HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

System Analysis and Design

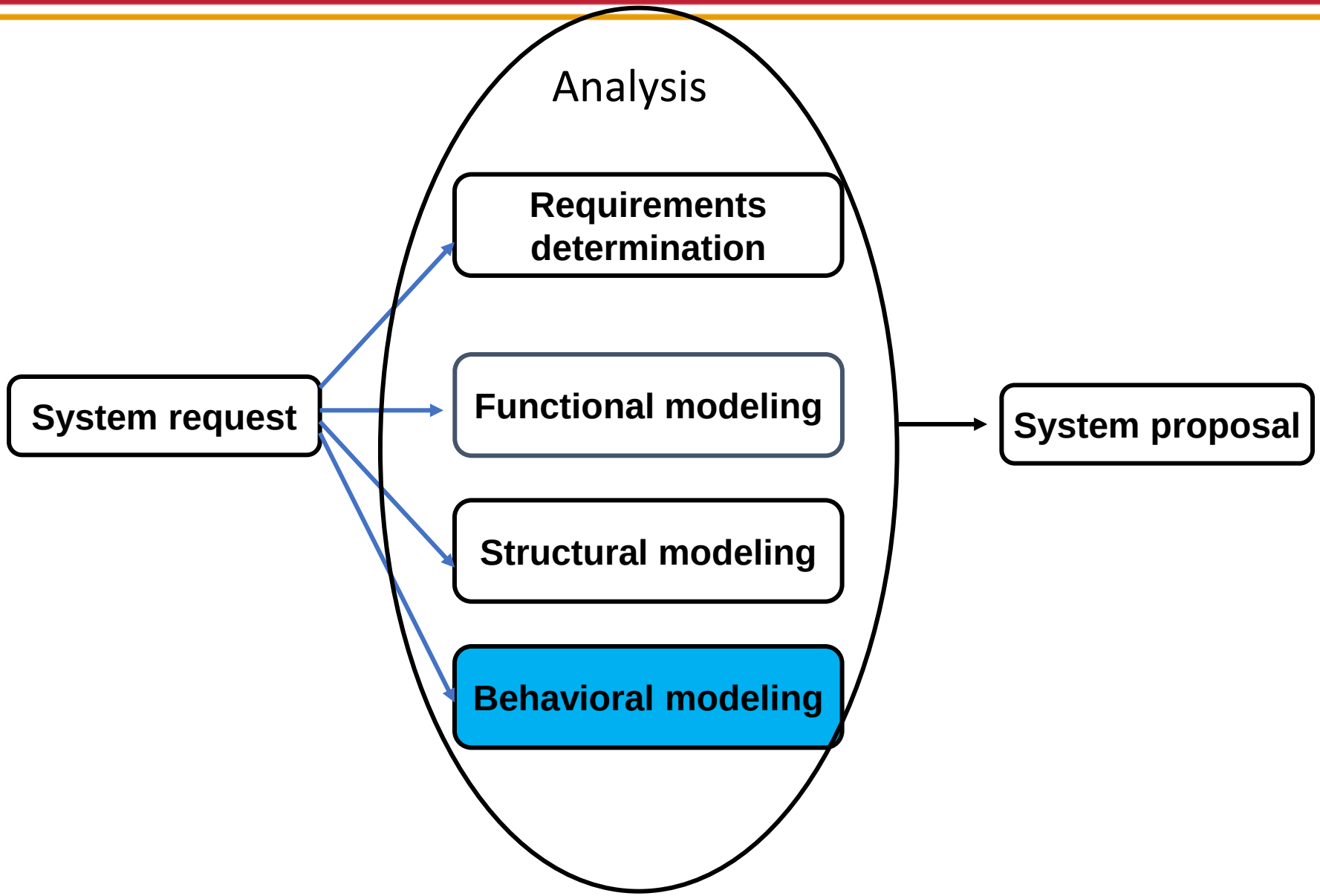
IT3120E

ONE LOVE. ONE FUTURE.

Part 2: System analysis

Chapter 7: *Behavioral modeling*

Part 2: System analysis



Chapter syllabus

- 7.1. Behavior Models
- 7.2. Interaction Diagrams
  - Sequence Diagram
  - Collaboration/Communication Diagram
- 7.3. State Machine Diagrams
  - Behavioral state machine diagram

7.1. Behavior Models

- Behavioral modeling describes how the identified objects in structural modeling interact and communicate to fulfill the uses cases.
  - In the development stage, the objects' behaviors are implemented as methods.

Functional Models	Structural Models	Behavioral Models
• External behavioral (functional) view	• Internal structural (static) view	• Internal behavior (dynamic) view
○ Use case diagram ○ Use case description ○ Activity diagram	○ CRC card ○ Class diagram ○ Object diagram	○ Sequence diagrams ○ Communication diagrams ○ State machine diagram ○ etc.

7.1. Behavior Models

- Behavioral models describe the *internal dynamic aspects* of the system
- Analysis phase:
  - describe what the internal logic of the process is without specifying how the process is implemented
- Design & Implementation phase:
  - fully specify the detailed design of the operations contained in the objects
- Typical behavioral models classification :
  - Interaction diagrams
    - To represent the distribution of the behavior of the system over the actors and objects in the system.
      - How actors and objects collaborate to provide the functionality defined in a use-case
    - *Sequence diagrams* and *Communication/Collaboration diagrams*
  - Behavioral state model
    - To represent the changes that occur in the underlying data
    - *Behavioral State Machine diagram*

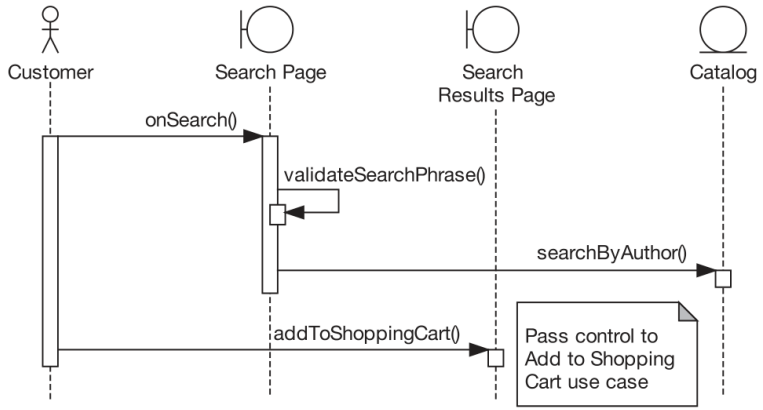
7.2. Interaction Diagrams

- Focus on object level
- *Sequence diagrams* and *Communication/Collaboration diagrams*
- Notions (rappel')
  - Objects: an instantiation of a class
  - Attributes: describe information about the object
  - Operation: the behaviors of an instance of a class
    - method/function call in Java/C++
    - message transfer in Ruby
    - Messages, methods, functions, operations, verbs, and controllers are all basically different terms of the same thing: the behavior that allocated to a class

Sequence Diagrams

Sequence Diagrams

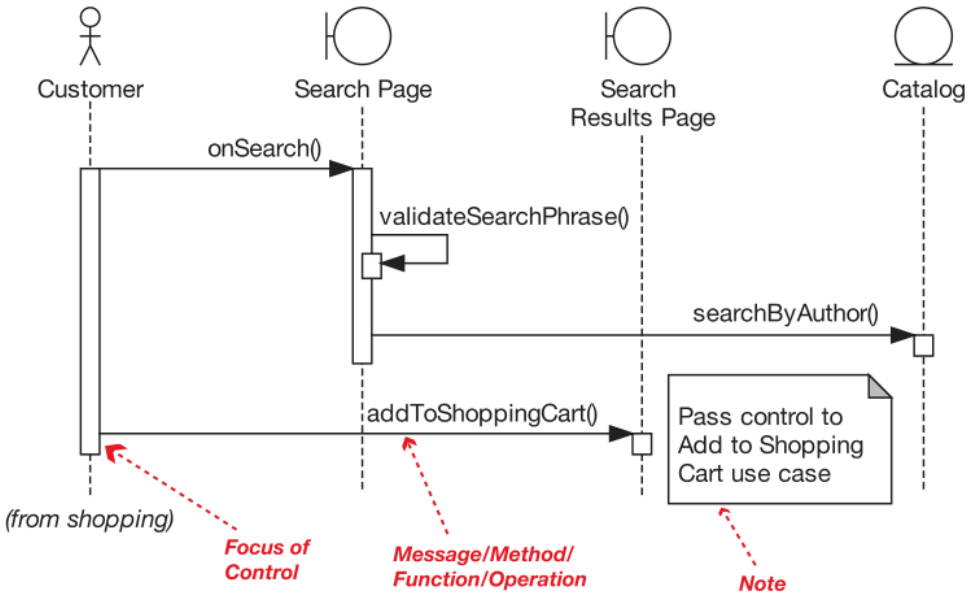
- *Sequence diagrams* illustrate the *objects* that participate in a use case and the *messages* that pass between them ***over time*** for *one* use case.



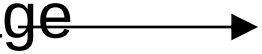
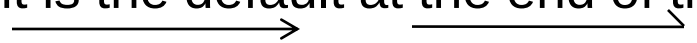
- Primary goals:
  - **Allocate behavior to objects/classes**
  - Show in detail how objects/classes interact with each other over the lifetime of the use case
  - Finalize the distribution of operations among classes
- Sequence diagram shows the design at a much more concrete and detailed view
  - It may show additional infrastructure objects, details of persistence storage mechanisms etc. depending on the architecture of the proposed system.

Sequence Diagram notation

- The objects across the top of the diagram are interacting with each other by passing messages back and forth
- The vertical dotted lines (or object life-lines) represent time
- Focus of control
  - indicate which object “has the focus,” or is currently in control
  - can be switched off to reduce the distraction
- Messages: arrows with title, can be numbered

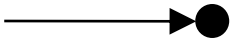


Type of messages

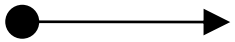
- Synchronous message 
  - An operation call.
  - The sender object transfers message to the receiver object, then pauses to wait for the receiver object to return control.
  - The receiver object performs the requested operation, if necessary, can transfer control to another object and when the operation is completed returns control to the sender object, possibly with a response result.
  - The return message *can be explicitly represented* by the dashed arrow  *or can be omitted*, since it is the default at the end of the operation. 
- Asynchronous message
  - Just sending of a signal/operation call.
  - The message enters the receiver's queue.
  - The sender continue his work without waiting for the results
  - The receiver performs an action and may also return a message to the sender. But if there is a return, it must be expressed explicitly.

Type of messages

- Lost message:
  - Known the sender, but don’t known the receiver
  - Receiver is out of description scope, or this is a distribution.



- Found message:
  - Know the receiver, but don’t known the sender
  - Sender is out of description scope

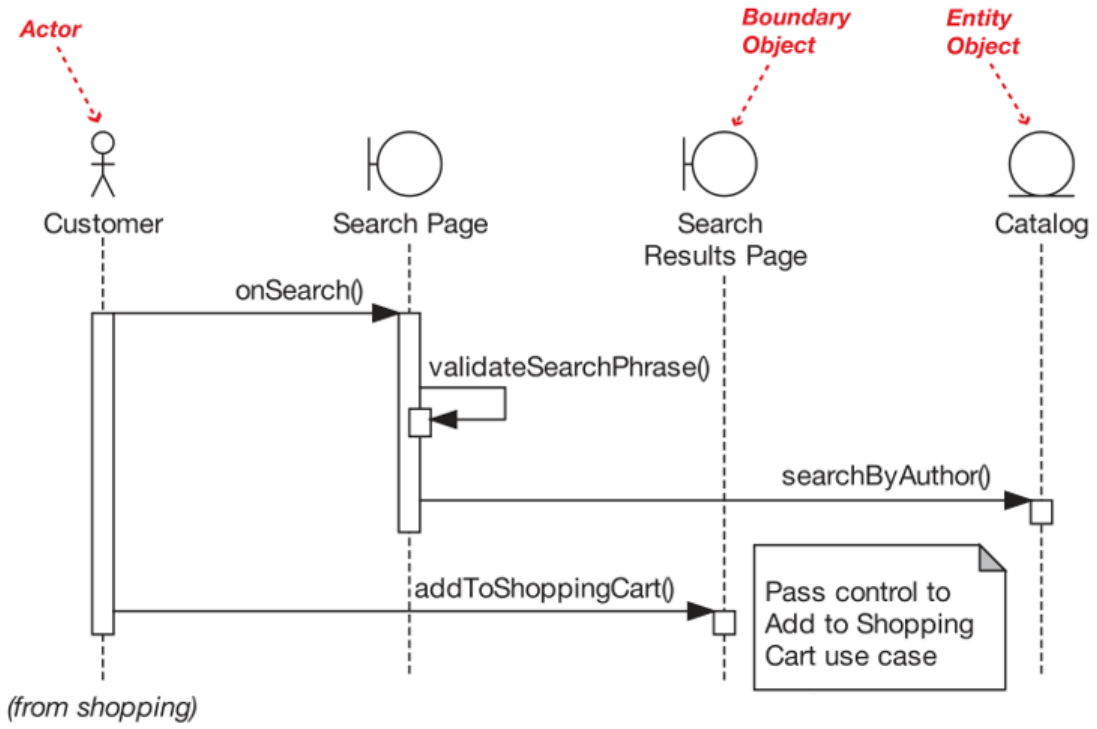


Sequence Diagram Building

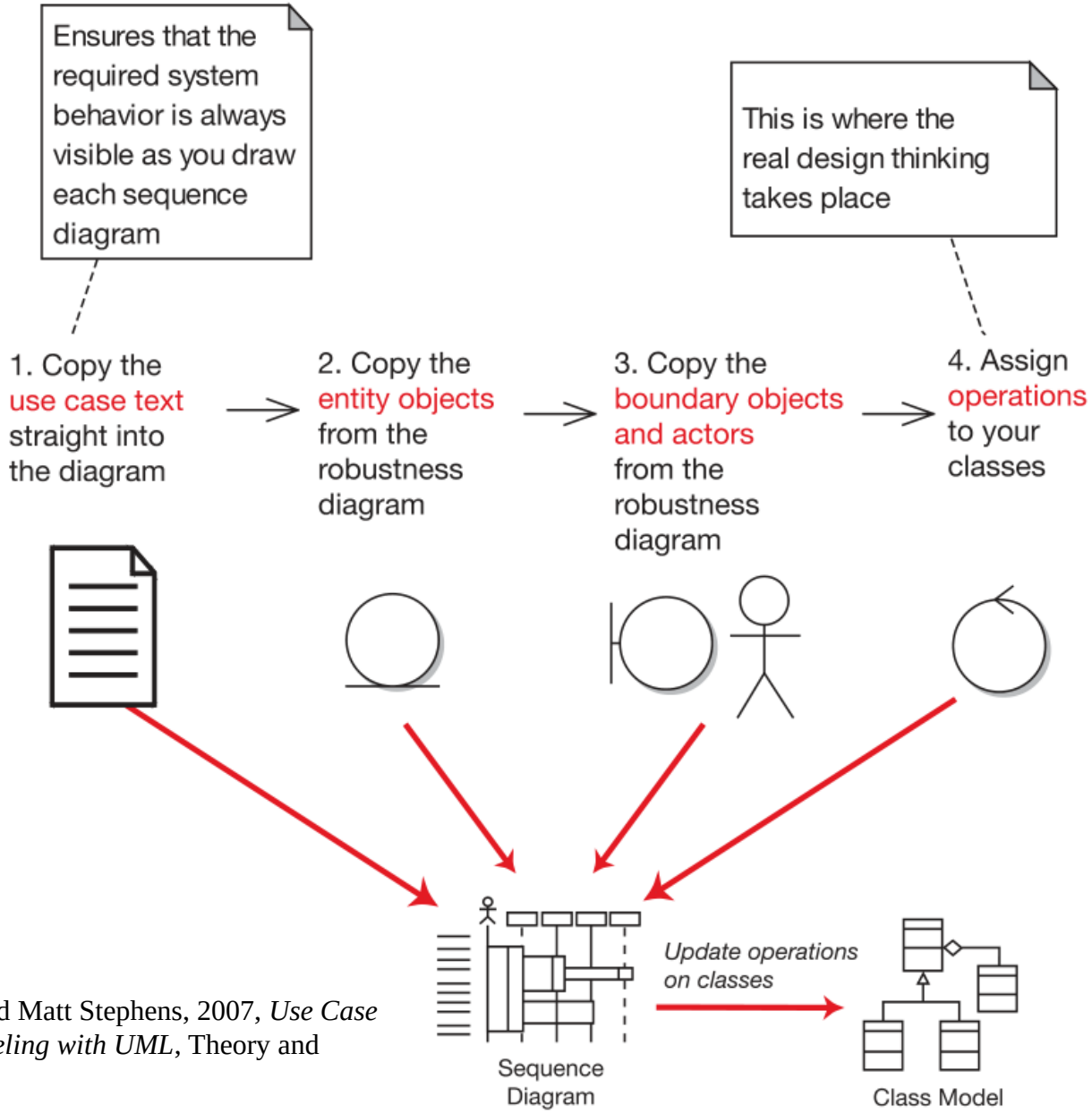
- There’s a direct link between each use case, its robustness diagram, and the sequence diagrams
  - The process of drawing the robustness diagram can rewrite the use case => use case text should be complete, correct, detailed, and explicit.
- Draw a sequence diagram for every use case, with both basic and alternate courses on the same diagram
- Map use case text to the messages on the sequence diagram
- Assign operations to classes while drawing messages and review class diagrams frequently

Sequence Diagram Building

- Objects: Actor, boundary object, entity object from robustness diagram
- Controller objects from robustness diagram
  - map to “logical” software functions
  - can be turned into 1 or more messages between objects (or occasionally a real controller class)
  - <=> allocating



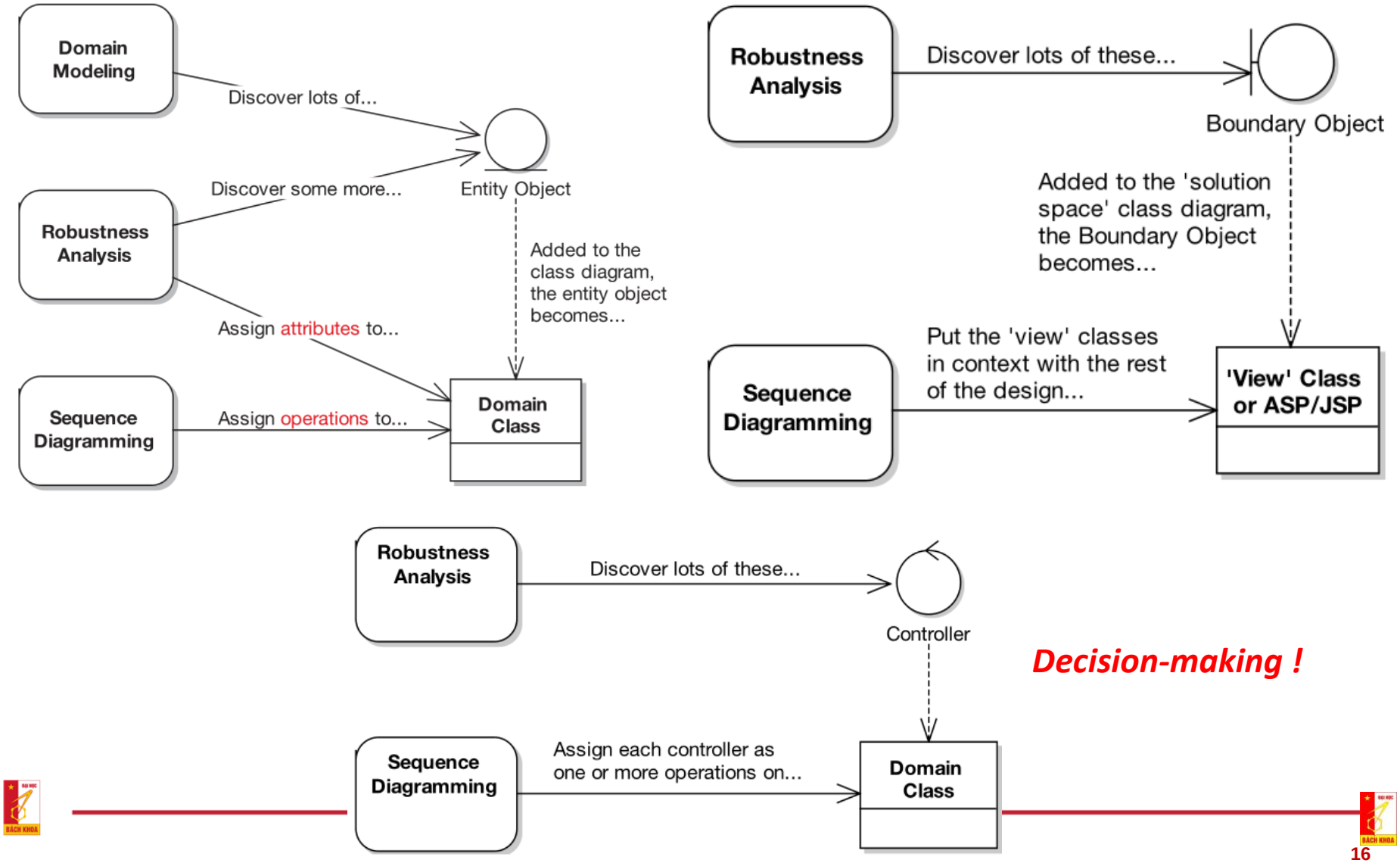
Sequence Diagram Steps



Doug Rosenberg and Matt Stephens, 2007, *Use Case Driven Object Modeling with UML*, Theory and Practice, Apress

Sequence Diagram Steps

- Primary goals: **Allocate behavior to objects/classes**



CASE STUDY: *Write Customer Review* sequence diagram

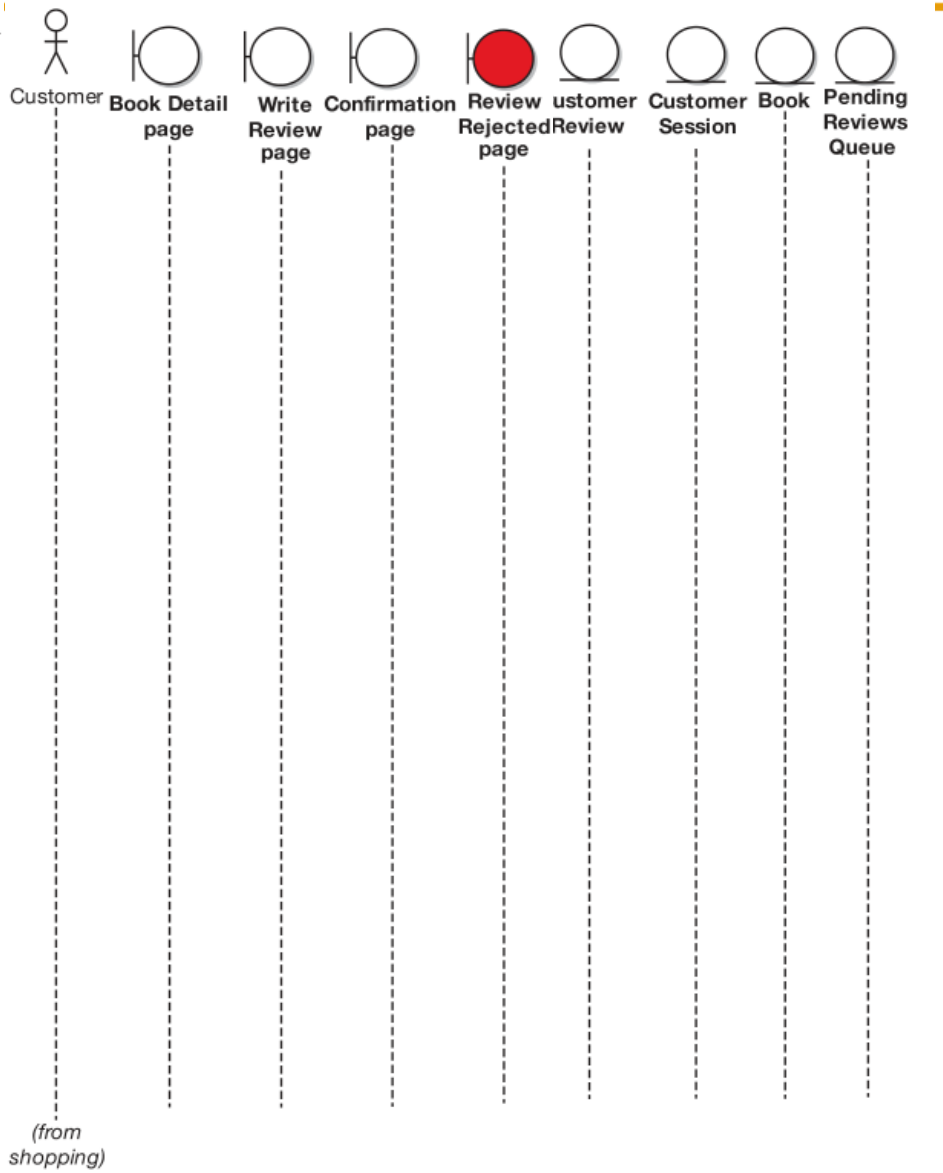
- Step 1: Copy Use case text
- Step 2: Copy entity objects
- Step 3: Copy boundary objects and actors

BASIC COURSE:
On the Book Detail page for the book currently being viewed, the Customer clicks the Write Review button. The system checks the Customer Session to make sure the Customer is logged in, and then displays the Write Review page. The Customer types in a Book Review, gives it a Book Rating out of 5 stars, and clicks the Send button. The system ensures that the Book Review isn't too long or short, and that the Book Rating is within 1-5 stars. The system then displays a confirmation page, and the review is added to the Pending Reviews Queue for moderation (this will be handled by the Moderate Customer Reviews use case).

ALTERNATE COURSES:
User not logged in: The user is first taken to the Login page, and then to the Write Review page once they've logged in.

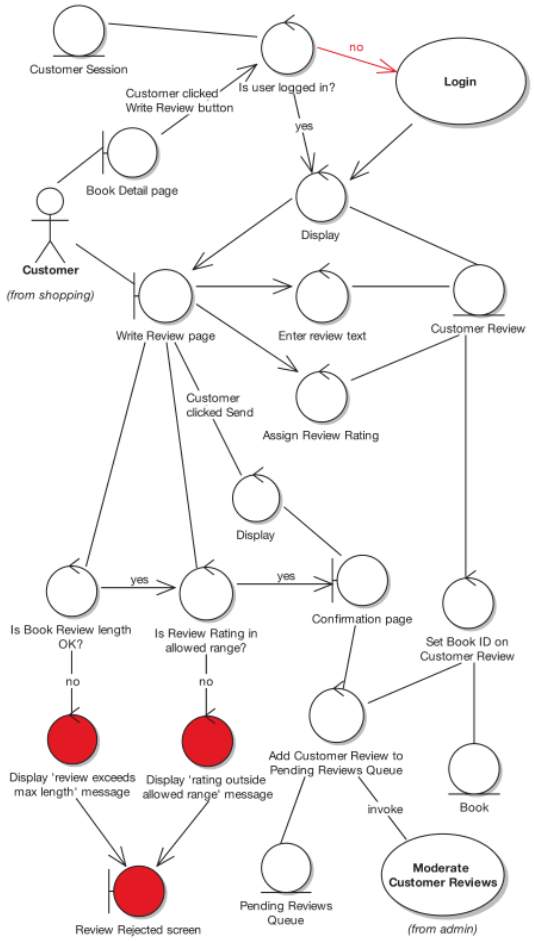
The user enters a review which is too long (text > 1MB): The system rejects the review, and responds with a message explaining why the review was rejected.

The review is too short (< 10 characters): The system rejects the review.



CASE STUDY

- Step 4: Walk through the controllers

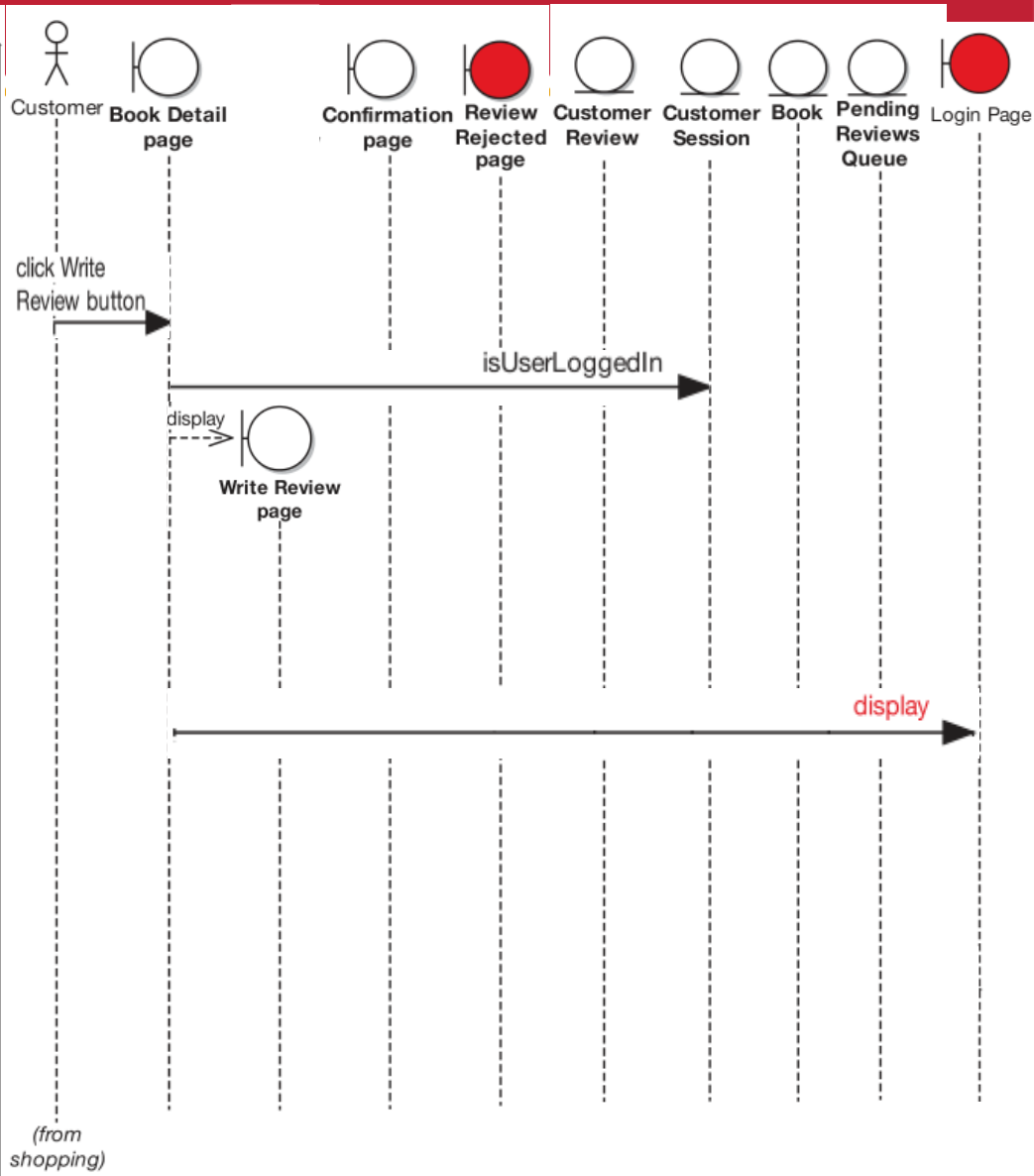


BASIC COURSE:
On the Book Detail page for the book currently being viewed, the Customer clicks the Write Review button. The system checks the Customer Session to make sure the Customer is logged in, and then displays the Write Review page. The Customer types in a Book Review, gives it a Book Rating out of 5 stars, and clicks the Send button. The system ensures that the Book Review isn't too long or short, and that the Book Rating is within 1-5 stars. The system then displays a confirmation page, and the review is added to the Pending Reviews Queue for moderation (this will be handled by the Moderate Customer Reviews use case).

ALTERNATE COURSES:
User not logged in: The user is first taken to the Login page, and then to the Write Review page once they've logged in.

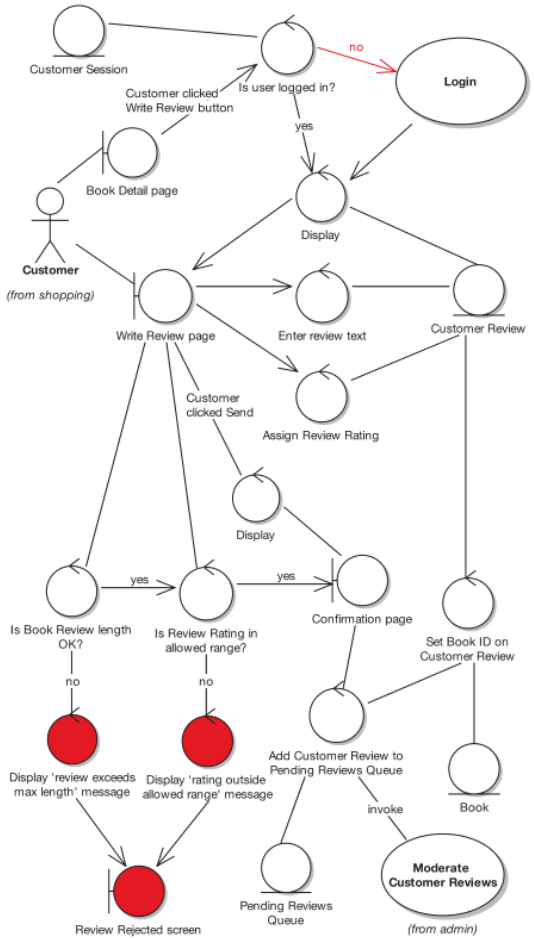
The user enters a review which is too long (text > 1MB): The system rejects the review, and responds with a message explaining why the review was rejected.

The review is too short (< 10 characters): The system rejects the review.



CASE STUDY

- Step 4: Walk through the controllers



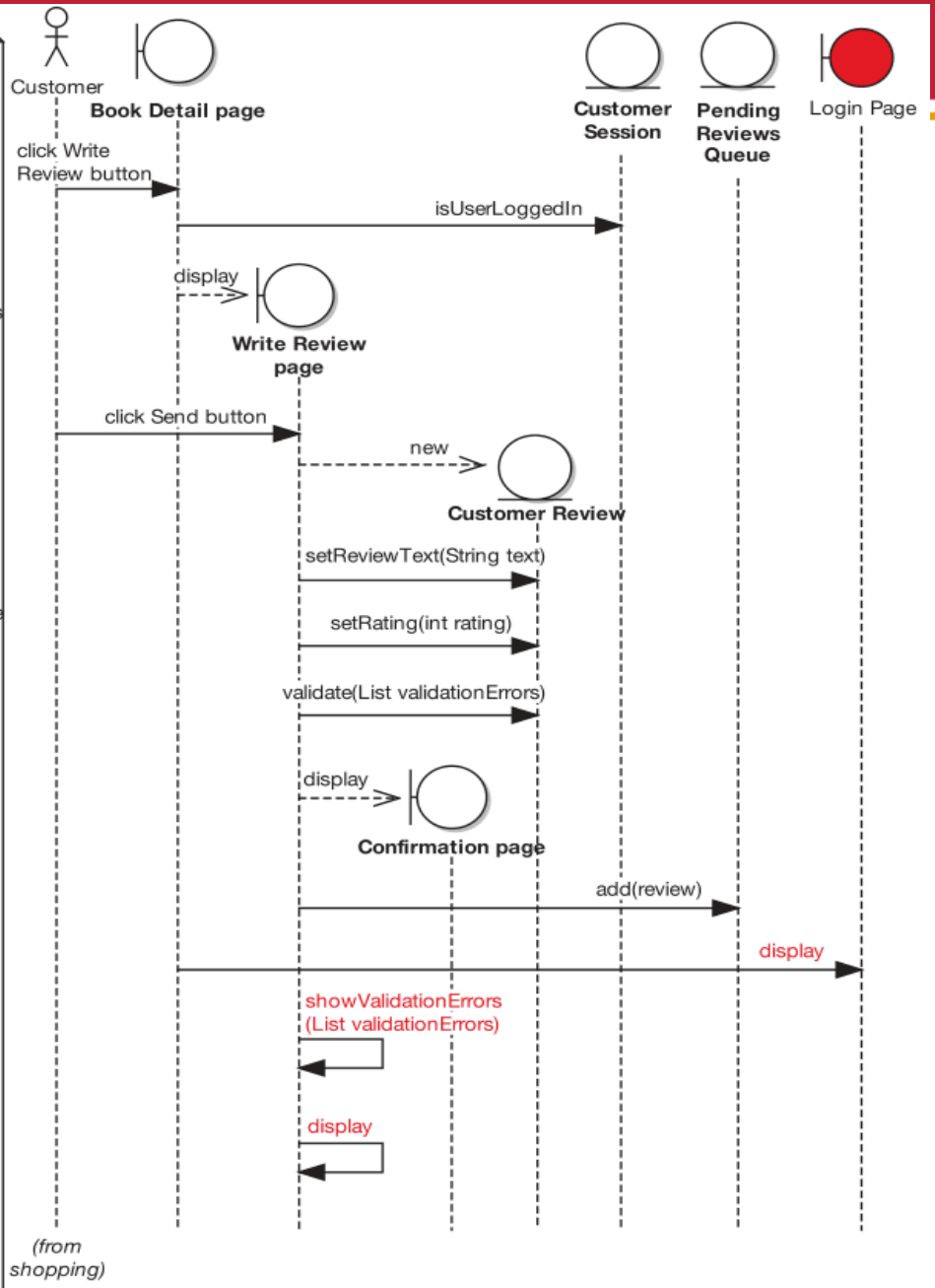
BASIC COURSE:
On the Book Detail page for the book currently being viewed, the Customer clicks the Write Review button. The system checks the Customer Session to make sure the Customer is logged in, and then displays the Write Review page. The Customer types in a Book Review, gives it a Book Rating out of 5 stars, and clicks the Send button. The system ensures that the Book Review isn't too long or short, and that the Book Rating is within 1-5 stars. The system then displays a confirmation page, and the review is added to the Pending Reviews Queue for moderation (this will be handled by the Moderate Customer Reviews use case).

ALTERNATE COURSES:

User not logged in: The user is first taken to the Login page, and then to the Write Review page once they've logged in.

The user enters a review which is too long (text > 1MB): The system rejects the review, and responds with a message explaining why the review was rejected.

The review is too short (< 10 characters): The system rejects the review.



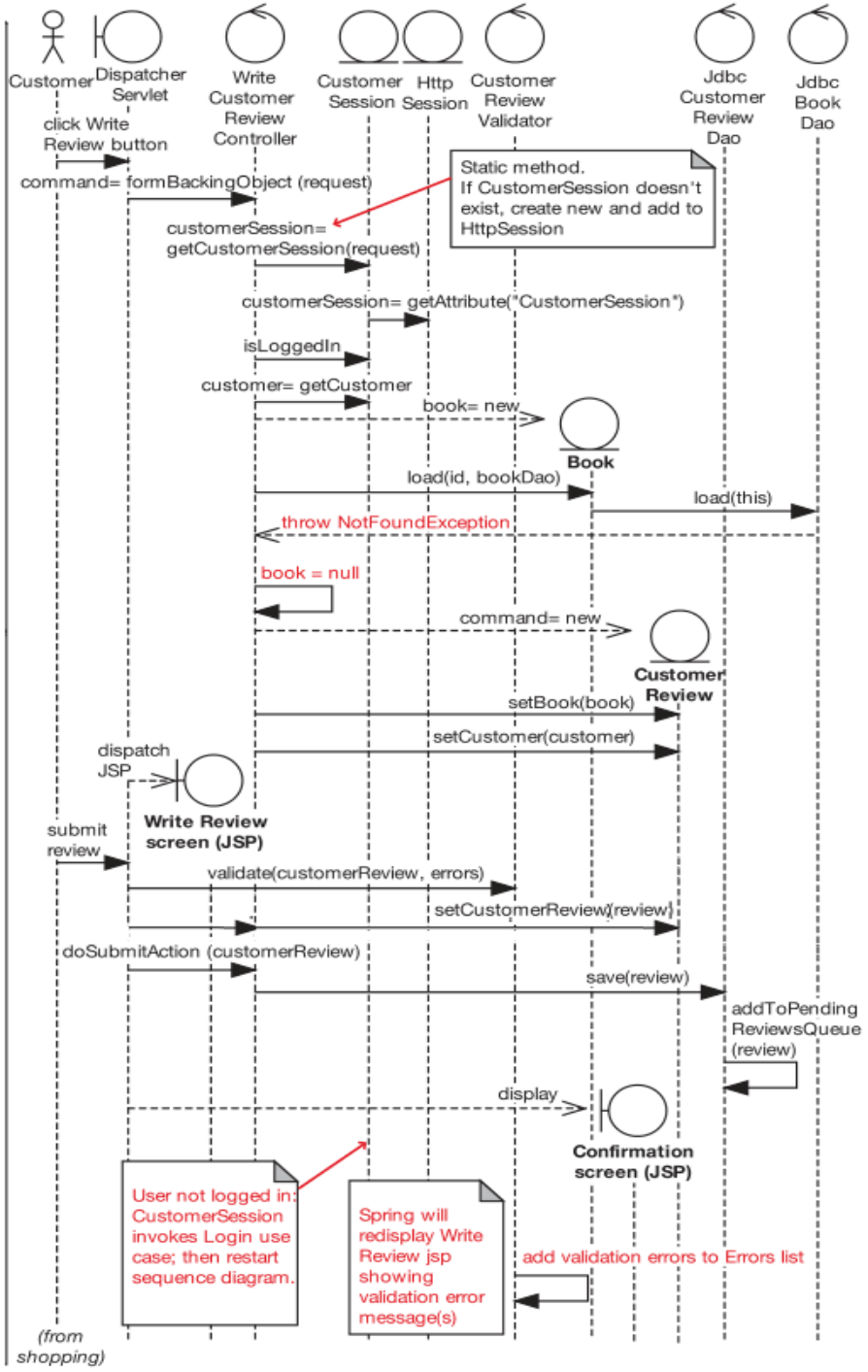
" a pure OO version"

CASE STUDY

- Can be extended
- Tie the design closely with the implementation details of the target framework before begin coding

Spring Framework

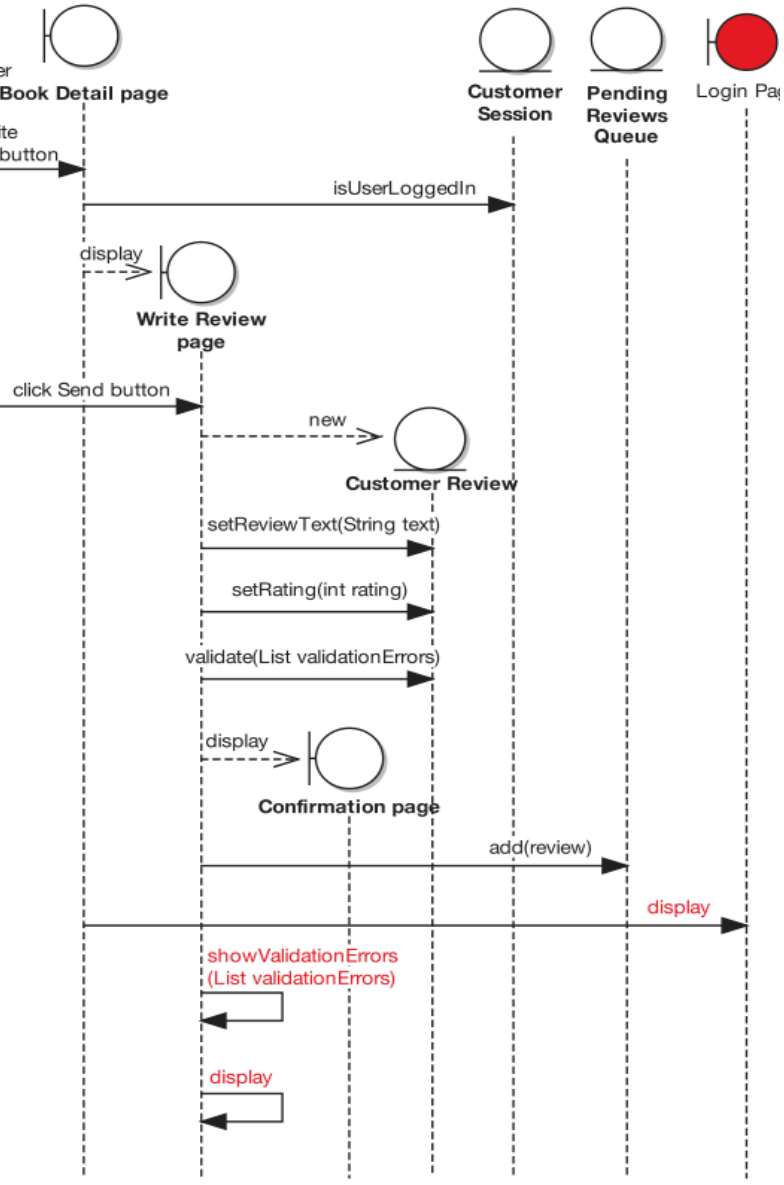
" a framework dependent version"

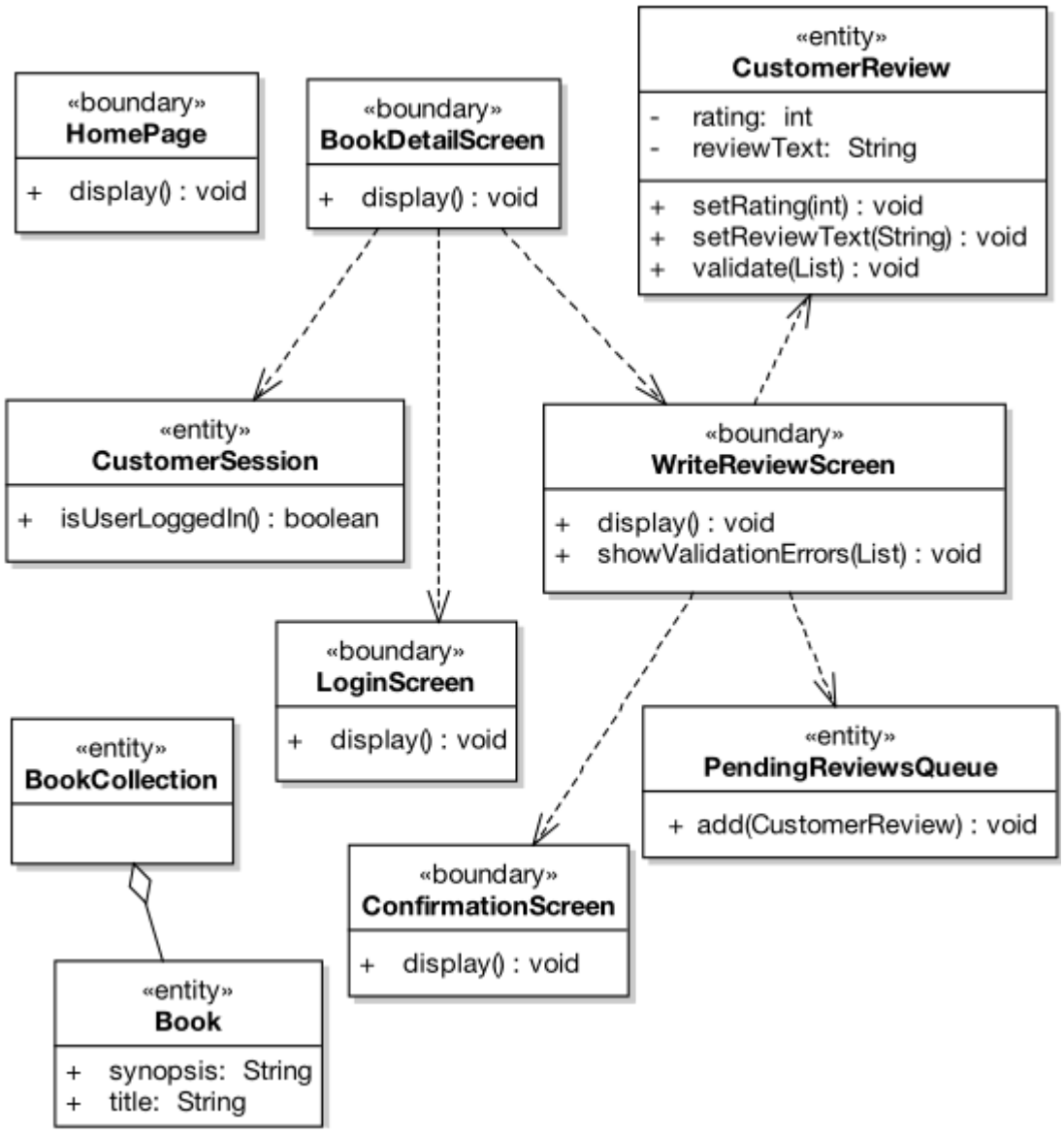


Sequence Diagram Building

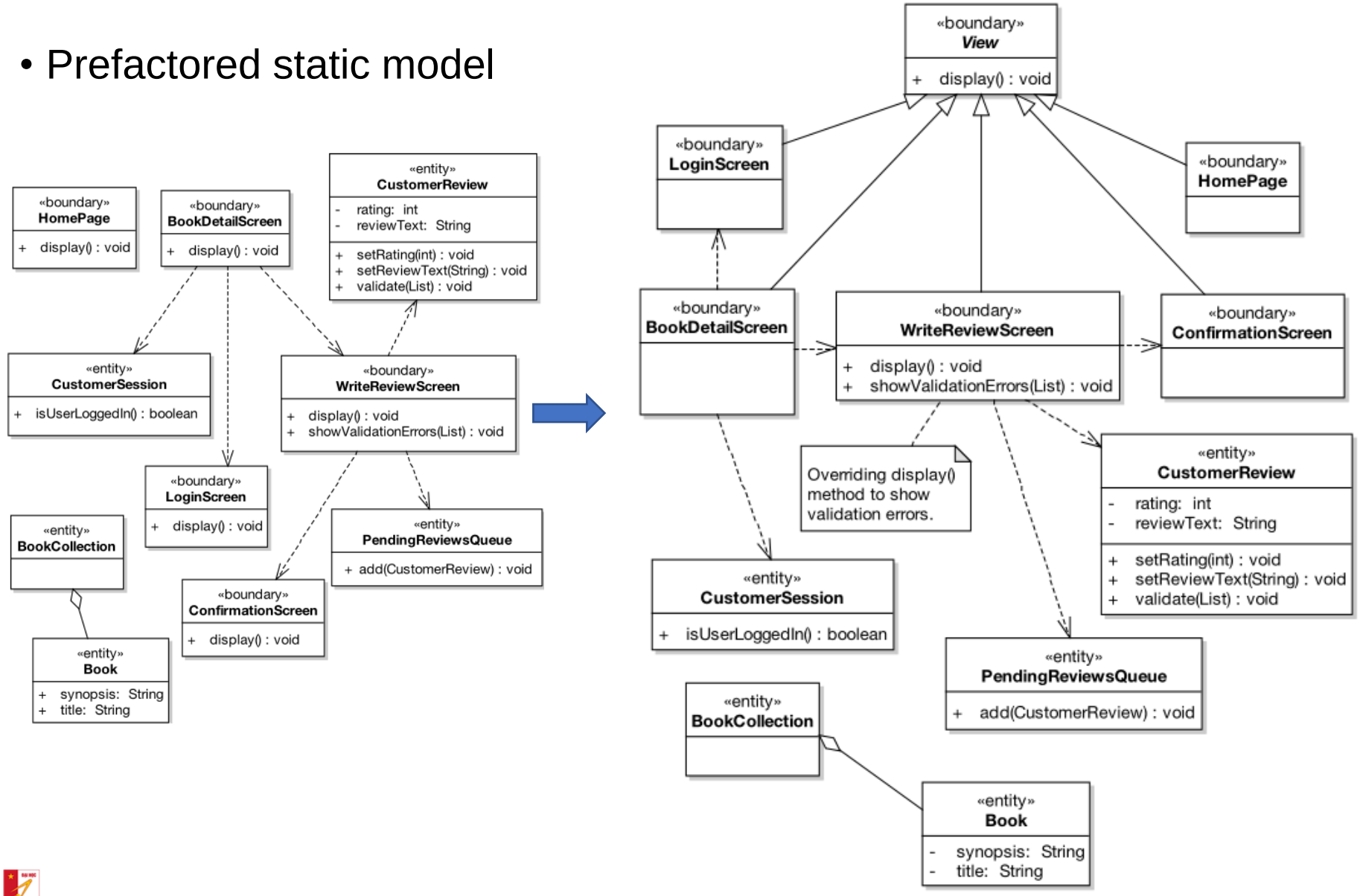
- Keep updating and refining the static model (the class diagrams)
  - These detailed class diagrams should use the same elements as on the sequence diagrams
  - When assigning a message on the sequence diagram (the dynamic model), an operation is automatically added to the appropriate class in the static model.
- Update the use case text if needed

CASE STUDY


Book Detail page
Customer Session
Pending Reviews Queue
Login Pa
button
isUserLoggedIn
display
Write Review page
click Send button
new
Customer Review
setReviewText(String text)
setRating(int rating)
validate(List validationErrors)
display
Confirmation page
add(review)
display
showValidationErrors (List validationErrors)
display


«boundary» HomePage
+ display() : void
«boundary» BookDetailScreen
+ display() : void
«entity» CustomerReview
- rating: int
- reviewText: String
+ setRating(int) : void
+ setReviewText(String) : void
+ validate(List) : void
«entity» CustomerSession
+ isUserLoggedIn() : boolean
«boundary» WriteReviewScreen
+ display() : void
+ showValidationErrors(List) : void
«boundary» LoginScreen
+ display() : void
«entity» BookCollection
«entity» PendingReviewsQueue
+ add(CustomerReview) : void
«boundary» ConfirmationScreen
+ display() : void
«entity» Book
+ synopsis: String
+ title: String

- Prefactored static model



Quiz

Figure out the errors and then draw the corrected diagram

- Question 1

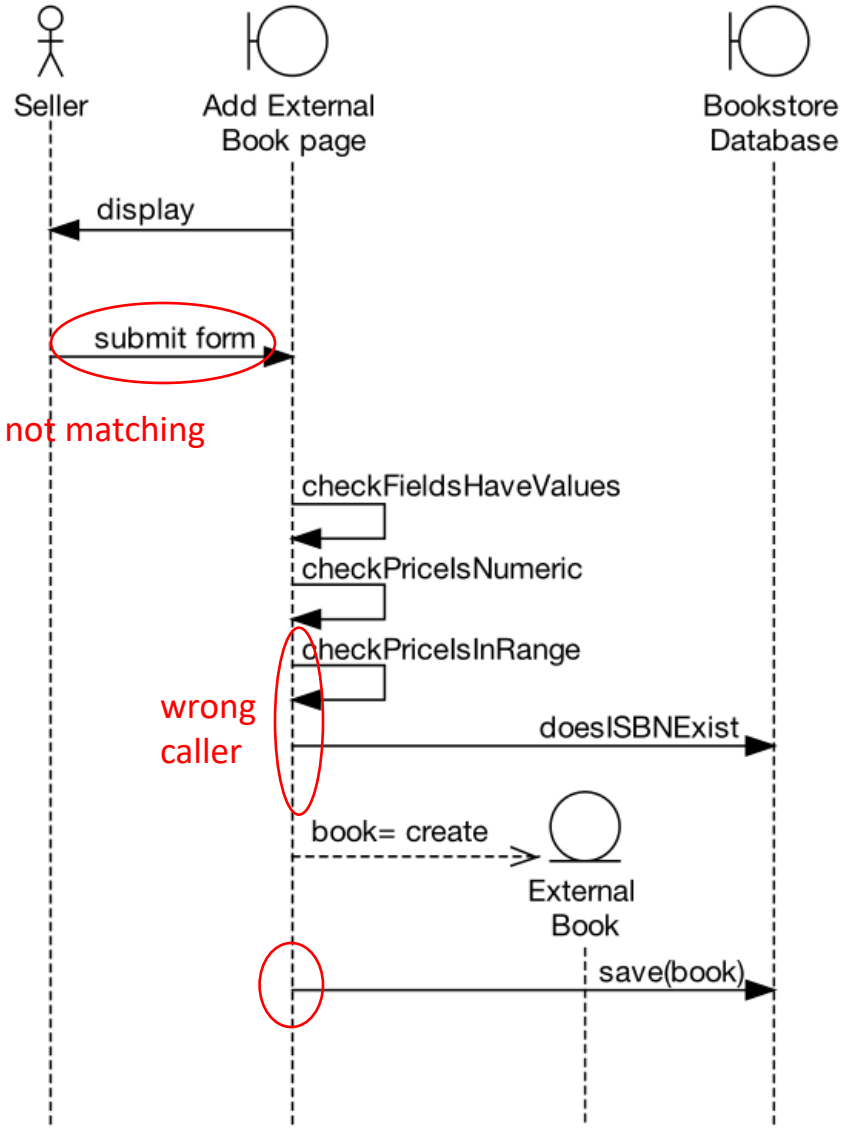
Add External Books to Catalog:

BASIC COURSE:
The system displays the Add External Book page. The Seller types in the External Book details (title, ISBN, price etc) and clicks the Add Book button.

The system checks that each field has a value, and that the price is numeric, isn't a negative value or > $1,000. The system also checks that there's a matching ISBN in the Bookstore database.

The system then creates the External Book in the database.

. . .



Quiz

- Question 1

Add External Books to Catalog:

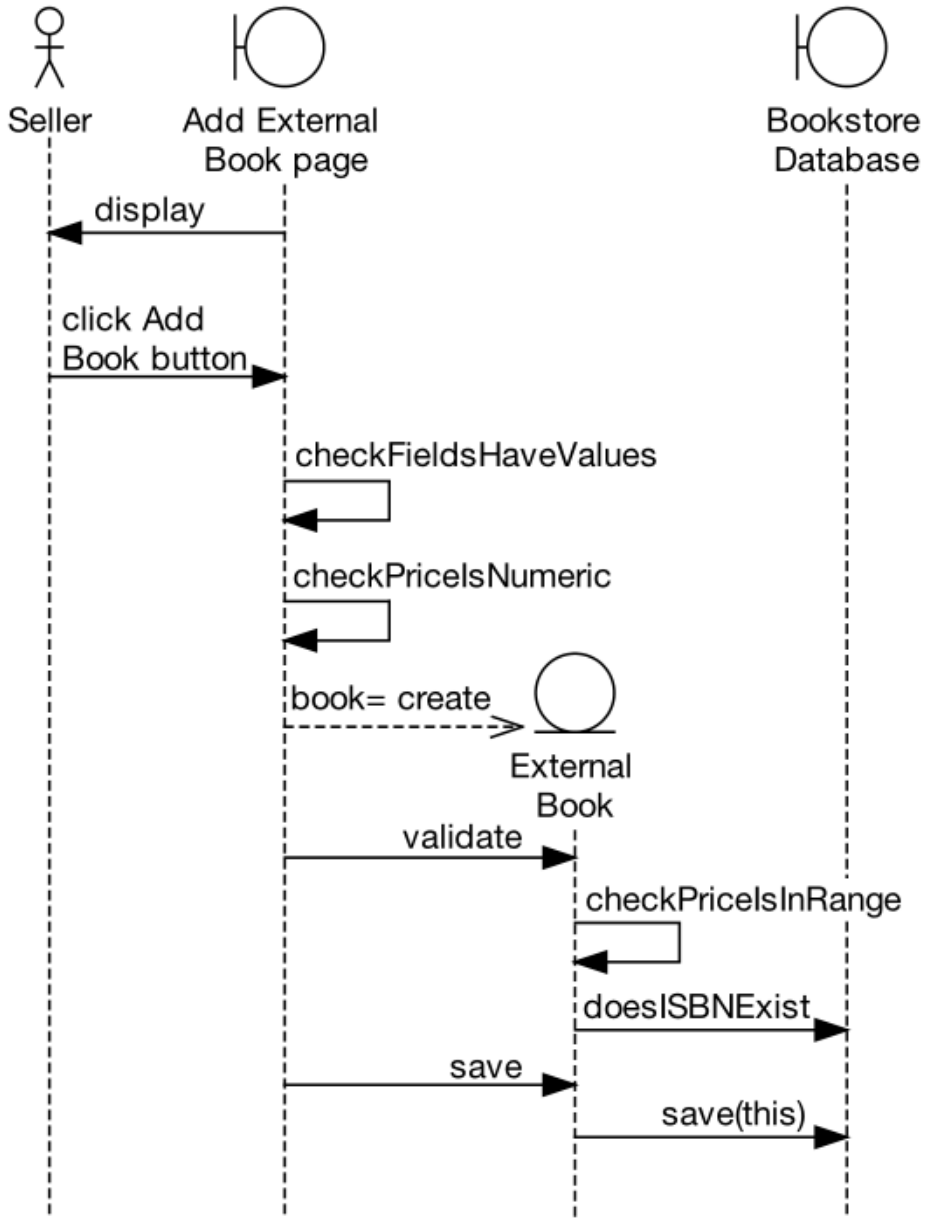
BASIC COURSE:
The system displays the Add External Book page. The Seller types in the External Book details (title, ISBN, price etc) and clicks the Add Book button.

The system checks that each field has a value, and that the price is numeric, isn't a negative value or > $1,000.

The system also checks that there's a matching ISBN in the Bookstore database.

The system then creates the External Book in the database.

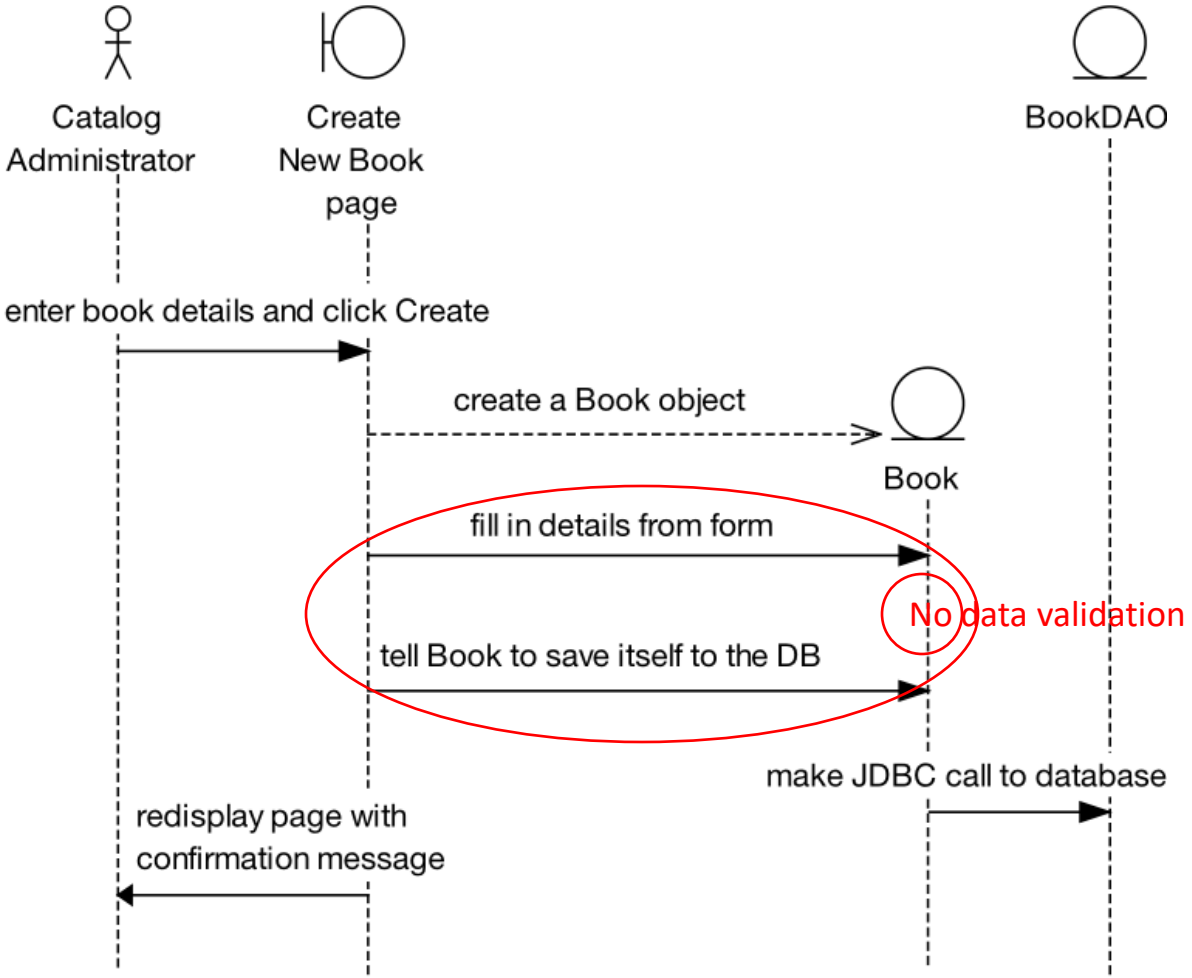
. . .



Quiz

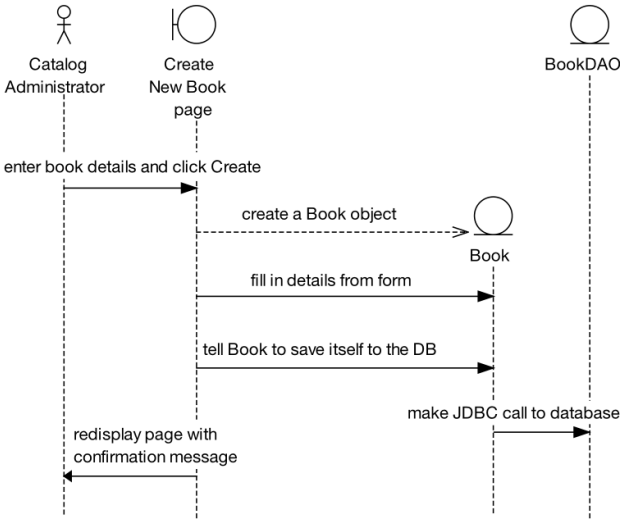
Figure out the errors and then draw the corrected diagram

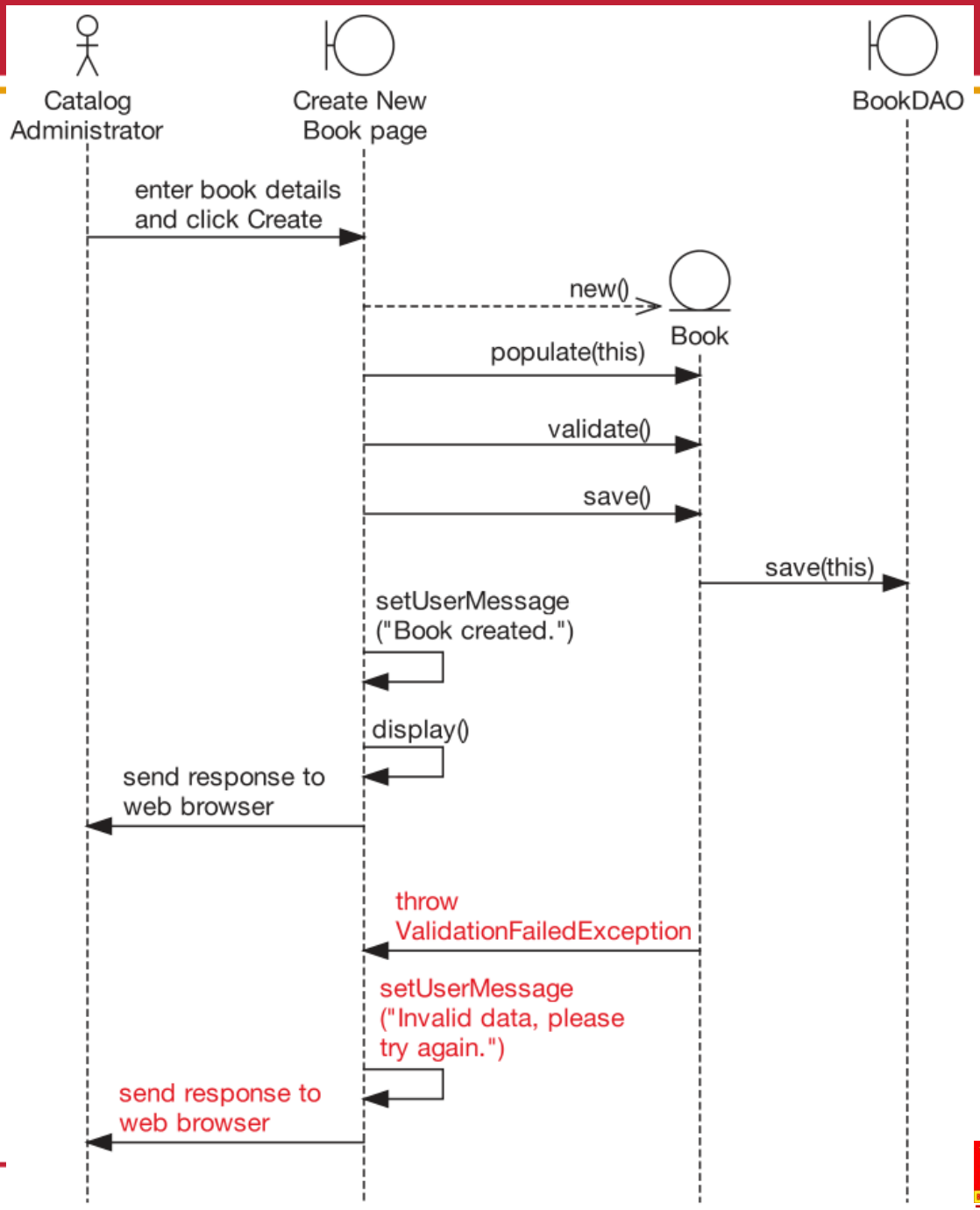
- Question 2: from Create New Book use case



Quiz

- Question 2





Quiz

- Question 3

Edit Shopping Cart use case

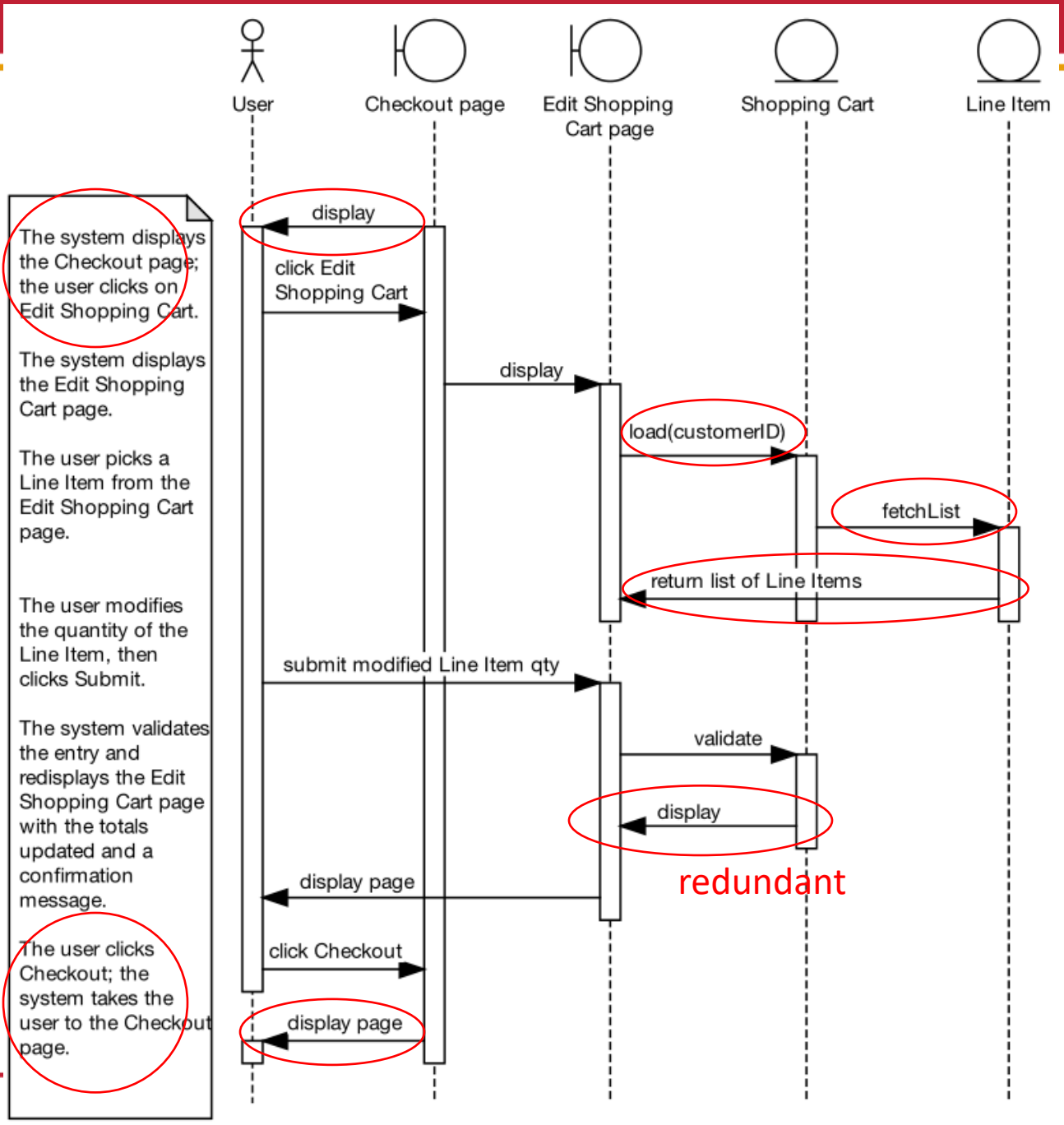
+ Begin/end problem

+ Where is *customerID*?

-> missing *Customer Account* corresponding to this *Shopping Cart, Customer Session* corresponding to this *Customer Account*

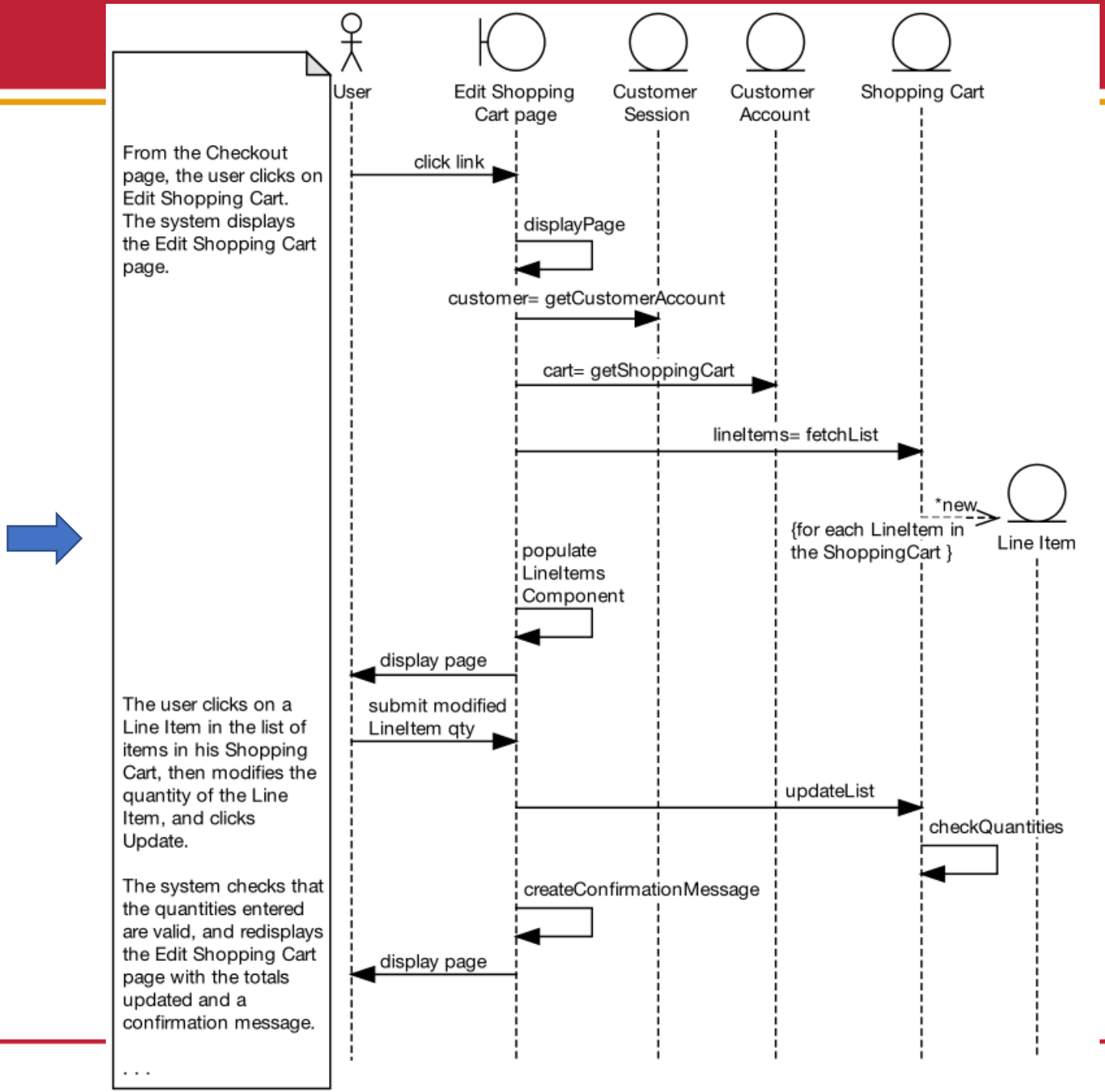
+ Wrong caller

+ Not need



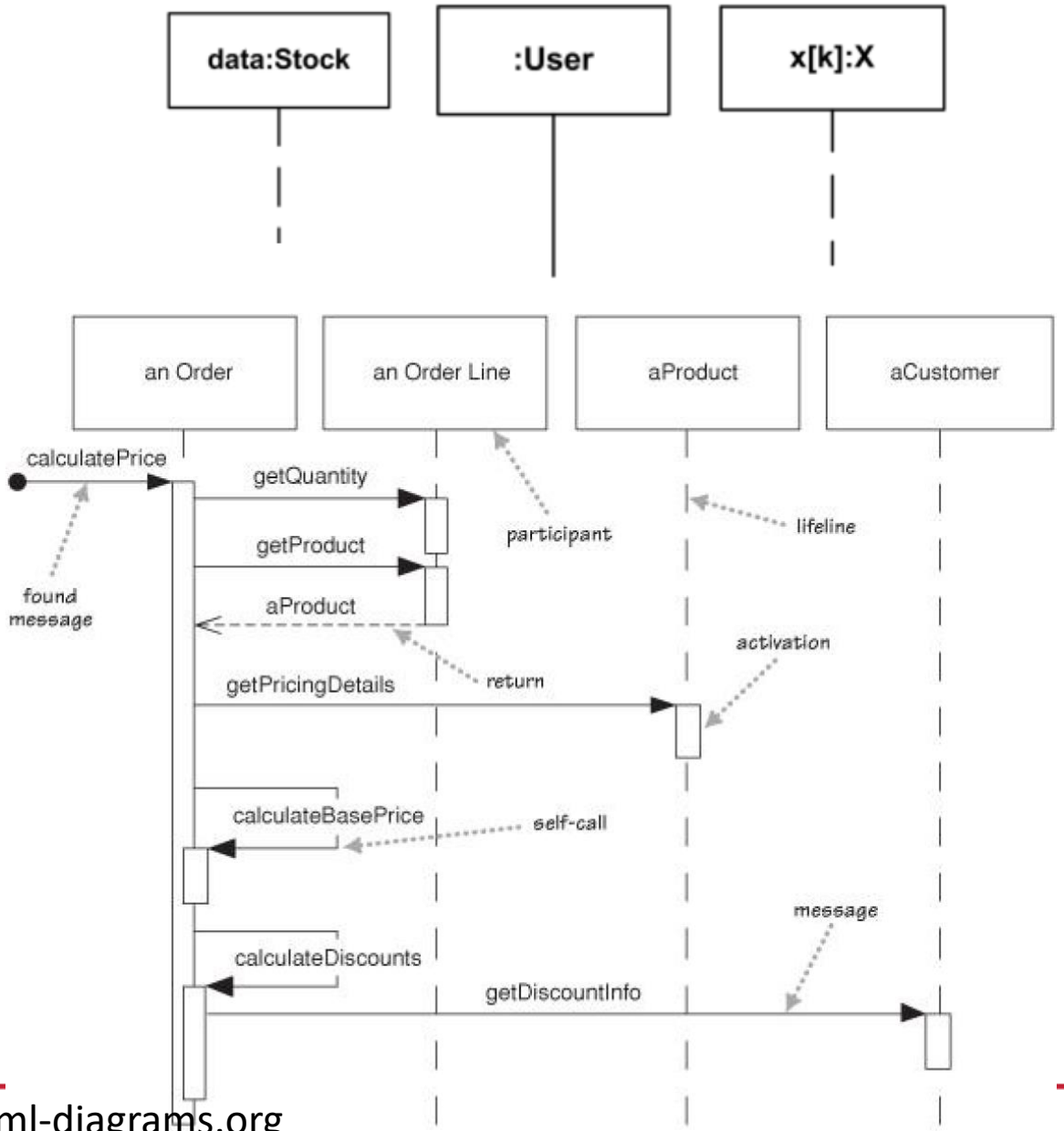
Quiz

- Question 3



Other Sequence Diagrams Reference

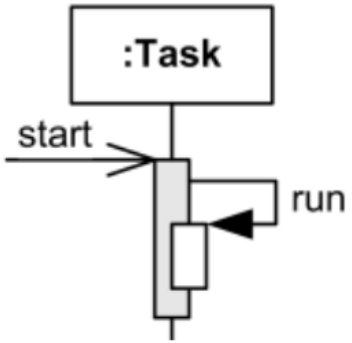
- Object



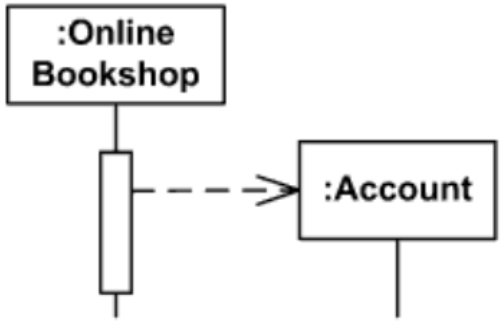
https://www.uml-diagrams.org

Other Sequence Diagrams Reference

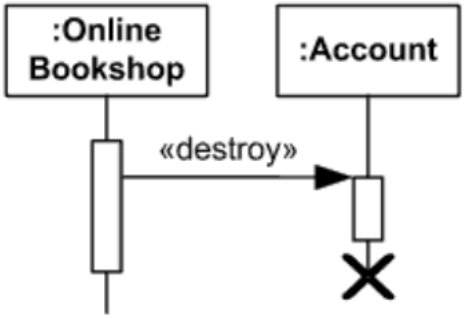
- Overlapping execution



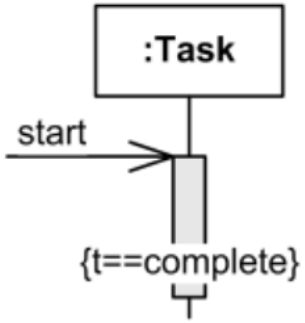
- Create Message

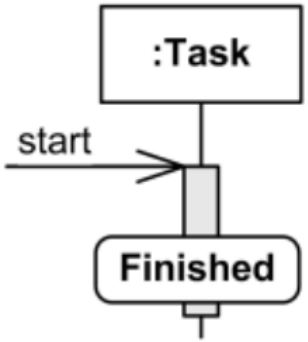


- Delete Message



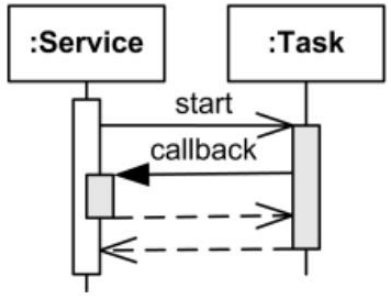
- Runtime constraint

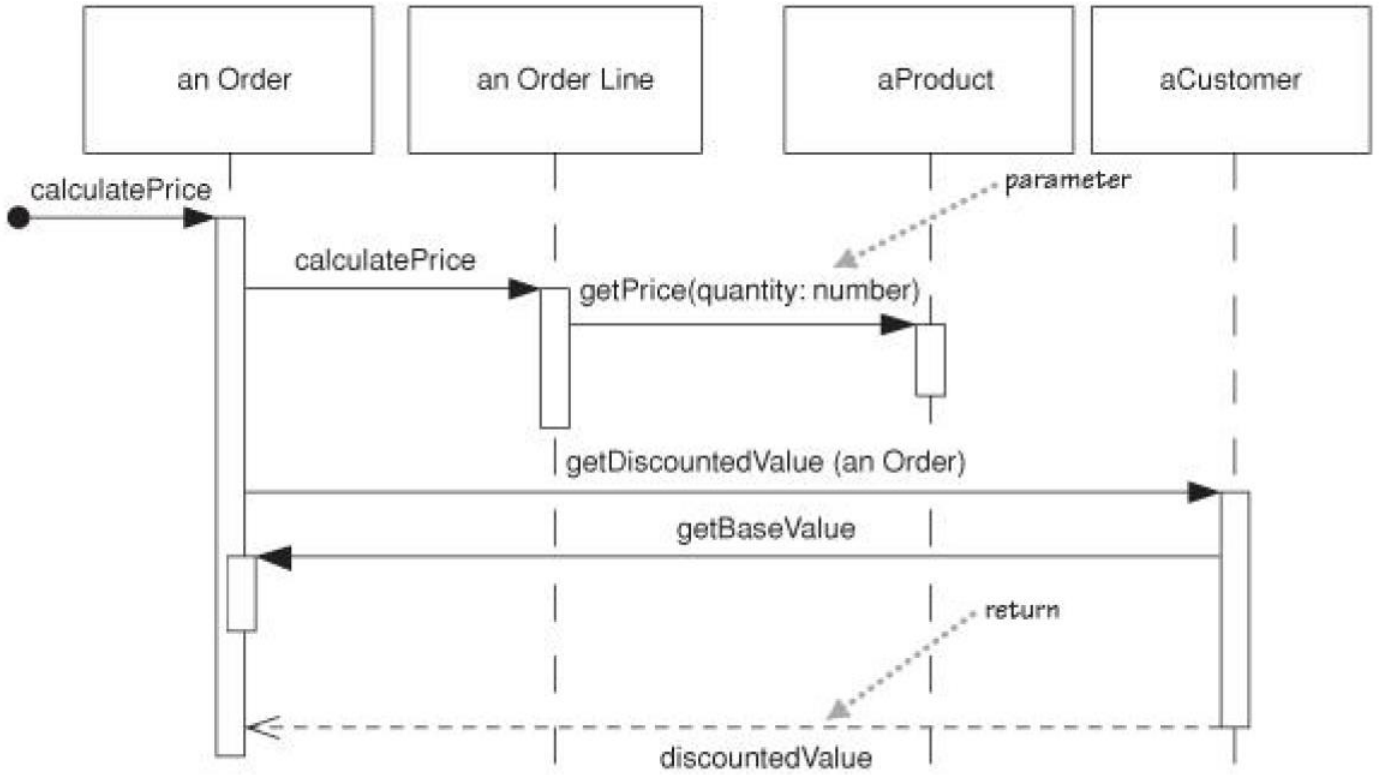




Other Sequence Diagrams Reference

- Callback

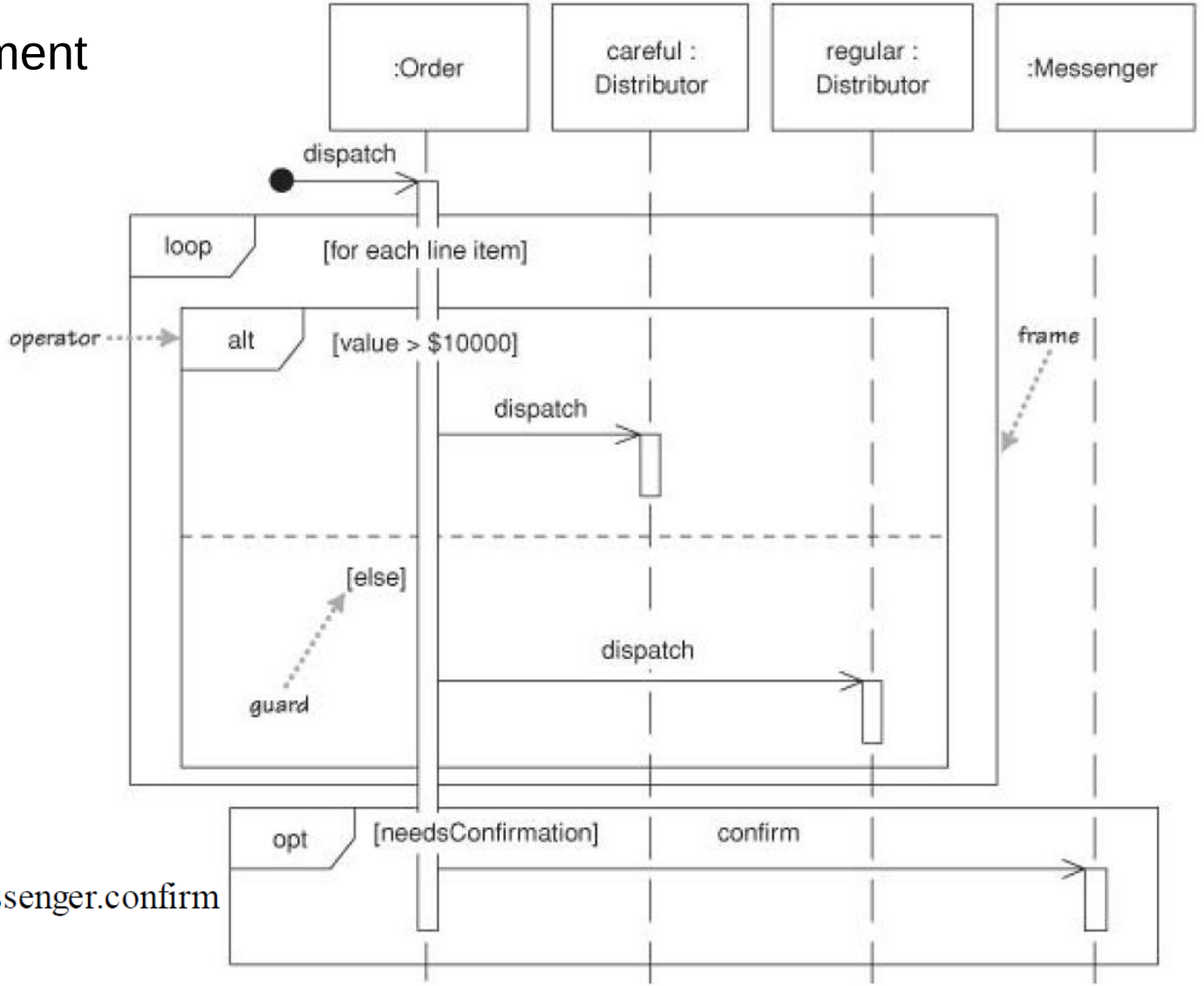




Other Sequence Diagrams Reference

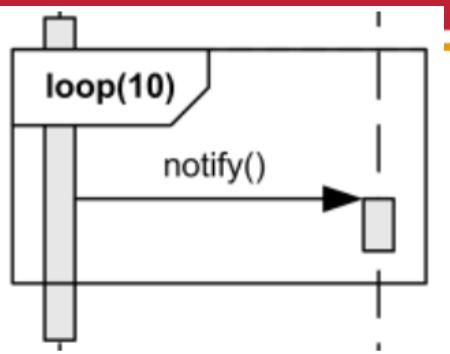
- Combined fragment

```
procedure dispatch
  foreach (lineitem)
    if (product.value> $10K)
      careful.dispatch
    else
      regular.dispatch
    end if
  end for
  if (needsConfirmation) messenger.confirm
end procedure
```

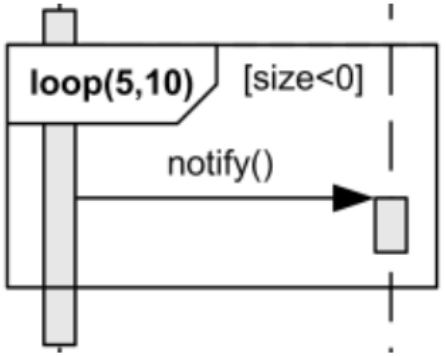


Other Sequence Diagrams Reference

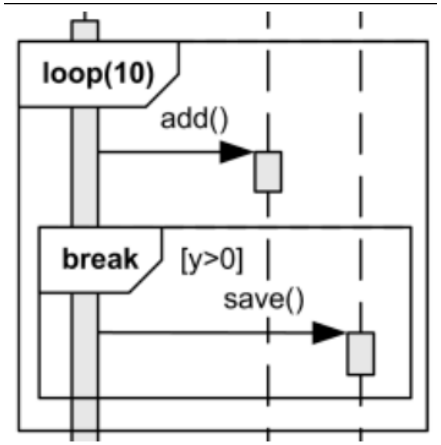
- Loop to execute exactly 10 times.



- Loop to execute from 5 times to 10 times. Stop when the guard is false.

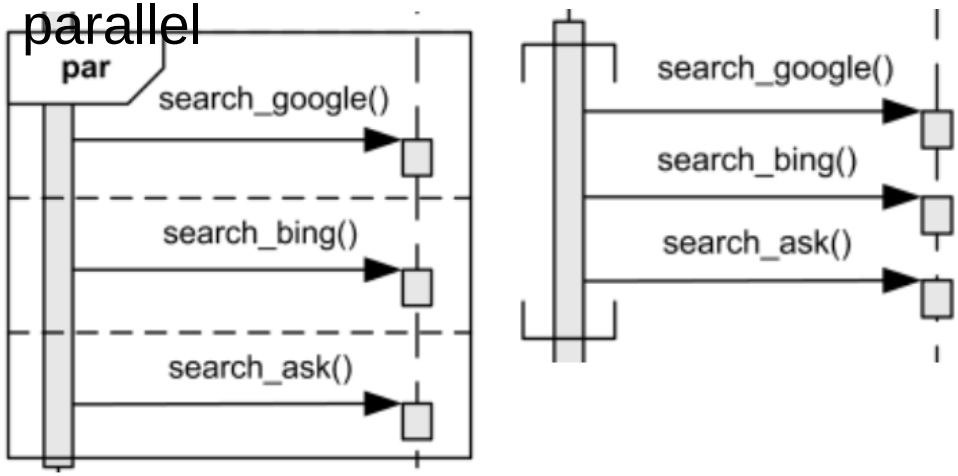


- Loop with break.

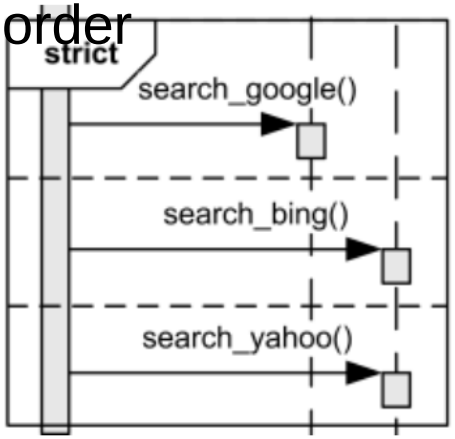


Other Sequence Diagrams Reference

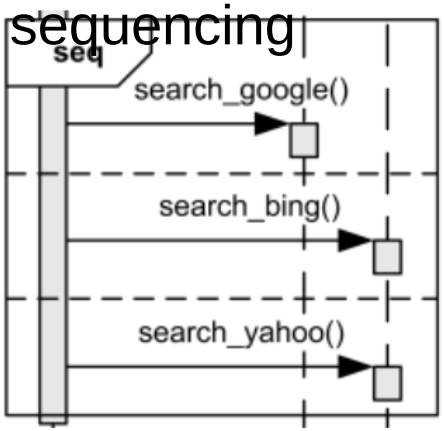
- in any order, possibly parallel



- in the strict sequential order



- in weak sequencing



https://www.uml-diagrams.org

Collaboration/Communication Diagrams

Collaboration/Communication Diagrams

- Communication diagram (called collaboration diagram in UML 1.x)
  - a kind of UML interaction diagram
  - shows interactions between objects using sequenced messages in a free-form arrangement.
- Communication and Sequence Diagrams
  - =View of the dynamic aspects of an OO system
  - =Show HOW a set of objects collaborate to implement a use case or a scenario
  - =Capture semantics of the use-case flow of events;
  - =Identify objects, classes, interactions, and responsibilities

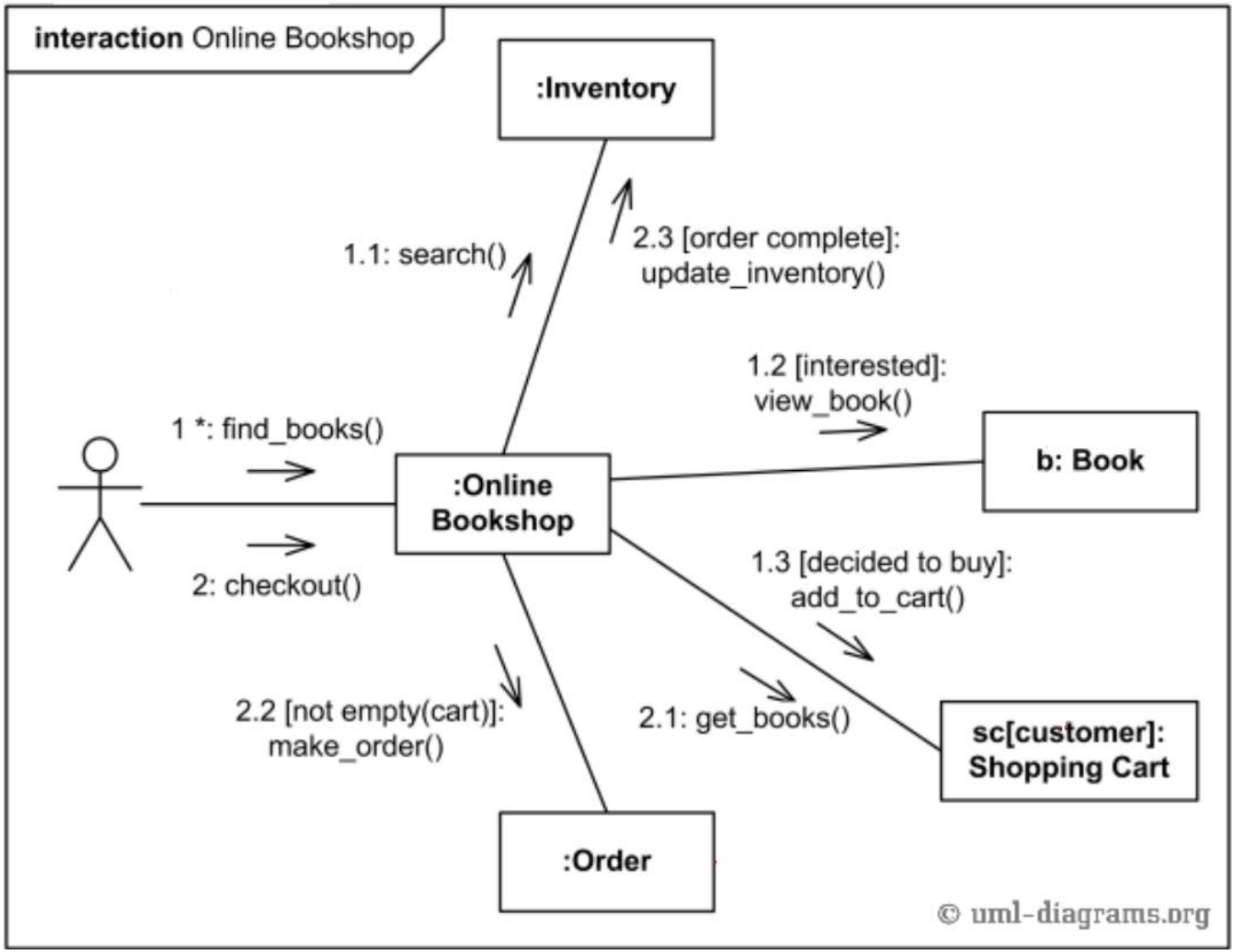
The CD shows HOW different objects depend on each other

Communication Diagrams

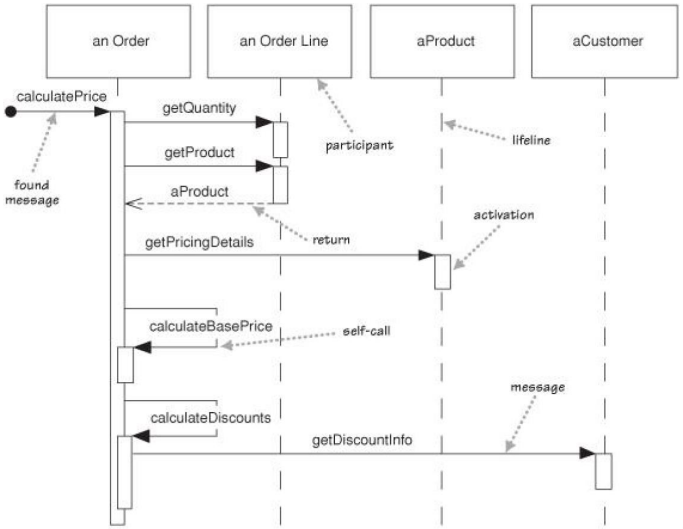
Term and Definition	Symbol
An actor: ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a collaboration by sending and/or receiving messages. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative).	 anActor <<actor>> anActor
An object: ■ Participates in a collaboration by sending and/or receiving messages.	anObject : aClass
An association: ■ Shows an association between actors and/or objects. ■ Is used to send messages.	
A message: ■ Conveys information from one object to another one. ■ Has direction shown using an arrowhead. ■ Has sequence shown by a sequence number.	SeqNumber: aMessage →
A guard condition: ■ Represents a test that must be met for the message to be sent.	SeqNumber: [aGuardCondition]: aMessage →
A frame: ■ Indicates the context of the communication diagram.	Context

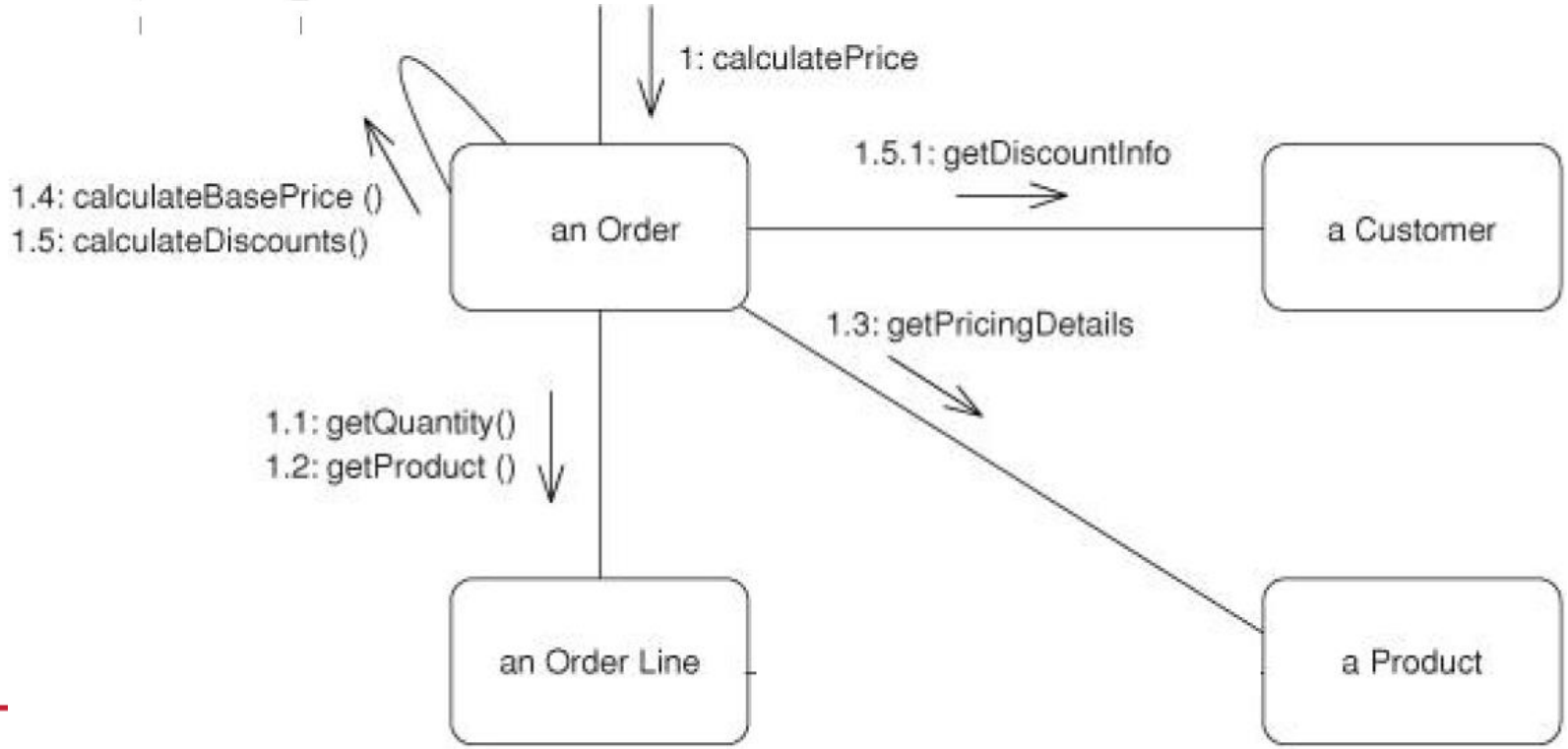
Communication Diagrams

- Example



Communication Diagrams





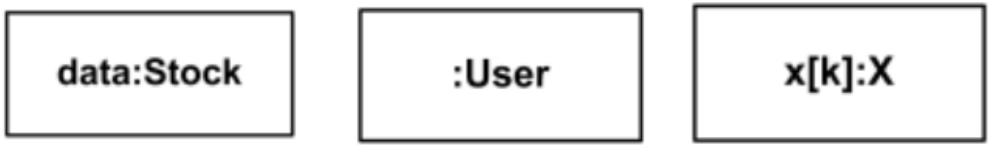
Collaboration Diagrams vs. Sequence Diagrams

- Collaboration Diagrams
  - Emphasize the structural collaboration of a society of objects (the relationships among objects)
  - To understand all the effects on a given object
  - Better for visualizing patterns of collaboration amongst the objects
  - Don’t have any notation for control logic (iteration markers and guards)
- Sequence Diagrams
  - Show the explicit sequence of messages
  - To understand the sequencing of messages in time order
  - not include object relationships
  - Better for visualizing overall flow
  - Have notation for control logic
  - Most people prefer
  - Better for real-time specifications and for complex scenarios

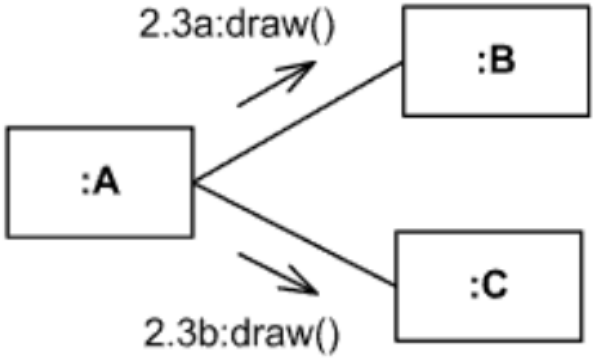
Do not use communication diagrams to model process flow (using activity diagram instead).

Other Communication Diagrams Reference https://www.uml-diagrams.org

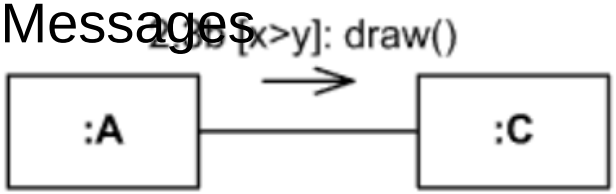
- Object



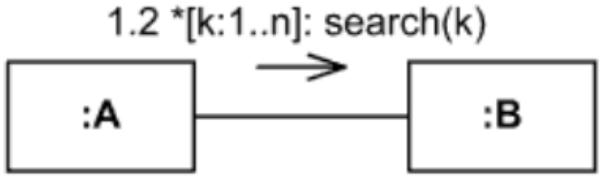
- Concurrent Messages



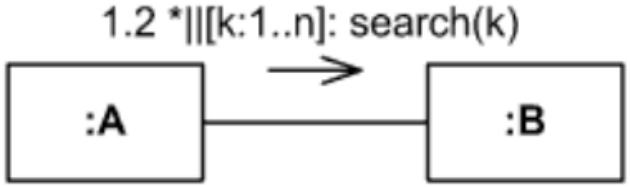
- Conditional Messages



- Sequential Loop



- Concurrent Loop



7.3 State Machine Diagrams

7.3 State Machine Diagrams

- Some of the objects are quite dynamic: can pass through a **variety of states** over their life time
- The dynamic model that shows the different states through which a single object passes during its life in response to events, along with its responses and actions
- Typically not used for all objects
  - Just for complex ones to further define them and to help simplify the design of algorithms for their methods

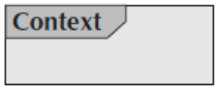
7.3 State Machine Diagrams

- A behavior diagram which shows discrete behavior of a part of designed system through **finite state transitions**.
- Two kinds of state machines
  - **Behavioral state machine**
  - Protocol state machine

- What does it show?
  - Different states of the object
  - What events cause the object to change from one state to another

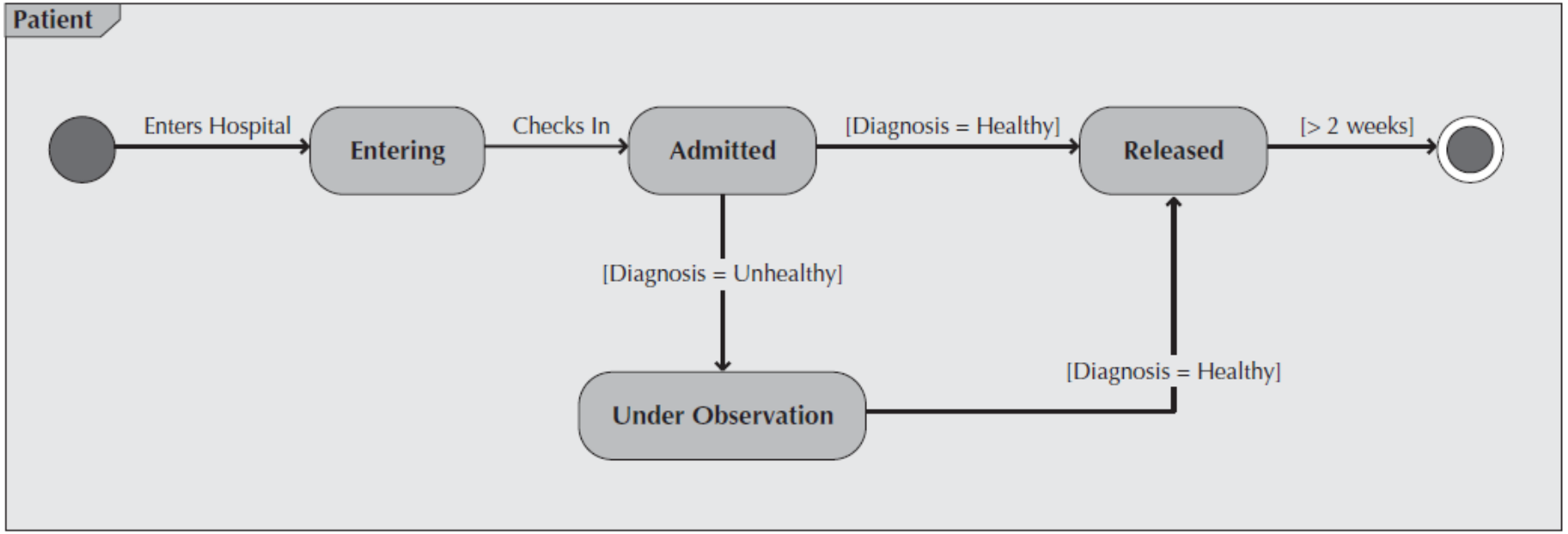
- When ?
  - To help understand the dynamic aspects of a single class
  - How its instances evolve over time

Behavioral State Machine elements

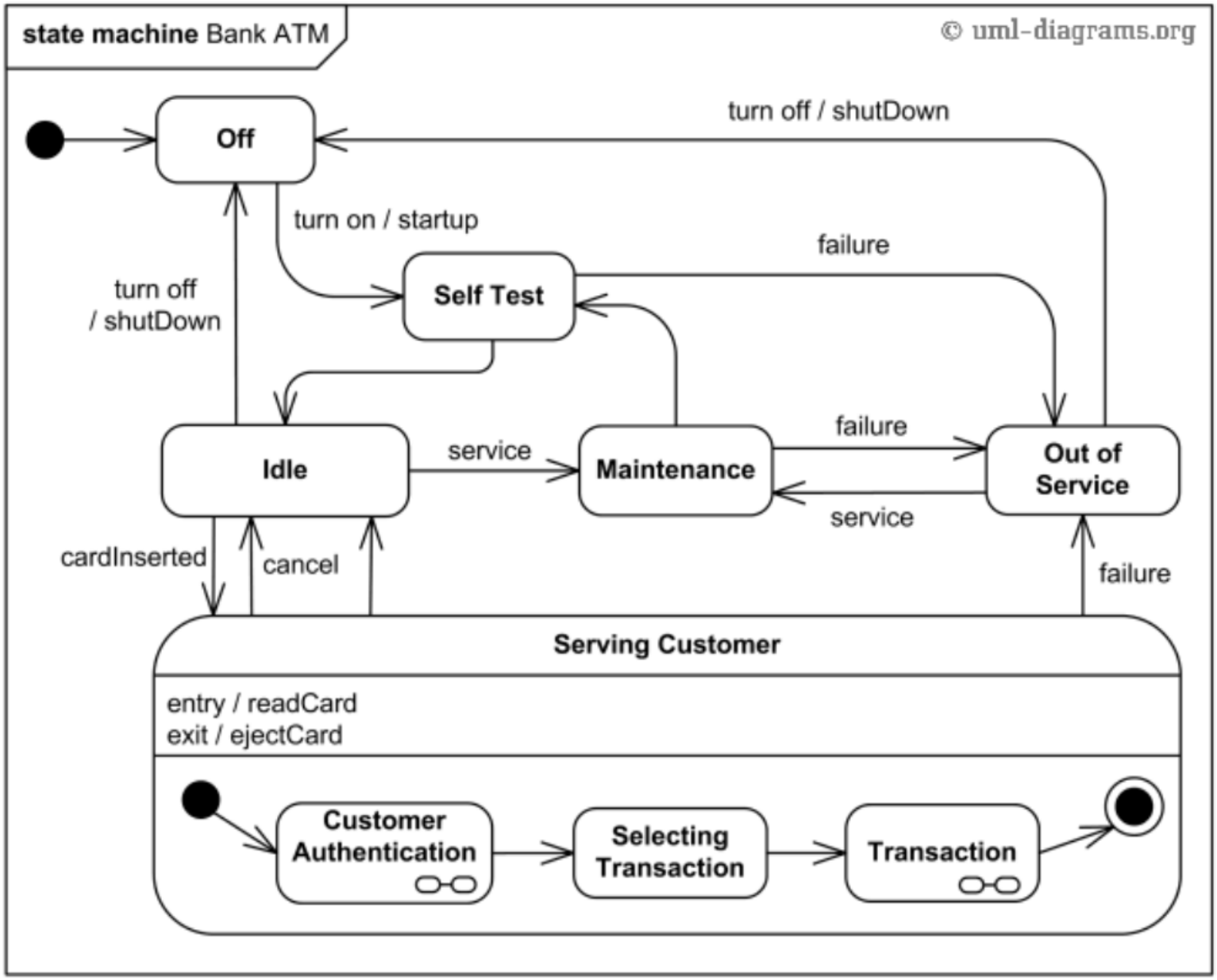
Term and Definition	Symbol
A state: ■ Is shown as a rectangle with rounded corners. ■ Has a name that represents the state of an object.	
An initial state: ■ Is shown as a small, filled-in circle. ■ Represents the point at which an object begins to exist.	
A final state: ■ Is shown as a circle surrounding a small, filled-in circle (bull's-eye). ■ Represents the completion of activity.	
An event: ■ Is a noteworthy occurrence that triggers a change in state. ■ Can be a designated condition becoming true, the receipt of an explicit signal from one object to another, or the passage of a designated period of time. ■ Is used to label a transition.	anEvent
A transition: ■ Indicates that an object in the first state will enter the second state. ■ Is triggered by the occurrence of the event labeling the transition. ■ Is shown as a solid arrow from one state to another, labeled by the event name.	
A frame: ■ Indicates the context of the behavioral state machine.	

Behavioral State Machine

- Behavior is modeled as a traversal of a graph of state nodes connected with transitions.
- Transitions are triggered by events.

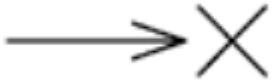


Behavioral State Machine

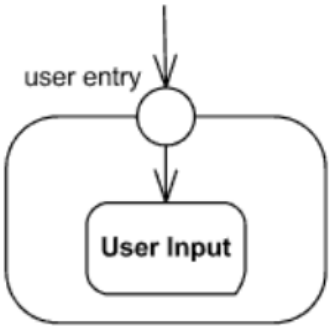


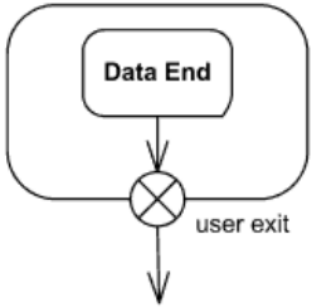
Other State Machine Diagrams Reference

- Transition to terminate pseudostate: invoking a DestroyObjectAction.

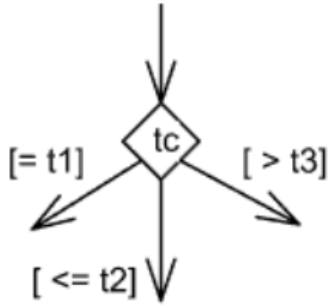


- Entry point, Exit Point



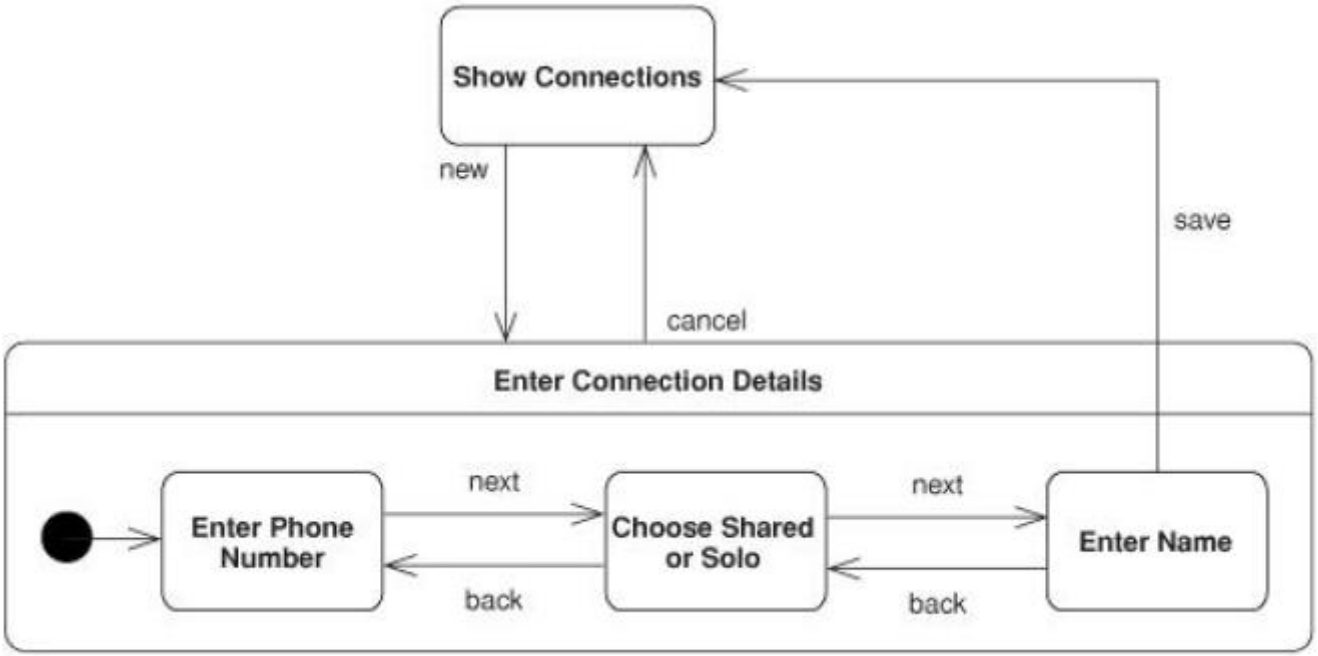


- Choice
  - Choice based on guards applied to the value inside diamond



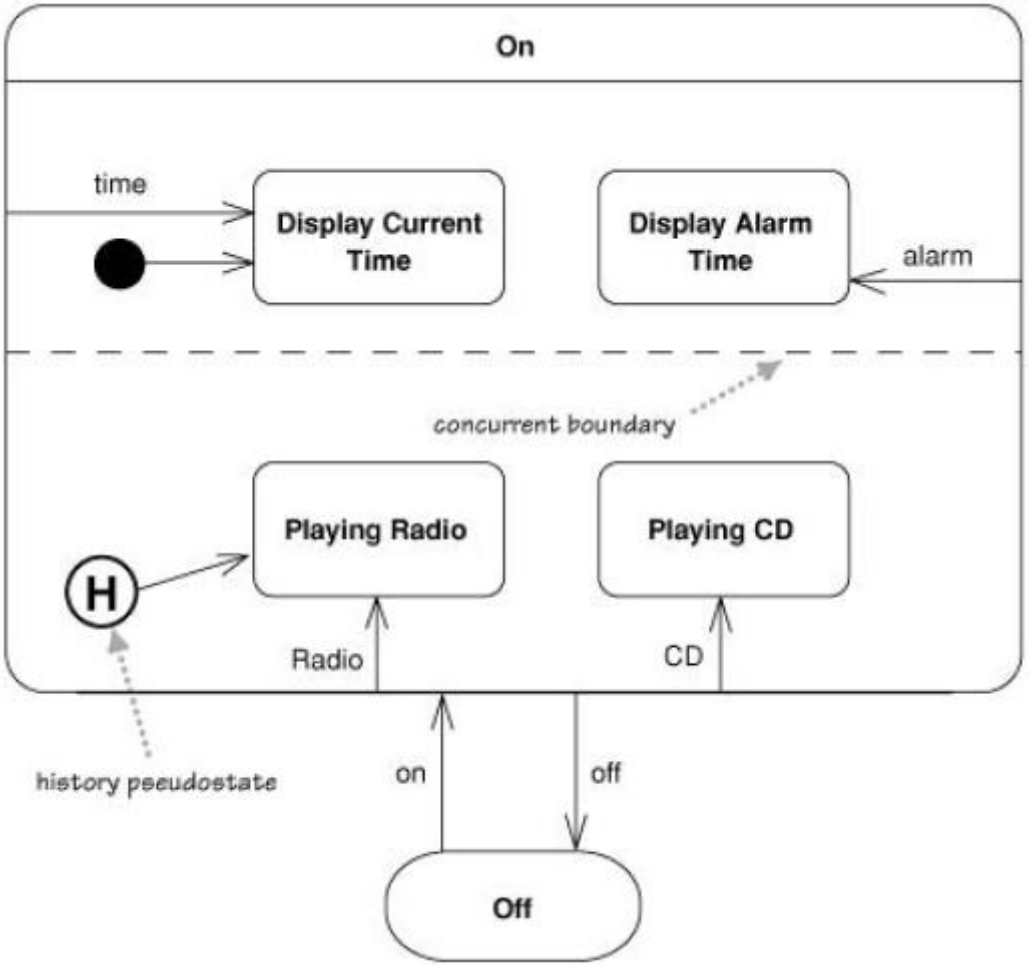
- Fork
  - Fork splits transition into two transitions.
- Join
  - Join merges transitions into single transition.

Other State Machine Diagrams Reference



Superstate with nested substates

Other State Machine Diagrams Reference



Guidelines for Creating Behavioral State Machines

1. Create a behavioral state machine for objects whose behavior changes based on the state of the object.
2. To adhere to the left -to-right and top-to-bottom reading conventions, the initial state should be drawn in the top left corner of the diagram and the final state should be drawn in the bottom right of the diagram.
3. Names of the states are simple, intuitively obvious, and descriptive.
4. Question black hole and miracle states. Black hole states, states that an object goes into and never comes out of, most likely are actually final states. Miracle states, states that an object comes out of but never went into, most likely are initial states.
5. Be sure that all guard conditions are mutually exclusive.
6. All transitions should be associated with a message and operation.

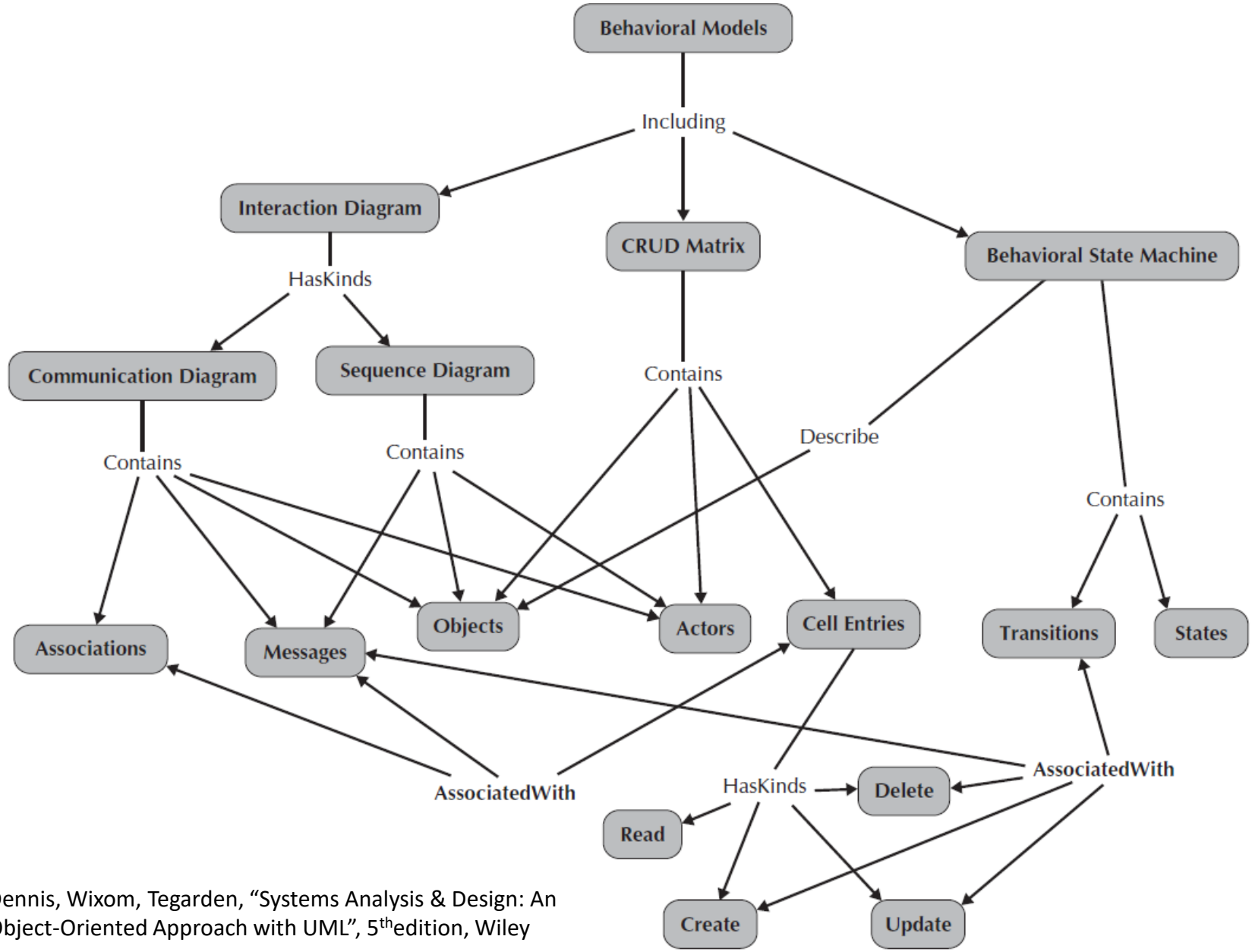
When to Use State Diagrams

- ☺ describing the behavior of an object across several use cases
- ☹ describing behavior that involves a number of objects collaborating
- => Combine state diagrams with other techniques
- => If you do use state diagrams, **don’t** try to draw them for every class in the system. Use state diagrams **only** for those classes that exhibit interesting behavior

Verifying and Validating The Behavioral Model

1. Every actor and object included on a sequence diagram must be included as an actor and an object on a communication diagram, and vice versa.
2. If there is a message on the sequence diagram, there must be an association on the communications diagram, and vice versa.
3. Every message that is included on a sequence diagram must appear as a message on an association in the corresponding communication diagram, and vice versa.
4. If a guard condition appears on a message in the sequence diagram, there must be an equivalent guard condition on the corresponding communication diagram, and vice versa.
5. The sequence number included as part of a message label in a communications diagram implies the sequential order in which the message will be sent.
6. Many representation-specific rules have been proposed

Verifying and Validating The Behavioral Model



Dennis, Wixom, Tegarden, "Systems Analysis & Design: An Object-Oriented Approach with UML", 5th edition, Wiley

Exercise

- Draw the sequence diagram, communication diagram for your project
- Select and draw one State Machine Diagram for your project

- Can be extended
- Tie the design closely with the implementation details of the target framework before begin coding

Spring Framework