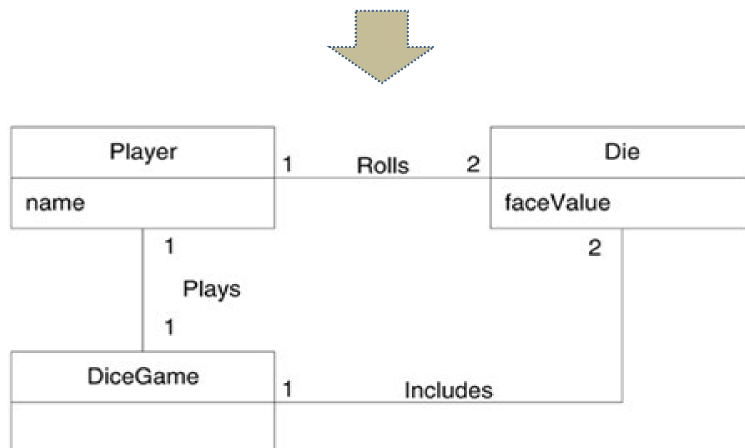
Object-Oriented Analysis and Design - Summary

An Short Example of OOAD - Dice Game

OOA

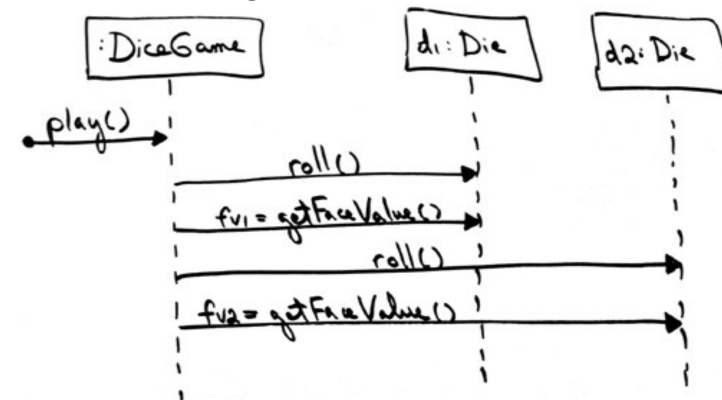
Use Case : Play a Dice Game
- Player requests to roll the dice.
- System presents results.
- If the dice's face value totals seven, player wins; otherwise, player loses.

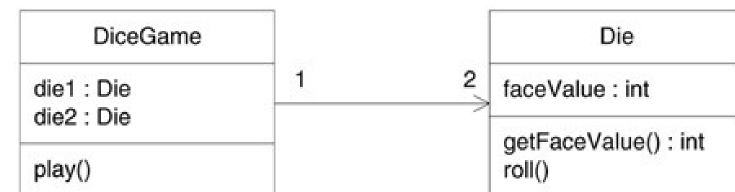


Domain Model

OOD

Interaction Diagram





Design Class Diagram

Software Development Process and the UP

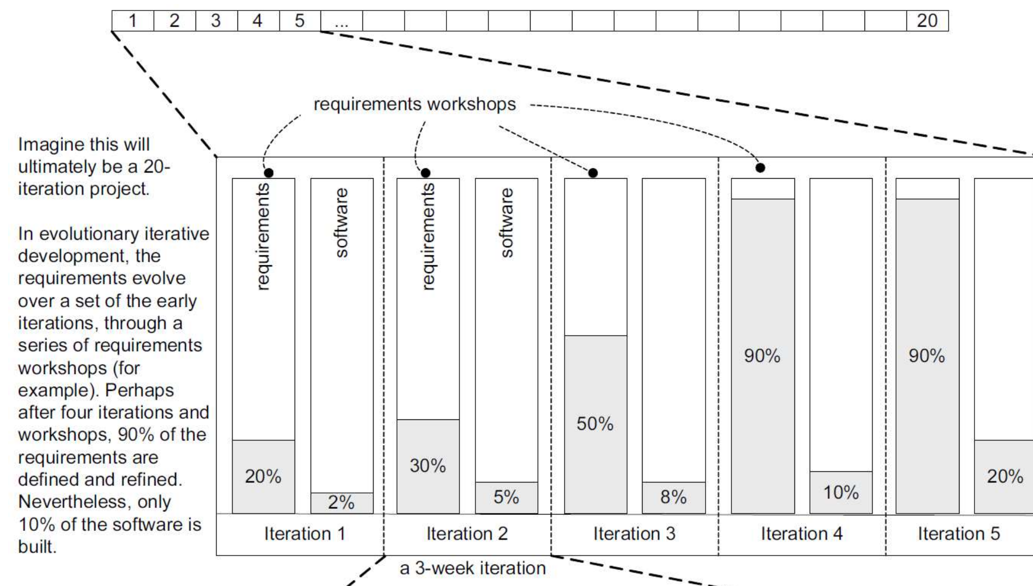
- **Software development process**
  - A **systematic approach** to building, deploying and possibly maintaining software

- **Unified Process (UP)**: a popular iterative software development process for building object-oriented systems
  - Inspired from Agile
  - Iterative
  - Provides an example structure for how to do OOA/D
  - Flexible (can be combined with practices from other OO processes)
  - A de-facto industry standard for developing OO software

Risk-Driven and Client-Driven Iterative Planning

- The **UP** encourages a combination of **risk-driven** and **client-driven** **iterative planning**.
  - To identify and drive down the high risks, and
  - To build visible features that clients care most about.

- **Risk-driven** iterative development includes more specifically the practice of **architecture-centric** iterative development.
  - Early iterations focus on building, testing, and stabilizing the core architecture.

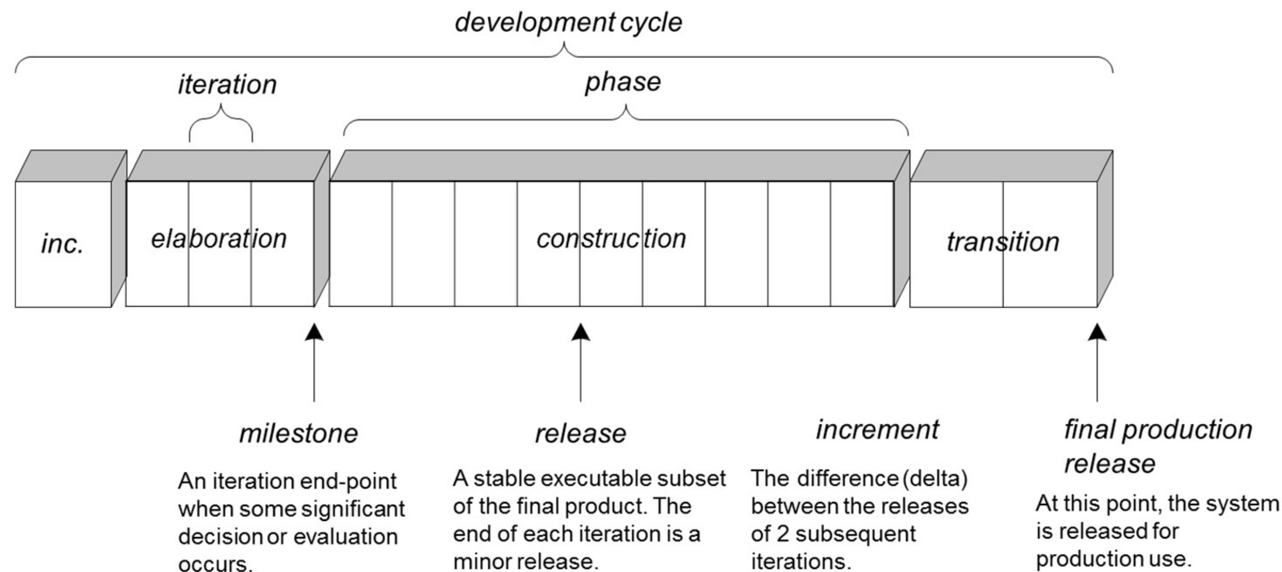


DEPENDABLE SOFTWARE LABORATORY

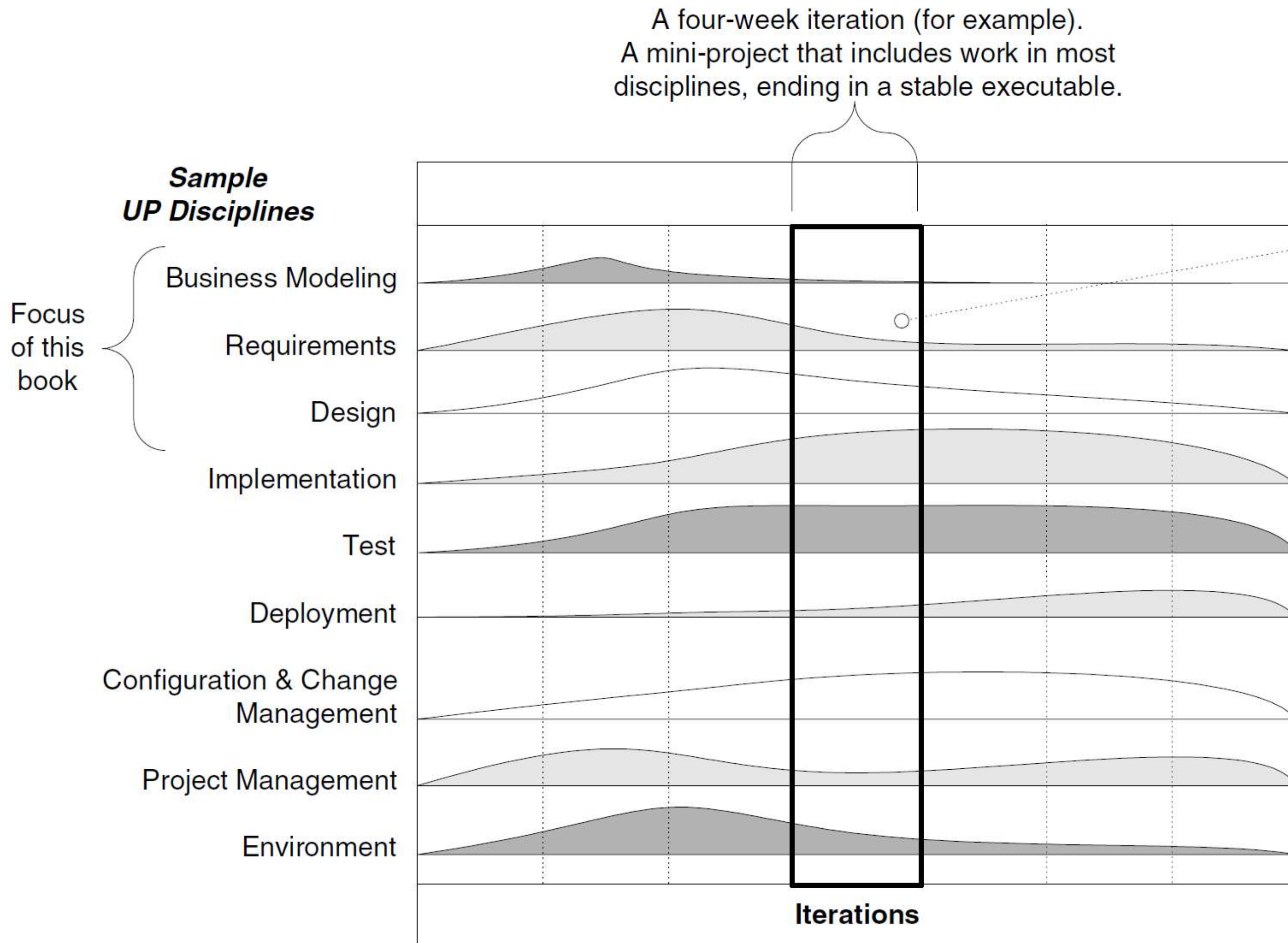
The UP Phases

KU KONKUK UNIVERSITY

- A UP project organizes the work and iterations across **4 major phases**:
  1. **Inception** : approximate vision, business case, scope, vague cost estimates
  2. **Elaboration** : refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates
  3. **Construction** : iterative implementation of the remaining lower risk and easier elements, and preparation for deployment
  4. **Transition** : beta tests, deployment



The UP Disciplines

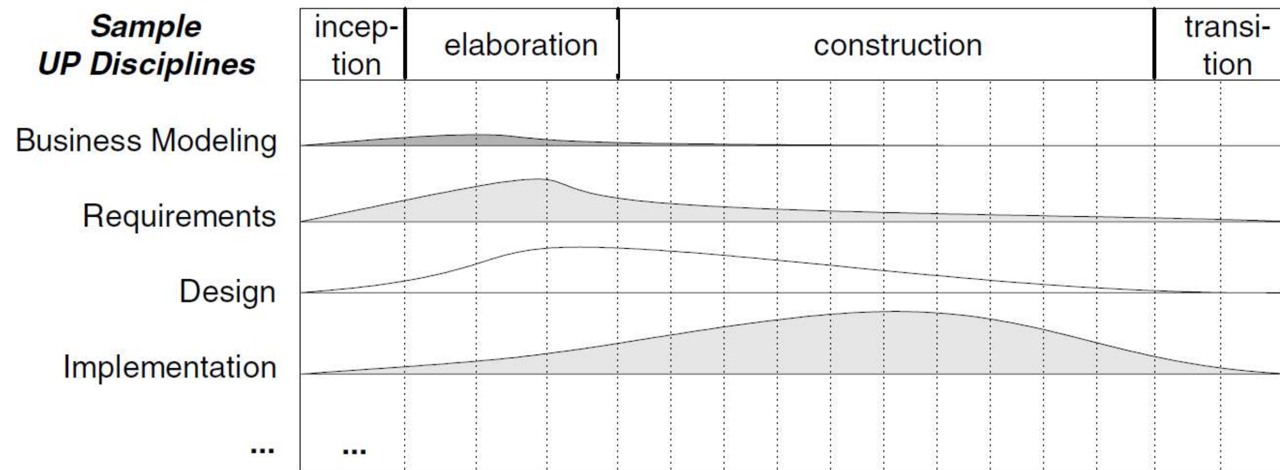


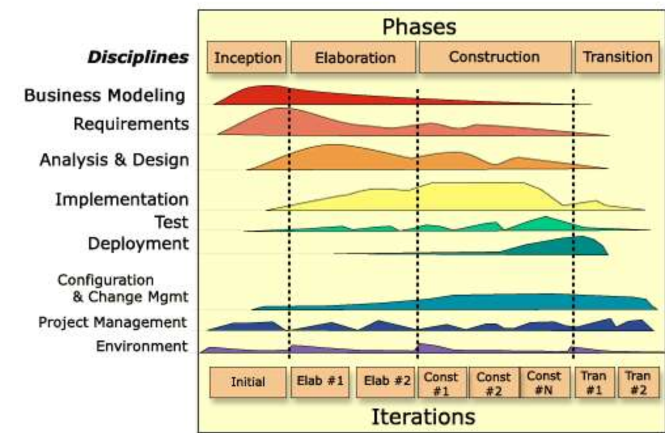
Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.

This example is suggestive, not literal.

Relationship Between the Disciplines and Phases

- The relative effort in disciplines shifts to across the phases.





The UP Artifacts and Timing

Sample Unified Process Artifacts and Timing (s-start; r-refine)

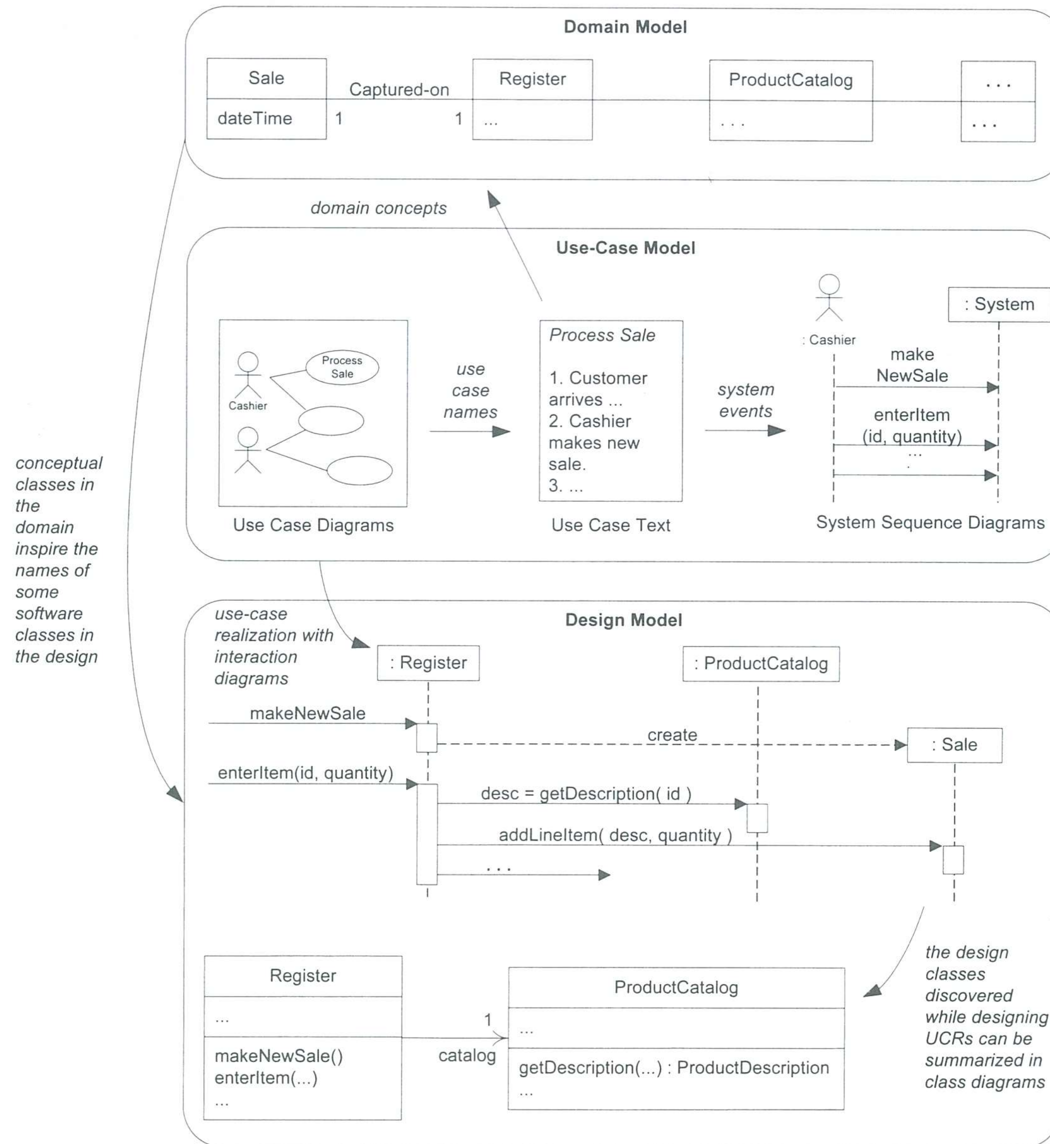
Discipline	Artifact Iteration➜	Incep. I1	Elab. E1..En	Const. C1..Cn	Trans. T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	s	r		
	Vision	s	r		
	Supplementary Specification	s	r		
	Glossary	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		
	Data Model		s	r	
Implementation	Implementation Model (code, html, ...)		s	r	r

+ System Sequence Diagram
+ Operation Contract

Design Model
  + Class Diagram
  + Interaction Diagram
  + Package Diagram

+ Statechart Diagram
+ Activity Diagram
+ Deployment Diagram

The UP Artifact Relationships

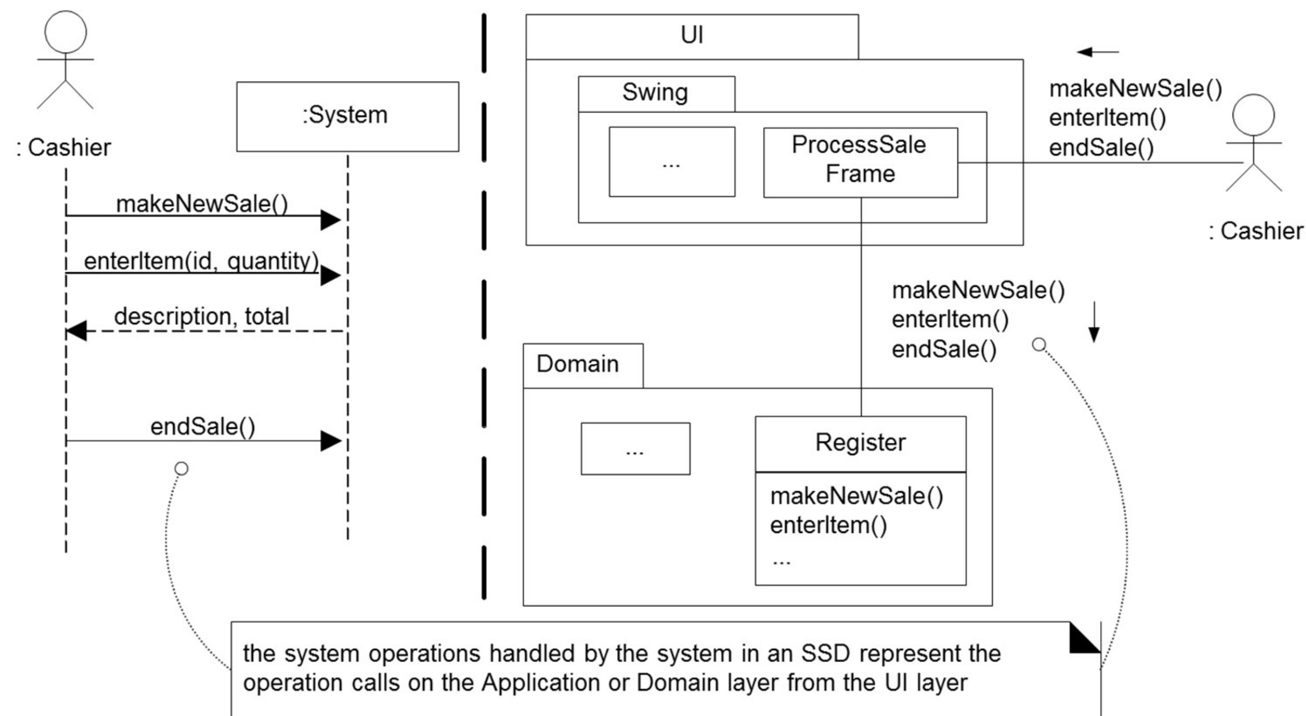


DEPENDABLE SOFTWARE LABORATORY

KU KONKUK UNIVERSITY

Connections Between SSDs, System Operations, and Layers

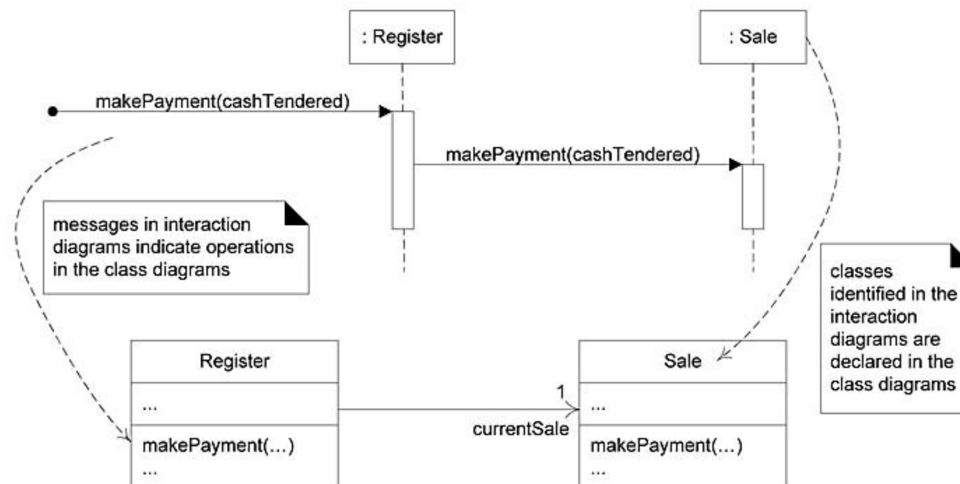
- In a well-designed layered architecture,
  - The UI layer objects will forward or delegate the requests from the UI layer (system operations) onto the domain layer for handling.
  - The messages sent from the UI layer to the domain layer will be the messages illustrated on the SSDs.



What's the Relationship between Interaction and Class Diagrams?

- From interaction diagrams, class diagrams can be generated iteratively.
  - When we draw interaction diagrams, a set of classes and their methods emerge.
  - Suggests a linear ordering of drawing interaction diagrams before class diagrams.
  - But in practice, these complementary dynamic and static views are drawn concurrently or iteratively.

  - Example:
    - if we started with the *makePayment* sequence diagram, we see that a *Register* and *Sale* class definition in a class diagram can be obviously derived.



OOD : Object-Oriented Design

- **OOD** is sometimes taught as some variation of the following:
  - "*After identifying your requirements and creating a domain model, then add methods to the appropriate classes, and define the messaging between the objects to fulfill the requirements.*"
- But, it is not enough, because OOD involves **deep principles**.
  - Deciding what methods belong to where and how objects should interact carries consequences should be undertaken seriously.
- Mastering OOD is hard.
  - Involving a large set of soft principles, with many degrees of freedom.
  - A mind well educated in design principles is important.
  - **Patterns** can be applied.

GRASP

- **<u>9 basic OO design principles</u>** or basic building blocks in design.
  - Focusing on using the pattern style as an excellent learning aid for naming, presenting and remembering basic/classic design ideas

  - Creator
  - Controller
  - Pure Fabrication
  - Information Expert
  - High Cohesion
  - Indirection
  - Low Coupling
  - Polymorphism
  - Protected Variations

Pattern/ Principle	Description
Information Expert	A general principle of object design and responsibility assignment? Assign a responsibility to the information expert—the class that has the information necessary to fulfill the responsibility.
Creator	Who creates? (Note that Factory is a common alternate solution.) Assign class B the responsibility to create an instance of class A if one of these is true: 1. B contains A 2. B aggregates A 3. B has the initializing data for A 4. B records A 5. B closely uses A
Controller	What first object beyond the UI layer receives and coordinates ("controls") a system operation? Assign the responsibility to an object representing one of these choices: 1. Represents the overall "system," a "root object," a device that the software is running within, or a major subsystem (these are all variations of a *facade controller*). 2. Represents a use case scenario within which the system operation occurs (a use-case or *session controller*)
Low Coupling (evaluative)	How to reduce the impact of change? Assign responsibilities so that (unnecessary) coupling remains low. Use this principle to evaluate alternatives.

23 Design Patterns of GoF

KU KONKUK UNIVERSITY

C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

Memento
Type: Behavioral
What it is: Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Chain of Responsibility
Type: Behavioral
What it is: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Observer
Type: Behavioral
What it is: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Command
Type: Behavioral
What it is: Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

State
Type: Behavioral
What it is: Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

Interpreter
Type: Behavioral
What it is: Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Strategy
Type: Behavioral
What it is: Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.

Iterator
Type: Behavioral
What it is: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Template Method
Type: Behavioral
What it is: Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Mediator
Type: Behavioral
What it is: Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

Visitor
Type: Behavioral
What it is: Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.

Copyright © 2007 Jason S. McDonald http://www.McDonaldLand.info

Adapter
Type: Structural
What it is: Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

Proxy
Type: Structural
What it is: Provide a surrogate or placeholder for another object to control access to it.

Bridge
Type: Structural
What it is: Decouple an abstraction from its implementation so that the two can vary independently.

Abstract Factory
Type: Creational
What it is: Provides an interface for creating families of related or dependent objects without specifying their concrete class.

Composite
Type: Structural
What it is: Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions of objects uniformly.

Builder
Type: Creational
What it is: Separate the construction of a complex object from its representing so that the same construction process can create different representations.

Decorator
Type: Structural
What it is: Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.

Factory Method
Type: Creational
What it is: Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.

Facade
Type: Structural
What it is: Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

Prototype
Type: Creational
What it is: Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Flyweight
Type: Structural
What it is: Use sharing to support large numbers of fine grained objects efficiently.

Singleton
Type: Creational
What it is: Ensure a class only has one instance and provide a global point of access to it.

Copyright © 2007 Jason S. McDonald http://www.McDonaldLand.info

Mapping Designs to Code

The Register.enterItem Method

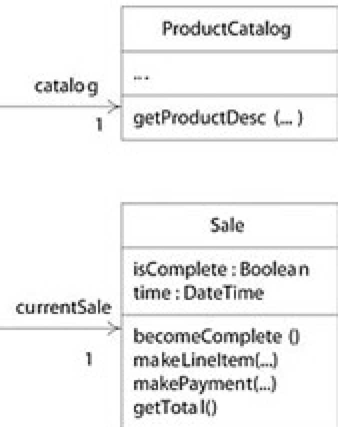
```
public class Register
{
private ProductCatalog catalog;
private Sale currentSale ;

public Register(ProductCatalog pc ) {... }

public  void endSale () {... }
public  void enterItem(ItemID id, int qty) {...  }
public void makeNewSale () {... }
public  void makePayment(Money cashTendered  ) {.. .}
}
```

enterItem()

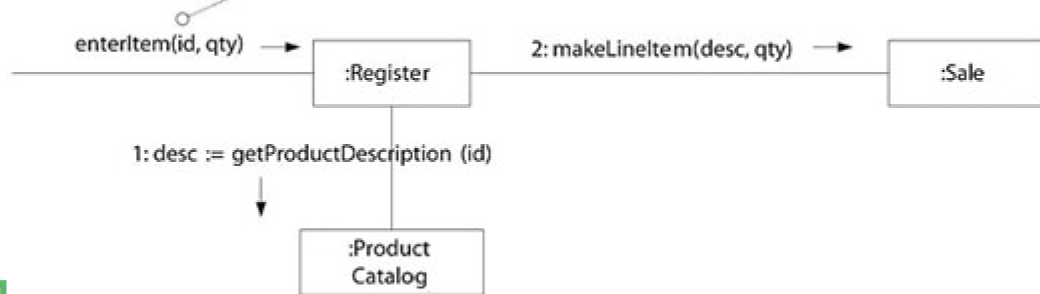
Register

...

endSale()
enterItem(id: ItemI D, qty : Integer)
makeNewSal e()
makePayment (cashTendered : Money)




```
{
  ProductDescription desc = catalog.ProductDescription (id);
  currentSale.makeLineItem(desc, qty)  ;
}
```



An Overview of Object-Oriented Development - What We Covered?

Software Development Sequence

Software Architecture Style

Object-Oriented Design Patterns

Object-Oriented Analysis and Design

Object-Oriented Concepts and Principles

Object-Oriented Programming

UP

UML + OOD Methods

Origination Sequence