Universität Zürich[UZH]

Institut für Informatik

Software Engineering HS'16

Lecture: Software Design

Thomas Fritz & Martin Glinz

Many thanks to Philippe Beaudoin, Gail Murphy, David Shepherd, Neil Ernst, Meghan Allen, and Elisa Baniassad

High Level Overview of the Design Unit

- Introduction to Design
- Architectural Design
- Detailed Design
- Modular Design / Design Principles
- Design Patterns

Learning Goals

By the end of this unit, you will be able to:

- Describe the context (goals and constraints) of the activity of software design and the process for developing it
- Define what is meant by “architectural style” and describe characteristics of main styles
- Understand the use of diagrams in software development
- Create a design for a given system and specify it in correct UML class/sequence diagram syntax

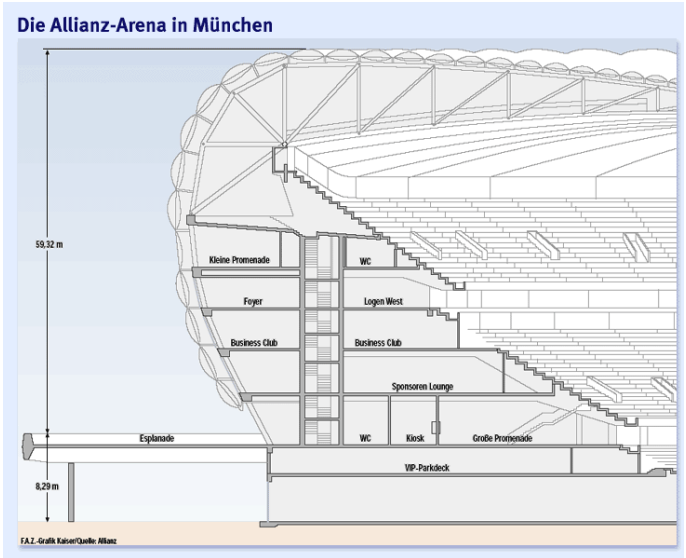
What is Clothing Design?

Picture from www.arcteryx.com

- Why might a designer decide to design such a jacket?
- What might have influenced the designer?

What is Building Design?

- Münchner Allianz Arena, built for FC Bayern and TSV1860 (2002-2005)
- Inputs?
- Constraints?



Picture from http://de.academic.ru/pictures/dewiki/65/Allianzarenacombo.jpg and www.faz.net

What is software design?

Requirements specification was about **WHAT** the system will do

Design is about **HOW** the system will perform its functions

What is design?

What is design? What makes something a design problem? It's where you stand with ***a foot in two worlds*** *– the world of technology and the world of people and human purposes – and you try to bring the two together.*

- Mitchel Kapor, A Software Design Manifesto (1991)

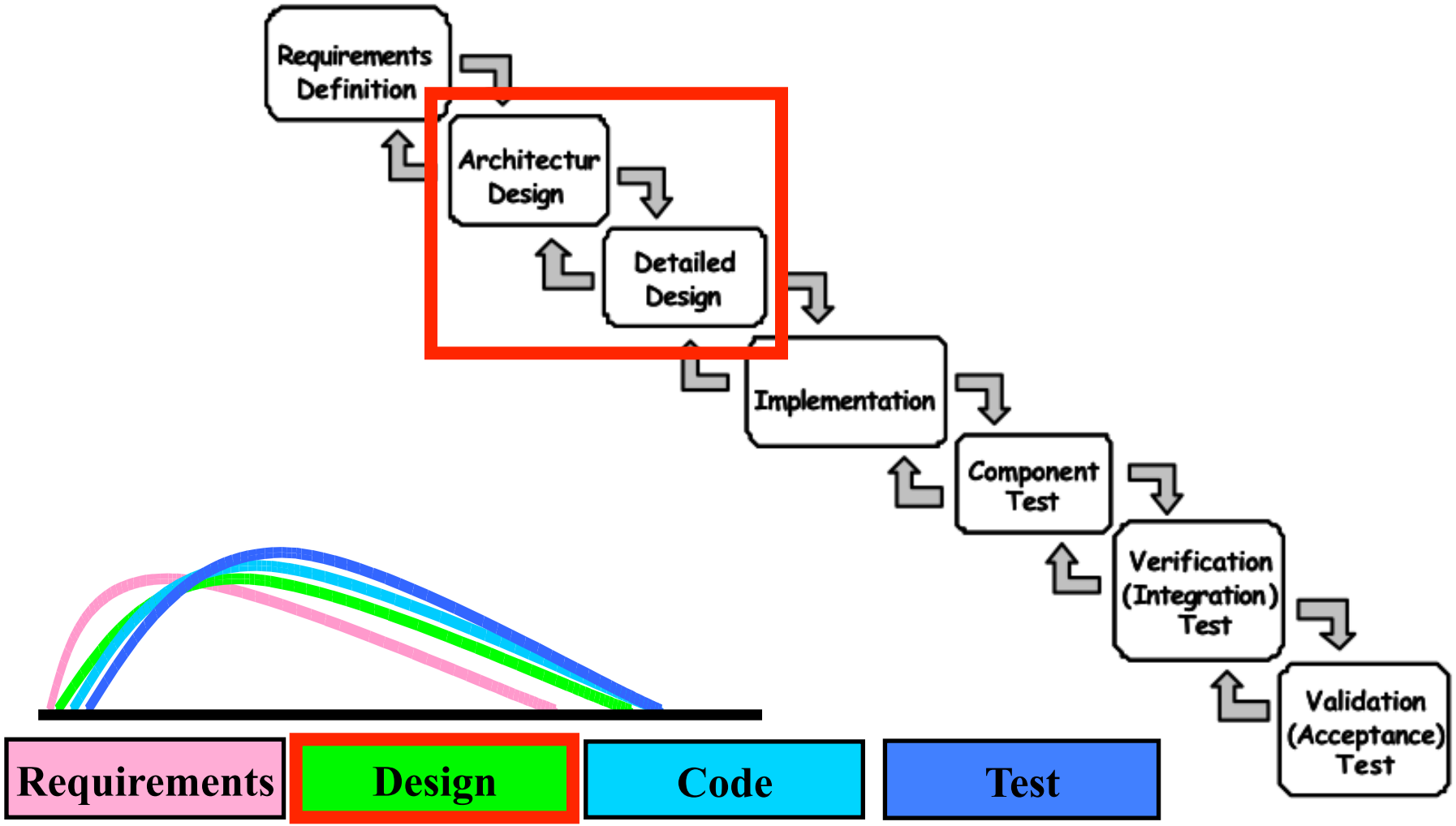
Kapor goes on to say…

Design disciplines are concerned with making artifacts for human use. Architects work in the medium of buildings, graphic designers work in paper and other print media, industrial designers on mass-produced manufactured goods, and software designers on software. ***The software designer should be the person with overall responsibility for the conception and realization of the program.***

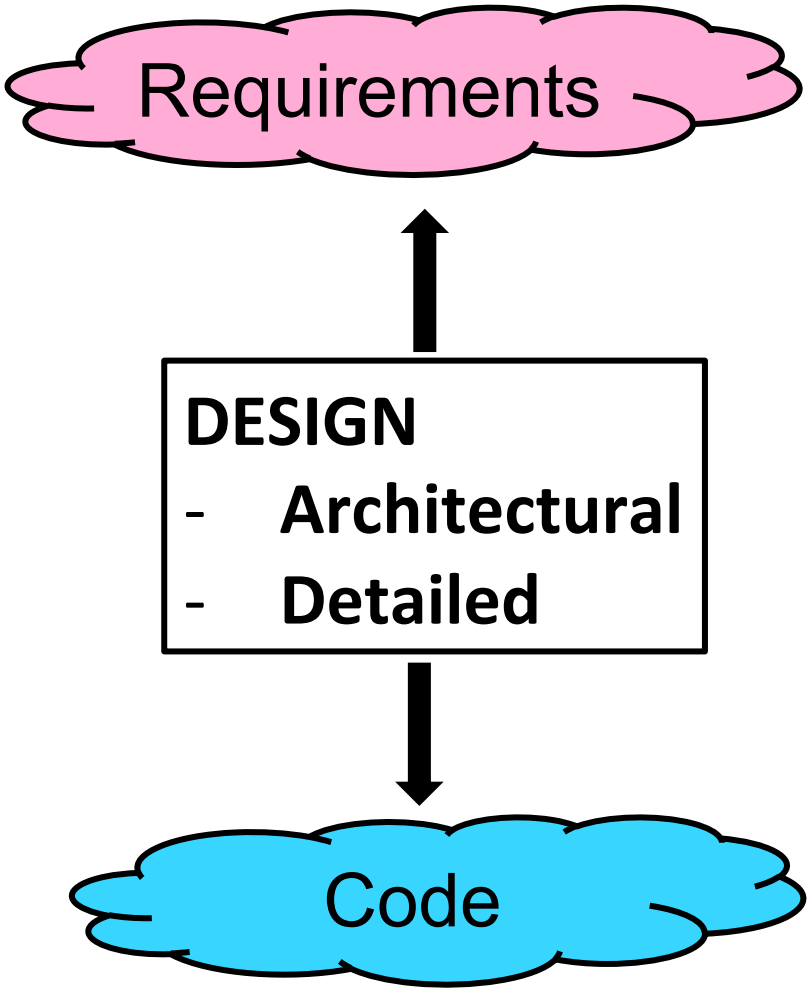
Software Design in this course

- Mainly focusing on technological (developer/ engineer) view of software design
- How do we realize the conceived product?
- Inputs include requirements (functional and non-functional), developer's experience
- Constraints include development organization, technical platform
- Goals: decomposition and determination of relationships, communication and more

Where does it fit in the process?



Design to Bridge the Gap



Why Design?

Facilitates communication

Eases system understanding

Eases implementation

Helps discover problems early

Increases product quality

Reduces maintenance costs

Facilitates product upgrade

Cost of not planning…

Another example of poor planning

How to approach Design?

*"Treat design as a wicked, sloppy, **heuristic** process. Don't settle for the first design that occurs to you. **Collaborate.** Strive for **simplicity**. Prototype when you need to. Iterate, iterate and **iterate again**. You'll be happy with your designs."*

McConnell, Steve. *Code Complete*. Ch. 5

How to approach Design?

Study and understand the problem from different viewpoints

Identify potential solutions and evaluate the trade-offs

Develop different models of system at different levels of abstraction: start global, subdivide (top-down), iterate (design is often a combination of top-down and bottom-up)

Two common phases of Software Design

Architectural design

- Overall structure: main components and their connections; determining which sub-systems you need (e.g., web server, DB…)

Detailed design

- Inner structure of main components
- Take programming language into account

Software Architecture

*The **fundamental concepts** or properties of a system in its environment embodied in its **elements**, **relationships**, and in the principles of its **design** and **evolution**.*

IEEE Standard 1471-2011

*The structure or **structures of the system**, which comprise software **elements**, the **externally visible properties** of those elements and the **relationships** among them.*

Software Architecture in Practice (2nd edition), Bass, Clements, Kazman

Software Architecture

A software architectures for a system describes

- Subsystems and components that comprise the system
  (client/server, web service, software package, …)
- Overall structure of those components and subsystems
  (e.g. pipe and filter, blackboard, MVC, …)
- Connectors
  (interactions and rules that govern interactions, e.g. client-server network protocol, procedure calls)
- Constraints
  (environmental constraints, quality attributes or non-functional requirements)

Architectural Styles & Patterns

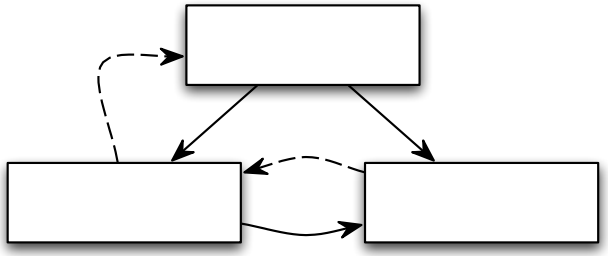
Architectural styles constrain architectural design decisions and dictate qualities the system will have

- e.g., modifiable? secure? scalable? reliable? Etc.

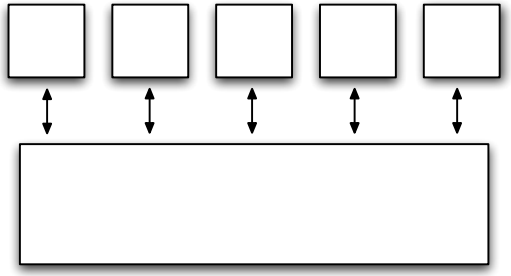
Architectural style is a name given to a common architectural design. Architectural pattern is a way of solving a common architectural problem. (sometimes used interchangeably)

Both provide common language of software architecture

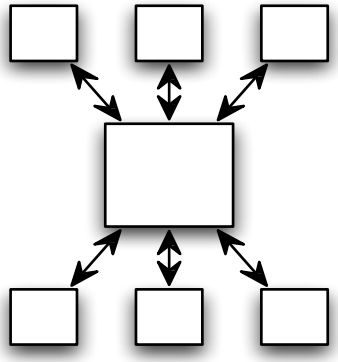
Common Architectural Styles & Patterns



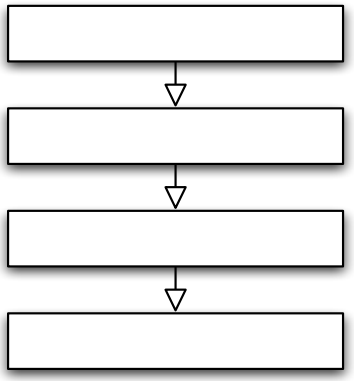
model view controller



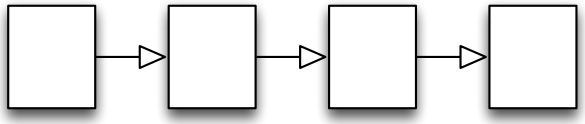
blackboard



client/server

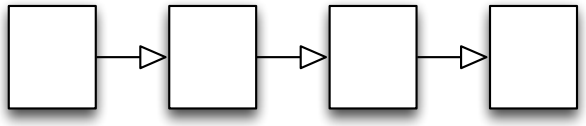


layered



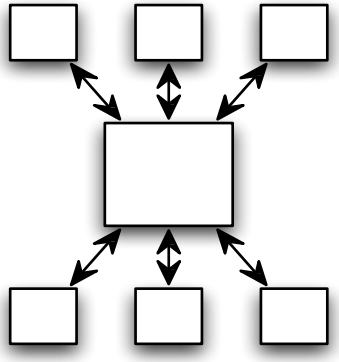
pipe-and-filter

Pipe-and-Filter



Components:	**filters** that read input data stream and transform it into output data stream
Connectors:	**pipes** that provide output of filter as input to other filter
Advantages:	simple, no complex interaction, high reusability, portability
Disadvantages:	require common data format, no shared state, redundancy in (un)parsing
Example:	unix shell (ls –l | grep key | more …)

Client/Server



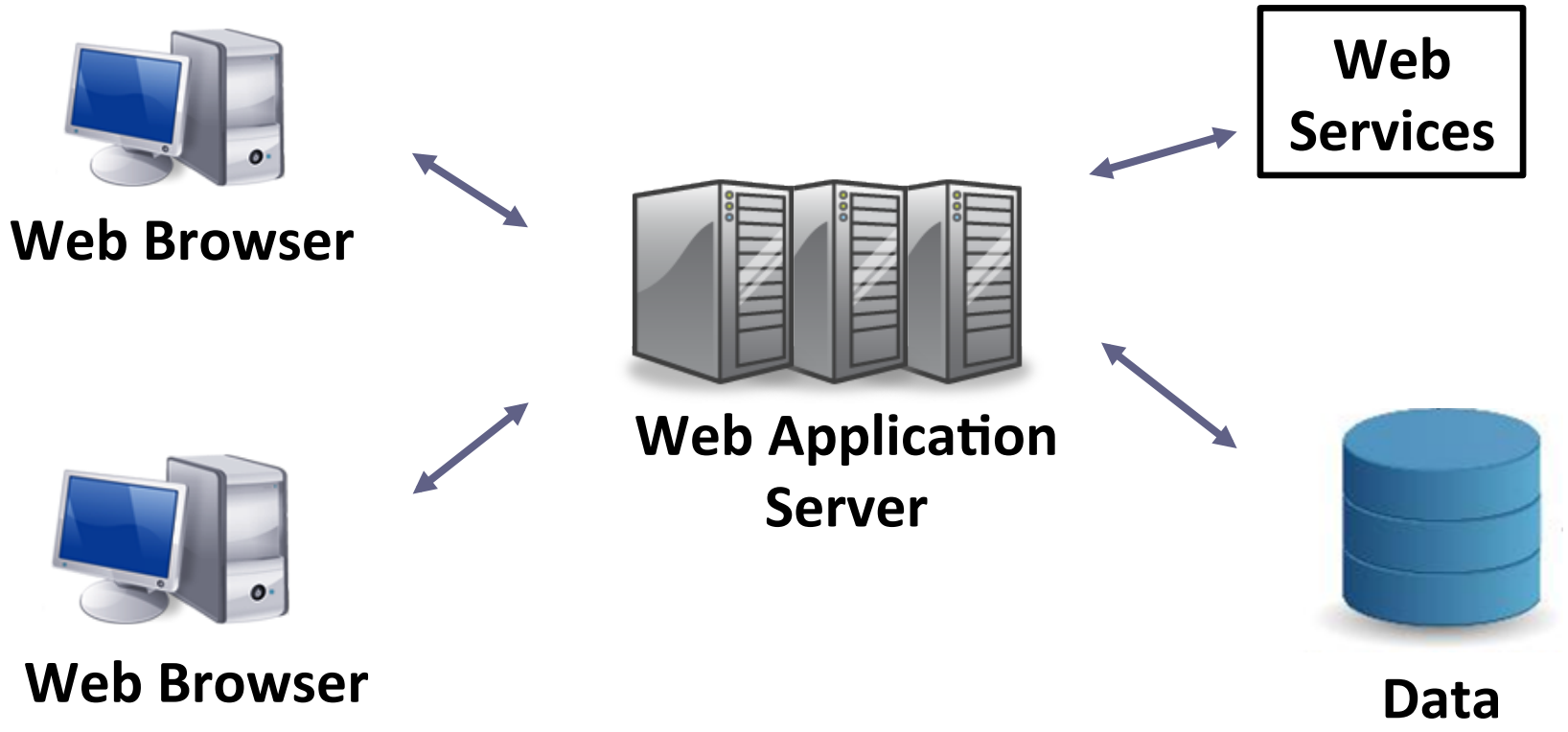
Components: **server** subsystem provides services to multiple instances of **client** subsystem;

Connectors: network; client typically request services from server

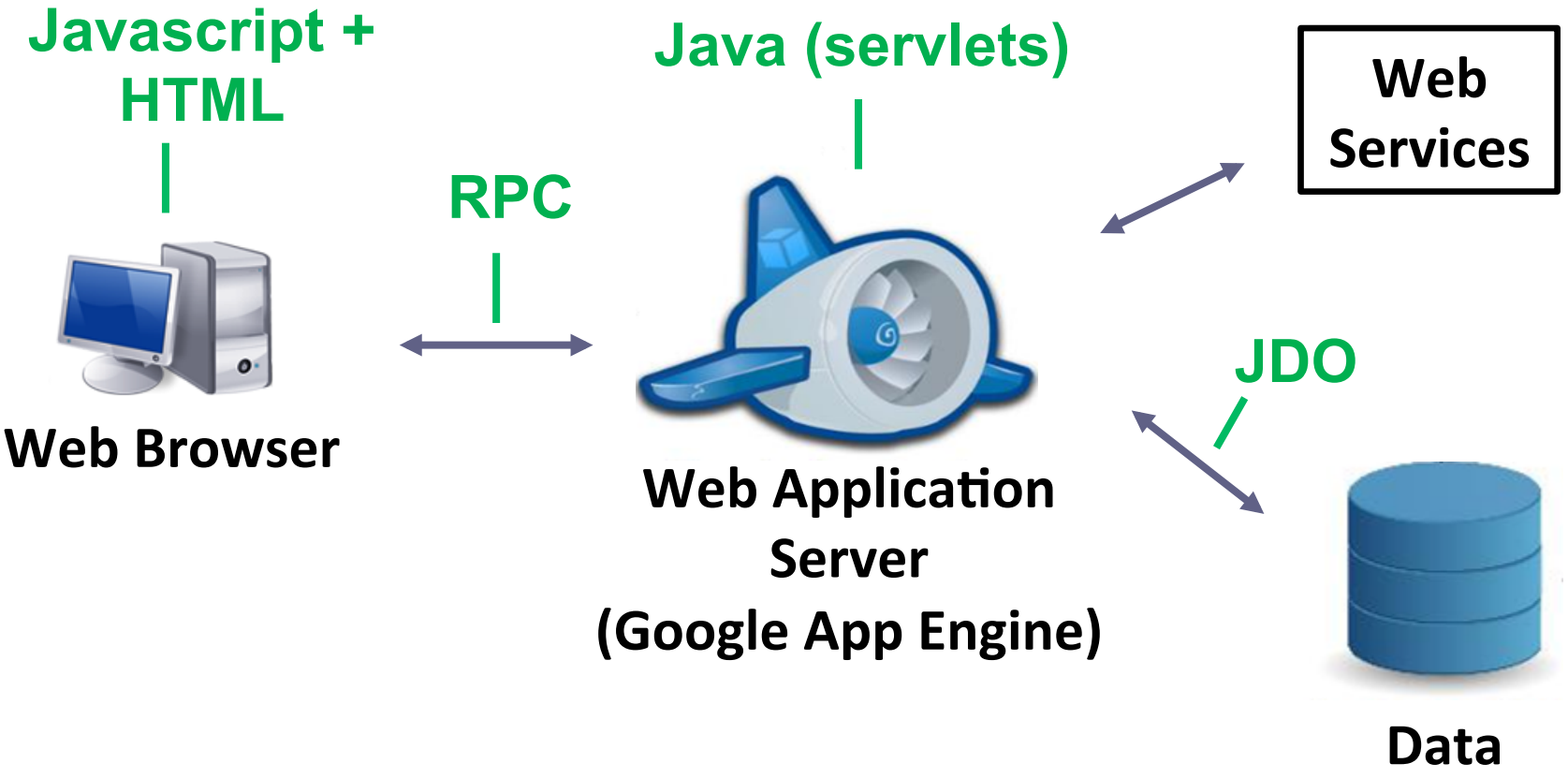
Advantages: distribution, scalability

Disadvantages: responsiveness (if network is slow), robustness (if server goes down)

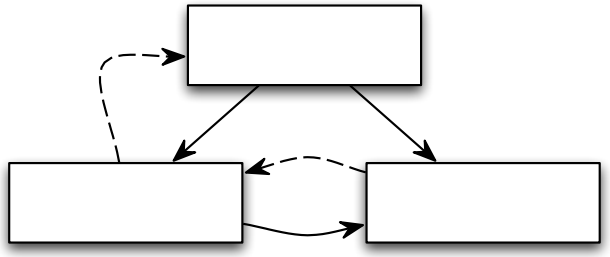
Web Architecture (Client / Server Style)



Google App Engine



Model View Controller



Components: **model** contains core functionality and data, **views** display information to the user, and **controllers** handle user input

Connectors: change-propagation mechanism (observer)

Advantages: interactivity, expandability, separation of model vs presentation

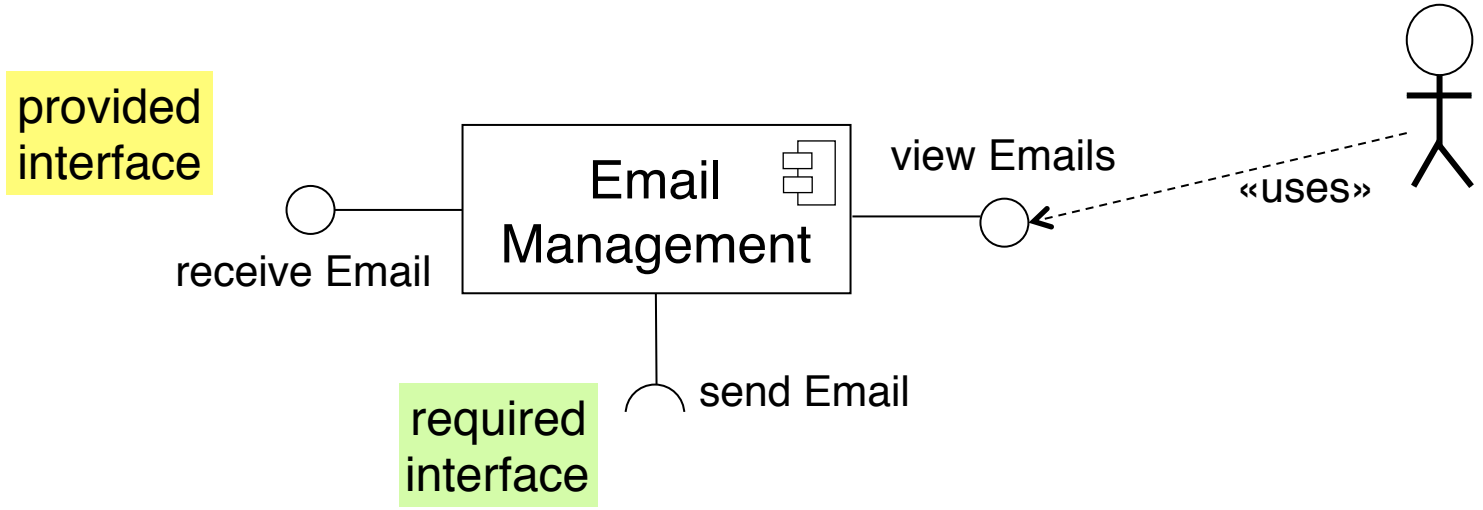
Disadvantages: very small scale (heavy design), might get complex

Class Activity

- In groups of two, find an example of a use of the Model-View-Controller pattern

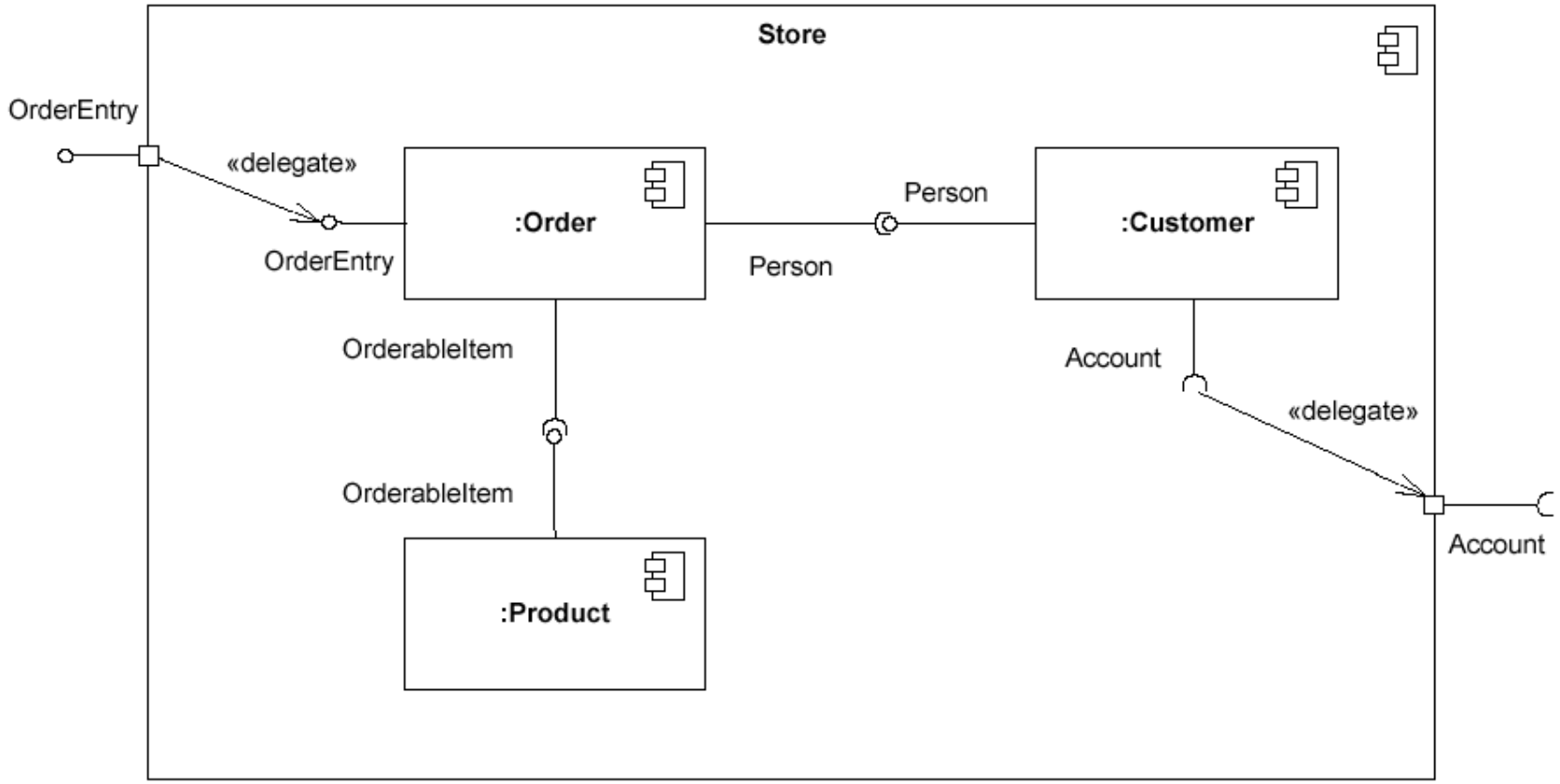
UML Component Diagram

- Several ways to depict architecture, depending on what is important
- UML Component Diagram to depict components and interfaces



More ways of documenting: http://en.wikipedia.org/wiki/4%2B1_architectural_view_model, https://sites.google.com/site/softwarearchitectureinpractice/9-documenting-software-architecture

UML Component Diagram – Hierarchy



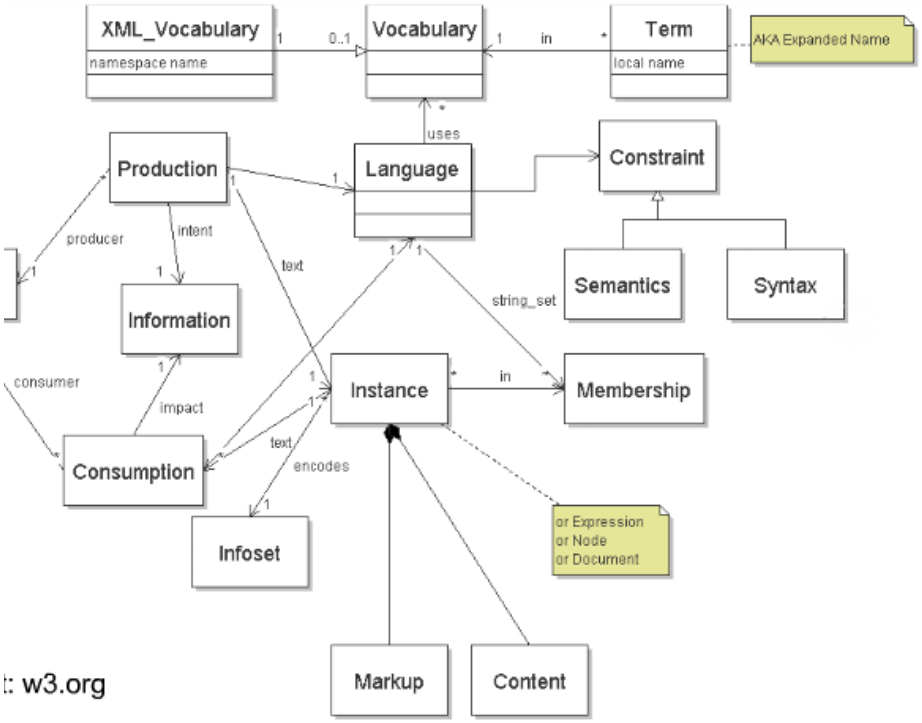
Components can be composed of other components or classes

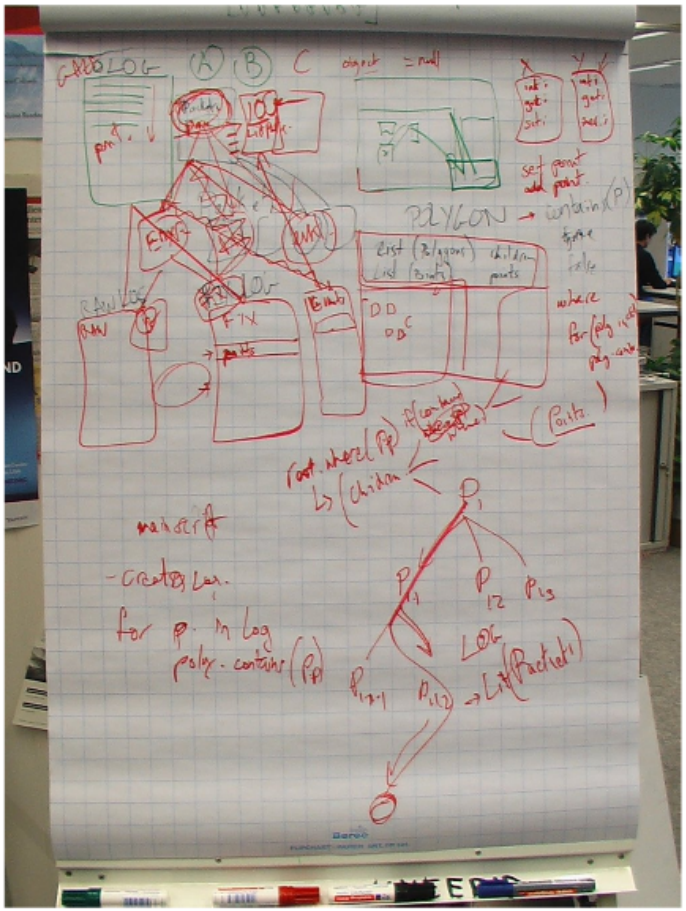
Detailed Design

- Concerned with programming concepts
  - Classes, Packages
  - Files
  - Communication protocols
  - Synchronization
  - …
- Mid-level design
  - class diagrams
- Low-level design
  - sequence diagrams

Class Activity

- Vote: Which of these two diagrams is more useful to software developers?





Diagrams

Diagrams are a ***communication*** tool

- End product is important, but discussion just as important

Quality of communication = Quality of design

- Hence, quality of end product

Tip for efficient communication:

- Start light-weight and flexible
- Then move on to details and more focused

In terms of diagrams:

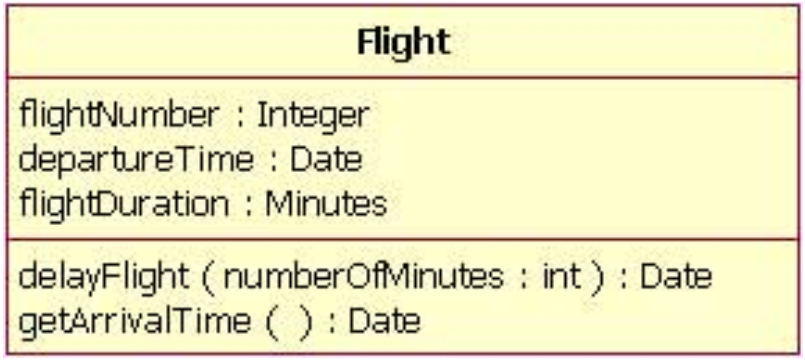
- Start with draft, hand-written diagrams that can change
- Towards the end, clean-up and make more readable
- Use a mutually understood language (a standard: UML)

Class Diagrams

- Mid-level design tool
- Used to describe the relationships between classes (and/or packages) in the system

- UML: Unified Modeling Language *(not only class diagrams)*
- Elements of UML class diagrams
  - Classes
  - Relationships
    - Generalization
    - Association
    - Aggregation

Class Diagrams: Class

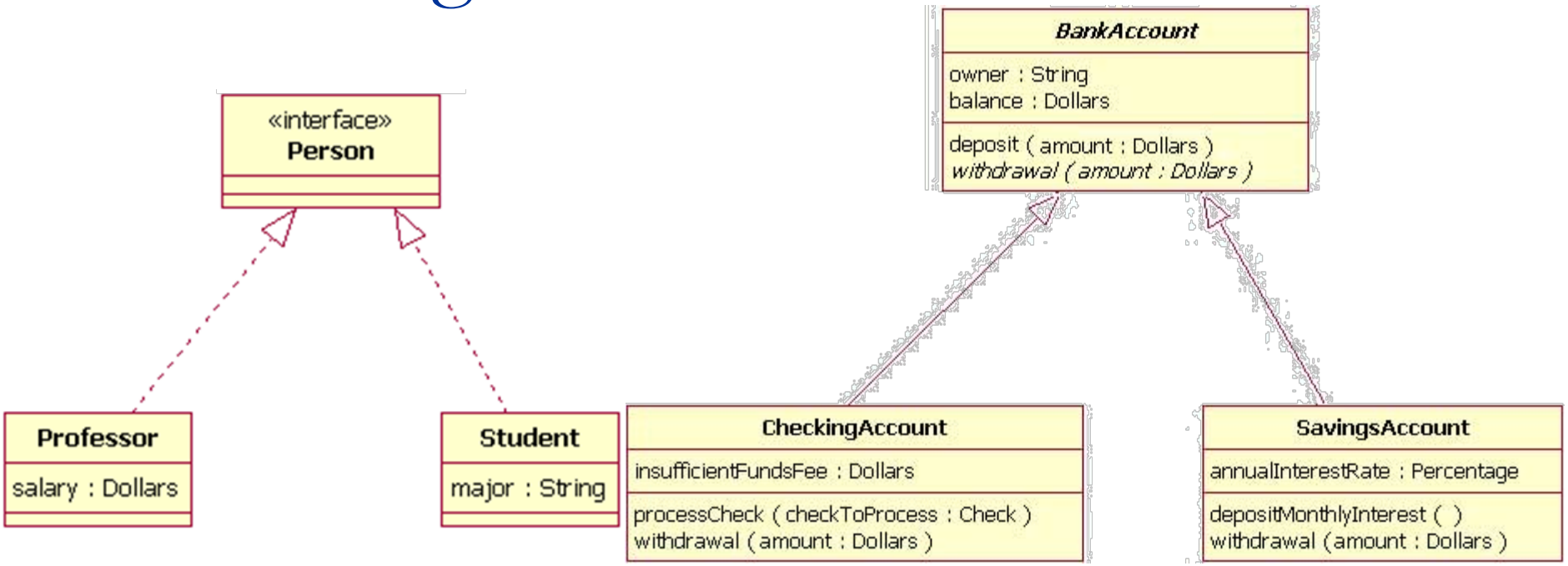


- Class name (*Italics* means abstract)
- Attributes (fields)

  `Name : Type`
- Operations (methods)

  `Parameters : Return Type`
- Can also be used for interfaces (without fields)

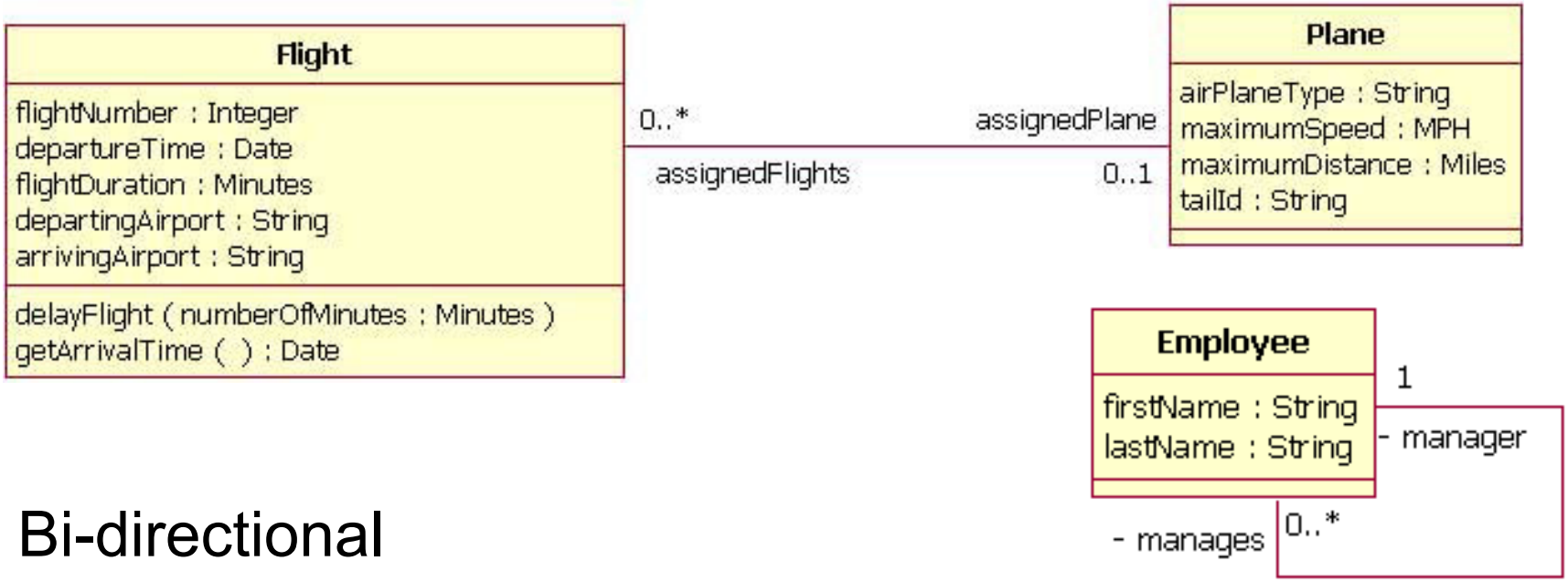
Class Diagrams: Generalization



Used for:

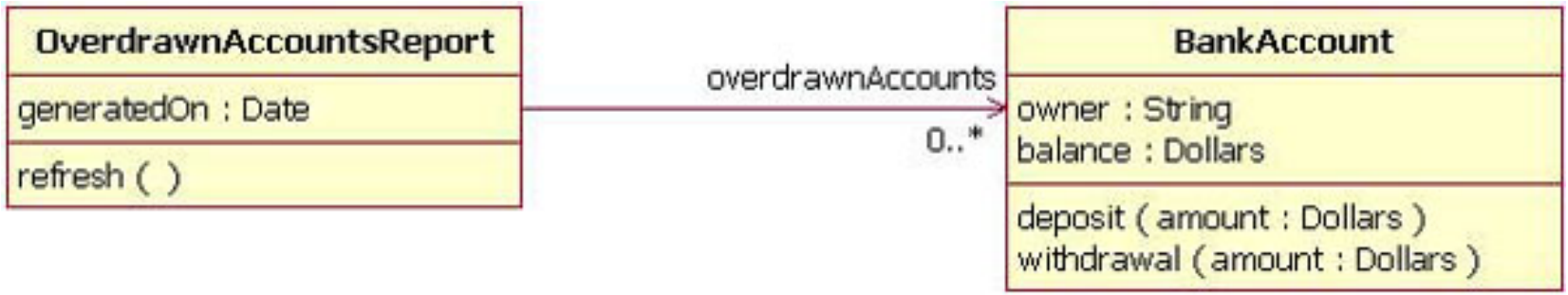
- Inheritance
- Interface implementation

Class Diagrams: Association



- Bi-directional
  - Both classes are aware of each other
- Role
  - Usually maps to a field name
- Multiplicity
  - Indicates how many instances can be linked (*i.e.* a list of…)

Class Diagrams: Uni-directional Association



- Only one class knows of the other
- Role
  - Only in one direction
- Multiplicity
  - Only on one end (BankAccount doesn't know report)

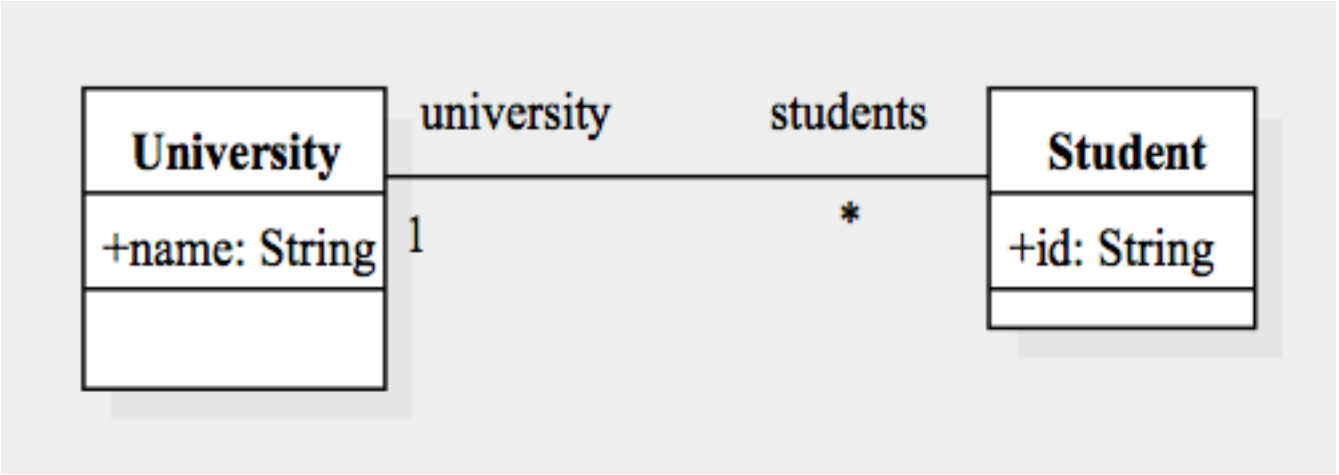
Class Diagrams: Aggregation

- An advanced type of association
- The contained object is *part* of the container
- Two type:
  - Basic aggregation: children can outlive parent

  - Composite aggregation: children life depends on parent

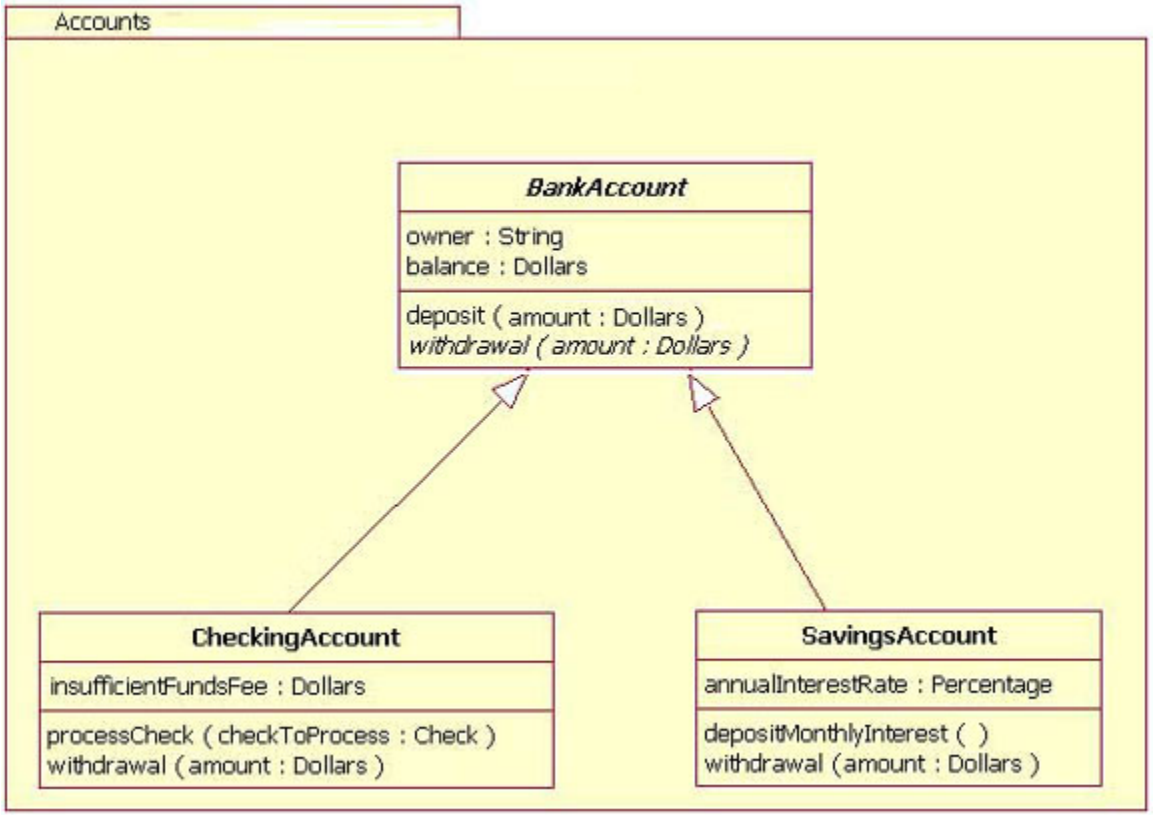
Class Activity

- How would you implement these two examples in Java?



Class Diagrams: Packages

- Group classes together



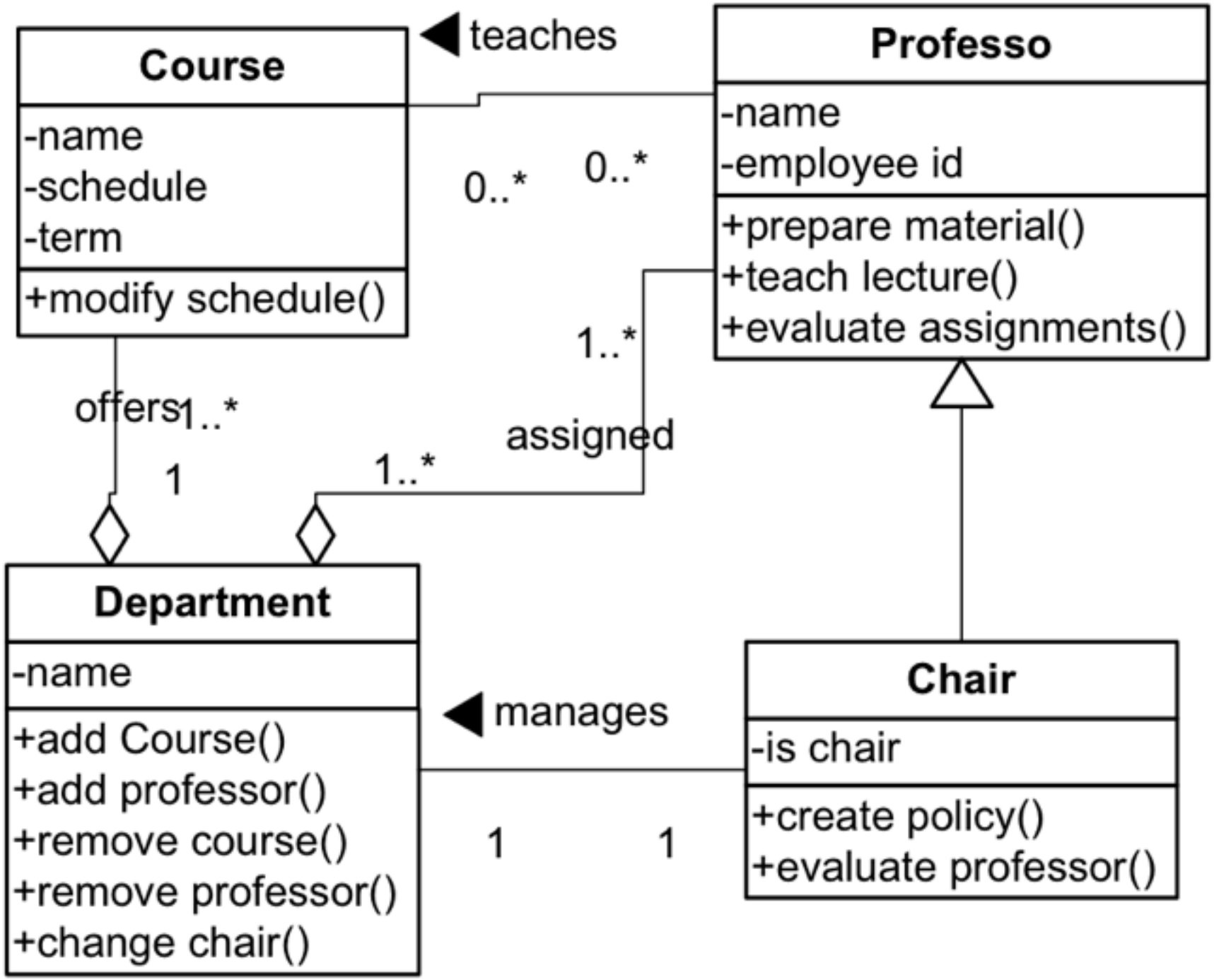
Class Activity

- In teams of 2, draw a class diagram for a software system for modeling a bank
- Each of the bank's customers can access their account(s) through withdrawals, deposits, or balance inquiries at a bank machine. Each transaction (ie, withdrawal, deposit or balance inquiry) must store the date and time that the transaction occurred. Once a month, a statement that contains a list of all of the transactions that were completed over the last month is generated for each account and mailed to the customer. The bank must be able to produce a list of all of its customers as well as a list of transactions that were completed by a particular bank machine
- Use classes that appear in the text.
- Ask questions if needed.

Class Activity

In small groups, draw a class diagram for the structure of professors, students, departments, dept. heads, and courses.

- Ask questions if needed.


Course
-name
-schedule
-term
+modify schedule()
teaches
Professo
-name
-employee id
+prepare material()
+teach lecture()
+evaluate assignments()
0..*
0..*
1..*
offers
1..*
1
assigned
1..*
Department
-name
+add Course()
+add professor()
+remove course()
+remove professor()
+change chair()
manages
1
1
Chair
-is chair
+create policy()
+evaluate professor()

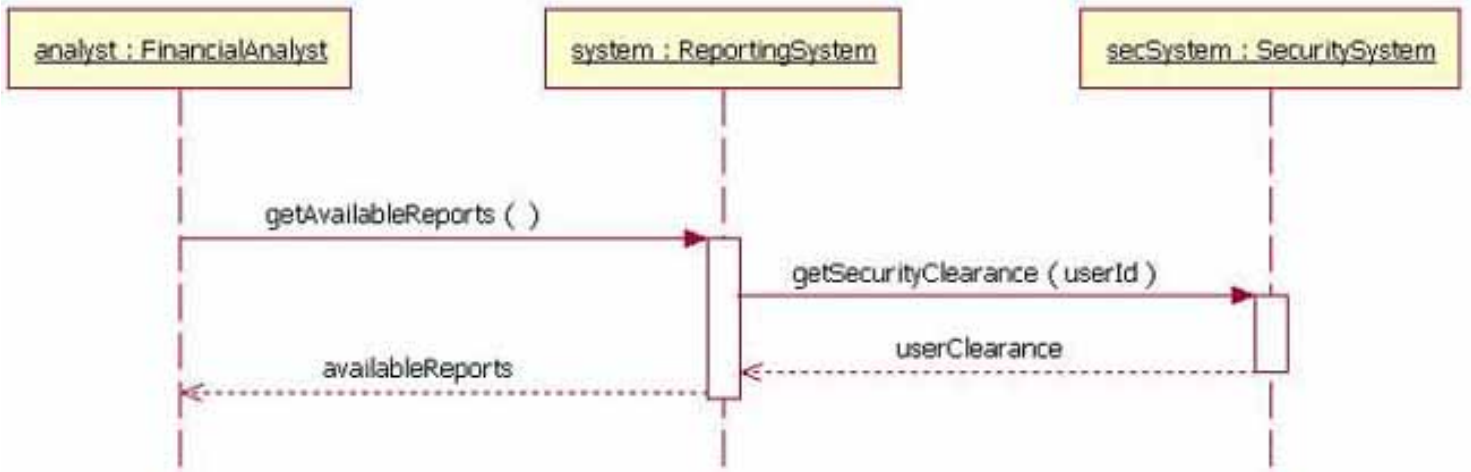
Comments on Diagrams

- Think about association/aggregation/composition as well as direction of them
- Make sure to specify roles if necessary (especially if there are two relations between two classes)
- Think about methods and attributes and where they belong

Sequence Diagrams

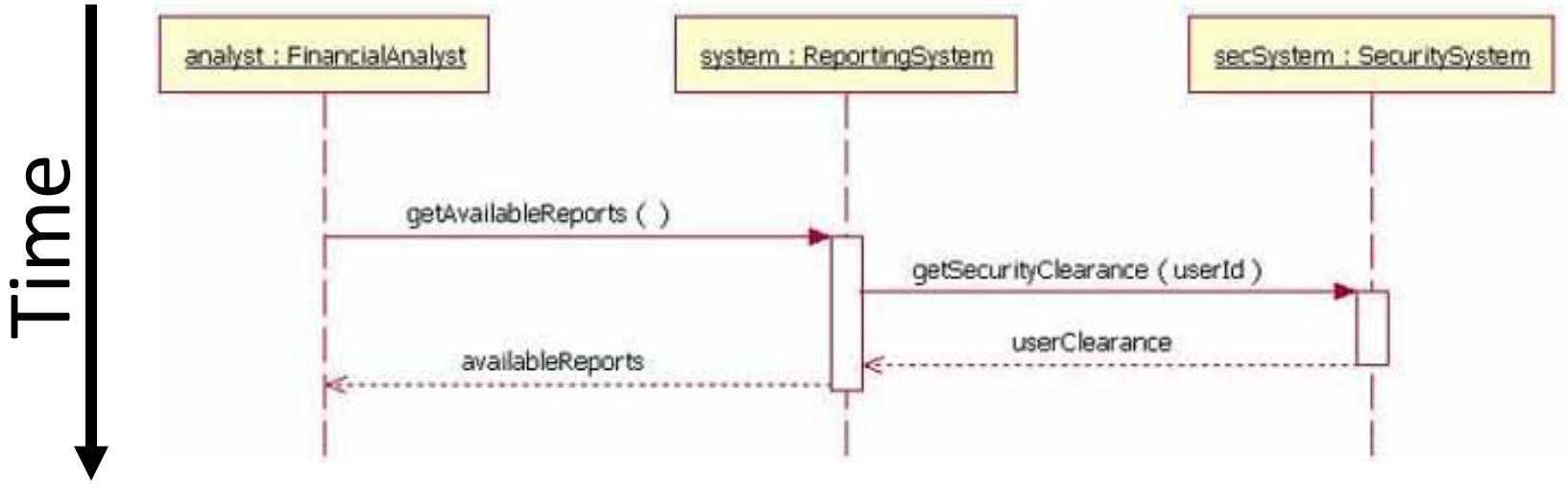
- Low-Level design tool
- Used to describe sequences of invocations between the objects that comprise the system
  - Focus less on *type of messages*, more on the *sequence* in which they are received
- UML (again!)
- Elements of UML sequence diagrams:
  - Lifelines
  - Messages
  - …

Sequence Diagrams: Lifeline



- Roles or object instances
- Participate in the sequence being modeled

Sequence Diagrams: Messages

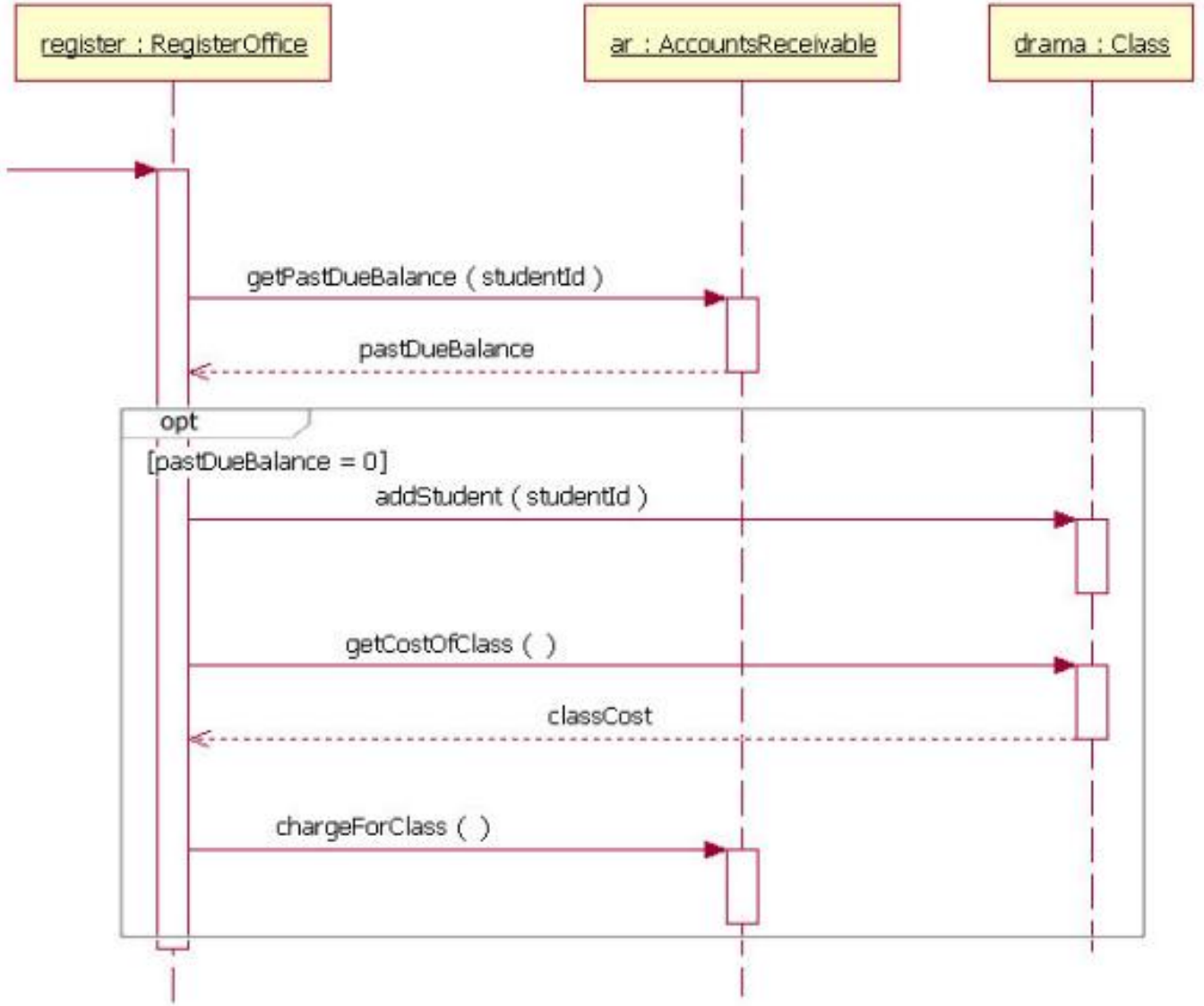


- Includes method name
- A box in the receiver's lifeline indicates activation (object's method is on the stack)
- Full arrow: synchronous (blocking)
- Optionally: information returned

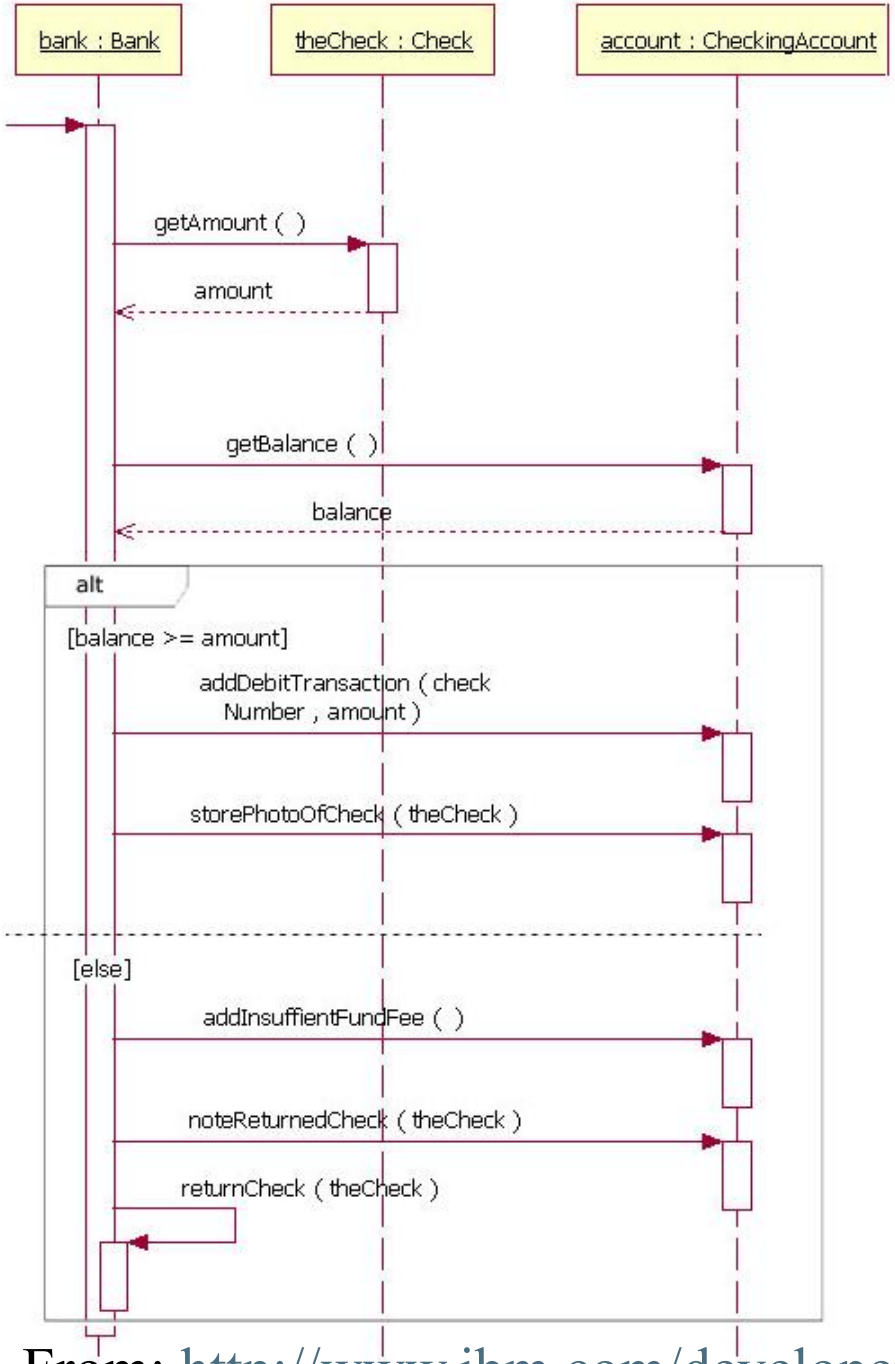
How do you start? (Sequence Diagram)

1. Identify process/algorithm/activity you want to capture (may be a use case)
2. Identify major objects involved
3. Map out flow of control/messages to achieve the result

Sequence diagram when some actions are inside an if



From: http://www.ibm.com/developerworks/rational/library/3101.html



Sequence diagram when some actions are inside an if/else

Loops are similar - put the actions inside a box labeled “loop”

From: http://www.ibm.com/developerworks/rational/library/3101.html

Class Activity

- In teams of 2, draw a sequence diagram illustrating what happens when *a customer withdraws money at a bank machine* .

Class Activity

- In teams of 2, draw a sequence diagram illustrating what happens when *a user searches for a course by course label* .
- Or for registering for a course.

What is the Software Design?

- The design consists of multiple views of the software
  - Static view (e.g. class diagram) shows decomposition of problem into parts and relationships
  - Dynamic view (e.g. sequence diagram) shows how parts interact to solve the problem
- Views have varying levels of granularity
- We can analyze these views to see if they support the requirements?
  - Modifiable (i.e. adding new view)?
  - …

How to Design? (Recap)

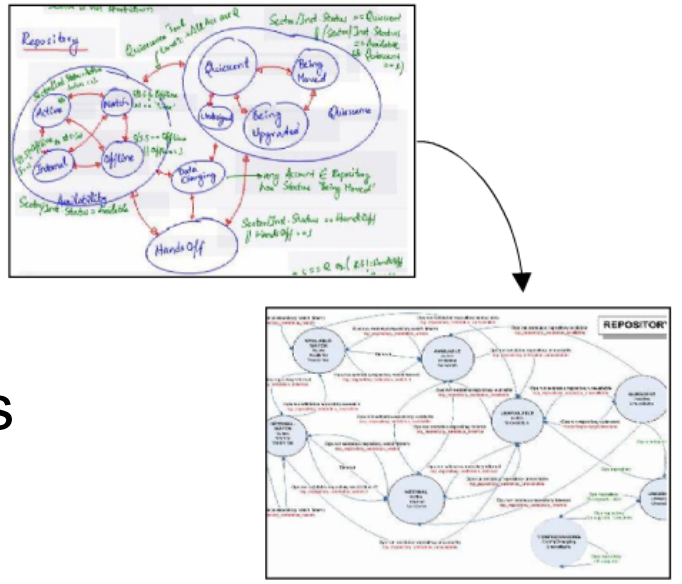
- "Treat design as a wicked, sloppy, heuristic process."
  - Pen & Paper, Whiteboard
- "Don't settle for the first design that occurs to you."
  - Scribble, Scratch, Thrash
- "Collaborate"
  - Brainstorm, Discuss, Argue
- "Strive for simplicity"
  - Reduce, Clean-up (with UML tools)
- "Iterate, iterate and iterate again."
  - Iterate!

How to Design?

- Start global
  - Architectural Design
  - Global concepts: components & connectors
- Subdivide
  - Detailed Design
  - Mid-level: classes and relationships
  - Low-level: how operations are carried out – what messages are sent and when
- Iterate
  - Are these the right subsystems? Update!
  - Are these the right classes? Update!

How to Design in a Team?

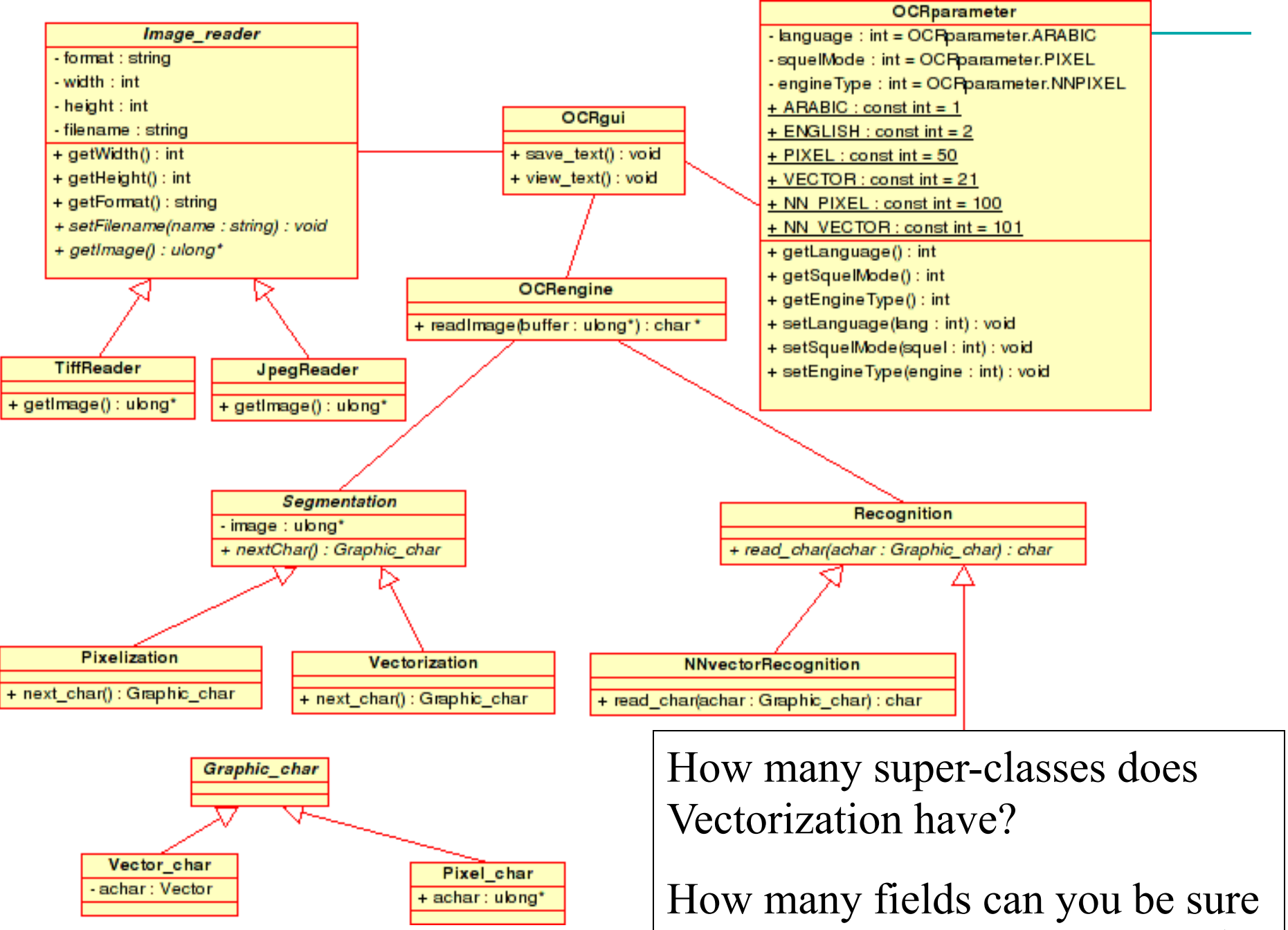
- Suggested steps
  1. Discussion
  2. Discussion with paper diagrams
  3. Clean-up with UML tools
  4. Discussion with printed diagrams
  5. Iterate over 3 and 4



- Again
  - First light-weight and flexible (words, hand drawn diagrams)
  - Then details and focused (printed diagrams)

Design Summary

- Design consists of multiple views
  - High-Level: Component Diagrams
  - Medium-Level: Class Diagrams
  - Low-Level: Sequence Diagrams
- Architectural styles encode common patterns for achieving certain quality goals
- Diagrams are *communication tools*
- Designing is an iterative refinement process



How many super-classes does Vectorization have?

How many fields can you be sure that OCRgui has?