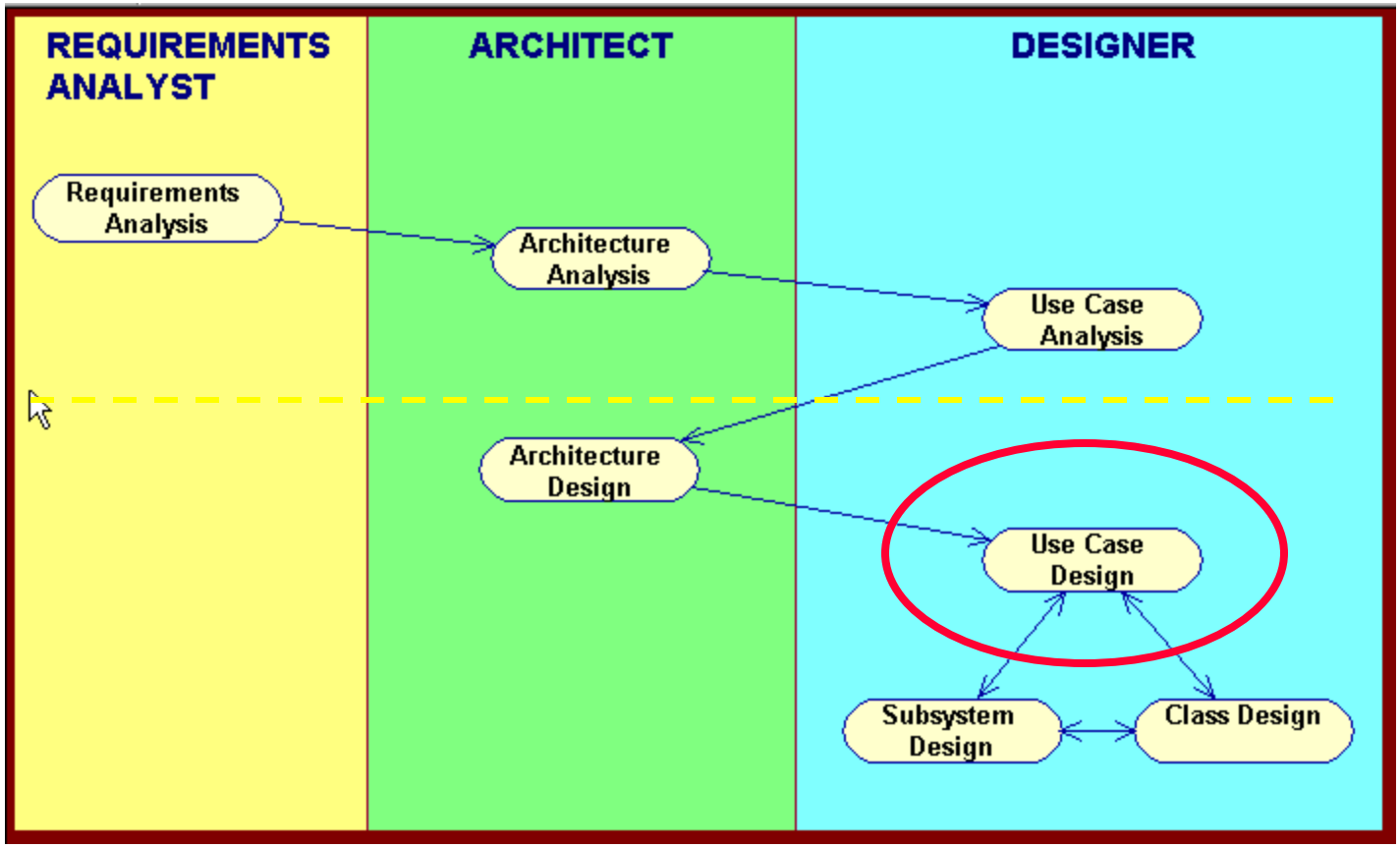
Use-Case Design

Fine Focus Of Attention

Basic RUP OOAD Activities



Architecture Design Outputs

- Each step of Use Case design is done by designers for their Use Cases based on the outputs from Architecture Design.
- Recall our Arch Design outputs are:

Architecture Design Outputs

- Arch Design outputs:

1. *Design Classes* -we refined our analysis classes
2. *Reuse opportunities* - we agreed on any inheritance in the design
3. *Design Mechanisms* – we chose our Enterprise architecture and frameworks
4. *Subsystem Context Diagram* - we now know our subsystem interfaces and we know the clients using the subsystems.

Use Case Design Steps

- All designers/developers will complete the following steps for their VOPC and Sequence diagrams:

1. Change analysis classes to design classes and refine class interactions.
2. Incorporate any inheritance hierarchy
3. Incorporate our Enterprise Framework
4. Incorporate any subsystems needed

Step 1: Incorporating Design Classes

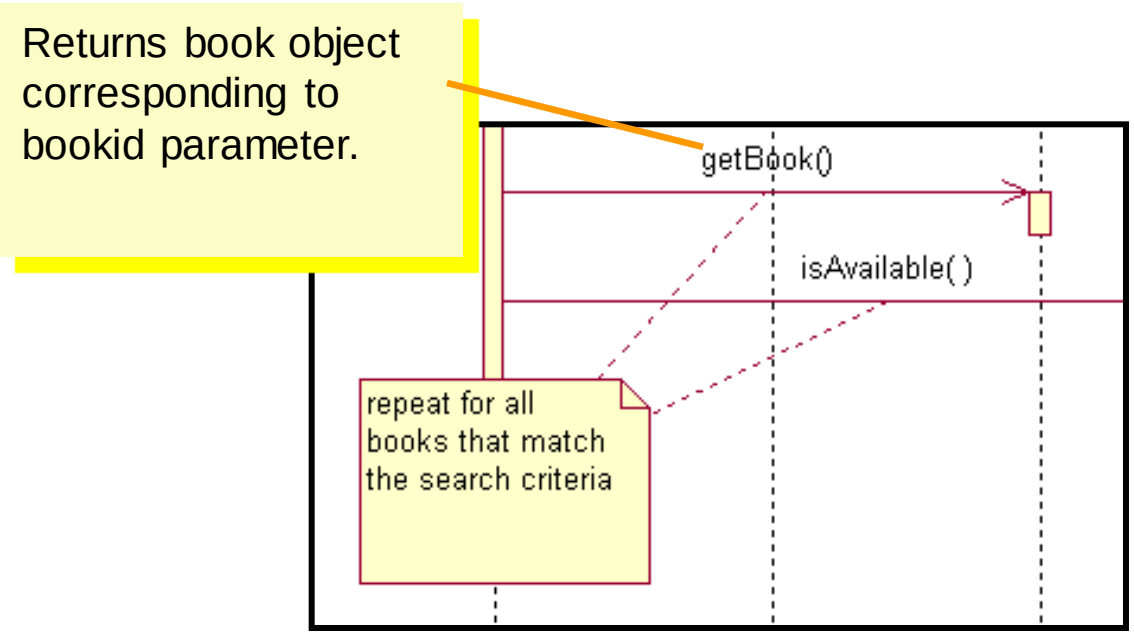
- We started with analysis classes in Use Case analysis. Now we change those to Design classes.
- In our VOPCs and Sequence diagrams we put in more detail on methods, including known parameters.
- We may add in more detail on returns from method calls

Step 1: Incorporating Design Classes

- Refine your analysis sequence diagram(s) for the use-case by going through the flow one step at a time and determining if you need to update the design element/actor interactions.
- For each of your use cases review the business rules and make sure you have a design that matches your requirements.

Step 1: Incorporating Design Classes

- You might add a description of what an object does when it receives a message.
- Attach note or comment to operation



Step 2: Incorporating Design Hierarchy

- If you decided to use inheritance for any of your key design classes now is the time to add those to your VOPC and Sequence diagrams.

Step 1 and Step 2: Main Point

Design classes replace their corresponding analysis classes in the use case realizations. There could be class refinements and incorporating inheritance.

As designers we will:

- update messages on interaction diagrams
- update relationships on VOPC diagrams.

These activities may lead to further refinement of subsystem responsibilities (that is, feedback to our architecture design.)

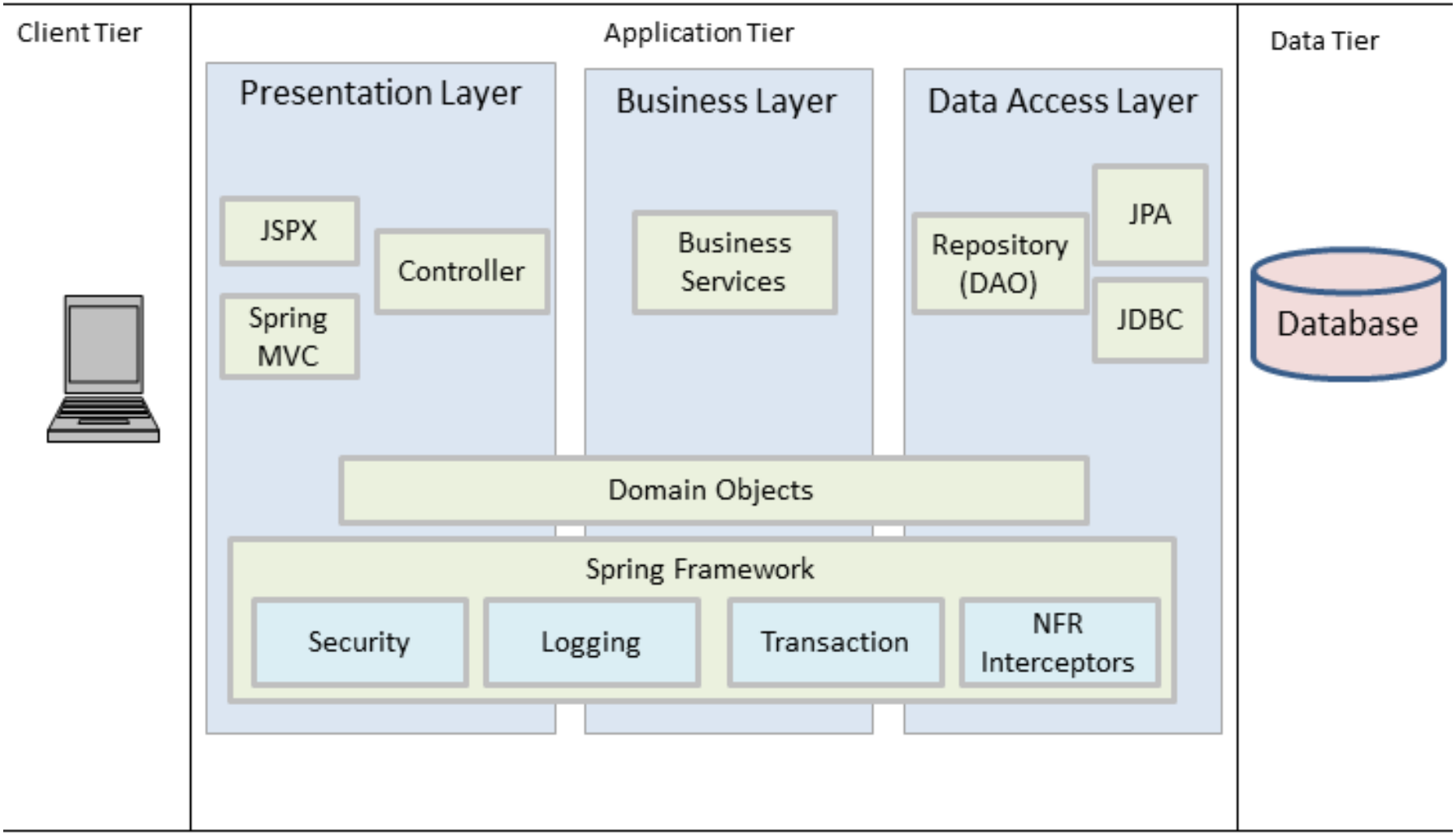
Step 3: Incorporate Design Mechanisms

- Our design mechanisms are based on our Enterprise System choice.
  - Based on our Enterprise system we now know our:
    - Persistence (e.g., adding DAO/Repository)
    - Control (e.g., adding a service class)
    - User Interface (e.g. show jsp/Thymeleaf/JSF/php blade)
- We annotate existing interaction diagrams describing where our design mechanisms come into play

Step 3: Design Mechanisms

- Spring – example design mechanisms:
  - Each entity class will have a Service Class that will persist the object using a Repository interface.
  - One DAO/Repository interface per entity class
  - Each UI Form analysis class will be a JSP or Thymeleaf page
  - We show how the controller navigates the pages and passes the model info

Spring MVC/Hibernate Architecture Basic Layers



Our Spring Use Case Design Conventions

- Sequence Diagrams
  - We show the JSP/Thymeleaf pages calling the Controller methods.
  - We show the model being passed in the methods as needed
  - We show //bindData from Page to Controller
  - We show //navigate from controller to Page
  - // indicates Spring MVC doing the work
  - With comments we can show checking bindingResult
  - We show the Service Implementation calling the DAO/Repository interface methods

Our Spring Use Case Design Conventions

- VOPC – Spring Enabled
  - In Use Case **Analysis** we showed all communication between classes as simple associations
  - For Use Case **Design** we will show how the Spring framework reduces associations and manages our dependencies

Our Spring Use Case Design Conventions

- VOPC – With Spring Comments
  - We show the JSP/Thymeleaf pages having a dependency (dotted line) with the Controller
    - use ***(@RequestMapping)*** as the label
  - We show Controller has associations with needed Service interfaces
    - Label with **(@Autowired)** to show Spring is doing the dependency injection
    - Use dependency between the Controller and the needed entity classes

Our Spring Use Case Design Conventions

- VOPC – Spring Commented
  - Show the Service Implementation of the Service Interface
  - Show that Service Implementation has a dependency with its entity class.
  - Show the Service Implementation has a simple association with the DAO class.
    - Label with **@Autowired** to show Spring is doing the dependency injection

Step 3: Class Exercise

Pick one or your group's use cases:

For example -- ***Admin creates a course***

1) Review the current VOPC and sequence diagram

2) Add in the additional classes needed based on our use case design conventions for Spring.

   Show the new labeled associations and dependencies and our container supplied services.

Group Exercise Challenge:

3) Create a VOPC and Sequence Diagram for login that shows clearly the services we get from the Spring Security system.

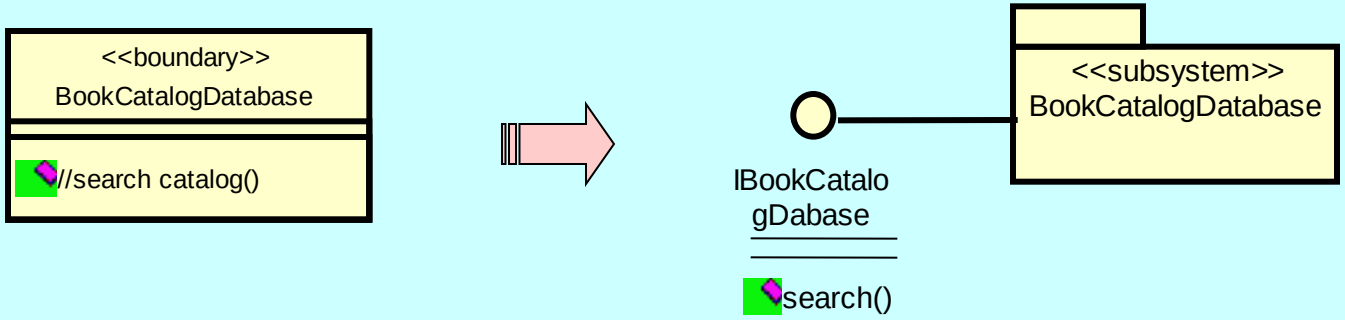
Step 3 Main Point

Incorporating our Enterprise Architecture choice into use case realizations involves:

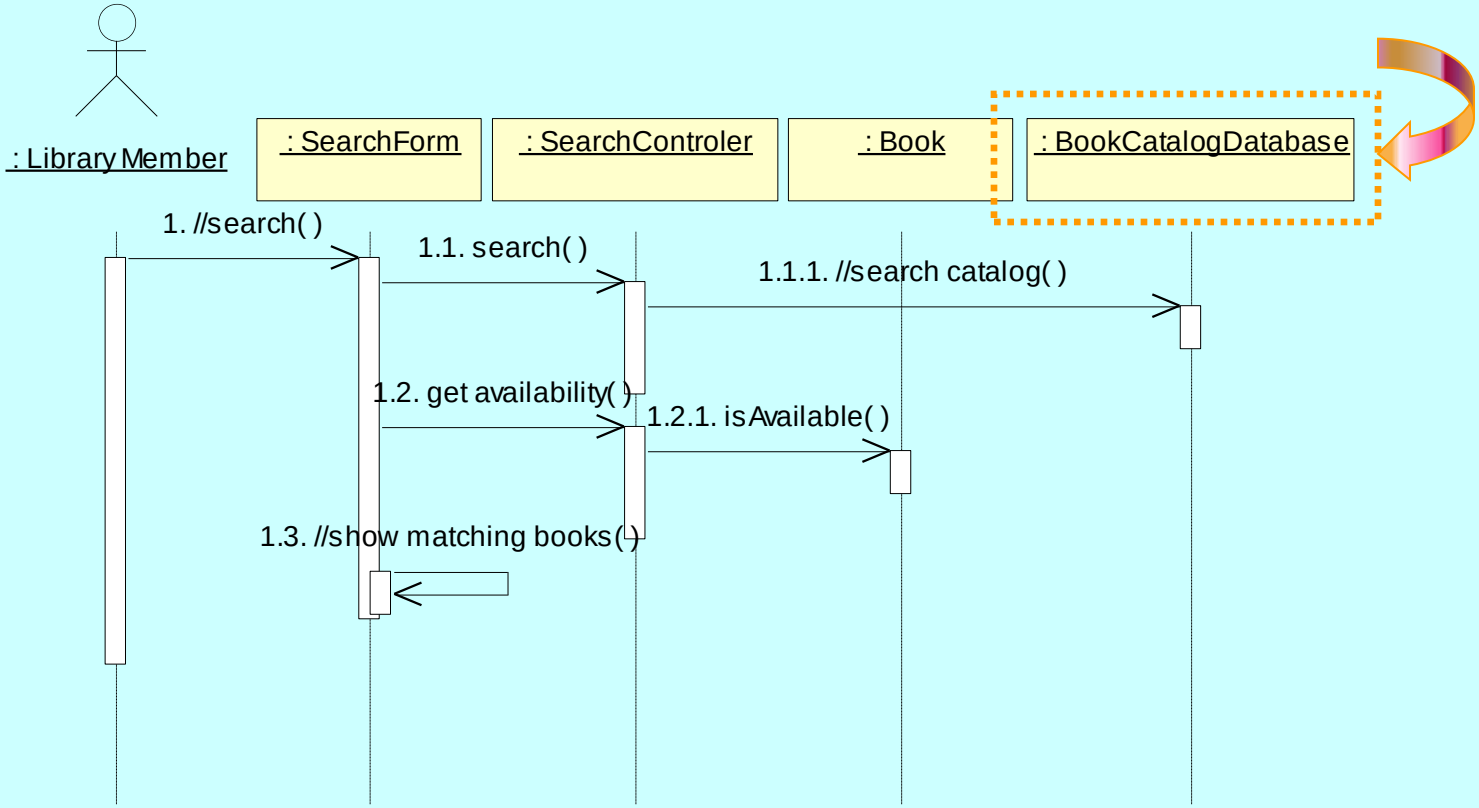
- adding additional design classes
- updating messages and relationships
- Showing how our container provides services to our use cases

Step 4: Incorporate Subsystems

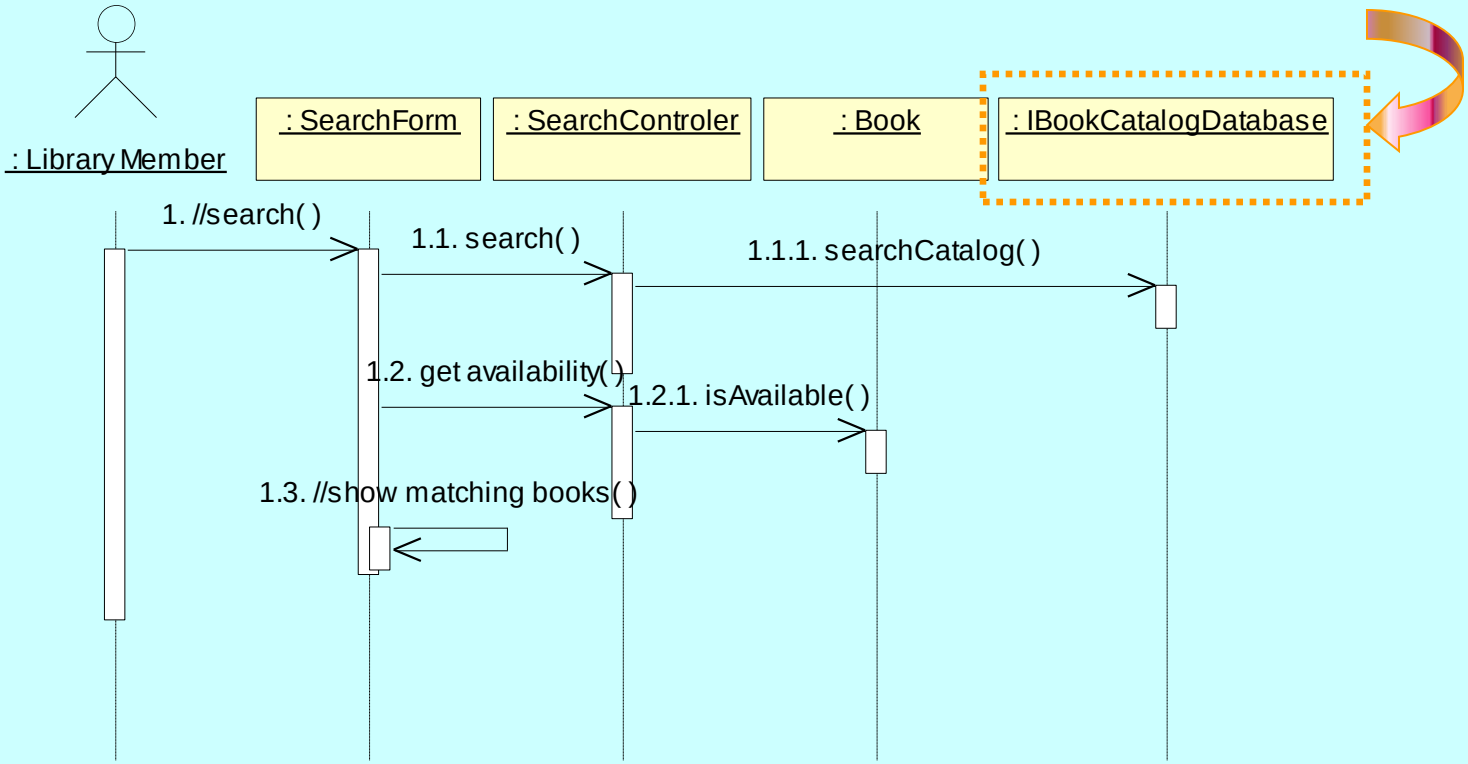
- For each analysis class that has been replaced by a subsystem, replace the class with the subsystem interface in all diagrams.



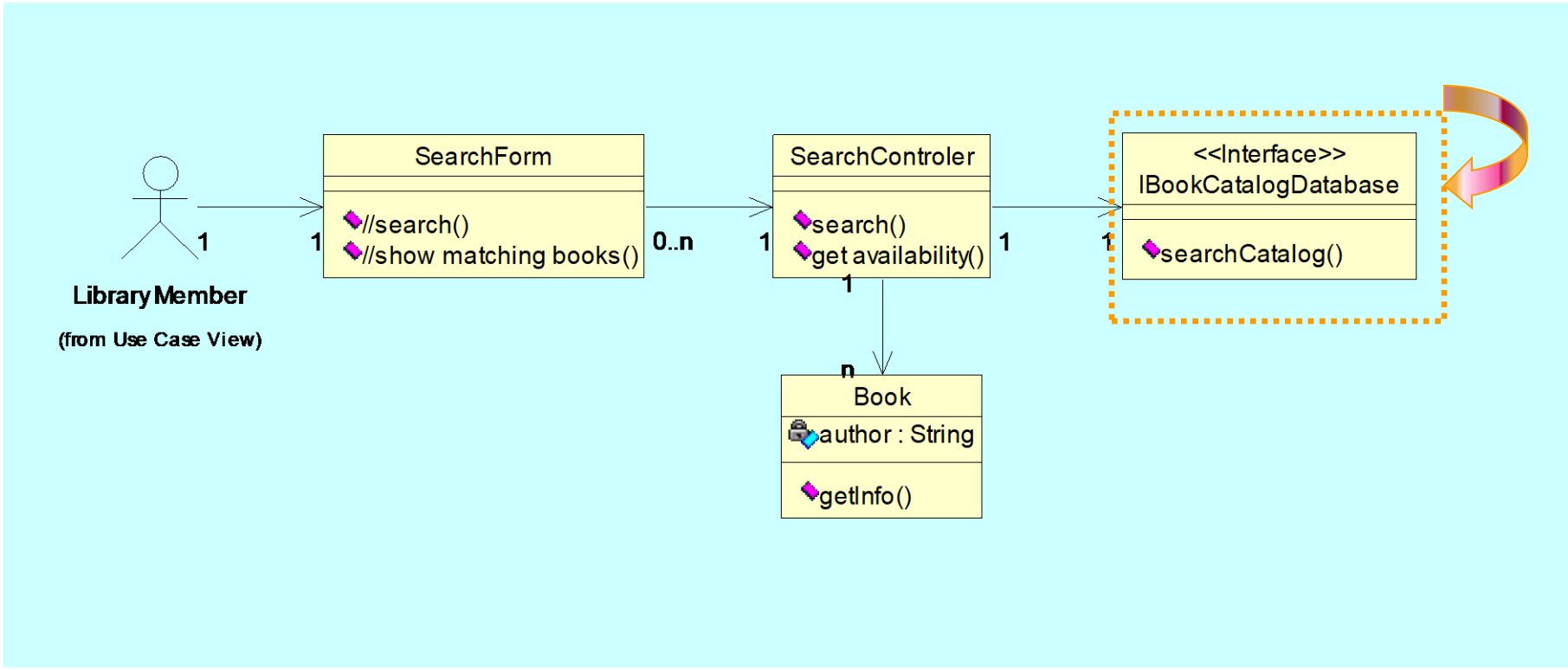
Before Incorporating Subsystem Interface



After Incorporating Subsystem Interface



Incorporating Subsystem into VOPC



Step 4: Class Exercise

Each group review use case sequence diagrams for:

Student Registers for a Section

- Show incorporating the Registration Subsystem Interface into this use case Sequence diagram and VOPC.

Review Questions

1. Who performs use case design?
2. What are the main inputs?
3. What are the main outputs?
4. What do we do to our to use case realizations?
5. Do we modify our subsystems?

Module Summary

1. Use case design is mainly a refinement of the use case realizations created during use case analysis.
2. Primary activities are:

Incorporating our

1. design classes,
2. inheritance design,
3. design mechanisms, and
4. Subsystems

identified in architectural design into our analysis diagrams.

This activity will often result in further refinements to subsystem responsibilities.

Use Case Design – Main Points

Design elements and mechanisms replace their corresponding analysis classes in the use case realizations. This is an exercise in moving from the system wide view in the Architecture Design phase to the detailed view of the Use Case Sequence Diagrams and View of Participating Classes. Design decisions will be more successful if they have system wide coherence.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

1. Use case design is a refinement of the use case realizations created during use case analysis.

2. The ability to alternate from a broad perspective to a focused detailed perspective is one of the key skills needed for successful software design and development.

.

Use Case Design – Main Points (cont.)

.

3. **Transcendental consciousness** is the experience of unbounded pure awareness which is the most broad perspective possible.

4. **Impulses within the transcendental field**: We can experience finely focused attention with the power of the broad perspective at the subtlest level of thought.

5. **Wholeness moving within itself:** In unity consciousness, all individual aspects of a system are appreciated in terms of the connection to the whole system.

.

.