server or internet based
  - Each presents different issues

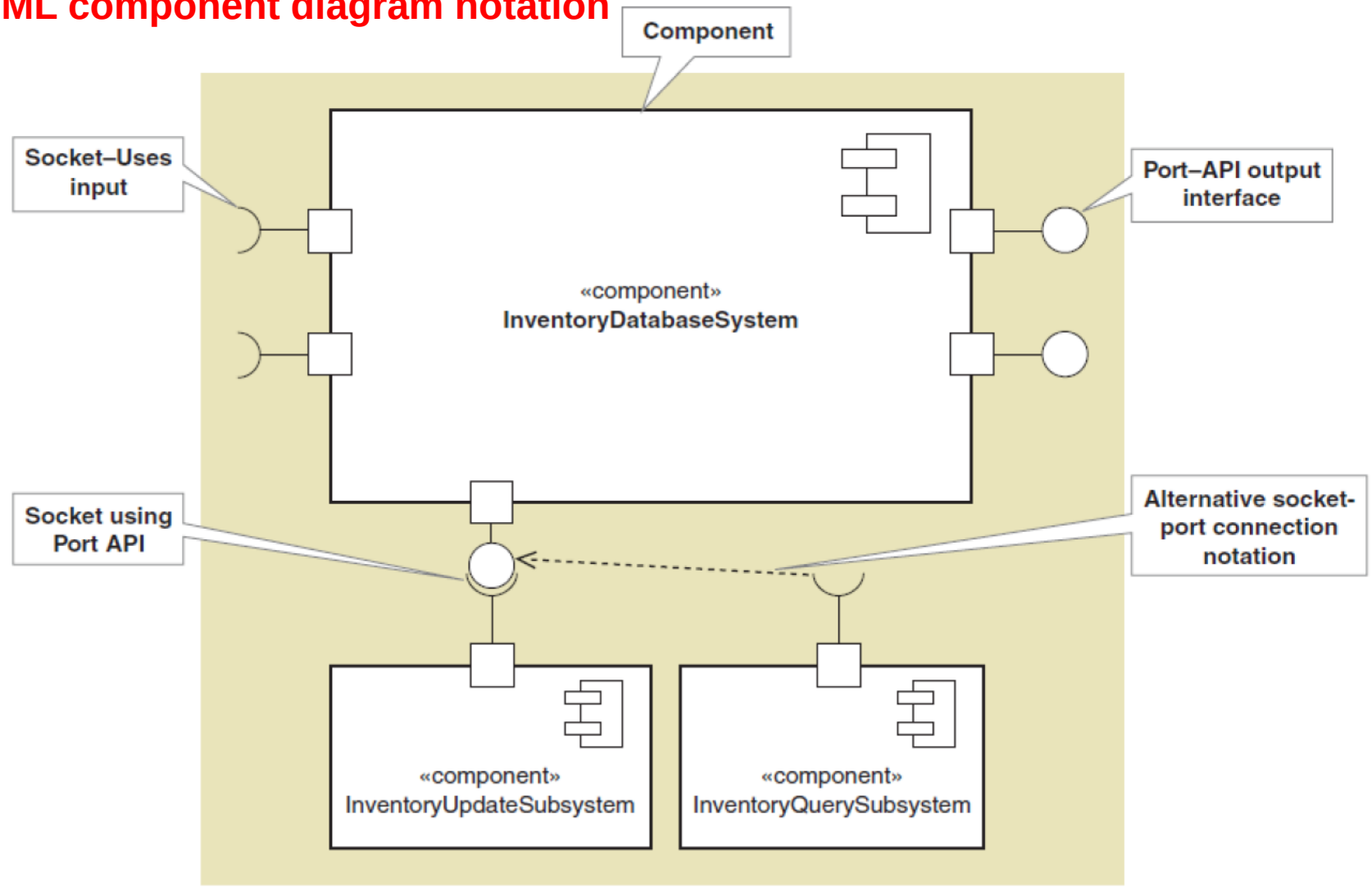
Design Issue	Client/Server Network System	Internet System
State	"Stateful" or state-based system—e.g., client/server connection is long term.	Stateless system—e.g., client/server connection is not long term and has no inherent memory.
Client configuration	Screens and forms that are programmed are displayed directly. Domain layer is often on the client or split between client and server machines.	Screens and forms are displayed only through a browser. They must conform to browser technology.
Server configuration	Application or data server directly connects to client tier.	Client tier connects indirectly to the application server through a Web server.

10.2.1 Component diagram for architectural design

- Component diagram
  - A type of design diagram that shows the overall system architecture and the logical components within it for how the system will be implemented
  - Identifies the logical, reusable, and transportable system components that define the system architecture
  - The essential element of a component diagram is the component element with its API.
- Application program interface (API)
  - The set of public methods that are available to the outside world

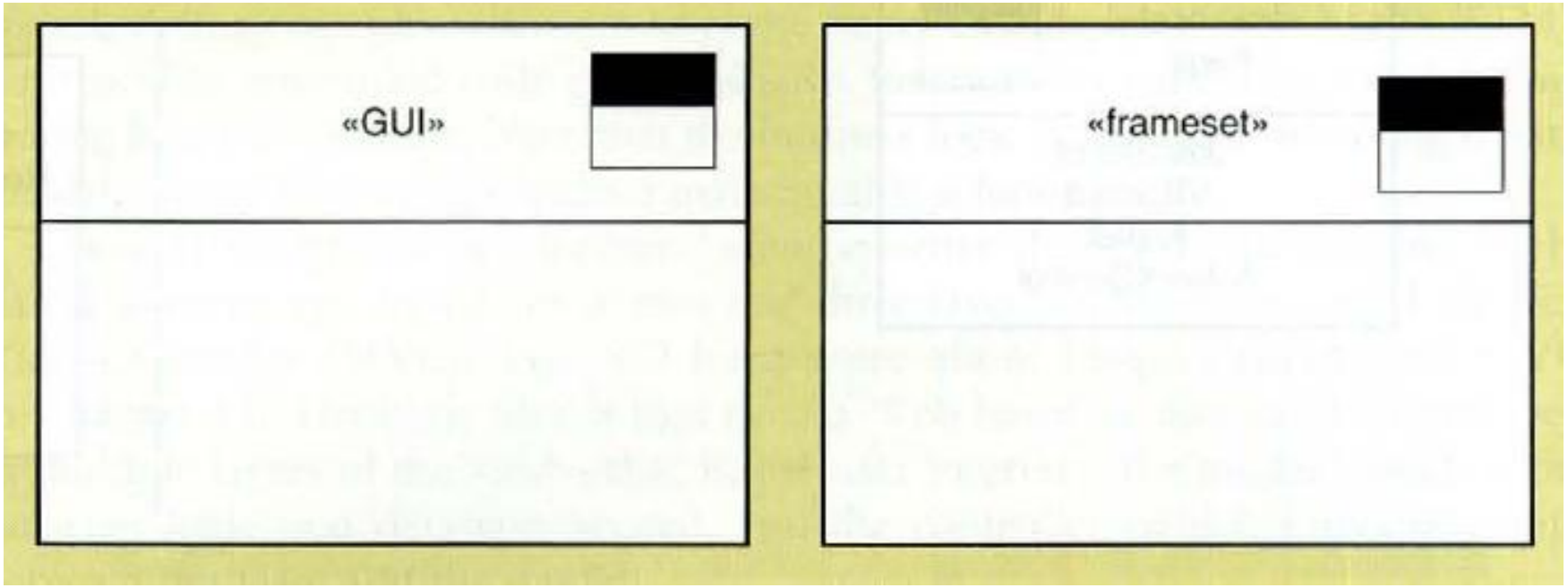
10.2.1
Component diagram for architectural design

UML component diagram notation



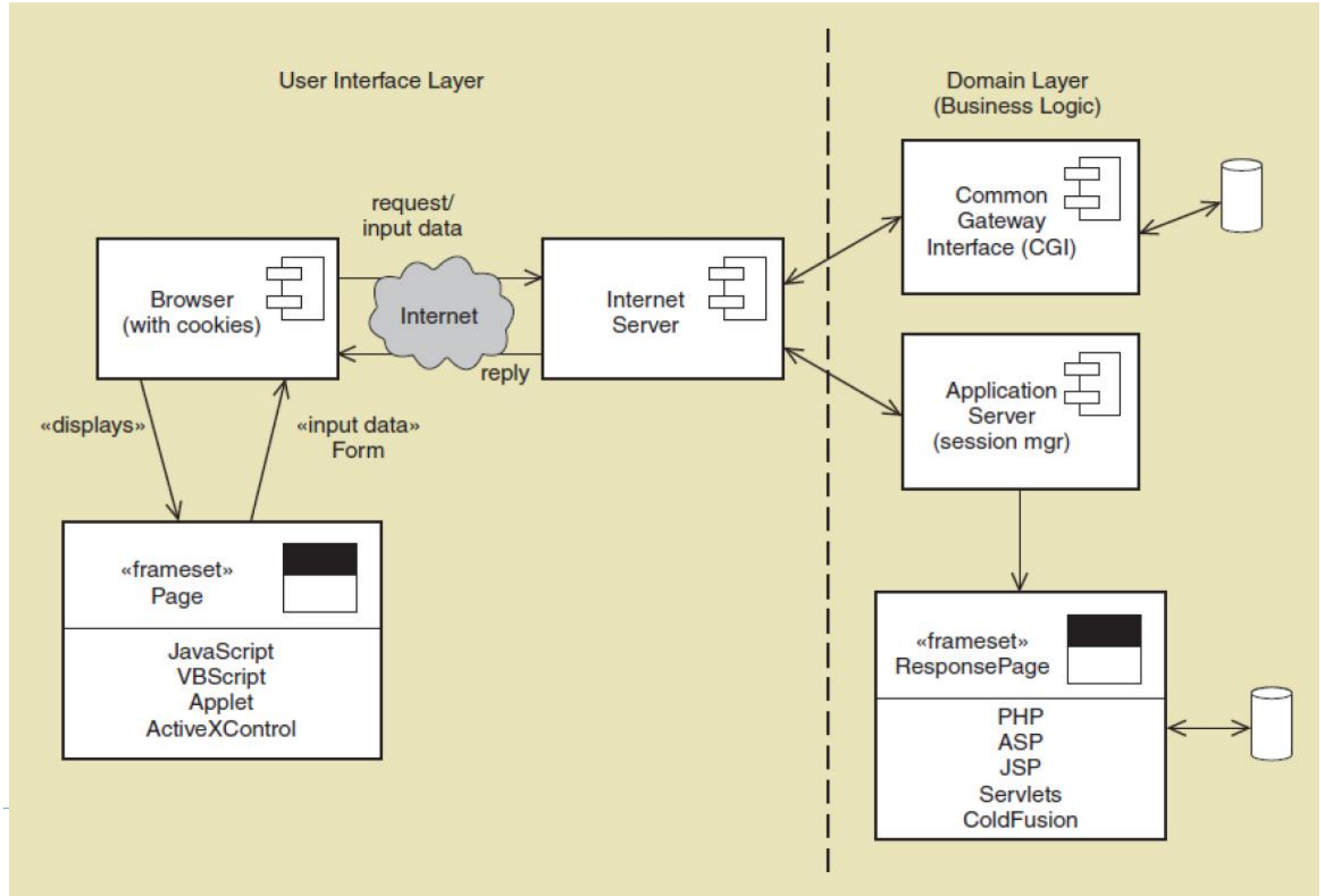
10.2.1 Component diagram for architectural design

UML webpage notation

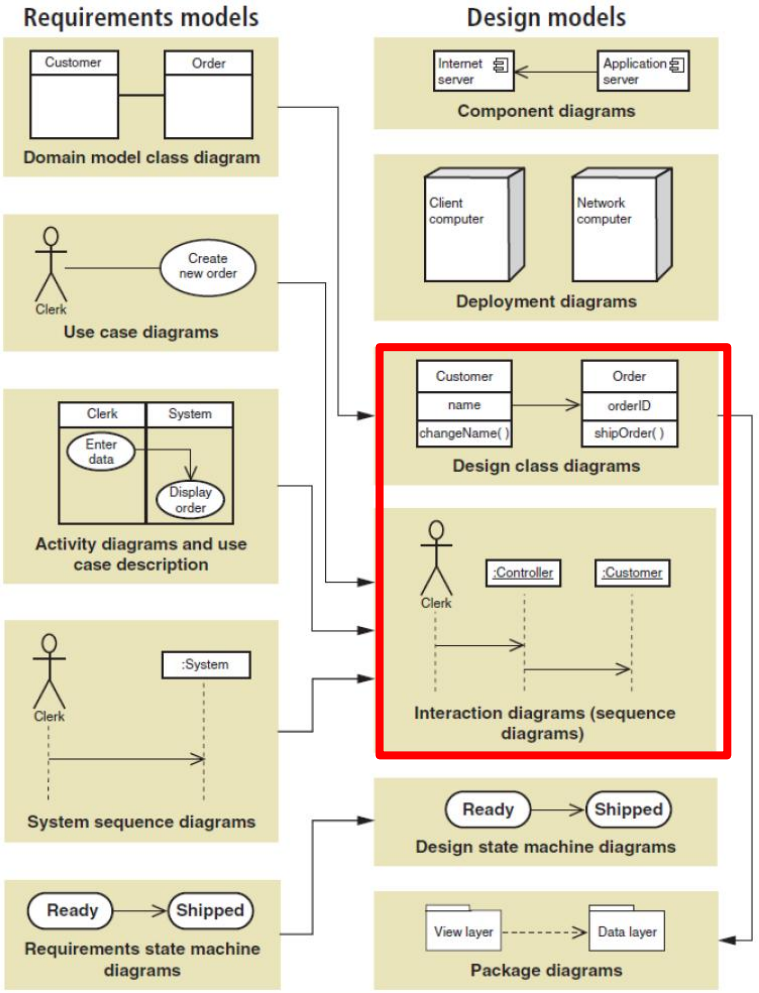


10.2.1 Component diagram for architectural design

Component diagram showing two-layer internet architecture



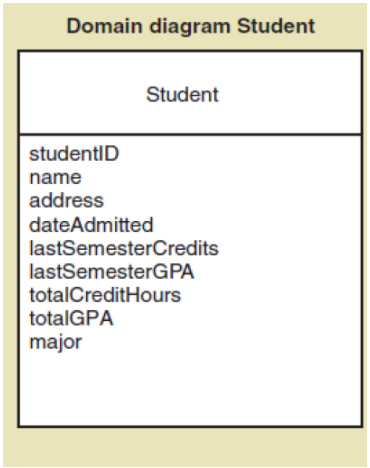
10.3 Fundamental principles of the object-oriented detailed design

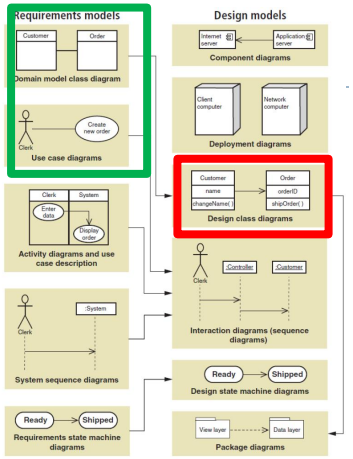


- Both the design class diagram and the interaction diagram are the most important for detail design

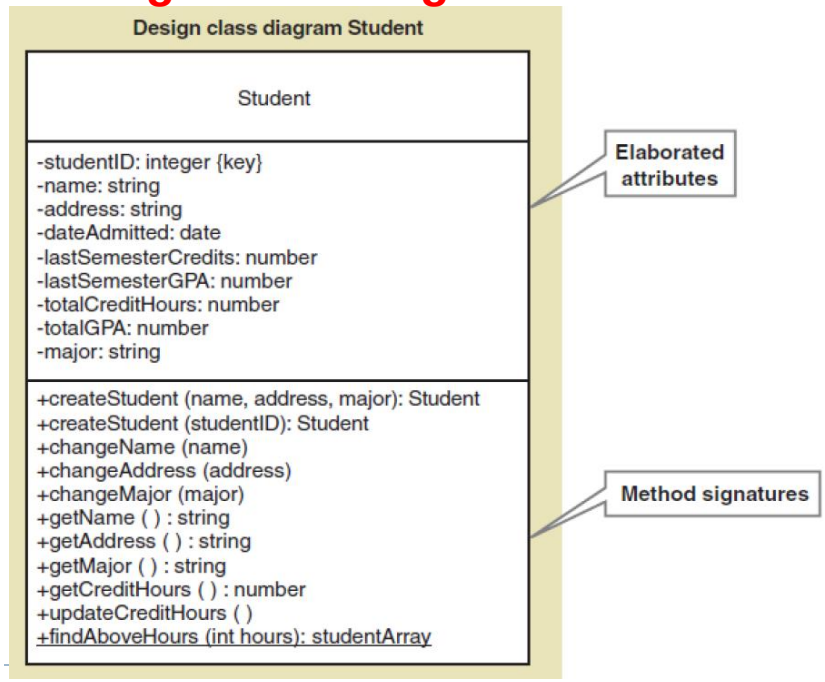
10.3 Fundamental principles of the object-oriented detailed design

Domain model student

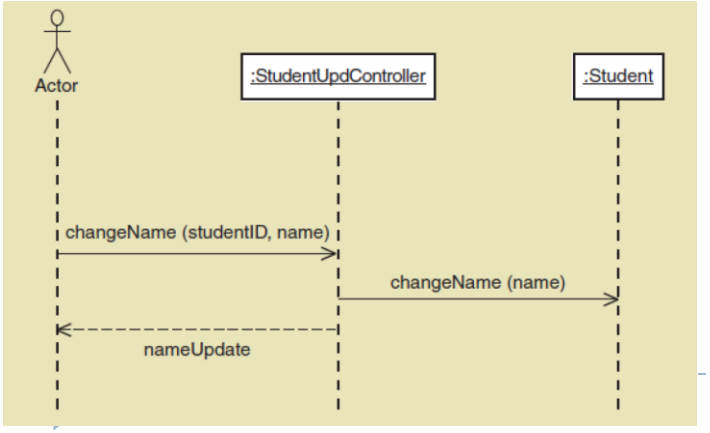




Design class diagram student

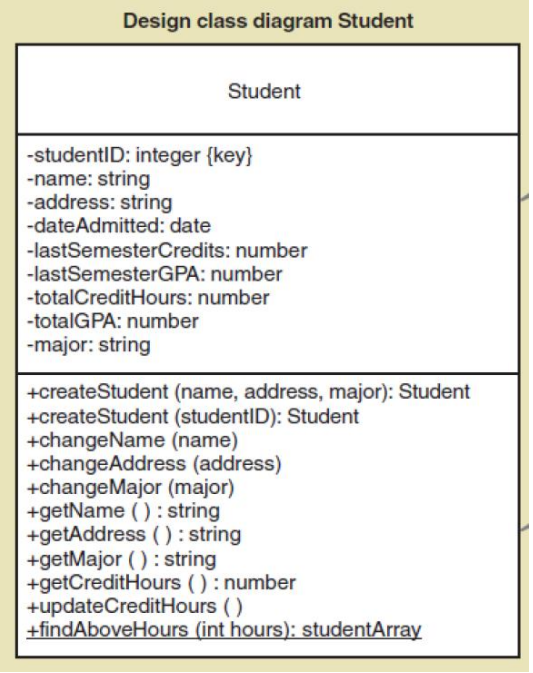


Sequence diagram for updating student name



Write the code for the design class to implement (Java)

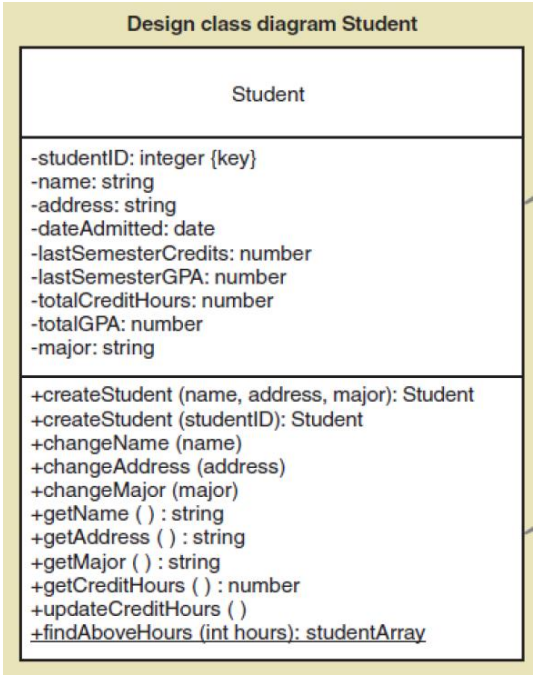
Design class diagram student




```
public class Student
{
	//attributes
	private int studentID;
	private String firstName;
	private String lastName;
	private String street;
	private String city;
	private String state;
	private String zipCode;
	private Date dateAdmitted;
	private float numberCredits;
	private String lastActiveSemester;
	private float lastActiveSemesterGPA;
	private float gradePointAverage;
	private String major;

	//constructors
	public Student (String inFirstName, String inLastName, String inStreet,
		String inCity, String inState, String inZip, Date inDate)
	{
		firstName = inFirstName;
		lastName = inLastName;
		...
	}
	public Student (int inStudentID)
	{
		//read database to get values
	}

	//get and set methods
	public String getFullName ( )
	{
		return firstName + " " + lastName;
	}
	public void setFirstName (String inFirstName)
	{
		firstName = inFirstName;
	}
	public float getGPA ( )
	{
		return gradePointAverage;
	}
	//and so on

	//processing methods
	public void updateGPA ( )
	{
		//access course records and update lastActiveSemester and
		//to-date credits and GPA
	}
}
```

Write the code for the design class to implement (VB.NET)

Design class diagram student




```
Public Class Student

    'attributes
    Private studentID As Integer
    Private firstName As String
    Private lastName As String
    Private street As String
    Private city As String
    Private state As String
    Private zipCode As String
    Private dateAdmitted As Date
    Private numberCredits As Single
    Private lastActiveSemester As String
    Private lastActiveSemesterGPA As Single
    Private gradePointAverage As Single
    Private major As String

    'constructor methods
    Public Sub New(ByVal inFirstName As String, ByVal inLastName As String,
           ByVal inStreet As String, ByVal inCity As String, ByVal inState As String,
           ByVal inZip As String, ByVal inDate As Date)
        firstName = inFirstName
        lastName = inLastName
         ...
    End Sub

    Public Sub New(ByVal inStudentID)
        'read database to get values
    End Sub

    'get and set accessor methods
    Public Function GetFullName() As String
        Dim info As String
        info = firstName & " " & lastName
        Return info
    End Function

    Public Property firstName()
        Get
            Return firstName
        End Get
        Set(ByVal Value)
        firstName = Value
        End Set
    End Property
```

10.3.1 Object-oriented design process

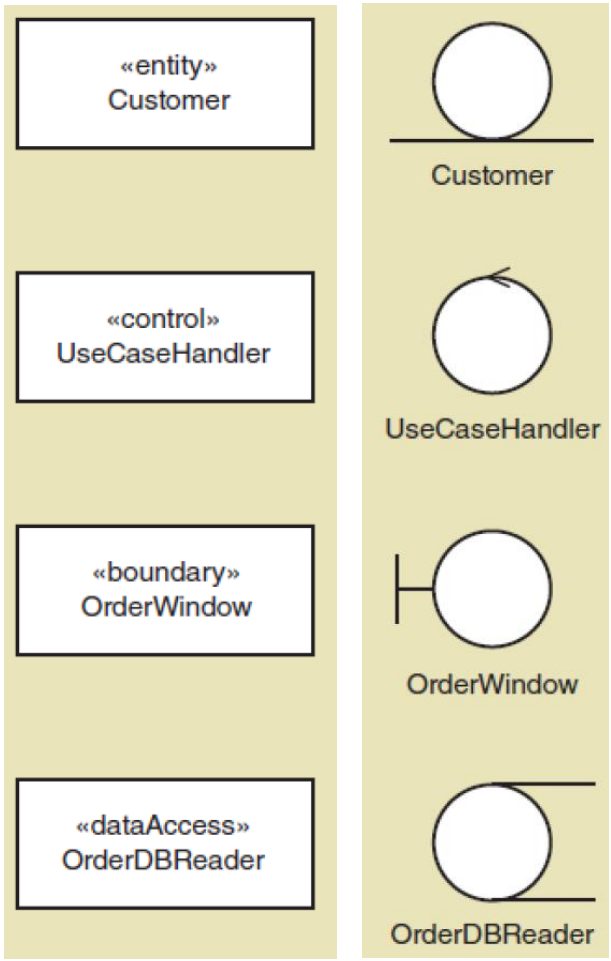
Design Step	Chapter
1. Develop the first-cut design class diagram showing navigation visibility.	10
2. Determine class responsibilities and class collaborations for each use case using class-responsibility-collaboration (CRC) cards.	10
3. Develop detailed sequence diagrams for each use case. (a) Develop the first-cut sequence diagrams. (b) Develop the multilayer sequence diagrams.	11
4. Update the design class diagram by adding method signatures and navigation information using CRC cards and/or sequence diagrams.	11
5. Partition the solution into packages as appropriate.	11

10.4.1 Design class symbols (1)

- stereotype is a notation that uses to categorize a model element as a certain type. It is placed in << >>
- persistent class an class whose objects exist after a system is shut down (data remembered)

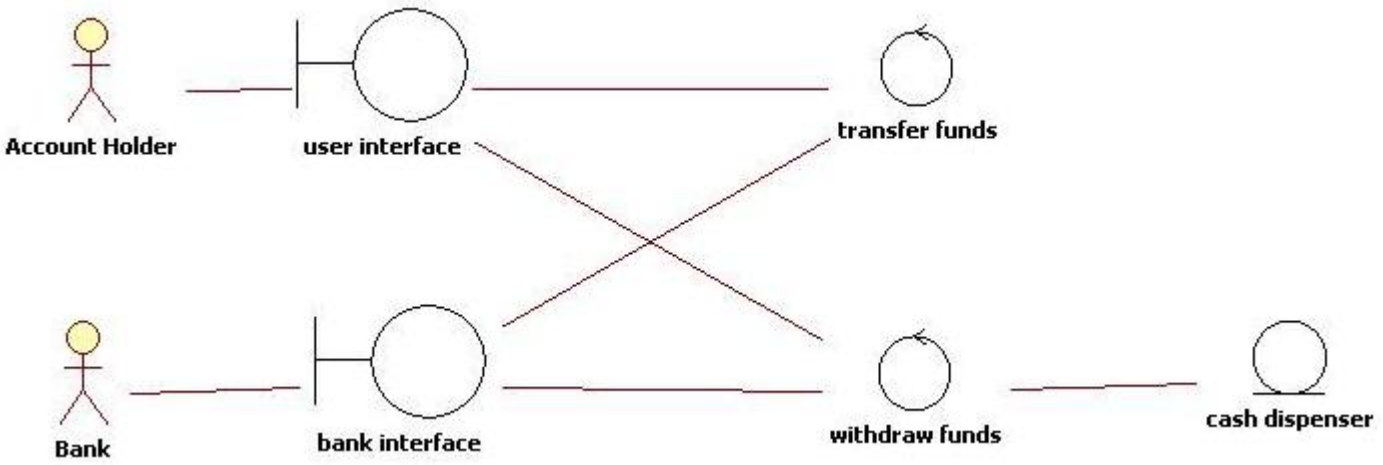
10.4.1 Design class symbols (2)

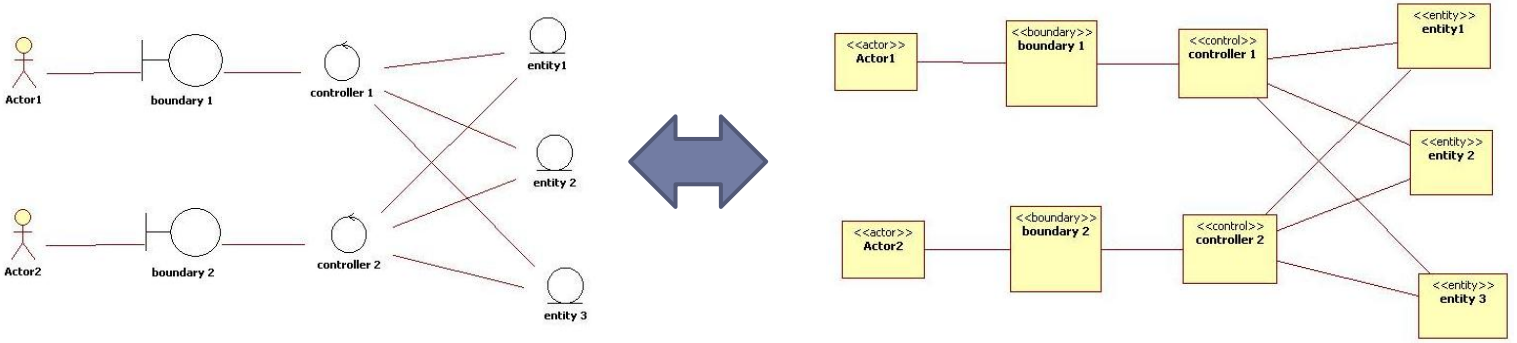
- entity class a design identifier for a problem domain class (usually persistent)
- control class a class that mediates between boundary classes and entity classes, acting as a switchboard between the view layer and domain layer
- boundary class or view class a class that exists on a system's automation boundary, such as an input window form or Web page
- data access class a class that is used to retrieve data from and send data to a database



Example UML design class symbols

ATM

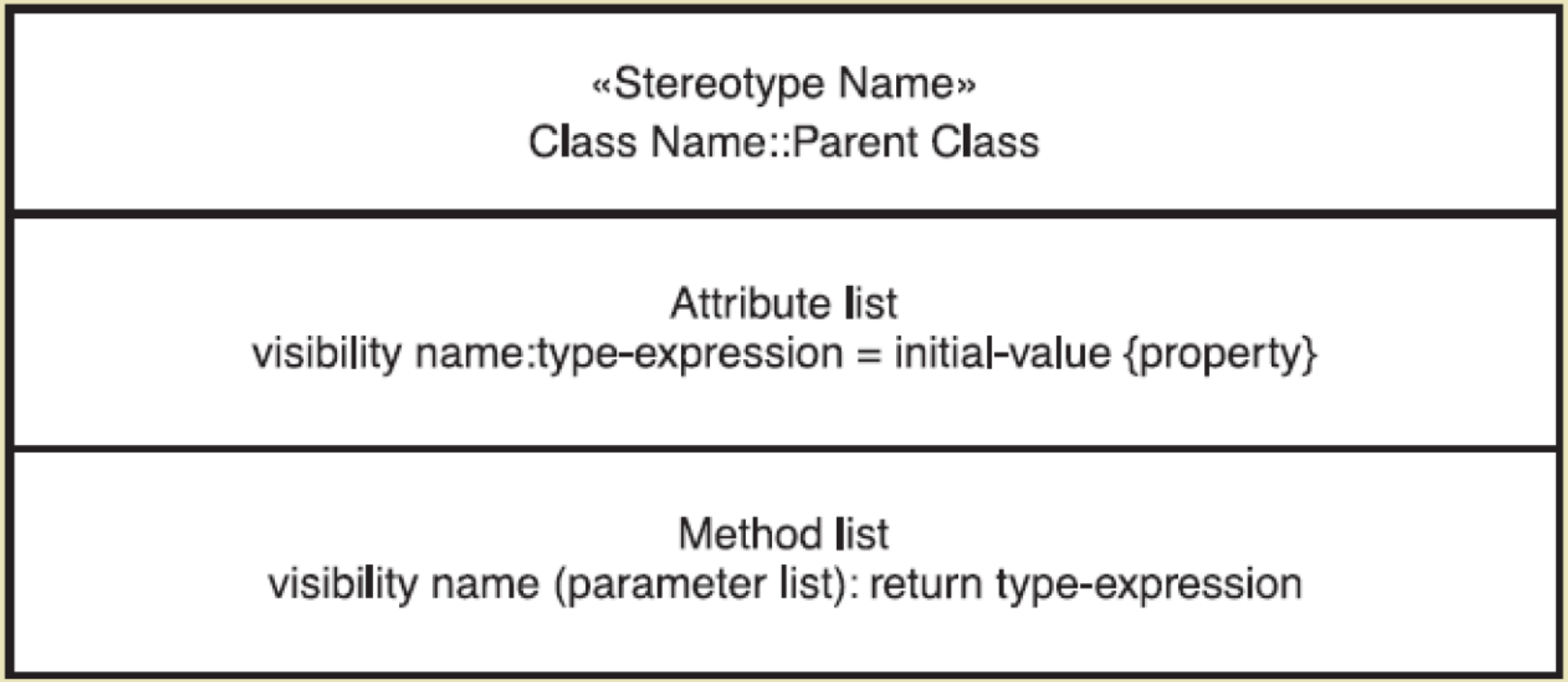




- Ref: http://www.cs.sjsu.edu/~pearce/modules/patterns/enterprise/ecb/ecb.htm

10.4.2
Notation for a design class

- Syntax for name, attributes, and methods



10.4.2
Notation for design classes

- Attributes
  - Visibility—indicates (+ or -) whether an attribute can be accessed directly by another object.
    - private (-) not visibility
    - public (+) visibility
  - Attribute name—Lower case *camelback* notation
  - Type expression—class, string, integer, double, date
  - Initial value—if applicable the default value
  - Property—if applicable, such as {key}
  - Examples:

- accountNo: String {key}
- startingJobCode: integer = 01

10.4.2
Notation for design classes

- Methods
  - Visibility—indicates (+ or -) whether an method can be invoked by another object.
    - private (-) not visibility
    - public (+) visibility
  - Method name—Lower case *camelback*, verb-noun
  - Parameters—variables passed to a method
  - Return type—the type of the data returned
  - Examples:

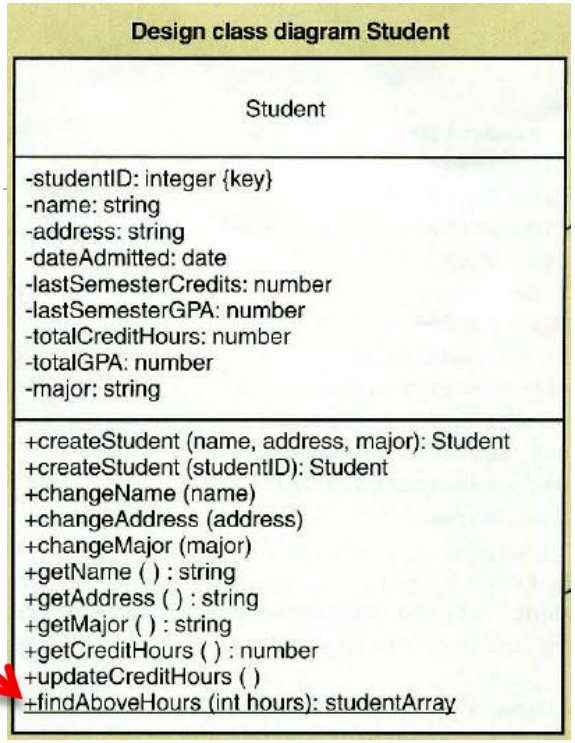
+setName(fName, lName) : void (void is usually let off)

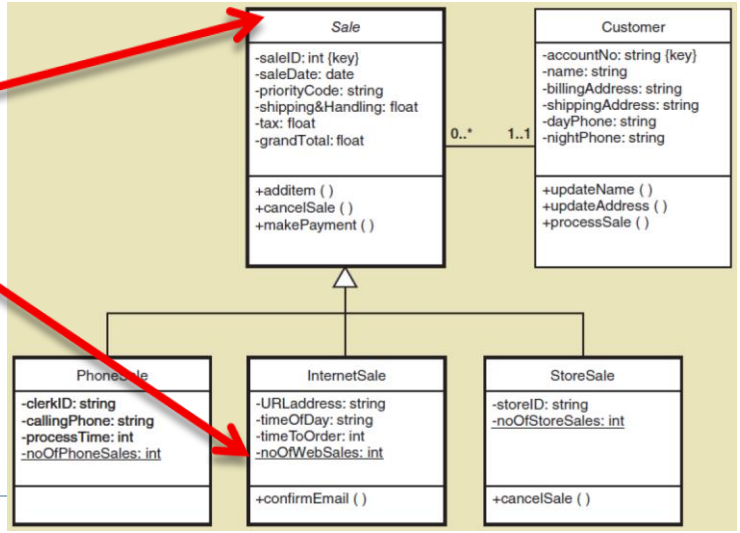
+getName(): string (what is returned is a string)

-checkValidity(date) : int (assuming int is a returned code)

10.4.2 Notation for design classes

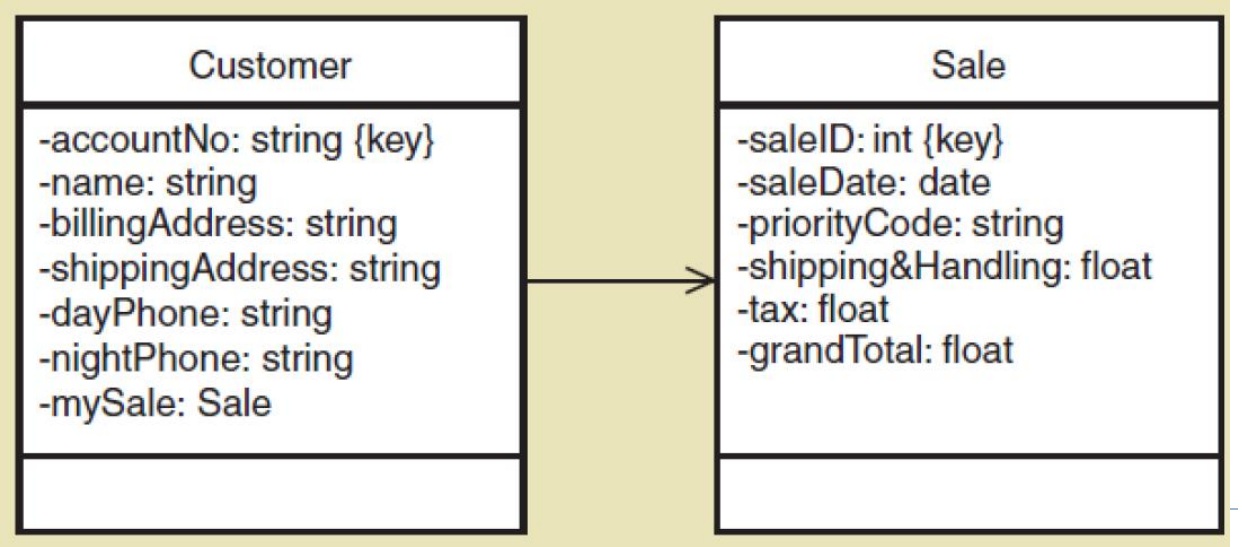
- Class level method—a method that is associated with a class instead of with objects of the class, Underline it.
  - +findStudentsAboveHours(hours): Array
- Class level attribute—an attribute that contains the same value for all object in the system. Underline it.
  - -noOfPhoneSales: int
- Abstract class– class that can't be instantiated.
  - Only for inheritance. Name in *Italics*.
- Concrete class—class that can be instantiated.





10.4.2
Notation for design classes

- Navigation Visibility
  - The ability of one object to view and interact with another object
  - Accomplished by adding an object reference variable to a class.
  - Shown as an arrow head on the association line—customer can find and interact with sale because it has mySale reference variable



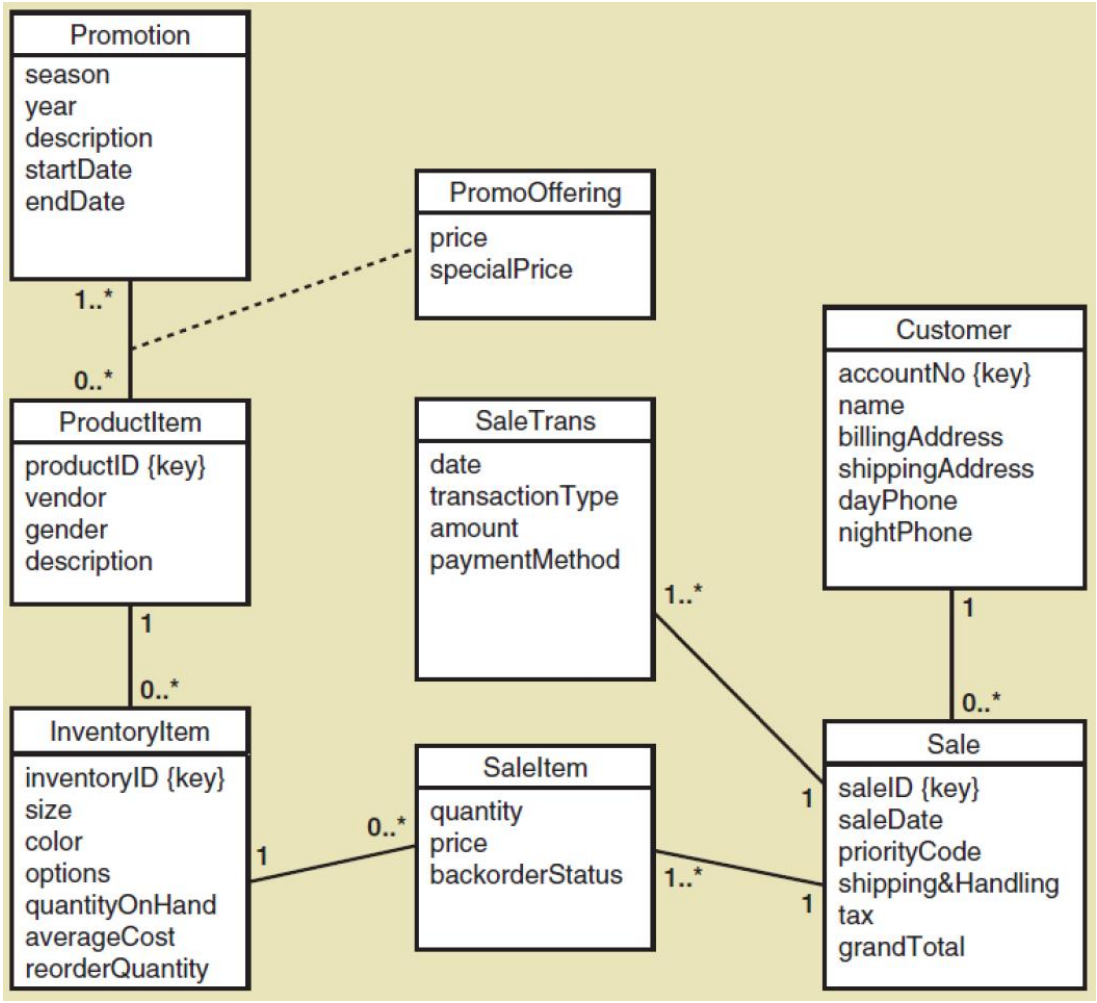
Navigation visibility guidelines

- One-to-many associations that indicate a superior/subordinate relationship are usually navigated from the superior to the subordinate
- Mandatory associations, in which objects in one class can't exist without objects of another class, are usually navigated from the more independent class to the dependent
- When an object needs information from another object, a navigation arrow might be required, pointing either to the object itself or to its parent in a hierarchy.
- Navigation arrows may be bidirectional.

First cut design class diagram

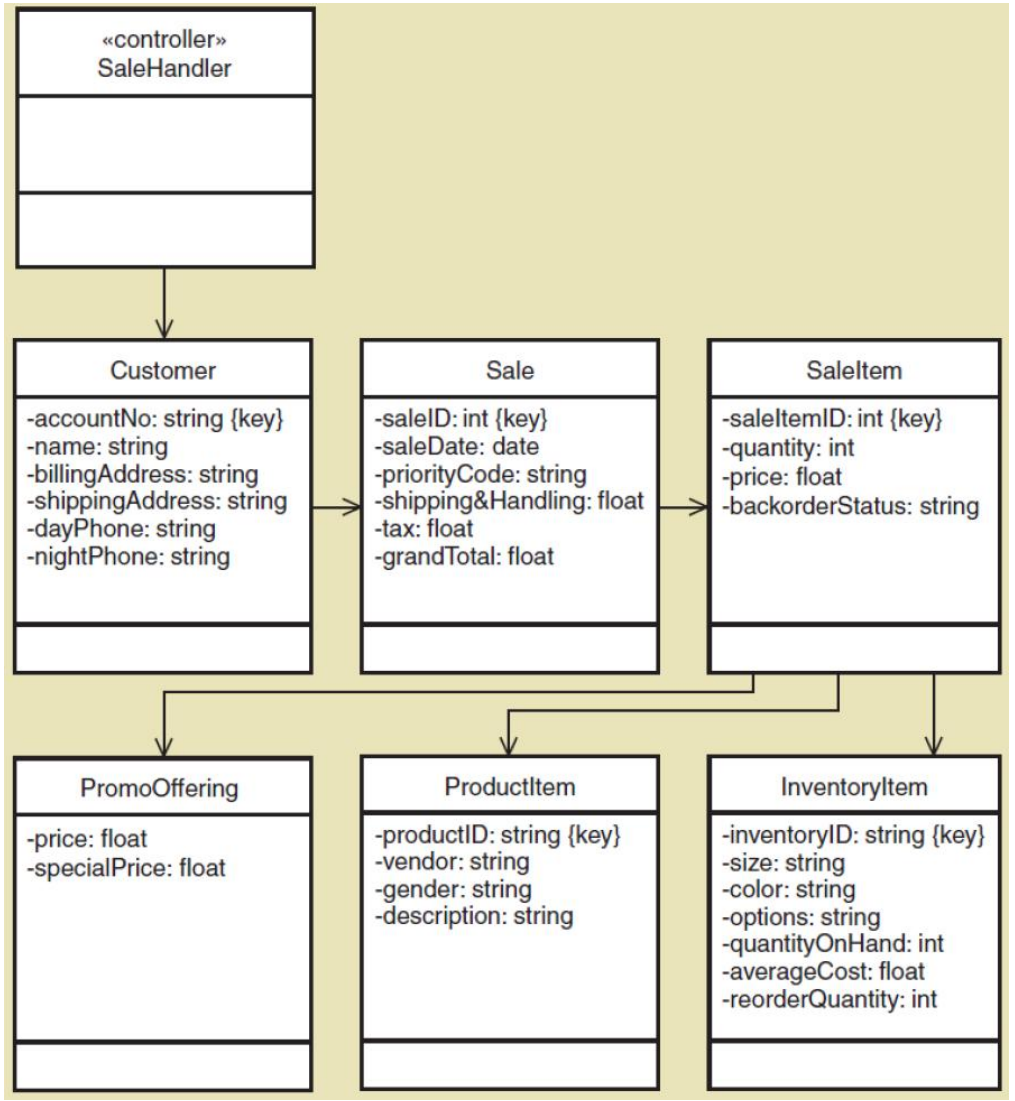
- Proceed use case by use case, adding to the diagram
- Pick the domain classes that are involved in the use case (see preconditions and post conditions for ideas)
- Add a controller class to be in charge of the use case
- Determine the initial navigation visibility requirements using the guidelines and add to diagram
- Elaborate the attributes of each class with visibility and type
- Note that often the associations and multiplicity are removed from the design class diagram as in text to emphasize navigation, but they are often left on

Start with domain class diagram
RMO sales subsystem



Create first cut design class diagram

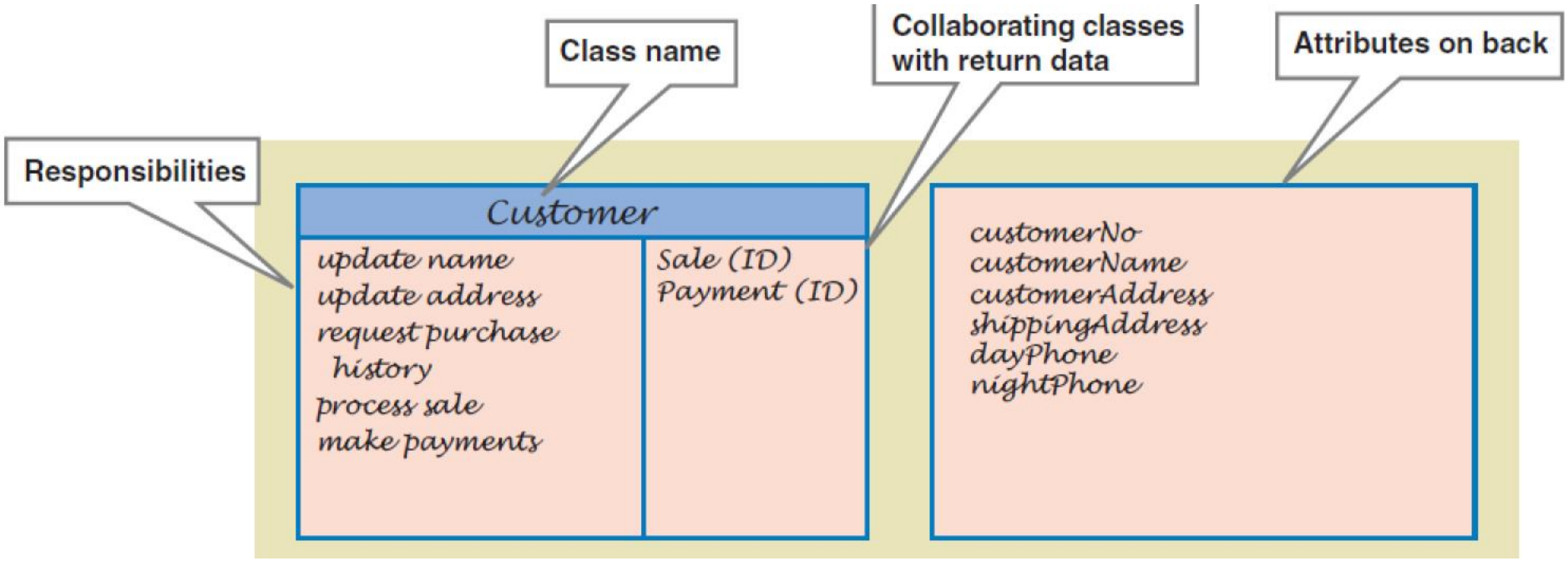
- Use case create phone sale with controller added



10.5 Design with CRC cards

- CRC Cards—Classes, Responsibilities, Collaboration Cards help to identify responsibilities of the class and the set of classes.
- Usually a manual process done in a brainstorming session
  - 3 X 5 note cards
  - One card per class
  - Front has responsibilities and collaborations
  - Back has attributes needed

Example of CRC card



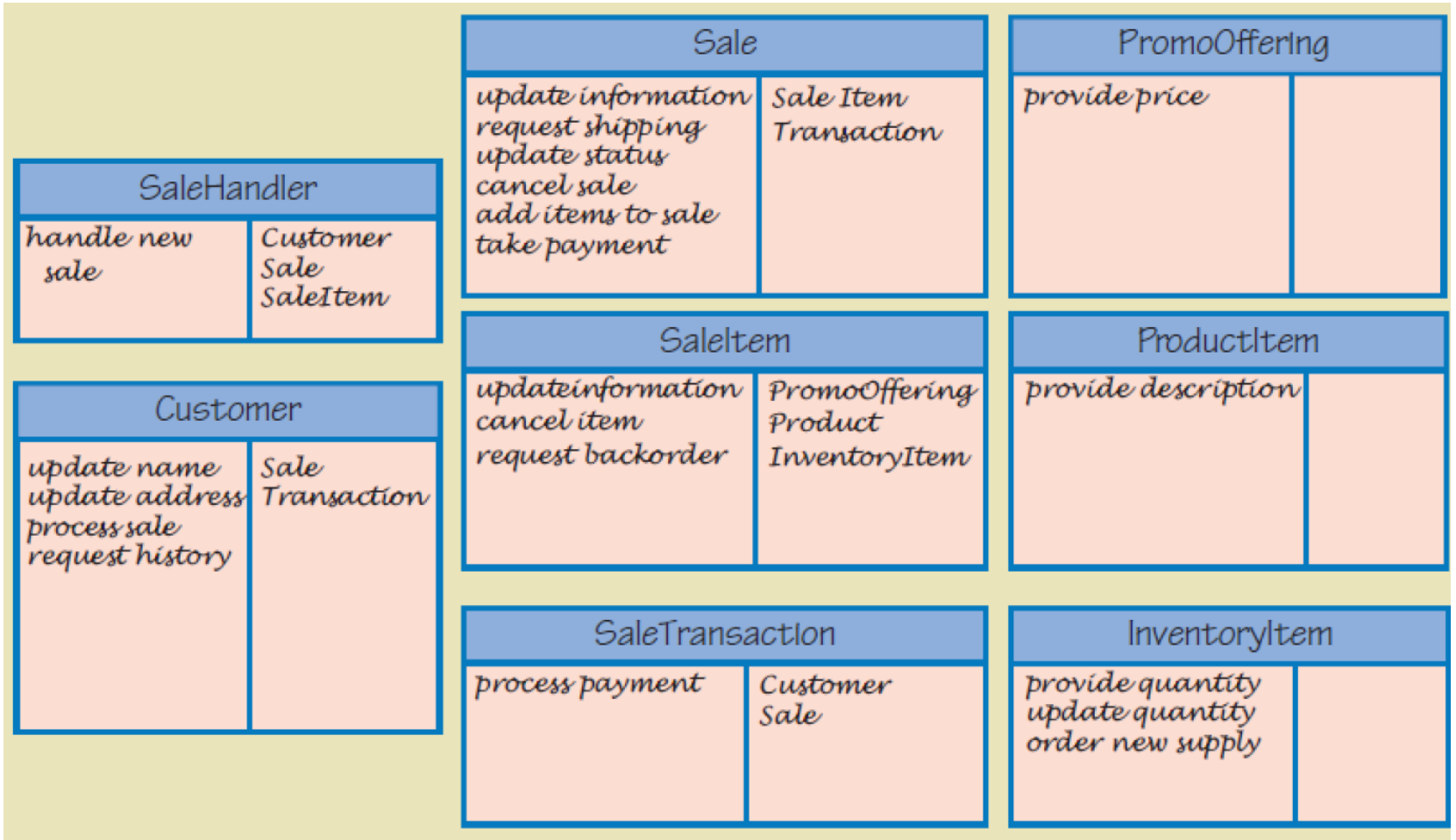
CRC cards procedure

- Because the process is to design, or realize, a single use case, start with a set of unused CRC cards. Add a controller class (Controller design pattern).
- Identify a problem domain class that has primary responsibility for this use case that will receive the first message from the use case controller. For example, a Customer object for new sale.
- Use the first cut design class diagram to identify other classes that must collaborate with the primary object class to complete the use case.
- Have use case descriptions and SSDs handy

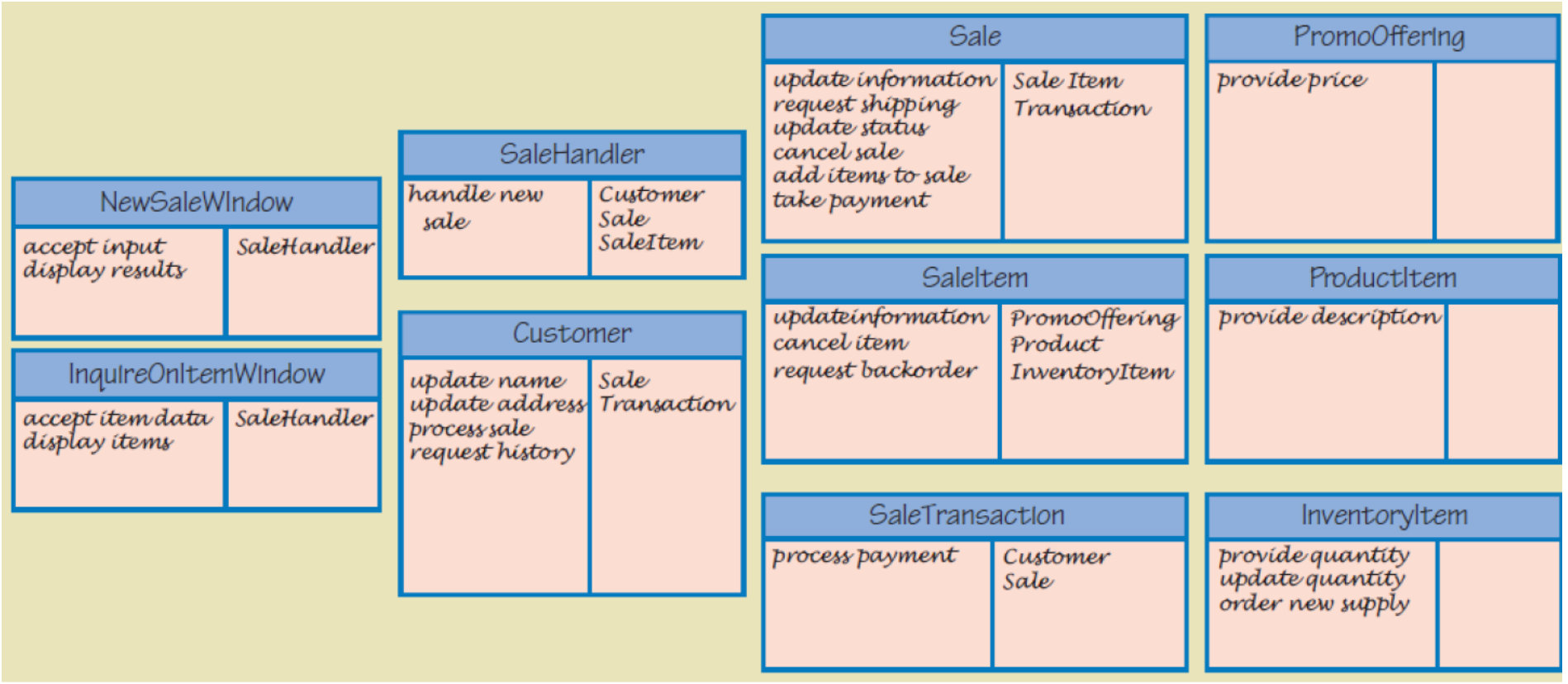
CRC card procedure (2)

- Start with the class that gets the first message from the controller. Name the responsibility and write it on card.
- Now ask what this first class needs to carry out the responsibility. Assign other classes responsibilities to satisfy each need. Write responsibilities on those cards.
- Sometimes different designers play the role of each class, acting out the use case by verbally sending messages to each other demonstrating responsibilities
- Add collaborators to cards showing which collaborate with which. Add attributes to back when data is used
- Eventually, user interface classes or even data access classes can be added

CRC cards results several use cases

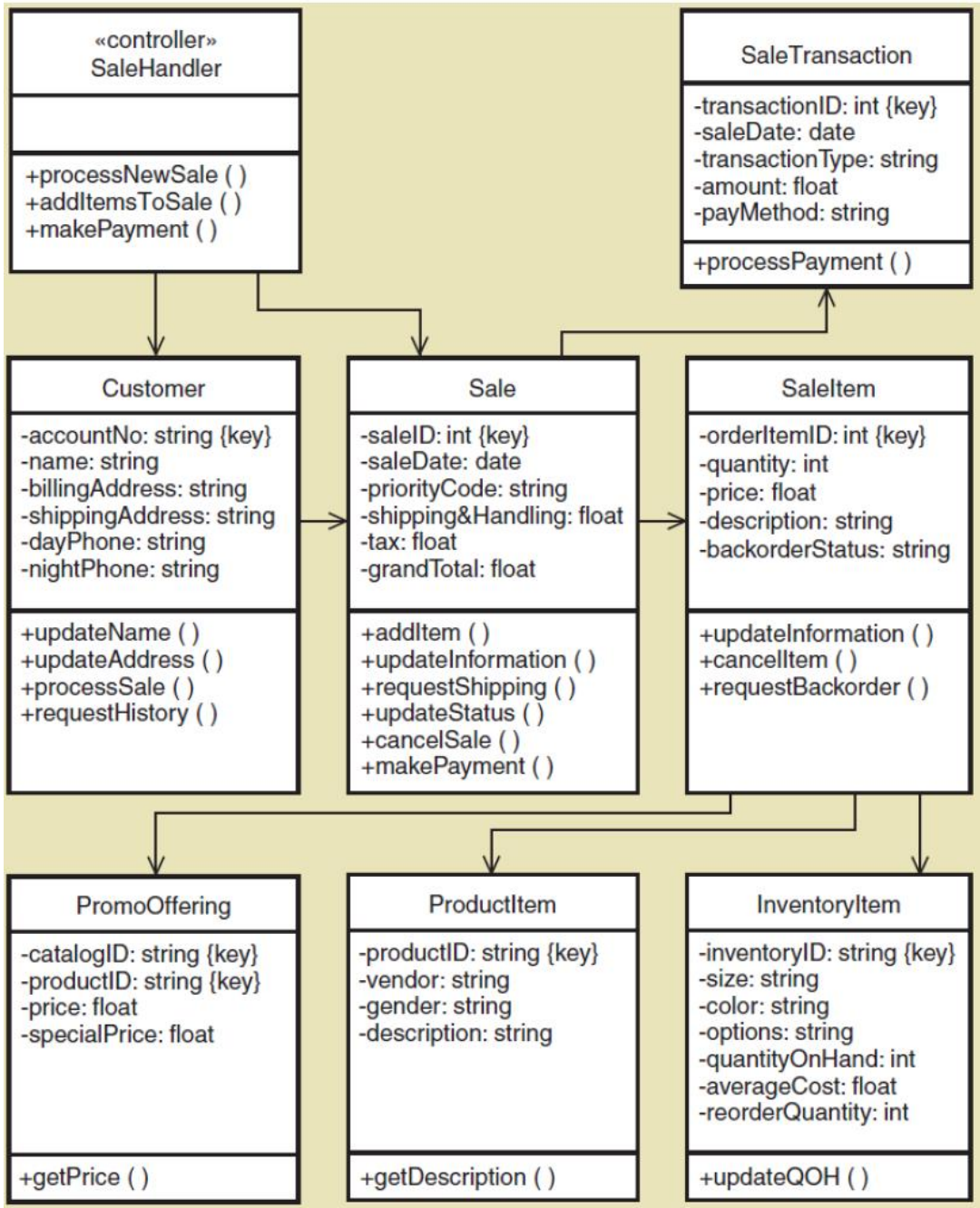


CRC cards results
Adding in user interface layer



CRC cards

- Update design class diagram based on CRC results
- Responsibilities become methods
- Arguments and return types not yet added



10.6 Fundamental design principle

- Coupling
  - A quantitative measure of how closely related classes are linked (tightly or loosely coupled)
- Cohesion
  - A quantitative measure of the focus or unity of purpose within a single class (high or low cohesiveness)
- Protection from variations
  - A design principle that states parts of a system unlikely to change are separated (protected) from those that will surely change
- Indirection
  - A design principle that states an intermediate class is placed between two classes to decouple them but still link them
- Object responsibility
  - A design principle that states objects are responsible for carrying out system processing (Knowing and Doing)

Summary

- Architectural design
- Detail design of software proceeds
- Detailed design models.
- Design class diagrams
- Key issues are attribute elaboration and adding methods. Method signatures include visibility, method name, arguments, and return types.
- abstract vs. concrete classes, navigation visibility, and class level attributes and methods,
- CRC Cards technique