CPSC 310 – Software Engineering

Design: UML Review

Short review assuming background from CPSC 210

Reading

Optional, but a very good resource about design

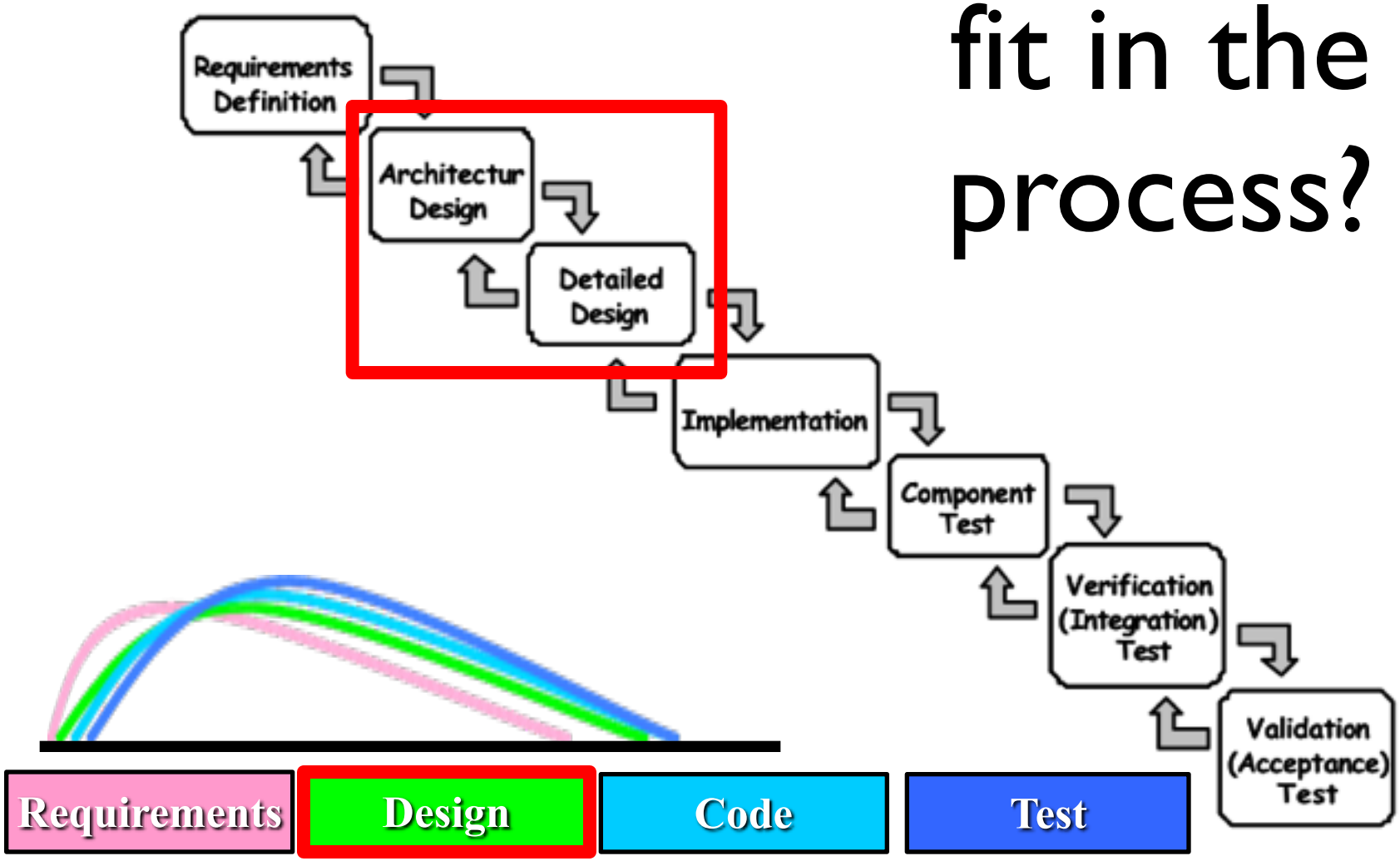
http://courses.cs.washington.edu/courses/cse403/07sp/assignments/Chapter5-Design.pdf

Learning Goals

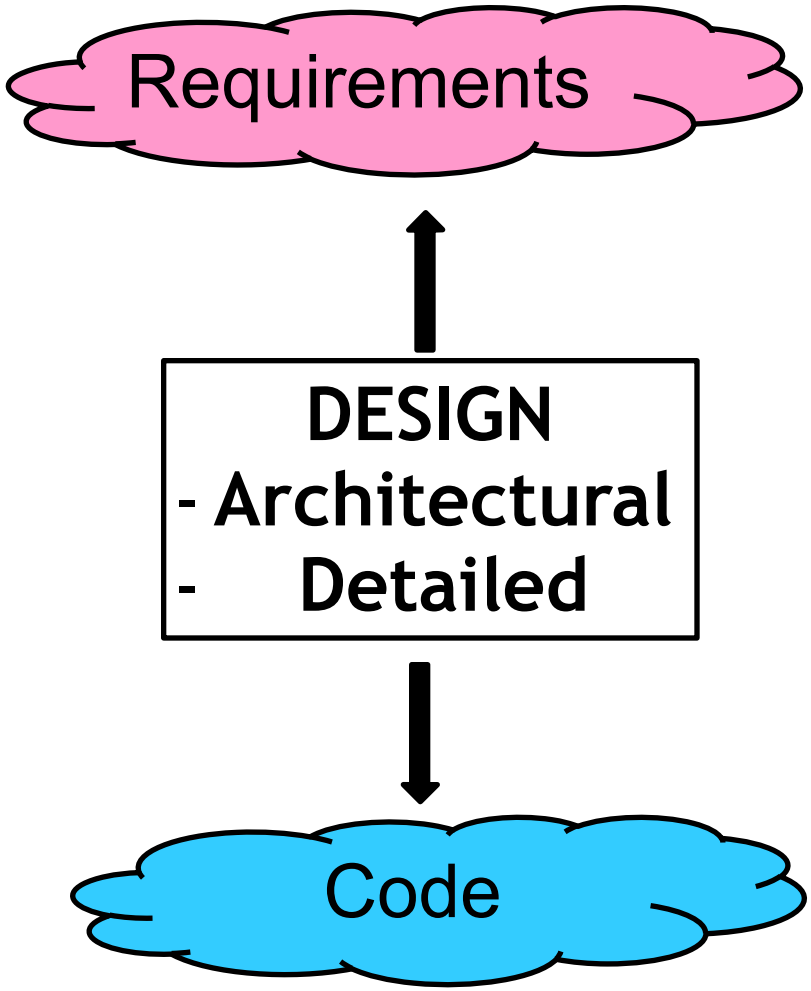
By the end of this unit, you will be able to:

- Describe the context (goals and constraints) of the activity of software design and explain why it's important
- Describe the kinds of information we must capture in a software design
- Understand the use of diagrams in software development (e.g. in what situations would different diagrams be useful)
- Create a design for a given system and specify it in correct UML class/sequence diagram syntax
- Describe a process for developing a design

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

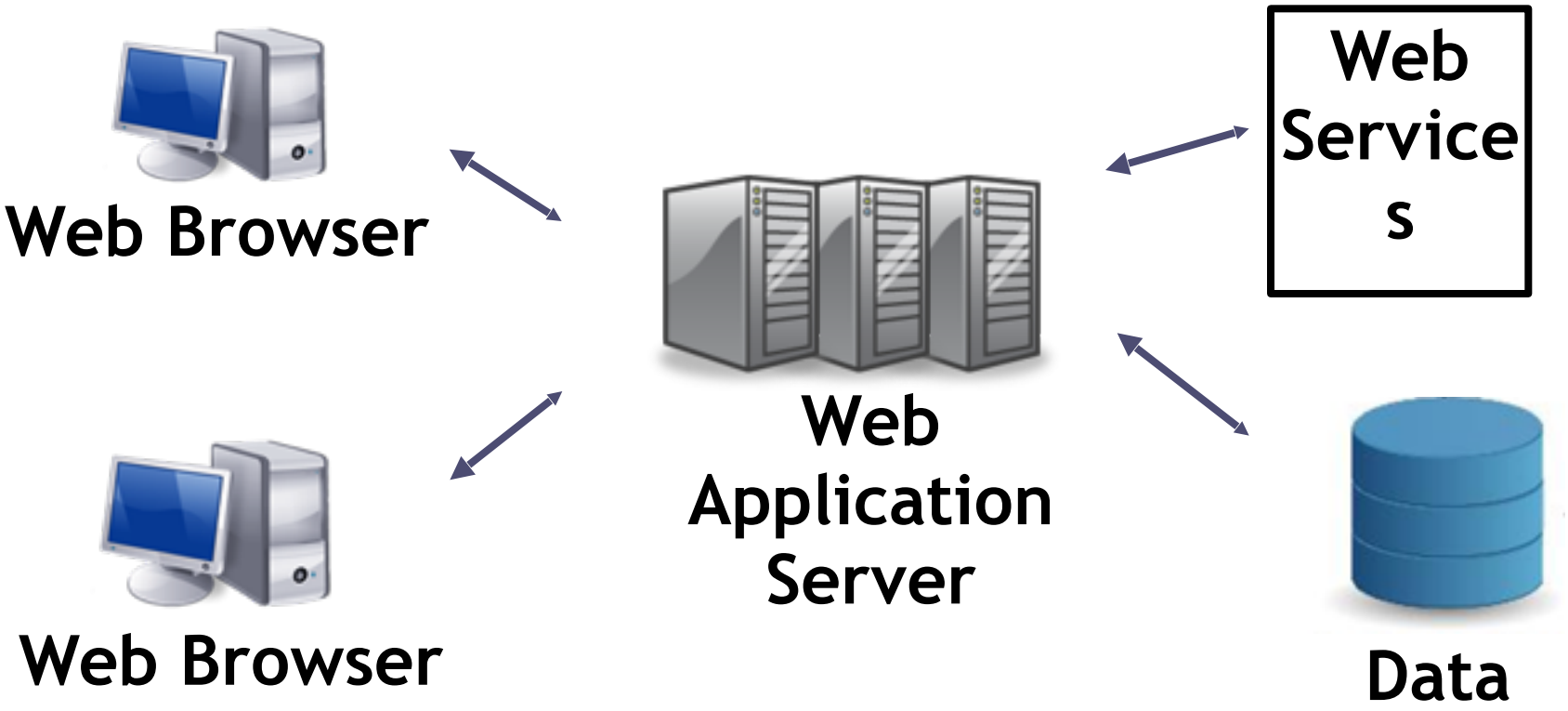
Two common phases of Software Design

1. **Architectural** design
   - Determining which sub-systems you need (e.g., web server, DB…)
   - Discussed more in CPSC 410
2. **Detailed** design
   - Looking at the statics and dynamics of your system (classes; sequences)

Architectural Design

- The architecture of a system describes its gross structure:
  - Main components and their behaviour (system level, sub-systems)
  - Connections between the components / communication (rough idea)
- Architectural Styles: data-flow, client/server,…
- Tools: UML, whiteboard, paper

Web Architecture (Client / Server Style)



Detailed Design

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

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)

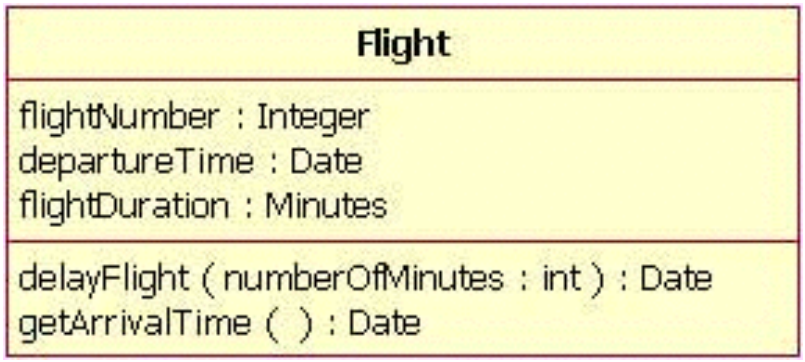
Class Diagrams

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

- UML: Unified Modeling Language *(not only class diagrams)*

- Main elements of UML class diagrams
  - Classes
  - Relationships
    - Generalization
    - Association
    - Aggregation

Class Diagrams: the Class

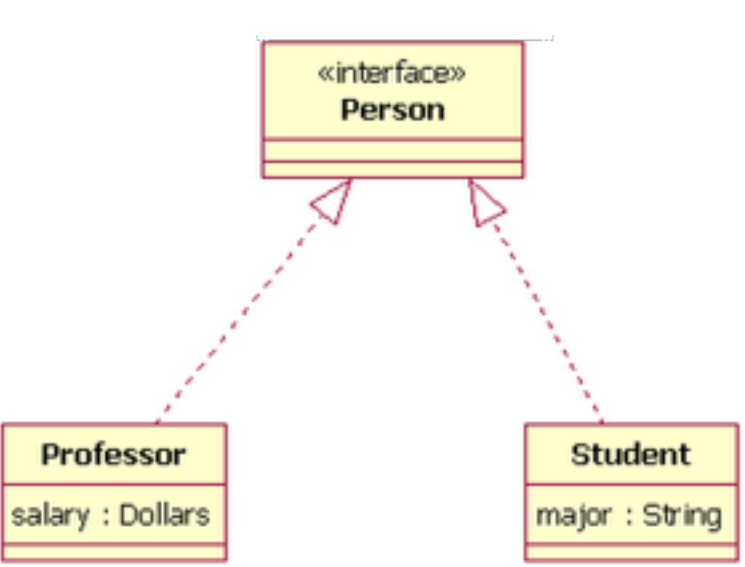


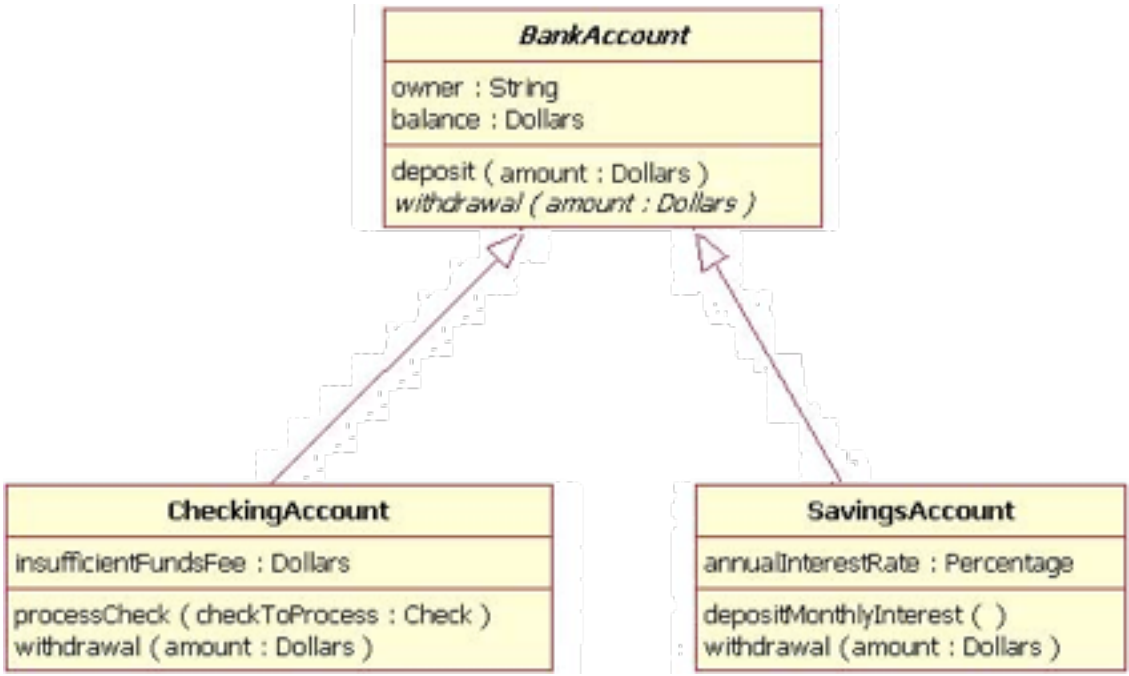
- Class name (*Italics* means abstract)
- Attributes (fields)

  `Name : Type`
- Operations (methods)

  `(parameters) : ReturnType`
- Can also be used for interfaces (without fields)

Class Diagrams: Generalization

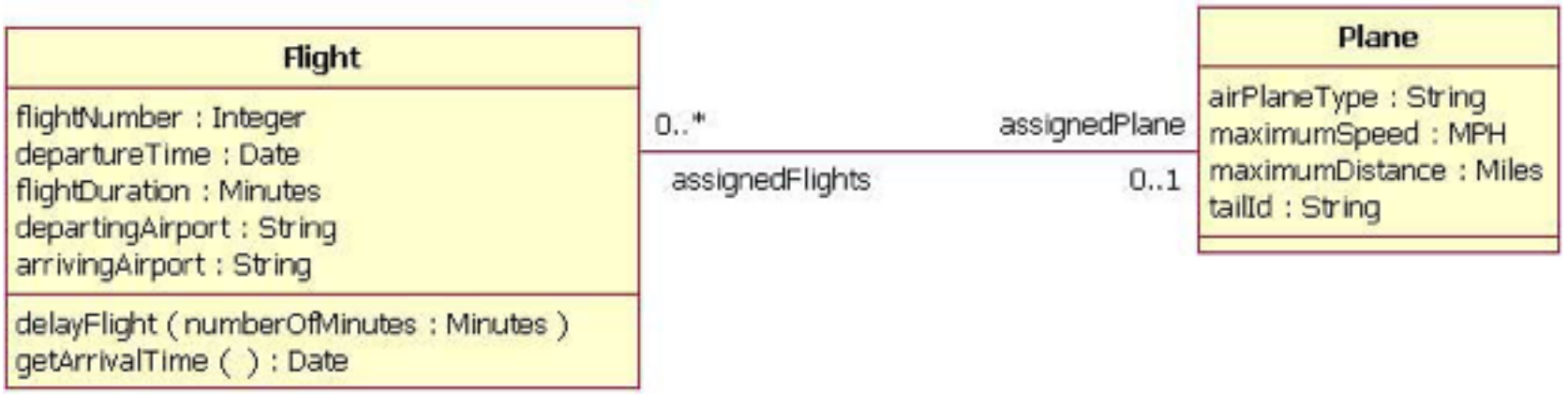




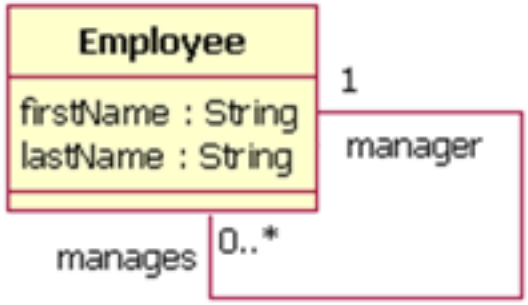
Used for:

- Inheritance
- Interface implementation

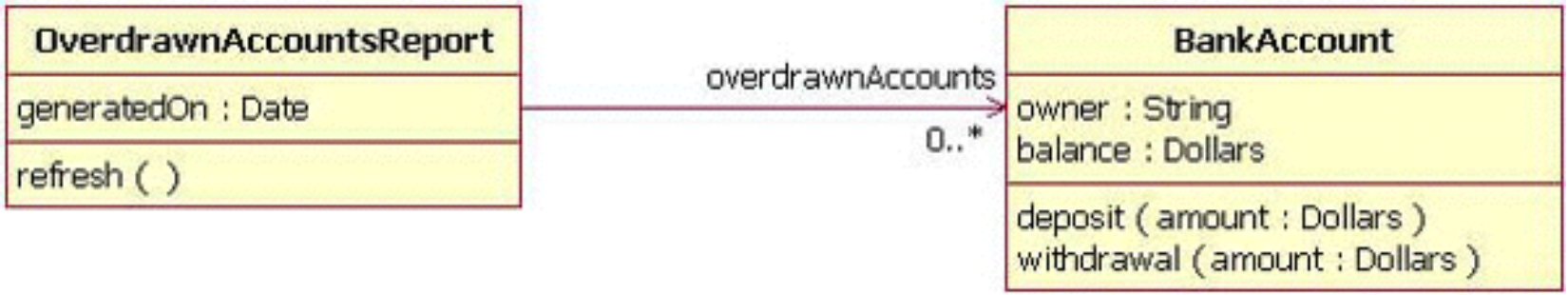
Class Diagrams: Association



- 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

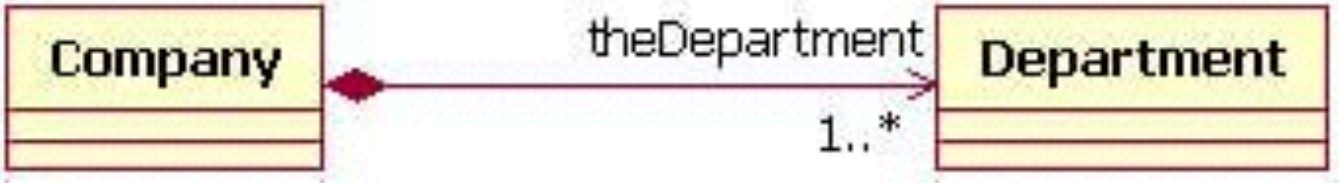


- Only one class knows of the other
- Role
  - Only in one direction
- Multiplicity
  - sometimes shown only on one end (BankAccount doesn't know report)

Class Diagrams: Aggregation

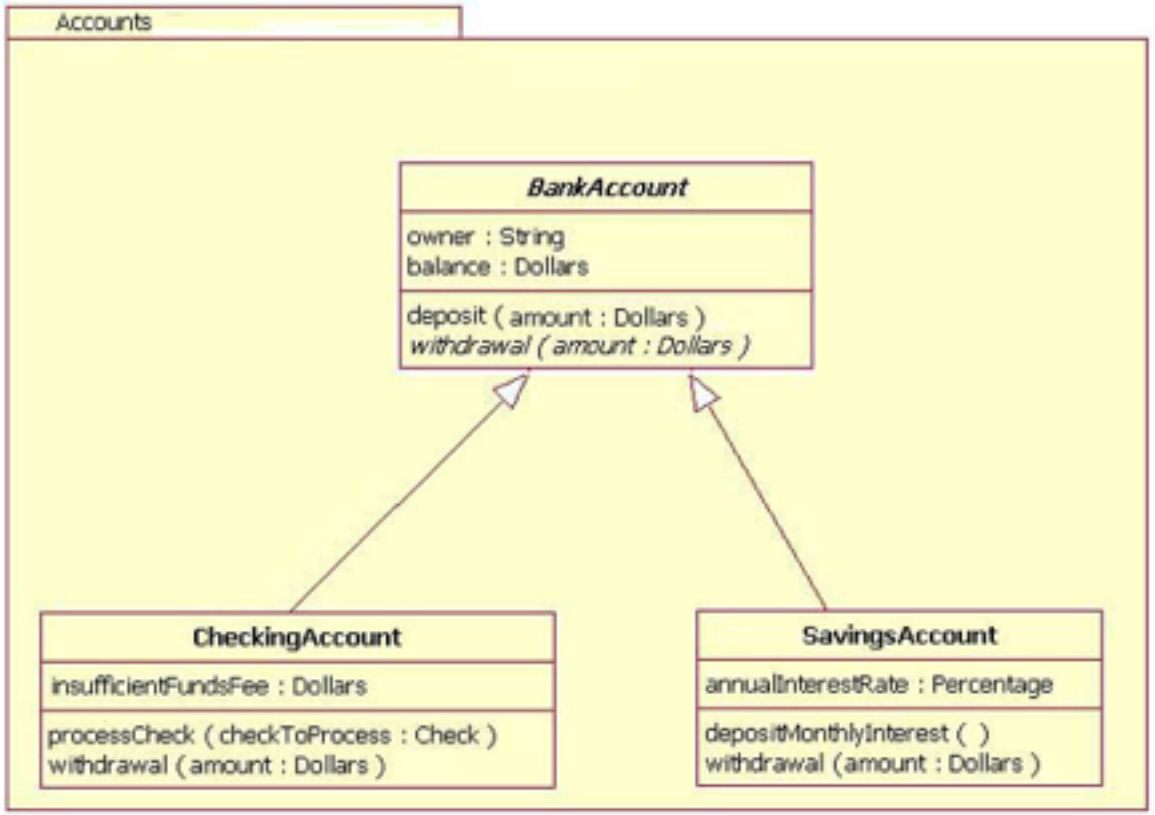
- An advanced type of association
- The contained object is *part* of the container
- Two types:
  - Aggregation: children can outlive parent

  - Composition: children's life depends on parent



Class Diagrams: Packages

- Group classes together



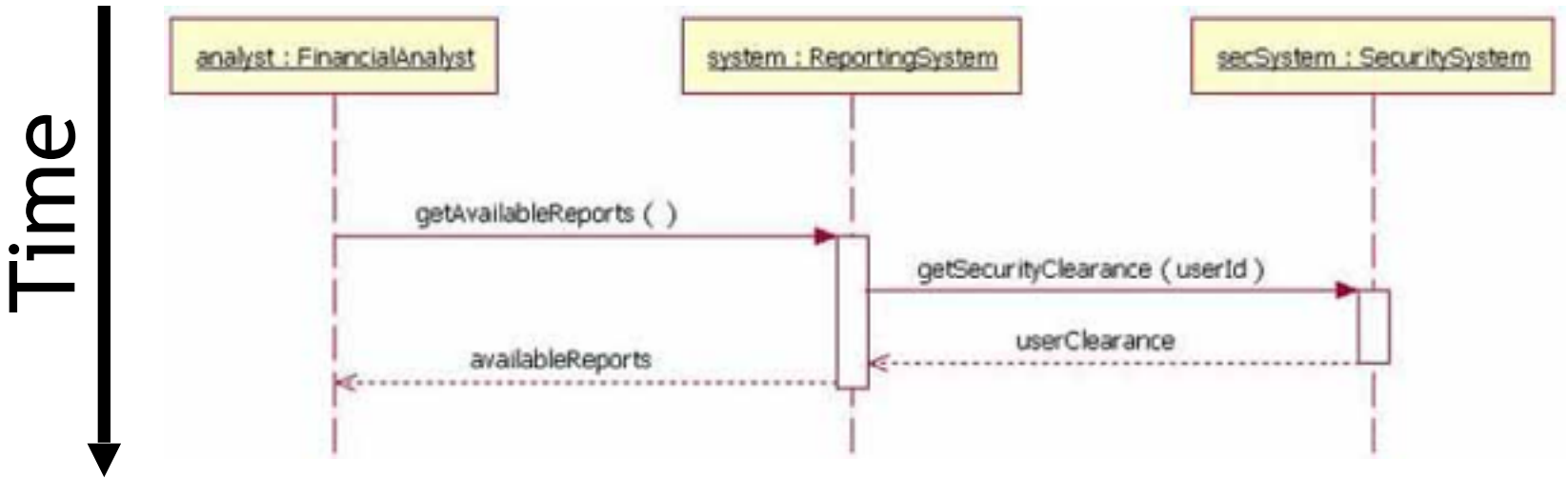
Sequence Diagrams

- 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
- Elements of UML sequence diagrams:
  - Lifelines
  - Messages

Sequence Diagrams: Lifeline

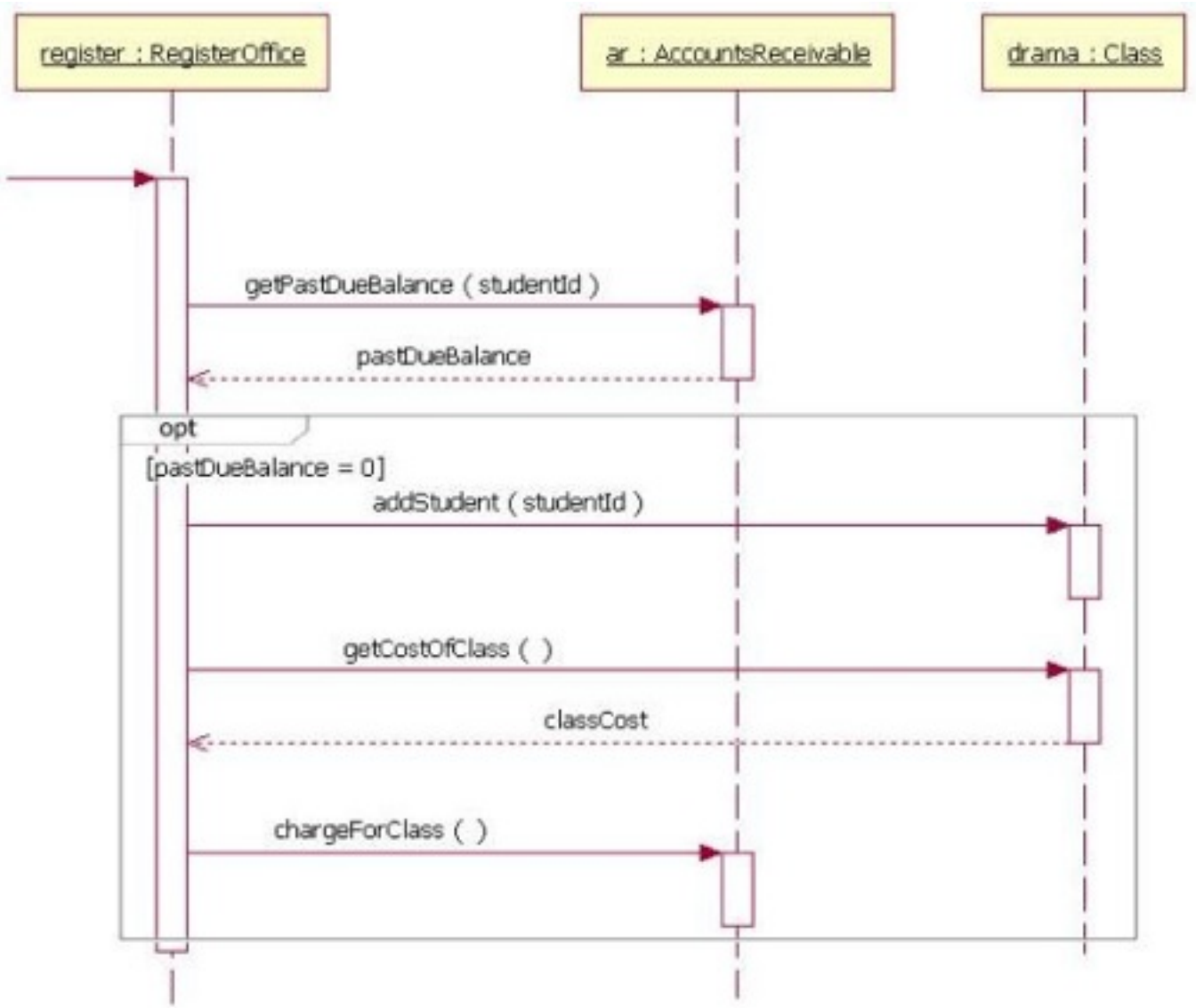
- Roles or object instances
- Participate in the sequence being modeled

Sequence Diagrams: Messages

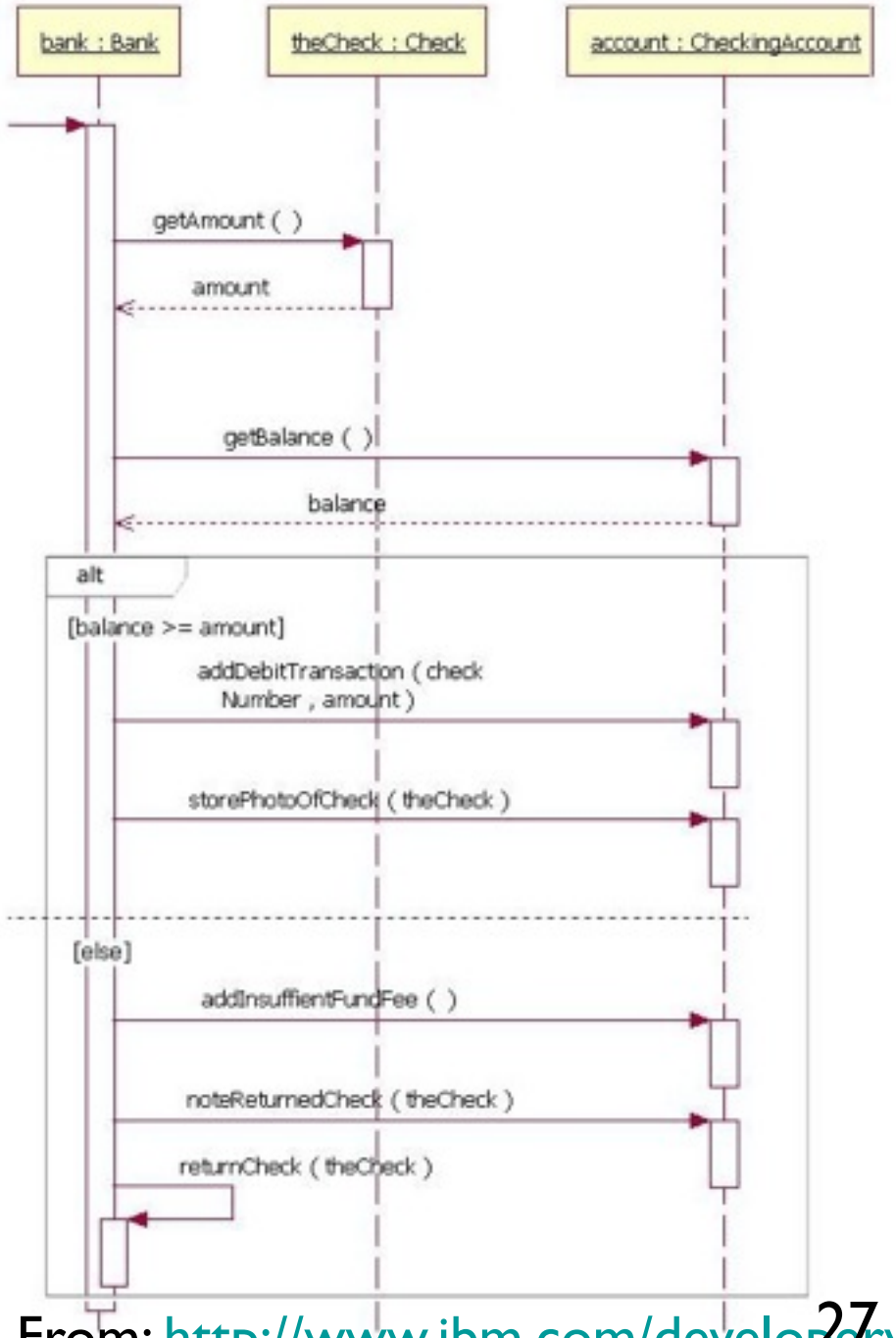


- Includes method name
- A box in the receiver's lifeline indicates activation (object's method is on the stack)
- Full arrow: synchronous (blocking)
- Optionally: information returned

Sequence diagram for conditionals



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



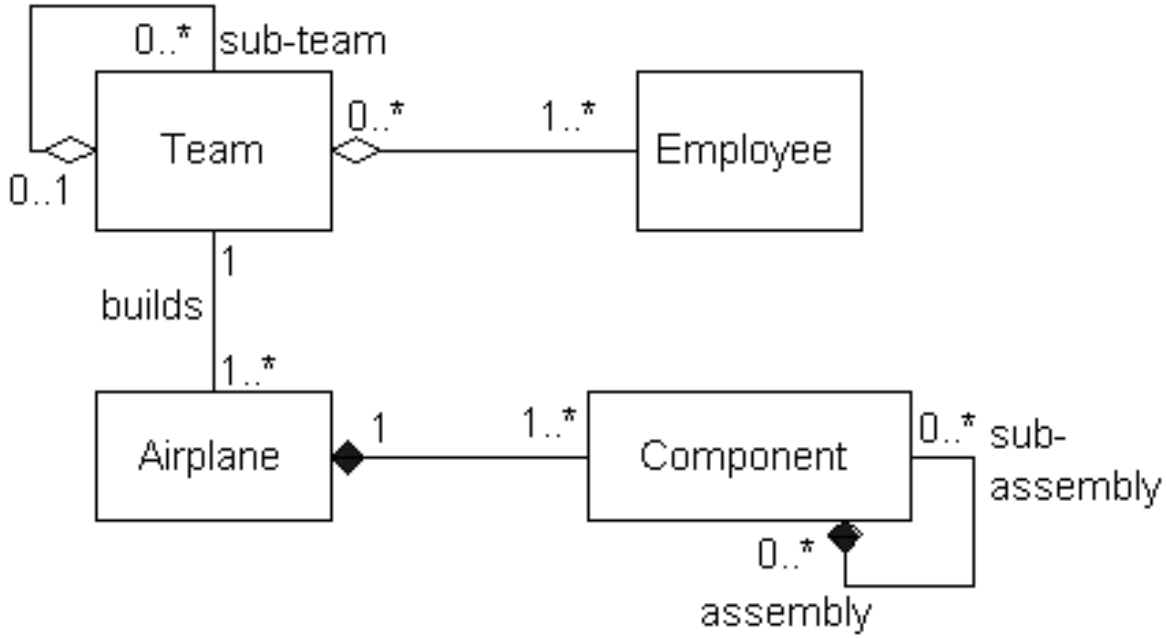
Sequence diagram when some actions are inside an if/else

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

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

Exercise

1. How many facts can you list from this diagram?
2. What questions do you have as you look at this diagram?



Conclusion

- UML diagrams used to focus on specific details of software structure
  - *Not intended for mocking-up every detail*

- A good example where UML shines is **Design Patterns**
  - *Reusable templates of design applicable across multiple projects*