Fundamentals of Object-Oriented Design

CSIT883 System Analysis and Project Management

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

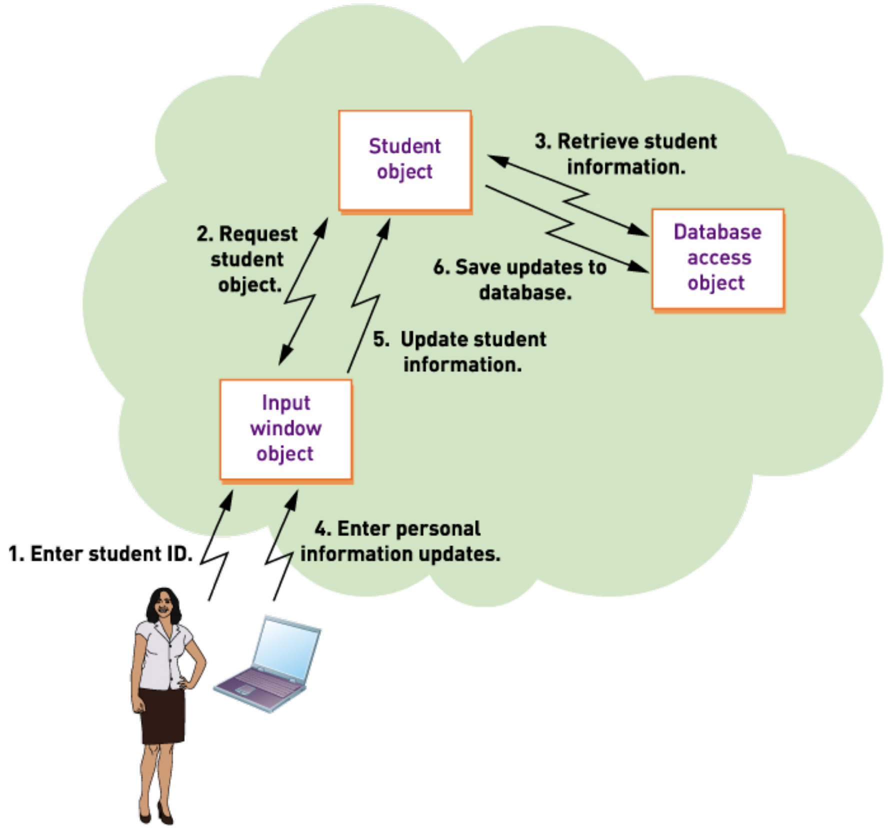
Outline

Object-Oriented Design: Bridging from Analysis to Implementation

- Focus: Software classes and methods
- What is Object-Oriented Design?
  - Process by which a set of detailed OO design models are built to be used for coding
- Strength: Requirements models from systems analysis are extended to design models.
- Design models are created in parallel to actual coding/implementation with iterative SDLC

Object-Oriented Program

- An object-oriented program consists of sets of computing ***objects***.
  - Each object has data and program logic encapsulated within itself.
- The structure of the program logic and data fields defines a ***class***.
- *Instantiation*
  - Creation of an object in memory based on the template provided by the class

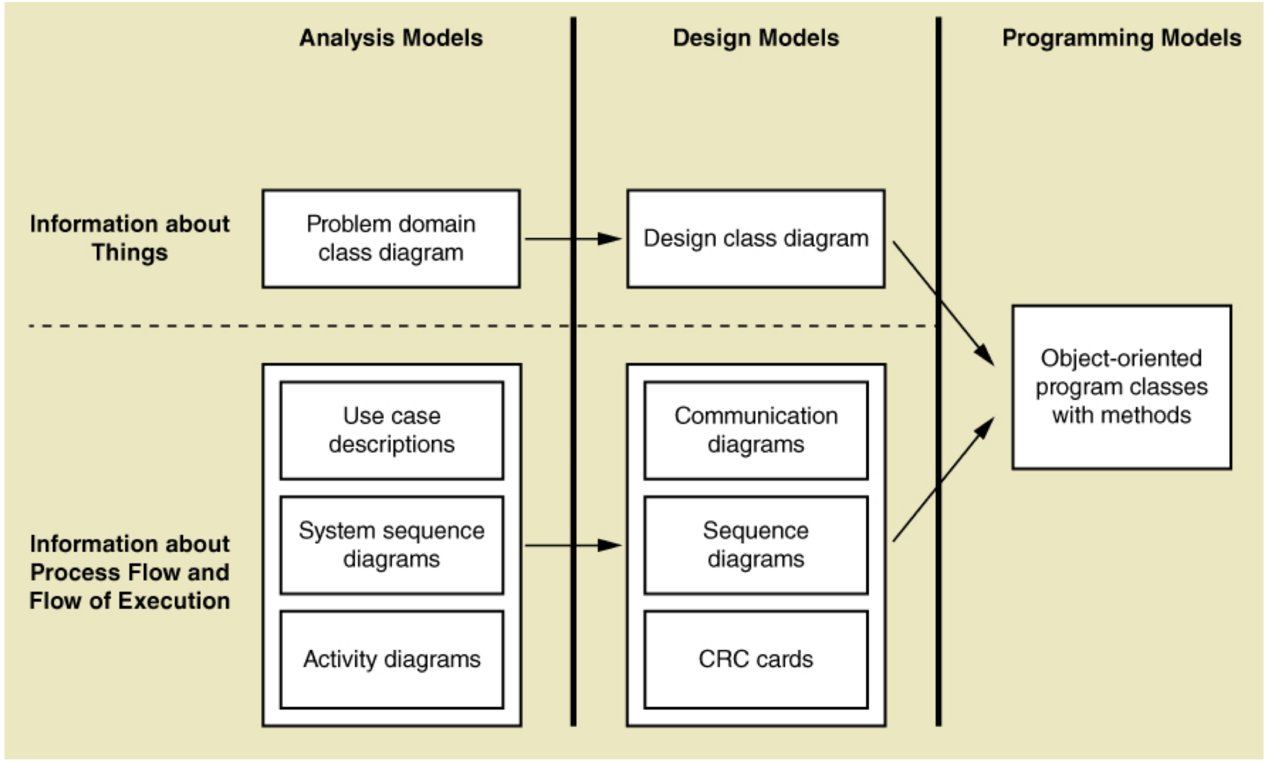

Student object
3. Retrieve student information.
Database access object
2. Request student object.
6. Save updates to database.
5. Update student information.
Input window object
1. Enter student ID.
4. Enter personal information updates.

Object-Oriented Programs

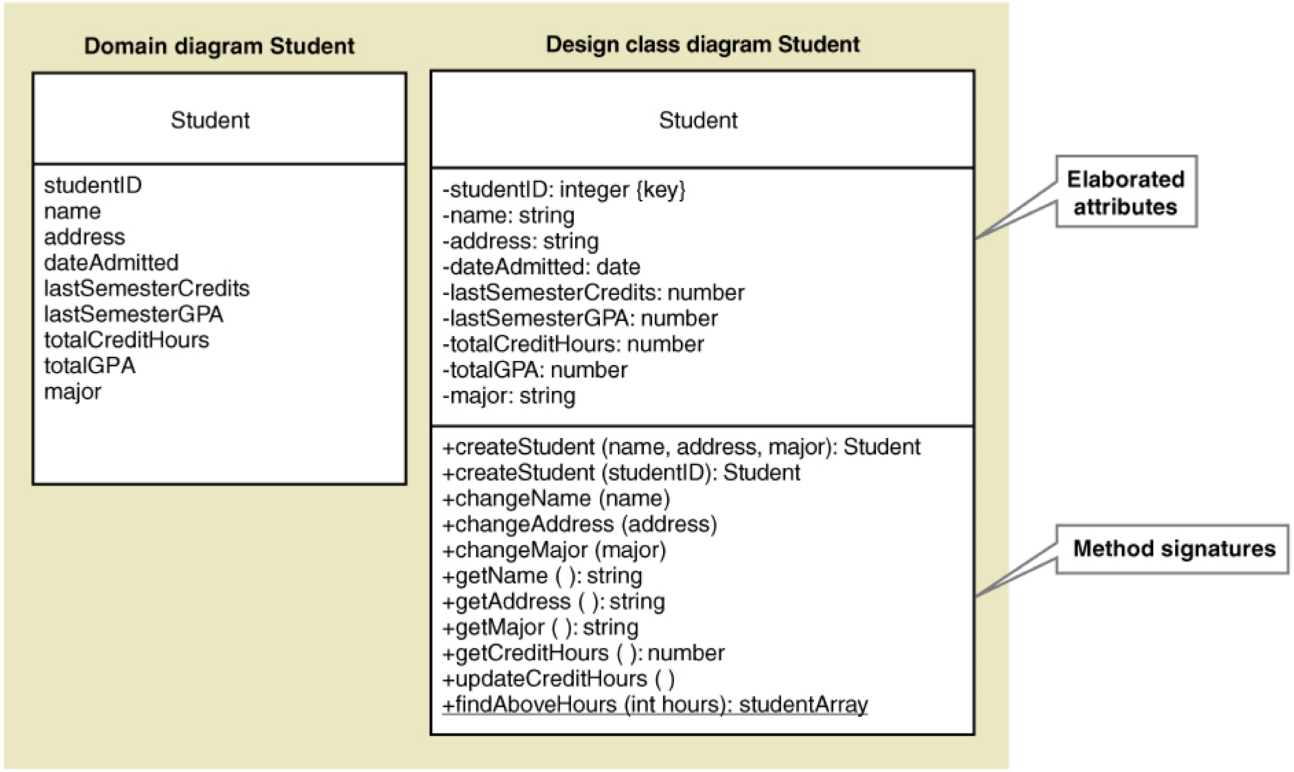
- Method
  - Fragments of the program logic (i.e. functions)
  - Called or invoked through messages or when a class is instantiated.

```
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;
        }
```

Analysis Models to Design Models



Introduction to the Design Models



```
Student
---------------------------
studentID
name
address
dateAdmitted
lastSemesterCredits
lastSemesterGPA
totalCreditHours
totalGPA
major
```

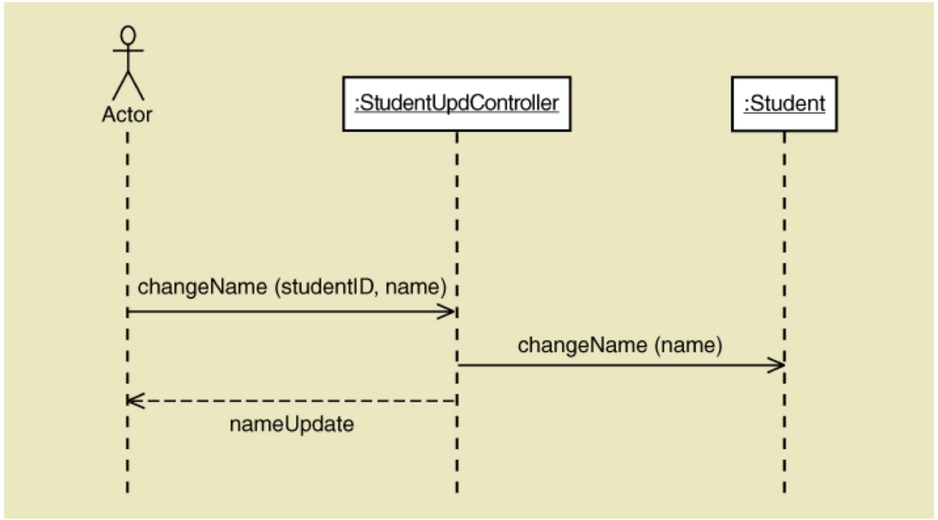
```
Student
---------------------------
-studentID: integer {key}
-name: string
-address: string
-dateAdmitted: date
-lastSemesterCredits: number
-lastSemesterGPA: number
-totalCreditHours: number
-totalGPA: number
-major: string
---------------------------
+createStudent (name, address, major): Student
+createStudent (studentID): Student
+changeName (name)
+changeAddress (address)
+changeMajor (major)
+getName ( ): string
+getAddress ( ): string
+getMajor ( ): string
+getCreditHours ( ): number
+updateCreditHours ( )
+findAboveHours (int hours): studentArray
```

Introduction to the Design Models

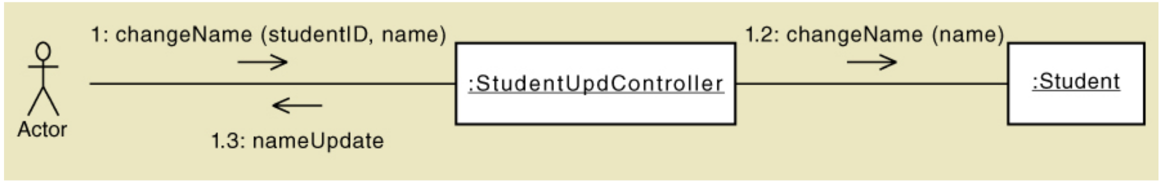
Documenting the flow of execution of a particular use case:

- **Sequence Diagram**
- **Communication Diagram**
- **Class-Responsibility-Collaboration (CRC) card**

Sequence Diagram for "Update student name":

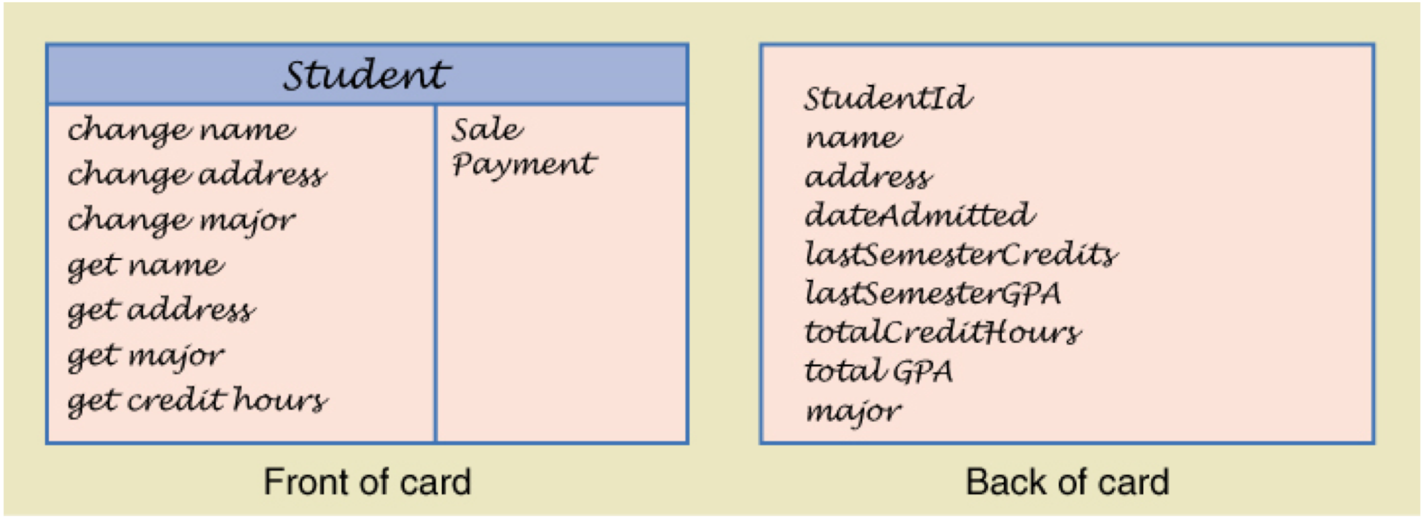


Communication Diagram for "Update student name":



Introduction to the Design Models

- Sample CRC card for the Student class



Student	
change name change address change major get name get address get major get credit hours	Sale Payment

Front of card

StudentId name address dateAdmitted lastSemesterCredits lastSemesterGPA totalCreditHours total GPA major

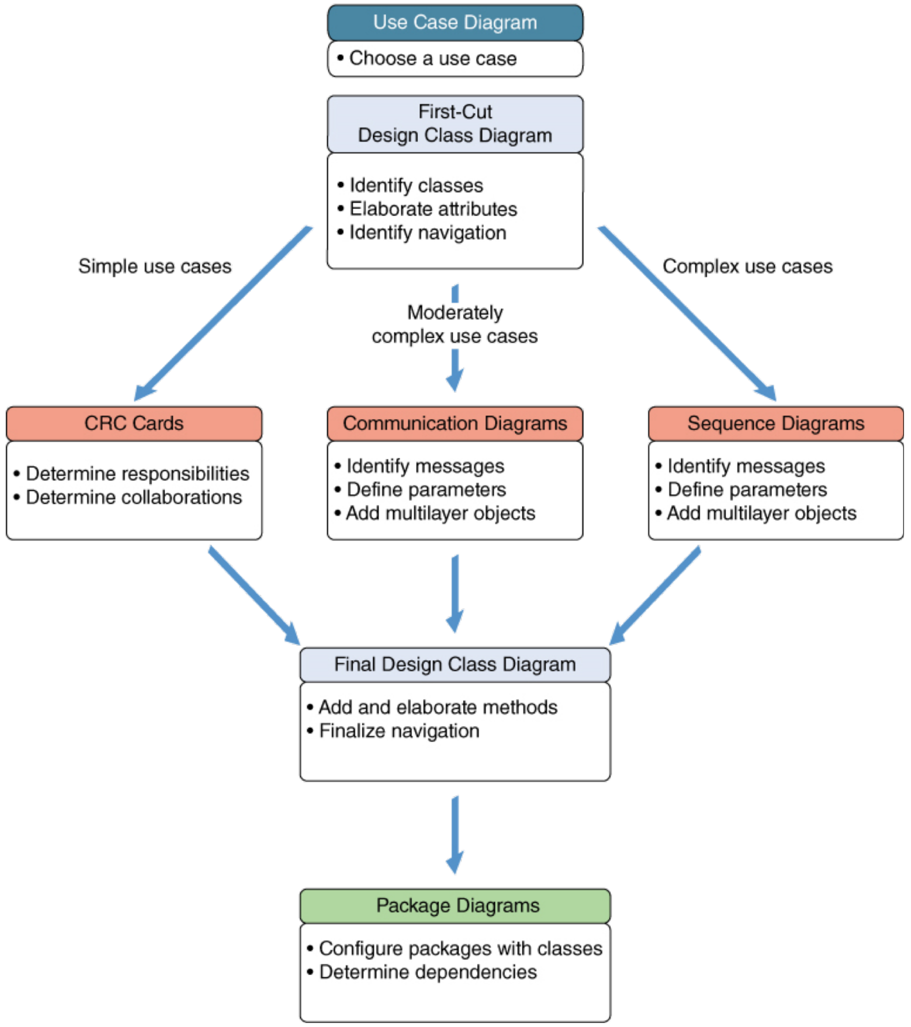
Back of card

Steps of Object-Oriented Design

- **Object-oriented design**
  - The process to identify the classes, their methods and the messages required for a use case
  - Usually designs classes in three layers: user interface, problem domain, and database access layers
- **Use case driven**
  - Design is carried out use case by use case
- Always build software design models that can assist the development of accurate and robust software!

Steps of OO Design

- Three paths
  - Simple use case use CRC Cards
  - Medium use case use Communication Diagram
  - Complex use case use Sequence Diagram



Fundamentals of Object-Oriented Design

CSIT883 System Analysis and Project Management

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

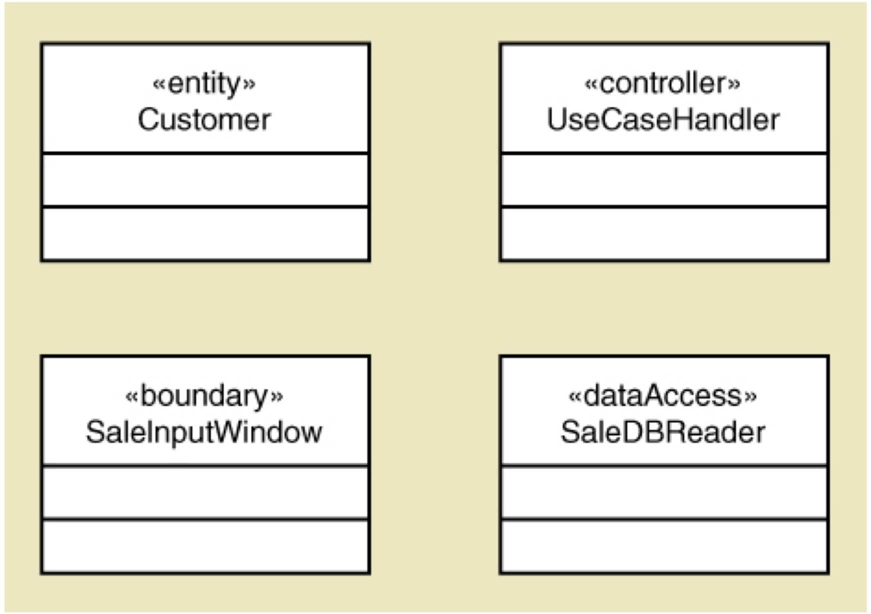
Outline

Design Classes and the Design Class Diagram

- The primary source of information for Design Class Diagram is the problem domain model (i.e., DMCD).
- Additional information (e.g., elaborated attributes method signatures) are added for OO design
- Additional objects (e.g., window objects and database access objects) are added.

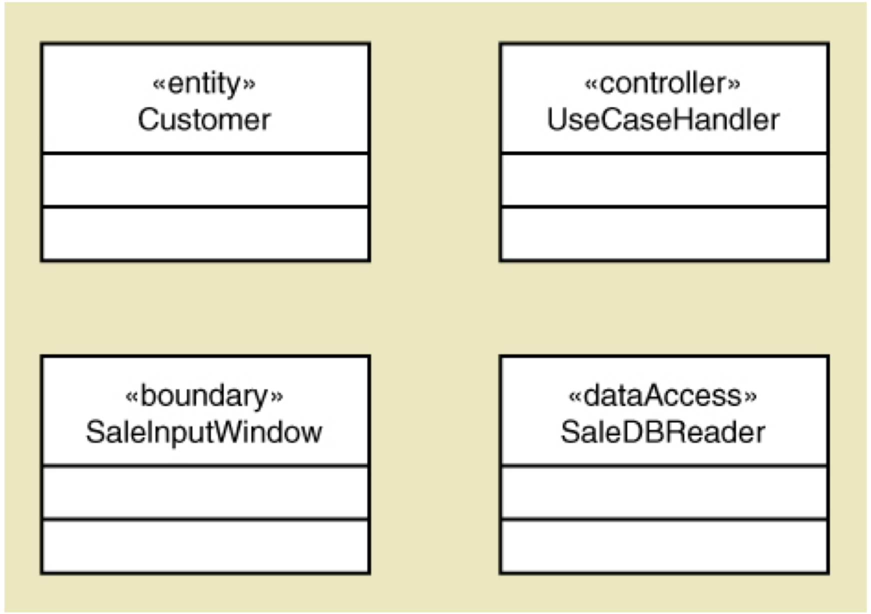
Design Classes and the Design Class Diagram

- **Stereotype**: a way of categorizing a model element by its characteristics
  - Name of the type placed within printer's guillemets (e.g., `<<strereotype>>`)
- **Entity class**: a design identifier for a problem domain class (usually persistent)
- **Persistent class**: an class whose objects exist after a system is shut down (data remembered)
  - Not indicated as stereotype.

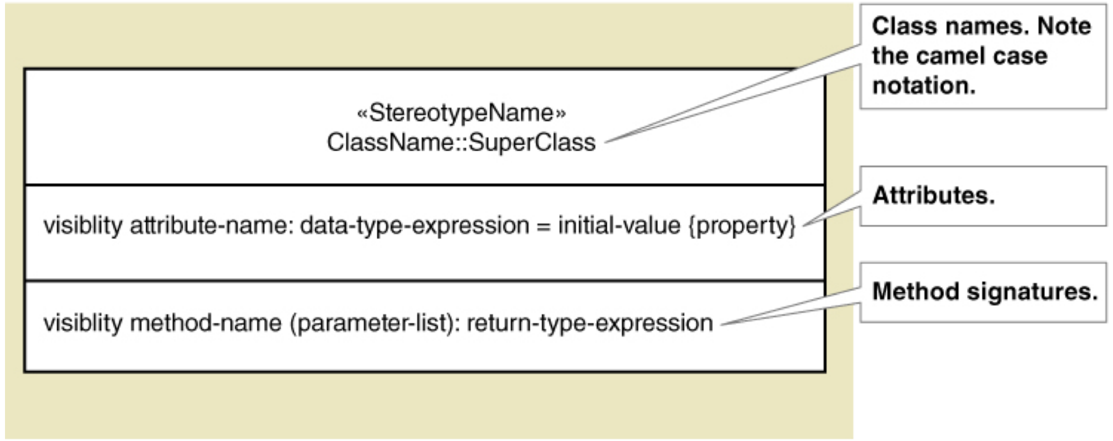


Design Classes and the Design Class Diagram

- **Boundary class or view class**: a class that exists on a system's automation boundary, such as an input window form or Web page
- **Controller class**: a class that mediates between boundary classes and entity classes, acting as a switchboard between the view layer and domain layer
- **Data access class**: a class that is used to retrieve data from and send data to a database

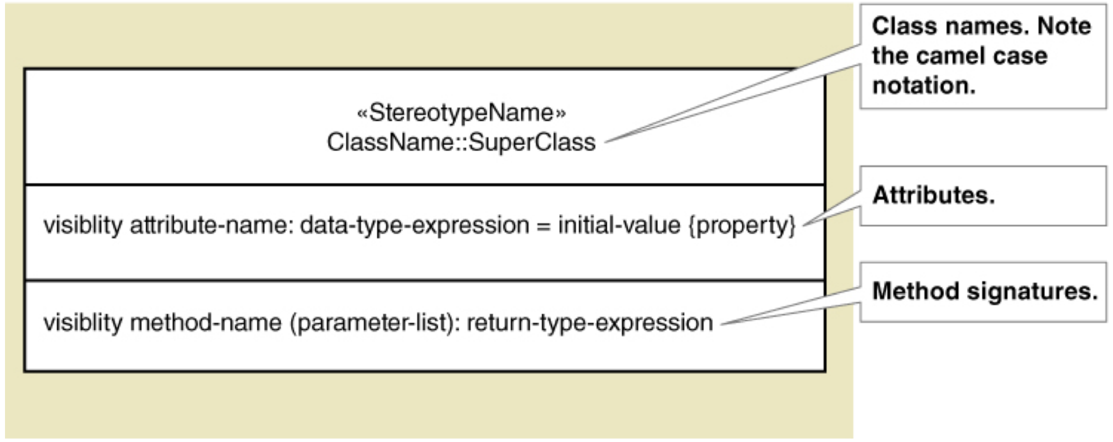


Notations for Design Classes


«StereotypeName»
ClassName::SuperClass
visiblity attribute-name: data-type-expression = initial-value {property}
visiblity method-name (parameter-list): return-type-expression
Class names. Note the camel case notation.
Attributes.
Method signatures.

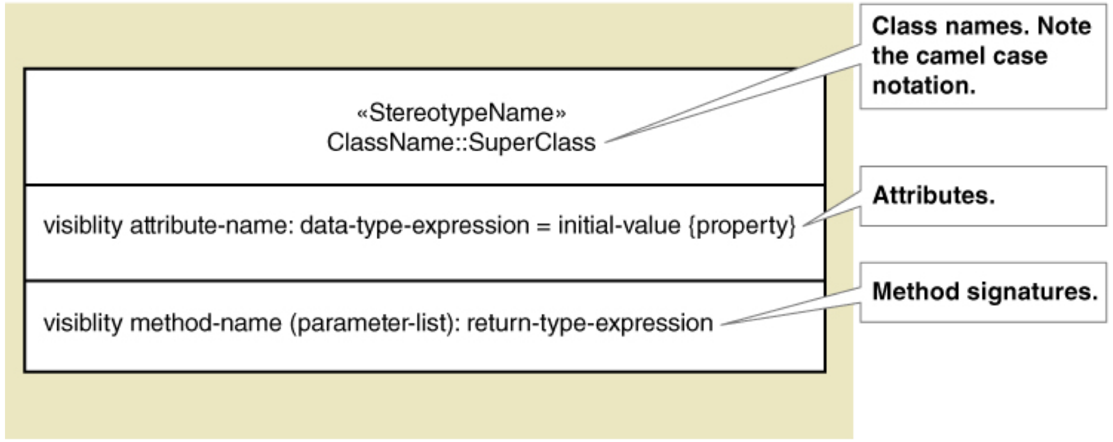
- **Visibility**—denotes whether other objects can directly access the attribute.
  - The values for visibility are a plus sign (+), which indicates that an attribute is visible, or public, and a minus sign (–), which indicates that it isn’t visible, or is private

Notations for Design Classes


«StereotypeName»
ClassName::SuperClass
visiblity attribute-name: data-type-expression = initial-value {property}
visiblity method-name (parameter-list): return-type-expression
Class names. Note the camel case notation.
Attributes.
Method signatures.

- **Elaborated attribute**
  - Attribute name
  - Data-type-expression (such as character, string, integer, number, currency or date)
  - Initial value, if applicable
  - Property (within curly braces), such as {key}, if applicable

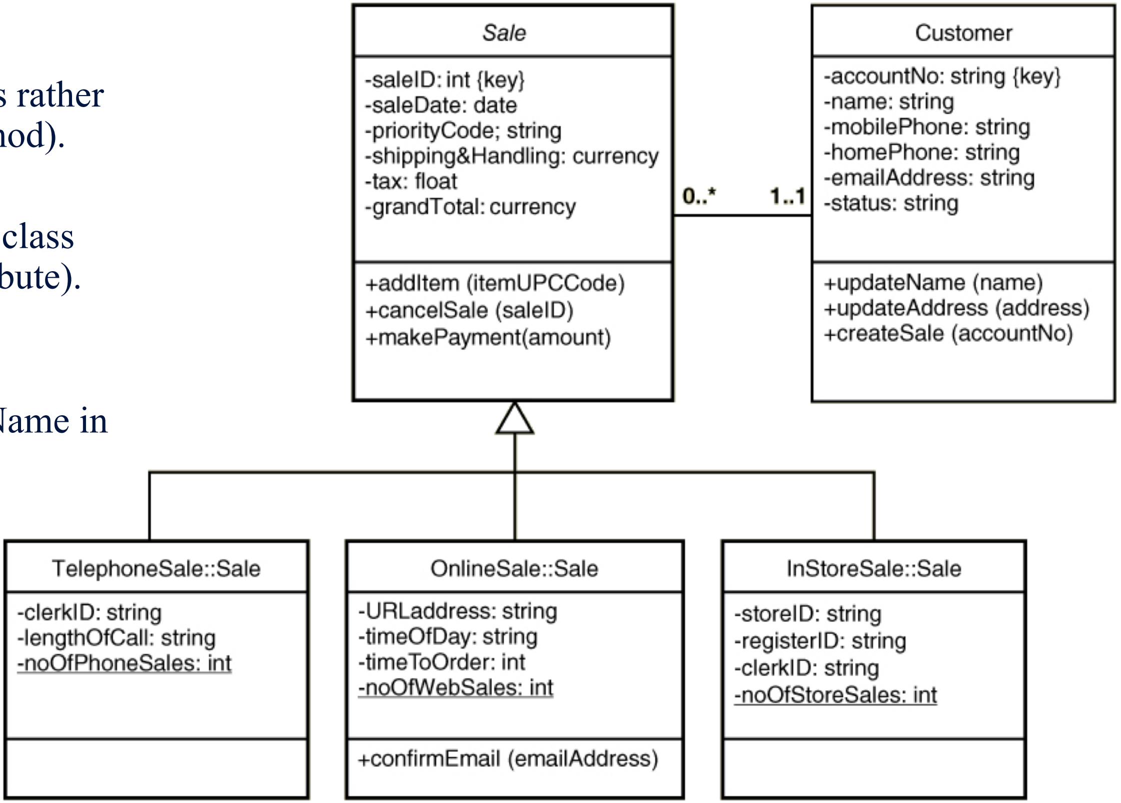
Notations for Design Classes


«StereotypeName»
ClassName::SuperClass
visiblity attribute-name: data-type-expression = initial-value {property}
visiblity method-name (parameter-list): return-type-expression
Class names. Note the camel case notation.
Attributes.
Method signatures.

- **Method signature**
  - the information needed to invoke (or call) the method, including:
  - Method visibility, Method-name, Method-parameter-list (incoming arguments), Return-type-expression (the type of the return parameter from the method)

Notations for Design Classes

- Class level method—applies to class rather than objects of class (aka static method). ***Underline*** it.
- Class level attribute—applies to the class rather than an object (aka static attribute). ***Underline*** it.
- Abstract class—class that can't be instantiated. Only for inheritance. Name in ***Italics***.
- Concrete class—class that can be instantiated.



Fundamentals of Object-Oriented Design

CSIT883 System Analysis and Project Management

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Outline

- Overview of Object-Oriented Design
- Object-Oriented Design Steps
- Design Class and Design Class Diagram
- **Designing with CRC Cards**

Part I

Part II

Part III

Developing the First-Cut Design Class Diagram

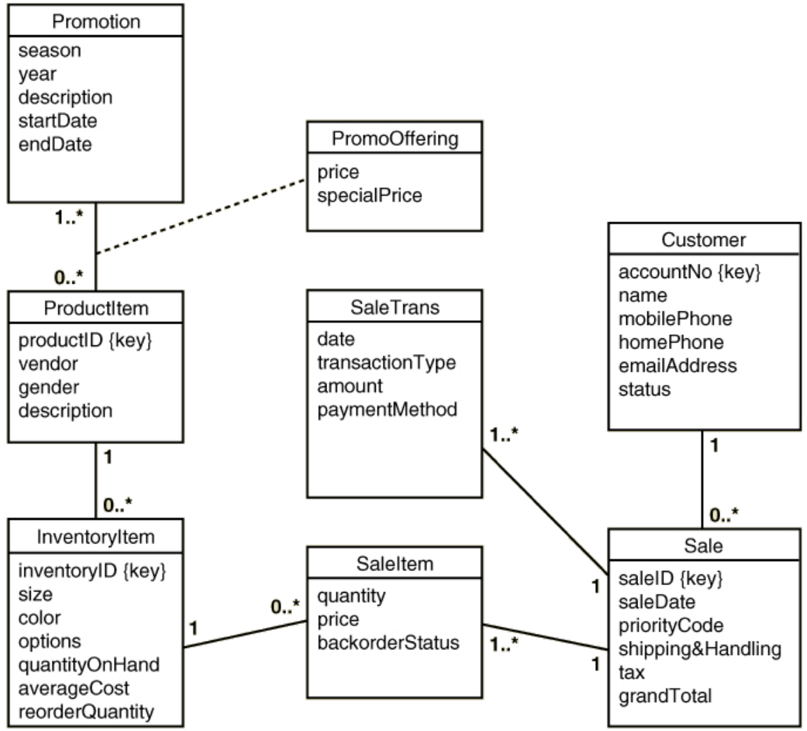
Choose a use case (e.g., *Create telephone sale*)

Source of information: Domain Model Class Diagram (DMCD)

Development of a first-cut DCD in three steps:

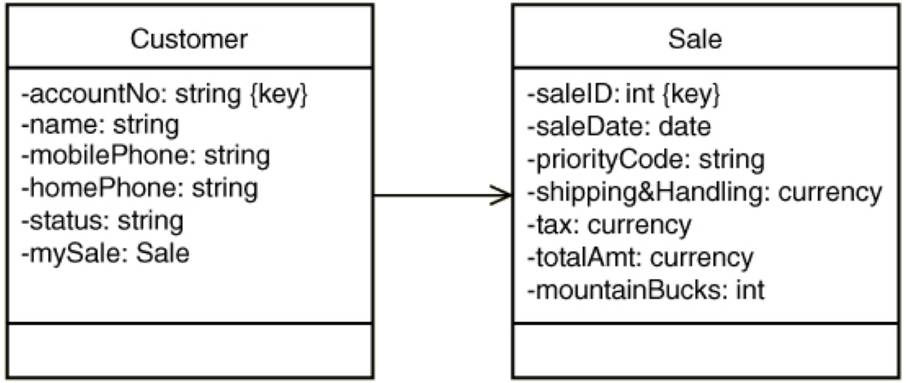
1. Add a *controller class* that handles the use case
2. Elaborate the attributes
3. Add navigation visibility arrows

DMCD for RMO Sales Subsystem



Developing the First-Cut Design Class Diagram

- **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



Developing the First-Cut Design Class Diagram

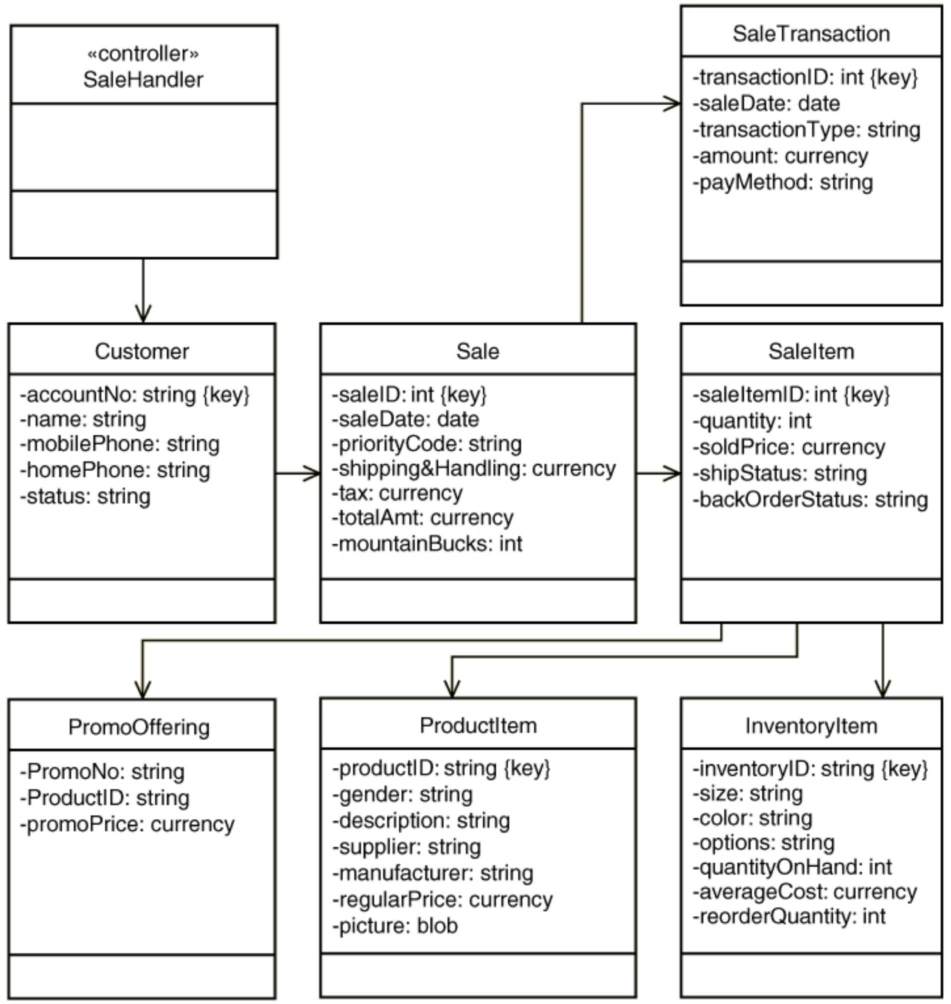
A basic question: *Which classes need to have references to or be able to access which other classes?*

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
- Navigation arrows may be bidirectional.

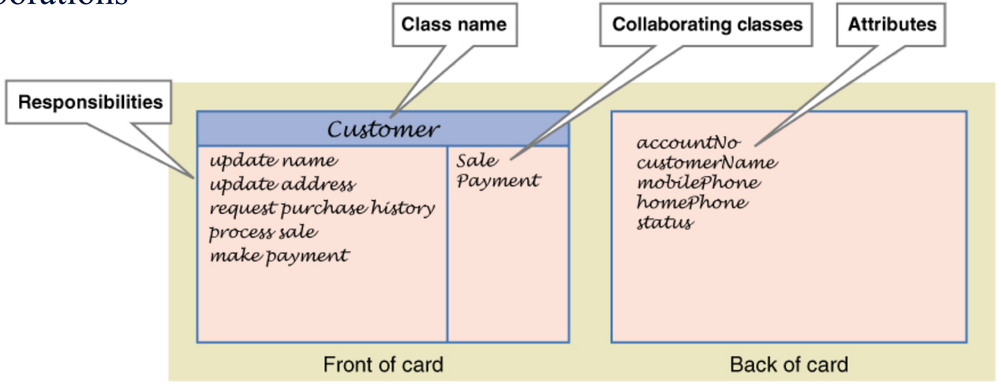
Developing the First-Cut Design Class Diagram

- Use case *Create telephone sale* with controller added



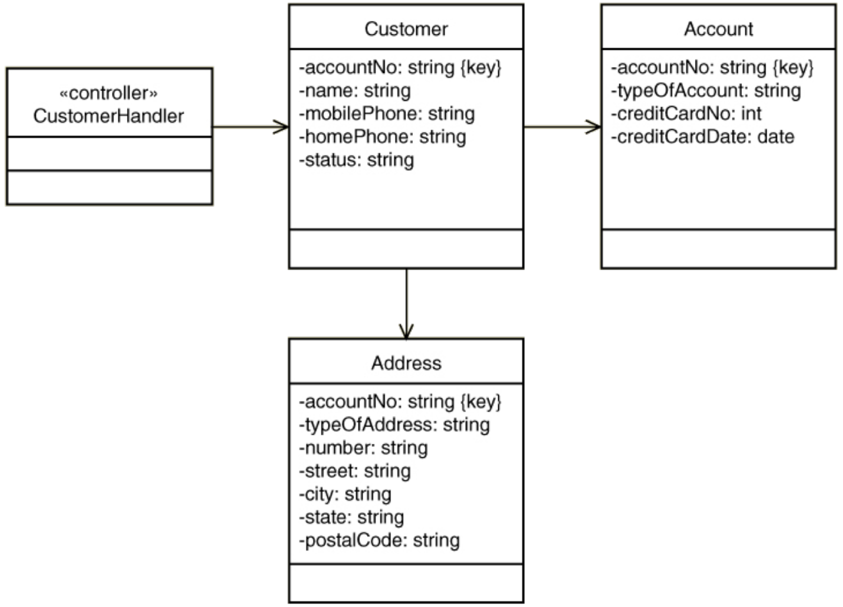
Designing With CRC Cards

- CRC Cards—Classes, Responsibilities, Collaboration Cards
- OO design is about assigning Responsibilities to Classes for how they Collaborate to accomplish a use case
- Usually a manual process done in a brainstorming session
  - One card per class
  - Front has responsibilities and collaborations
  - Back has attributes needed



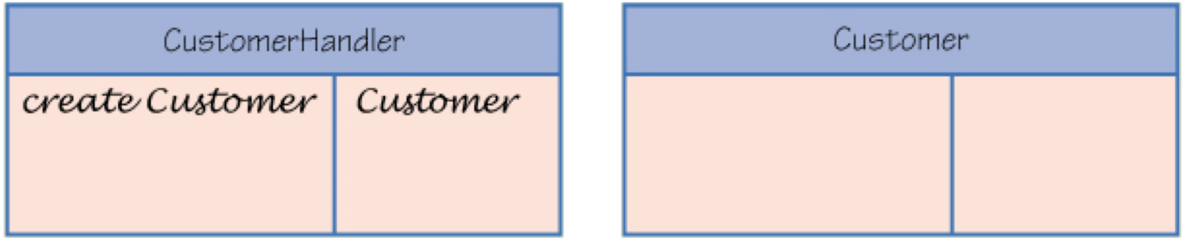
Building Design Class Diagram

- Example: *Create Customer Account*
- First-cut Design Class Diagram



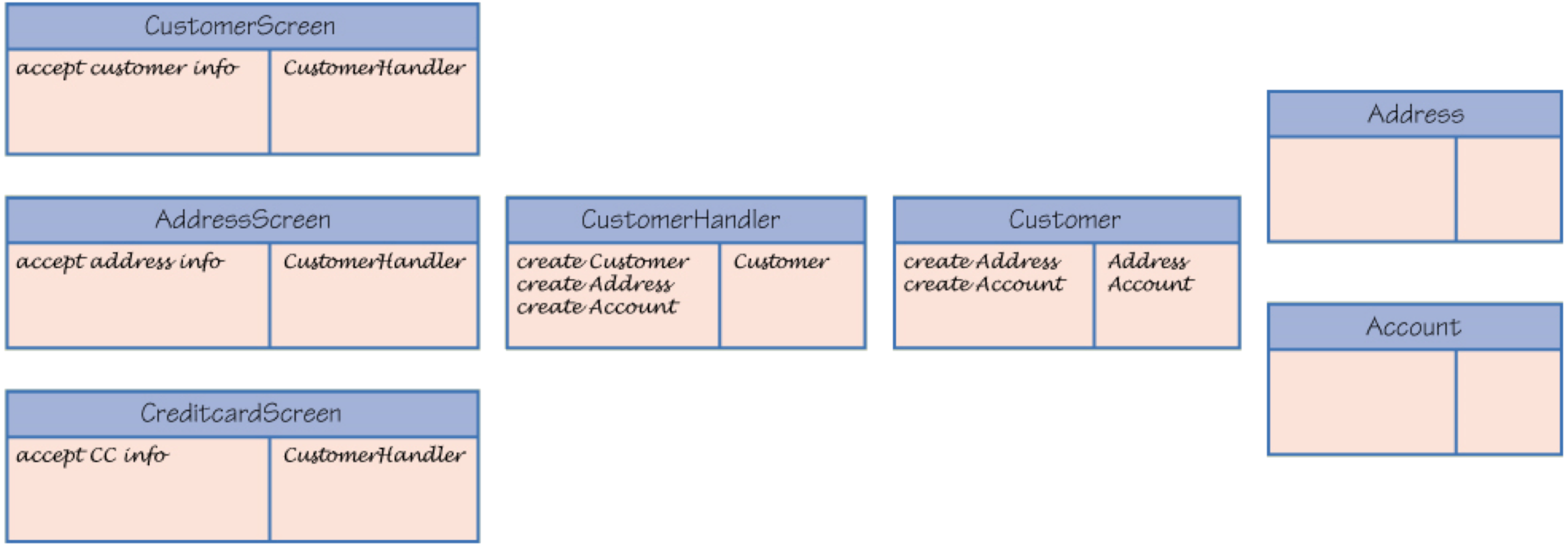
Building Design Class Diagram

- Example: *Create Customer Account*
- Add controller and determine *primary* problem domain class for this use case:



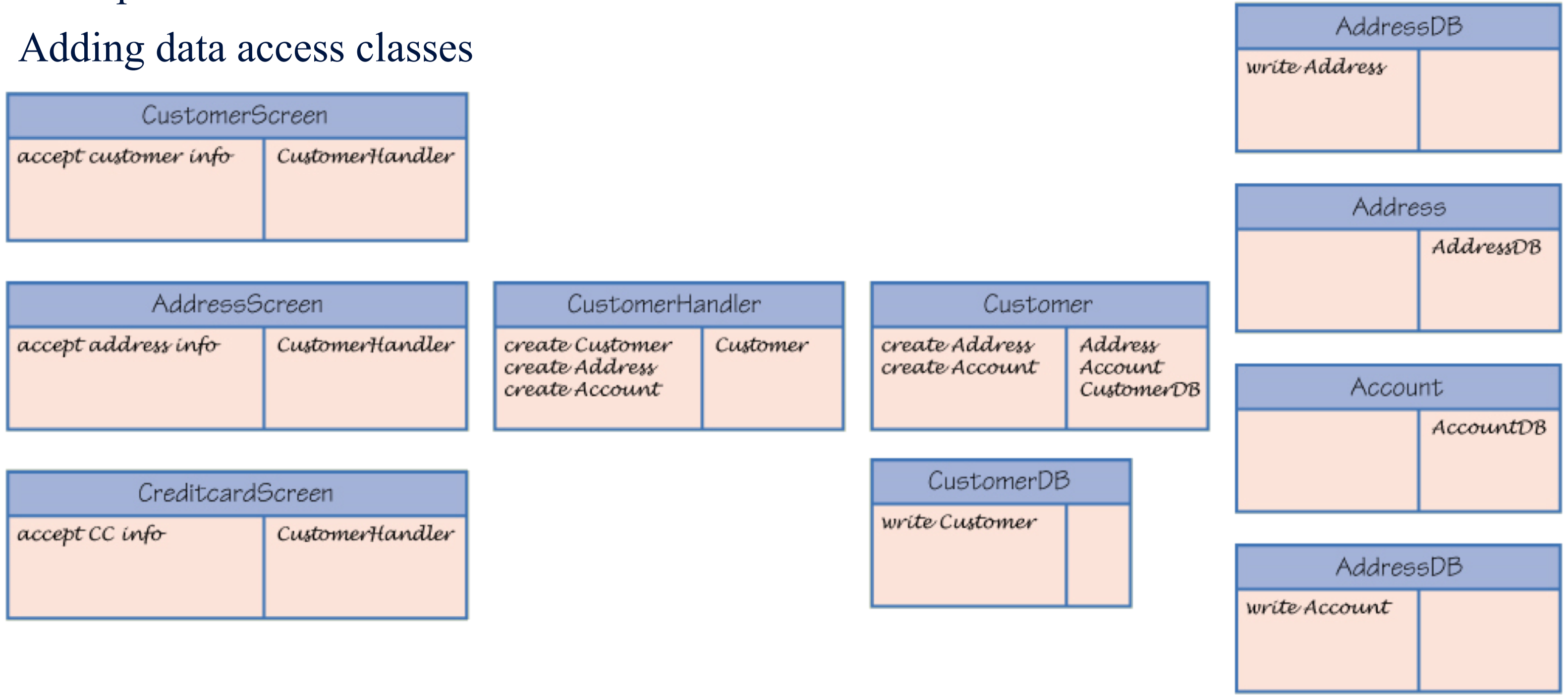
Building Design Class Diagram

- Example: *Create Customer Account*
- Problem domain classes and user interface classes



Building Design Class Diagram

- Example: *Create Customer Account*
- Adding data access classes



CustomerScreen	
accept customer info	CustomerHandler

AddressScreen	
accept address info	CustomerHandler

CreditcardScreen	
accept CC info	CustomerHandler

CustomerHandler	
create Customer create Address create Account	Customer

Customer	
create Address create Account	Address Account CustomerDB

CustomerDB	
write Customer	

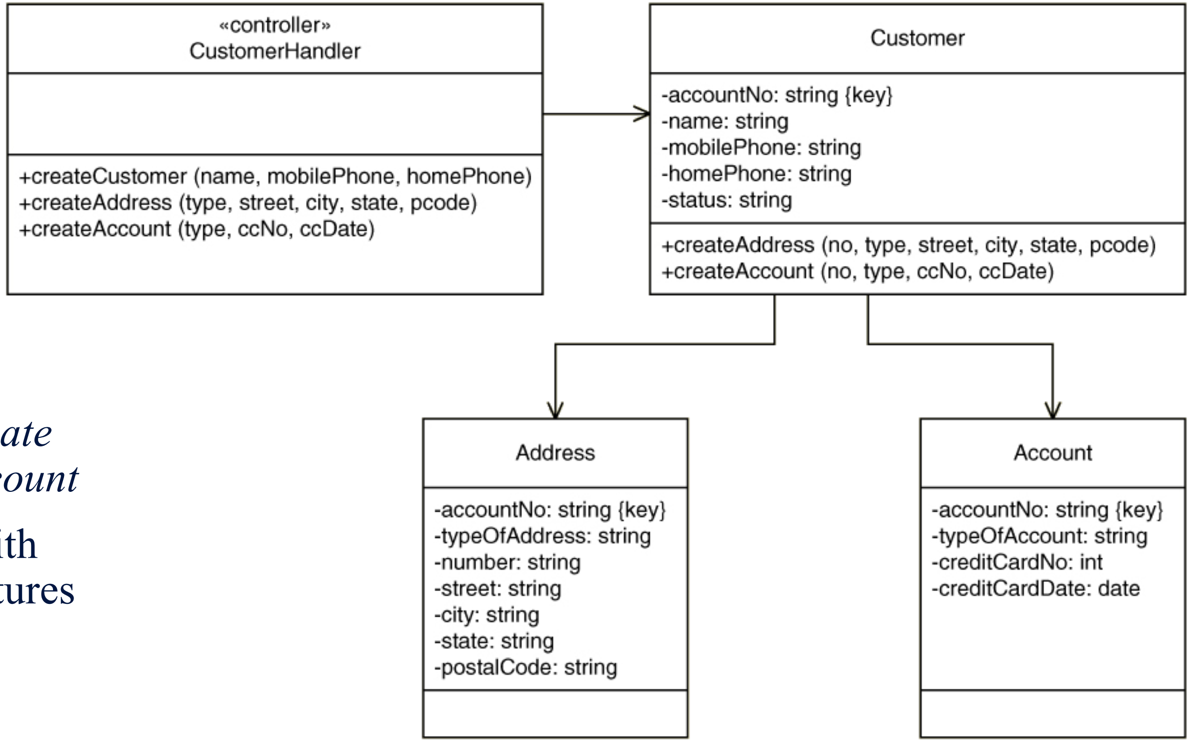
AddressDB	
write Address	

Address	
	AddressDB

Account	
	AccountDB

AddressDB	
write Account	

Building Design Class Diagram



- Example: *Create Customer Account*
- Final DCD with method signatures

Summary

- Systems design is the bridge that puts business requirements in terms that the programmers can use to write the software.
- The design class diagram (DCD) is usually developed in two steps:
  - A first-cut DCD is created based on the domain model class diagram, but then it is refined and expanded as the sequence diagrams are developed.
- One method of determining which objects collaborate is using CRC cards to define the interactions between design classes.