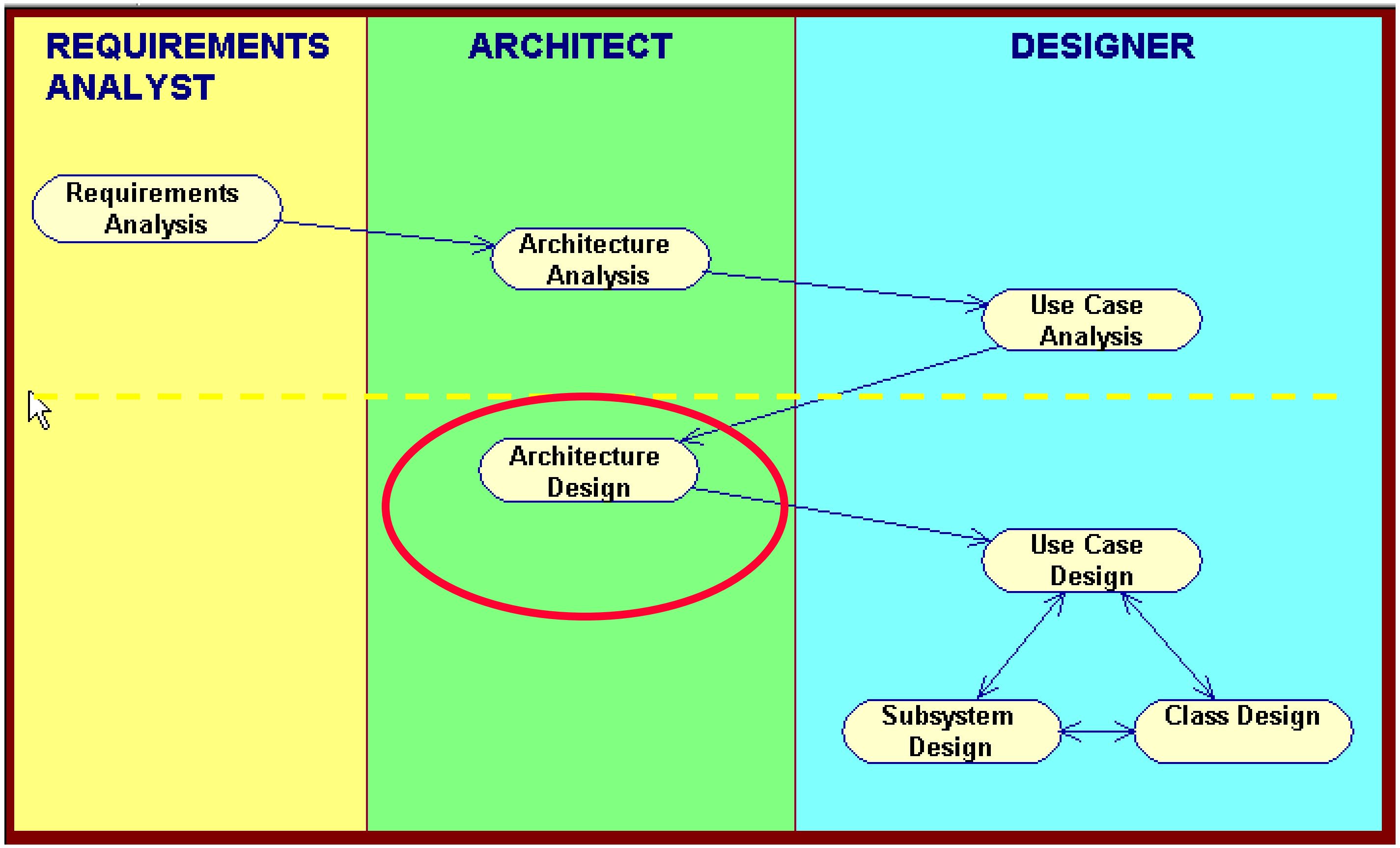
Architecture Design

The whole is greater than the sum of the parts.

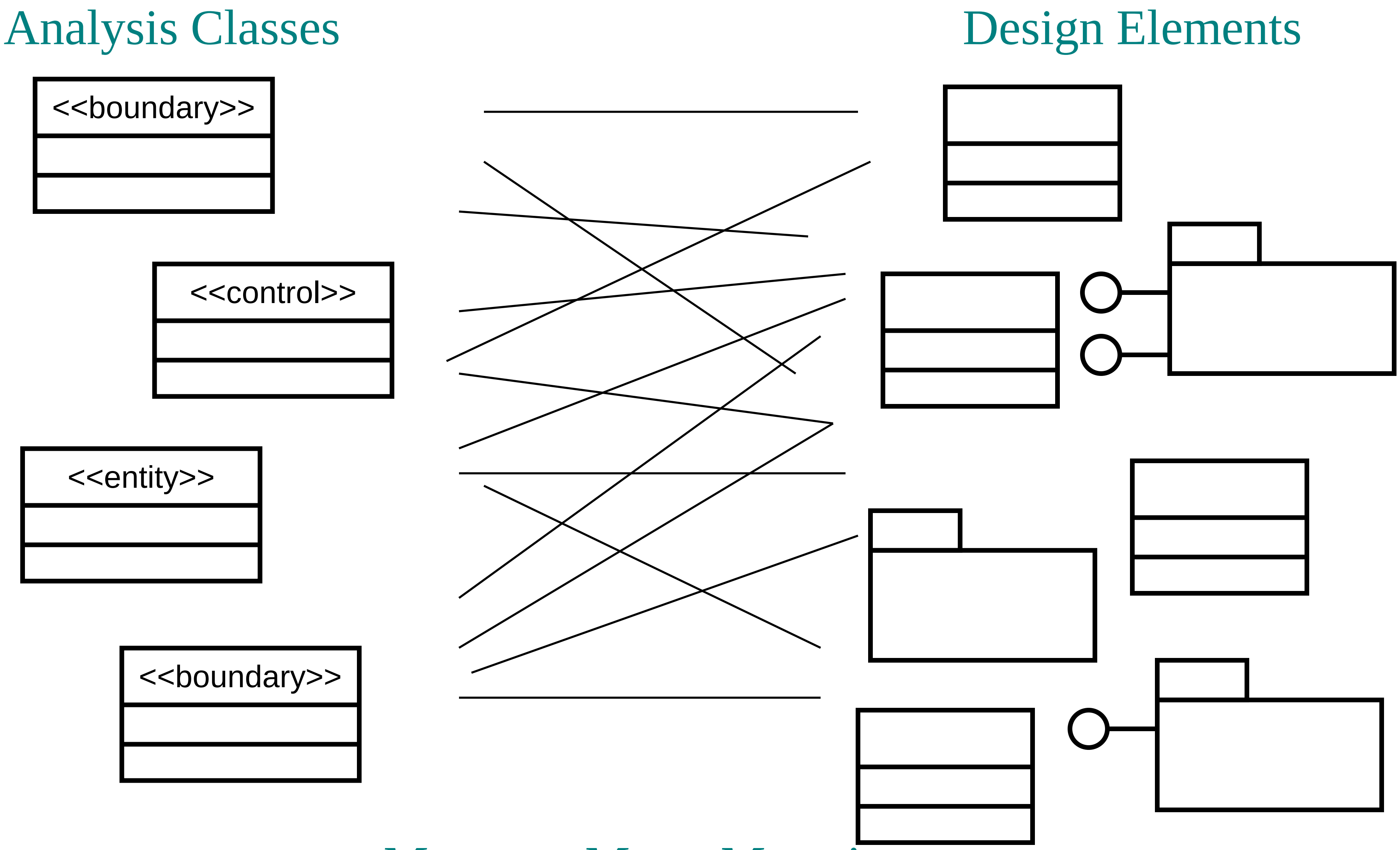
Basic RUP OOAD Activities



Module Main Points

- As architects we make system-wide preparations that guide and coordinate the design activities of individual designers.
- Our Main Architecture Design Activities:
  1. First, transform analysis classes to design classes
  2. Second, do we see any reuse opportunities from the class to the system level?
  3. Third, identify subsystems and specify their interfaces
  4. Finally, identify our Design Mechanisms - *i.e., our Enterprise Architecture and Frameworks*

From Analysis Classes to Design Elements



Many-to-Many Mapping

Identifying Design Classes

- Analysis class maps directly to a design class if:
  - Simple
  - Represents a single logical abstraction
- More complex analysis classes may
  - Split into multiple classes
  - Become a package
  - Become a subsystem (discussed later)
  - Any combination …

Example: Analysis-Class-To-Design-Element Map

Analysis class	**Design elements**
SearchForm	SearchForm
SearchController	SearchController
Book	Book
	BookList
LibraryMember	LibraryMember
CheckoutRecord	CheckoutRecord
	CheckoutRecordList
BookCatalogDatabase	BookCatalogDB subsystem
	IBookCatalogDB interface
CheckoutForm	CheckoutForm
CheckoutController	CheckoutController
CheckinForm	CheckinForm
CheckinController	CheckinController
	LibraryDB subsystem
	ILibraryDB interface

Architecture Step Two

- Our second step as Architects is to identify any resuse opportunities for the whole system.

Reuse Levels

- Class
- Inheritance Hierarchy
- Aggregation Hierarchy
- Cluster (Packages and Subsystem) Step 3 for us

The following are part of our Enterprise Arch choice in our Step 4

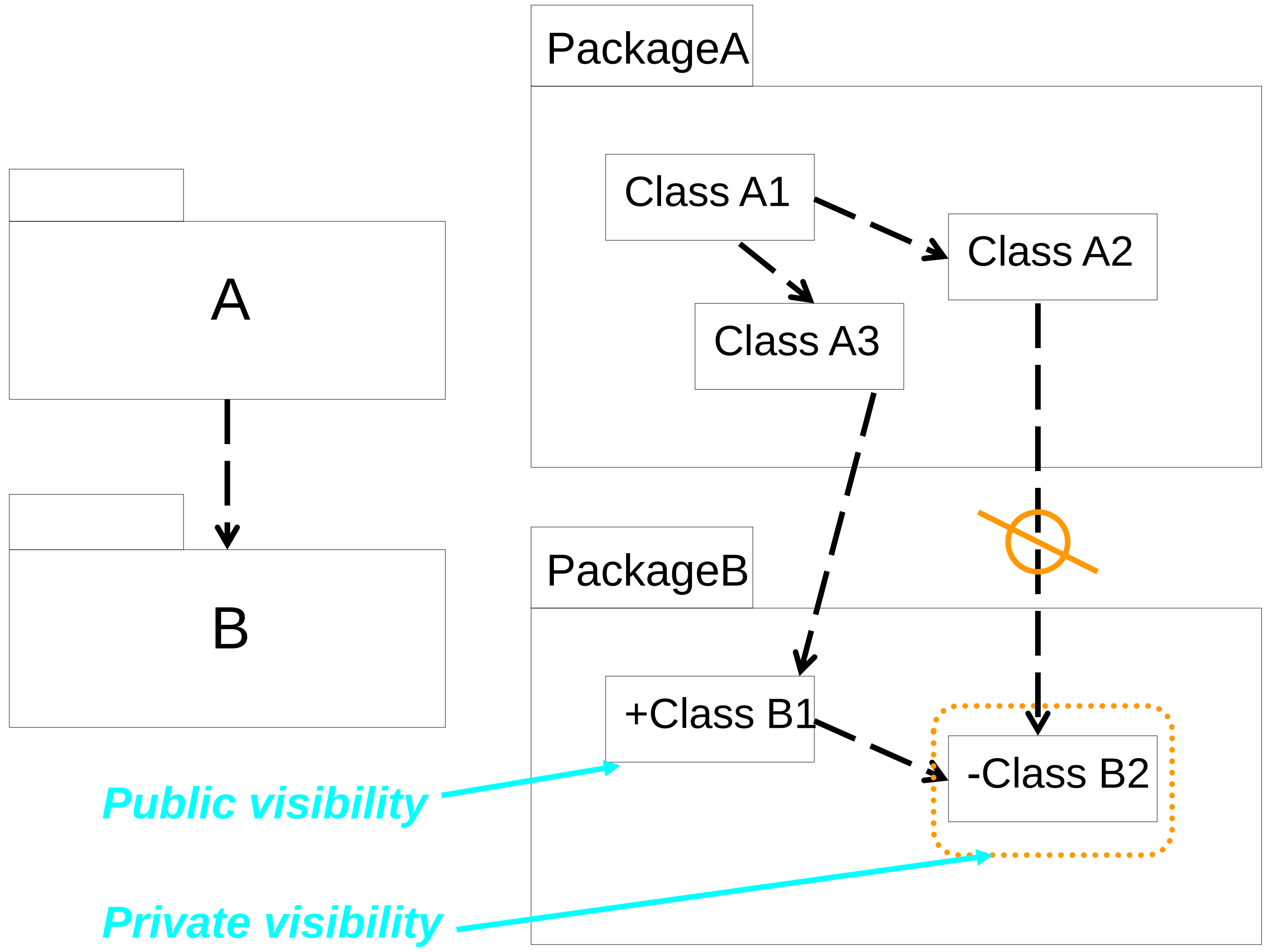
  - Framework
  - Component
  - Patterns
  - Generic Architecture

Architecture Step Three

- Our third step as Architects is to identify our packaging and subsystems.

- Specifying subsystem interfaces early on allows developers to work in parallel, relying on interfaces of other subsystems long before design and implementation of those subsystems are complete.
- Any substantial system you work on will have many subsystems. We will have at least one for our project.

Package Dependencies: Package Element Visibility

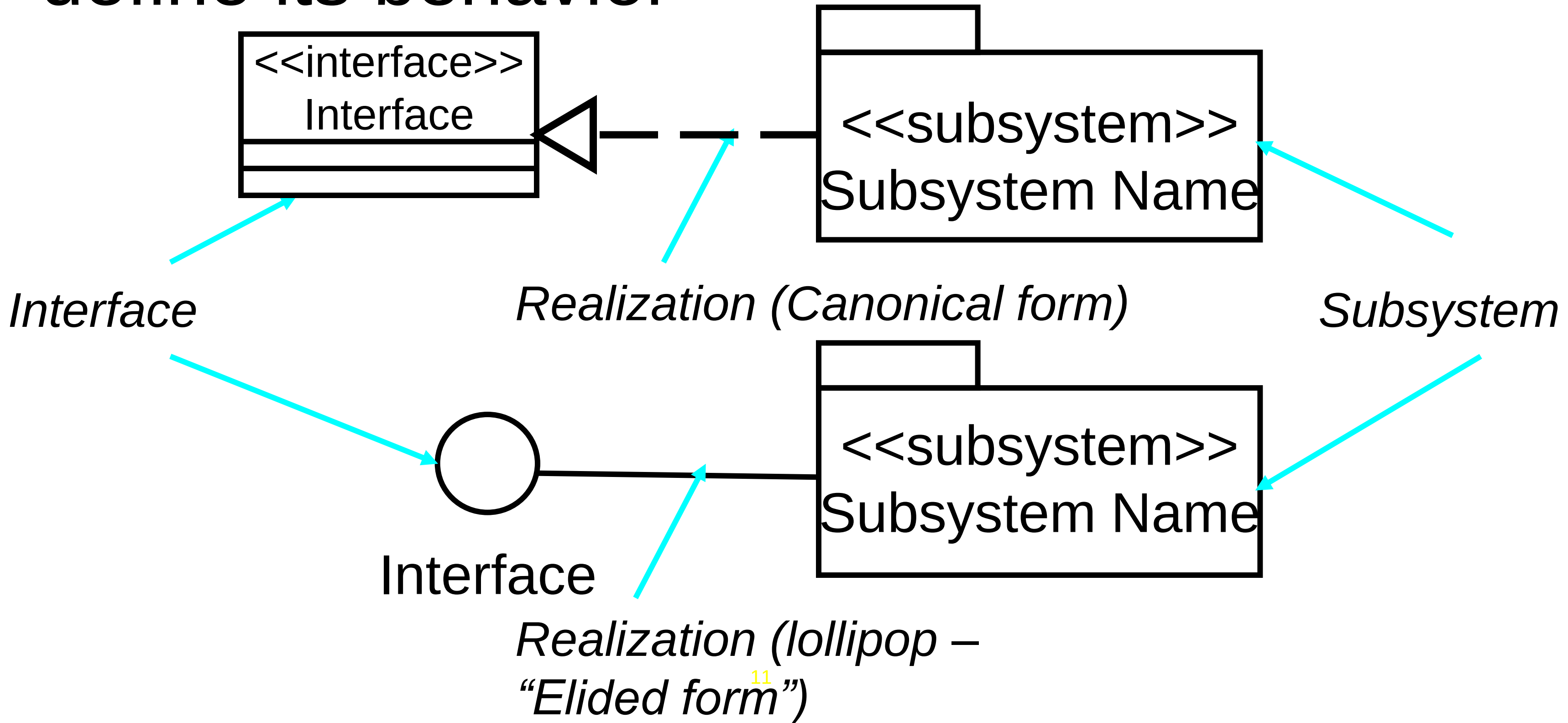


Only public classes can be referenced outside of the owning package

OO Principle: Encapsulation

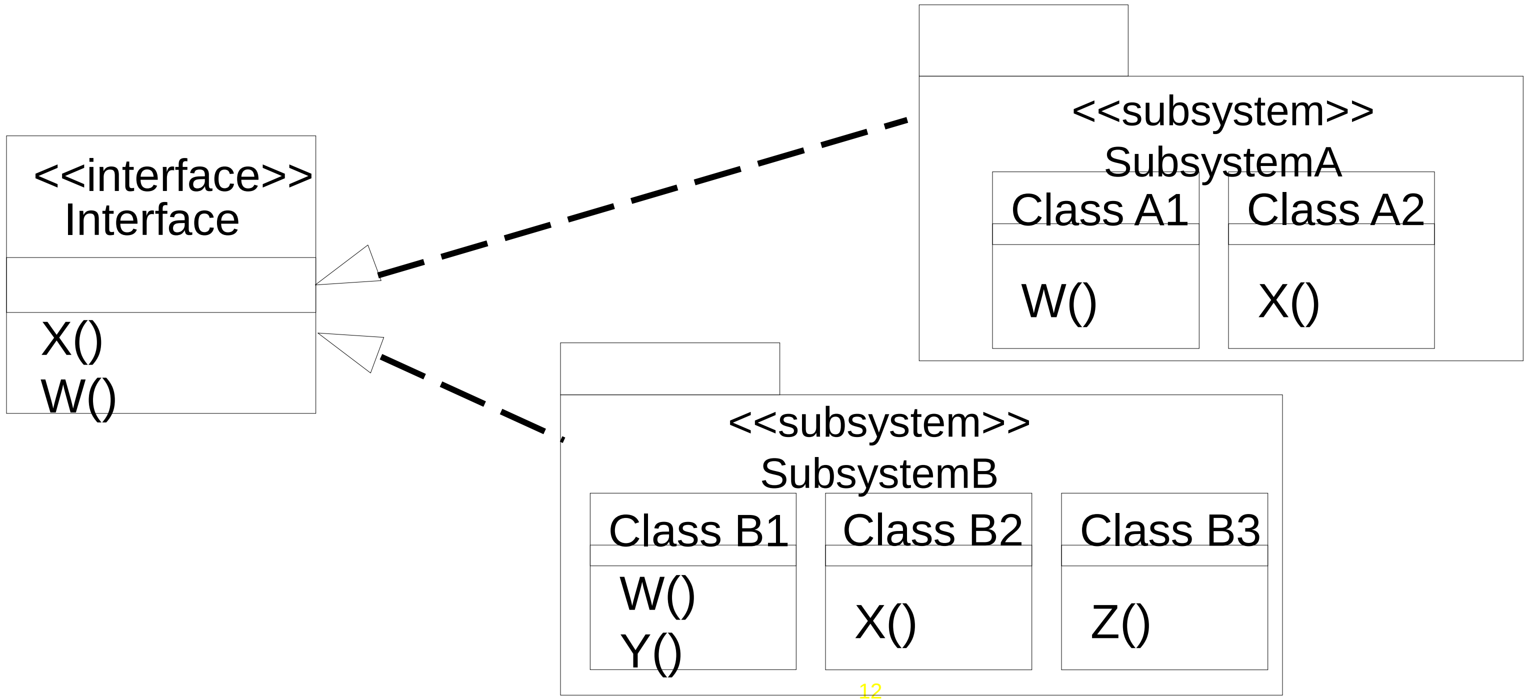
Subsystems and Interfaces

- A cross between a package (can contain other model elements) and a class (has behavior)
- Realizes one or more interfaces which define its behavior



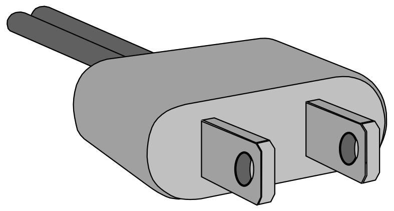
Subsystems and Interfaces (cont.)

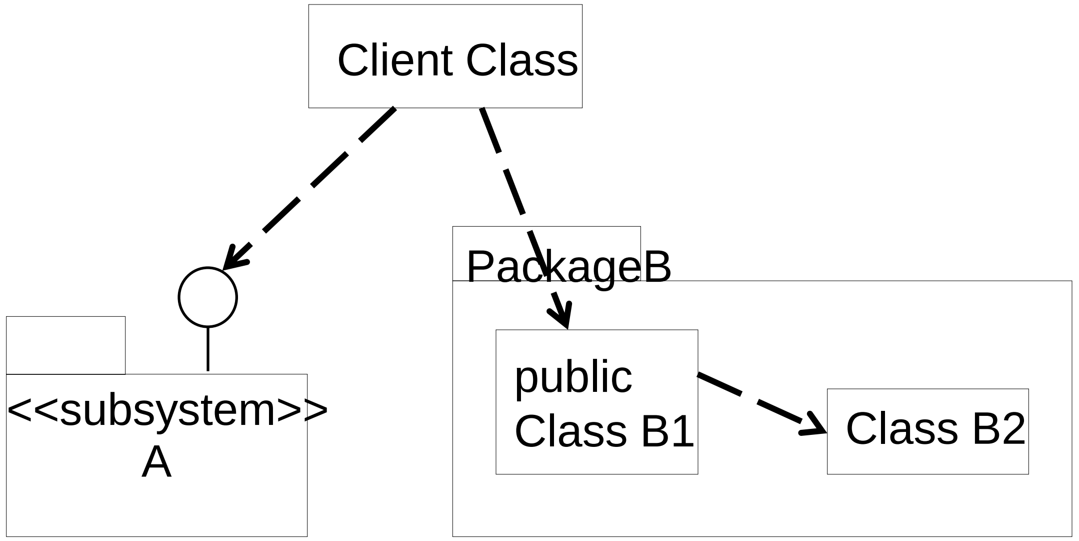
- Subsystems :
  - Encapsulate behavior
  - Represent an independent capability with clear interfaces (potential for reuse)
  - Allow multiple implementation variants (difference from a package)



Packages Vs. Subsystems

- Subsystems provide behavior, packages just provide organization of classes
- Subsystems encapsulate their contents, packages do not – dependencies on packages imply dependencies on contents (However, dependencies even here must be on *public* classes.)
- Subsystems are easily replaceable





Encapsulation is the key!

Subsystem Usage

- Subsystems can be used to partition the system into parts which can be independently:
  - ordered, configured, or delivered
  - developed, as long as the interfaces remain unchanged
  - changed without breaking other parts of the systems
- Subsystems can also be used to:
  - partition the system into units which can provide restricted security over key resources
  - represent existing products or external systems in the design (e.g. components)

Identifying Subsystems

- *High coupling.* If the classes in a collaboration interact only with each other to produce a well-defined set of results, then encapsulate them within a subsystem.

- *Optional behavior.* If collaborations model optional behavior, or features which may be removed, upgraded, or replaced with alternatives, then encapsulate them within a subsystem. “Pluggable components”

- *Different actors.* Separate functionality used by two different actors, since each actor may independently change their requirements.

- *Different service levels.* Represent different service levels for a particular capability (e.g., high, medium and low availability) as a separate subsystem, each of which realizes the same interfaces.

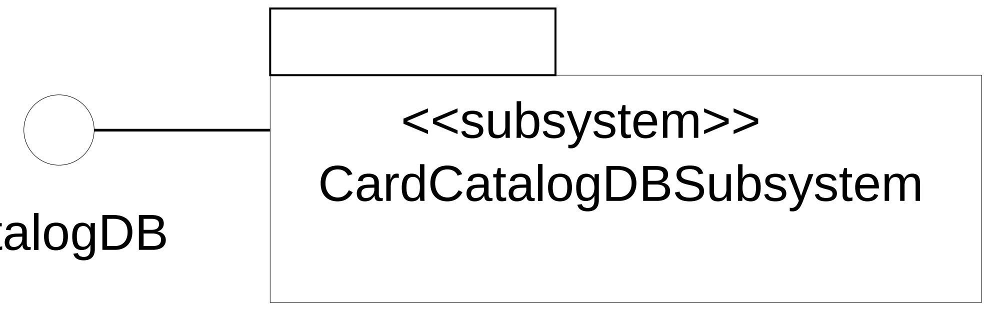
- *Volatility.* You will want to encapsulate chunks of you system that you expect to change.

Candidate Subsystems

- Analysis classes which may evolve into subsystems:
  - Classes providing complex services and/or utilities
    - E.g., mortgage calculator, insurance rate calculations, …
  - Boundary classes (user interfaces and external system interfaces)
- Existing products or external systems in the design (e.g. components)
  - Communication/Networking software
  - Database access support
  - Common utilities (math libraries, string parsing)
  - Application-specific products (billing system, scheduler)

Modeling Convention: Subsystems and Interfaces

3 Items In a Model:

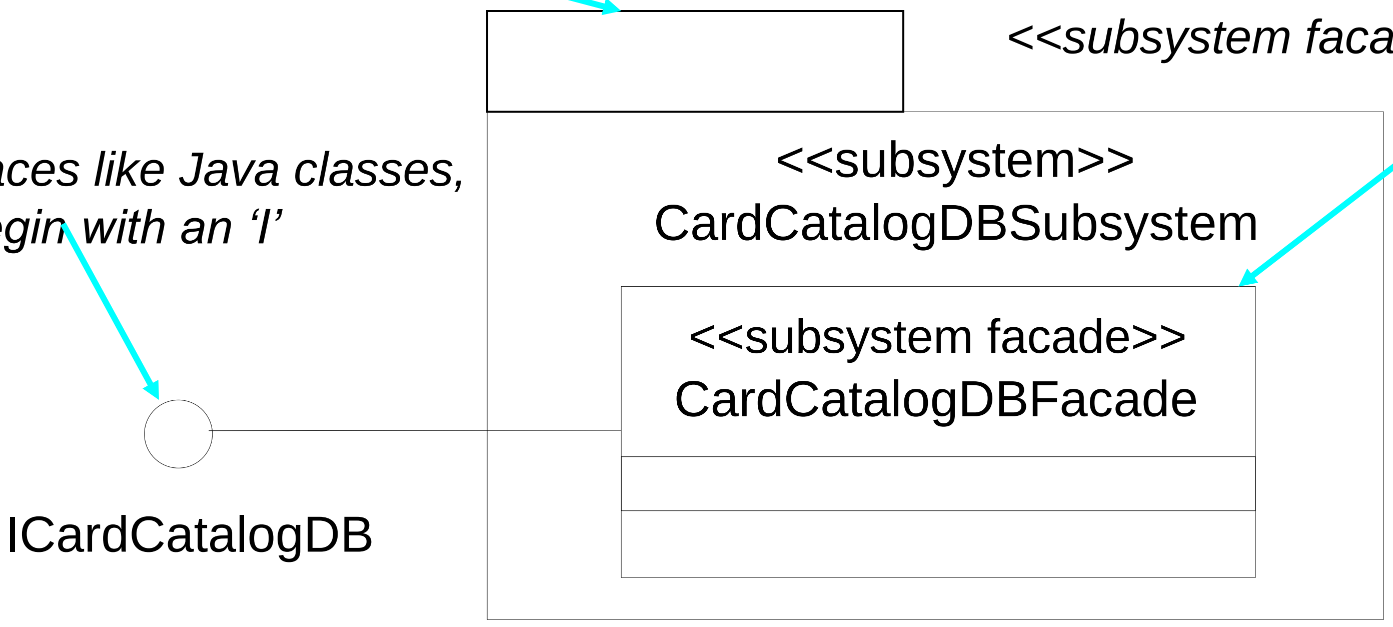


modeled as

<<subsystem>> package

<<subsystem facade>> class

Interfaces like Java classes, but begin with an 'I'



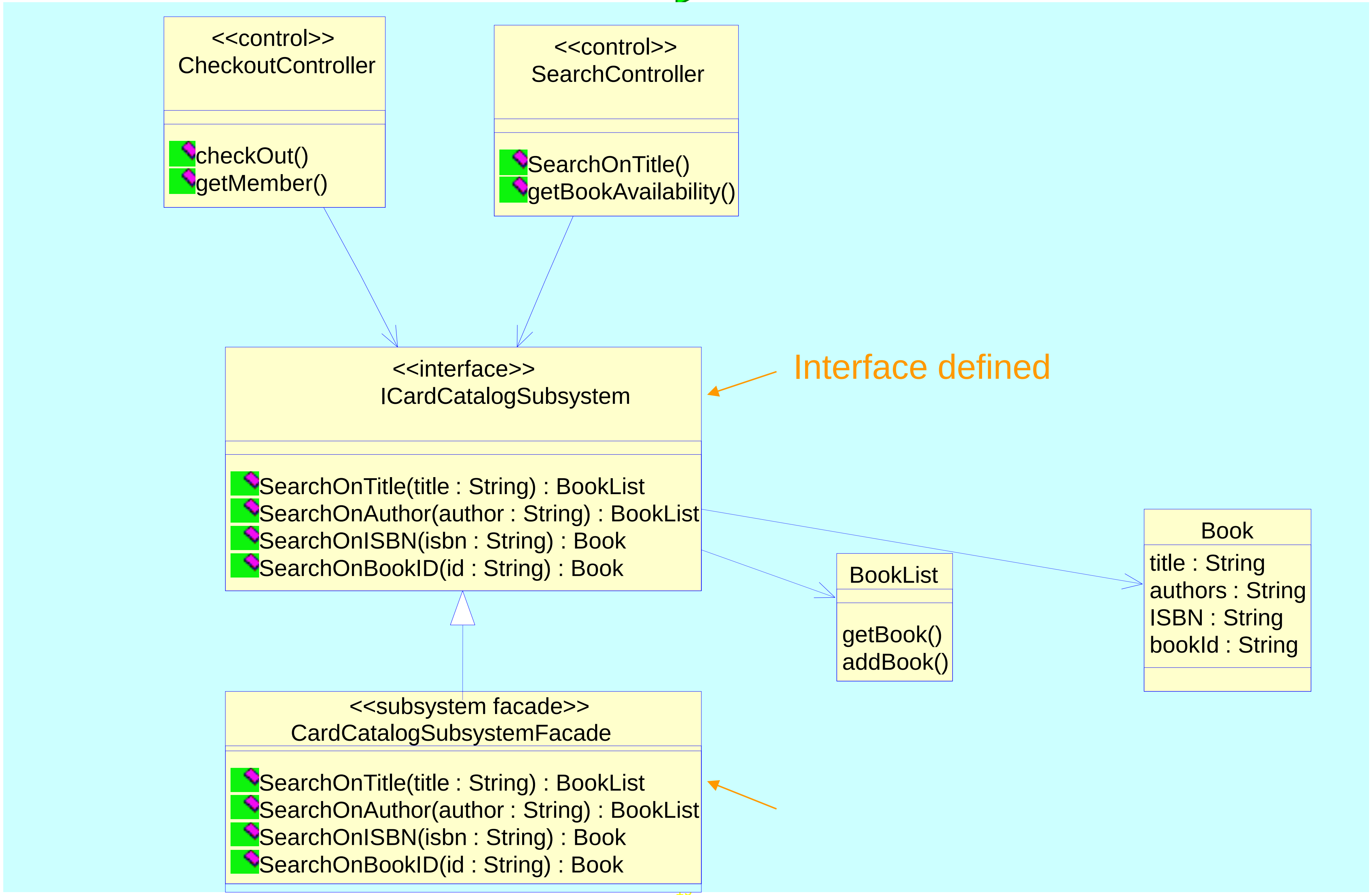
Subsystem Context Diagrams

- Once the subsystems have been identified and their interfaces specified, the architect determines the relationships between each subsystem and the rest of the system in a *subsystem context diagram*

- The subsystem context diagram displays the subsystem interface (and typically also the implementing façade(s)) together with
  - Classes that will *use* or depend upon the subsystem
  - Classes that are *used by* the subsystem

- During architecture design, no attempt is made to specify the detailed interactions *inside* each subsystem -- this step is done during *subsystem design*

Example: Subsystem Context: Library System



<<control>> CheckoutController
checkOut()
getMember()
<<control>> SearchController
SearchOnTitle()
getBookAvailability()
<<interface>> ICardCatalogSubsystem
SearchOnTitle(title : String) : BookList
SearchOnAuthor(author : String) : BookList
SearchOnISBN(isbn : String) : Book
SearchOnBookID(id : String) : Book
Interface defined
BookList
getBook()
addBook()
Book
title : String
authors : String
ISBN : String
bookId : String
<<subsystem facade>> CardCatalogSubsystemFacade
SearchOnTitle(title : String) : BookList
SearchOnAuthor(author : String) : BookList
SearchOnISBN(isbn : String) : Book
SearchOnBookID(id : String) : Book

Main Points

- **During Architectural Design, first we determine the design classes from the old analysis classes.**
- **Second, we looked for reuse opportunities.**
- **Third we identified clusters of analysis classes that work together -- namely, *subsystems*. We define the interface of each subsystem.**
- **Carefully defining the interfaces between subsystems is a critical activity because it establishes the coordinating interface for parallel development efforts.**

Group Exercise

- **Assume we will have a Registration subsystem.**
- **List the interface methods for our Registration subsystem.**
- **Think of what services Registration will provide to MUMSched users and clients of the subsystem.**

Architecture Step Four

- Our final step as Architects is to identify our design mechanisms. These come with our Enterprise Architecture

Architecture Design Mechanisms

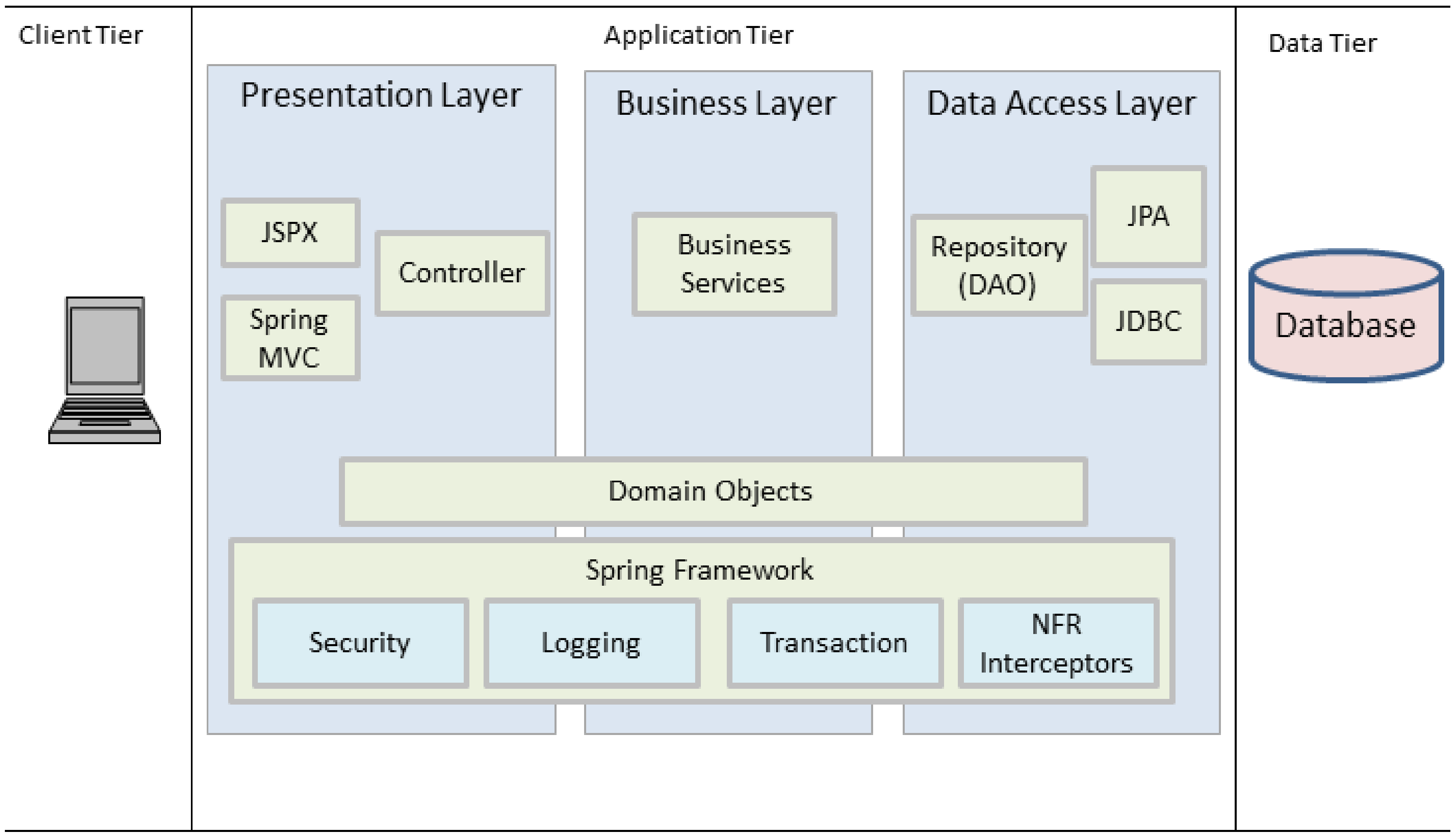
1. We have figured out the evolution from our analysis classes to our design classes.
2. We identified any reuse opportunities at the class level
3. We have figured out that we want a registration subsystem.
4. Finally, we make the choice on our Enterprise System which leads us to the our project design mechanisms and organization.

Most groups will use Spring as our Enterprise System. We now compare the organization of Spring to Java Enterprise Edition 7 and .Net projects.

Spring MVC/Hibernate Architecture Basic Layers

1. Spring MVC
   a) Model (view resolver/ backing bean)
   b) View (for example, a JSP or Thymeleaf web page)
   c) Controller Class
2. Business Logic → Spring
   a) Entities
   b) Service Layer
3. JPA →DAO per Entity
4. DataBase → JavaDB or MySQL

Spring MVC/Hibernate Architecture Basic Layers



Spring Boot Project Organization

See our StudentSpringBoot Project in sakai for a typical Spring Boot Application structure

- *StudentSpringBoot\src\main\java\edu\mum\mumsched* has the StudentSpringBootApplication class
- Sub-directories are:
  - config -- has the class DataSrc
  - controllers -- includes StudentRegController. @Autowired to their service classes.
  - dao - includes @Repository that extends CrudRepository

    one repository per entity class
  - domain – includes @Entity classes
  - service – includes service interfaces (one per Entity class) and

  the @Service implementation for each service interface. These are @Autowired to their respective DAO.

Spring Boot Project Organization

- *StudentSpringBoot\src\main\resources* has the application.properties file - e.g. persistence properties
  - templates has the web pages (using Thymeleaf)

- *StudentSpringBoot\src*
  - pom.xml -- all our dependencies

JEE7 Web Architecture Basic Layers

1. Presentation → JSF web pages
2. Navigation → Named/Managed/Backing beans –Referred to by JSF Expression Language
3. Business Logic →
   a) EJBs – our control classes (if we need more than provided by navigation.)
   b) Entities
4. JPA →Generic DAO and one DTO per Entity
5. DataBase → JavaDB or MySQL

JEE7/NetBeans7 Packages Organization

- By Convention - NetBeans will provide:
  1. Web Pages
     a. WEB-INF (xml config files)
     b. Web pages and web page packages
     c. Resources - css files, images, javascript, etc.
  2. Source Packages – divide into packages by MVC pattern. E.g:
     1. edu.mum.msched.view package for all the managed beans (per use case)
     2. edu.mum.msched.control package for all the control classes and business logic (per use case)
     3. edu.mum.msched.model package for all the data access classes (per use case)
  3. Libraries
     1. JDK 1.x
     2. GlassFish Server 4.x

JEE7/NetBeans7 Organization (cont.)

4. Enterprise Beans

All source classes that are annotated to be EJBs are run in an EJB container that provides services like: interprocess communication, dependency injection, life cycle management, transaction management, etc. This provides a list all your EJBs.

5. Configuration Files

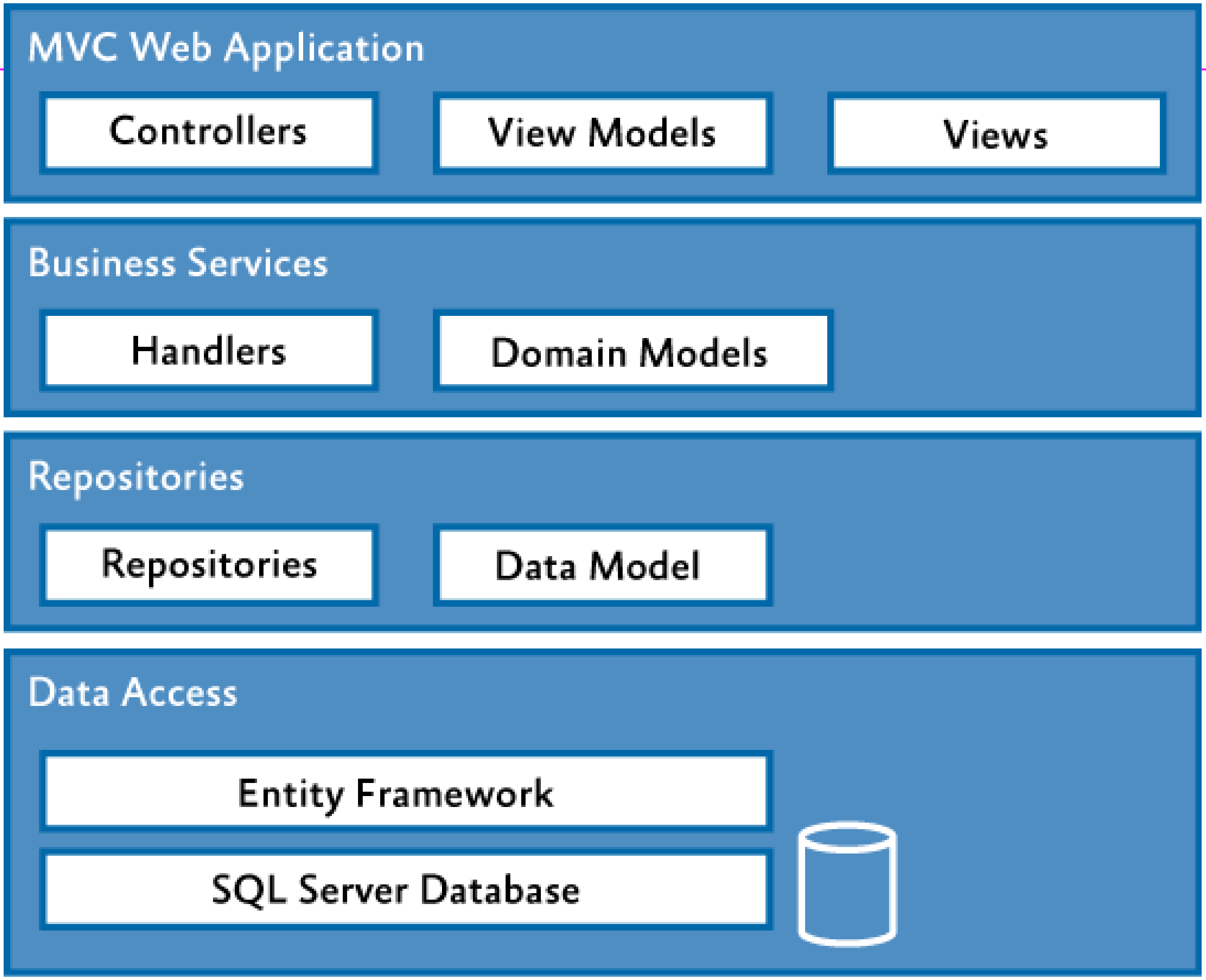
.xml files

6. Server Resources

glassfish-resources.xml

.NET Web Architecture Possible Layers

1. Presentation → MVC Web Application
2. ~~Navigation → backing beans~~
3. Business Logic →
   a) Domain Model
   b) Handlers
   c) Security, Concurrency & Transacation →.NET + EF built-in features
4. JPA →Entity Framework + Repositories
5. DataBase → SQL Server


MVC Web Application
Controllers
View Models
Views
Business Services
Handlers
Domain Models
Repositories
Repositories
Data Model
Data Access
Entity Framework
SQL Server Database

Architecture Design Organization for .NET

Visual Studio

- Namespaces – like subsystems, but they are actually declared (unlike Java subsystems.)
- Assemblies – a unit of deployment (.exe or .dll) – like a super-package with more descriptive information

1. edu.mum.mscrum.view namespace for all views-(per use case)
2. edu.mum.mscrum.control namespace for all the control classes and business logic (per use case)
3. edu.mum.mscrum.model namespace for all the data access classes (per use case)

JAVA Design and Implementation Mechanisms

Analysis Mechanism (Conceptual)	Design Mechanism (Concrete)	Implementation Mechanism (Actual)
Persistency →	Legacy Data RDBMS →	JDBC
ORM →	New Data JPA →	Hibernate, EclipseLink, etc.
UI/Web Framework →	Java Server Faces, Spring MVC →	JSF 2.x, Primefaces, JSP, Thymeleaf, etc.
Analysis	**Design**	**Implementation**

.NET Design and Implementation Mechanisms

Analysis Mechanism (Conceptual)		Design Mechanism (Concrete)		Implementation Mechanism (Actual)
		Legacy Data		
Persistency	→	RDBMS	→	ADO. NET
		New Data		
ORM	→	Repositories + UoW	→	POCO Entity Framework (Code First)
UI/Web Framework	→	Asp.net MVC	→	Asp.net MVC
Analysis		**Design**		**Implementation**

Architectural Design Review

- Purpose of Architectural Design?
- What is a subsystem? How different from a package? How similar to a class?
- How do you identify subsystems?
- What are some reuse considerations?
- What are Design and Implementation Mechanisms? Give examples.

Architecture Design Summary

Architecture design looks at the entire system and prepares an overall structure to guide and coordinate specific design activities.

The major activities are

1. identifying our design classes
2. identifying reuse opportunities
3. establishing the system structure and organization in terms of subsystems and packages
4. identifying design mechanisms/ enterprise architecture – now we settle on our frameworks, persistence implementations, etc.

Architecture Design Main Points

The Organizing Power of Pure Consciousness

1. During Architecture Design, the Architect further defines the internal structure of the system all the way from the design class level to the enterprise architecture level. With our system architecture, we can now complete the design for each use case with the actual classes that will be used for our implementation.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

1. In Architecture Design, we make and document the decisions on how our solution will be structured to guide further detailed design by our developers.
2. The key to a successful architecture is one that effectively breaks the system up into independent parts while maintaining system-wide coherence and structure.

Architecture Design Main Points (cont)

The Organizing Power of Pure Consciousness

Transcendental consciousness is the field beyond differences. Distinctions between parts dissolve into this unbounded field. On the basis of the clarity of the unbounded field, parts are then appreciated in proper perspective; relationships between parts are naturally optimized.

Impulses within the transcendental field: As parts unfold from within wholeness, in the transcendental field, each part remains fully connected to its source. The unbounded value of intelligence is never lost in its finite expressions.

Wholeness moving within itself: In unity consciousness, every part is appreciated as a lively and unique expression of wholeness.