Logical Architecture & Design Preliminaries

CSSE 374: Session 8

Shawn Bohner
Office: Moench Room F212
Phone: (812) 877-8685
Email: bohner@rose-hulman.edu

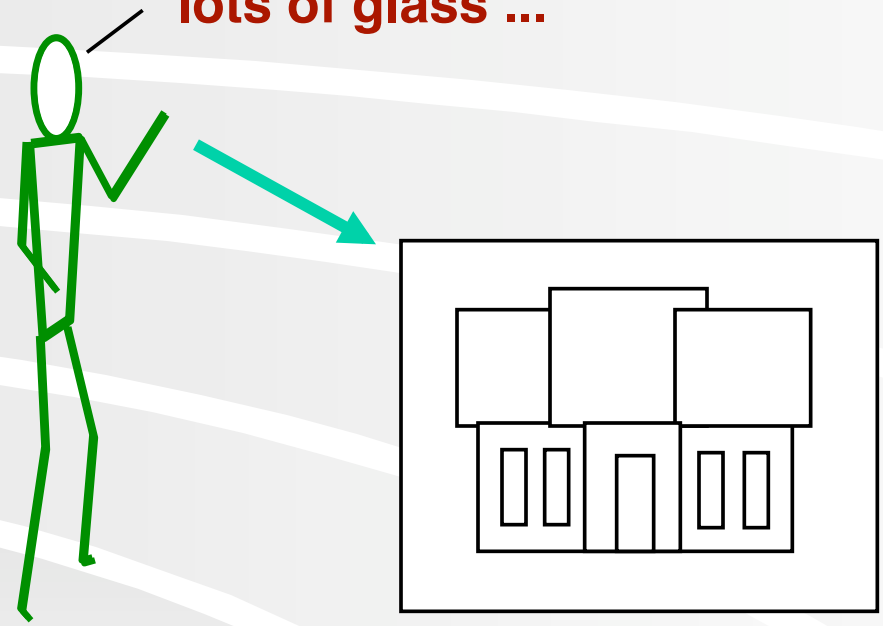
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Q1

From Requirements to Architecture

Customer Requirements

"four bedrooms, three baths, lots of glass ..."

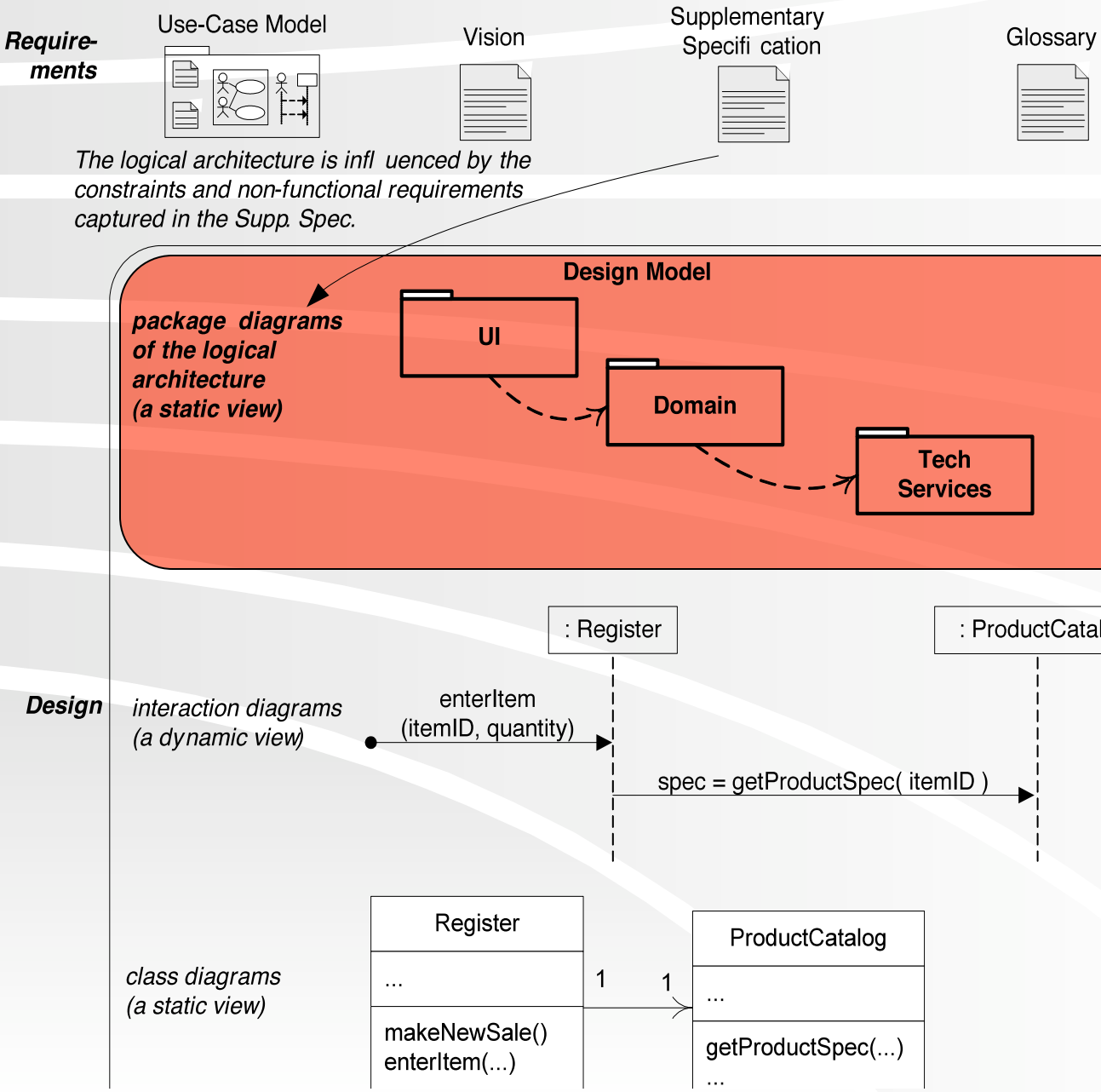


How do we get from there

to here?

Architectural Design

Where is Logical Architecture?



Defining Software Architecture

- **Software architecture: the large-scale motivations, constraints, organization, patterns, responsibilities, and connections of a system**

  Craig Larman 2003

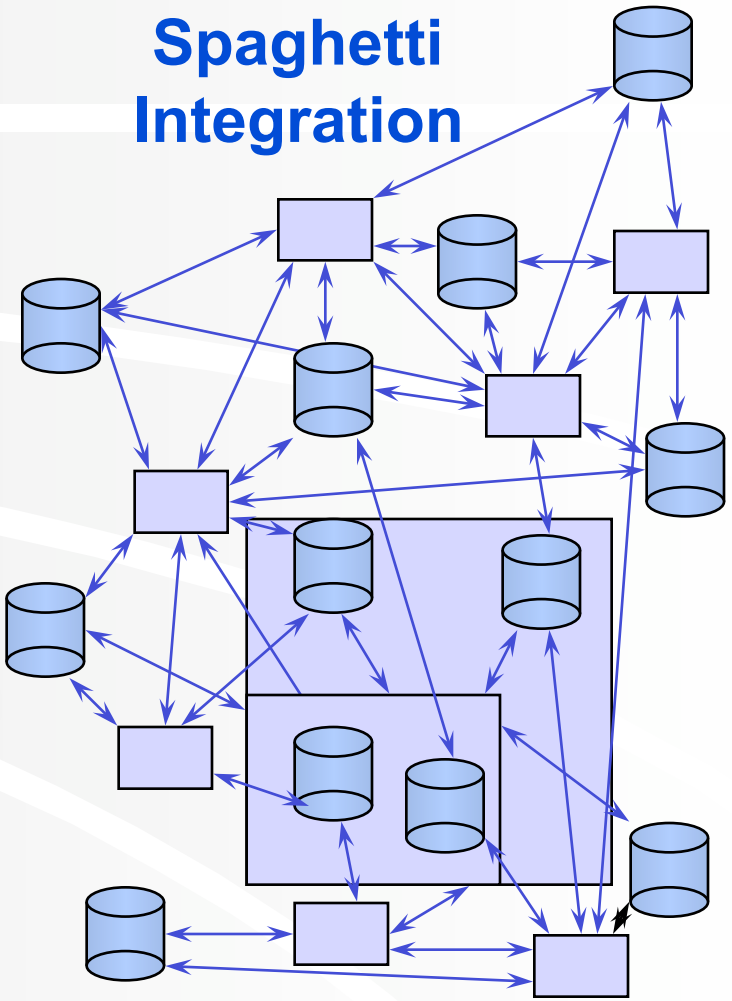
- **The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them.**

  Bass, et al, 1998

Why Software Architecture?

The architecture is a representation that enables a software engineer to:

1. Analyze the effectiveness of the design in meeting its stated requirements,
2. Consider architectural alternatives at a stage when making design changes is still relatively easy, and
3. Reduce the risks associated with the construction of the software.
4. Provide key Abstractions in reasoning about design
5. Establish Design Plan using Software Architecture



Architectural Building Blocks

Component – a unit of computation or a data store

Connector – an architectural element that models interactions among components and rules that govern those interactions

Configuration (or topology) – a connected graph (composite) of components and connectors which describe architectural structure

UML Architectural Views

- **Logical architecture** – describes the system in terms of its organization in layers, packages, classes, interfaces & subsystems

- **Deployment architecture** – describes the system in terms of the allocation of processes to processing units and network configurations

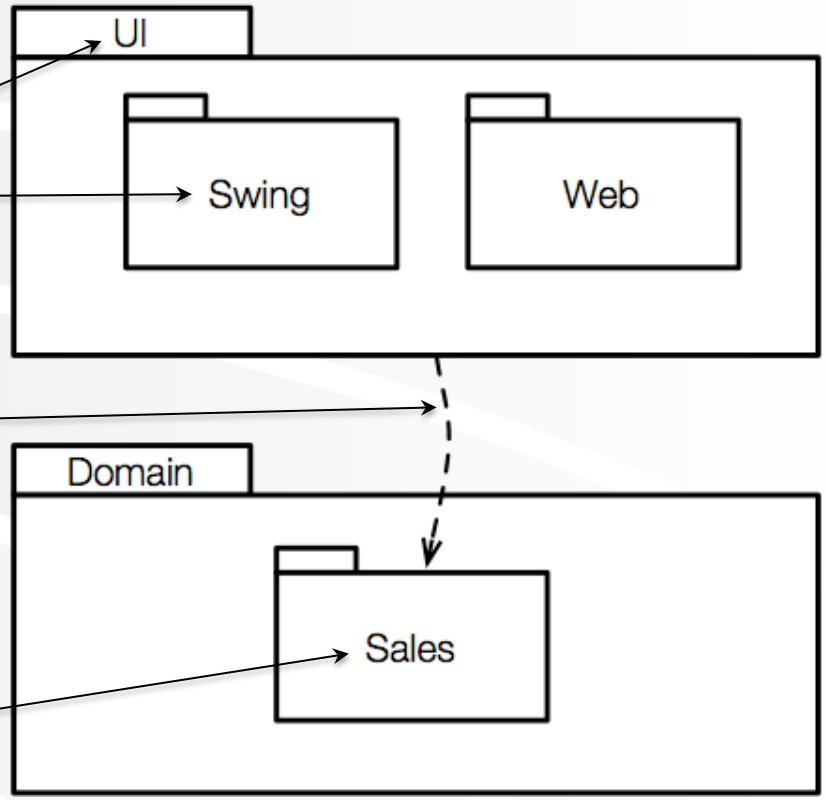
UML Package Diagrams

- Describes grouping of elements
- Can group anything:
  - Classes
  - Other pack
- More gene va
  packages or
  nces

Package Names

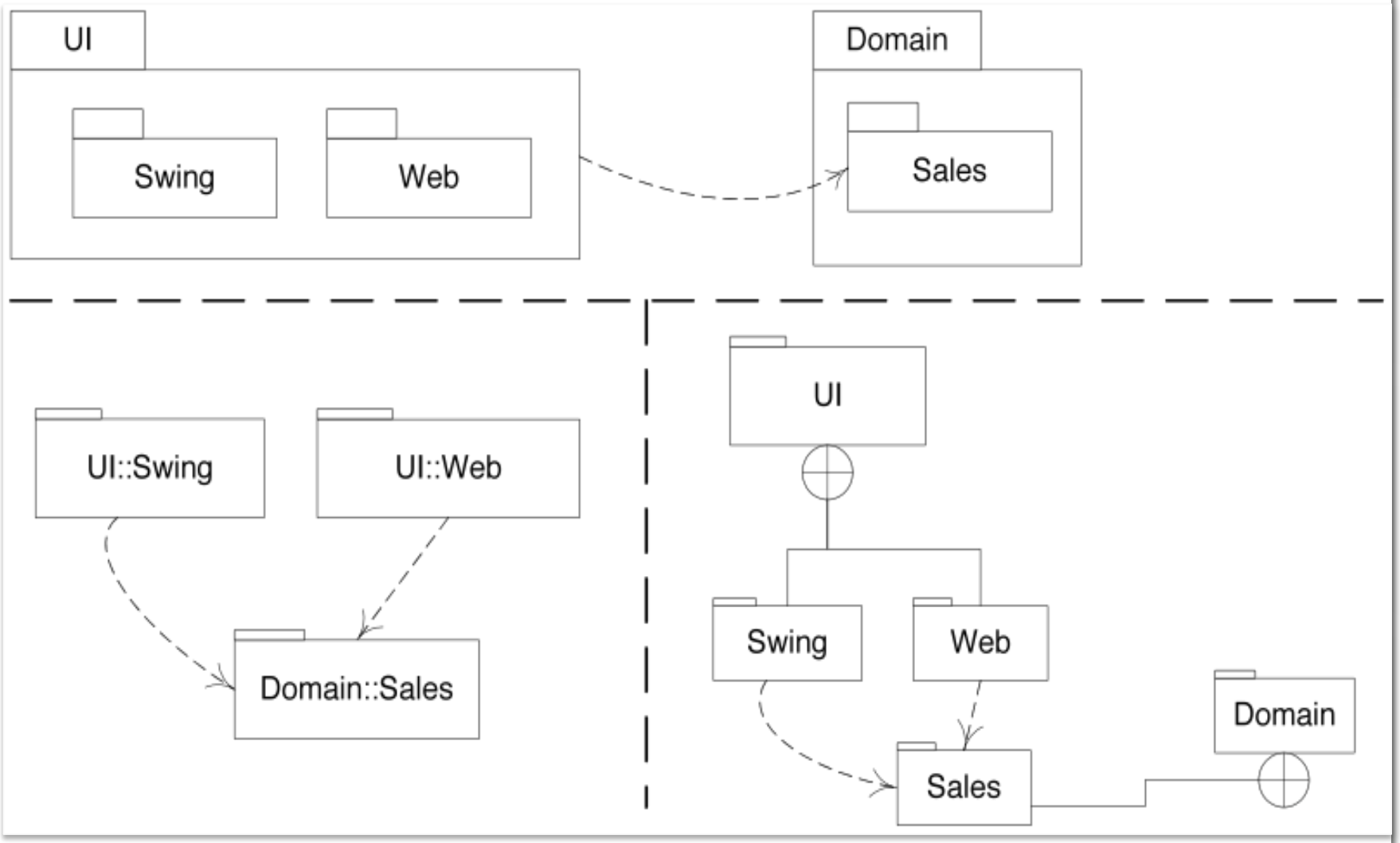
Dependency Line

Fully qualified name is: *Domain::Sales*



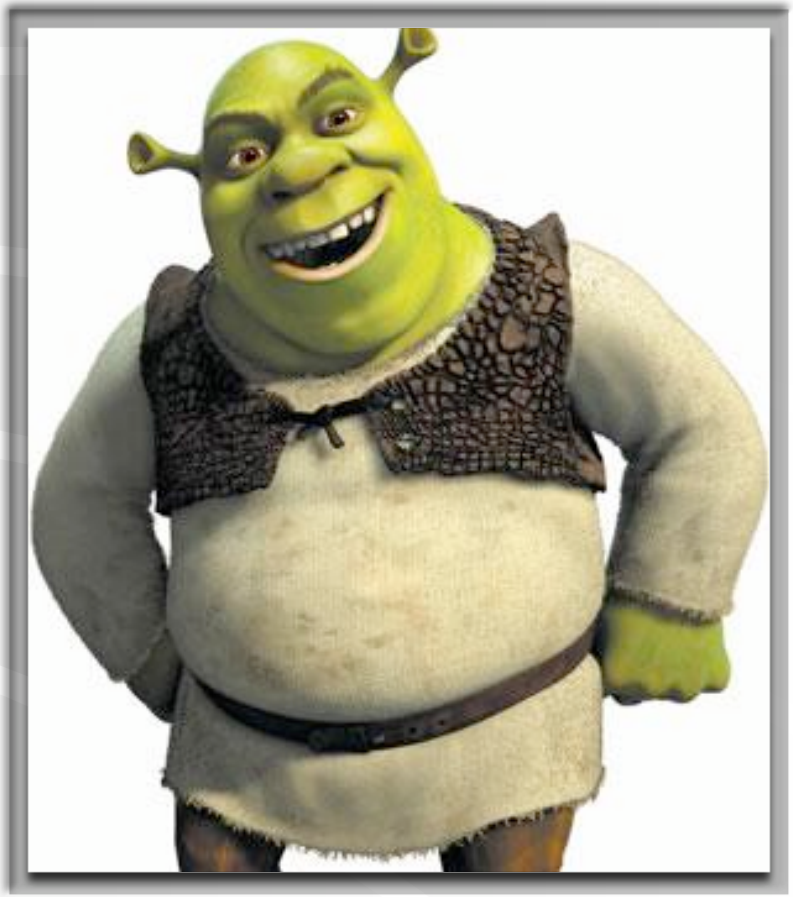
Alternative Nesting Notations

Traditional Notation



Designing with Layers Solves Problems

- Rippling source code changes
- Intertwining of application and UI logic
- Intertwining of application logic and technical services
- Difficult division of labor



Layers of Benefits

- **Separation** of concerns
  - Reduces coupling and dependencies; improves cohesion; increases reuse potential and clarity
- Essential **complexity** is encapsulated
- Can **replace** some layers with new implementations (e.g., platform independence)
- Can **distribute** some layers
- Can divide development within/across **teams**

Common Layers in More Detail (1 of 2)

GUI windows
reports
speech interface
HTML, XML, XSLT, JSP, Javascript, ...

→ **UI**
(AKA **Presentation**, View)

handles presentation layer requests
workfl ow
session state
window/page transitions
consolidation/transformation of disparate data for presentation

→ **Application**
(AKA Workfl ow, Process, Mediation, App Controller)

handles application layer requests
implementation of domain rules
domain services (*POS*, *Inventory*)
- services may be used by just one application, but there is also the possibility of multi-application services

→ **Domain**
(AKA Business, Application Logic, Model)

very general low-level business services used in many business domains
CurrencyConverter

→ **Business Infrastructure**
(AKA Low-level Business Services)

Common Layers in More Detail (2 of 2)

very general low-level business services used in many business domains
CurrencyConverter

Business Infrastructure
(AKA Low-level Business Services)

(relatively) high-level technical services and frameworks
Persistence, *Security*

Technical Services
(AKA Technical Infrastructure,
High-level Technical Services)

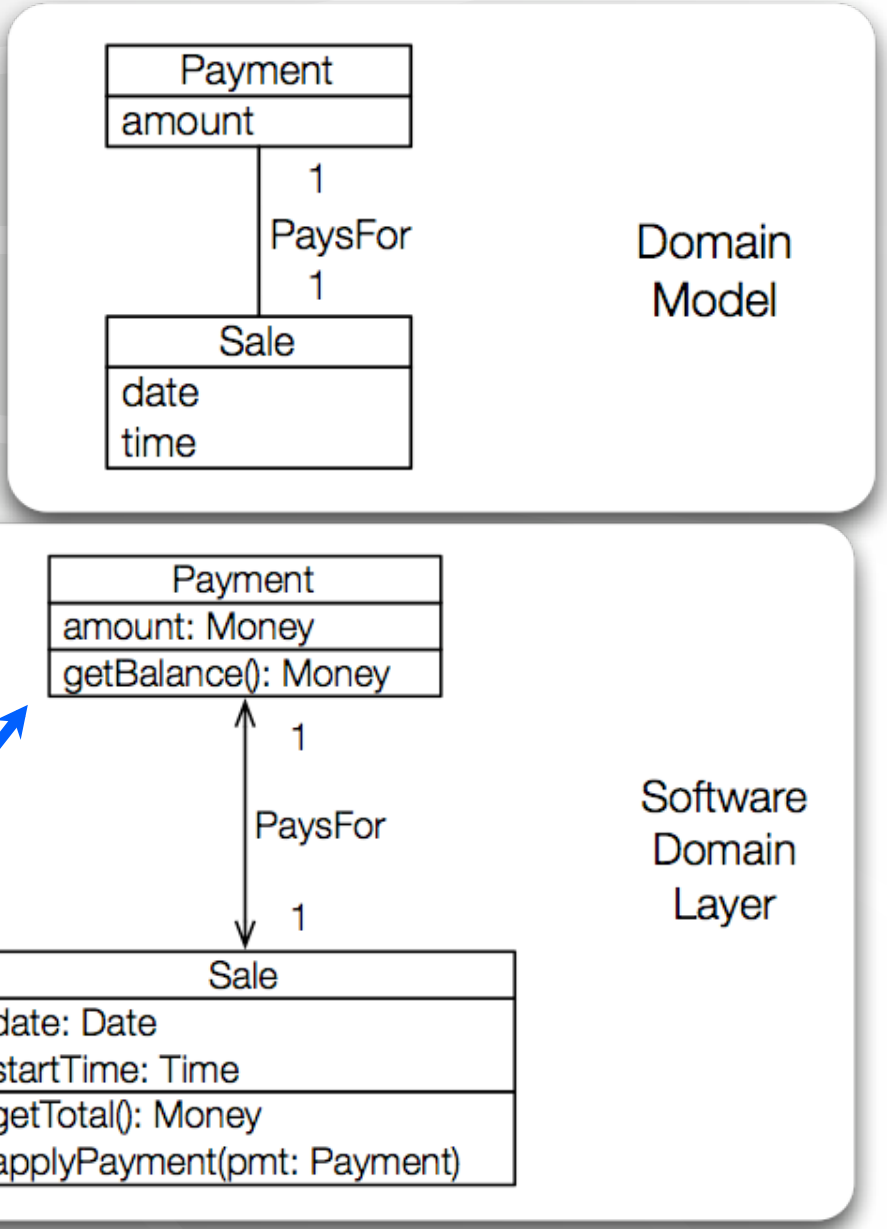
low-level technical services, utilities, and frameworks
data structures, threads, math, fi leDB, and network I/O

Foundation
(AKA Core Services, Base Services,
Low-level Technical Services/Infrastructure)

Systems will have many, but not necessarily all, of these

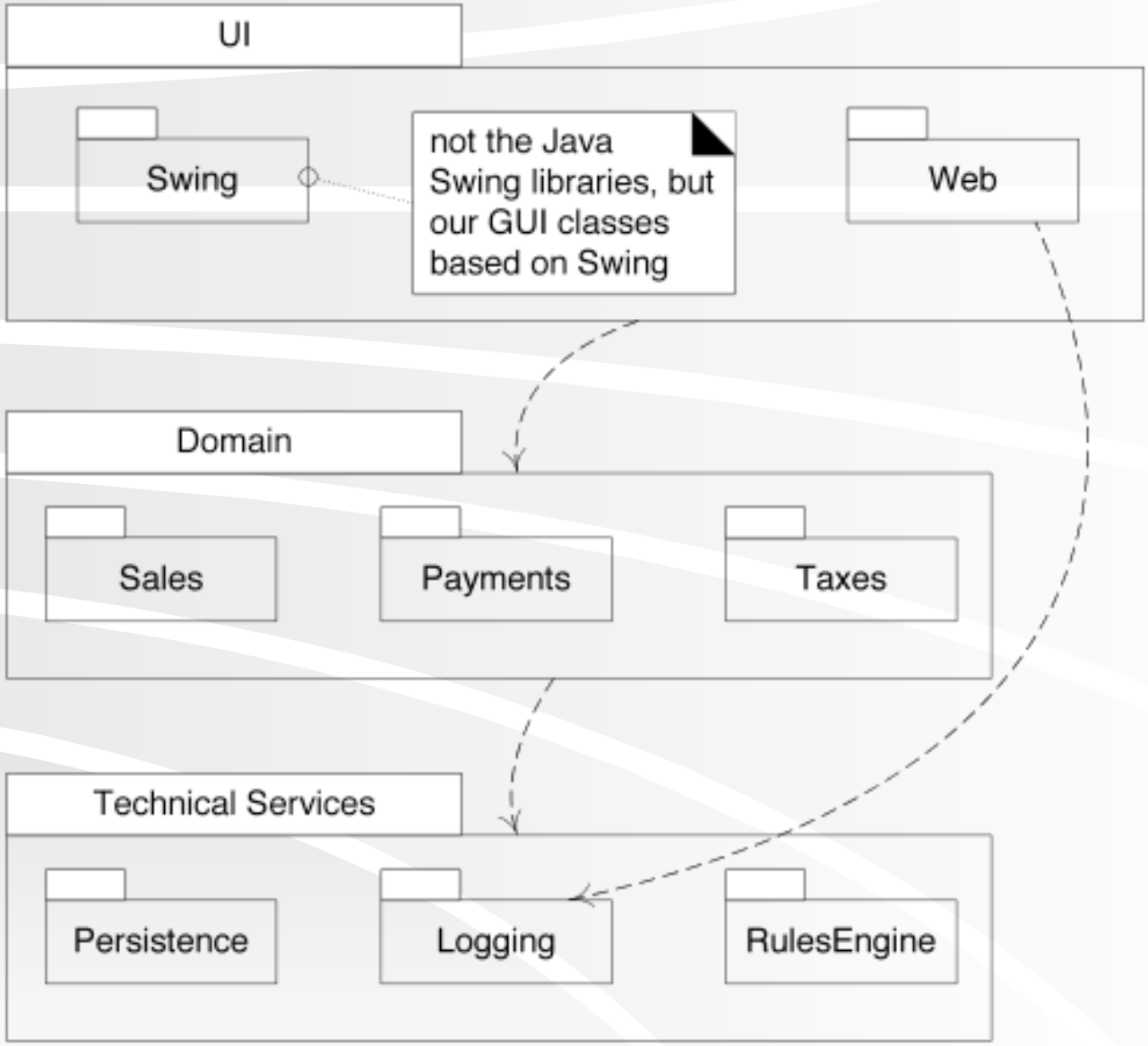
Designing the Domain Layer

- Create software objects with names and information similar to the real-world domain
- Assign application logic responsibilities

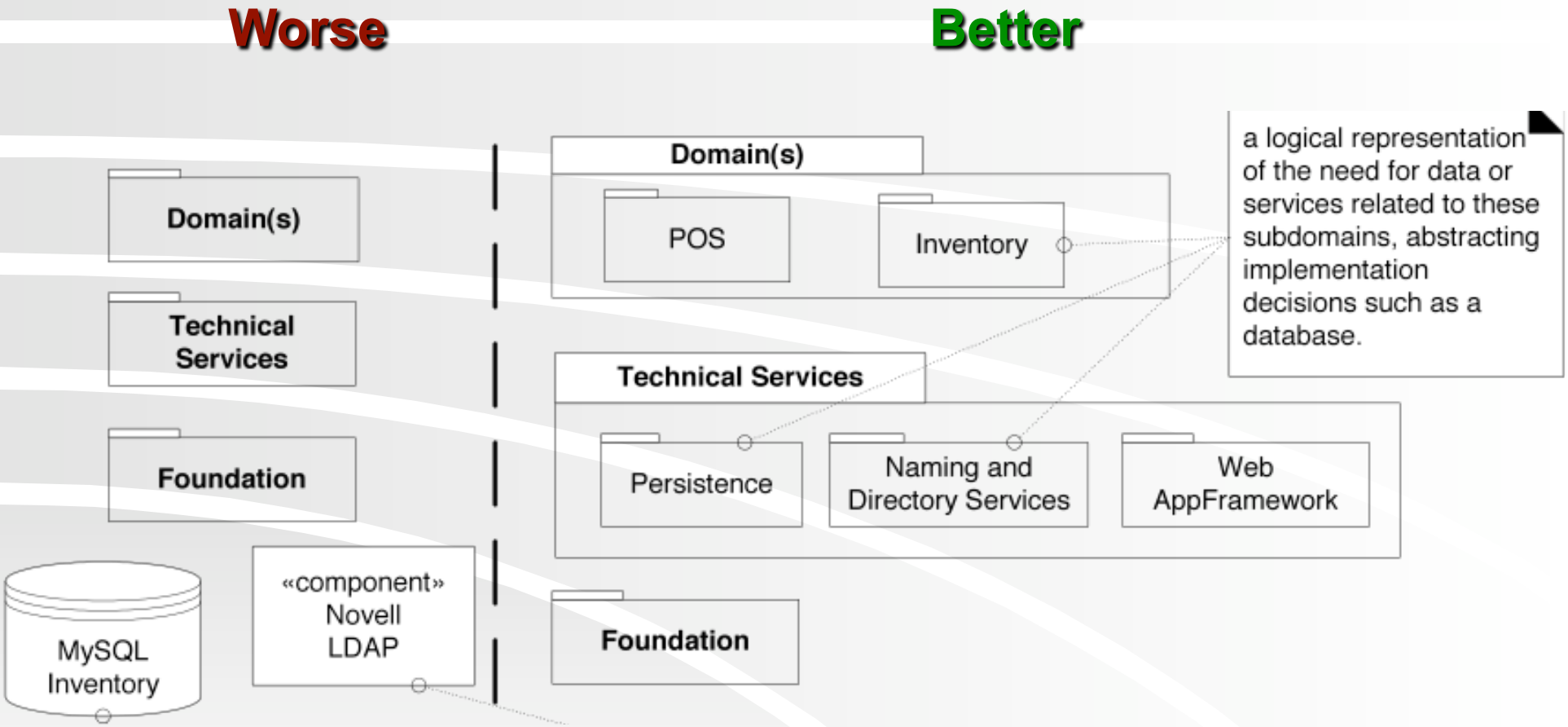


"Domain Objects"

Terminology: Layers vs. Partitions

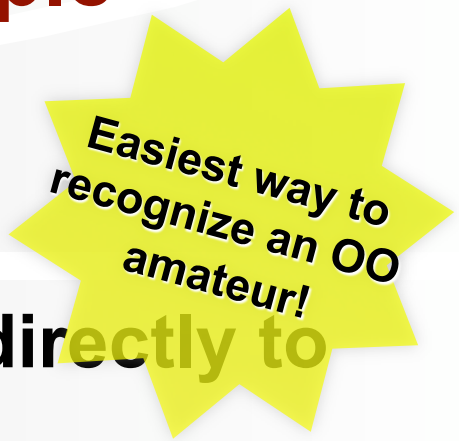


Common Mistake: Showing External Resources



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Model-View Separation Principle



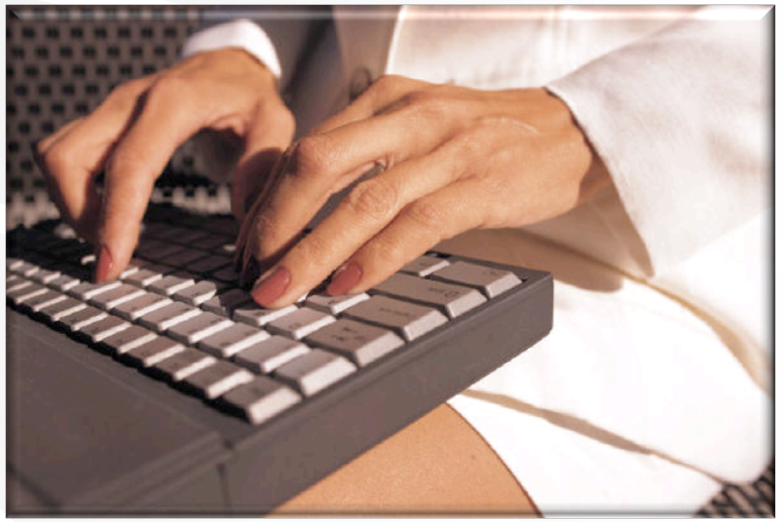
- Do not connect non-UI objects directly to UI objects
  - A Sale object shouldn't have a reference to a JFrame
- Do not put application logic in UI object methods
  - A UI event handler should just delegate to the domain layer
- Model == domain layer, View == UI layer

Benefits of Model-View Separation

- Provides cohesive model definitions
- Enables separate development
- Localizes changes to interface requirements
- Can add new views
- Allows simultaneous views
- Allows execution of model without UI

From SSDs to Layers

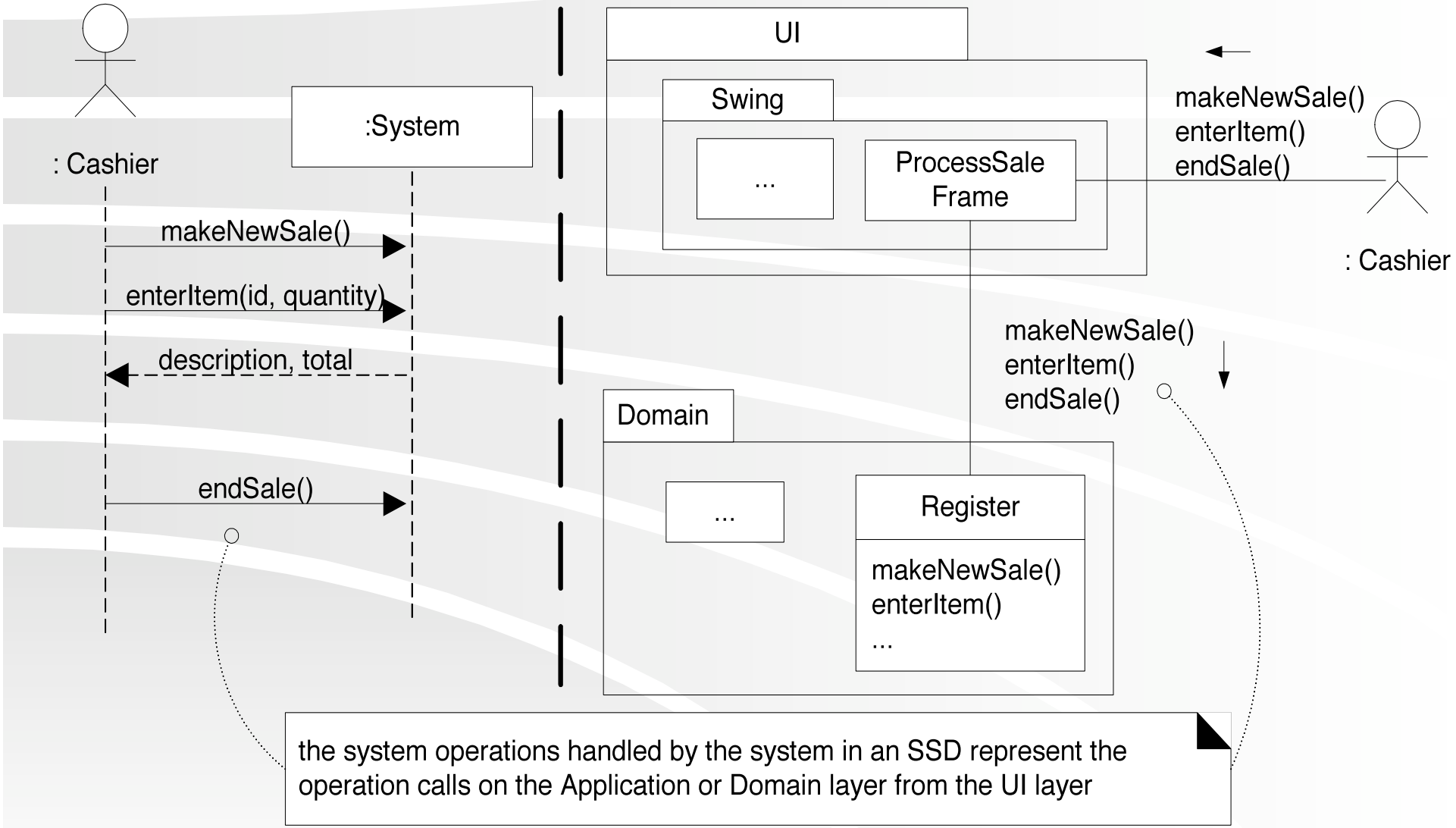
System operations on the SSDs will become the messages sent from the UI layer to the domain layer



Q9

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

SSDs in Layers



What's Next?

Techniques for Object Design!

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Common Object Design Techniques

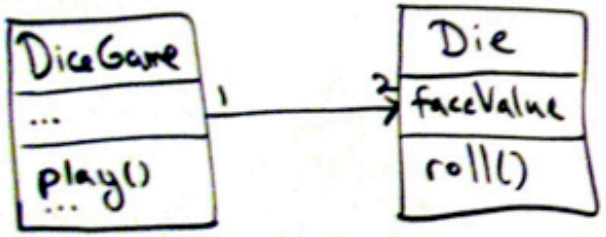
- **Just code it:** design while coding, heavy emphasis on refactoring and powerful IDEs
- **Draw, then code:** sketch some UML, then code it
- **Just draw it:** generate code from diagrams

http://www.virginmedia.com/movies/galleries/previews/indiana-jones-idols.php?ssid=7

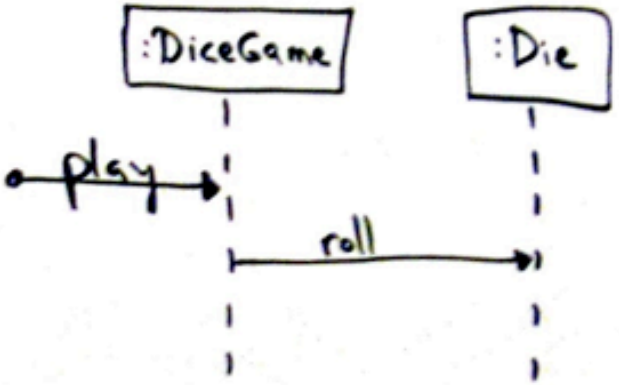
Static vs. Dynamic Modeling

- **Static models**
  - Class diagrams
- **Dynamic models**
  - Sequence diagrams
  - Communication diagrams

Interaction Diagrams



UML Class Diagram

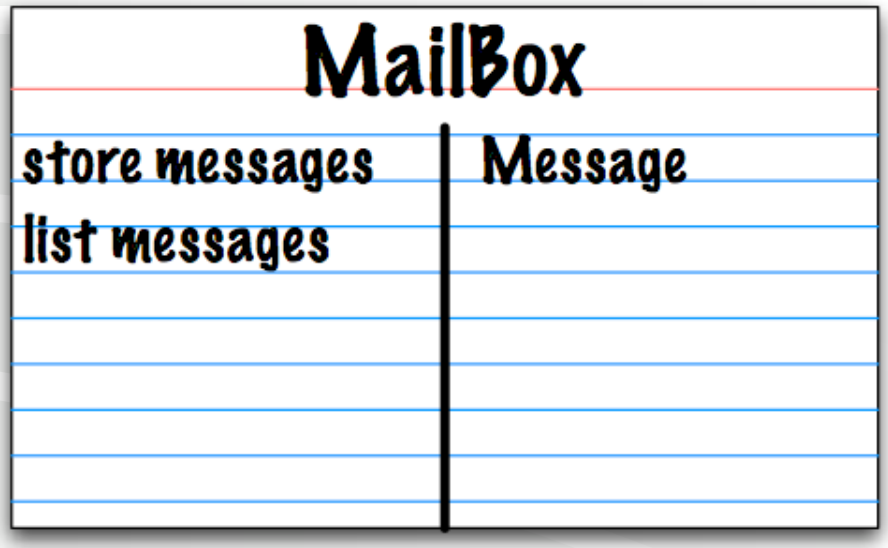


UML Sequence Diagram

Spend time on interaction diagrams, not just class diagrams

CRC Cards: A Text-based Technique

- Class
- Responsibilities
- Collaborators



Prefer Design Skill over UML skill

- UML is only a tool for object design
- The real skill is the design,
  …NOT the diagramming

- Fundamental object design requires knowledge of:
  - Principles of responsibility assignment
  - Design patterns

Homework and Milestone Reminders

- Read Chapter 15 on Interaction Diagrams
- Homework 3 – Dog-eDoctor SSDs and Operations Contracts
  - Due by 5:00pm on Tuesday, December 15th, 2009
- Milestone 3 – Iteration 1: Junior Project
  - Finish Analysis Model (SSDs, OCs)
  - Logical Architecture - Package Diagrams, and
  - 1st (initial) Version of System
  - Due by 11:59pm on Friday, January 8th, 2009