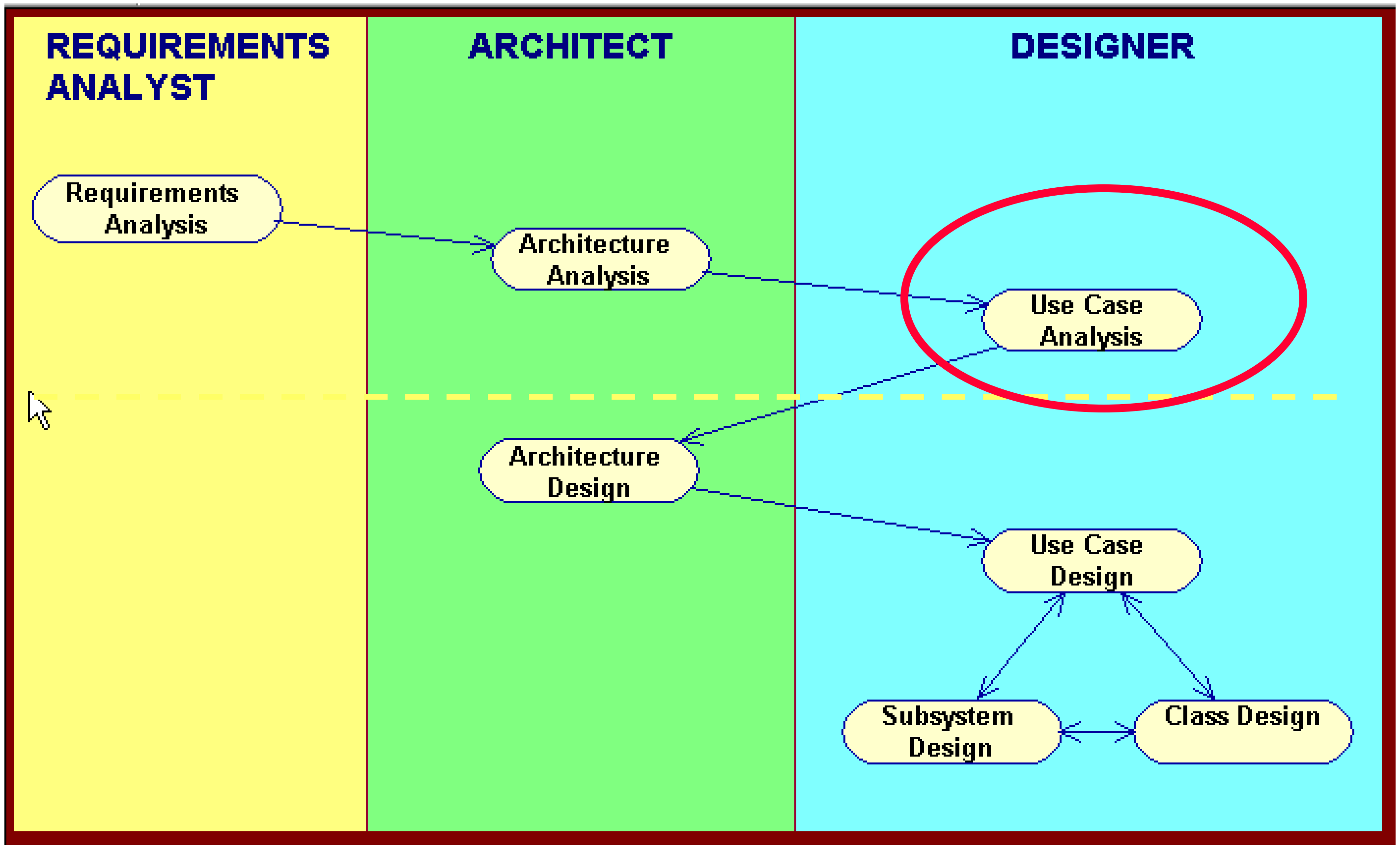
Use Case Analysis

Basic RUP OOAD Activities

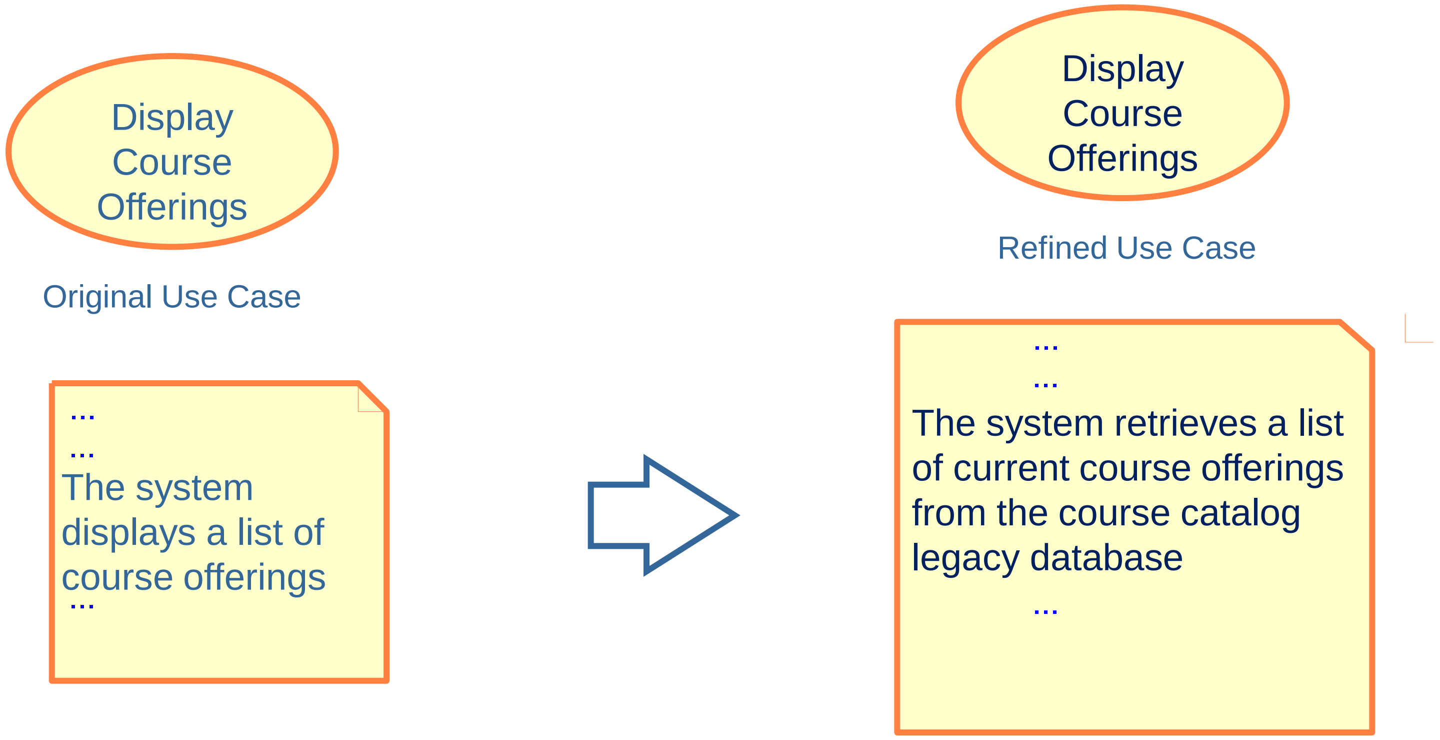


Use-Case Analysis Steps

- Refine the use-case description
- Model behavior using a sequence diagram
  - Identify analysis classes for the use-case
  - Model object collaborations
- Model structure in VOPC diagram
  - Capture responsibilities from sequence diagram
  - Add analysis-level attributes and associations
  - Note analysis mechanisms
- Integrate analysis classes
- Document business rules

J203-6, Q73-4

Refine Use-Case Description



...
...
The system displays a list of course offerings
...

...
...
The system retrieves a list of current course offerings from the course catalog legacy database
...

Use-Case Analysis Steps

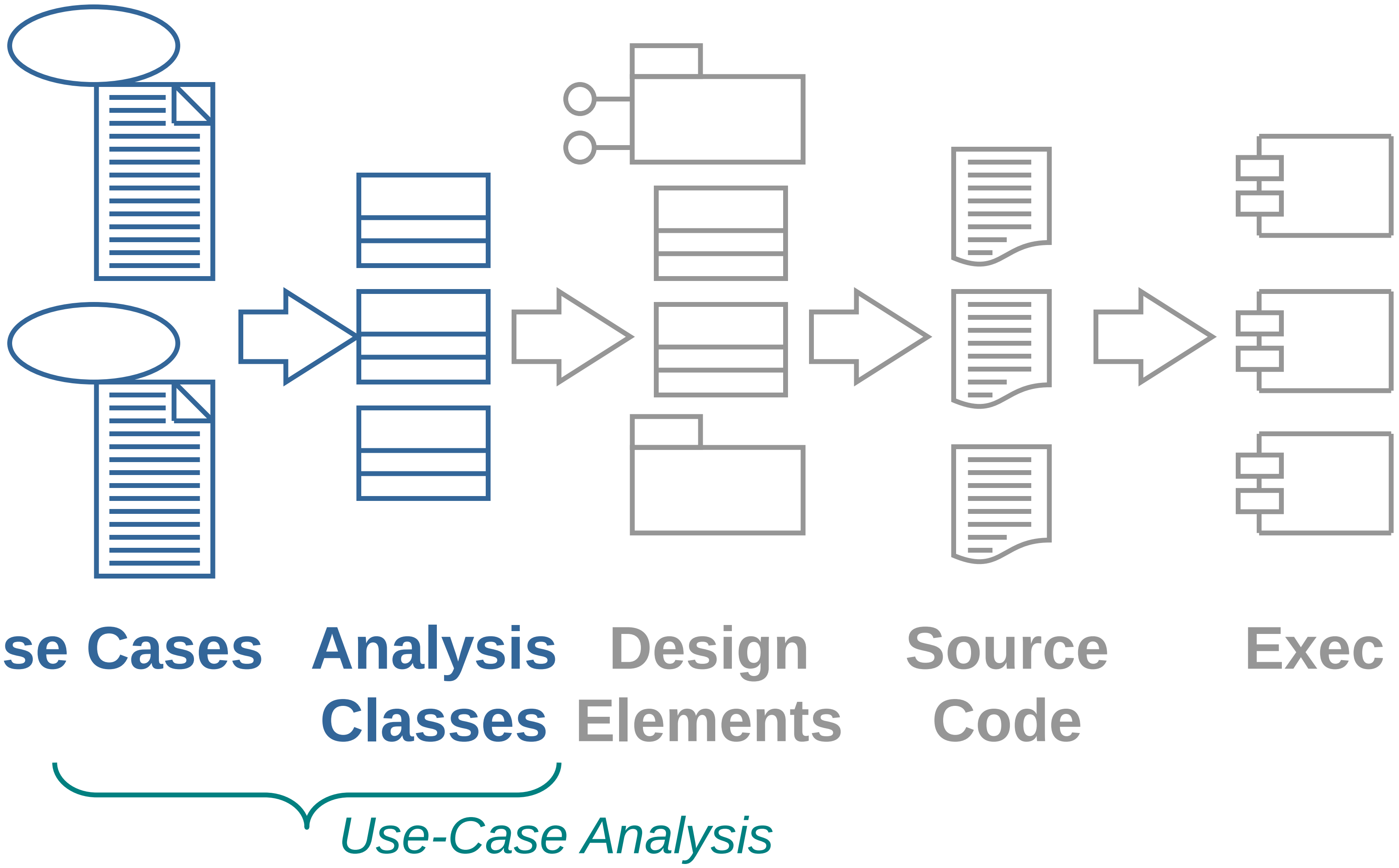
- Refine the use-case description
- Model behavior using a sequence diagram
  - Identify analysis classes for the use-case
  - Model object collaborations
- Model structure in VOPC diagram
  - Capture responsibilities from sequence diagram
  - Add analysis-level attributes and associations
  - Note analysis mechanisms
- Integrate analysis classes
- Document business rules

Analysis Classes

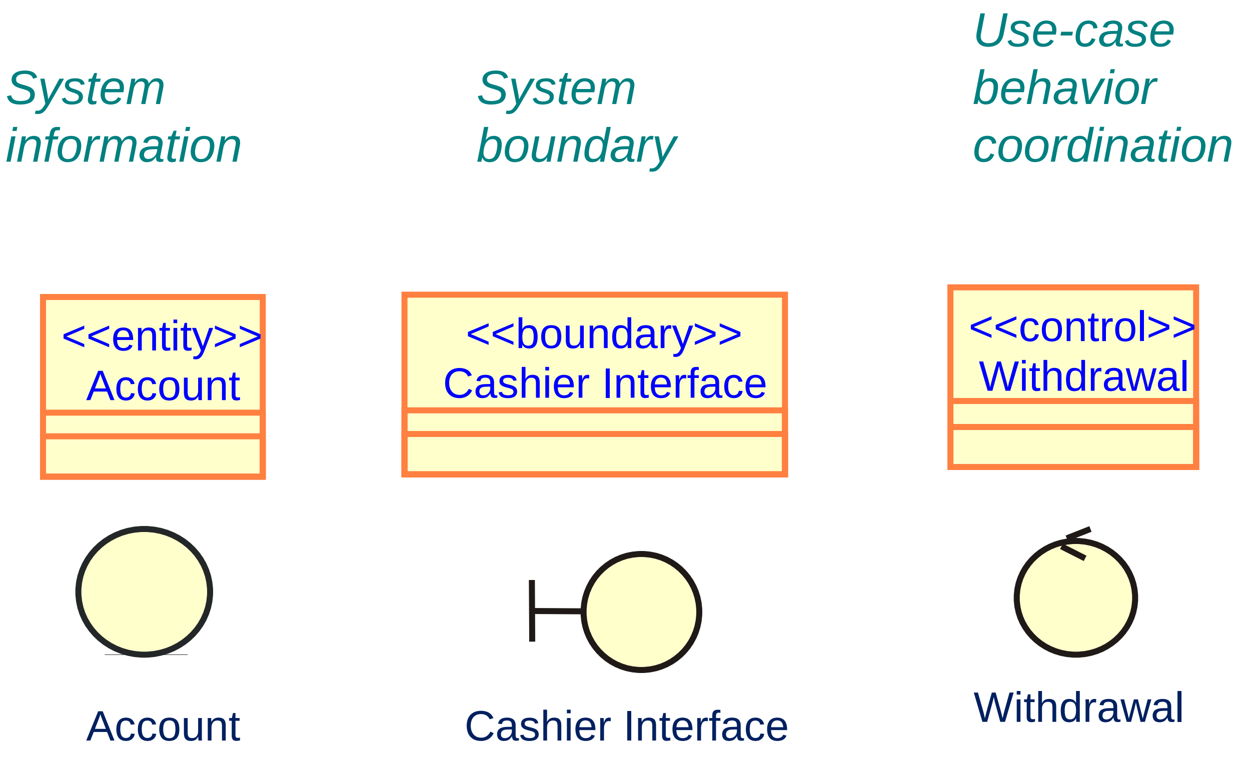
- Represent high level abstractions of one or more classes and/or subsystems
- Emphasize functional requirements
- Capture behavior in terms of conceptual responsibilities
- Model attributes at high level
- Use 3 “stereotypes”: entity, boundary, control

Q57-9, J181-5, J204-5

Analysis Classes: A First Step Towards Executables



3 Analysis Class “Stereotypes”

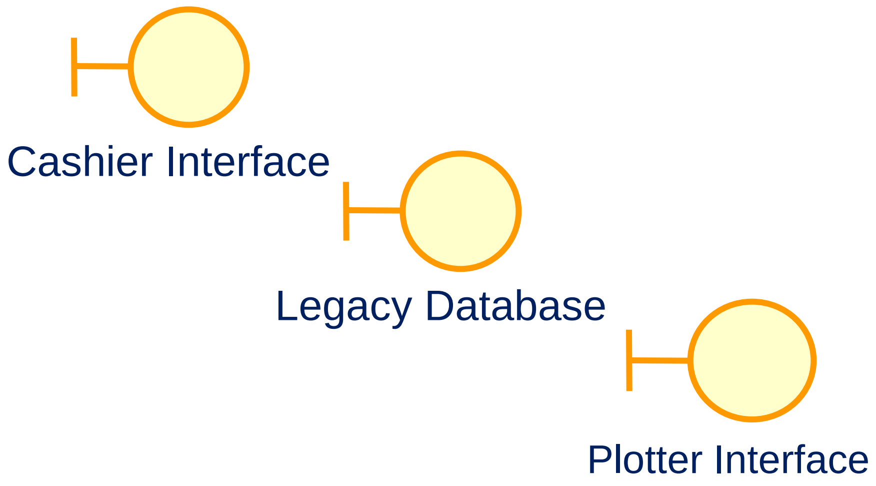


1. **Analysis classes -- ‘things in the system which have responsibilities and behavior’**
2. **Analysis classes are used to capture a ‘first-draft’, rough-cut of the object model of the system**
3. **Analysis classes model objects from the problem domain.**
4. **Analysis classes can be used to represent "the objects we want the system to support"**

J183

Boundary Classes

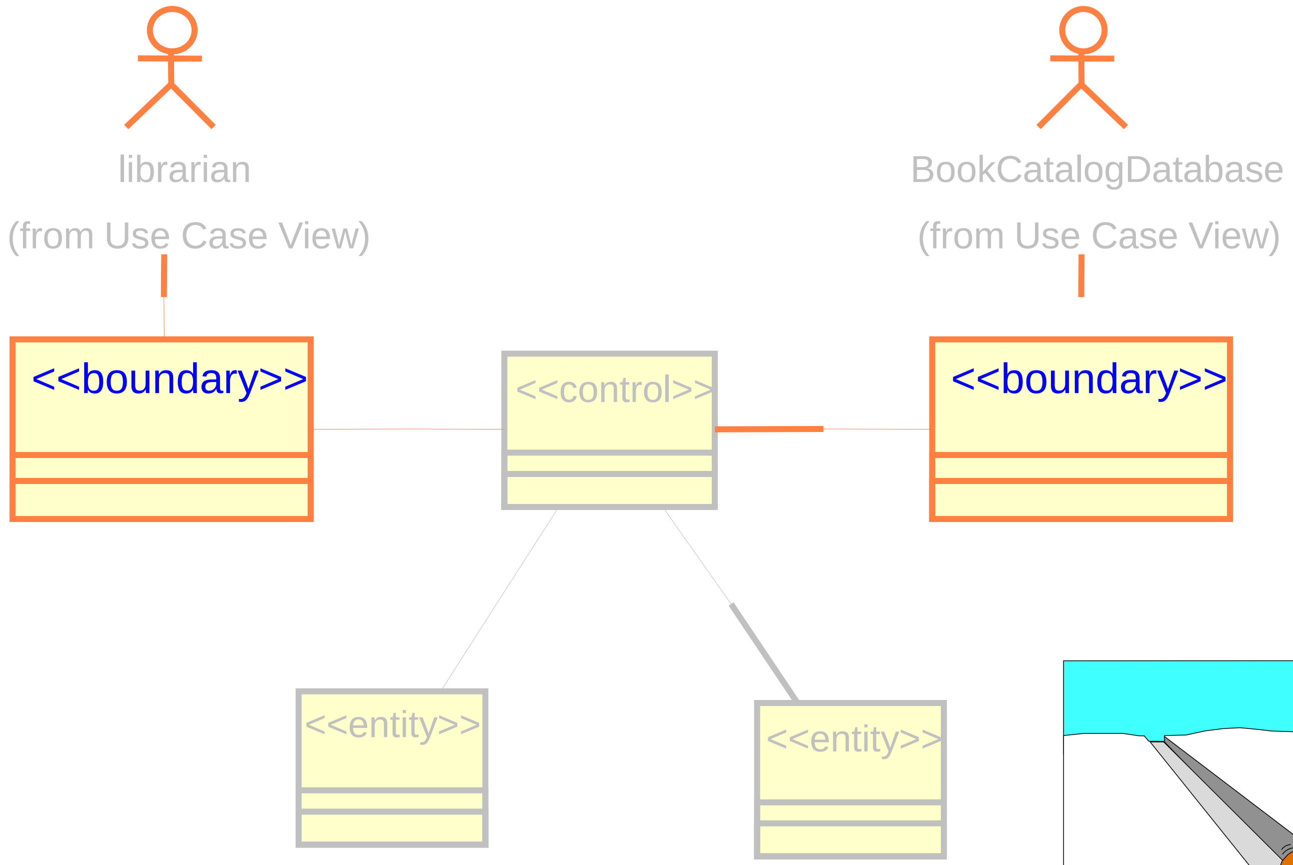
- Capture interaction between system and actors (external systems)
- Help isolate changes to external systems
- Typical examples
  - User interfaces
  - System interfaces
  - Device interfaces

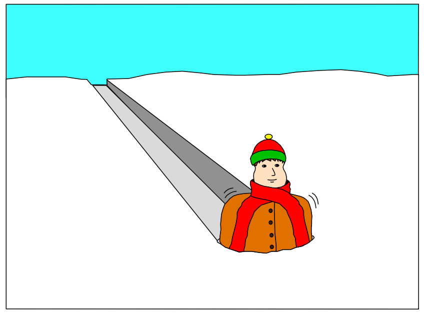


- *Guideline: One boundary class per actor/use-case pair*

J183-4

Boundary Classes Isolate System/Actor Interactions

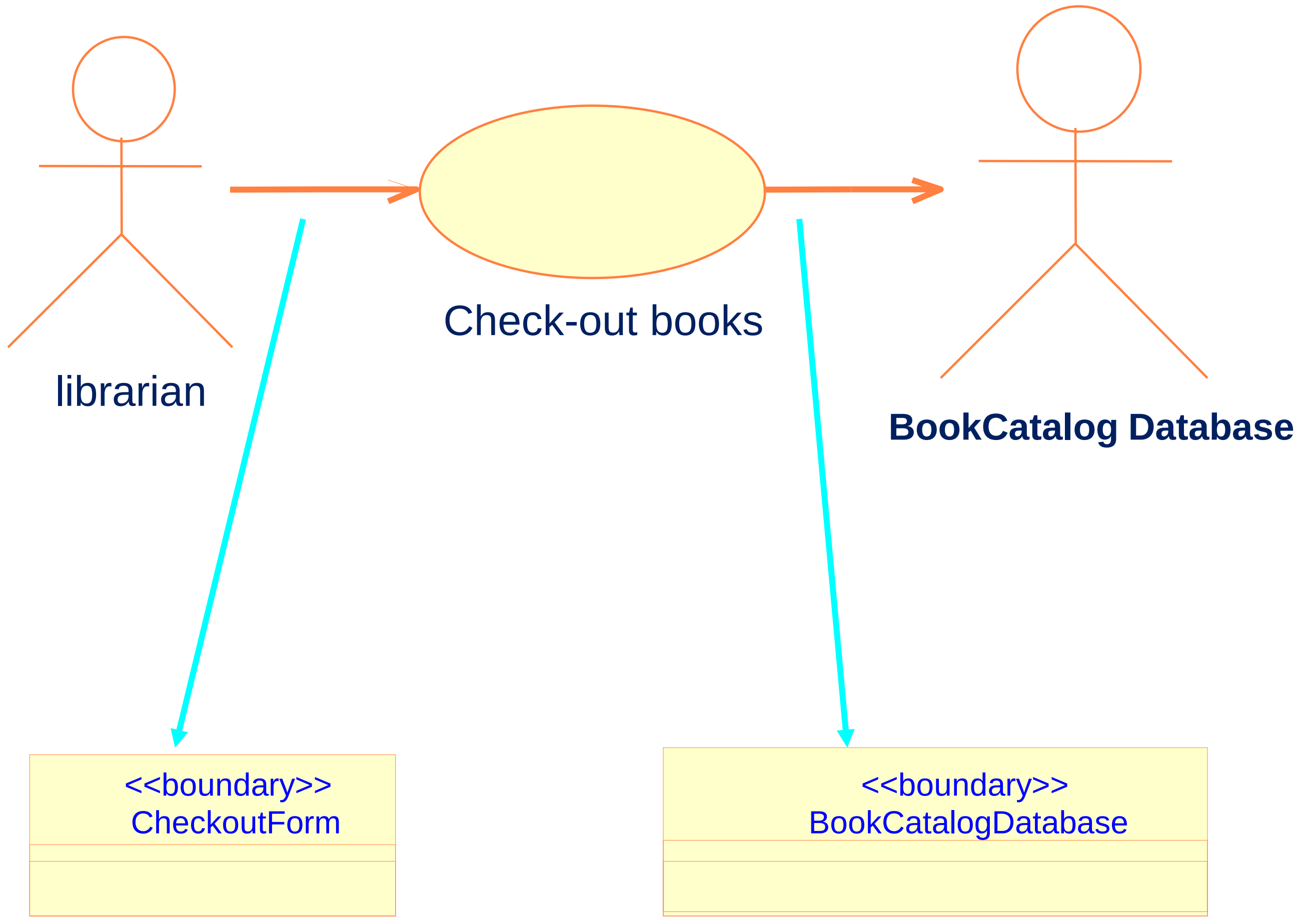




J183-4

Identify Boundary Classes

- For each actor/use case pair



J183, J204-5

Boundary Class Guidelines

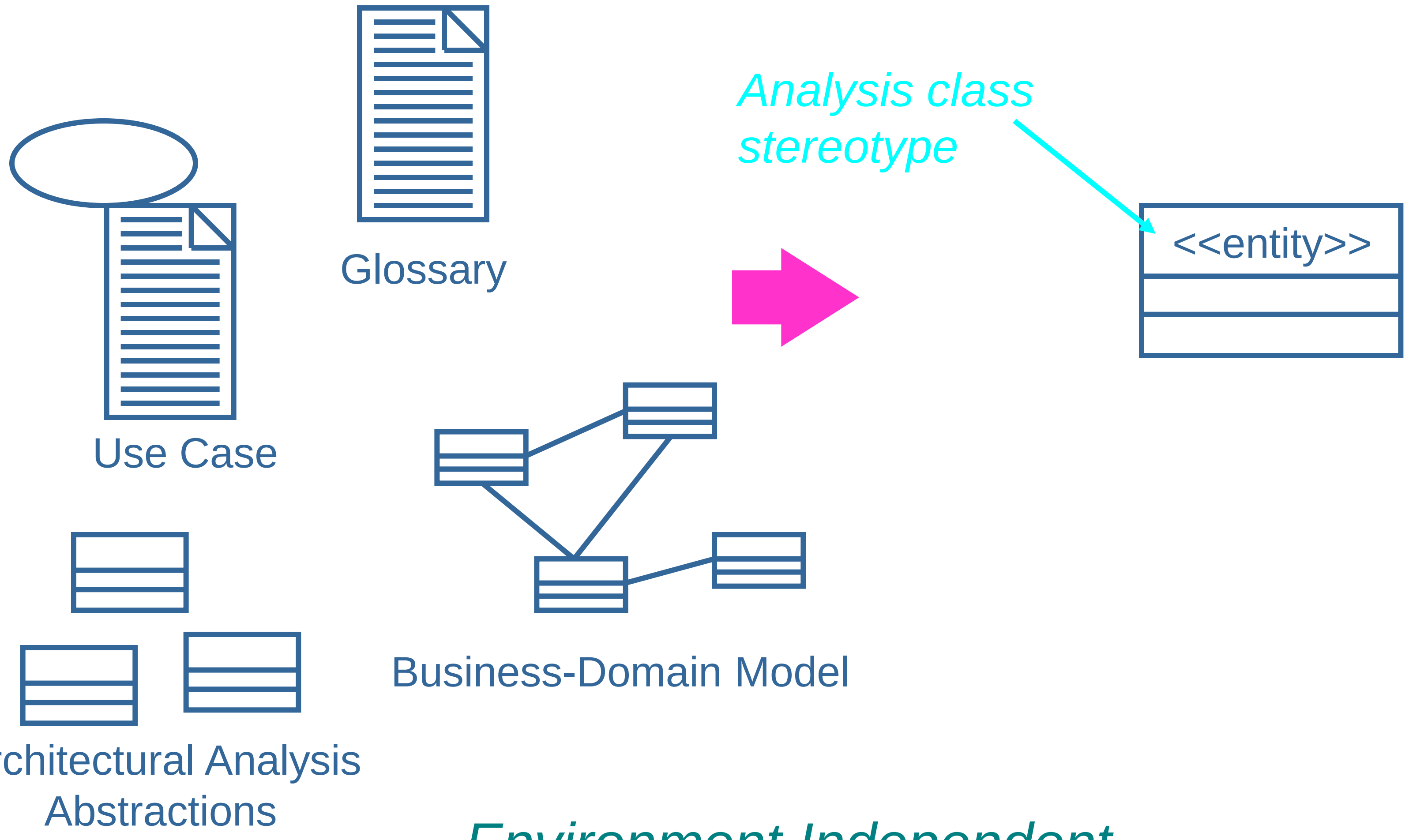
- User Interface
  - Focus on what information is presented
  - UI details get worked out in design and implementation
- System and Device Interface
  - Focus on <u>what</u> is needed to facilitate communication with external systems
  - <u>How</u> to implement is worked out later

Describe "what" is to be achieved, not "how" to achieve it

J183

What is an Entity Class?

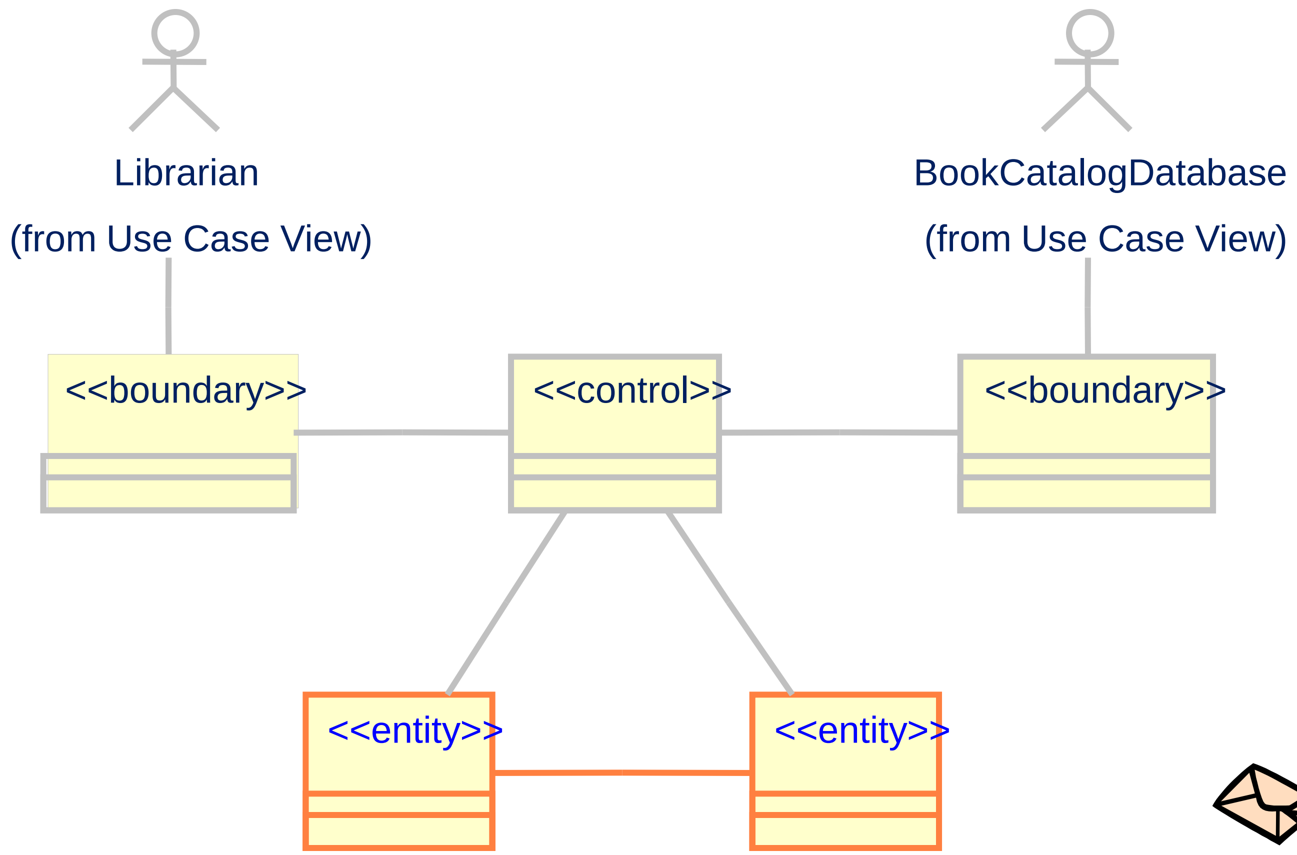
- Key abstractions of the system

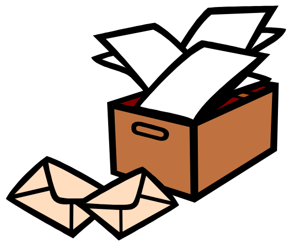

Glossary
Use Case
Analysis class stereotype
<<entity>>
Business-Domain Model
Architectural Analysis Abstractions

Environment Independent

J184-5, Q57

Entity Classes Model Persistent Information



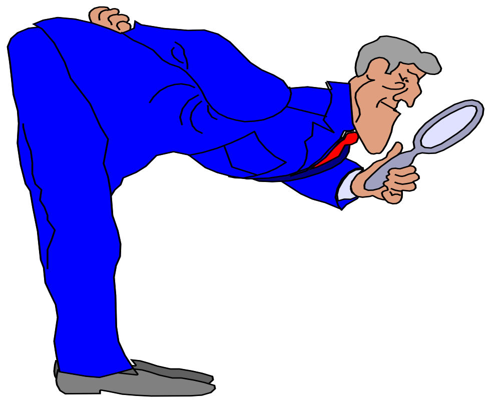


- Responsibilities: to store and manage information in the system
- Hold and update information about events or a real-life object
- Usually persistent

J184-5, Q57

Finding Entity Classes by Filtering Use-Case Nouns

- Start with key abstractions
- Noun clauses
  - Ignore redundant or vague candidates
  - Ignore actors (out of scope)
  - Ignore implementation constructs
  - Look for things acted on by business rules



J184-5, J204, Q57

Library System Problem Statement

You have been hired by Prince University to update their library record keeping. Currently the library has an electronic card catalog that contains information such as author, title, publisher, description, and location of all of the books in the library. All the library member information and book check-in and checkout information, however, is still kept on paper. This system was previously workable, because Prince University had only a few hundred students enrolled. Due to the increasing enrollment, the library now needs to automate the check-in/checkout system.

The new system will have a windows-based desktop interface to allow librarians to check-in and checkout books.

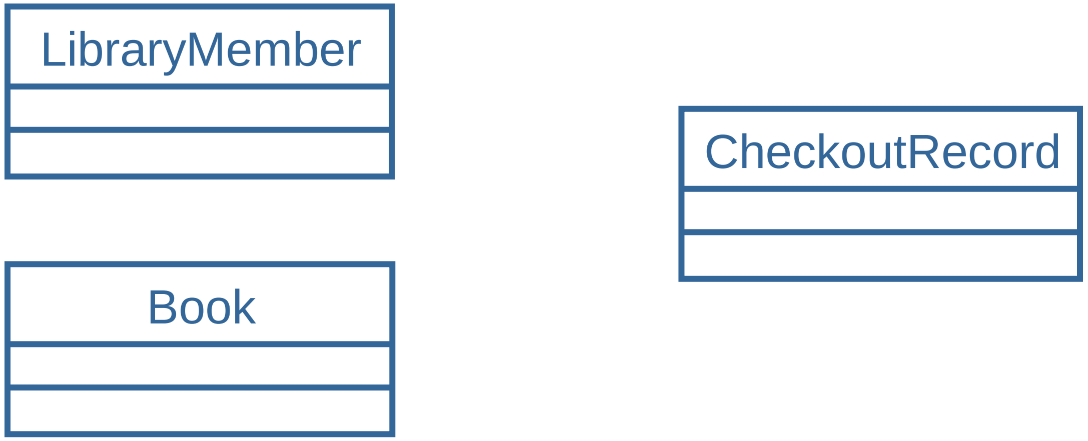
All books in the library have a unique bookid.The books in the library are ordered on the shelves by their bookid. The new system must allow library members to search through the electronic card catalog to find the bookid of the desired book.

The system will run on a number of individual desktops throughout the library. Librarians will have their own desktop computers that are not accessible by library members. Only librarians are able to check-in and checkout books.

The system will retain information on all library members. Only university students, faculty and staff can become library members. Students can check-out books for a maximum of 21 days. If a student returns a book later than 21 days, then he/she has to pay an overdue fee of 25 cents per day. University staff can also checkout books for a maximum of 21 days, but pay an overdue fee of 10 cents per day. Faculty can checkout books for a maximum of 100 days, and pay only 5 cents per day for every book returned late. The system will keep track of the amount of money that library members owe the library.

Example: Find Entity Classes

- Check-out book



Question: Why is LibraryMember considered an Entity class but not CardCatalog?

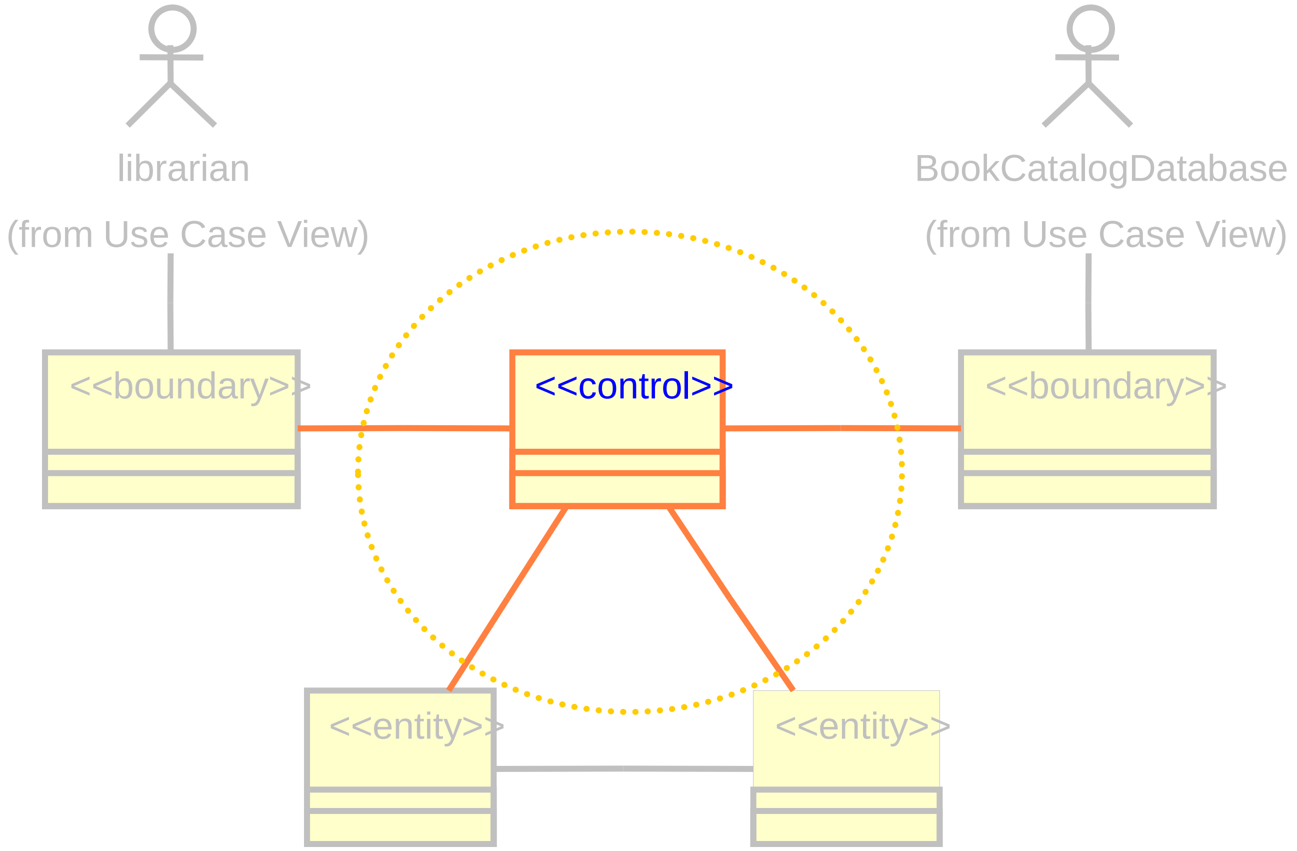
J184-5, J204, Q57

Control Classes

- Represent coordination, sequencing, transactions among groups of objects
- Encapsulate control of individual use-cases
- Guideline: One control class per use case
- Control classes provide behavior that:
  - Defines control logic (order between events) and transactions within a use case. Changes little if the internal structure or behavior of the entity classes changes
  - Uses or sets the contents of several entity classes, and therefore needs to coordinate the behavior of these entity classes
  - Is not performed in the same way every time it is activated (flow of events features several states)

J185, J205, Q58-9

Control Class as Use-Case Coordinator

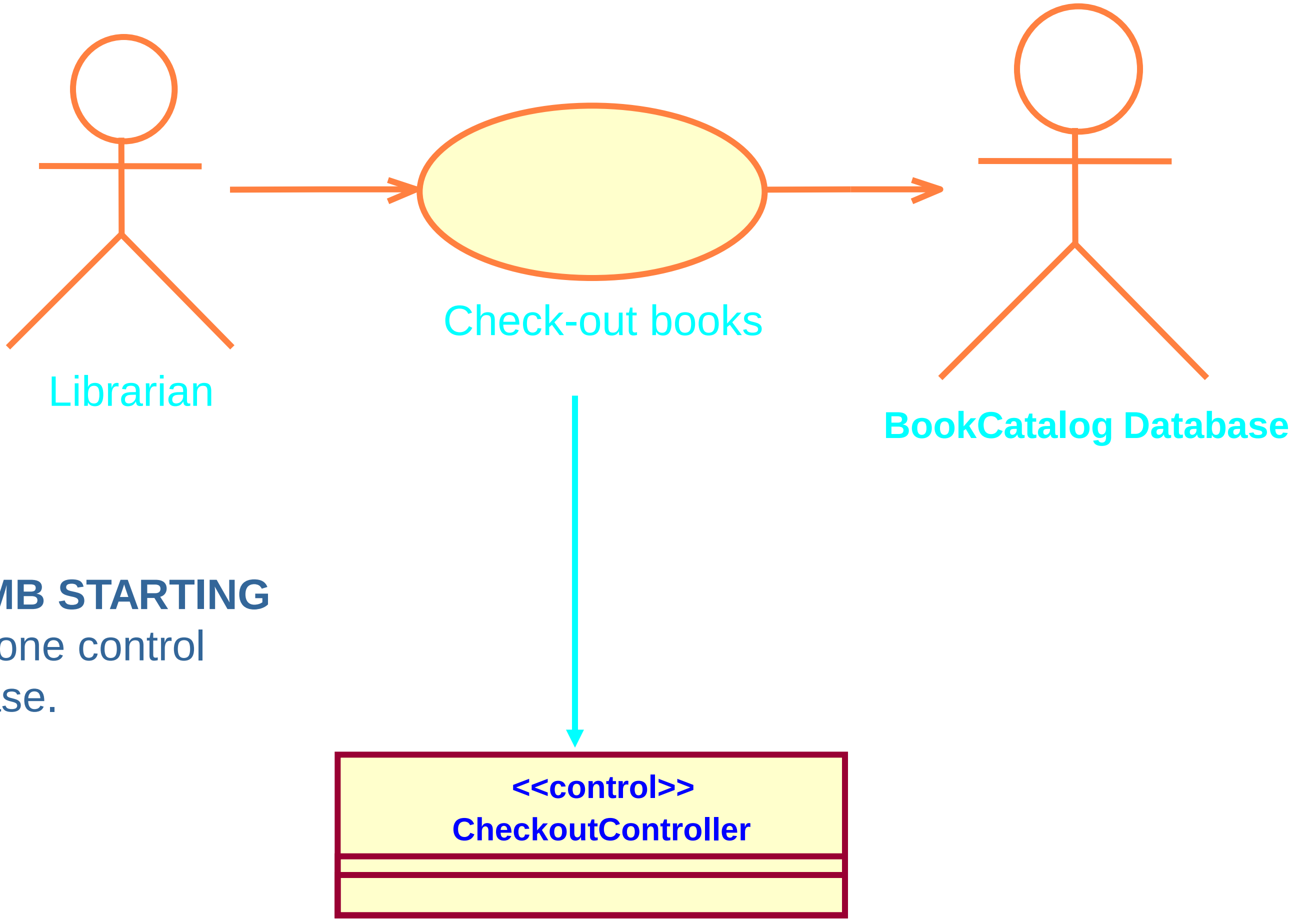


Note: Not always necessary for a use-case realization to contain a control class. Sometimes control is better encapsulated in a boundary class.

J185, J205, Q58-9

Introduce Control Classes

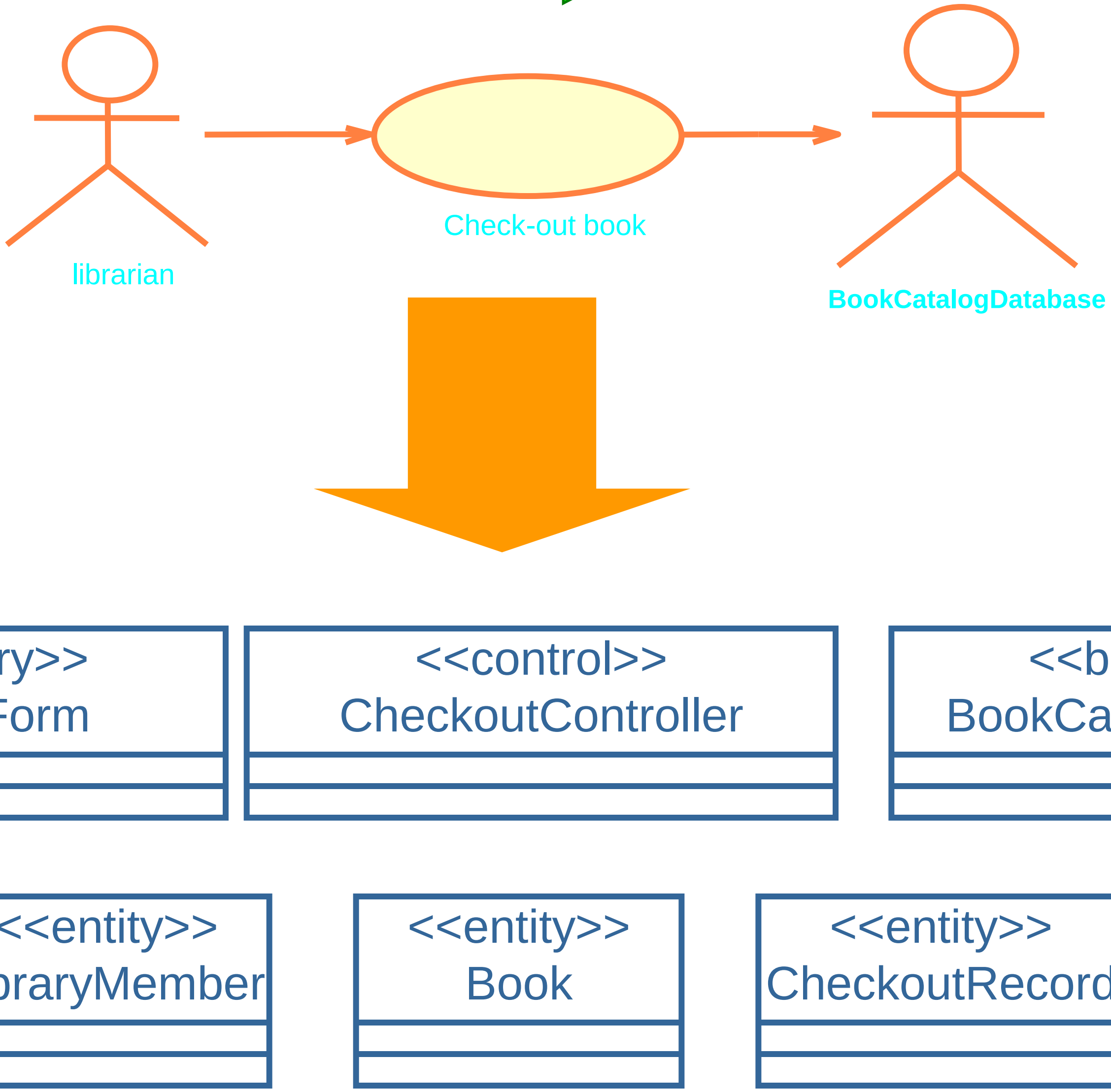
- For each use case



RULE OF THUMB STARTING OUT: Introduce one control class per use case.

J185, J205, Q58-9

Example: Analysis Classes


librarian
Check-out book
BookCatalogDatabase
<<boundary>> CheckoutForm
<<control>> CheckoutController
<<boundary>> BookCatalogDatabase
<<entity>> LibraryMember
<<entity>> Book
<<entity>> CheckoutRecord

J181-5, J204-5, Q57-9

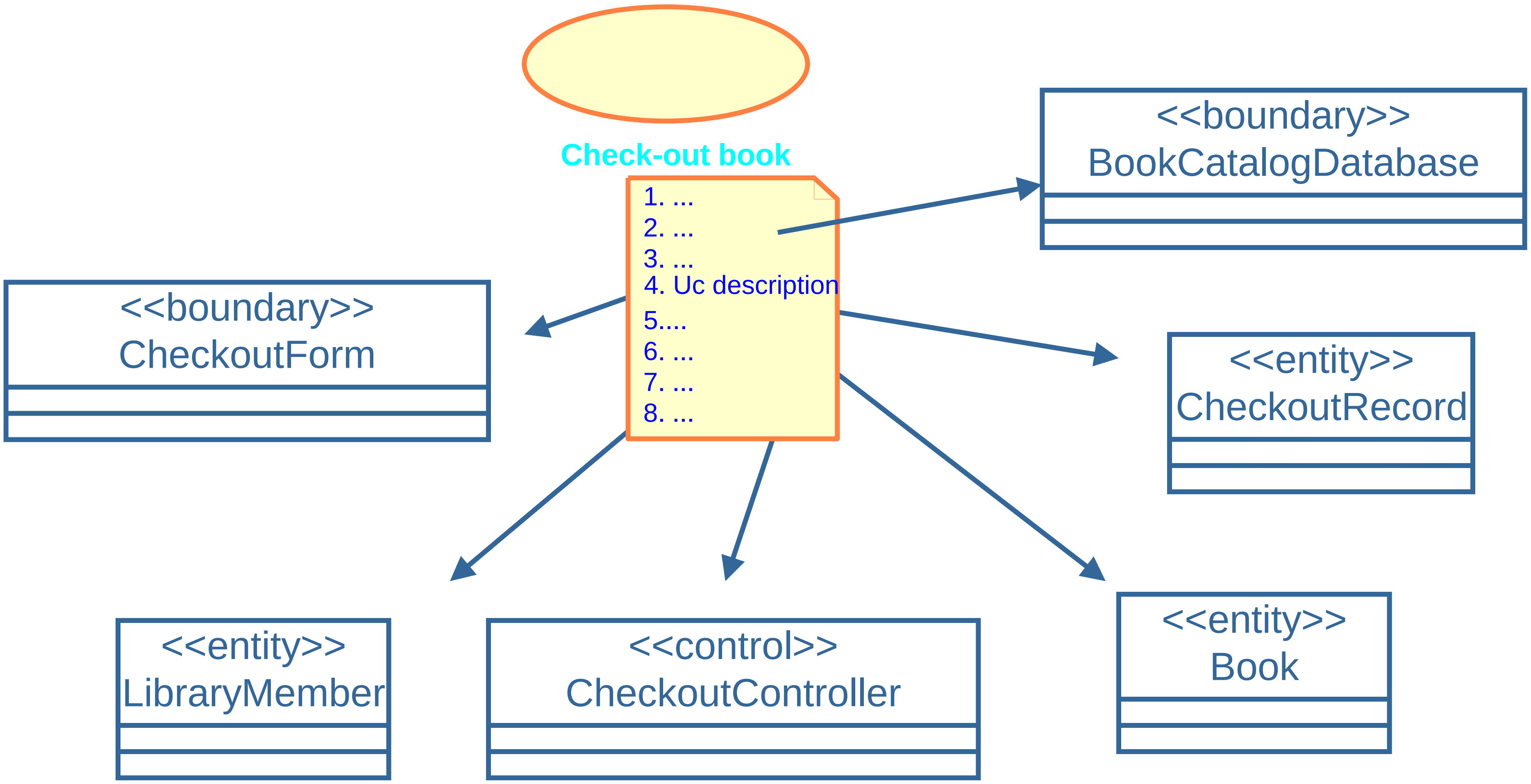
Use-Case Analysis Steps

- Refine the use-case description
- Identify analysis classes for the use-case
- Model behavior using a sequence diagram
- Model object collaborations
- Model structure in VOPC diagram
  - Capture responsibilities from sequence diagram
  - Add analysis-level attributes and associations
  - Note analysis mechanisms
- Integrate analysis classes
- Document business rules

J203-6, Q73-4

Assign Use-Case Behavior to Classes

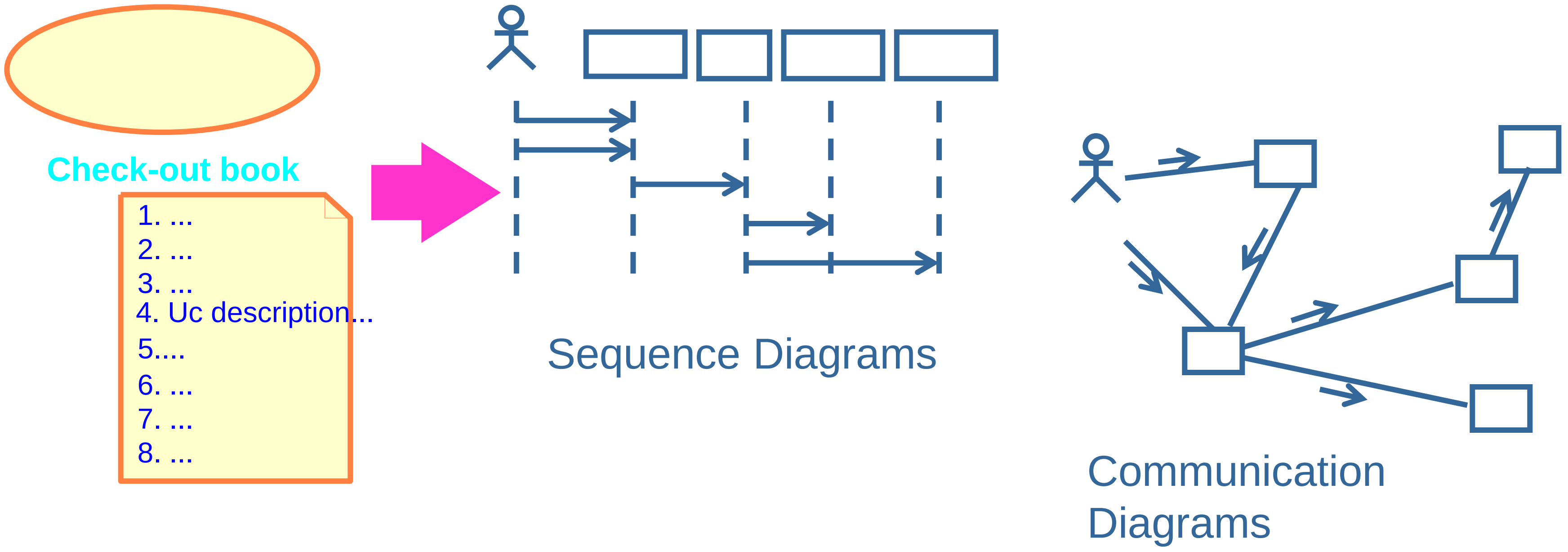
- Assign all use case steps to the analysis classes.



J203-6, Q73-4

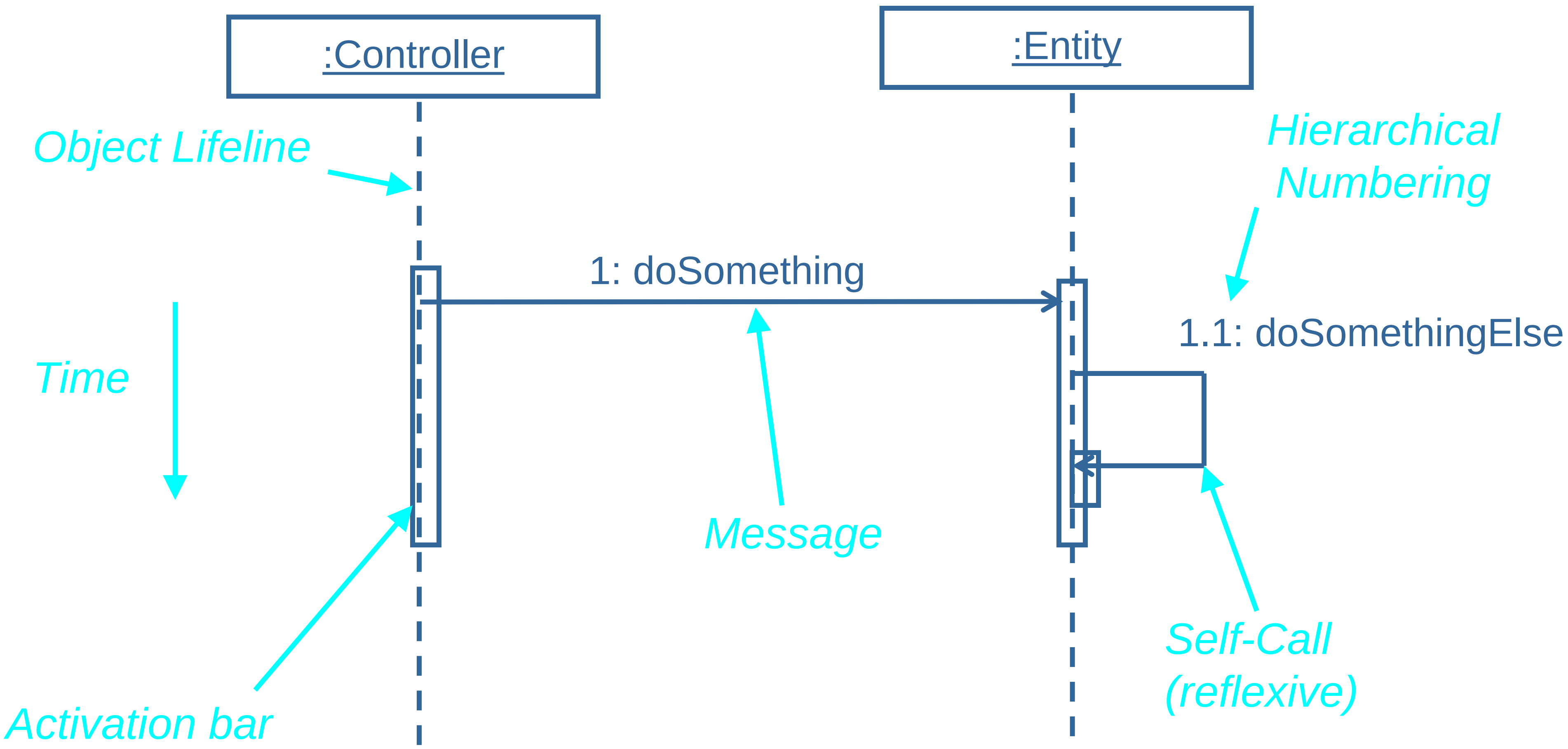
Assign Use-Case Behavior to Classes

- For each flow of events
  - Identify analysis classes
  - Model use-case behavior in interaction diagrams
  - Don’t model interactions between actors



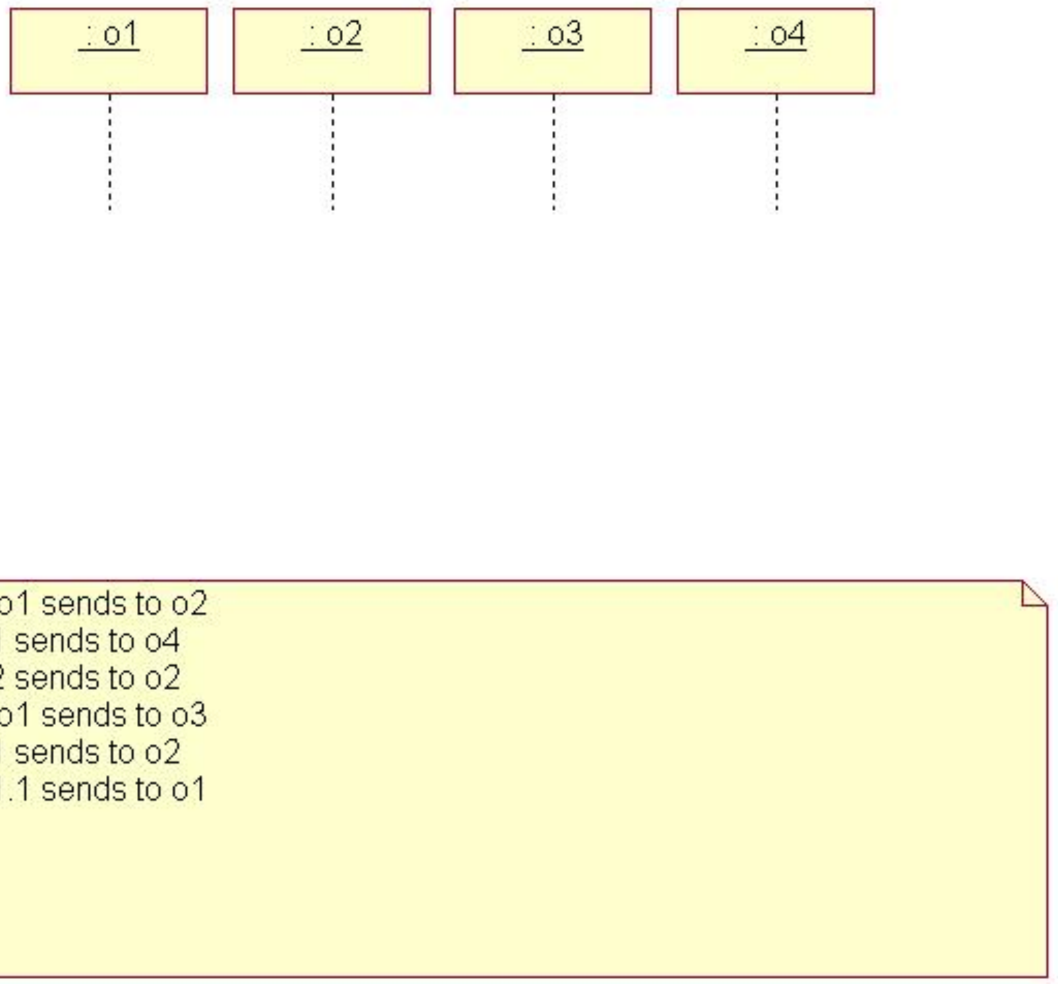
J203-6, Q73-4

Sequence Diagram Features



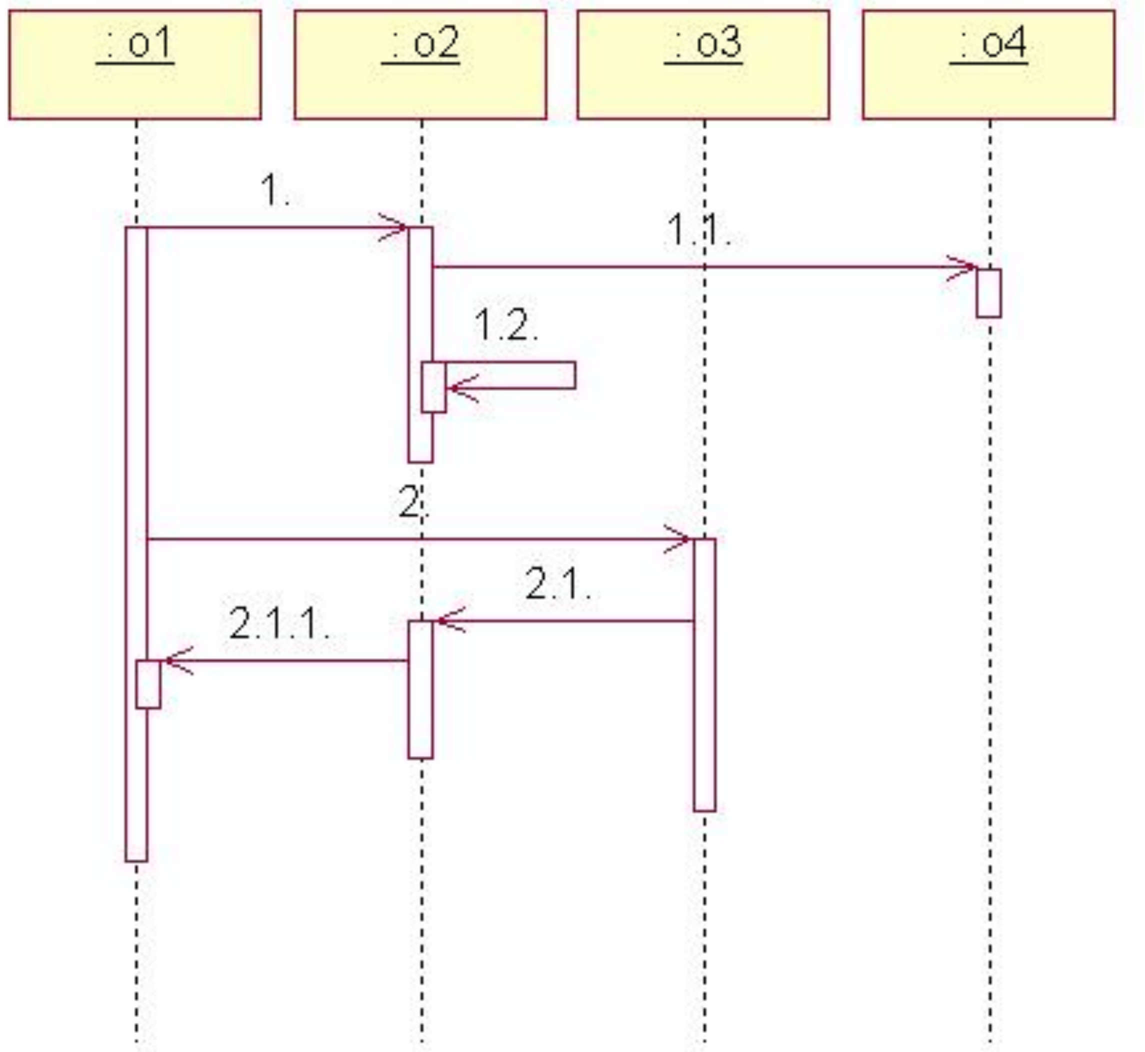
F68-71

Example



1. o1 sends to o2
1.1 sends to o4
1.2 sends to o2
2. o1 sends to o3
2.1 sends to o2
2.1.1 sends to o1

Example (cont)



Example: Sequence Diagram

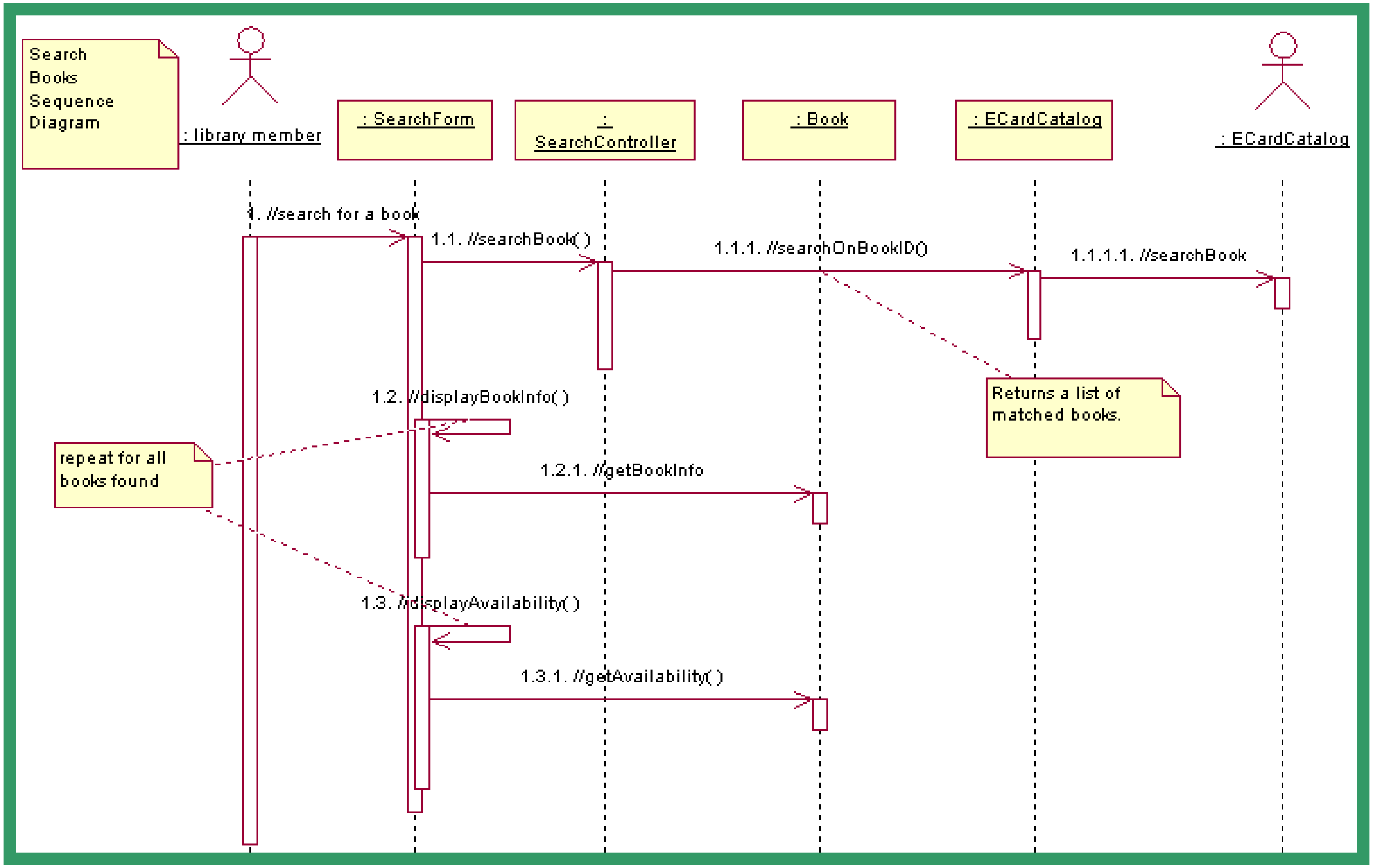
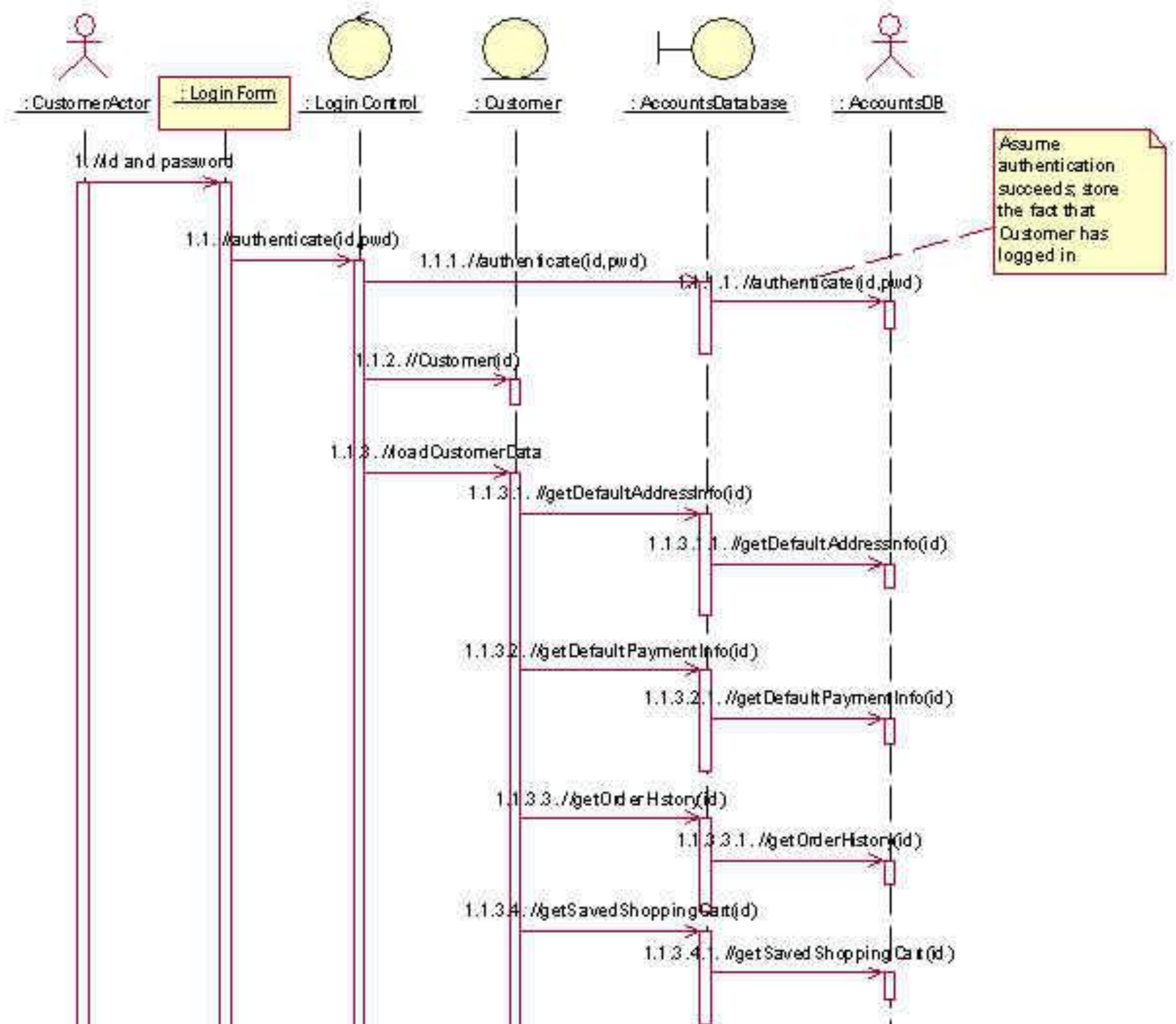

Search Books Sequence Diagram
: library member
: SearchForm
: SearchController
: Book
: ECardCatalog
: ECardCatalog
1. //search for a book
1.1. //searchBook( )
1.1.1. //searchOnBookID()
1.1.1.1. //searchBook
Returns a list of matched books.
1.2. //displayBookInfo( )
repeat for all books found
1.2.1. //getBookInfo
1.3. //displayAvailability( )
1.3.1. //getAvailability( )

Diagram Scenarios vs Flows vs Use Case?

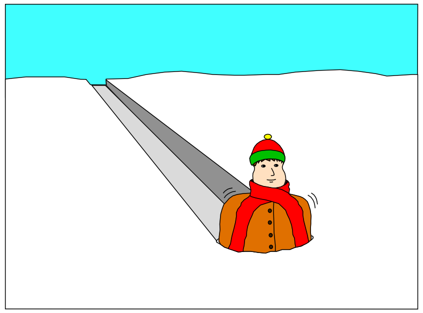
- A scenario is a single pass
  - Straightforward to create and read
  - Full coverage infeasible
- A flow is a set of similar scenarios
  - Might involve conditions or loops
  - Separate diagrams for significant flows
- A use-case is the set of all flows
  - In general, too complex for interaction diagrams

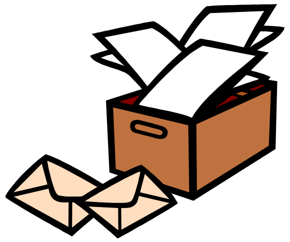
E-Bazaar Example of Sequence Diagram


: CustomerActor
: Login Form
: Login Control
: Customer
: AccountsDatabase
: AccountsDB
Assume authentication succeeds; store the fact that Customer has logged in
1. //id and password
1.1. //authenticate(id,pwd)
1.1.1. //authenticate(id,pwd)
1.1.1.1. //authenticate(id,pwd)
1.1.2. //Customer(id)
1.1.3. //loadCustomerData
1.1.3.1. //getDefaultAddressInfo(id)
1.1.3.1.1. //getDefaultAddressInfo(id)
1.1.3.2. //getDefaultPaymentInfo(id)
1.1.3.2.1. //getDefaultPaymentInfo(id)
1.1.3.3. //getOrderHistory(id)
1.1.3.3.1. //getOrderHistory(id)
1.1.3.4. //getSavedShoppingCart(id)
1.1.3.4.1. //getSavedShoppingCart(id)

Analysis Class Stereotypes Guide Assignments

- Boundary classes responsible for actor communications
- Entity classes responsible for processing persistent information
- Control classes responsible for use-case specific interactions or mediating other event flows



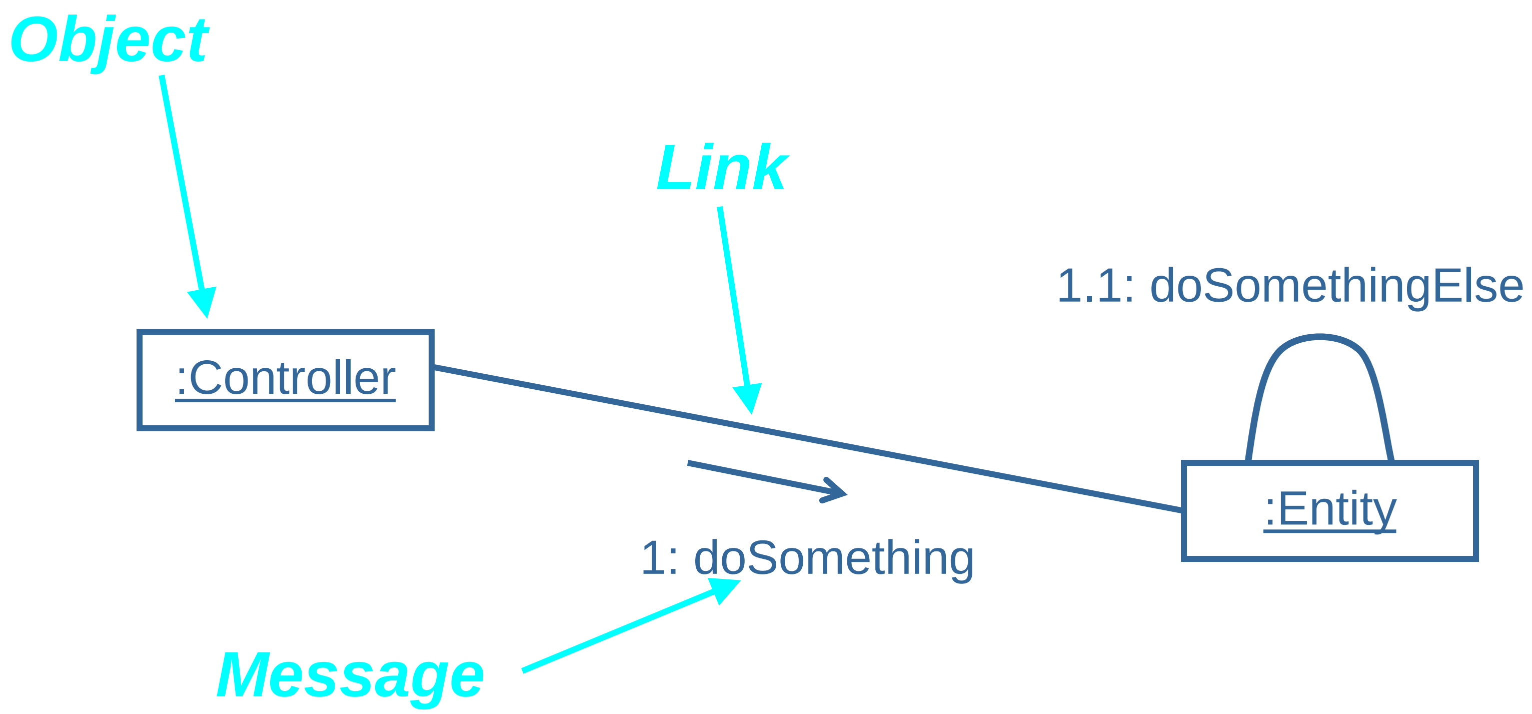


Q57-9, J181-5, J204-5

Further Responsibility Assignment Guidelines

- In general, class with the data should have the responsibility
- If data is spread across classes
  - Put the responsibility with one class and add a relationship to the other
  - Create a new class, put the responsibility in the new class, and add relationships to classes needed to perform the responsibility
  - Put the responsibility in the control class, and add relationships to classes needed to perform the responsibility

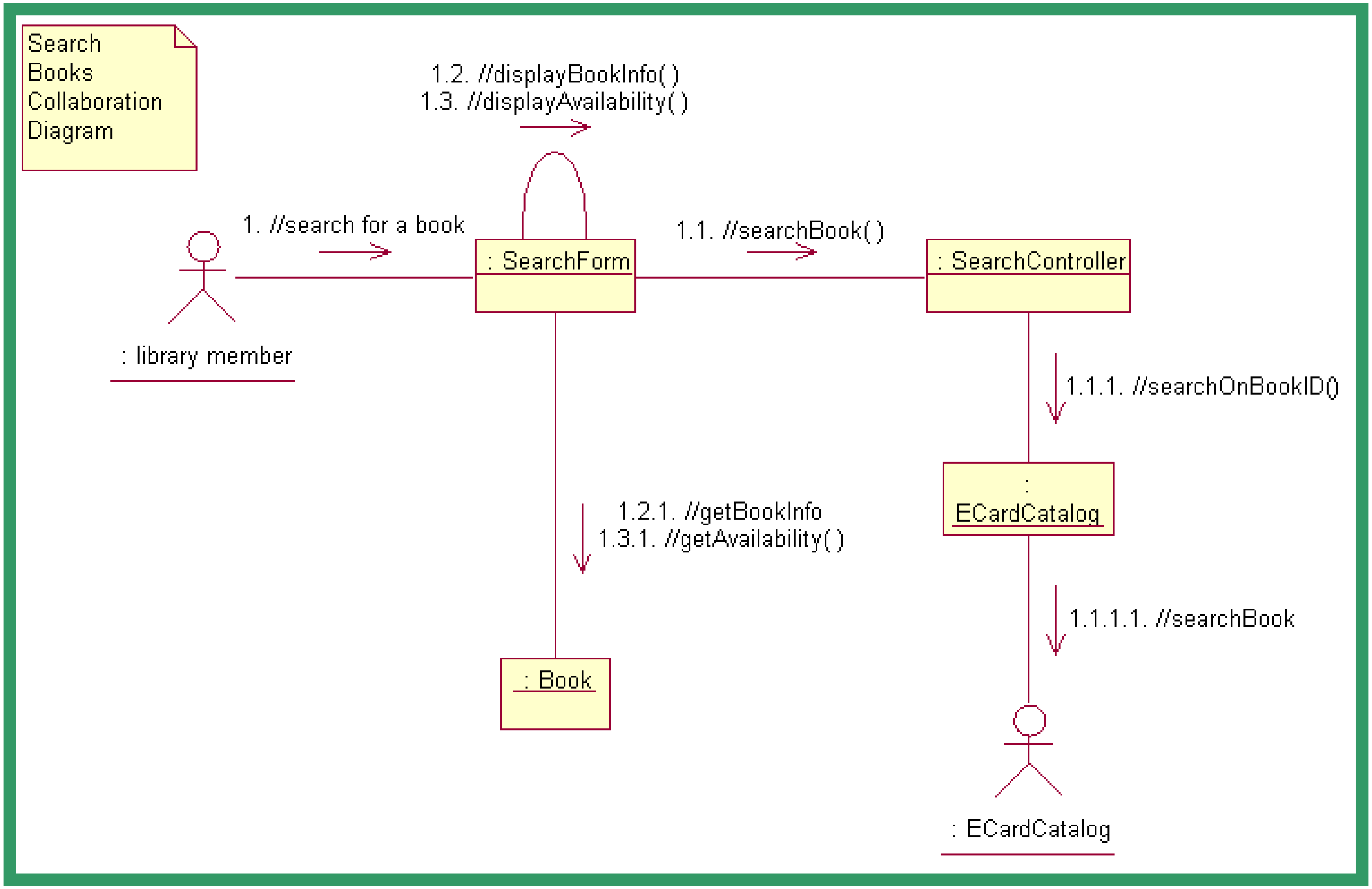
Collaboration Diagrams



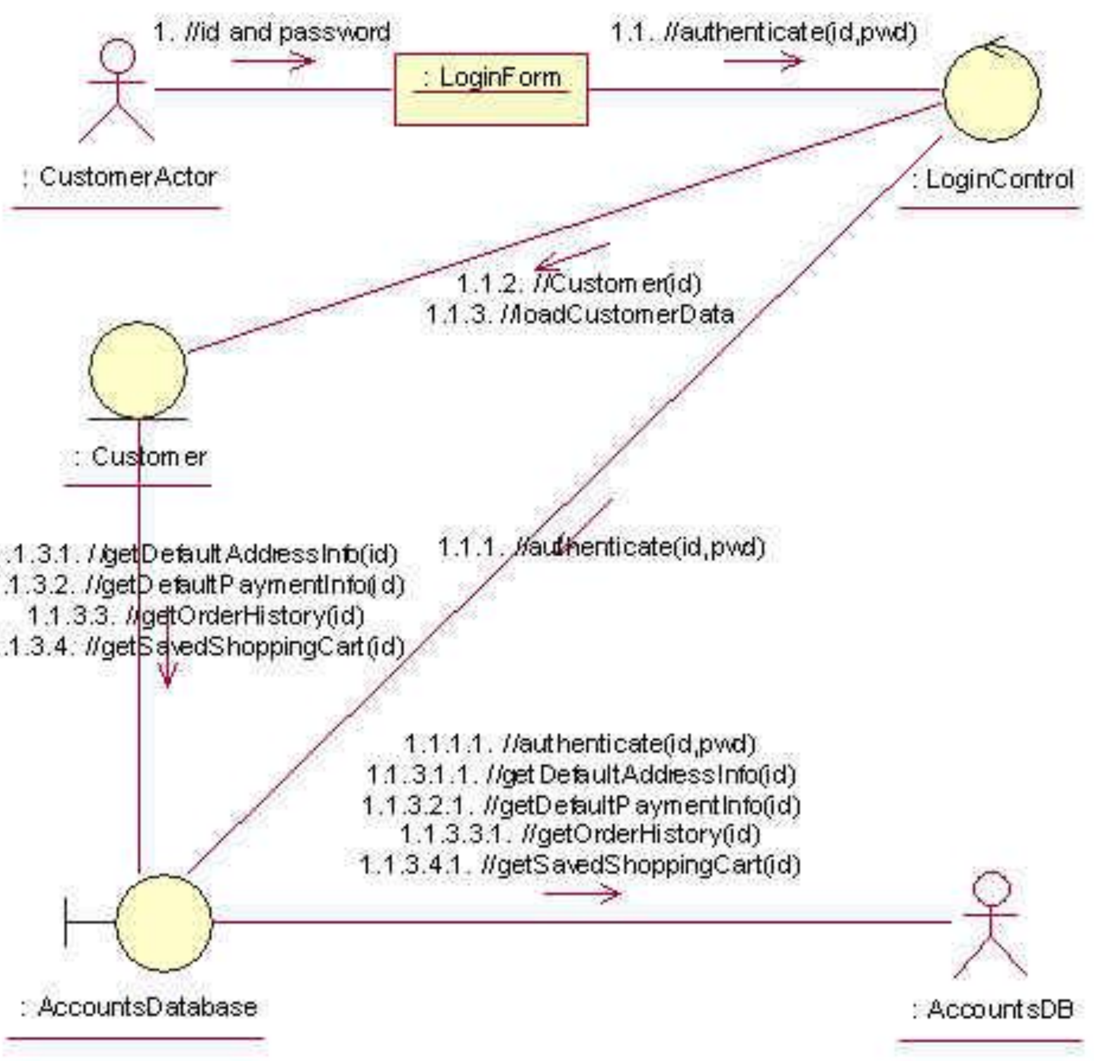
1. A *link* is a relationship among objects across which a message can be sent
2. A *message* is a communication between objects that conveys information resulting in some activity – shown with a labeled arrow
3. Often use sequence numbers to label messages to keep track of the flow of events

F:72-4

Example: Collaboration Diagram



E-Bazaar Example of Collaboration Diagram



1. //id and password
1.1. //authenticate(id,pwd)
: LoginForm
: CustomerActor
: LoginControl
1.1.2. //Customer(id)
1.1.3. //loadCustomerData
: Customer
1.1.3.1. //getDefaultAddressInfo(id)
1.1.3.2. //getDefaultPaymentInfo(id)
1.1.3.3. //getOrderHistory(id)
1.1.3.4. //getSavedShoppingCart(id)
1.1.1. //authenticate(id,pwd)
1.1.1.1. //authenticate(id,pwd)
1.1.3.1.1. //getDefaultAddressInfo(id)
1.1.3.2.1. //getDefaultPaymentInfo(id)
1.1.3.3.1. //getOrderHistory(id)
1.1.3.4.1. //getSavedShoppingCart(id)
: AccountsDatabase
: AccountsDB

Sequence vs Collaboration Diagrams

- Sequence Diagrams
  - Show the explicit sequence of messages
  - Better for visualizing overall flow
  - Better for real-time specifications

- Collaboration Diagrams
  - Show relationships in addition to interactions (use for VOPC diagrams)
  - Better for visualizing patterns of collaboration
  - Better for visualizing all of the effects on a given object

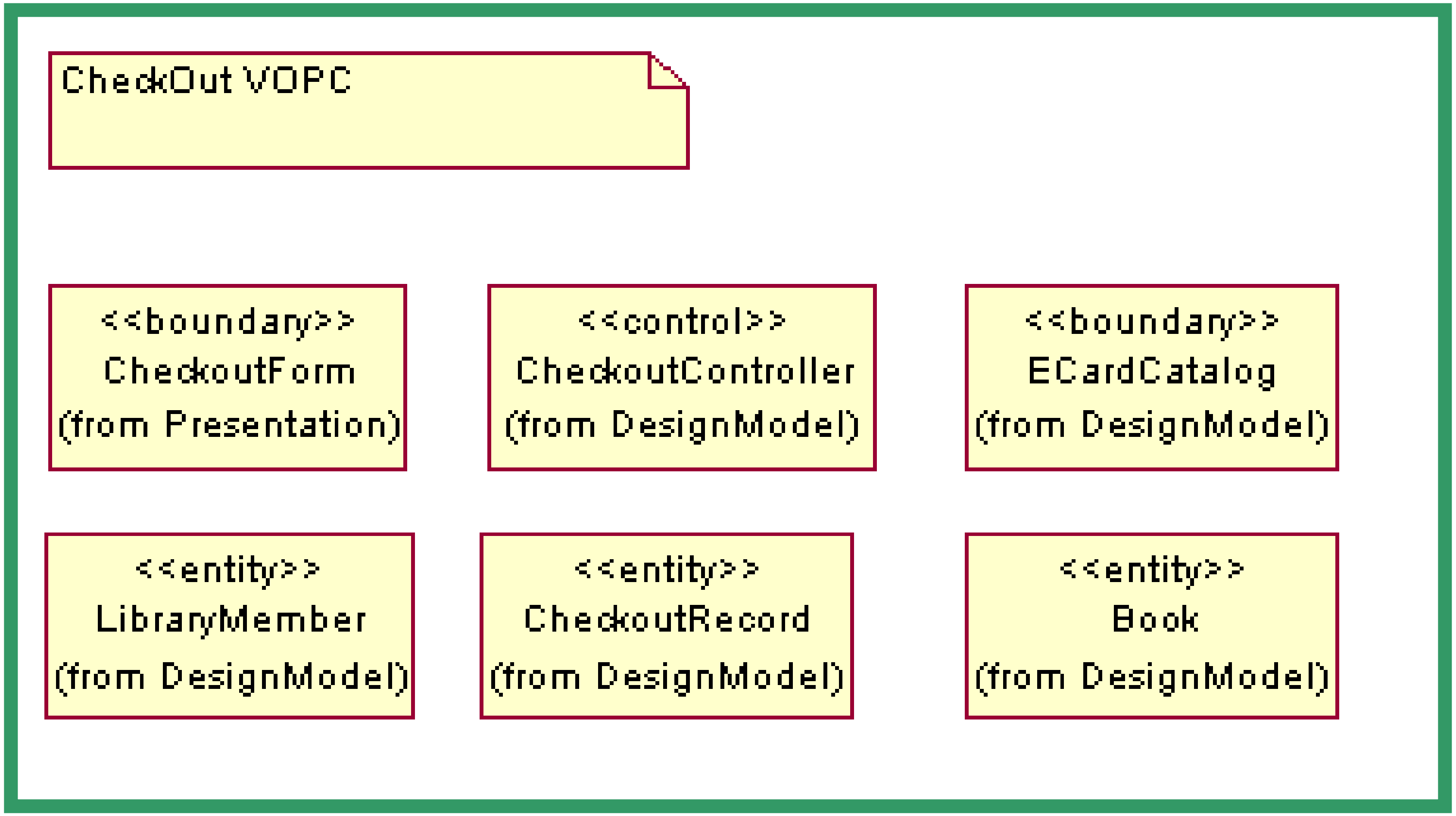
F75, Q84

Use-Case Analysis Steps

- Refine the use-case description
- Model behavior using a sequence diagram
  - Identify analysis classes for the use-case
  - Model object collaborations
- Model structure in VOPC diagram
  - Capture responsibilities from sequence diagram
  - Add analysis-level attributes and associations
  - Note analysis mechanisms
- Integrate analysis classes
- Document business rules

Building the VOPC Diagram

May use the Key Abstractions diagram as a starting point.


CheckOut VOPC
<<boundary>> CheckoutForm (from Presentation)
<<control>> CheckoutController (from DesignModel)
<<boundary>> ECardCatalog (from DesignModel)
<<entity>> LibraryMember (from DesignModel)
<<entity>> CheckoutRecord (from DesignModel)
<<entity>> Book (from DesignModel)

J186,Q85

Use-Case Analysis Steps

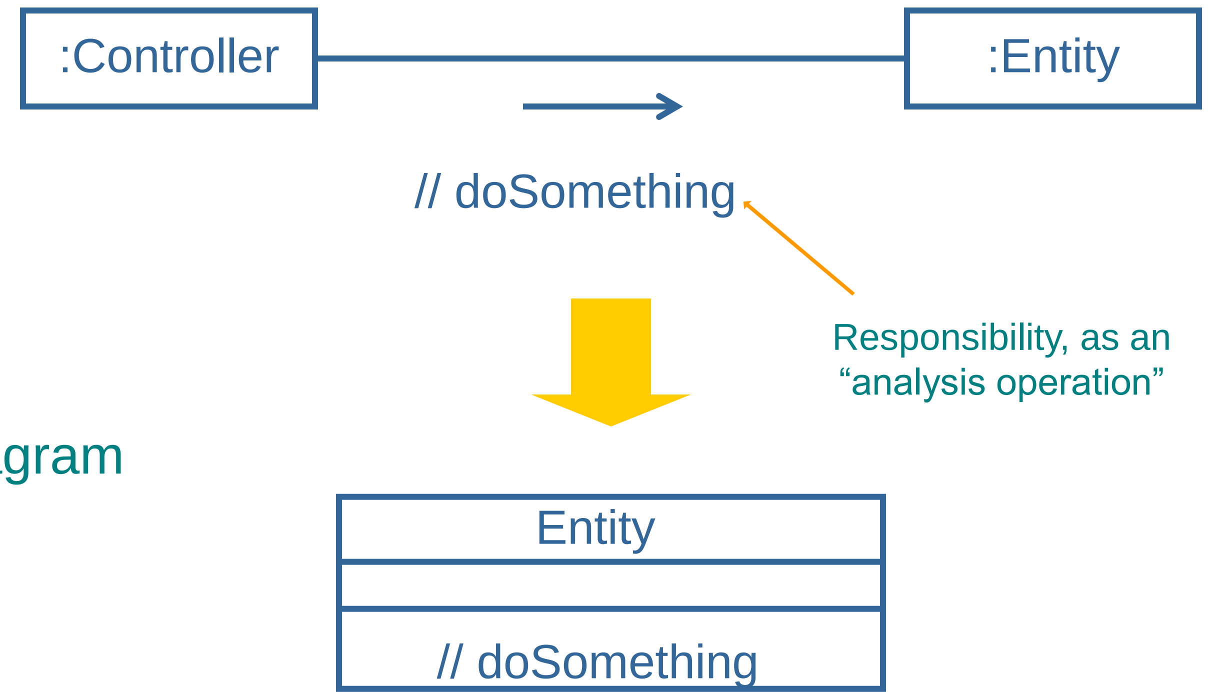
- Refine the use-case description
- Model behavior using a sequence diagram
  - Identify analysis classes for the use-case
  - Model object collaborations
- Model structure in VOPC diagram
  - Capture responsibilities from interaction diagram
  - Add analysis-level attributes and relationships
  - Note analysis mechanisms
- Integrate analysis classes
- Document business rules

J207-8, Q85

Responsibilities from Collaboration Diagram

- Responsibilities are specifications of object behavior

Collaboration Diagram



Class Diagram

Guidelines for (re)Structuring Responsibilities and Classes

- “Orthogonal” responsibilities within classes
  - ➔separate (e.g. //parseMessage and //displayMessage)
- Redundant responsibilities across classes
  - ➔Integrate (e.g. two classes determining look of gui)
- Each analysis class should have several compatible responsibilities. A class with only one responsibility is probably too simple

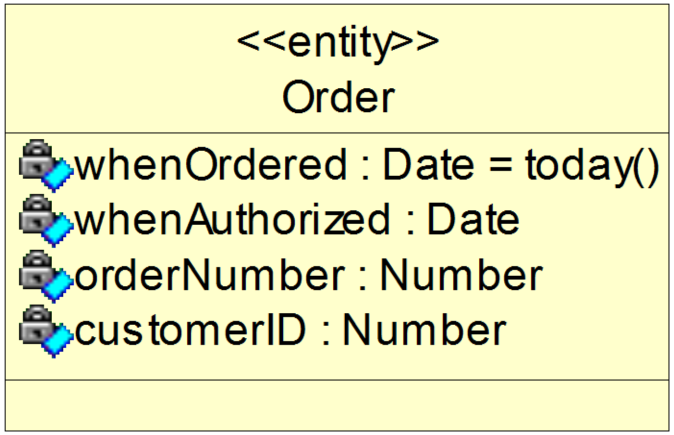
Q105

Use-Case Analysis Steps

- Refine the use-case description
- Model behavior using a sequence diagram
  - Identify analysis classes for the use-case
  - Model object collaborations
- Model structure in VOPC diagram
  - Capture responsibilities from sequence diagram
  - Add analysis-level attributes and relationships
  - Note analysis mechanisms
- Integrate analysis classes
- Document business rules

Attributes

- A named property of a class that describes a range of values that instances of the property may hold.
  - Types can be conceptual during analysis. E.g. ‘amount’ might become ‘integer’ or ‘double’ during design.
- Information retained by identified classes



B50,R208

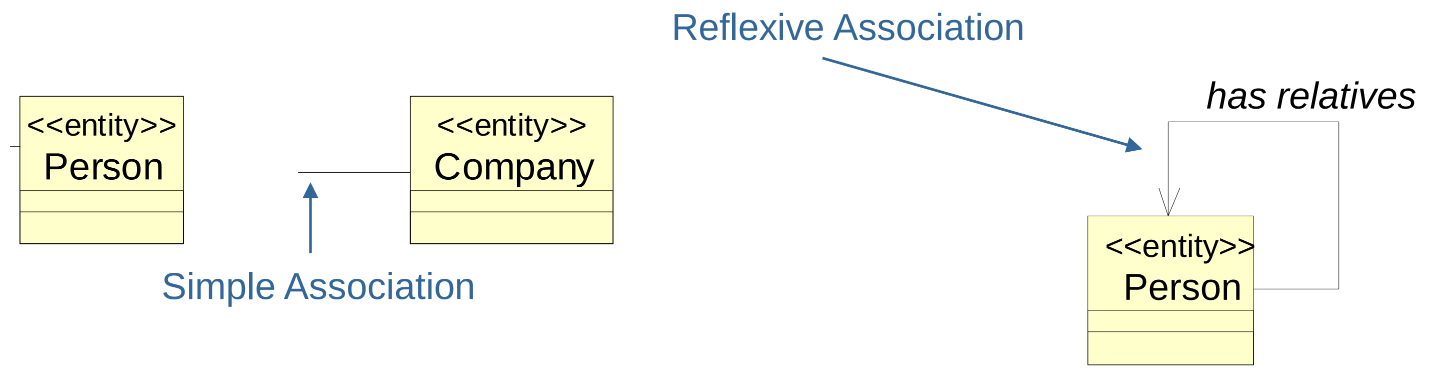
Where To Find Attributes

- Requirements: problem statement, set of system requirements, and flow of events documentation
- Domain expert
- “Nouns” that did not become classes
  - Information whose value is the important thing
  - Information that is uniquely "owned” by an object
  - Information that has no behavior
  - Note: Attributes are often realized as objects like instances of Date or List

Q110, J208-209

Review of Associations

- An association models a structural relationship between objects



B65-66, Q89

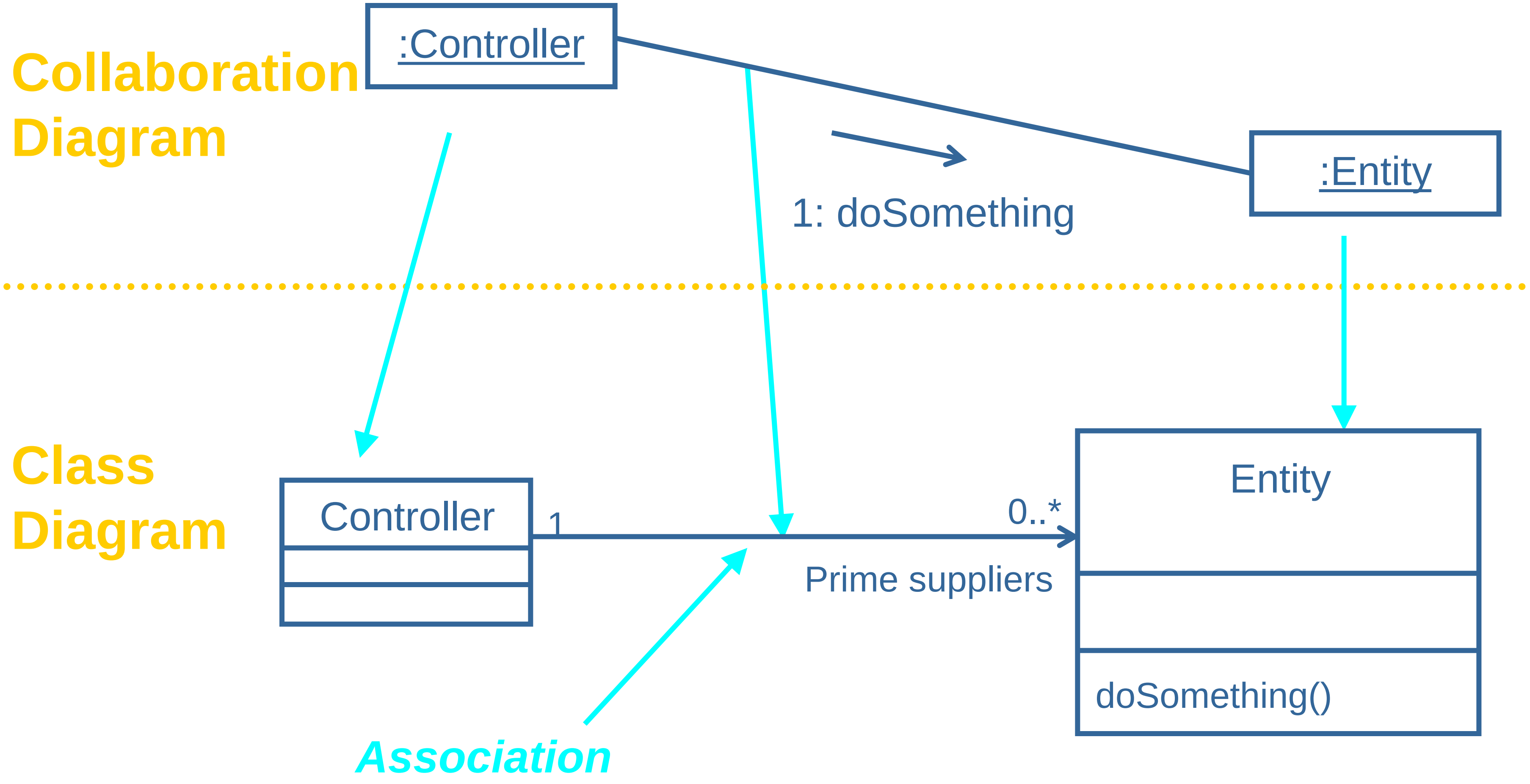
Review of Association Adornments

- Name
- Role
- Multiplicity
- Aggregation

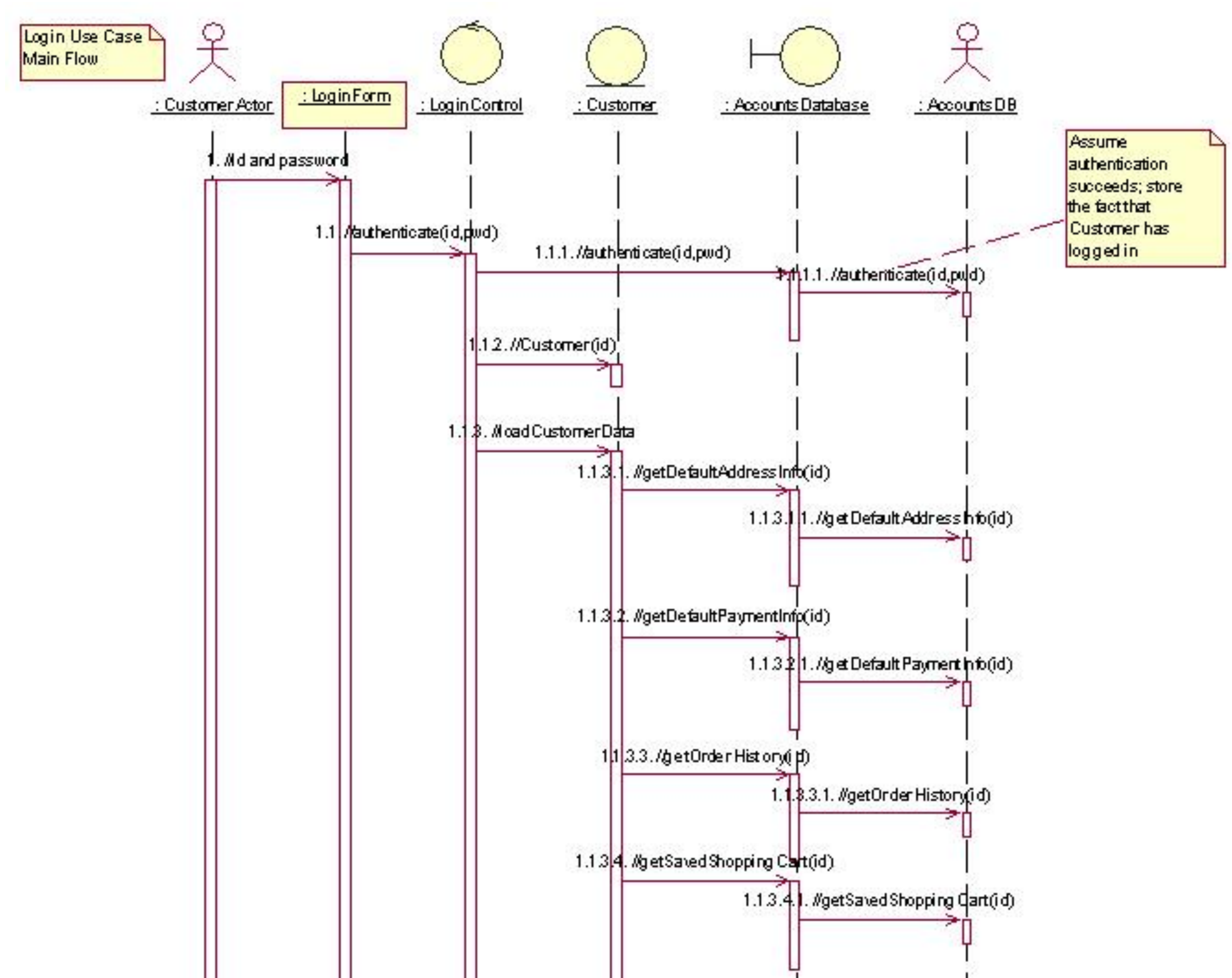
B66

Relationships from Collaboration Diagram

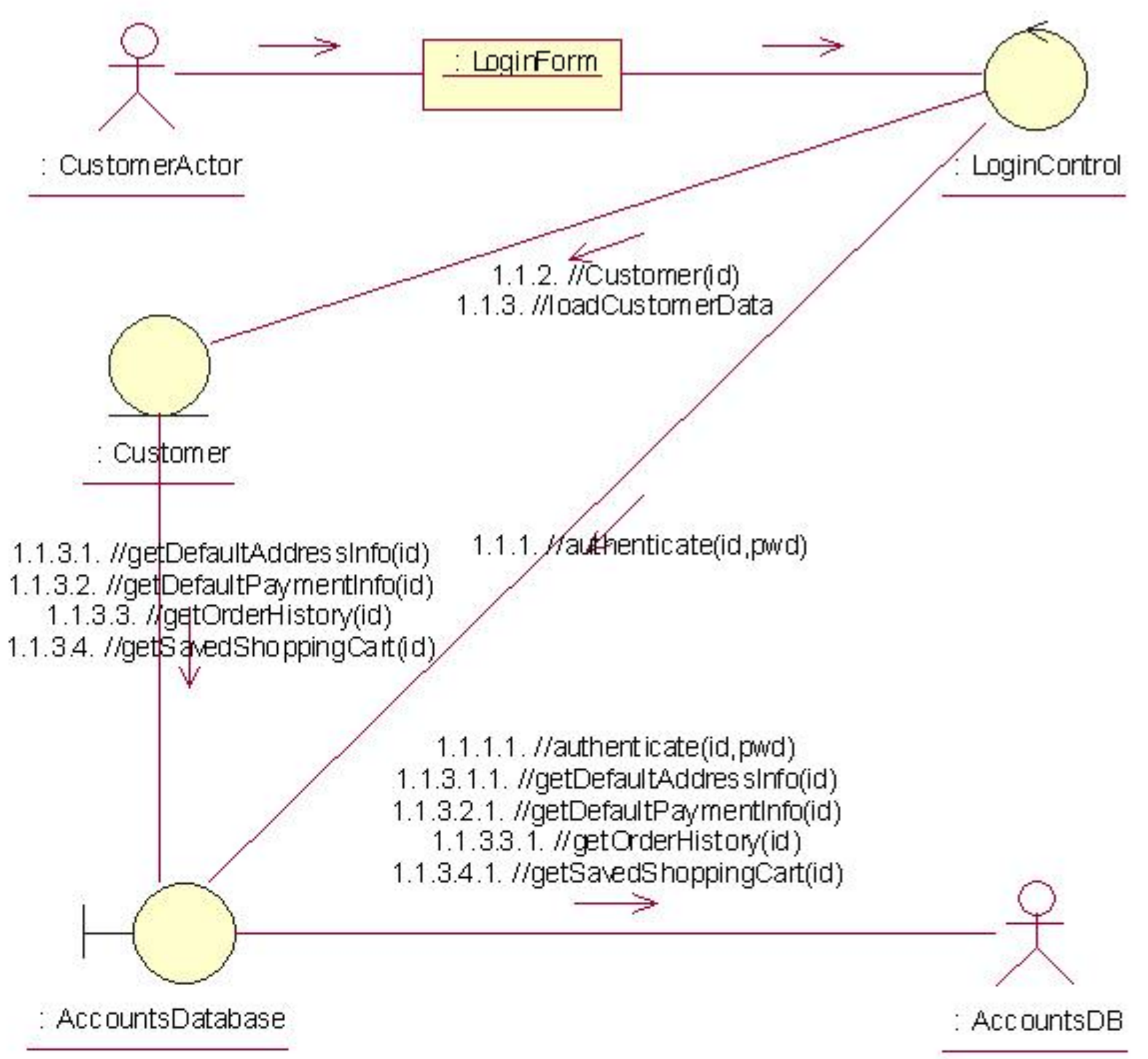
- Create a relationship for each link



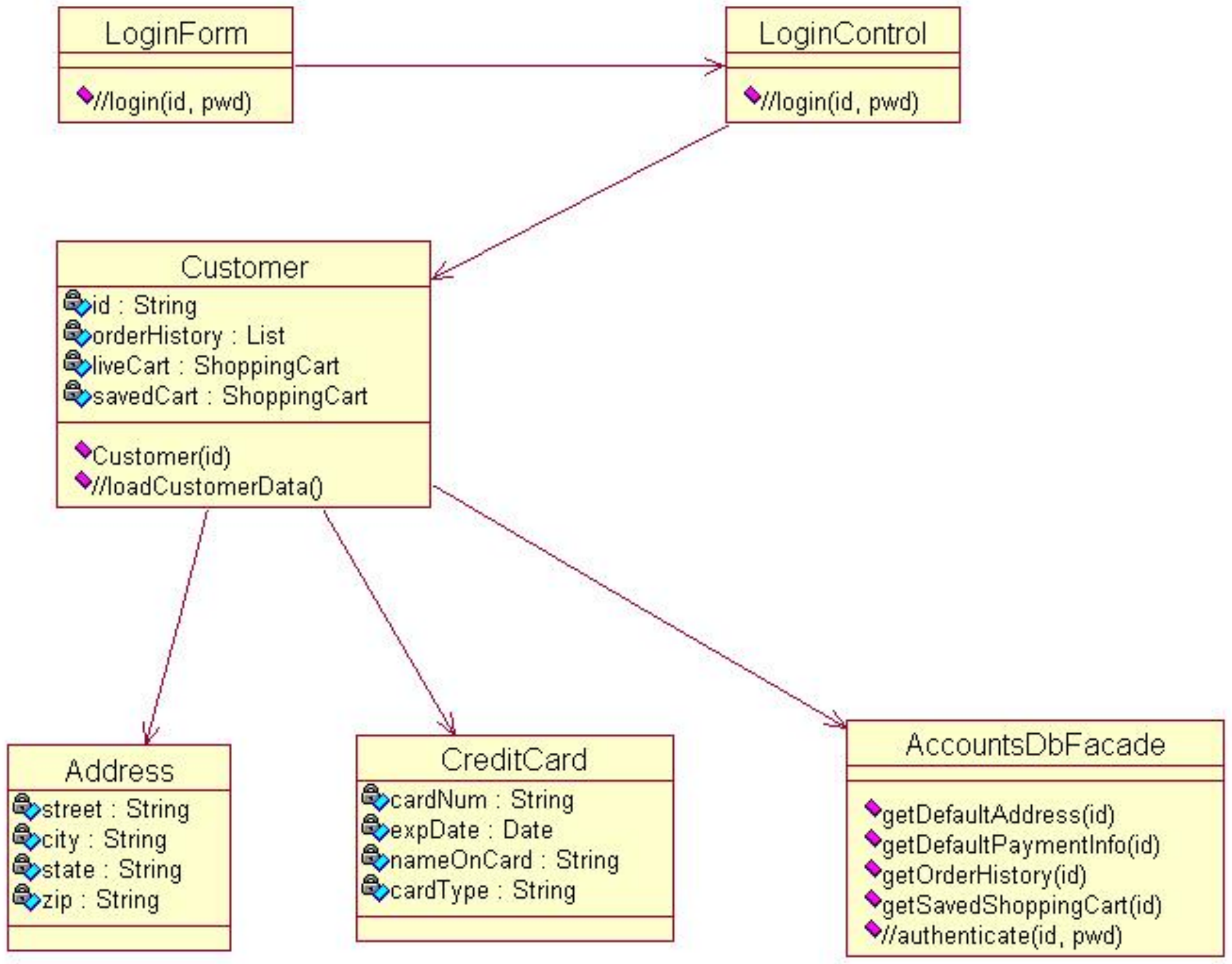
Example: Login Sequence Diag in EBazaar


Login Use Case Main Flow
: Customer Actor
: LoginForm
: Login Control
: Customer
: Accounts Database
: Accounts DB
Assume authentication succeeds; store the fact that Customer has logged in
1. //id and password
1.1. //authenticate(id,pwd)
1.1.1. //authenticate(id,pwd)
1.1.1.1. //authenticate(id,pwd)
1.1.2. //Customer(id)
1.1.3. //loadCustomerData
1.1.3.1. //getDefaultAddressInfo(id)
1.1.3.1.1. //getDefaultAddressInfo(id)
1.1.3.2. //getDefaultPaymentInfo(id)
1.1.3.2.1. //getDefaultPaymentInfo(id)
1.1.3.3. //getOrderHistory(id)
1.1.3.3.1. //getOrderHistory(id)
1.1.3.4. //getSavedShoppingCart(id)
1.1.3.4.1. //getSavedShoppingCart(id)

Example: Login Collab Diag in EBazaar

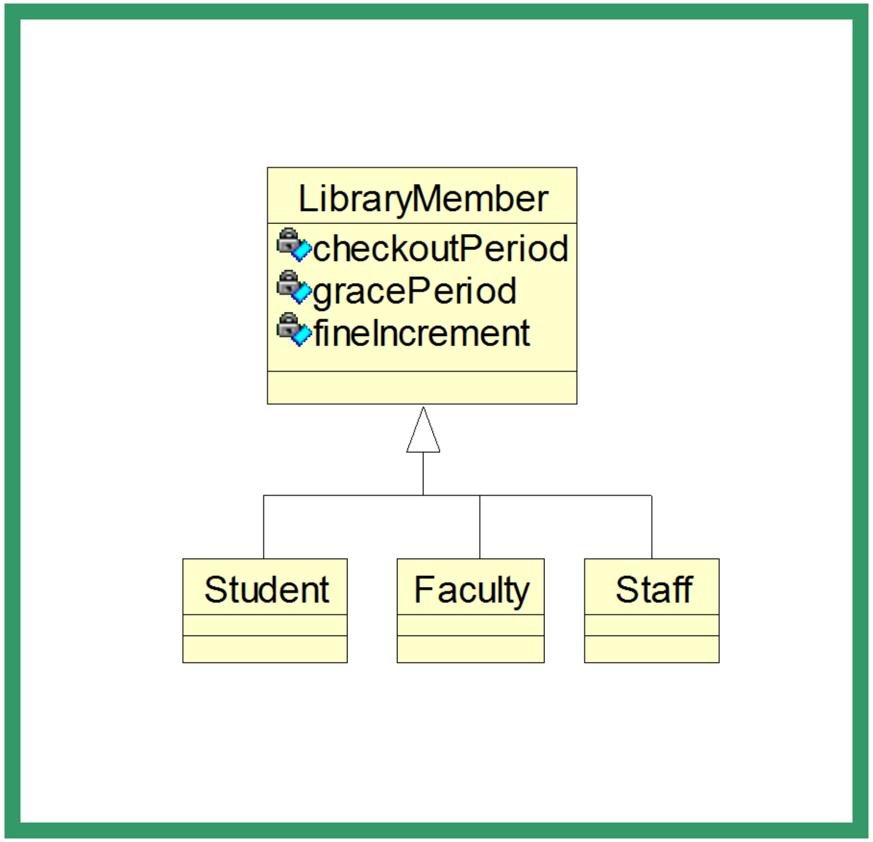

: CustomerActor
: LoginForm
: LoginControl
1.1.2. //Customer(id)
1.1.3. //loadCustomerData
: Customer
1.1.3.1. //getDefaultAddressInfo(id)
1.1.3.2. //getDefaultPaymentInfo(id)
1.1.3.3. //getOrderHistory(id)
1.1.3.4. //getSavedShoppingCart(id)
1.1.1. //authenticate(id,pwd)
1.1.1.1. //authenticate(id,pwd)
1.1.3.1.1. //getDefaultAddressInfo(id)
1.1.3.2.1. //getDefaultPaymentInfo(id)
1.1.3.3.1. //getOrderHistory(id)
1.1.3.4.1. //getSavedShoppingCart(id)
: AccountsDatabase
: AccountsDB

Example: Login VOPC in EBazaar



Review of Generalization

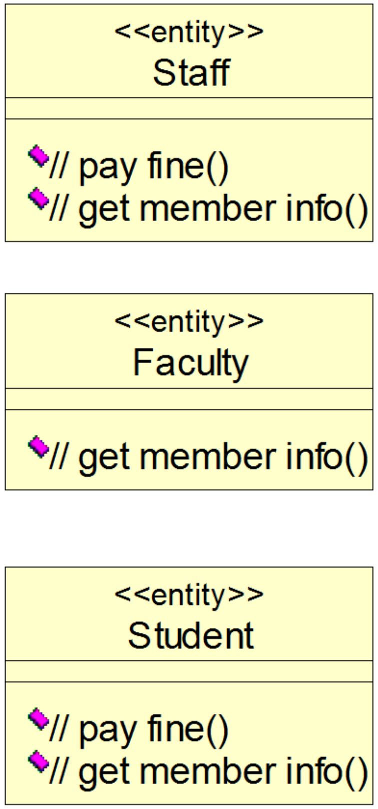
- One class shares the attributes and/or behavior of one or more classes
- “Is-a” relationship
- In analysis, keep on conceptual level just to make model easier to understand

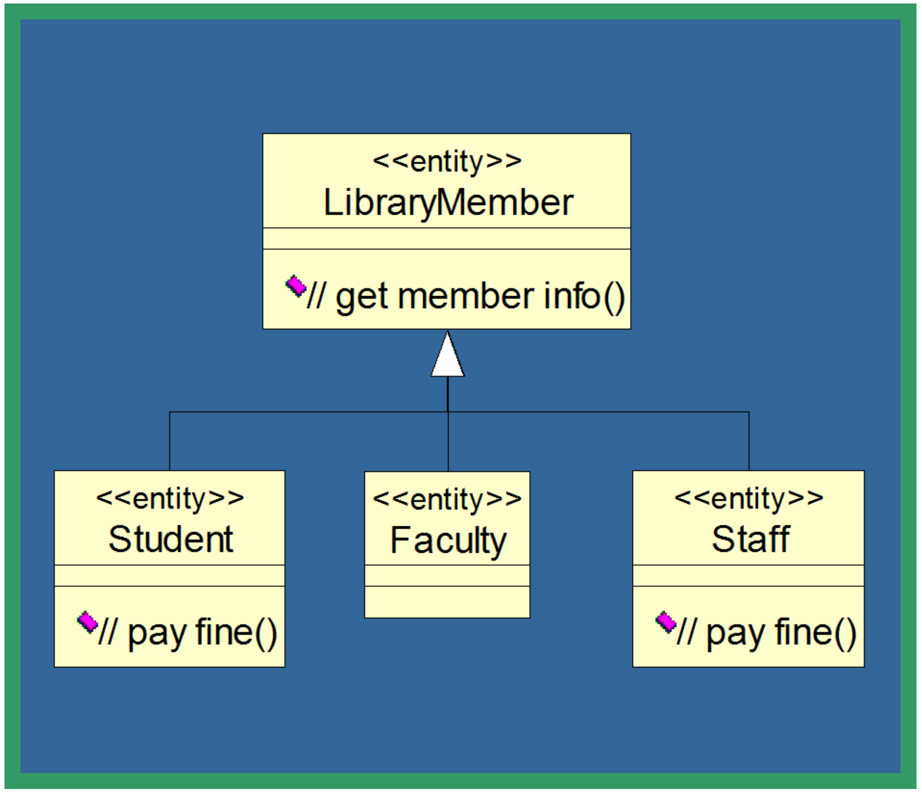


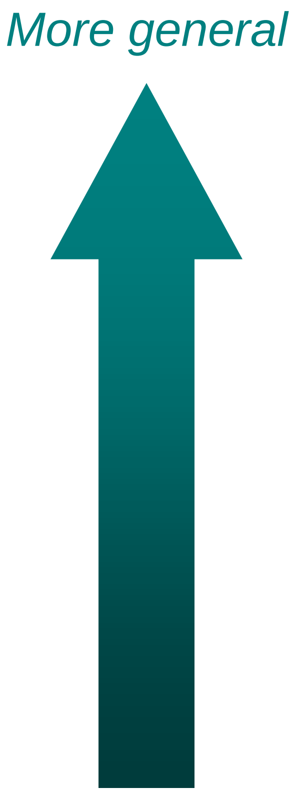
J210-211

Finding Generalization: Generalization of Classes

- Create superclasses which encapsulate structure and common behavior of several classes.



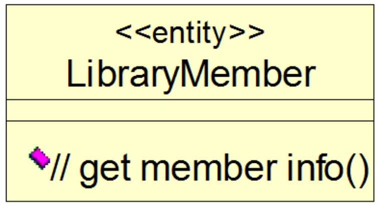


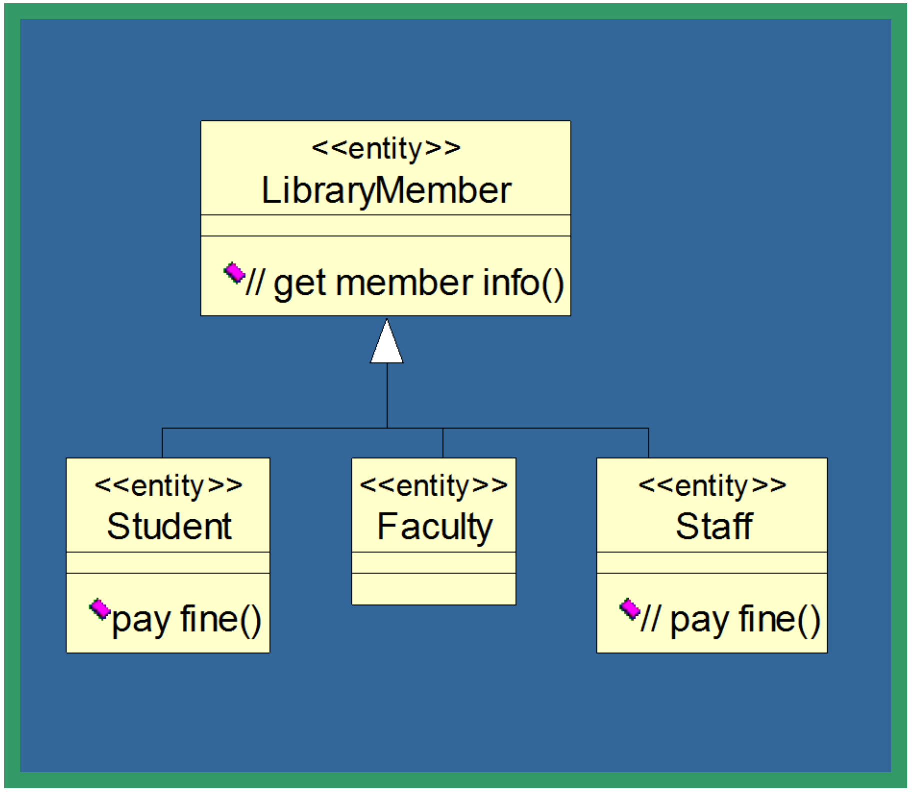


Q210-211

Finding Generalization: Specialization of Classes

- Create subclasses that add refinements of the superclass





More specific

Q210-211

Use-Case Analysis Steps

- Refine the use-case description
- Model behavior using a sequence diagram
  - Identify analysis classes for the use-case
  - Model object collaborations
- Model structure in VOPC diagram
  - Capture responsibilities from sequence diagram
  - Add analysis-level attributes and associations
  - Document analysis mechanisms
- Document business rules
- Integrate analysis classes

Document Analysis Mechanisms

- Map classes to mechanisms

Analysis Class	Analysis Mechanism(s)

- Identify characteristics of the mechanisms

Example: Document Analysis Mechanisms

Analysis class	analysis mechanism(s)
SearchForm	None
SearchController	None
Book	Persistency
LibraryMember	Persistency
CheckoutRecord	Persistency
ECardCatalog	Legacy Interface
CheckoutForm	None
CheckoutController	None
CheckinForm	None
CheckinController	None

Use Case Analysis steps

- ➢Supplement the Use Case Descriptions
- ➢For each use case realization, find classes and distribute use case behavior to classes
- ➢Model the analysis classes with
  - ➢Sequence diagrams to show the flow of the application
  - ➢Collaboration diagrams to suggest relationships
  - ➢VOPC diagrams to specify static relationships and to elaborate attributes of each class

Use-Case Analysis Review

- What is done during Use Case analysis?
- What is a use-case realization?
- What are the input and output artifacts of Use-Case Analysis?
- What is an analysis class? Describe the three analysis stereotypes.
- What are the dynamic and static aspects of use case analysis?
- How many sequence diagrams should be produced during Use-Case Analysis?