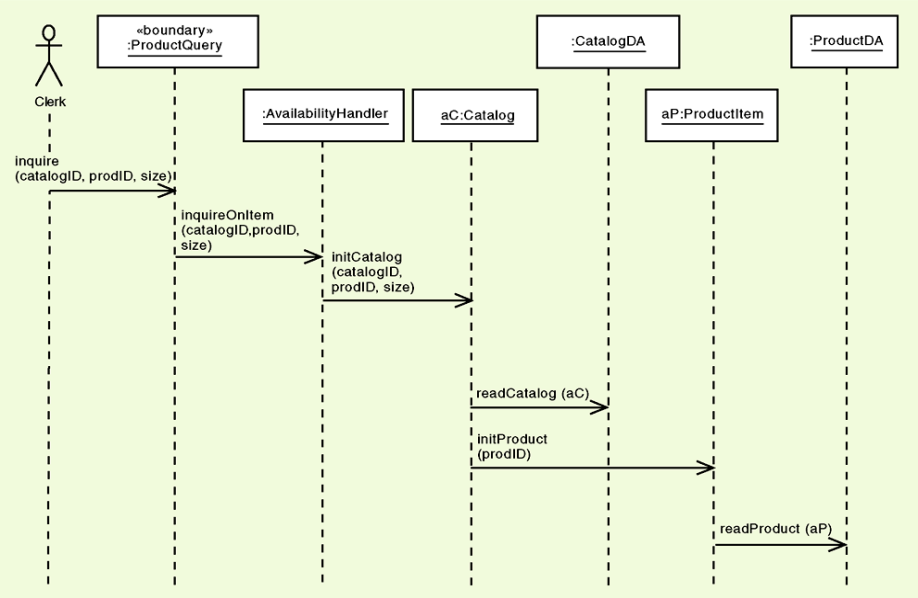
MONASH University

Information Technology

FIT2001 – Systems Development

Seminar 9: Use case realisation – Sequence diagrams

Chris Gonsalvez



Our road map:

- Use case realisation
- Design Class Diagrams, Sequence diagrams,

- What are Information Systems?
- How do we develop them? Systems Development (SDLC) – key phases
- Traditional vs. Agile approaches to developing systems
- Some System Development roles and skills
- Understand the requirements gathering process
- Managing stakeholders
- A range of Requirements gathering and documentation techniques
- Designing systems that our clients want – Usability
- Interface design principles & tips

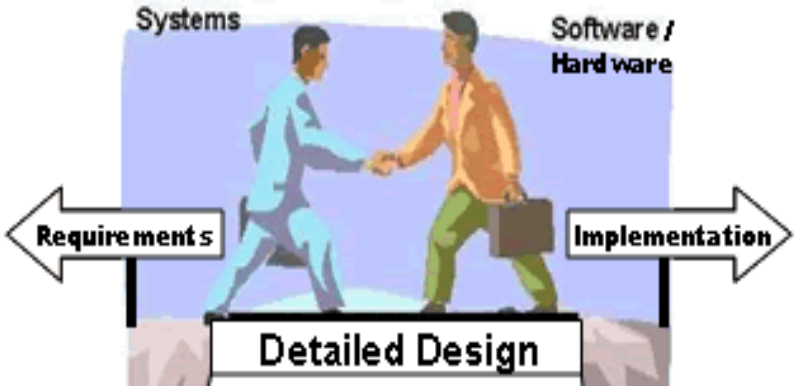
At the end of this topic you will:

- Be aware of object oriented design fundamentals
- Understand the process of use case realisation and be able to apply the techniques to develop detailed UML models
  - design class diagrams and sequence diagrams

Lecture Outline

1. Overview of Object Oriented (OO) Design
2. Overview of OO Programs
3. Fundamental Design principles
4. Iterative OO Design process
    - 4.1 Create first-cut Design Class Diagram
        - Definition, Notation, Process, Example
    - 4.2 Create Sequence Diagrams
        - 4.2.1 Background information: System Sequence Diagram
        - 4.2.2 First cut sequence diagram – How?, Example
        - 4.2.3 Final cut sequence diagram – How? Example
    - 4.3 Update Design Class Diagram
5. Summary

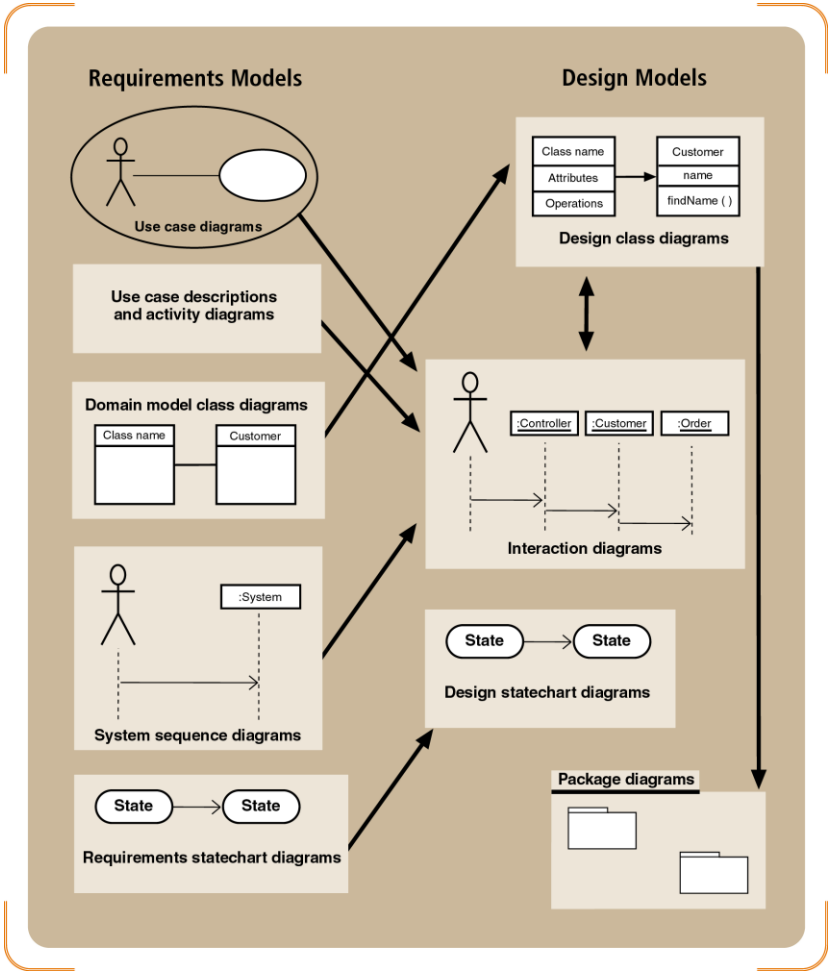
What is Object Oriented (OO) Design?



- The bridge between users requirements and programming for the new system
- A process by which detailed OO Design models (blueprints) are created – extends the requirements models
  - *These models are built only if necessary (Agile tenet)*
- The design models are used to build the new system, develop the database, user-interfaces, networks, controls and security

Relationship between UML analysis and design models

- Models such as activity diagrams, use cases diagrams, class models can inform various design models such as sequence diagrams and design class diagrams



Our focus:

Design models:

- Design class diagrams
- Interaction diagrams – in particular:
  - Sequence Diagrams

- Before we start:
  - Overview of OO programs
  - Overview of fundamental design principles

Overview of object-oriented programs

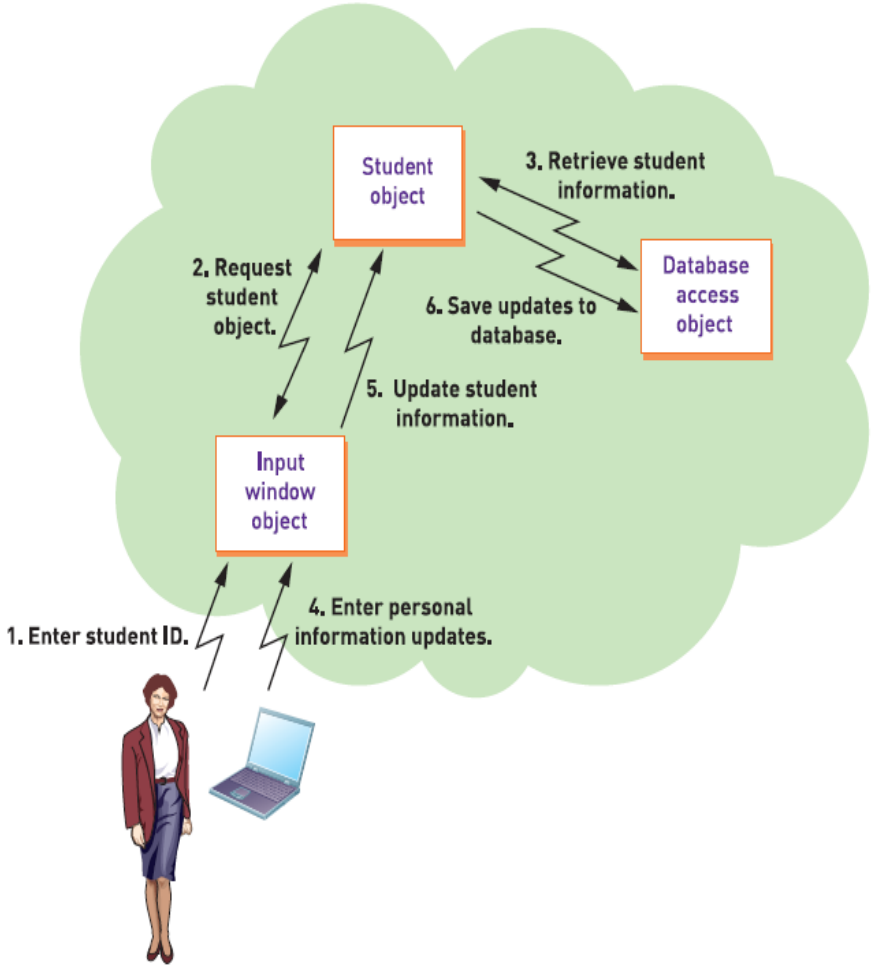
(so that we know what our design is trying to support)

- Set of program objects that cooperate by communicating to accomplish result
- Each object contains program logic and necessary attributes in a single encapsulated unit
- Objects send each other messages and work together to support functions of main program
- OO system design provides detail for programmers

  Eg. design class diagrams, interaction diagrams – communication diagrams, sequence diagrams

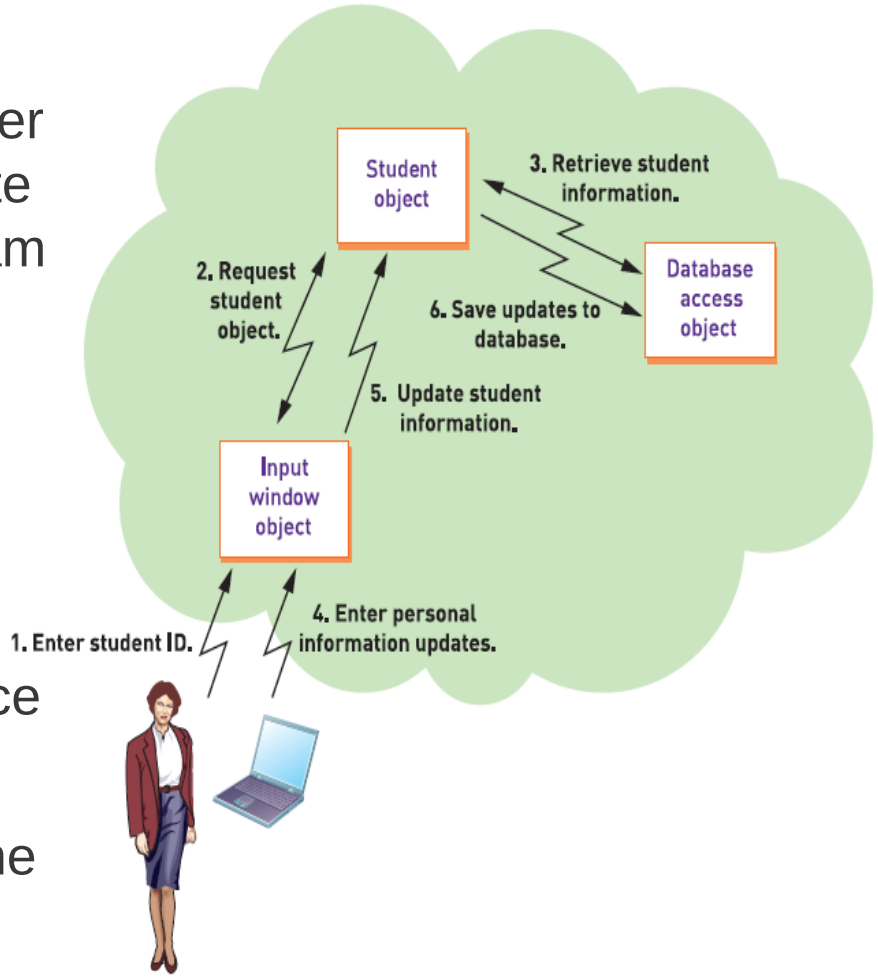
OOO event-driven program flow.1

- Windows object displays a form to enter Student ID and other info. (1)
- After student ID entered, windows object sends a message (2) to the Student Class telling it to create a new Student Object (an instance of Student Class), and to go to the database, retrieve the student information (3) and put it in the Student Object.
- Then the Student Object sends the information back to the Windows Object to display on the screen (2)



OOO event-driven program flow.2

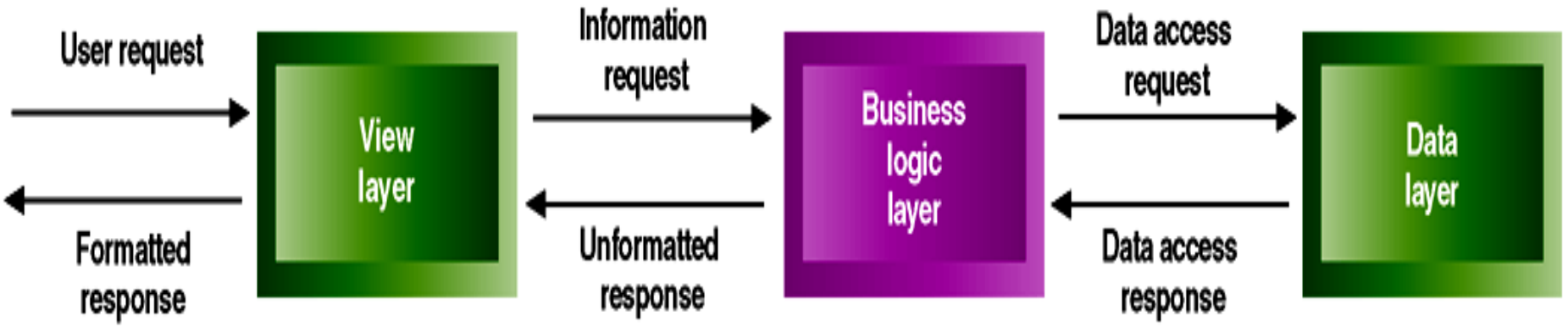
- The student then updates their personal information (4), and another series of messages is sent to update the Student Object (5) in the program and then the student information is used to update in the database (6).
- Analysts define the structure of the program logic and the data fields when they define the class.
- The object only comes into existence when the program runs
- This is called INSTANTIATION of the class – making an instance (object) based on the class template



OOO event-driven program flow.3
Three layer architecture

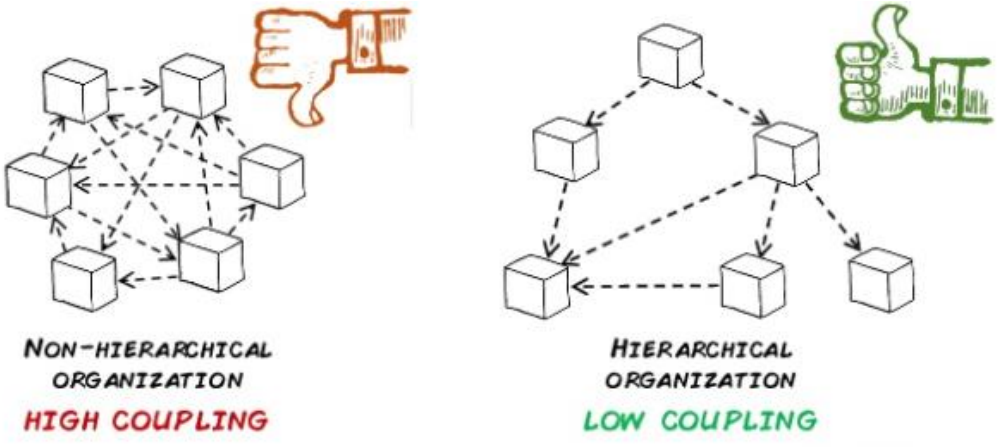
The three objects on the previous slide represent a 3 layer architecture structure

1. **Input window object (View layer)** accepts user inputs and formats/ displays processing results
2. **Student Object (Business logic or domain layer)** implements the rules and procedures of business processing
3. **Database access object (data layer)** manages stored data usually in one or more databases and communicates with the domain layer



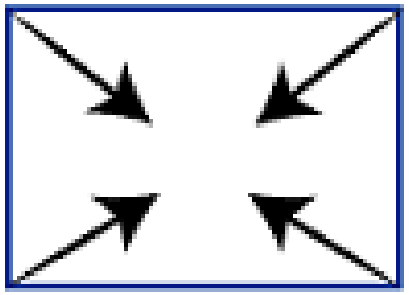
Fundamental Design Principles - Coupling

- Qualitative measure of how closely classes are linked
  - interconnected by the data in messages
  - number of navigation arrows on design class diagram
  - number of parameters passed by methods
- Low or loose coupling makes system easier to understand and maintain, minimal ripple effect

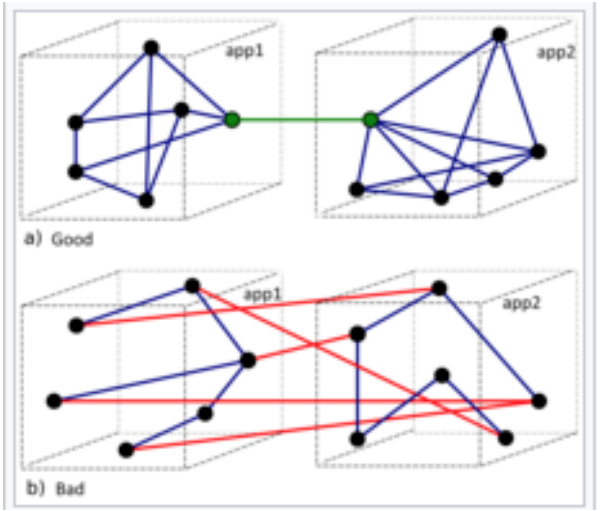


Fundamental Design Principles - Cohesion

- Measures the consistency of functions in a class … the unity of purpose of the methods within a class and the class itself
- Ensure clear separation of responsibility – divide low cohesive class into several highly cohesive classes



- Tight coupling often leads to low cohesion
  - increases the extent to which objects are dependent
  - can causes ripple-through effects
  - reduces the ability to reuse parts of the system
  - hard to maintain



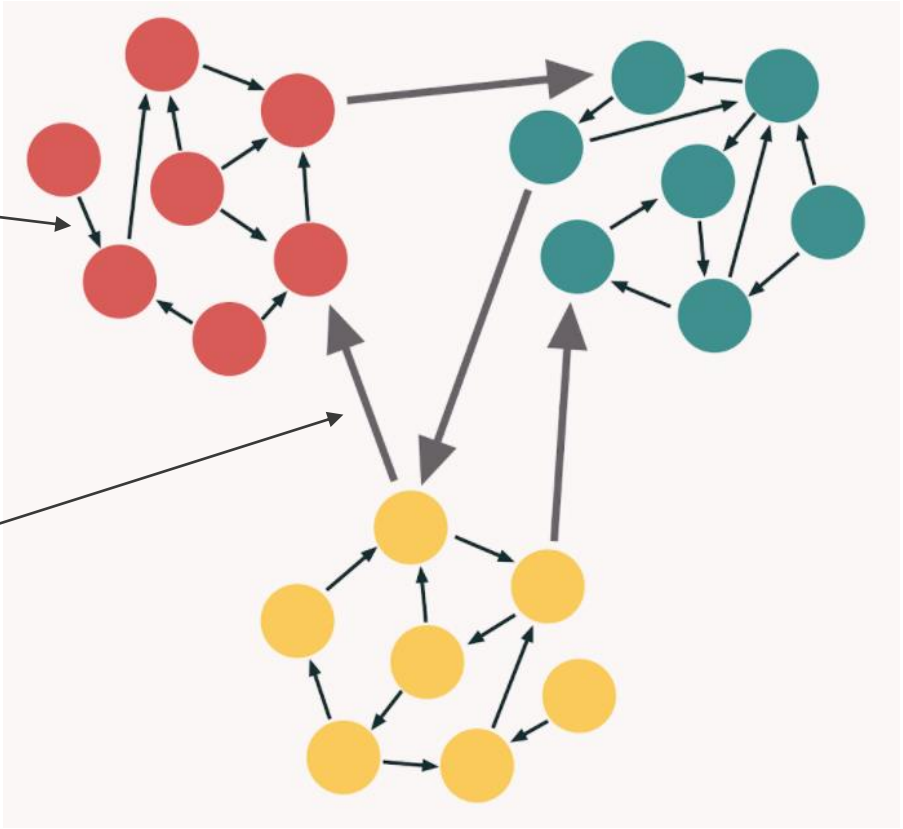
Difference between coupling and cohesion

Cohesion

within a module

Coupling

between modules



Iterative OO Design process

1. **Create first-cut (preliminary) design class diagram**
2. Create sequence diagrams, for each use case
   …realisation of use cases
3. Return to design class diagram and update based on design of sequence diagrams

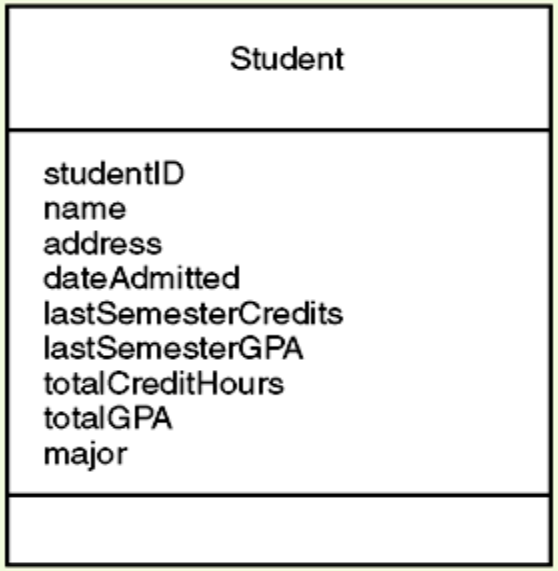
Design class diagrams and detailed interaction diagrams inform each other and should be developed in parallel

Design class diagrams (DCD) – What?

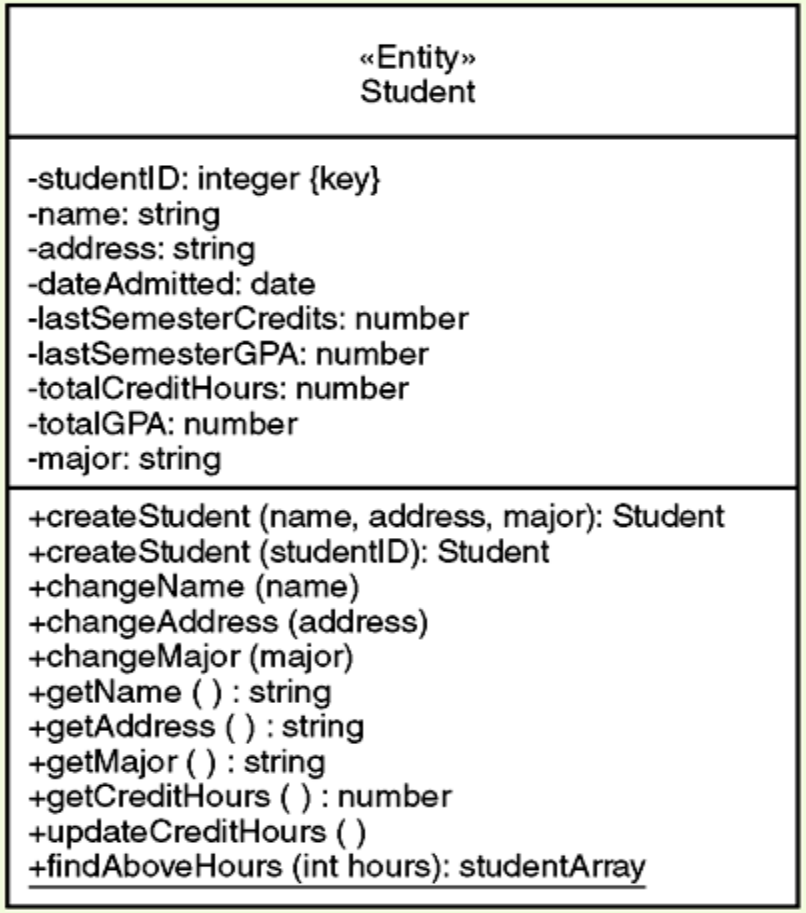
- Extend the domain model class diagram developed during OO analysis
- Design and document the programming classes that will be built for the new system .. describes the design component within the classes:
  - navigation between classes, attribute names and properties, method names and properties
- Shows set of problem domain classes and their association relationships
  - associations focus on navigation not multiplicity
  - direction of messages passed between objects
    - indicate the extent of coupling between objects

Moving from the Domain model class diagram to the Design Class Diagram

Domain diagram Student



Design class diagram Student



Design class stereotypes

- Domain model class diagrams show a conceptual view of users' business environment
- Design classes define software building blocks
  - When developers build design class diagrams, new classes are abstracted and identified as part of the design process
  - UML uses **stereotype** notation to categorise a type of design class – it has some special characteristic we want to highlight

Design class notation: Stereotypes

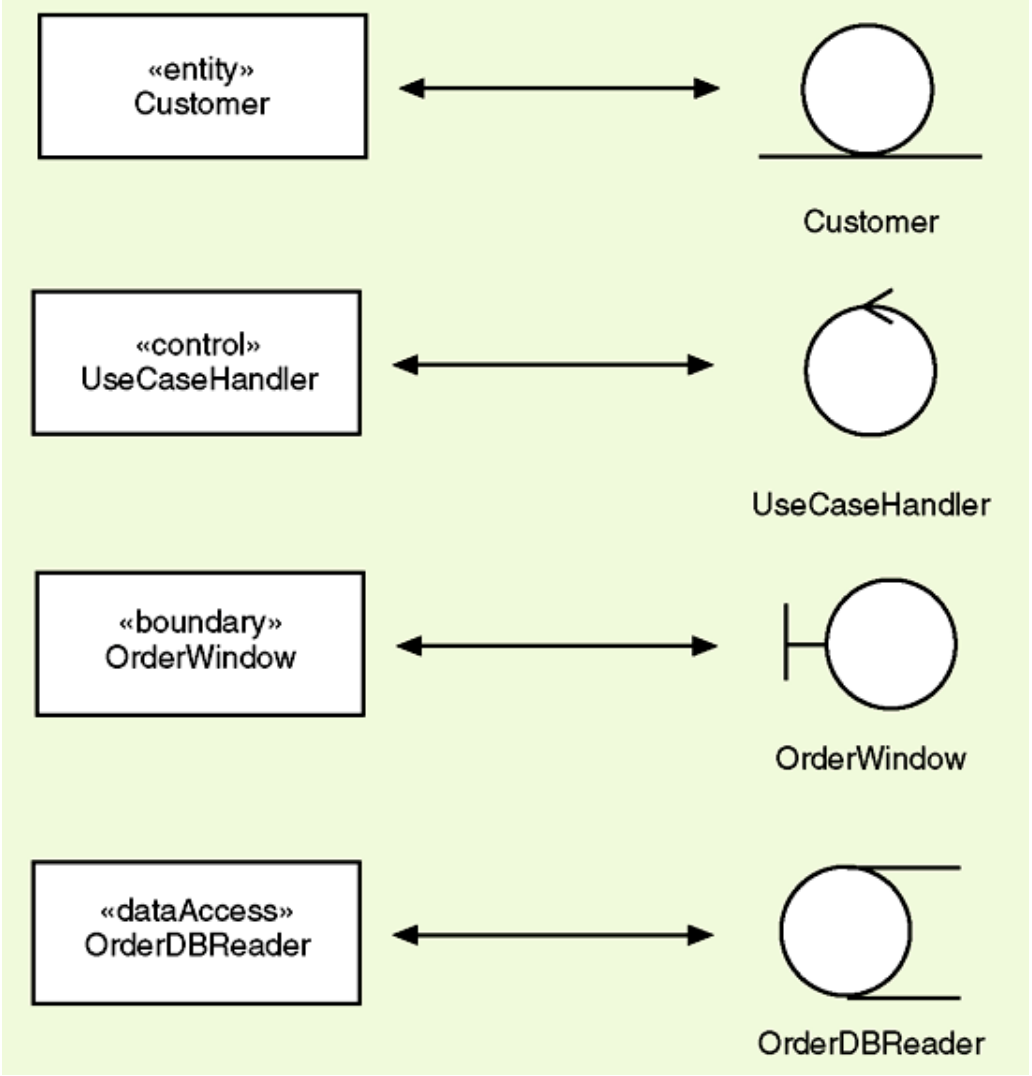
- **Entity** – design stereotype for problem domain class
  - something users deal with when doing their work
  - persistent class .. exists after system is shut down
- **Boundary** – designed to live on system's automation boundary
  - user interface class and windows classes
- **Control** – mediates between boundary and entity classes, between the view layer and domain layer
- **Data access** – retrieves from and sends data to the database

Stereotypes Notation:

Full notation with «»

OR

Shorthand notation with circular icons



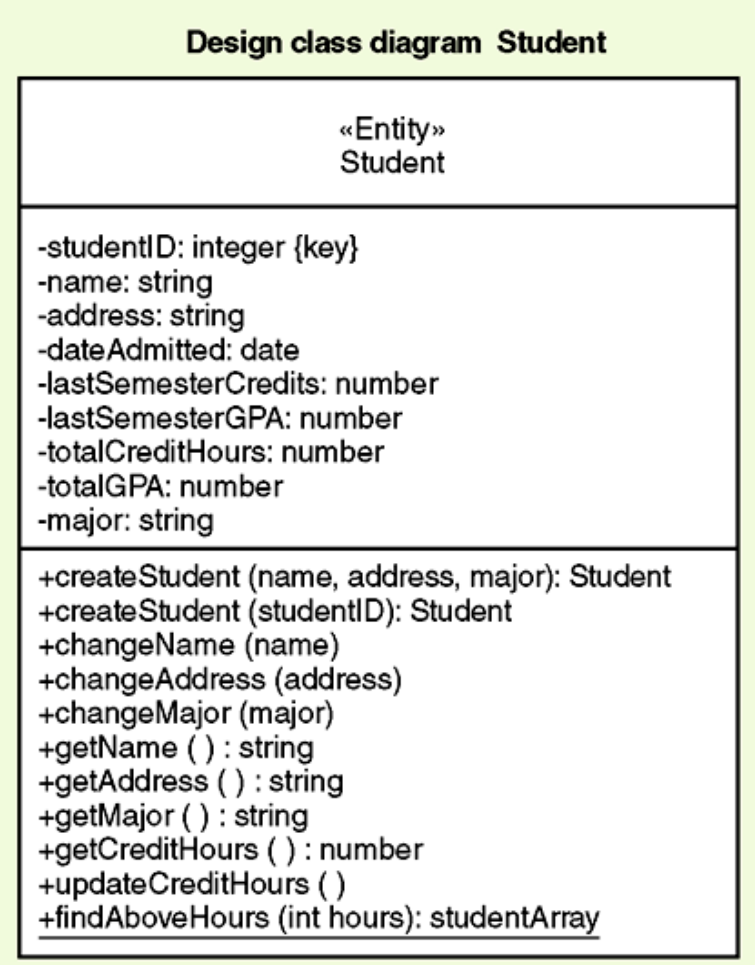
Canonical representation

Iconic representation

Design class notation

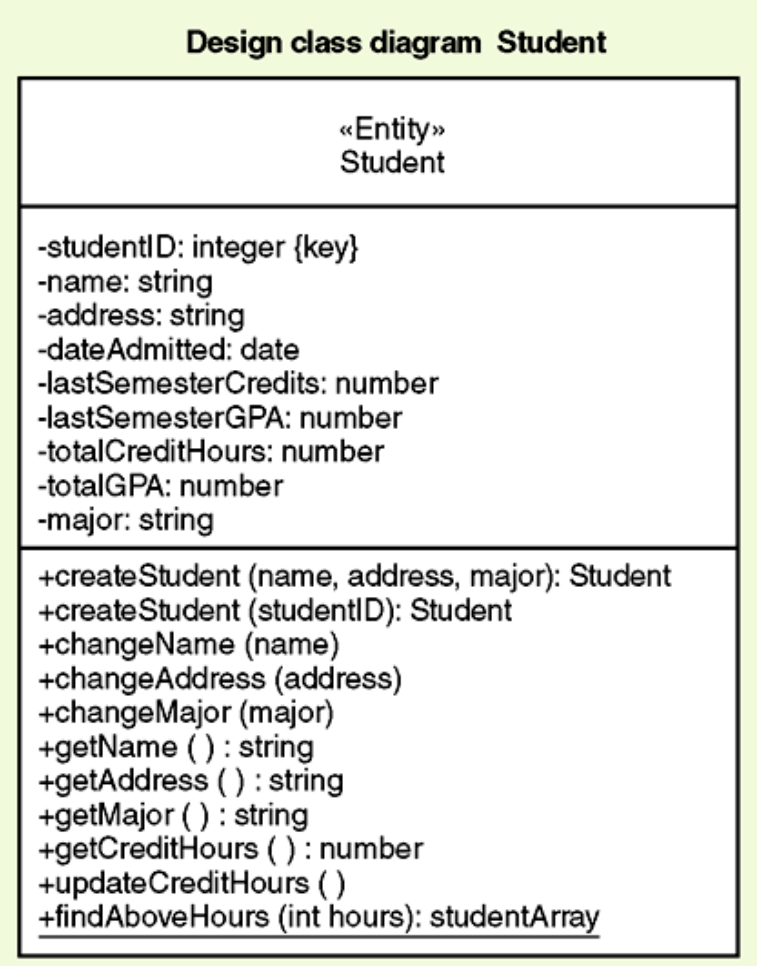
Design class notation: Class, Attributes

- Stereotype name & Class name
- Attributes
  - Visibility .. Can other objects access the attribute
    - private, public or protected
  - Attribute name
  - Data-type-expression
    - character, integer, string, number, currency or date
  - Initial value (if applicable)
  - Property within curly brackets eg. {key}



Design class notation: Methods

- Method signature – all information needed to call a method:
  - Method Visibility
  - Method name
  - Method parameter list (input arguments)
  - Return type-expression (return parameter)



Attribute and method visibility

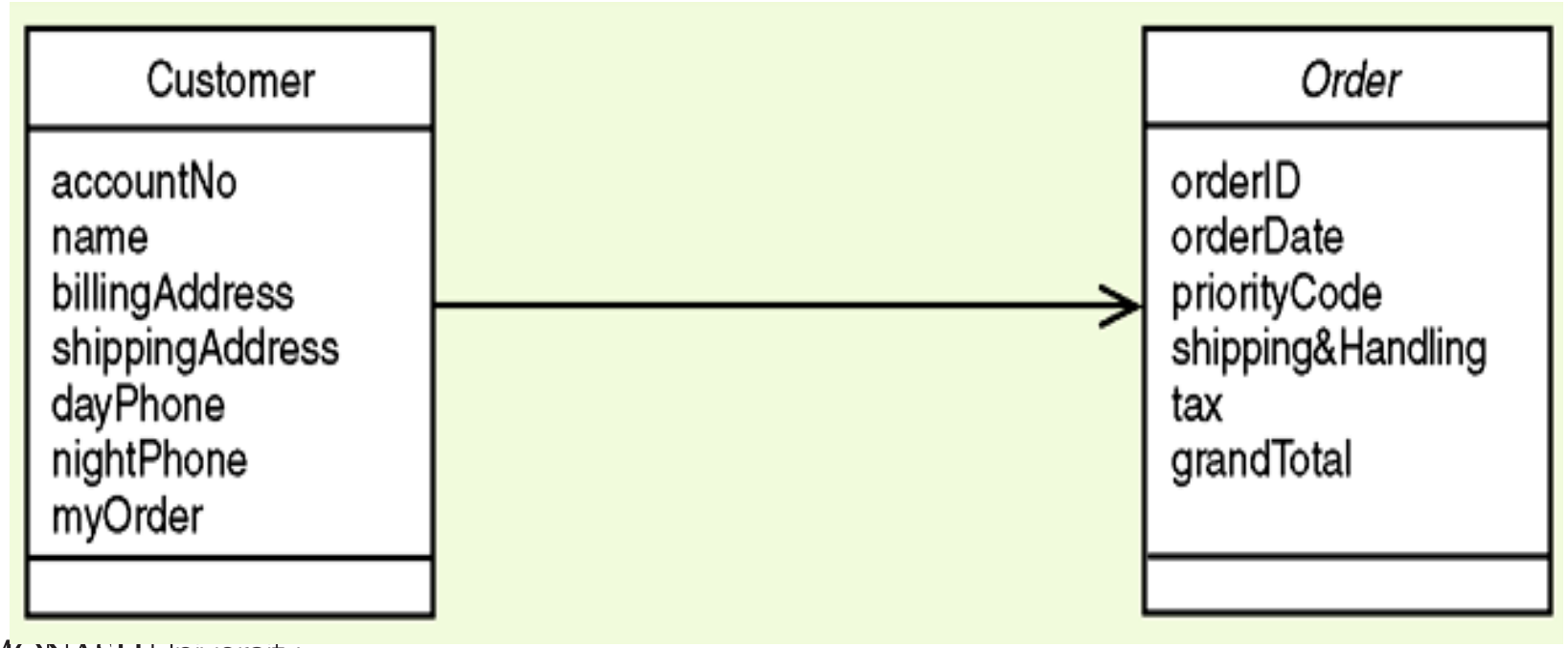
- Public (+)
  - available to other objects in system
  - get, set, display, constructor, destructor
- Private (-)
  - not accessible by other objects
  - methods only accessible to the object itself
  - often process-specific functions
- Protected (#)
  - can only be accessed by the object or its children

Design class diagrams – How?

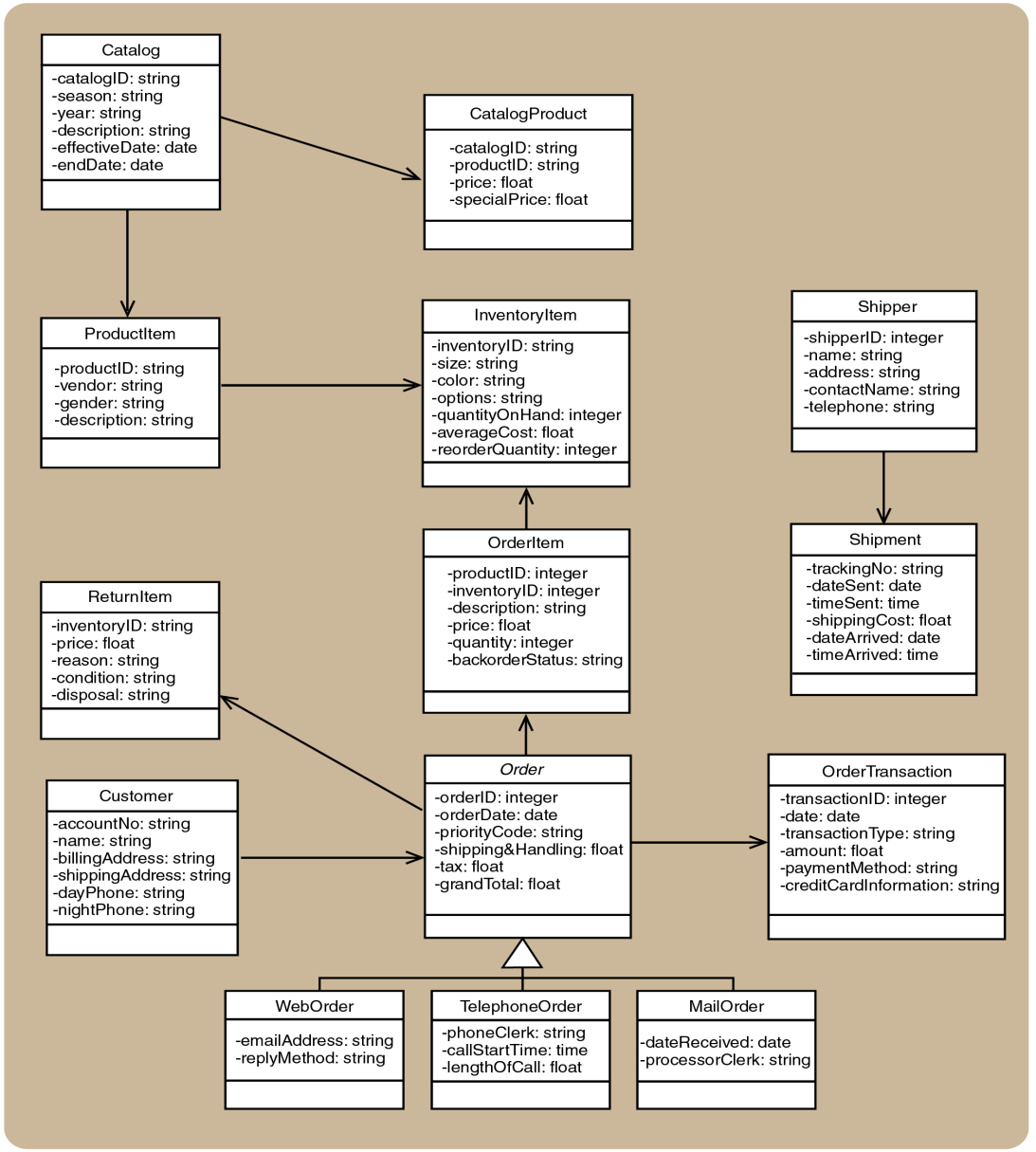
- Create a first-cut Design class diagram
  - Elaborate the attributes
    - add additional attributes if required
    - define their type
    - declare visibility (most should be private)
  - Show navigation visibility (add visibility arrows)
    - The ability of one object to view and interact with another object
    - Can be bidirectional
    - Will need to be updated as design progresses
- The first-cut Design class diagram is used to help develop the sequence diagram which will then be used to refine the Design class diagram

Navigation visibility between classes

- Navigation arrow indicates that the Order object must be visible to the Customer object
- Need to ask: Which classes need to have references to or be able to access other classes?



First-cut Design class diagram: Example



Iterative OO Design process

1. Create first-cut (preliminary) design class diagram
2. **Create sequence diagrams, for each use case …realisation of use cases**
3. Return to design class diagram and update based on design of sequence diagrams

Design class diagrams and detailed interaction diagrams inform each other and should be developed in parallel

Use case realisation with interaction diagrams

- Interaction diagrams are at the heart of object-oriented design. There are 2 types:
  - Sequence
  - Communication

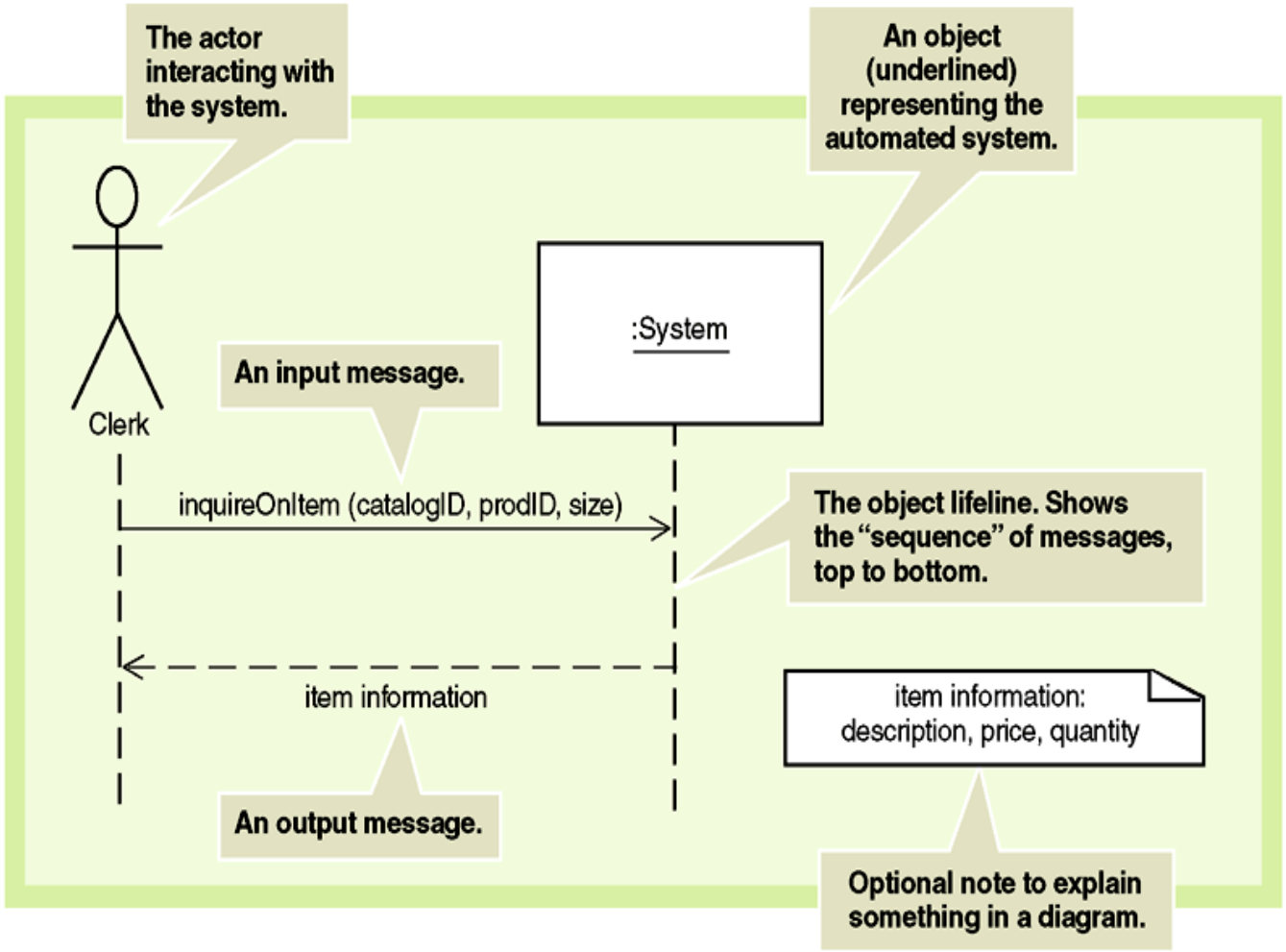
  (we will be doing an overview of Sequence Diagrams only)
- Realisation of a use case - Determine what objects communicate by sending messages to each other

Some background information: The System Sequence Diagram

- System sequence diagram
  - Shows sequence of messages between the actor and the system for a single use case
  - Only shows actor and one object – part of the system
  - Shows input & output messaging requirements for the use case

  - Actor, :System, object lifeline
  - Messages

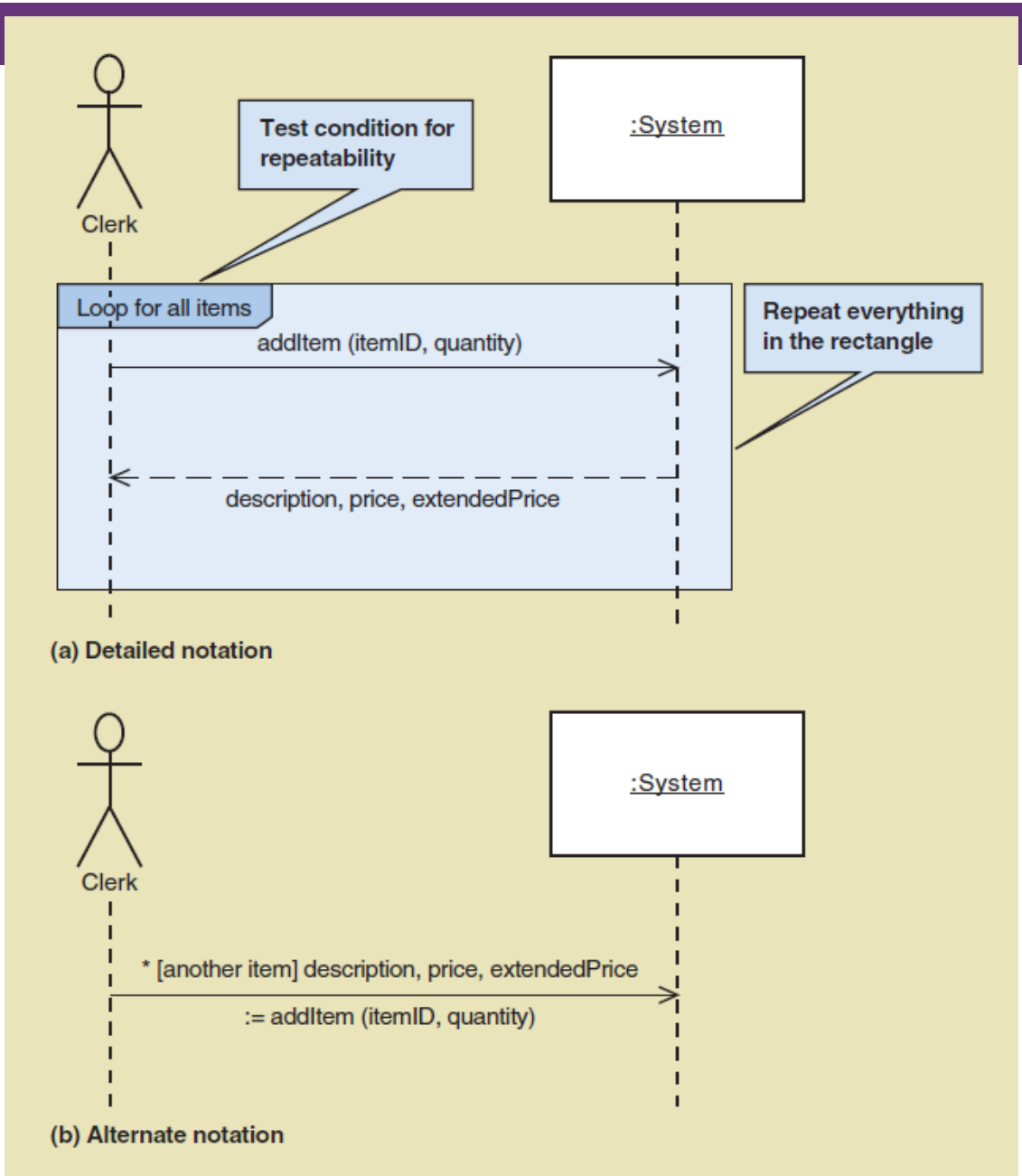
System Sequence Diagram (SSD) - Notation



SSD Notation

- Actor represented by stick figure – person (or role) that “interacts” with system by entering input data and receiving output data
- Object notation is rectangle with name of object underlined – shows individual object and not class of all similar objects
- Lifeline is vertical line under object or actor to show passage of time for object
- Messages use arrows to show messages sent or received by actor or system

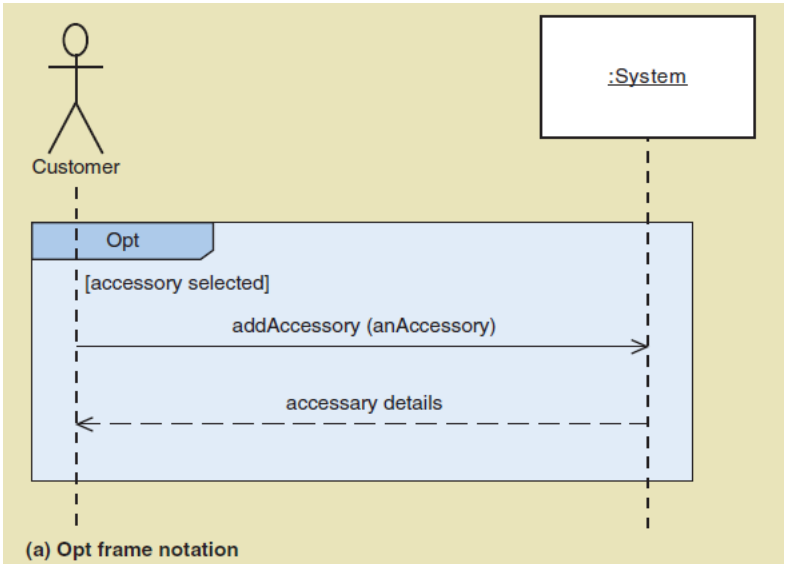
SSD Message Examples with Loop Frame



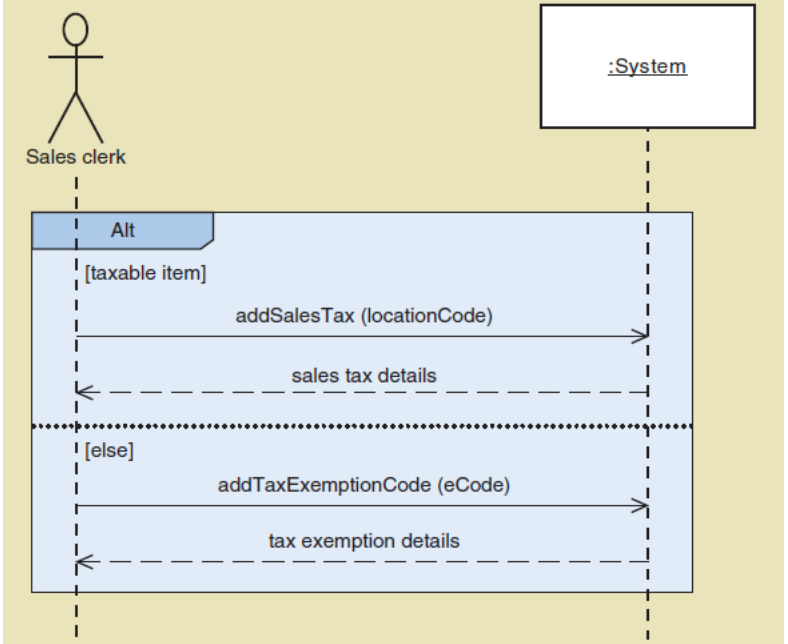
SSD Message Examples

Opt Frame (optional)

Alt Frame (if-else)



(a) Opt frame notation

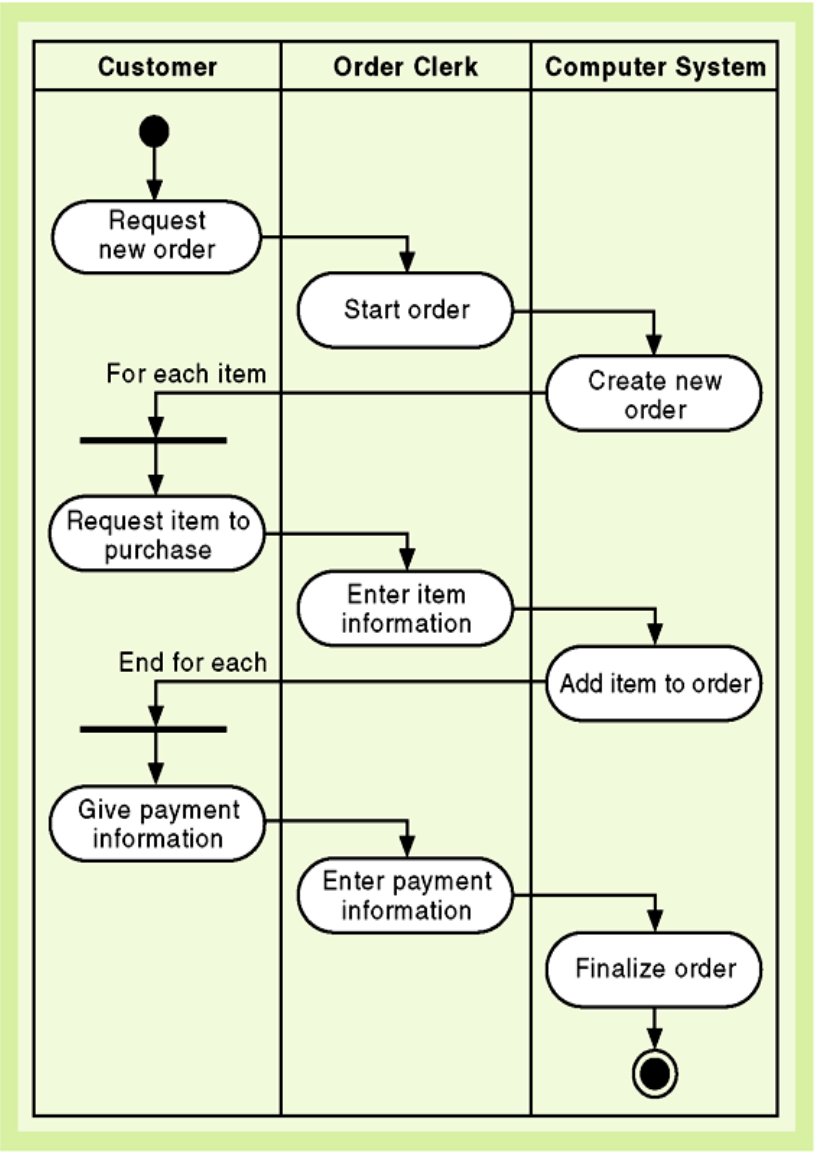


(b) Alt frame notation

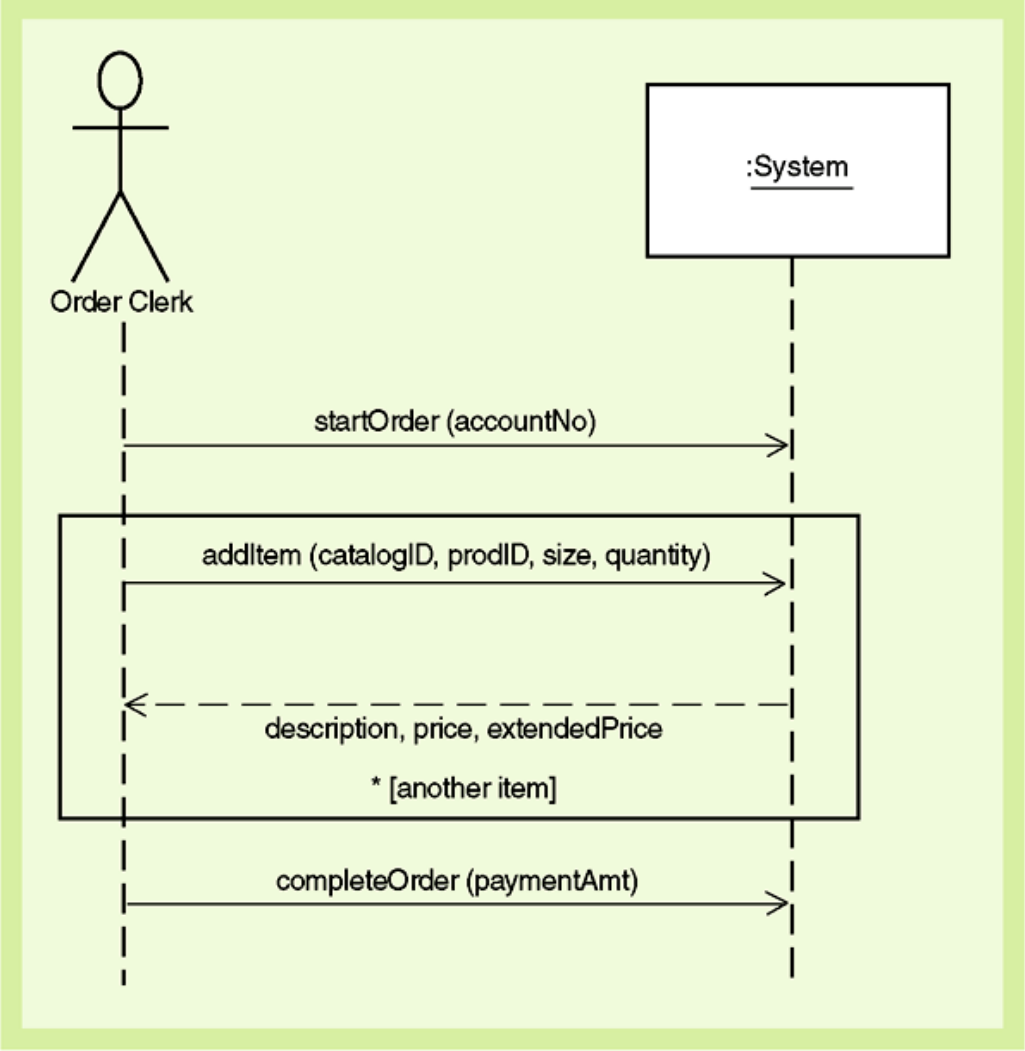
Developing a System Sequence Diagram

- Identify input messages – from use case descriptions or activity diagrams
- Describe message from external actor to system using message notation
  - Name message verb-noun: what the system is asked to do
  - Consider parameters the system will need
- Identify and add any special conditions on input message
  - Iteration/loop frame
  - Opt or Alt frame
- Identify and add output return messages
  - On message itself: aValue:= getValue(valueID)
  - As explicit return on separate dashed line

Simplified Activity Diagram of the Phone Order Scenario



SSD of Simplified Telephone Order Scenario for Create New Order Use Case



Example: use cases to SSD

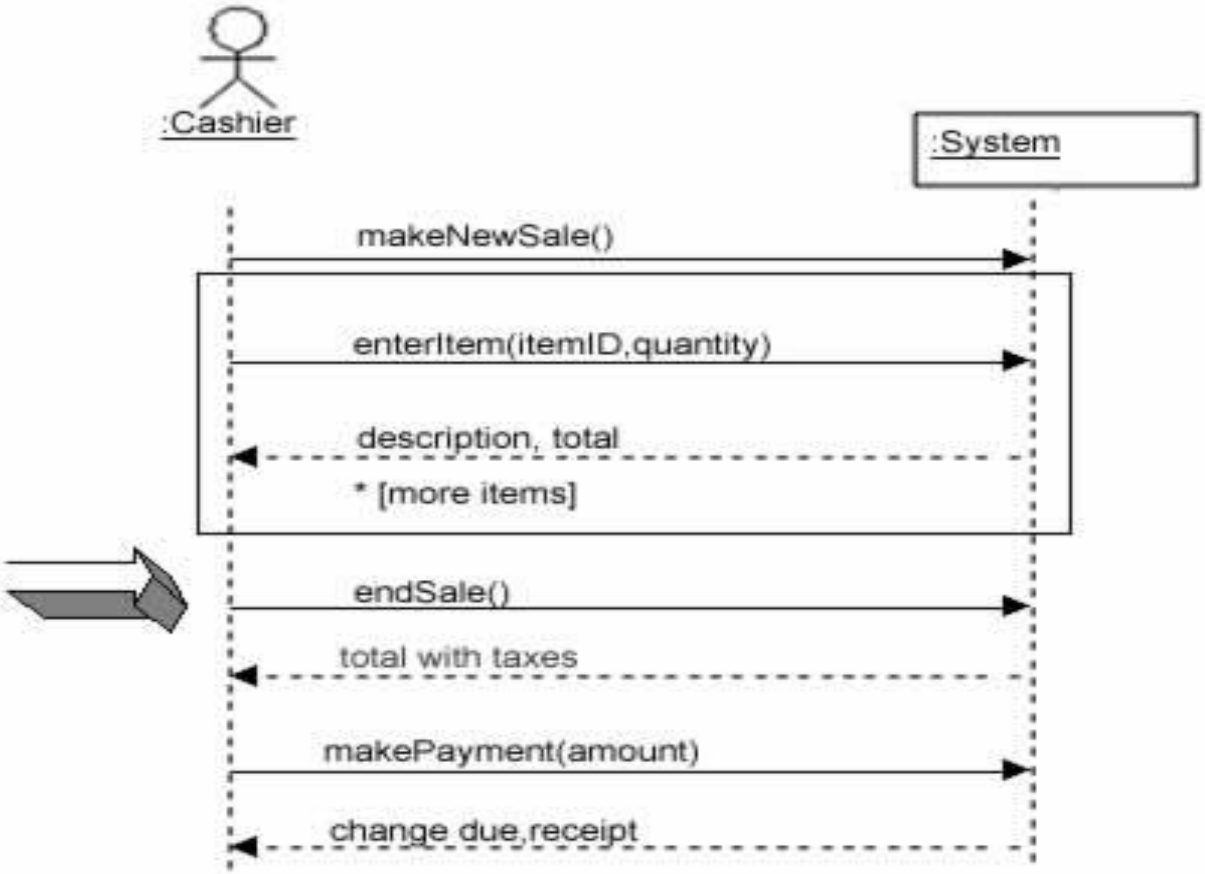
SSD can be created based on a particular use case scenario (description)

Simple Cash-only Process Sale scenario:

1.customer arrives at aPOS check out with goods and/or services to purchase.
2.Cashier starts a new sale.
3.Cashier enters a new item identifier.
4.System records new sale line item and presents item description,price and running total.

Cashier repeats steps steps 3-4 until indicates done.

5.System presents total with taxes calculated.
6.Cashier tells Customer the total, and asks for payment.
7.Customer pays and System handles payment.
....

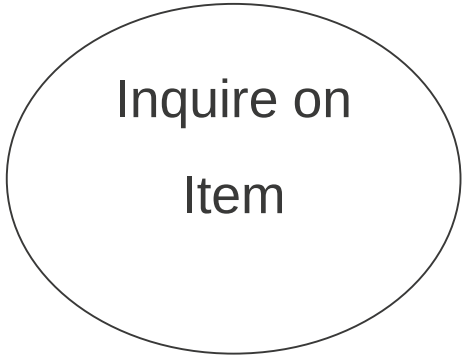


Sequence diagrams: Definition

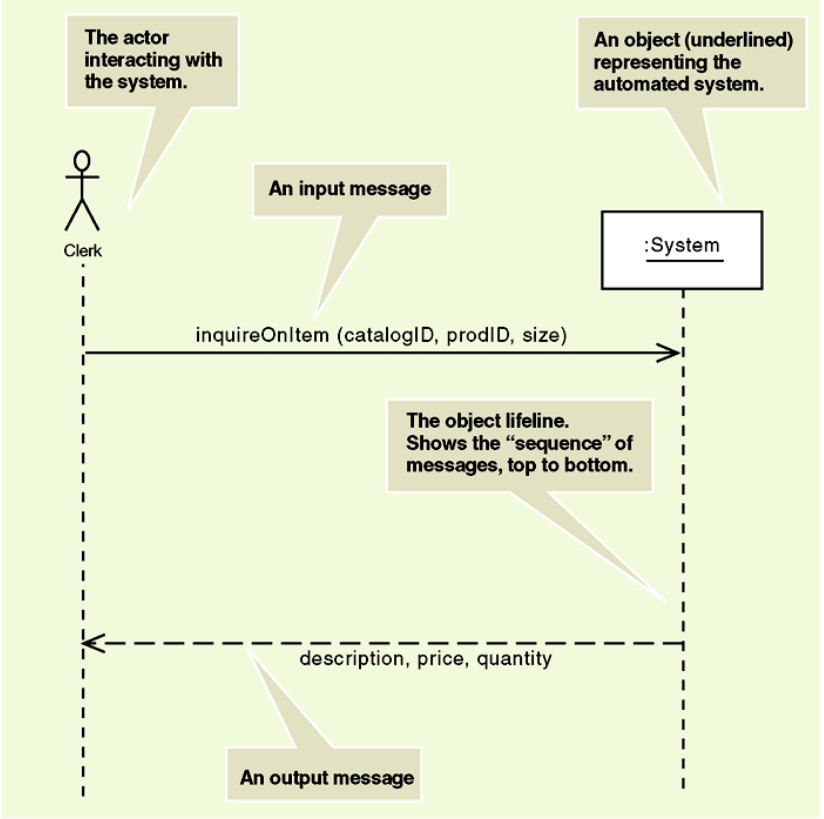
- Sequence diagrams extend the system sequence diagram (SSD) to show:
  - View layer objects
  - Domain layer objects (usually done first)
  - Data access layer objects
- Sequence diagrams used to explain object interactions and document design decisions
- For each use case and scenario
  - documents inputs to and outputs from system
  - captures interactions between system and external world as represented by actors
  - captures and defines specific interactions between collaborating objects within each use case

First-cut sequence diagram: How.1

1. We are going to work with the Use Case for Inquire on Item

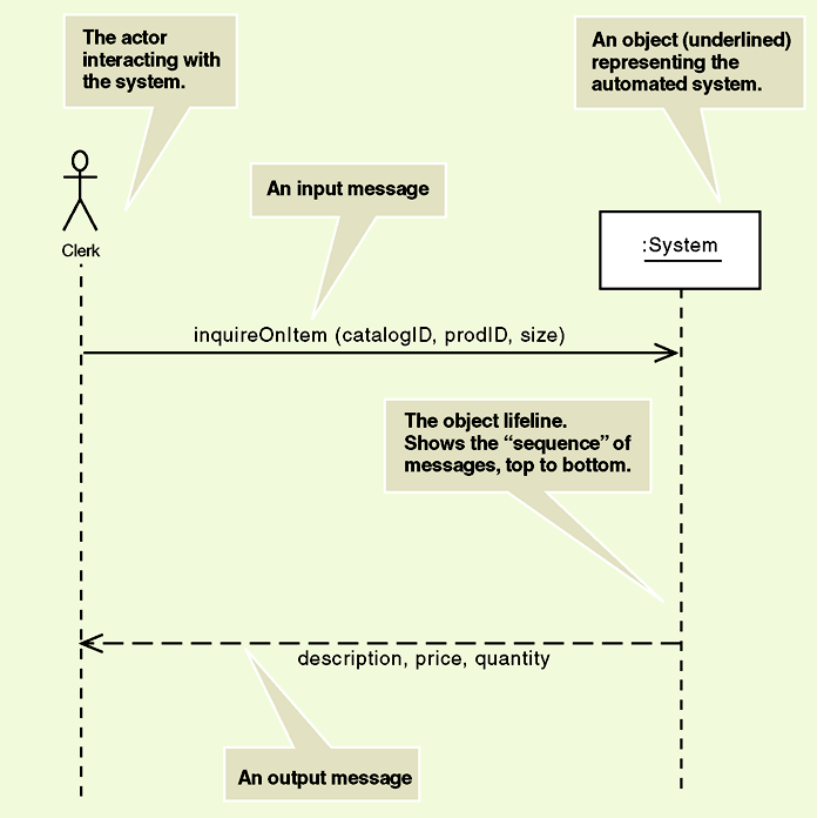


2. Description: For a particular size of product in a catalogue, we want to know the description, price and quantity available



First-cut sequence diagram: How.2

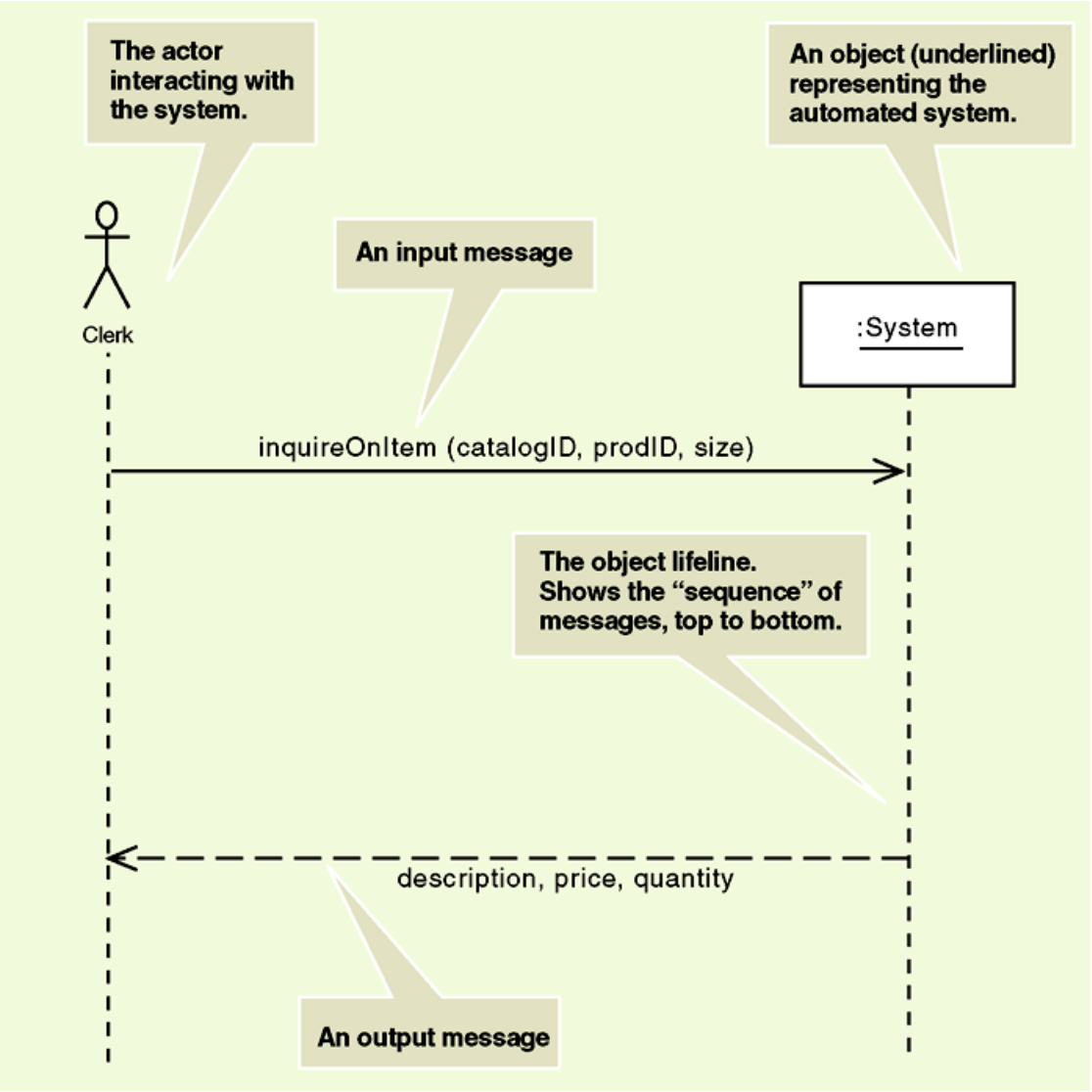
1. Start with elements from System Sequence Diagram (SSD) and first-cut Design Class Diagram
   - a Sequence Diagram uses all elements of an SSD .. System object replaced by all internal objects and messages
2. Replace the :System object with an appropriately named use case controller
3. For each input message
   - determine all internal messages that result from that input



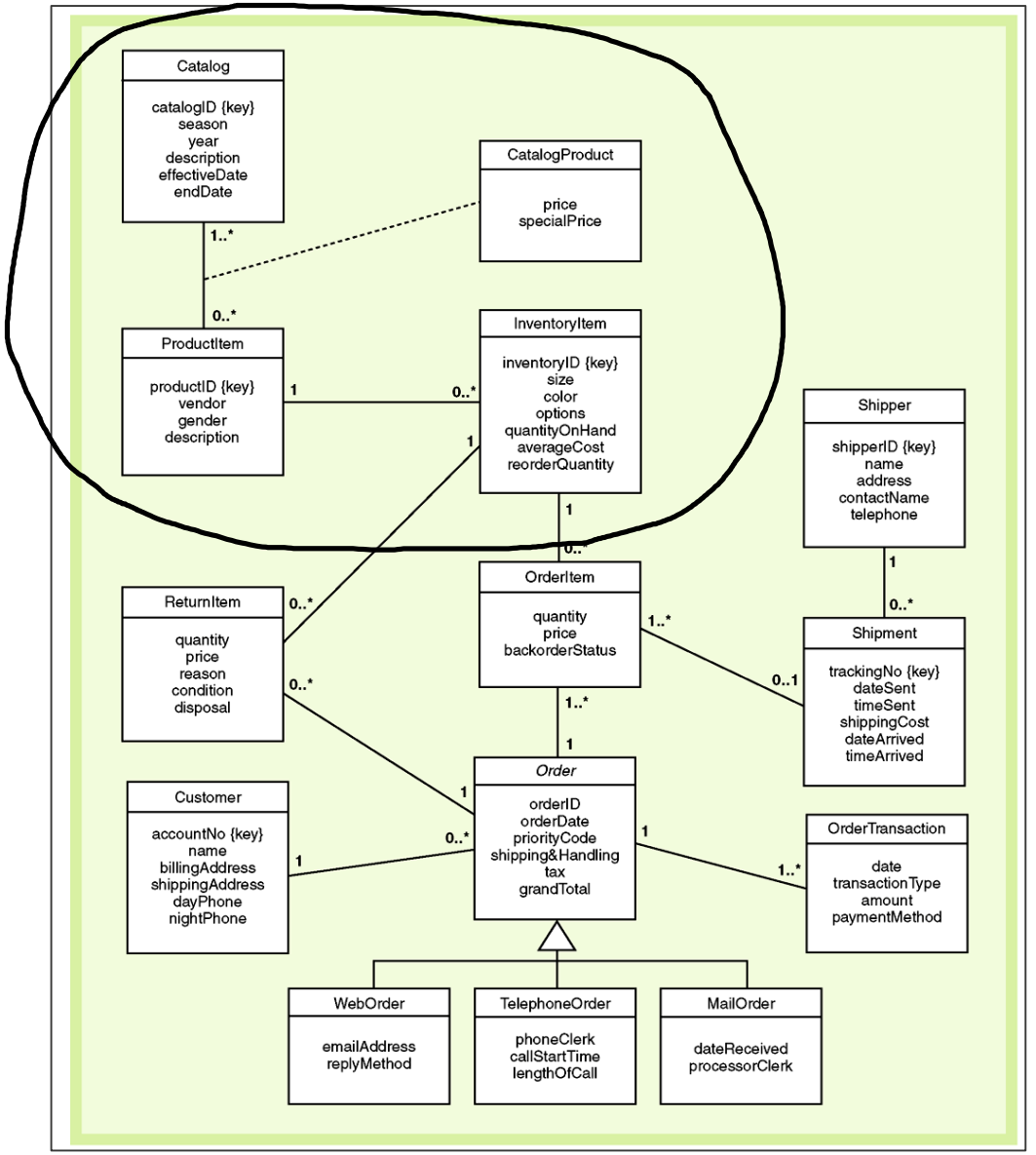
First-cut sequence diagram: How.3

4. For each internal message
   - Determine its objective, what information is needed
   - Identify the complete set of classes affected by the message
     - What class needs it (destination)
     - What class provides it (source)
     - Whether any objects are created as a result of the input
   - Flesh out the components for each message
     - Iteration, true/false conditions, return values, passed parameters

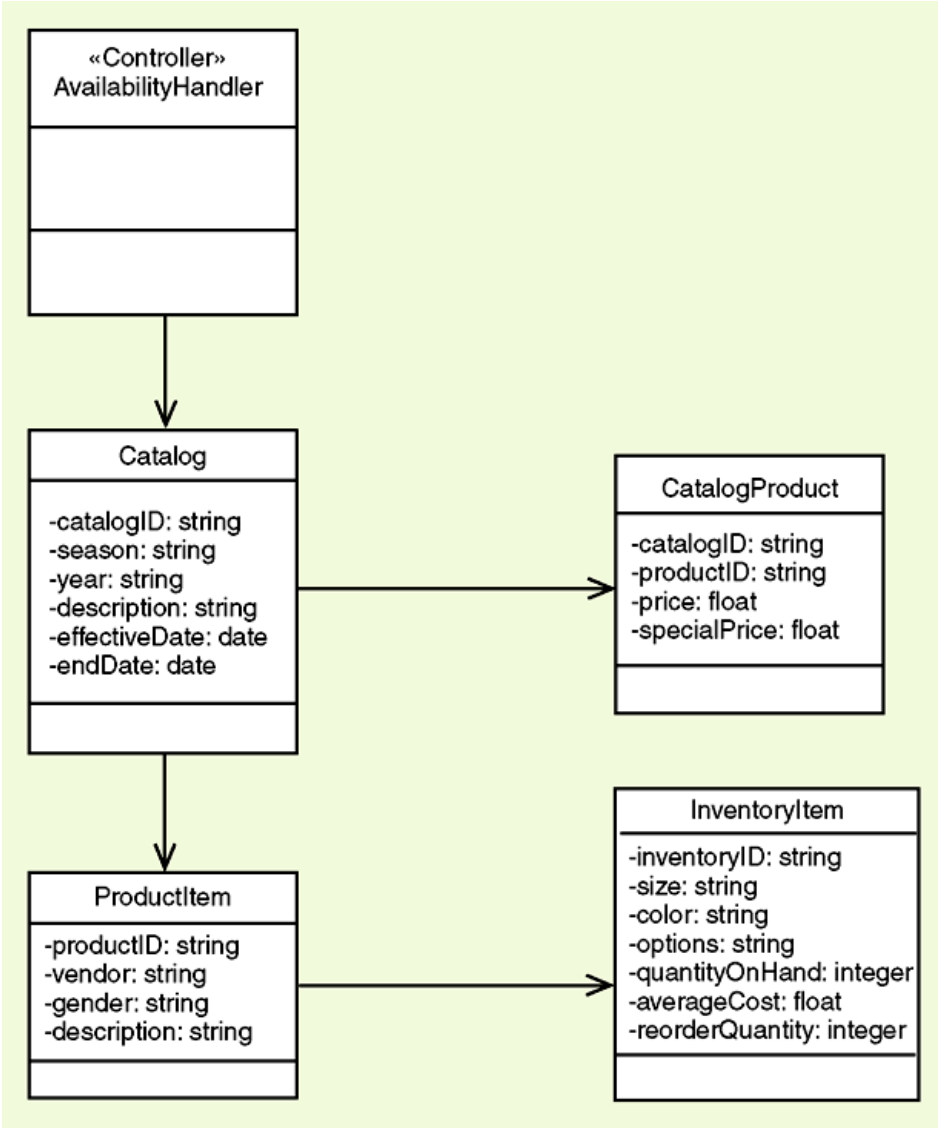
System Sequence Diagram (SSD) for 'look up item availability' use case



Domain class diagram for RMO sales subsystem



Partial design class diagram for the 'look up item availability' use case



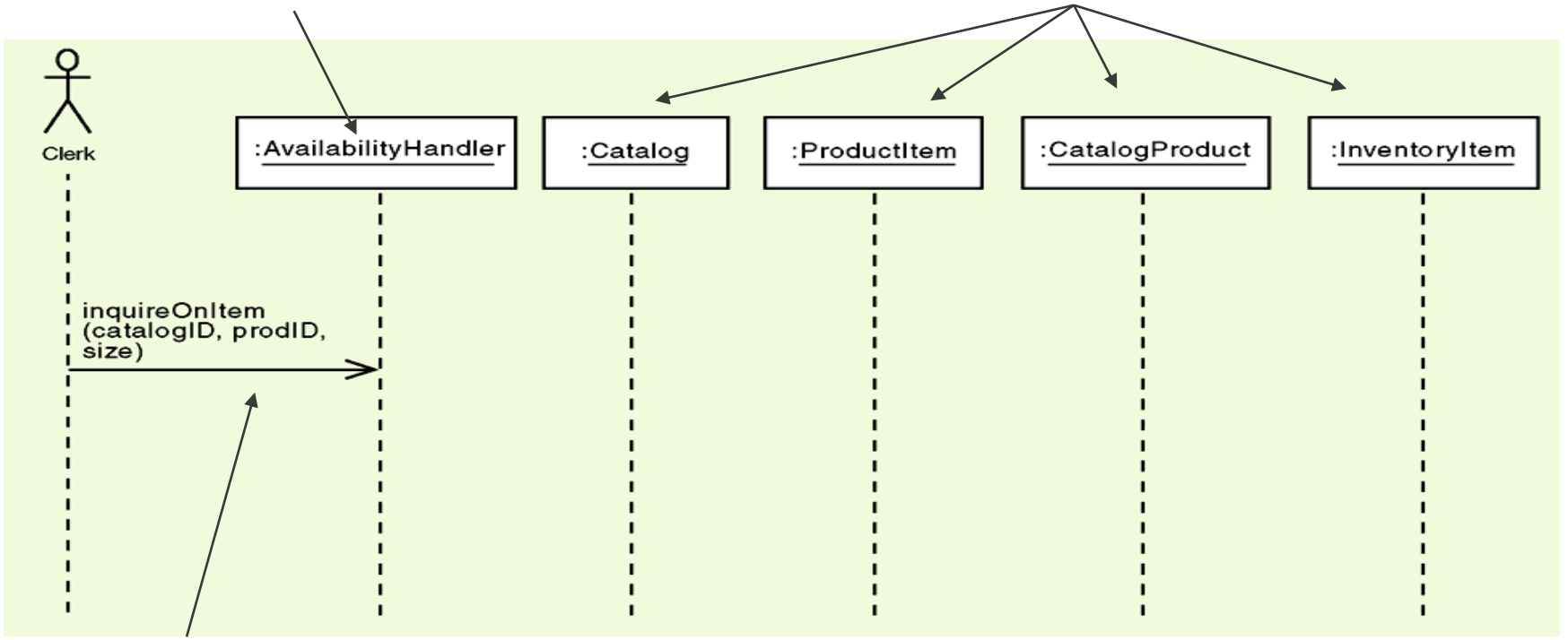
Utility class: Use case controller

- Design pattern - a repeatable approach to solving a problem .. a recipe
- Use case controller is one of the oldest software design pattern classes
  - a utility class - created to handle necessary system functions, tasks not part of domain object responsibility
  - designed as collection point for incoming messages
  - use case controller acts as intermediary between outside world and internal system
  - contains only that logic necessary to coordinate but not control the execution of the use case

Objects included in ‘look up item availability’ use case

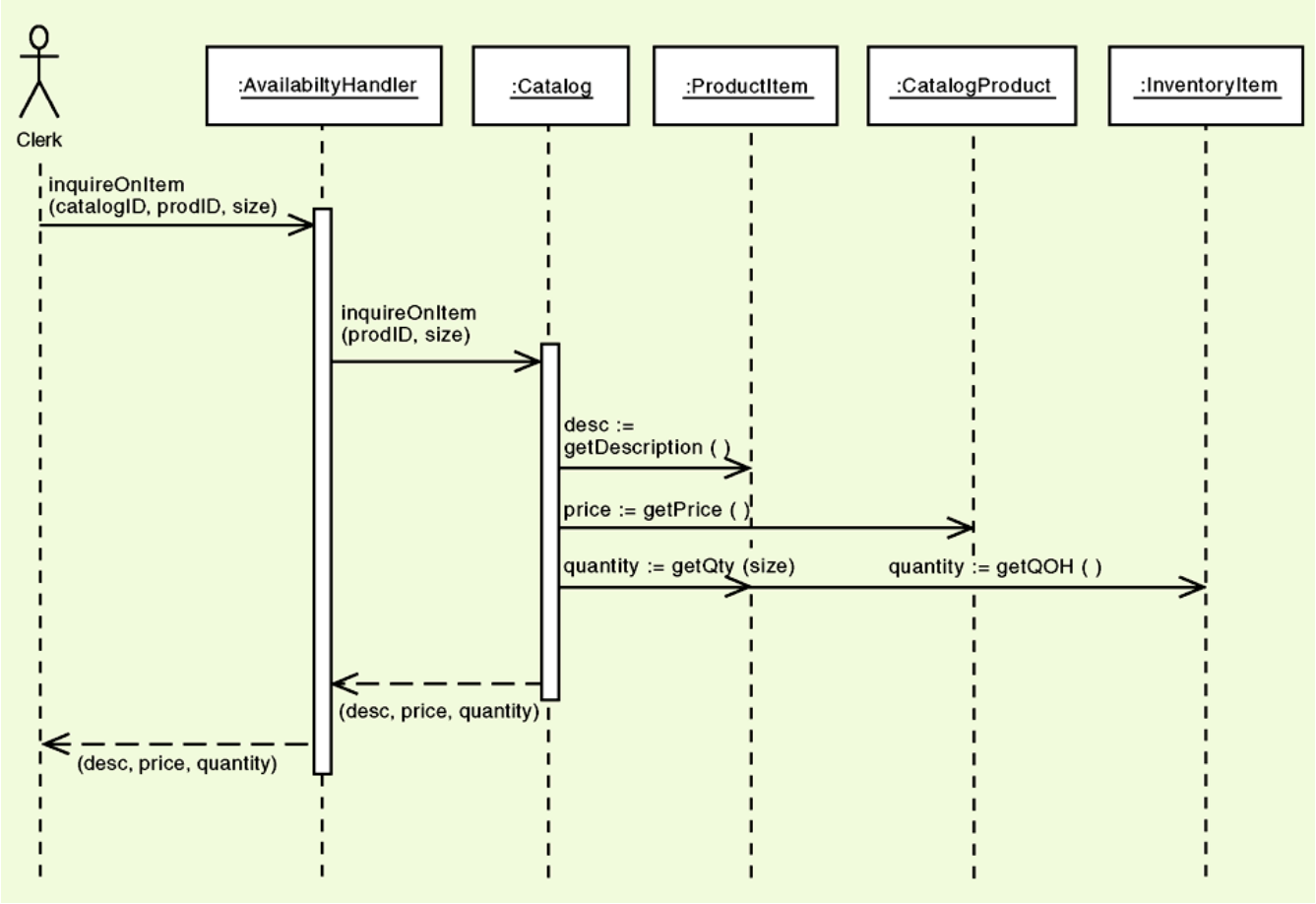
Replace :System with Use Case Controller

Instances of objects that have data we care about



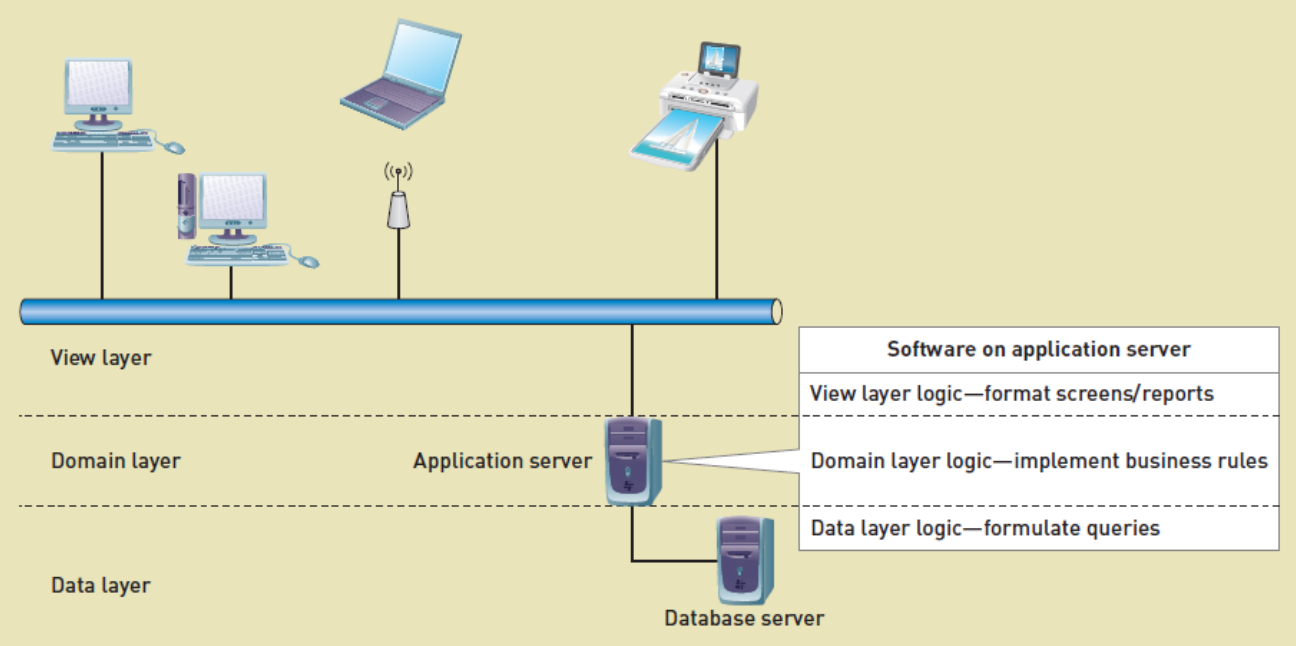
Same message that was originally sent to :System

First-cut sequence diagram for ‘look up item availability’ use case



Final cut sequence diagrams … multi-layer design – How?

- Final cut sequence diagrams represent a three-layer architectural design

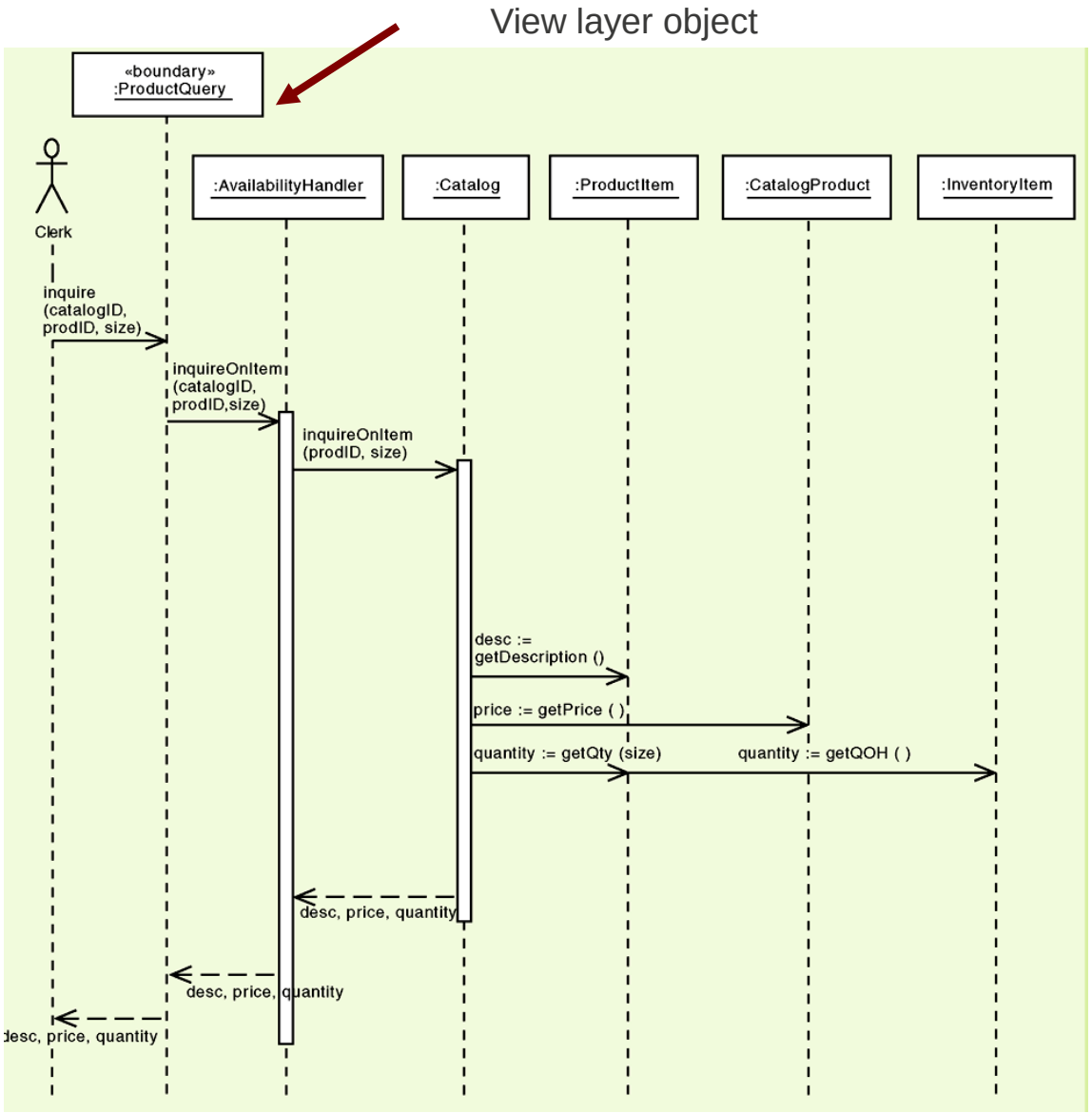


Final cut sequence diagrams … multi-layer design – How?

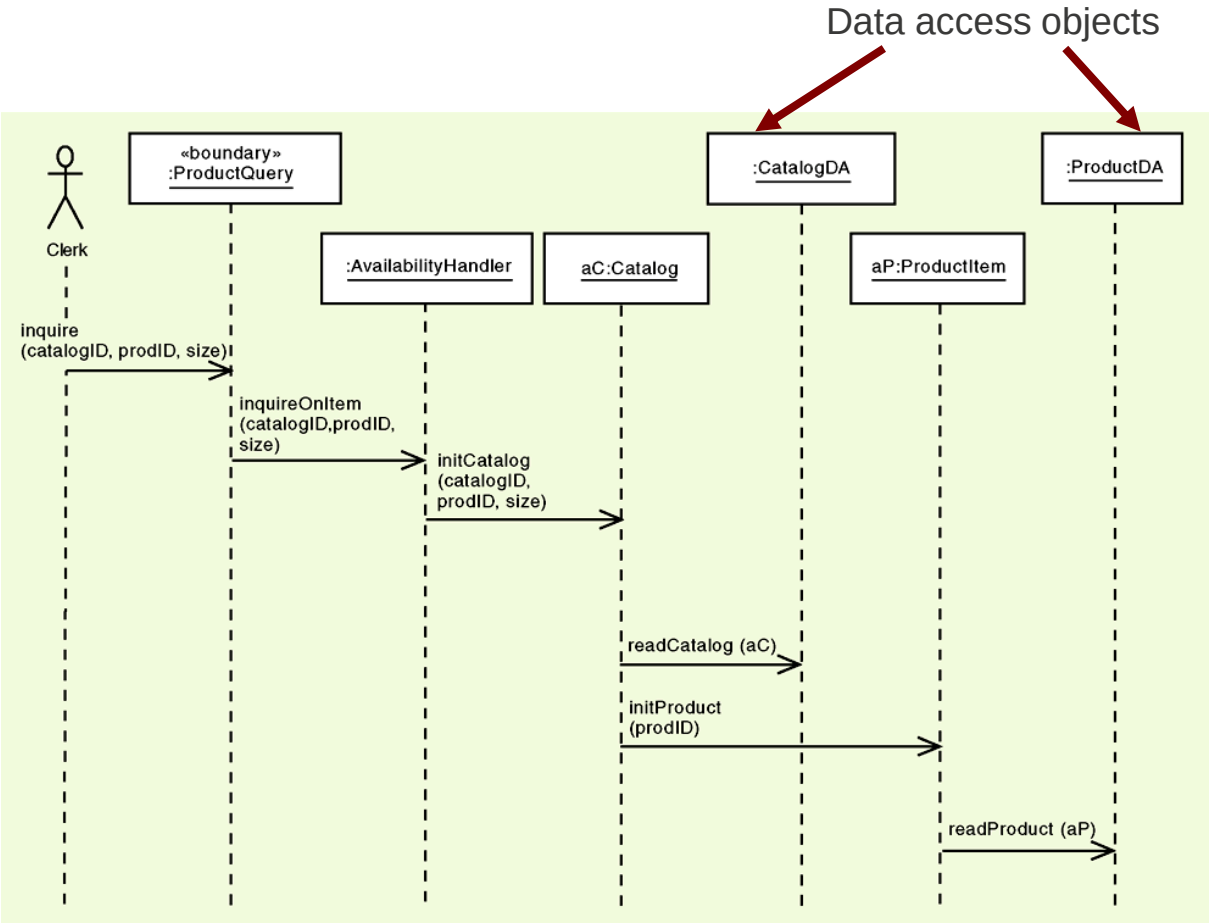
- Add a view layer interface class before the controller either as a single GUI class or as Windows classes
- Add a data access class for each problem domain class
  - data access layer should only support database CRUD operations so classes maintain a high level of cohesion and are loosely coupled with the business layer

4.2.3 Final cut sequence diagrams – Example

‘Look up item availability’ use case with view layer and user interface object

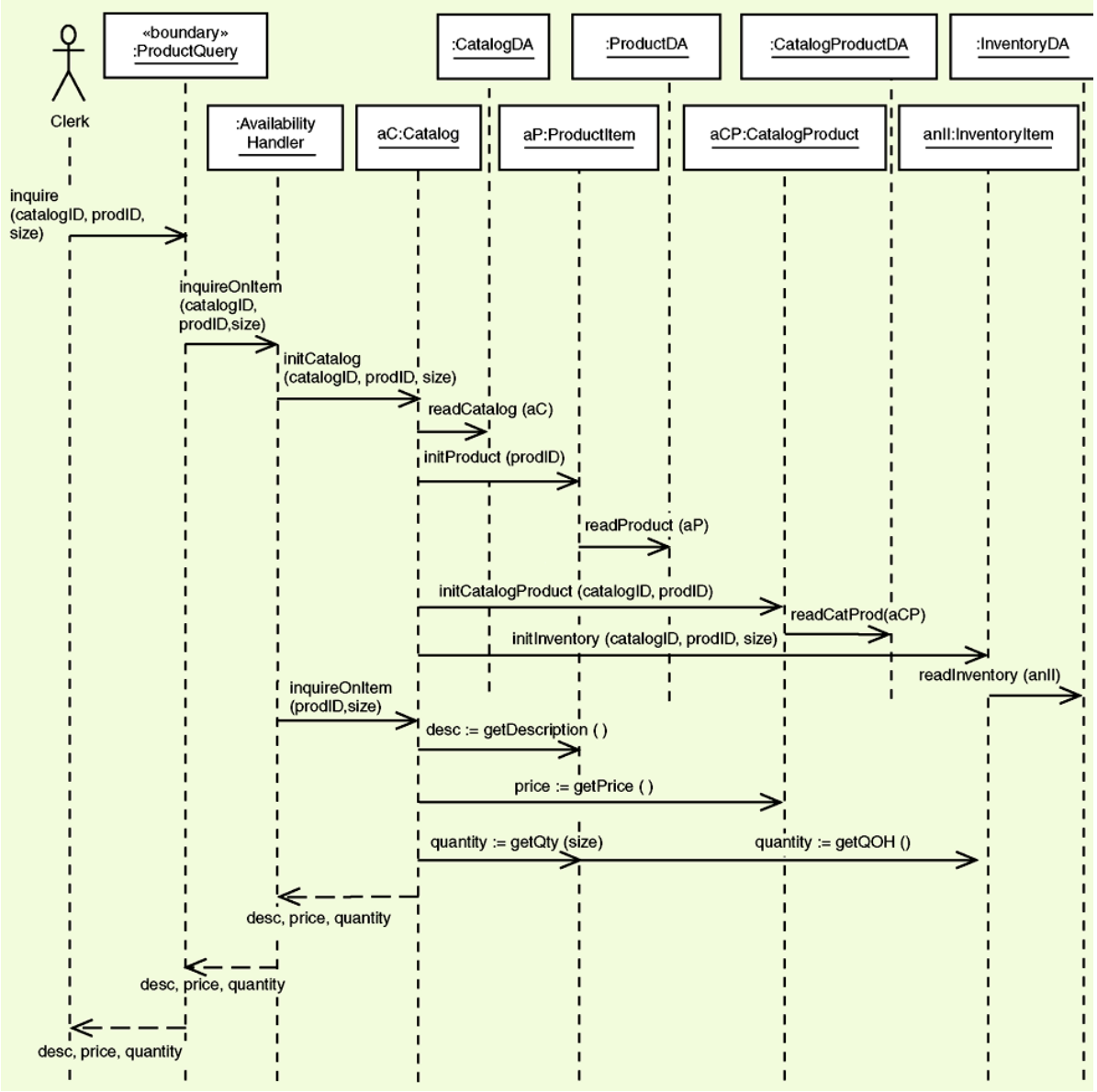


Partial three-layer design for ‘look up item availability’ use case with data layer



4.2.3 Final cut sequence diagrams – Example

Completed three-layer design for 'look up item availability' use case



Iterative OO Design process

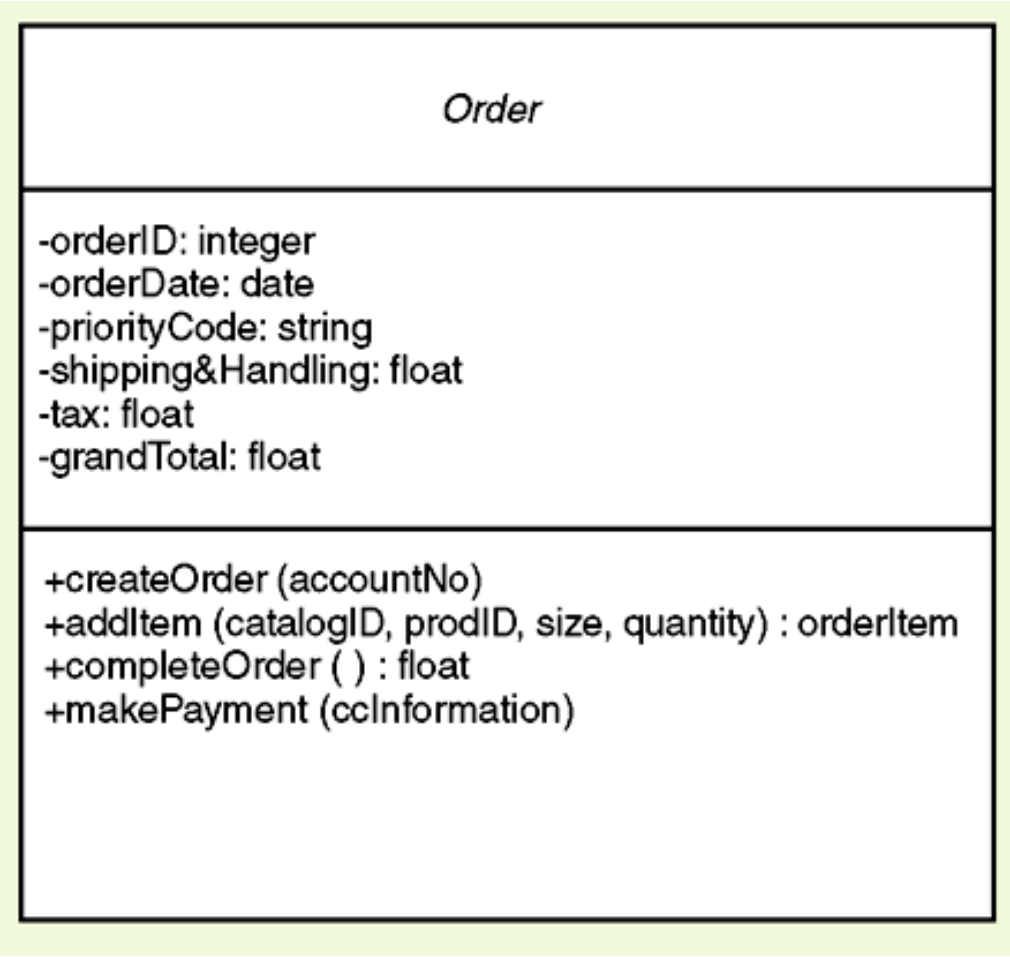
1. Create first-cut (preliminary) design class diagram
2. Create sequence diagrams, for each use case
   …realisation of use cases
3. **Return to design class diagram and update based on design of sequence diagrams**

Design class diagrams and detailed interaction diagrams inform each other and should be developed in parallel

Updating the design class diagram

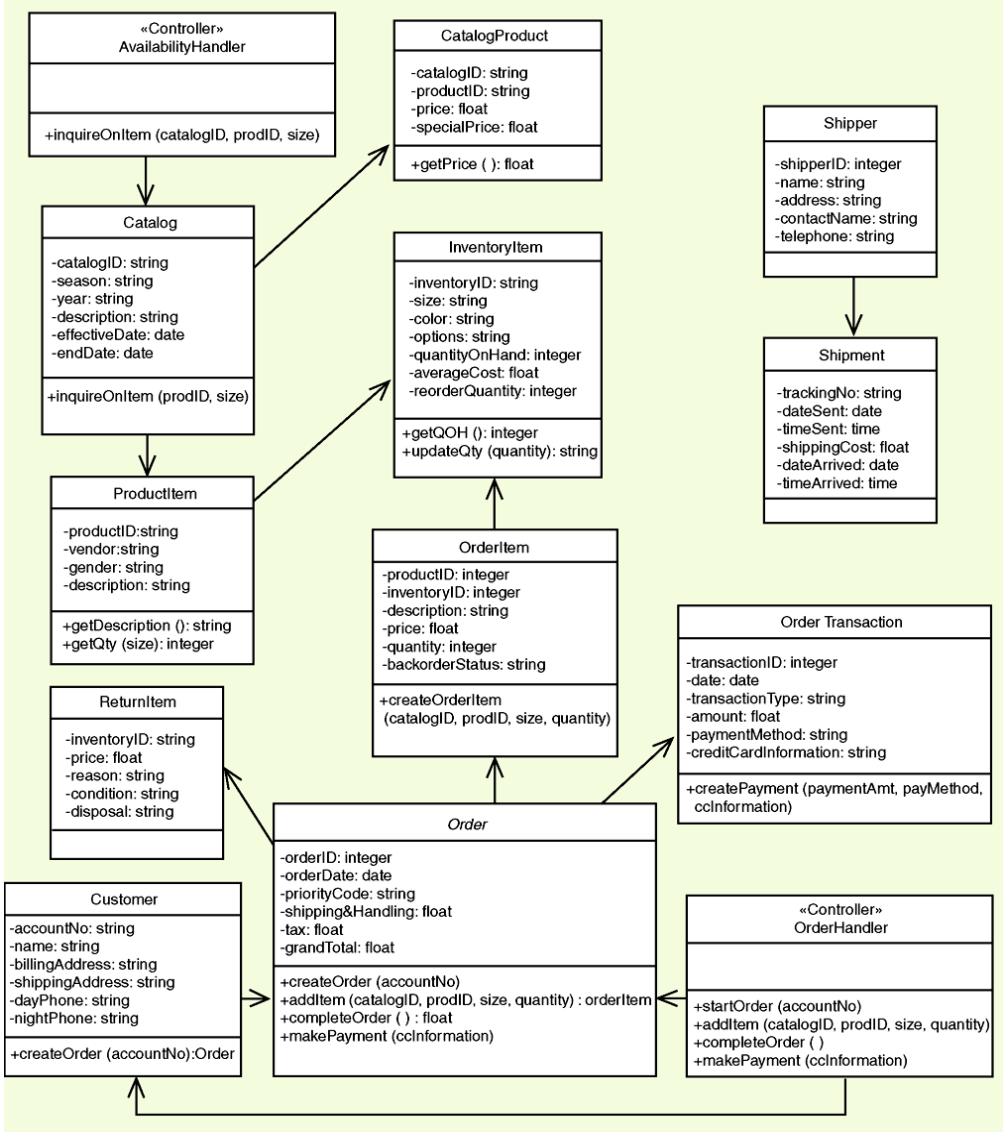
- Design class diagra**m**s developed for each layer
  - new classes for view layer and data access layer
  - new classes for domain layer use case controllers
- Sequence diagrams messages used to add methods to class diagram
  - constructor, data get/set methods, data access CRUD requests
  - any business-specific methods needed for use case
- Any additional attributes should be added
- Internal consistency of models is crucial

Design class with method signatures



4.3 Update Design Class Diagram - Example

Updated design class diagram for the domain layer



Summary

- Object-oriented design is the bridge between user requirements (in analysis models) and final system (constructed in programming language)
- Systems design is driven by use cases, and focuses on specifying system architecture in detail
  - design class diagrams and sequence diagrams
    - domain class diagrams are transformed into design class diagrams
- sequence diagrams are extensions of system sequence diagrams
- Object-oriented design principles must be applied
  - Coupling: connectivity between classes
  - Cohesion: unity of purpose of an individual class
- Three-layer design used to ensure maintainability

Workshop Preparation

- Watch Seminar 9 before the workshop

Thanks for watching

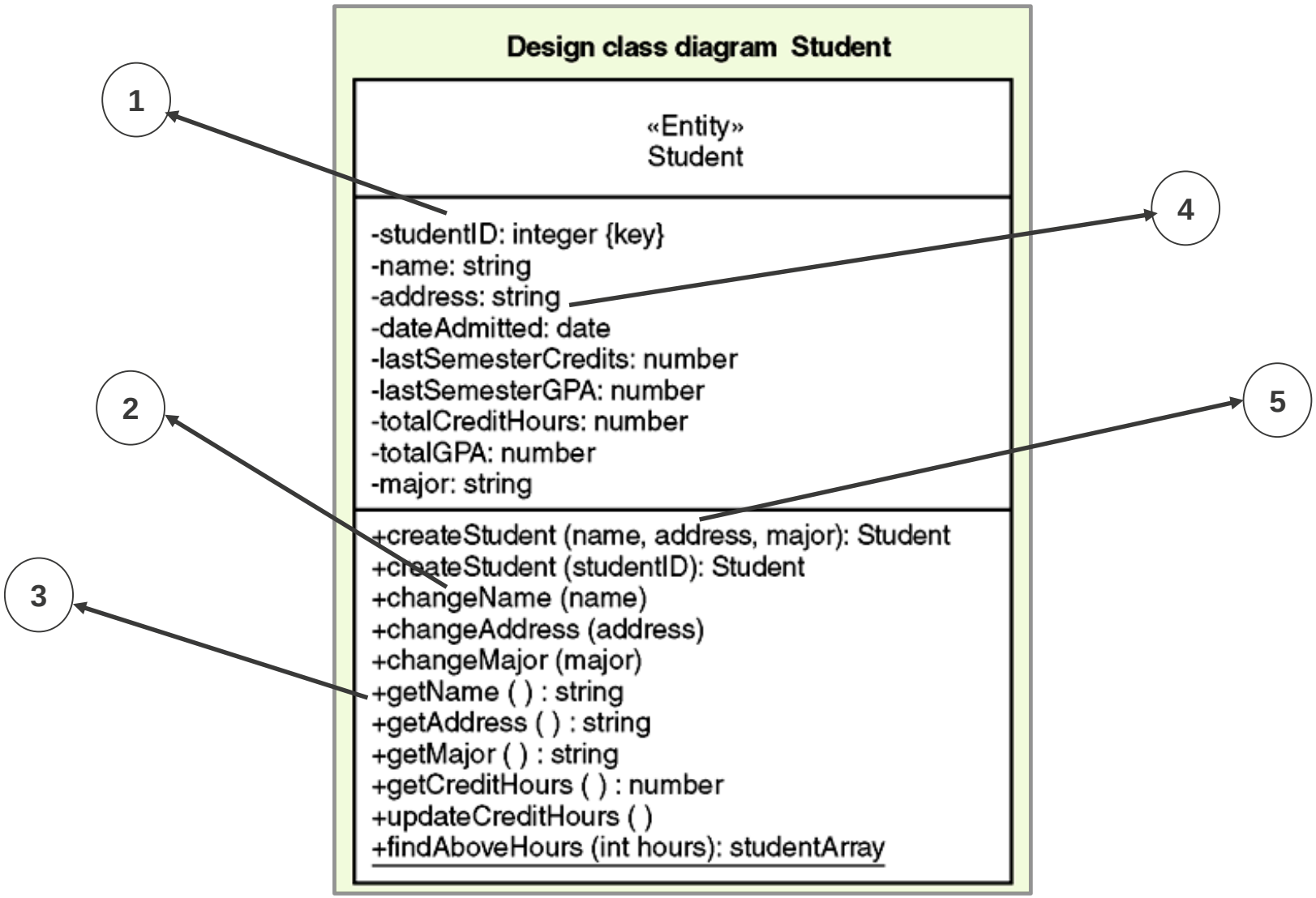
See you next week

Resources:

Prescribed text:

- Satzinger, J. W., Jackson, R.B., and Burd, S.D.(2016) Systems Analysis and Design in a Changing World, 7th Edition, Cengage Learning, Chapter 13 (pp. 398-424),
  - For Design Class Diagrams - Chapter 12 (376-382),
  - For System Sequence Diagrams Chapter 3 (139-146)

4.1 Create first-cut DCD: Notation – Methods



For next time

- Next time add types of message and the rectangle boxes on the lifelines

- Clearly explain the purpose of sequence diagrams