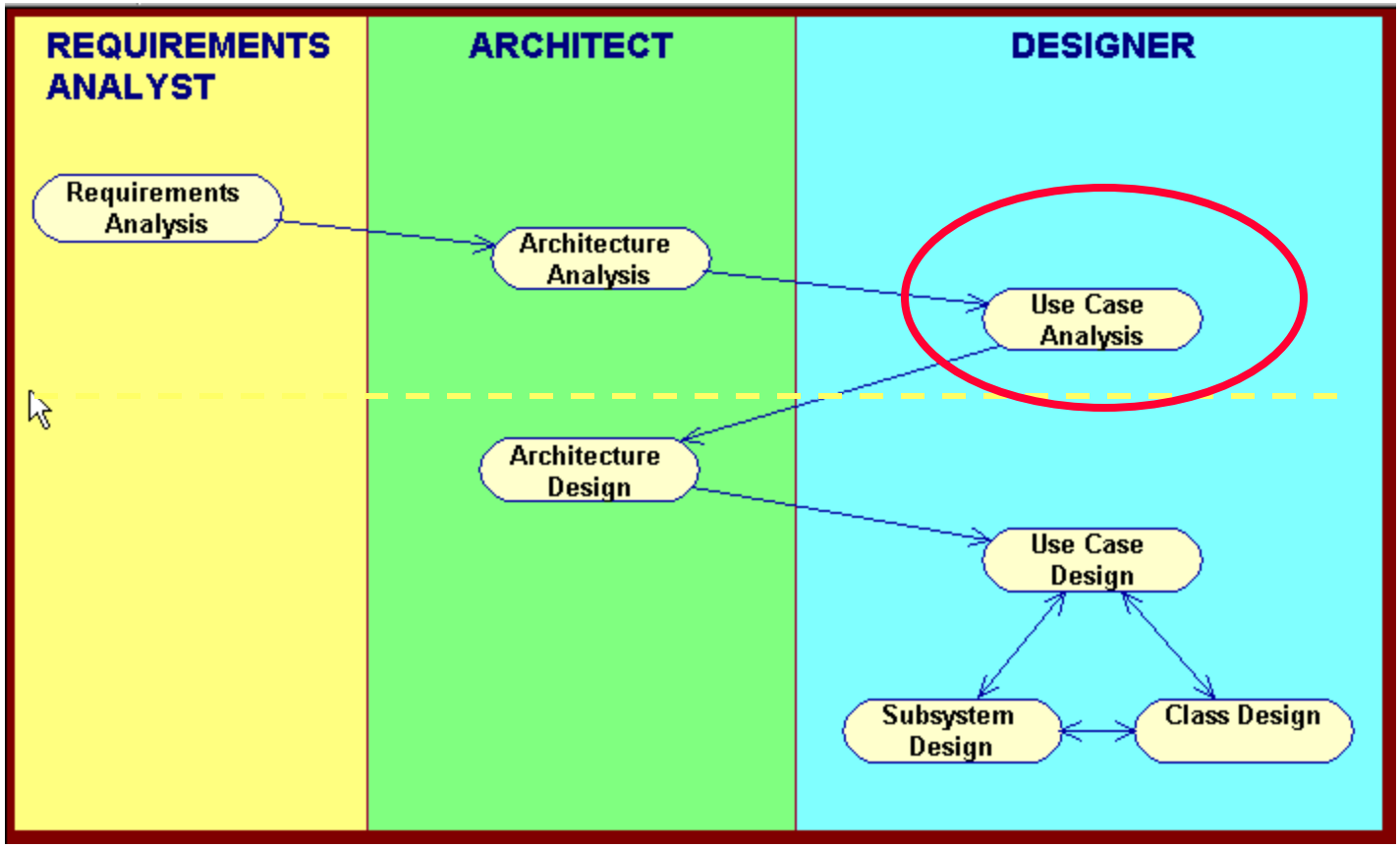
Use Case Analysis

Yoga Is Skill In Action

Basic RUP OOAD Activities



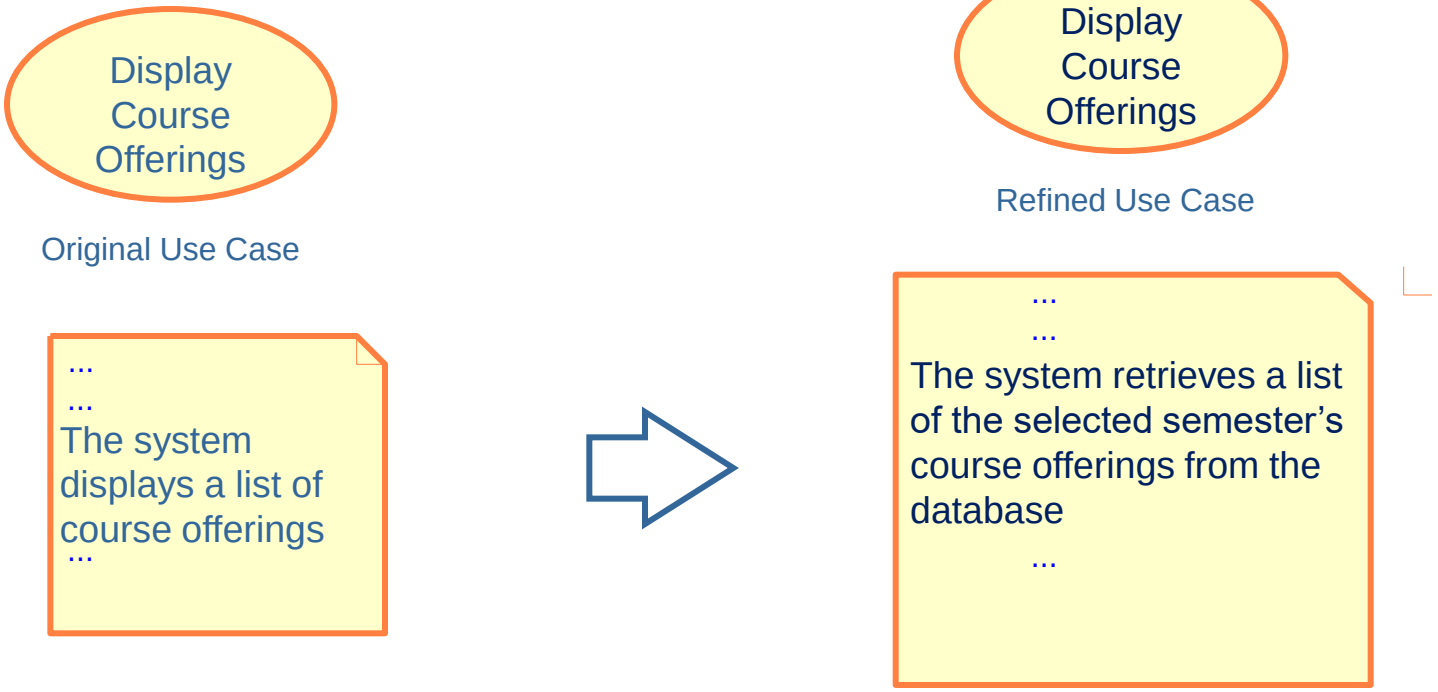
Use Case Analysis Main Points

1. RUP takes the point of view that the designer or architect should do the "obvious" things at each step in the process, and details become apparent in due course.
2. In Use Case Analysis, objects are classified, responsibilities assigned and attributes discovered, all in a relatively effortless manner by following our rules for using boundary, controller, and entity classes.
3. Sequence diagrams are suggested by the UC descriptions; collaboration diagrams can be automatically generated, and a VOPC diagram for a use case basically just "falls out" of the collaboration diagram.
4. This effortless quality shows how RUP follows the characteristics of Nature's functioning. Nature accomplishes its complex behavior in small effortless steps.

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
- Document business rules

Refine Use-Case Description



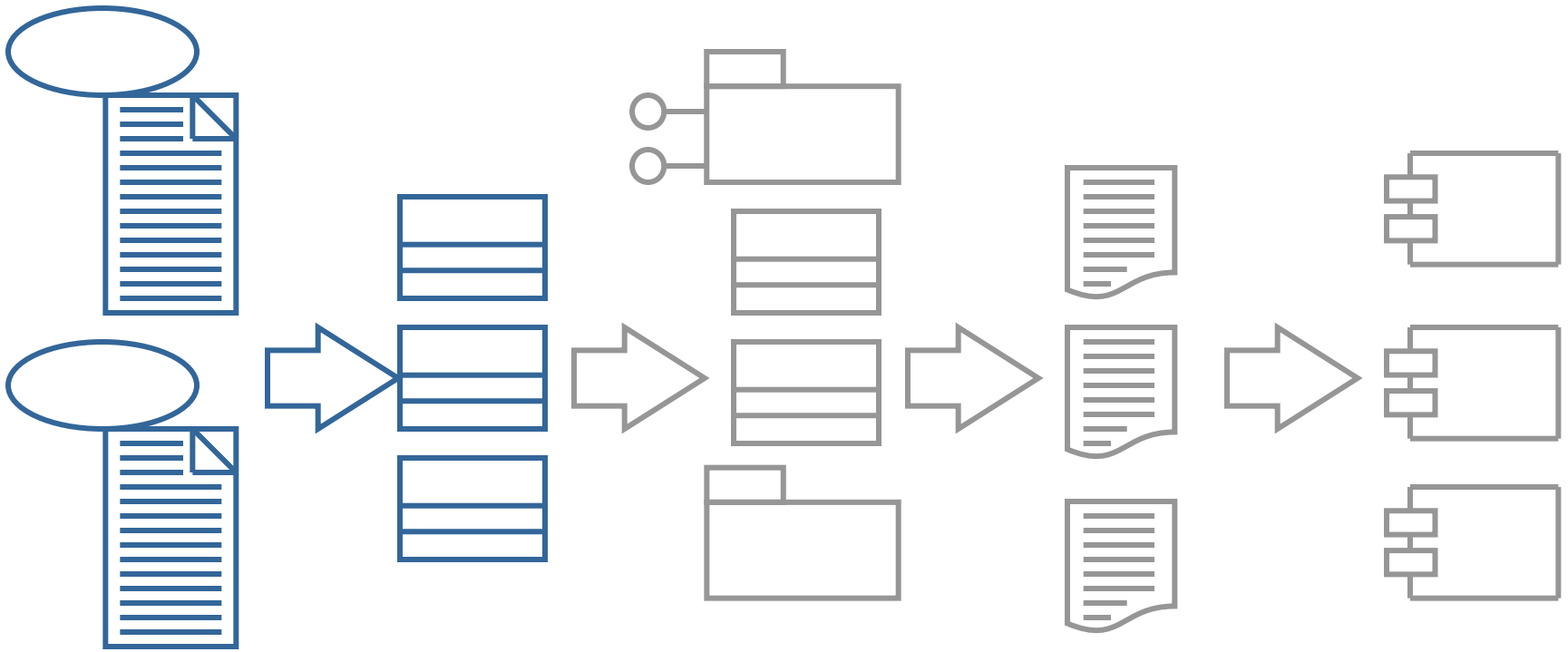
Use-Case Analysis Steps

- 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

Analysis Classes: A First Step Towards Executables

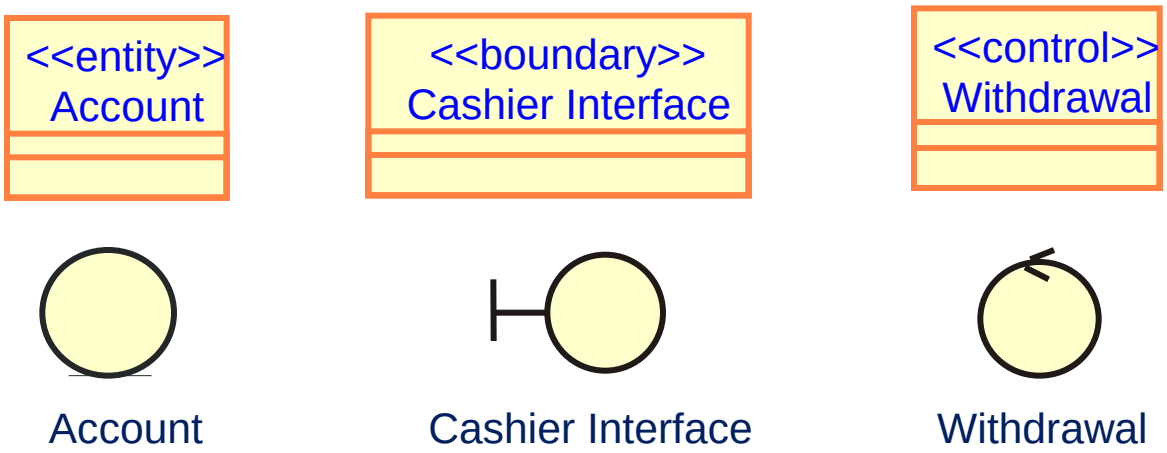


3 Analysis Class "Stereotypes"

System information

System boundary

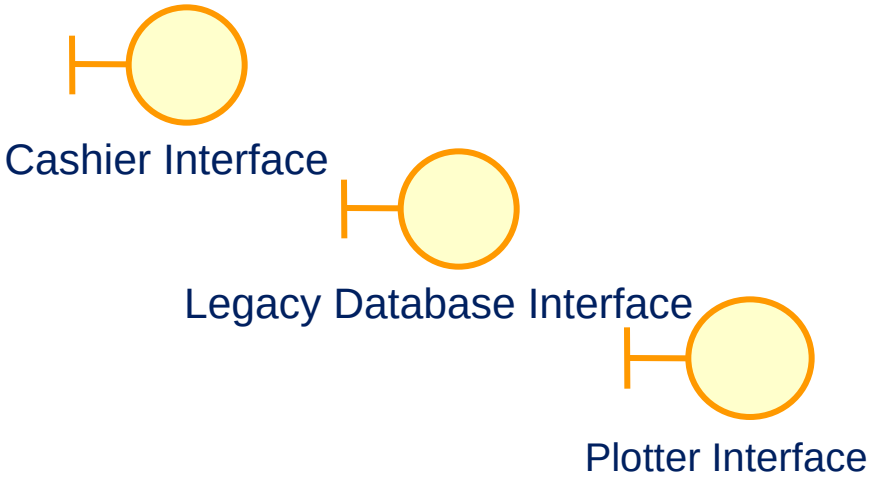
Use-case behavior coordination



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"***

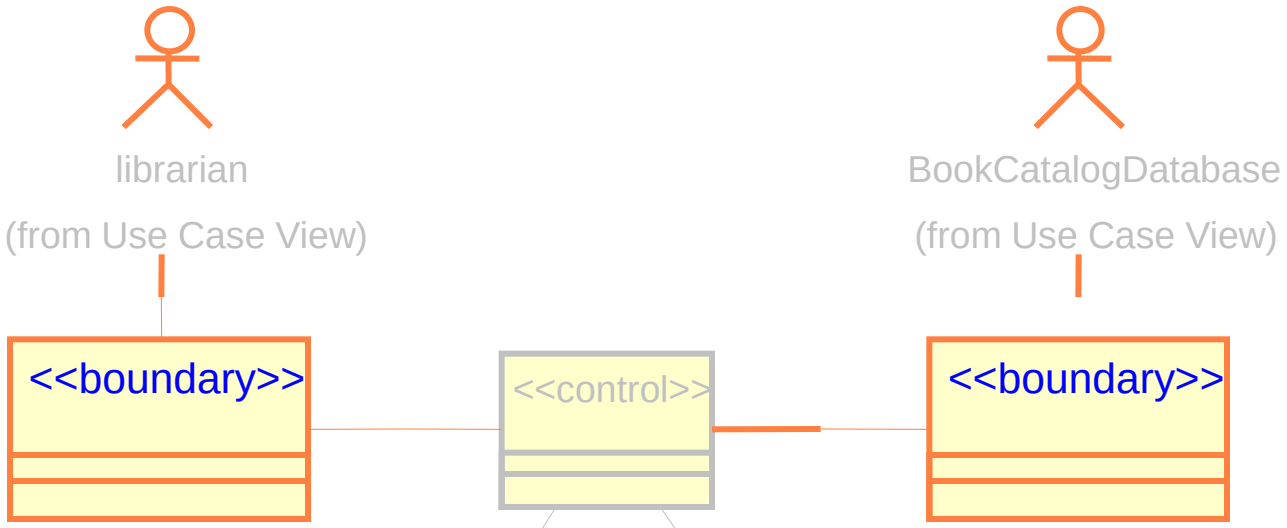
Boundary Classes

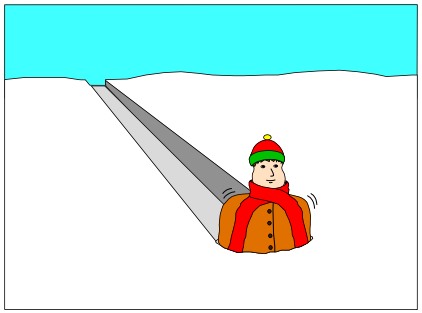
- Capture interaction between system and actors (external systems)
- Help isolate changes to external systems
- Typical examples
  - User interfaces
  - System interfaces
  - Device interfaces



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

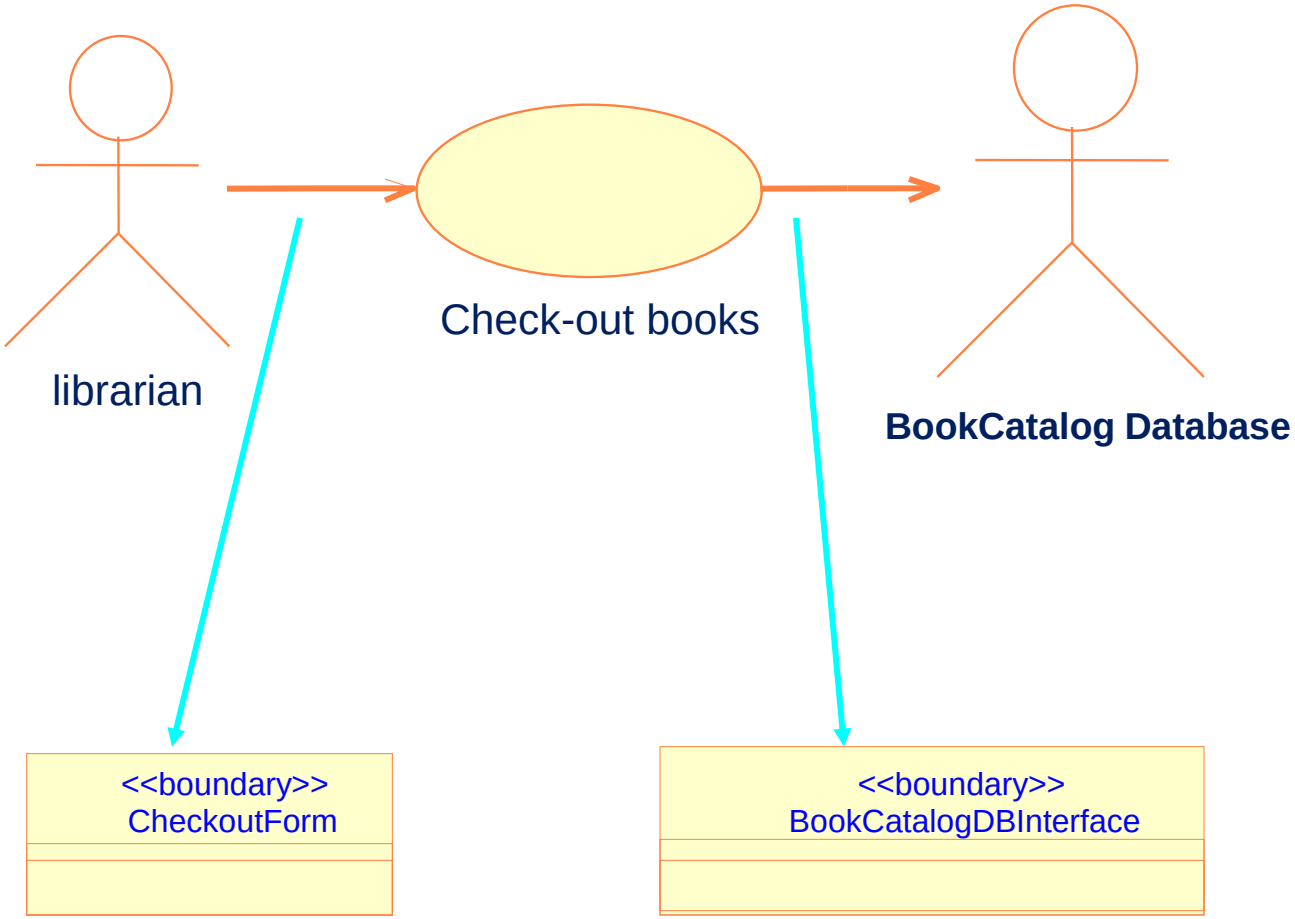
Boundary Classes Isolate System/Actor Interactions





Identify Boundary Classes

- For each actor/use case pair



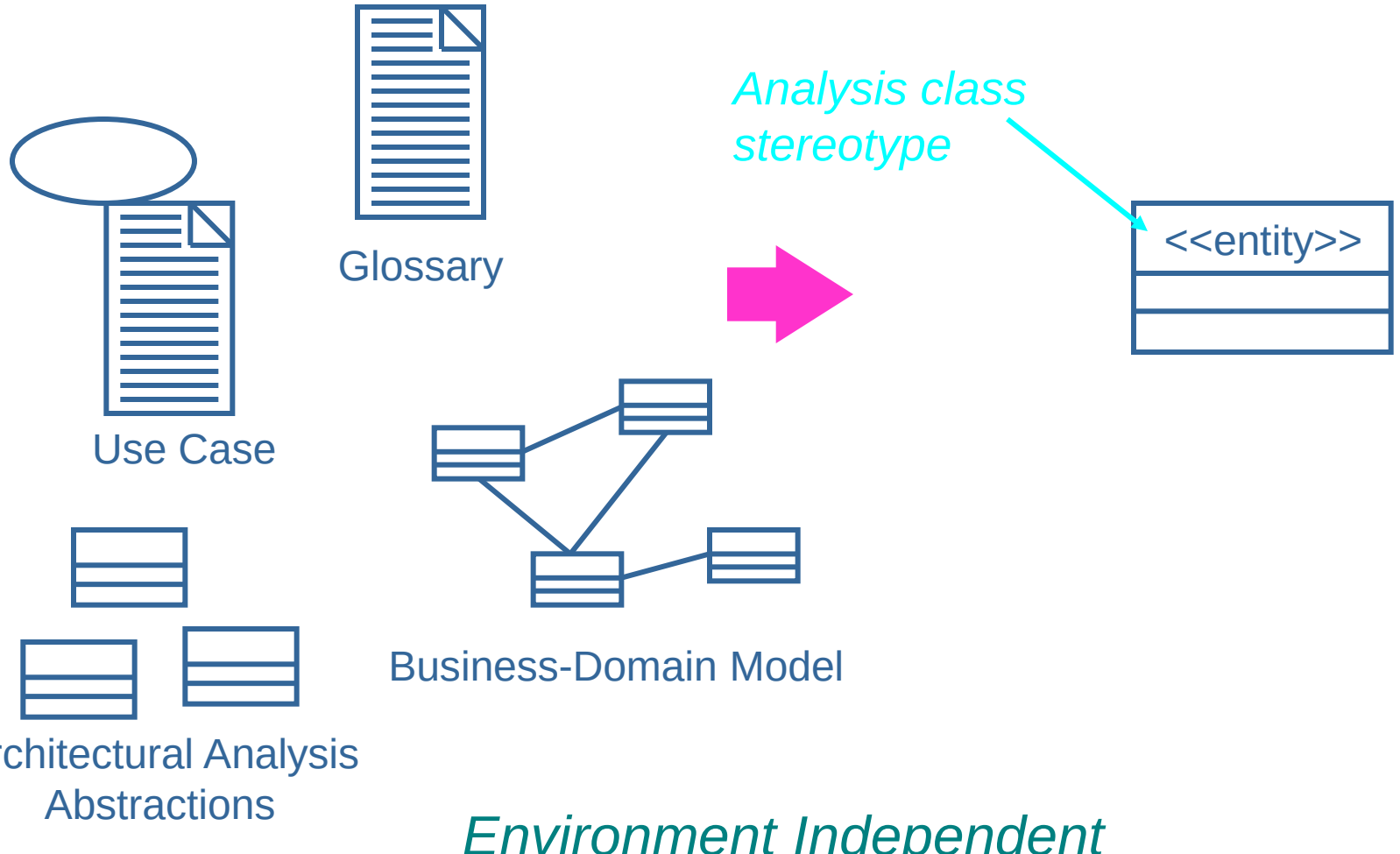
Boundary Class Guidelines

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

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

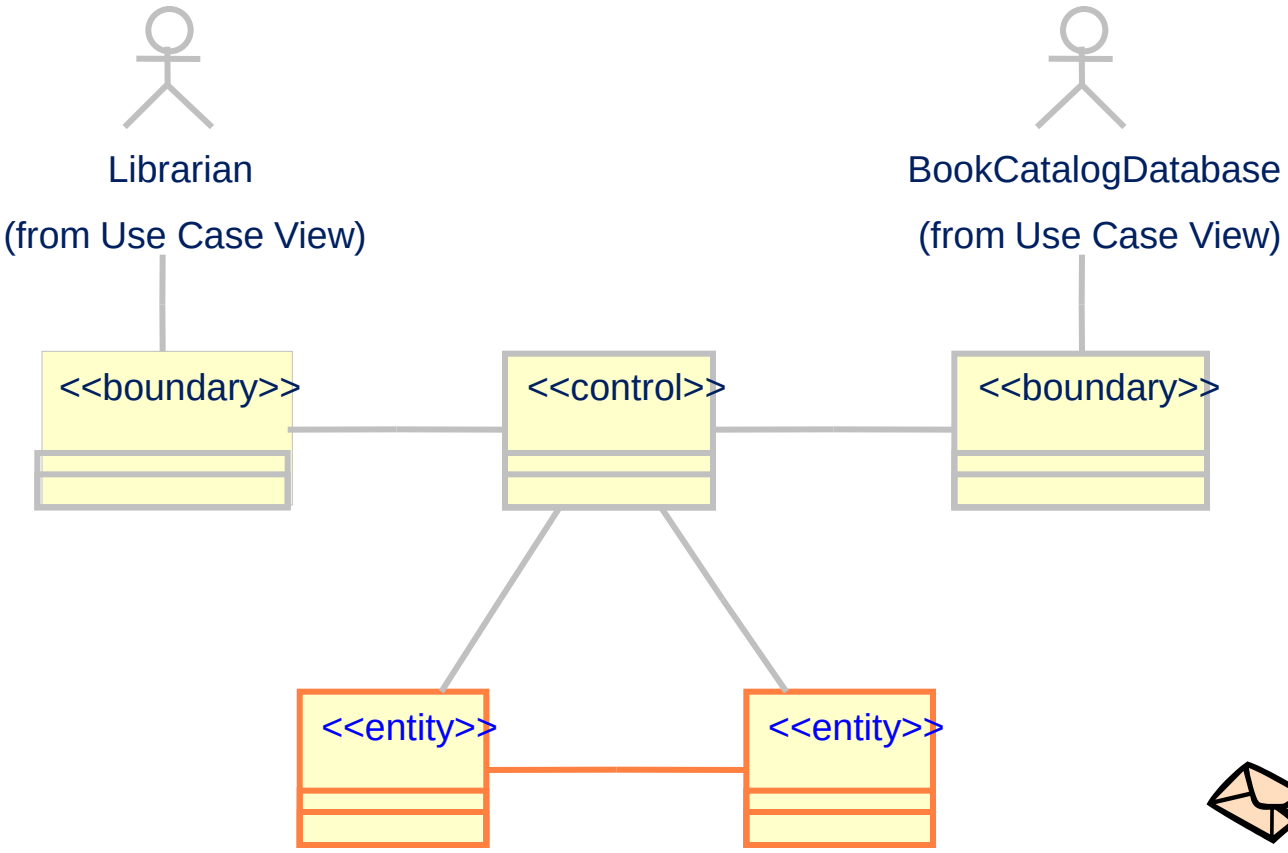
What is an Entity Class?

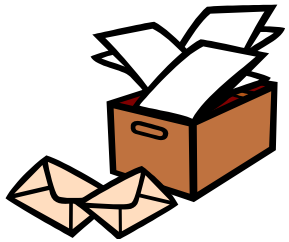
- Key abstractions of the system



Environment Independent

Entity Classes Model Persistent Information

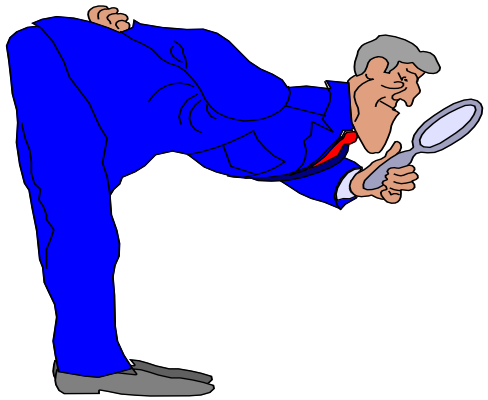




- **Responsibilities: to store and manage information in the system**
- **Hold and update information about events or a real-life object**
- **Usually persistent**

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



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.

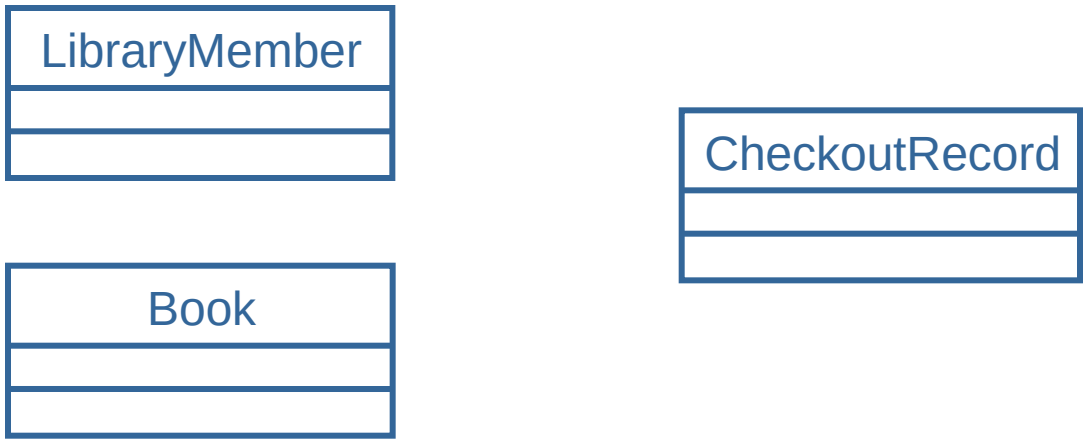
All books in the library have a unique bookid.The books in the library are ordered on the shelves by their bookid. The new system must allow library members to search through the electronic card catalog to find the bookid of the desired book.

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

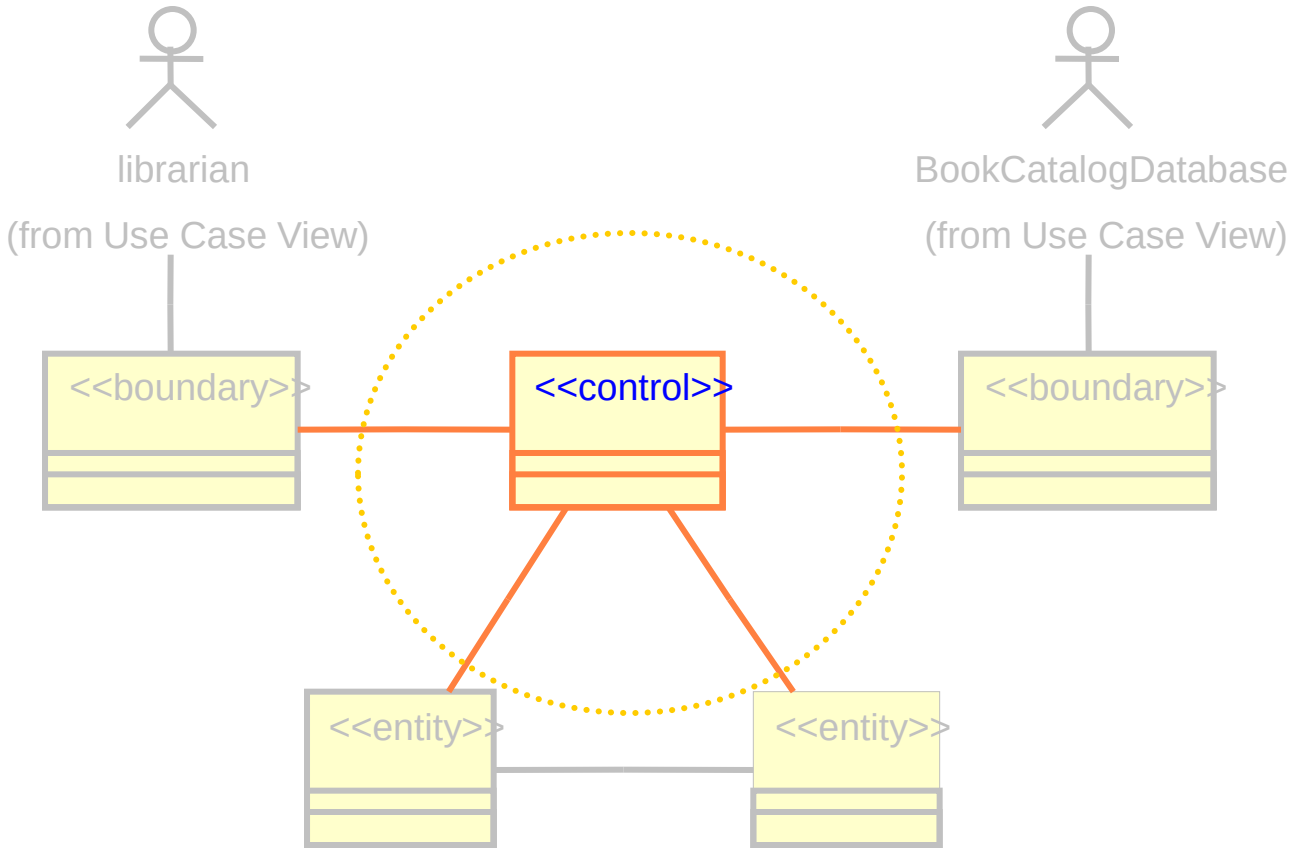


Question: Why is LibraryMember considered an Entity class but not CardCatalog?

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)

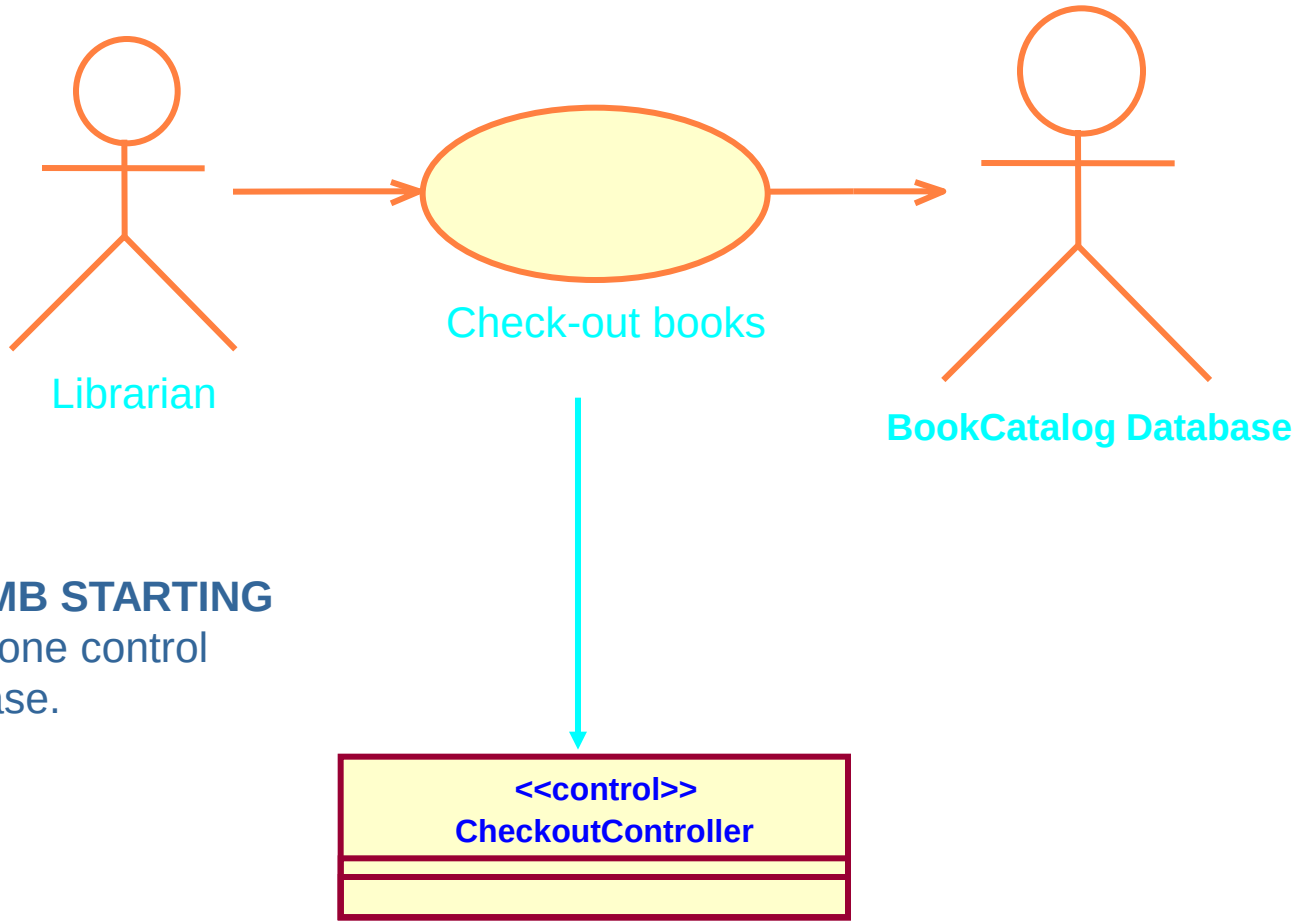
Control Class as Use-Case Coordinator



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

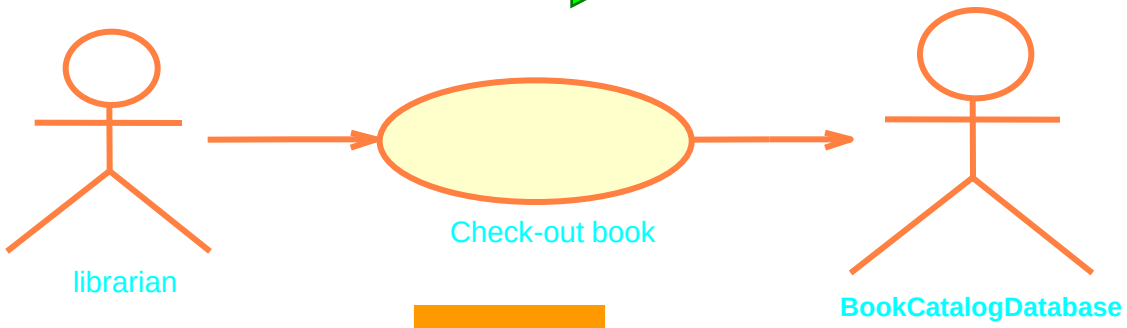
Introduce Control Classes

- For each use case



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

Example: Analysis Classes



<<boundary>> CheckoutForm	<<control>> CheckoutController	<<boundary>> BookCatalogDBInterface

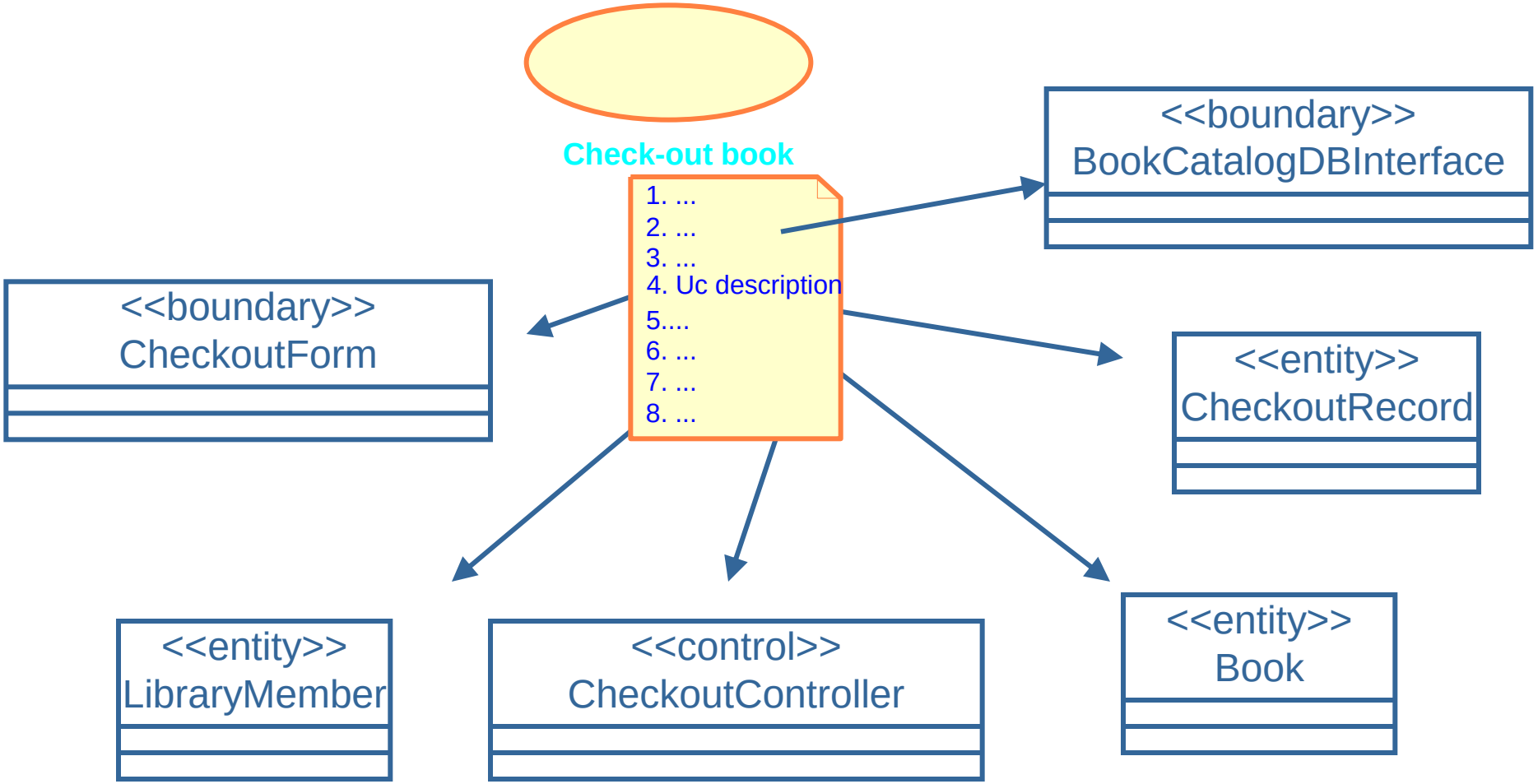
<<entity>> LibraryMember	<<entity>> Book	<<entity>> CheckoutRecord

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

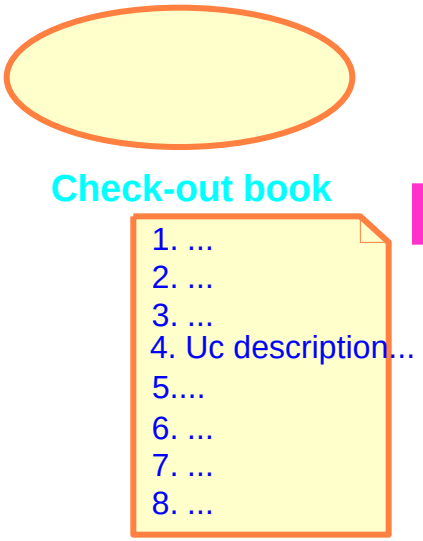
Assign Use-Case Behavior to Classes

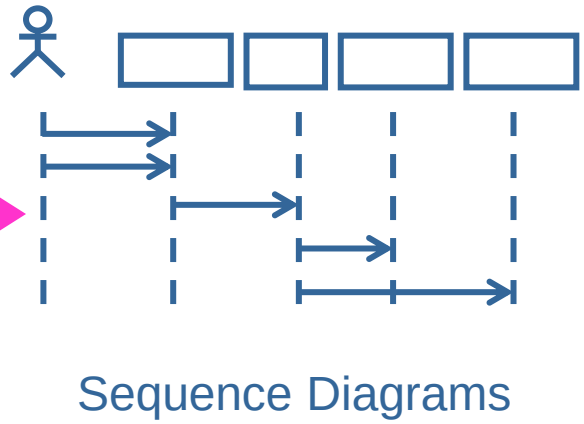
- Assign all use case steps to the analysis classes.



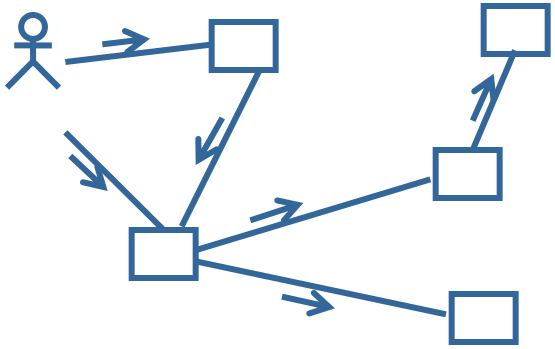
Assign Use-Case Behavior to Classes

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



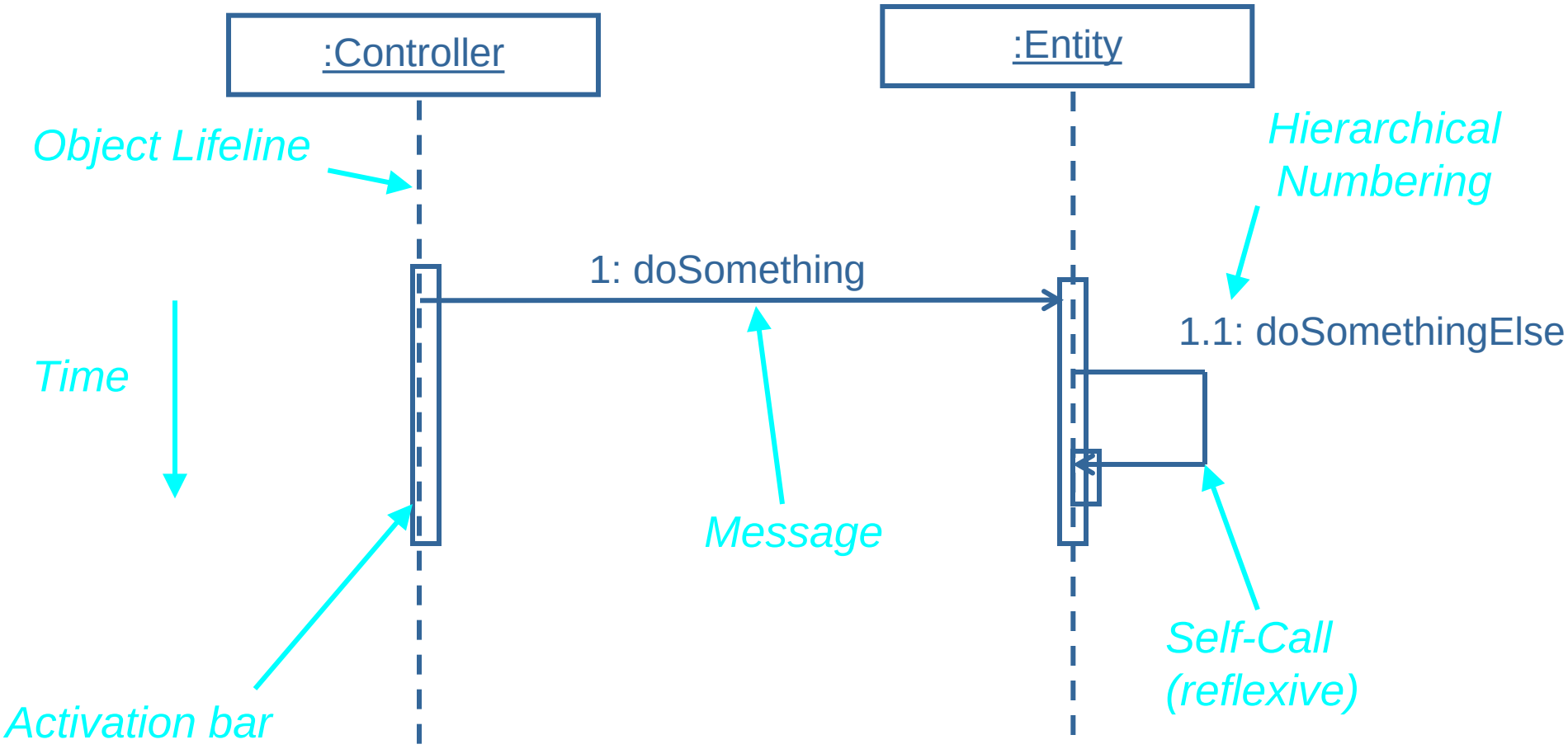


Sequence Diagrams

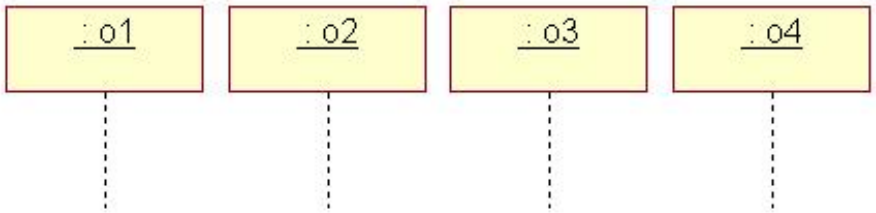


Communication Diagrams

Sequence Diagram Features

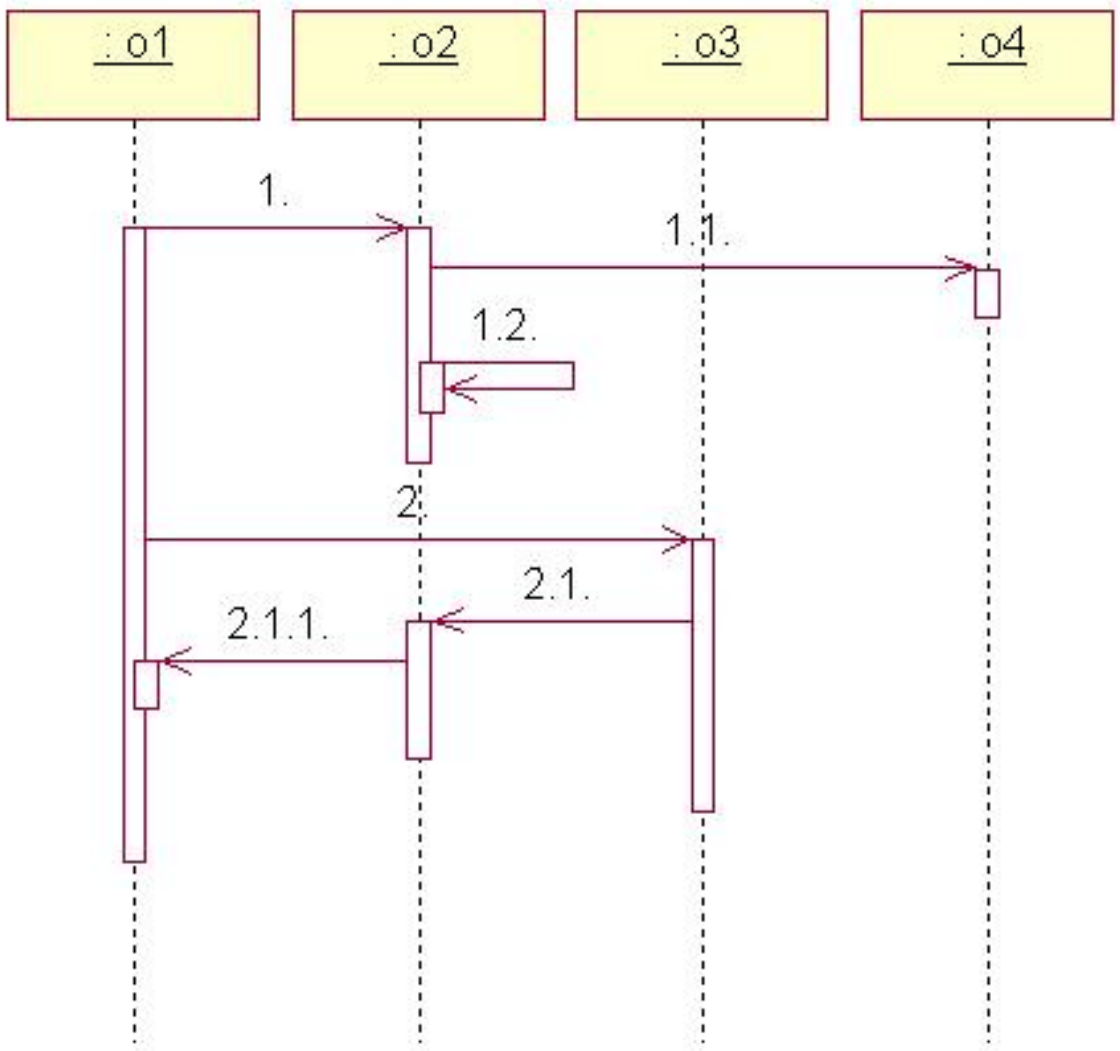


Example

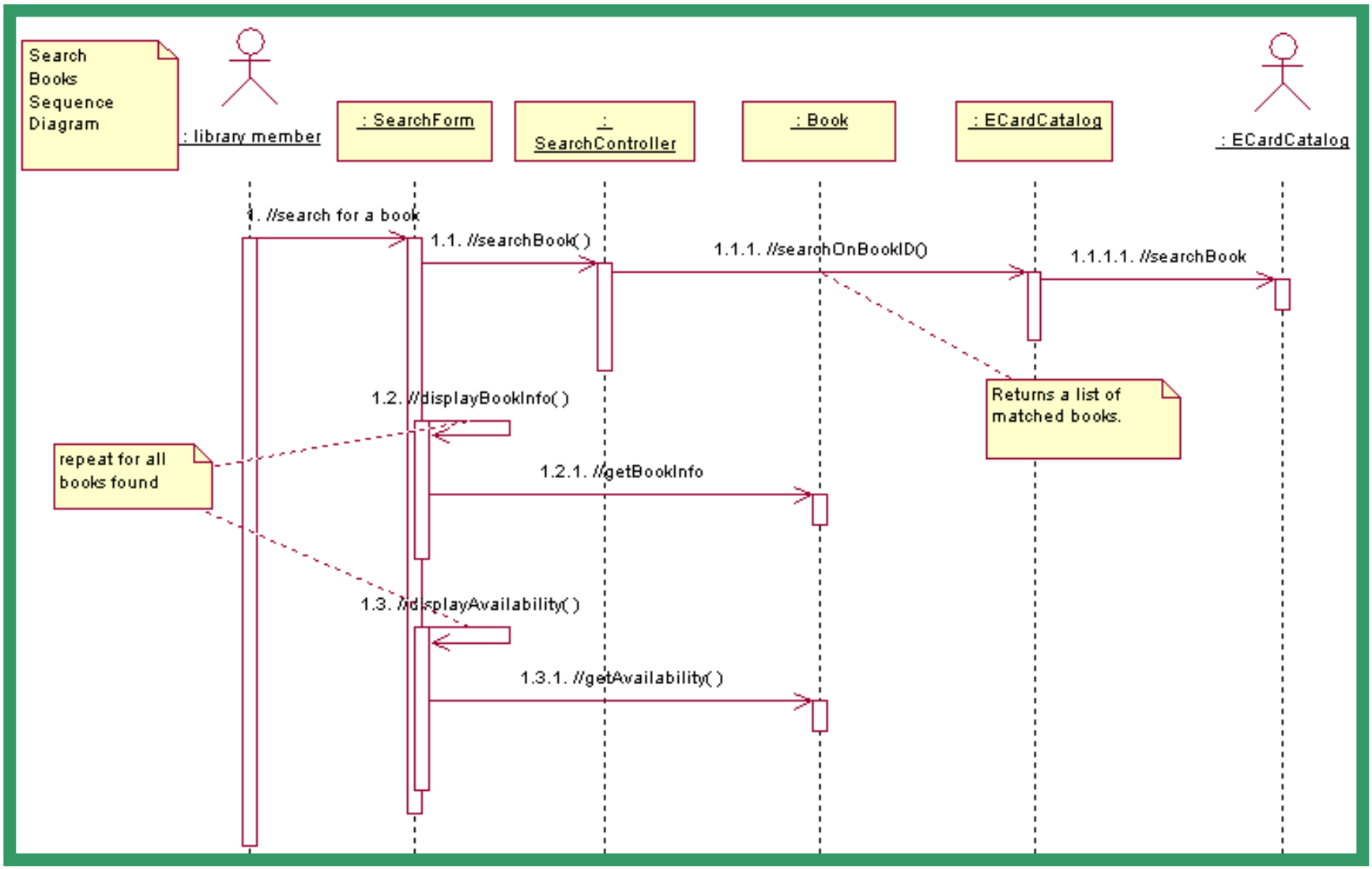


1. o1 sends to o2
1.1 sends to o4
1.2 sends to o2
2. o1 sends to o3
2.1 sends to o2
2.1.1 sends to o1

Example (cont)



Example: Sequence Diagram
(what is missing in this diagram?)



Has the form modifying the entity class directly

- need labels for entity, control, and boundary

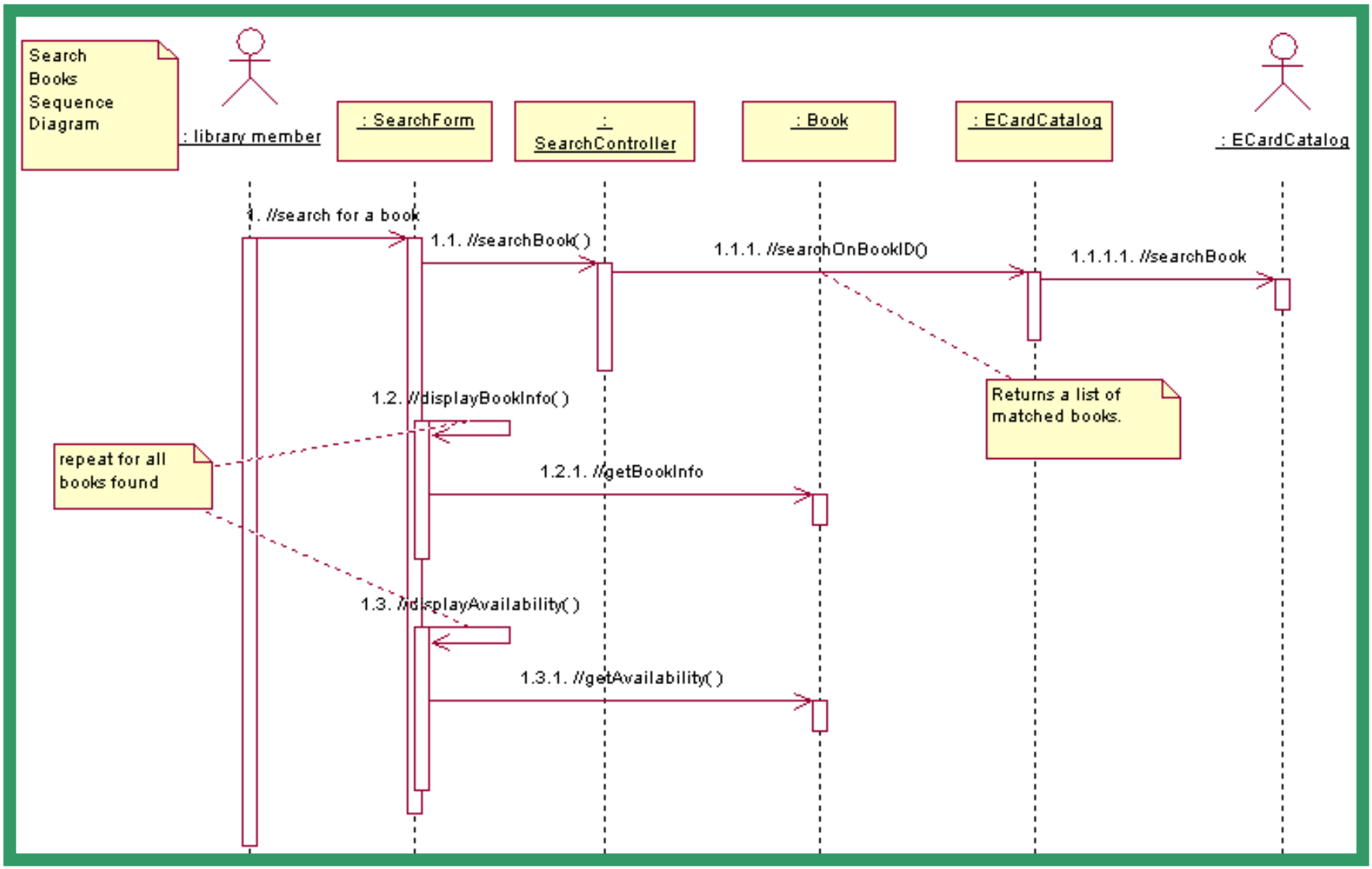


Diagram Scenarios vs Flows vs Use Case?

- A scenario is a single pass
  - Straightforward to create and read
- 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
  - Full coverage infeasible
- Sequence diagrams are most useful to show what **classes** we will need and their **interactions**
- Sequence diagram are ***not*** very good at modeling complex logic flow

Our Sequence Diagram Conventions

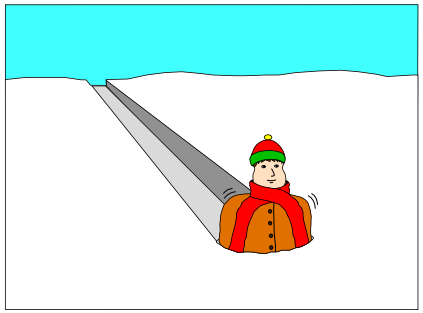
- Label our analysis classes with <<Boundary>>, <<Control>>, or <<Entity>>
- We show communication with our Data Base thru a data base boundary class. ORM is part of our OO Analysis and Design.
  - If we are ***adding*** an entity class to the DB we show the controller "new" of the entity class and then later the DB boundary class will show the ***create*** to the DB.
  - If we are ***updating***, ***reading***, or ***deleting*** an entity class that already exists in the DB, we show ***first*** the controller reading the entity class from the DB.
  - We show the return of the entity class with a comment.
  - We show the entity class is active when an object (will usually be the controller) is getting or setting data in the entity class.
  - We show the **update** or ***delete*** operations from the DB boundary class to the DB at the time the save or delete occurs.

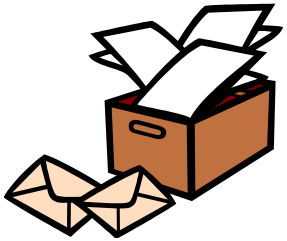
Group Exercise for MUMSched Sequence Diagram

- As a group create/review one of your Use Case Descriptions.
- As a group create a ***first draft*** sequence diagram for *that use case.*
- Identify boundary, control and entity classes.
- Show the basic success flow.
- Any preconditions for this use case?
- How would you show your alternative paths and business rules?

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

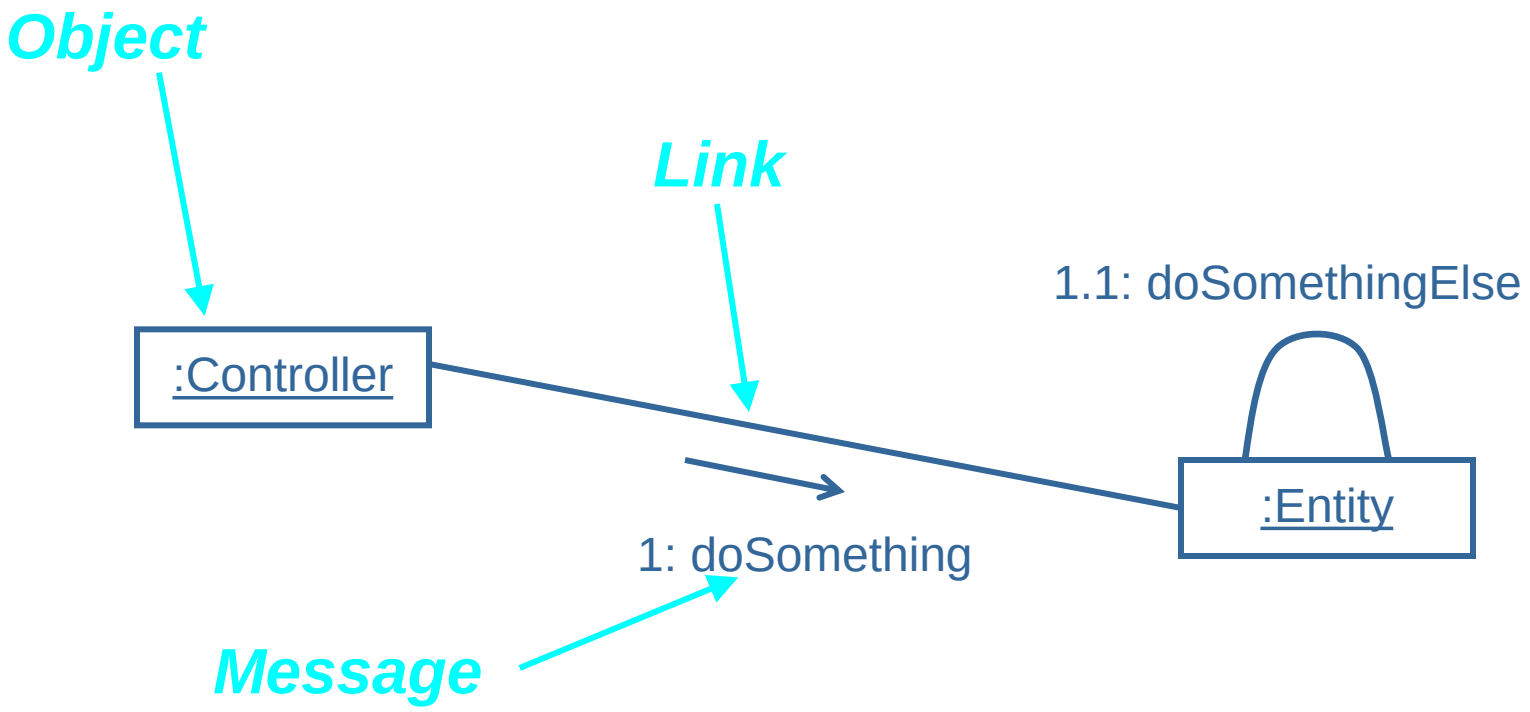




Further Responsibility Assignment Guidelines

- In OO we want to couple the class logic with the class data .
- We favor ***entity*** classes having operations on *their own data.*
- Entity classes in an enterprise architecture will normally have only getters and setters.
- If data is spread across classes in an enterprise architecture a control class will access the entity classes and do the necessary processing.

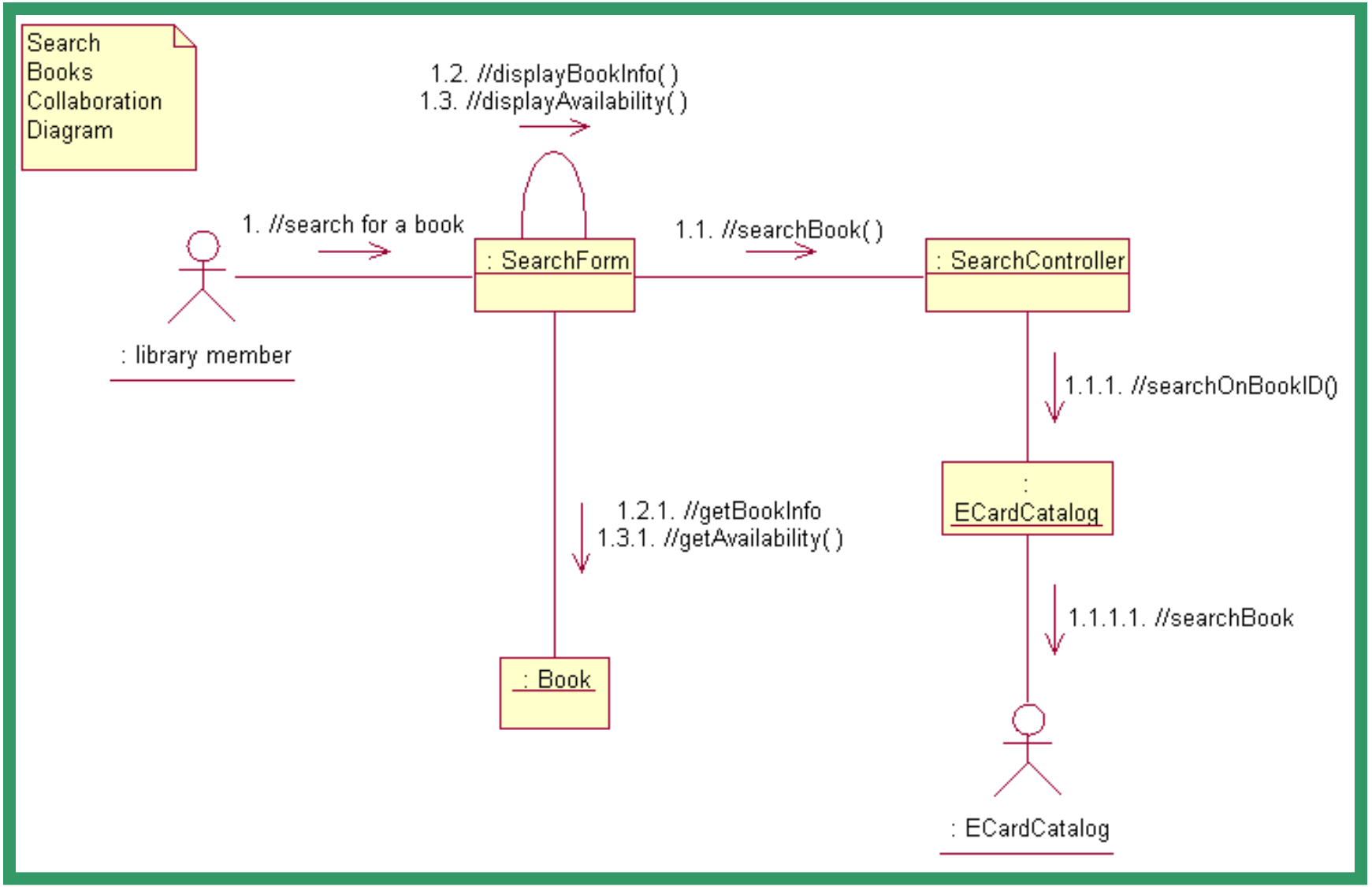
Collaboration Diagrams



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



Sequence vs Collaboration Diagrams

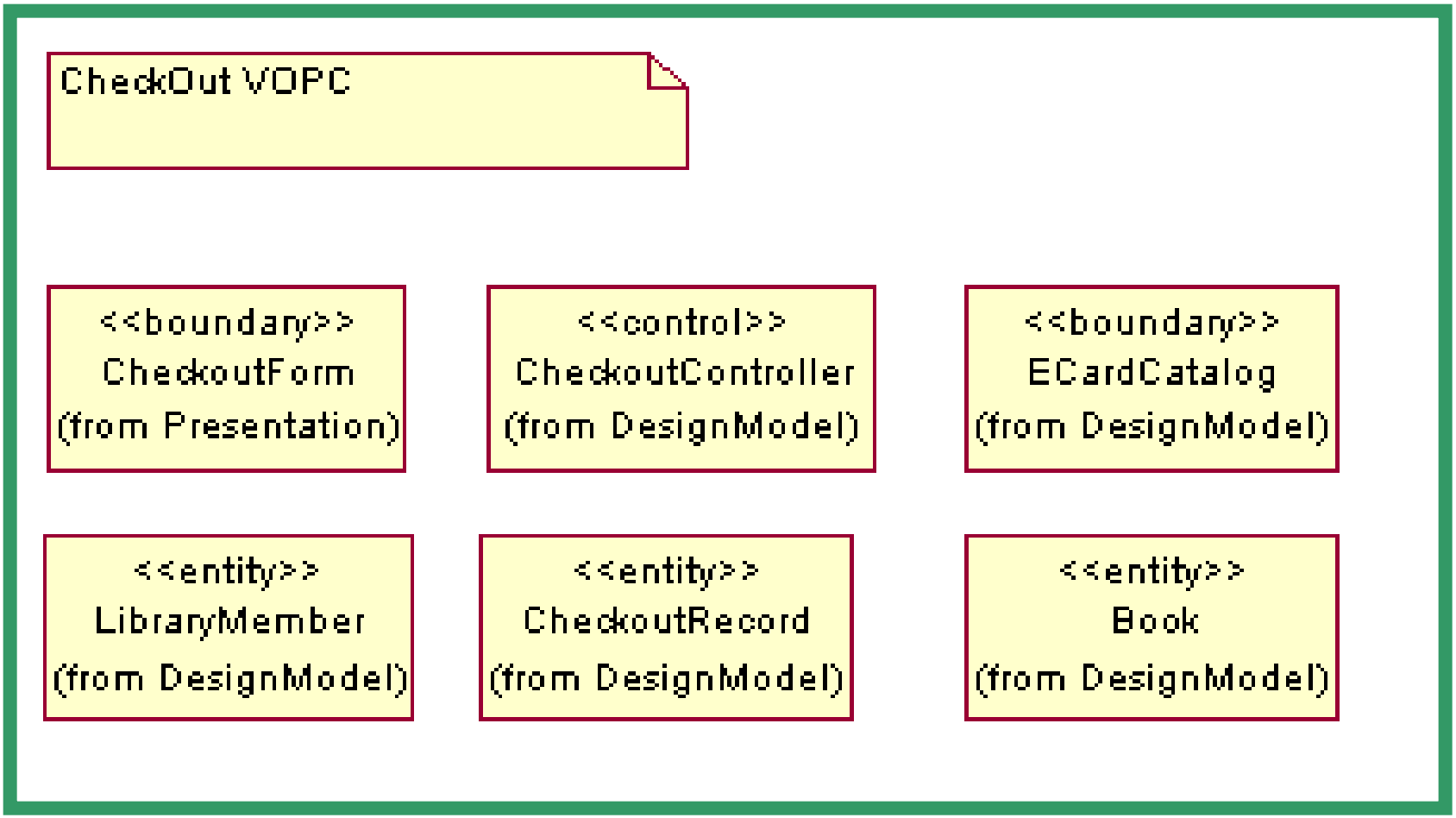
- Sequence Diagrams
  - Show the explicit sequence of messages
  - Better for visualizing overall flow
  - Better for real-time specifications
- Collaboration Diagrams
  - Show relationships in addition to interactions (use for VOPC diagrams)
  - Better for visualizing patterns of collaboration
  - Better for visualizing all of the effects on a given object
  - **Notice it is easier to see the MVC pattern with the Collaboration Diagram**

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.



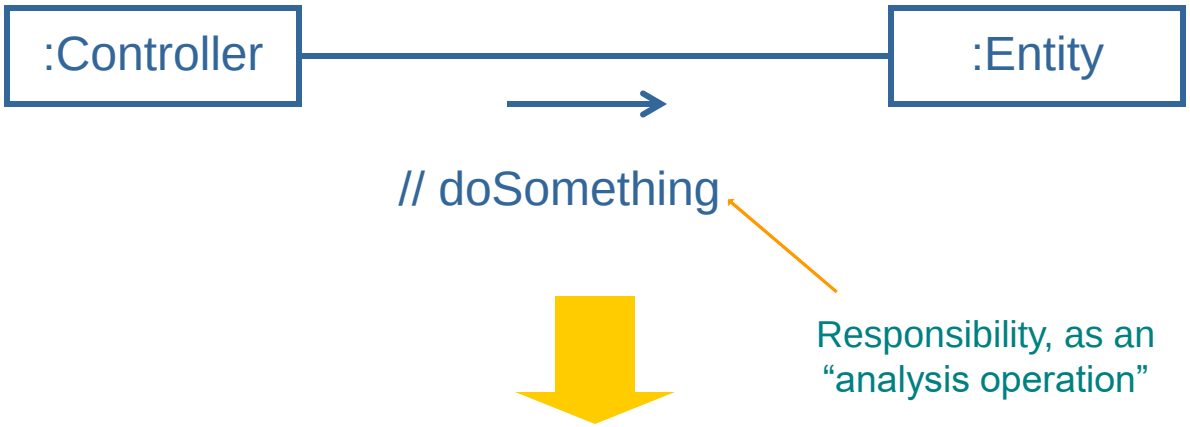
Use-Case Analysis Steps

- 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

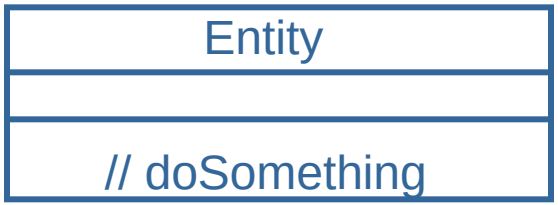
Responsibilities from Collaboration Diagram

- Responsibilities are specifications of object behavior

Collaboration Diagram



Class Diagram



Guidelines for (re)Structuring Responsibilities and Classes

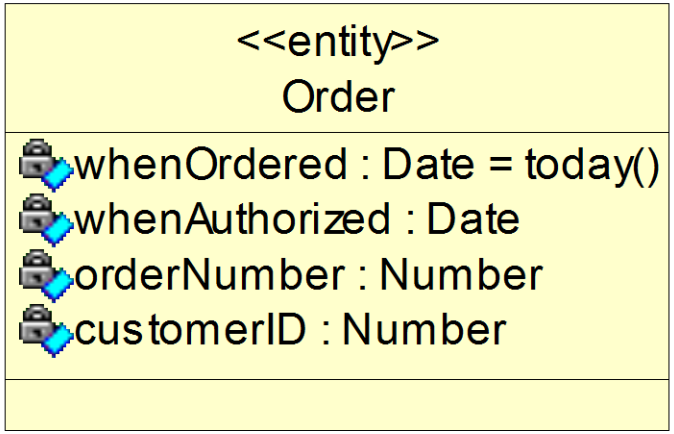
- "Orthogonal" responsibilities within classes
  - ➔separate (e.g. //parseMessage and //displayMessage)
- Redundant responsibilities across classes
  - ➔Integrate (e.g. two classes determining look of gui)
- Each analysis class should have several compatible responsibilities. A class with only one responsibility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

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

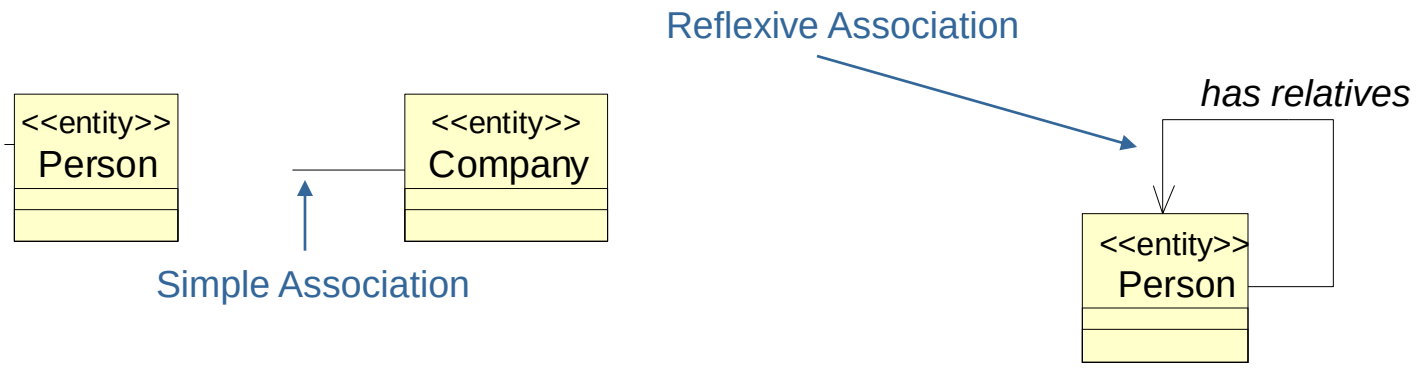


Where To Find Attributes

- 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

Review of Associations

- An association models a structural relationship between objects

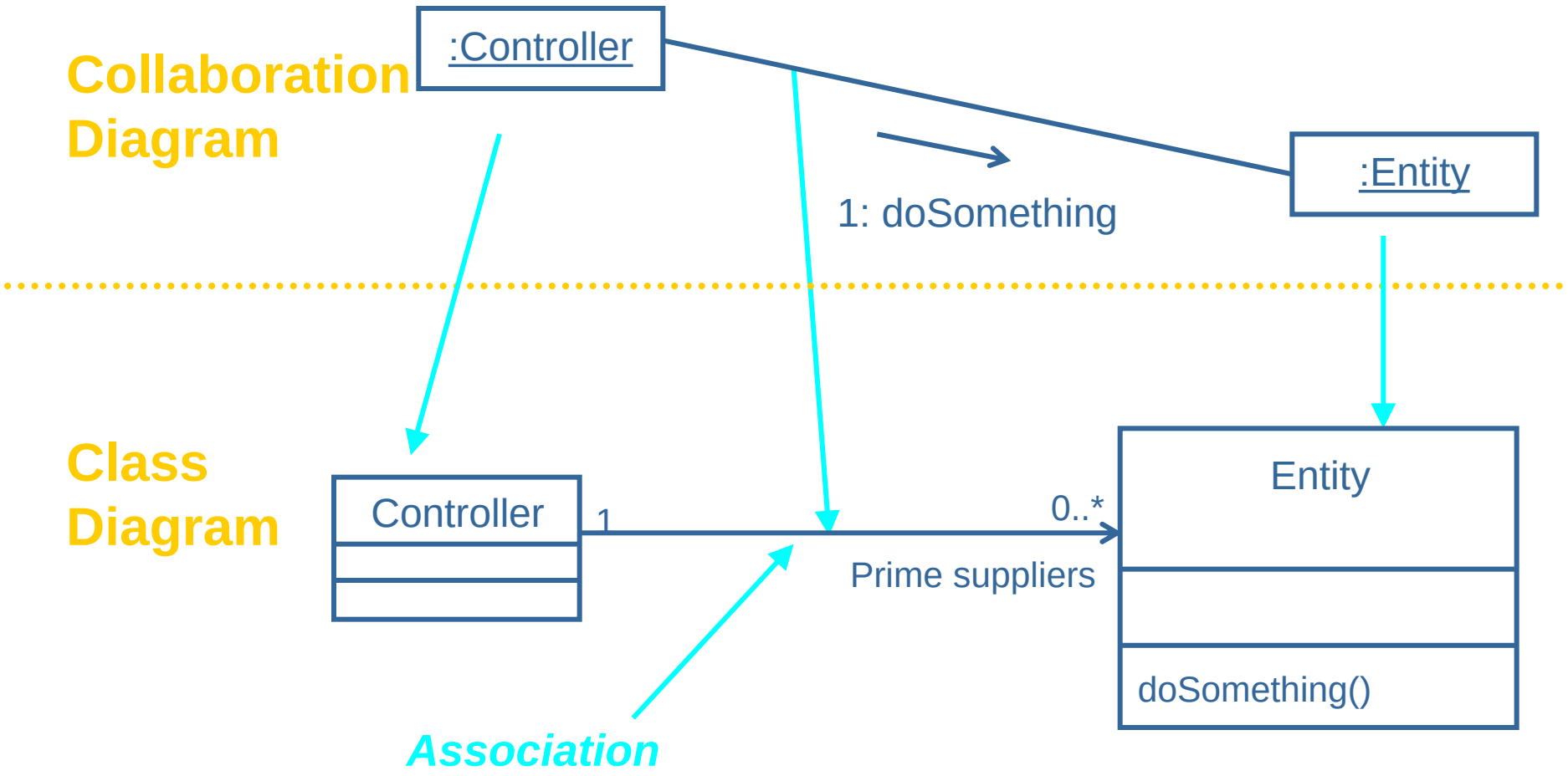


Review of Association Adornments

- Name
- Role
- Multiplicity
- Aggregation

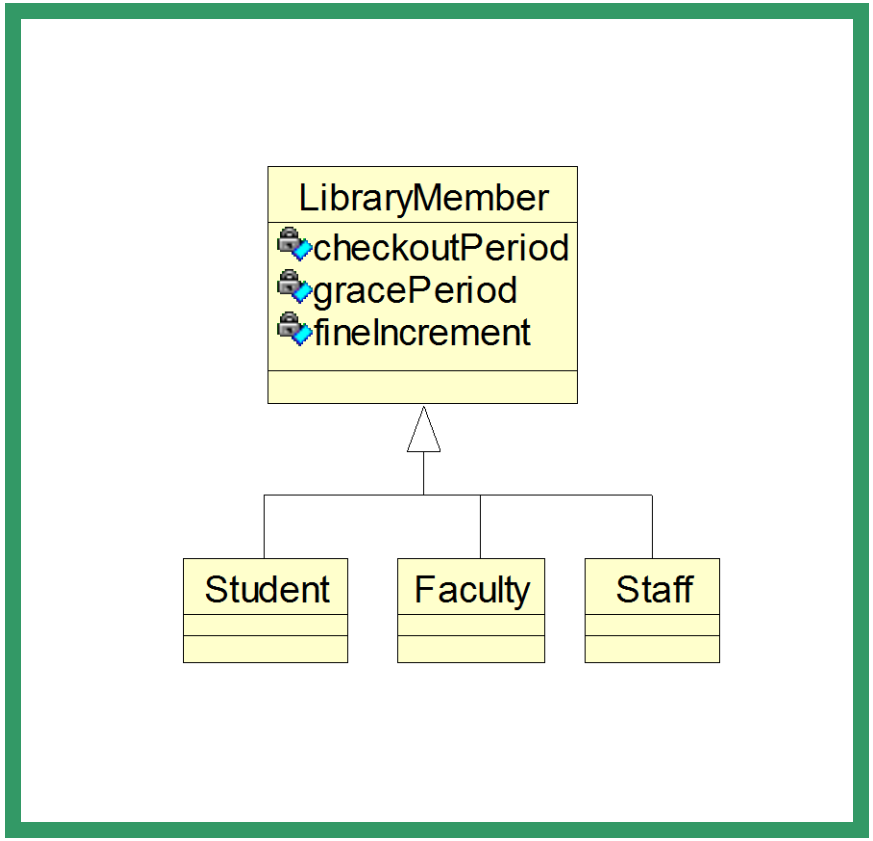
Relationships from Collaboration Diagram

- Create a relationship for each link



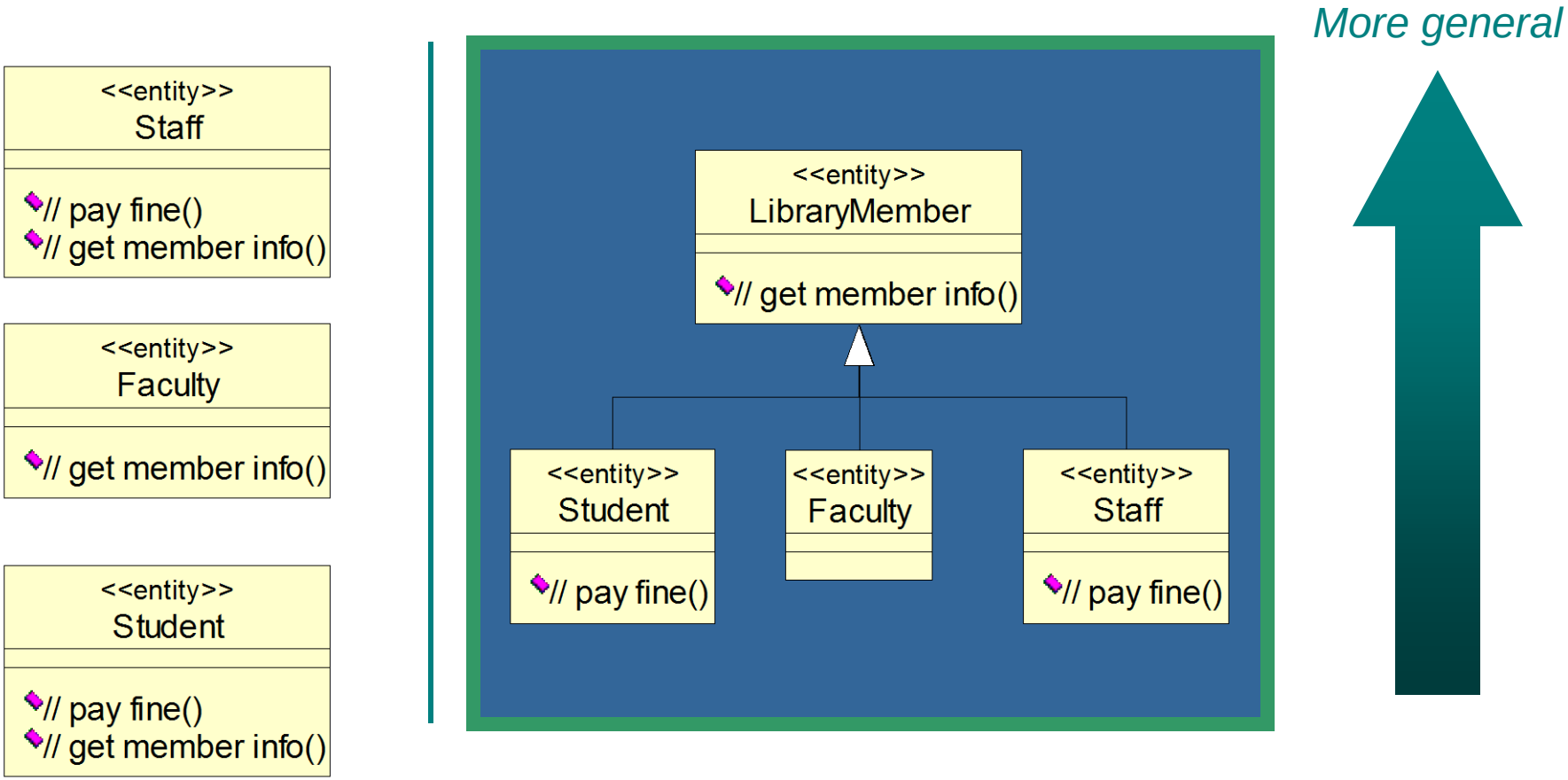
Review of Generalization

- 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
- This is the only reason for our VOPC and our Sequence Diagram to have different classes.
- Sequence diagram can use the **superclass** or the **subclasses** as needed.



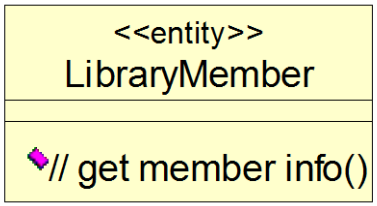
Finding Generalization: Generalization of Classes

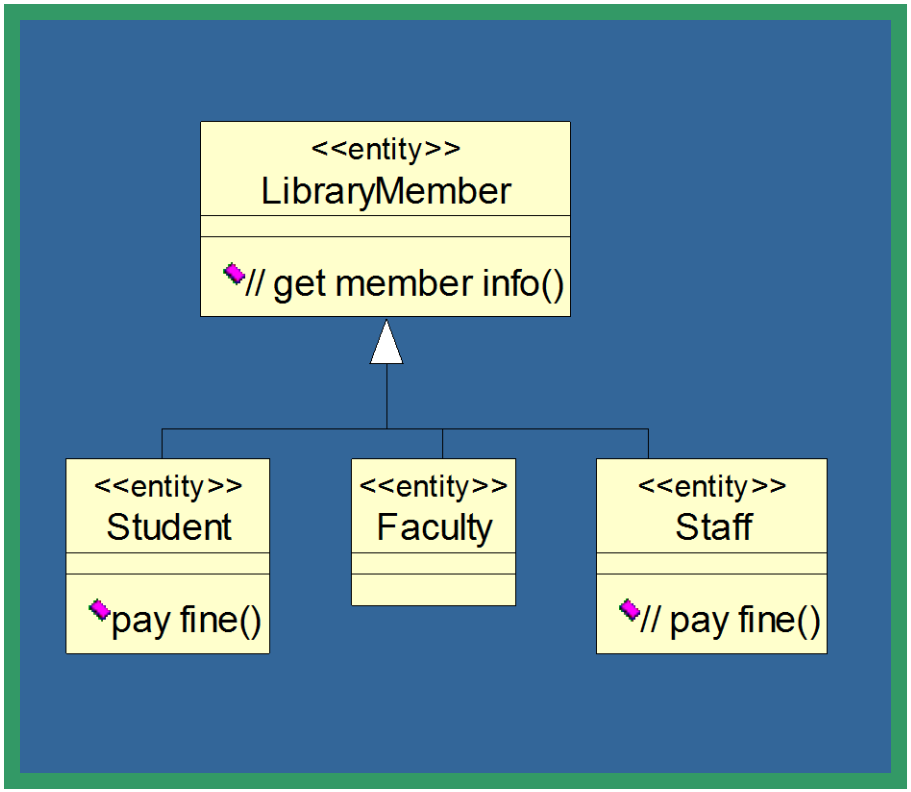
- Create superclasses which encapsulate structure and common behavior of several classes.



Finding Generalization: Specialization of Classes

- Create subclasses that add refinements of the superclass





More specific

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
- Verify business rules are covered
  - Control classes usually check the business rules

Use Case Analysis steps

1. Verify that all your business rules are reflected in your use case diagrams. Our general rules are:
   A. If a business rule is simple (like checking if a User Story name is unique) we will model it with a simple if/else conditional action on the control class active line.
   B. If a business rule requires multiple conditional checks it is better to show an alternative flow in a separate sequence diagram.

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?

Use Case Analysis steps

1. Supplement the Use Case Descriptions
2. For each use case realization, find classes and distribute use case behavior to classes
3. Model the analysis classes with
   A. Sequence diagrams to show the flow of the application
   B. Collaboration diagrams to suggest relationships
   C. VOPC diagrams to specify static relationships and to elaborate attributes of each class
4. Verify that all your business rules are reflected in your use case diagrams. Your diagrams should show:
   A. Business rules are checked/verified
   B. An alternative path is noted for Business Rule violation.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

1. During analysis we identify classes and their relationships at a conceptual level. We follow some basic rules to construct use case diagrams in a simple and effective way.
2. Use case analysis is the first step in connecting end user goals (use cases) to their realization in code.

3. **Transcendental consciousness** is the source of natural law, intelligence, and all possibilities.

4. **Impulses within the transcendental field:** Action at the subtlest level of the Unified Field is most in tune with natural law and the most effective.

5. **Wholeness moving within itself:** In unity consciousness, all steps of activity are spontaneously appreciated in terms of the simplest, most direct path to success and fulfillment.