CPSC 310 – Software Engineering

Design: UML Review

Short review assuming background from CPSC 210

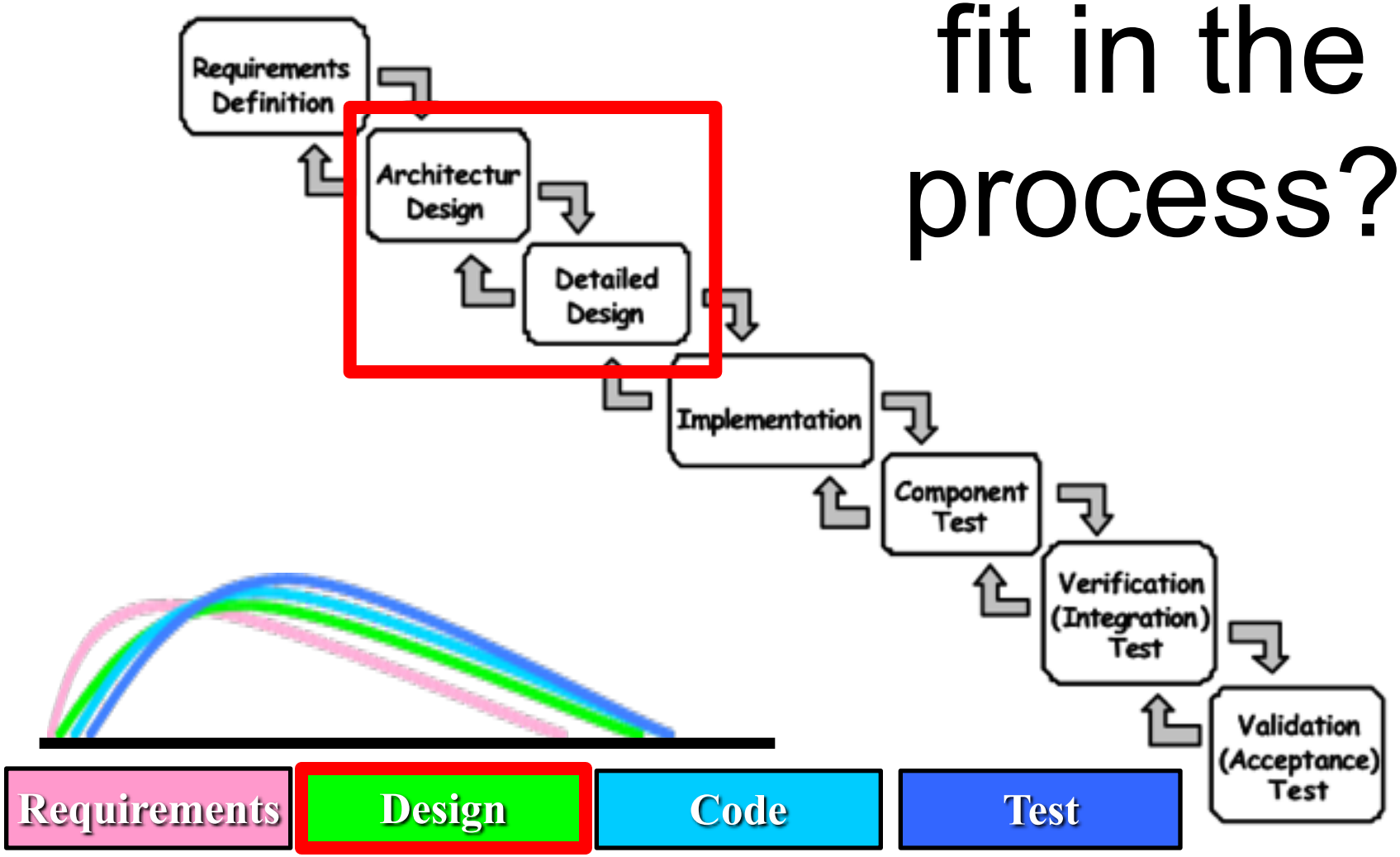
Reading

Code Complete Chapter 5 (available on line)

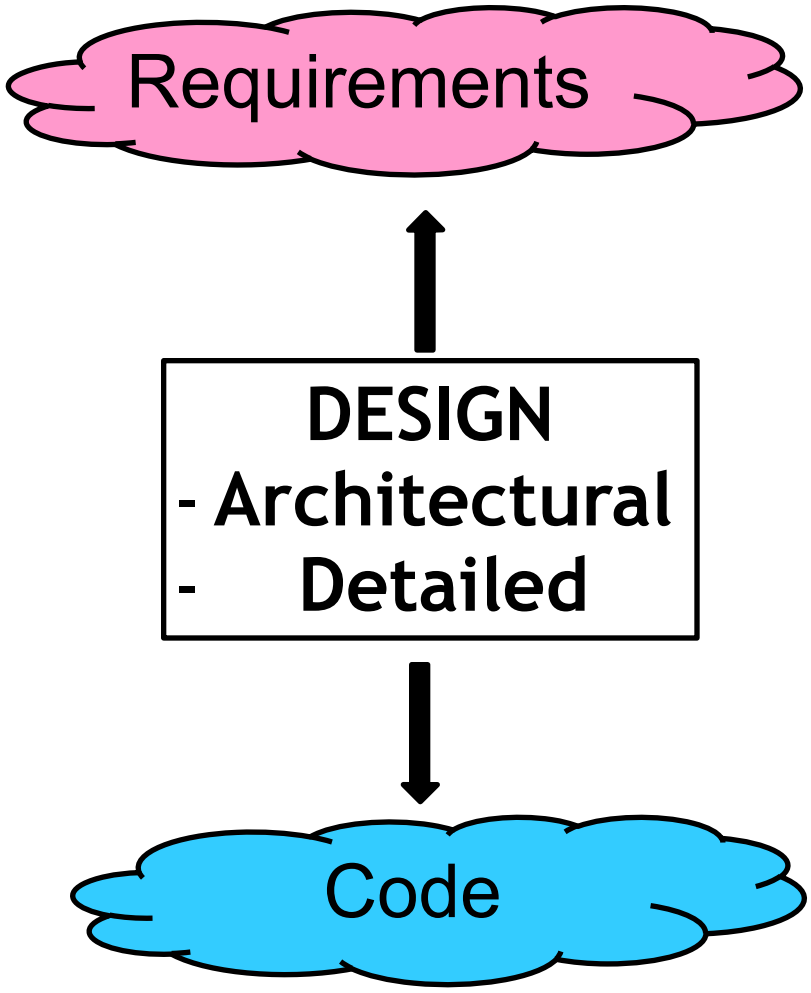
http://cc2e.com/File.ashx?cid=336

This reading will be referred to for the rest of the semester. It will be indicated later which sections you need to read for the midterm and final specifically.

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

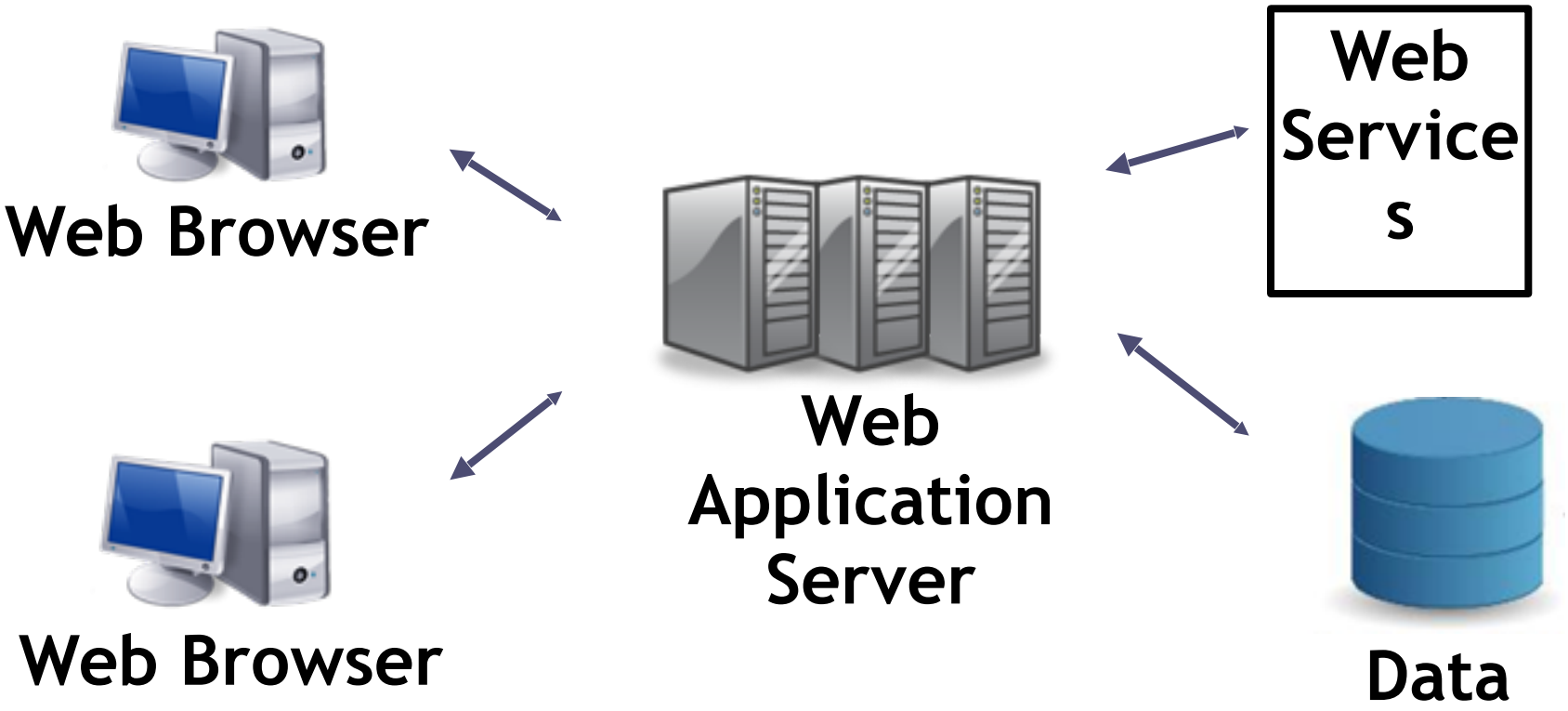
Two common phases of Software Design

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

Elements of OO

- Types:
  - Classes (can be or have subclasses)
  - Abstract Classes (can be or have subclasses
  - Interfaces (can be subclasses of other interfaces, or be implemented by classes)
- Fields (can be of any type)
- Methods (can take parameters of any type, and have a return value of any type)

CLASSES

- Always named as nouns, and names start with a capital letter
- Can be entities or “doers” — and should centre around one concept or responsibility
- All the methods in a class should pertain to the central concept of the class, or contribute to the class’s responsibility

ABSTRACT CLASSES

- Cannot be instantiated
- Are always named as nouns, starting with a capital letter
- Can be entities or “doers” - methods should be centred around one thing or responsibility
- Are typically used for a general type of element that you wouldn’t want to

INTERFACES

- Are sometimes used to encapsulate a role
- Cannot be instantiated
- Should contain a coherent list of methods centered around a single role, responsibility or kind of entity

METHODS

- Are always verbs or actions
- Always start with a lower case letter

Subtyping

- A subtype must pass the IS-A test:
  - *a __subtype__ is a kind of __supertpe__*
  - For instance, do these pass or fail?
    - a Dog is a kind of Animal
    - an Account is a kind of Bank

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)

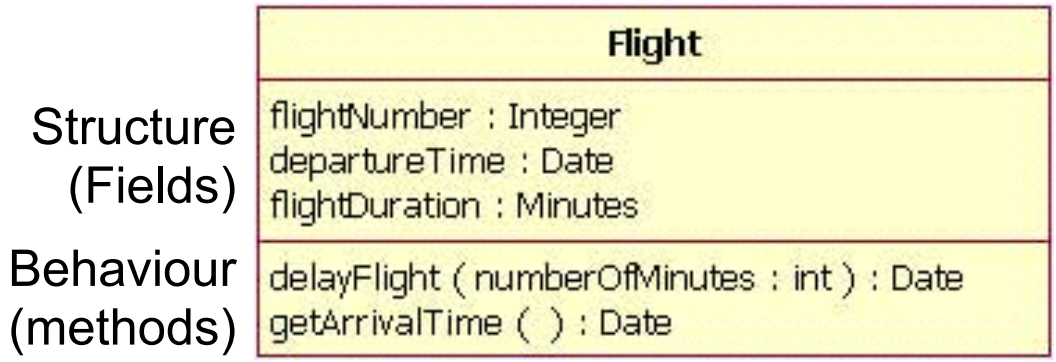
Class Diagrams

- 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 (Subtyping)
    - Association
    - Aggregation

Class Diagrams: the Class



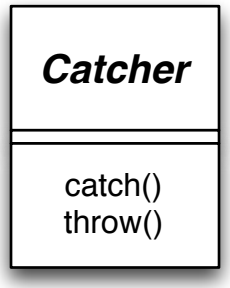
- Class name (*Italics* means abstract)
- Attributes (fields)

  `Name : Type`
- Operations (methods)

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

Class Diagrams: the Interface

Behaviour

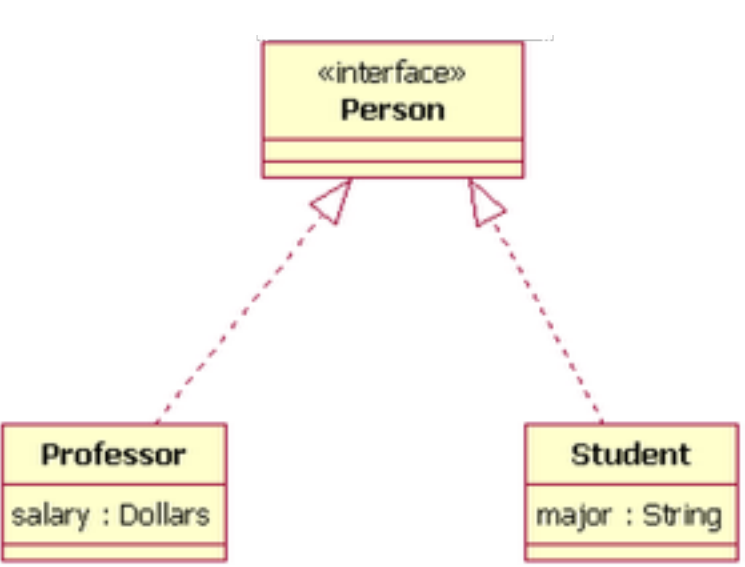


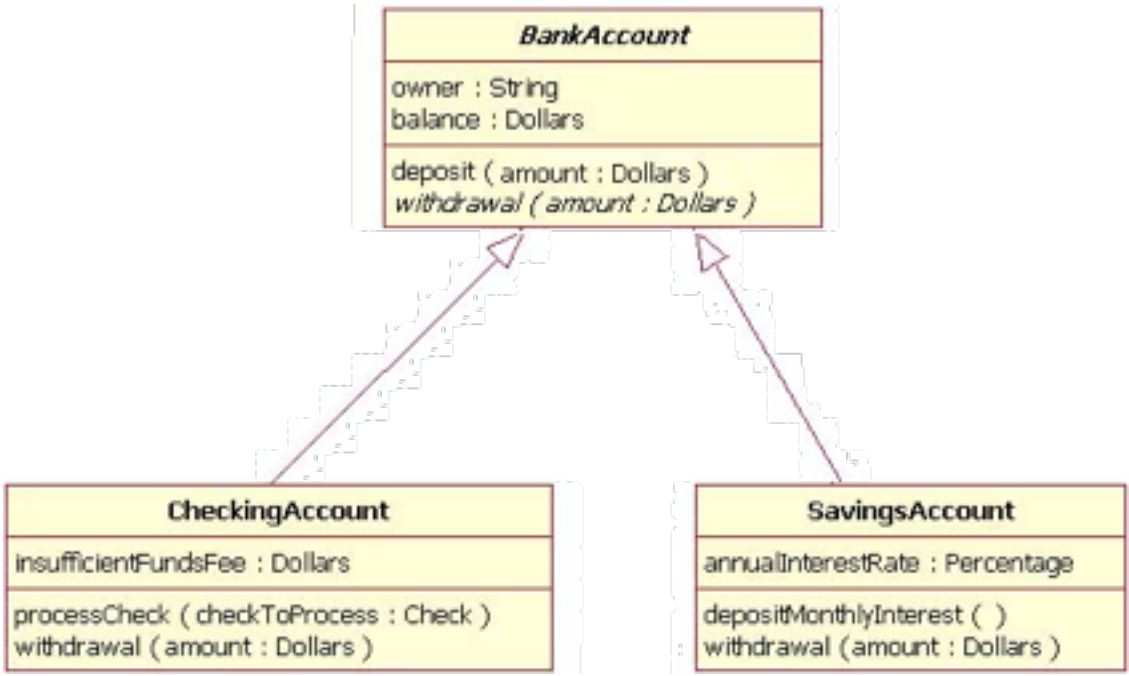
~~Structure~~

- Interfaces are sometimes named as roles
- They never have structure — just behaviour

Class Diagrams: Generalization

Are you doing it correctly? Remember:
Use the Is-A test,

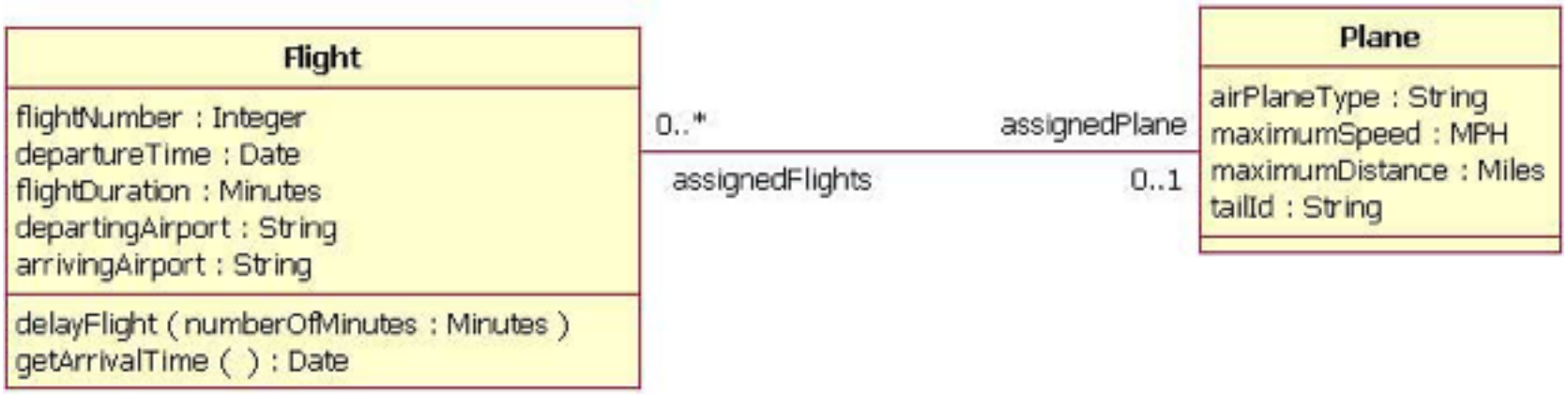




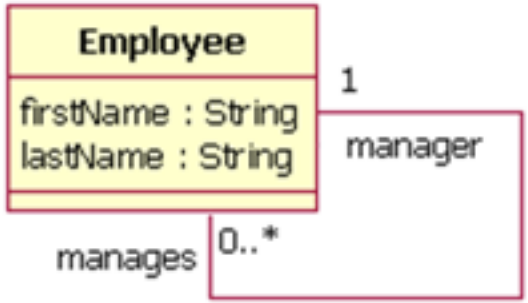
Used for:

- Inheritance
- Interface implementation

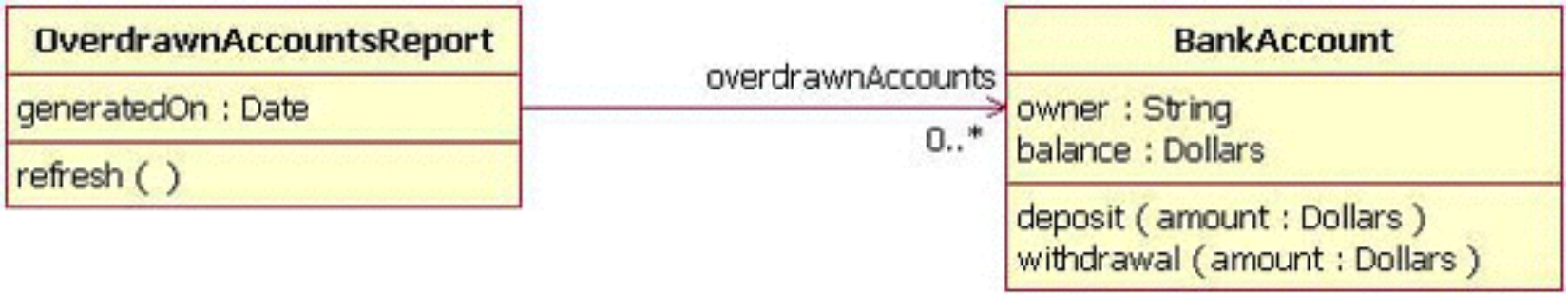
Class Diagrams: Association



- 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

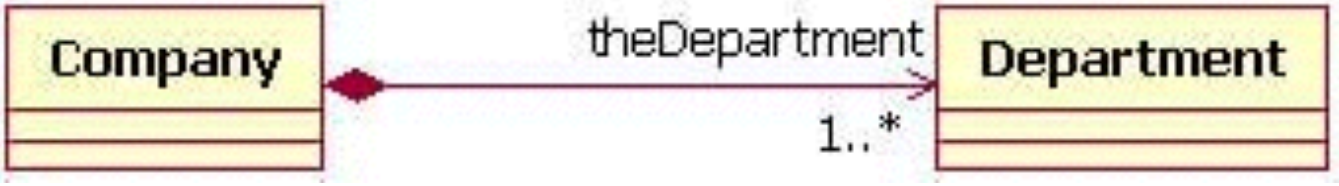


- Only one class knows of the other
- Role
  - Only in one direction
- Multiplicity
  - sometimes shown only on one end (BankAccount doesn't know report)

Class Diagrams: Aggregation

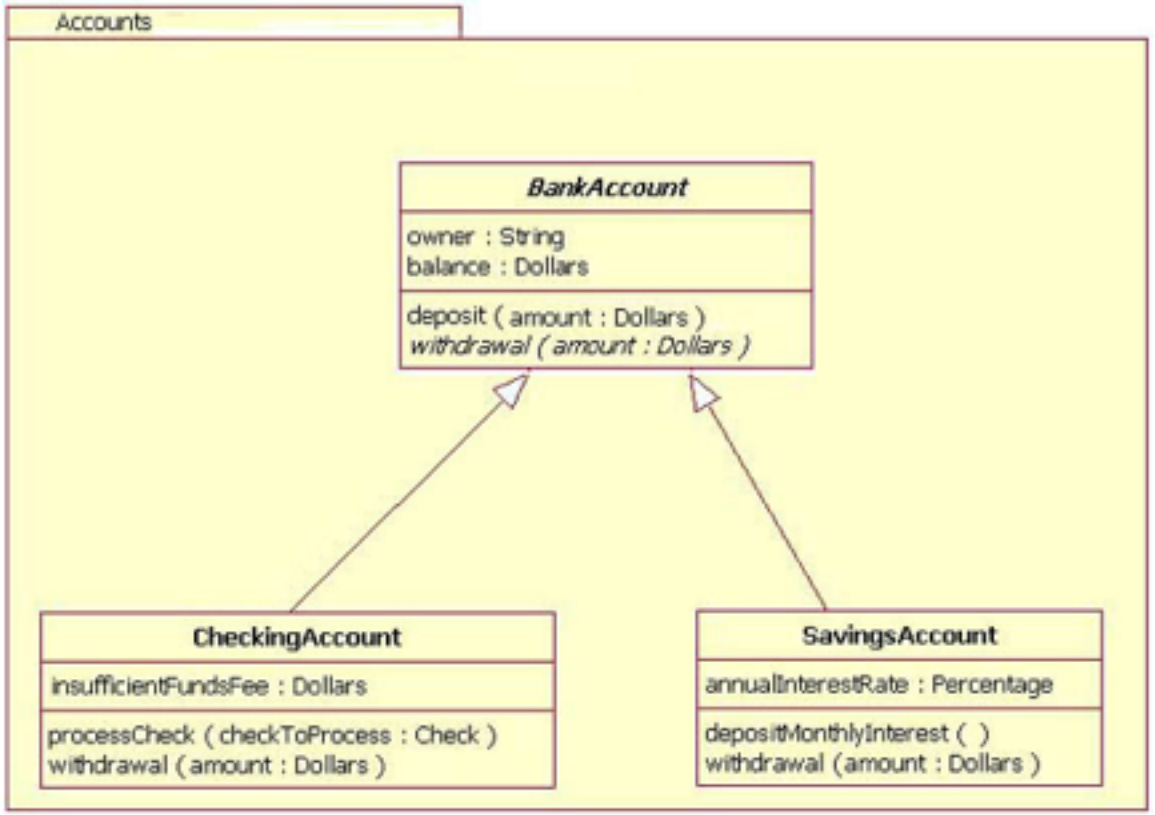
- An advanced type of association
- The contained object is *part* of the container
- Two types:
  - Aggregation: children can outlive parent

  - Composition: children's life depends on parent



Class Diagrams: Packages

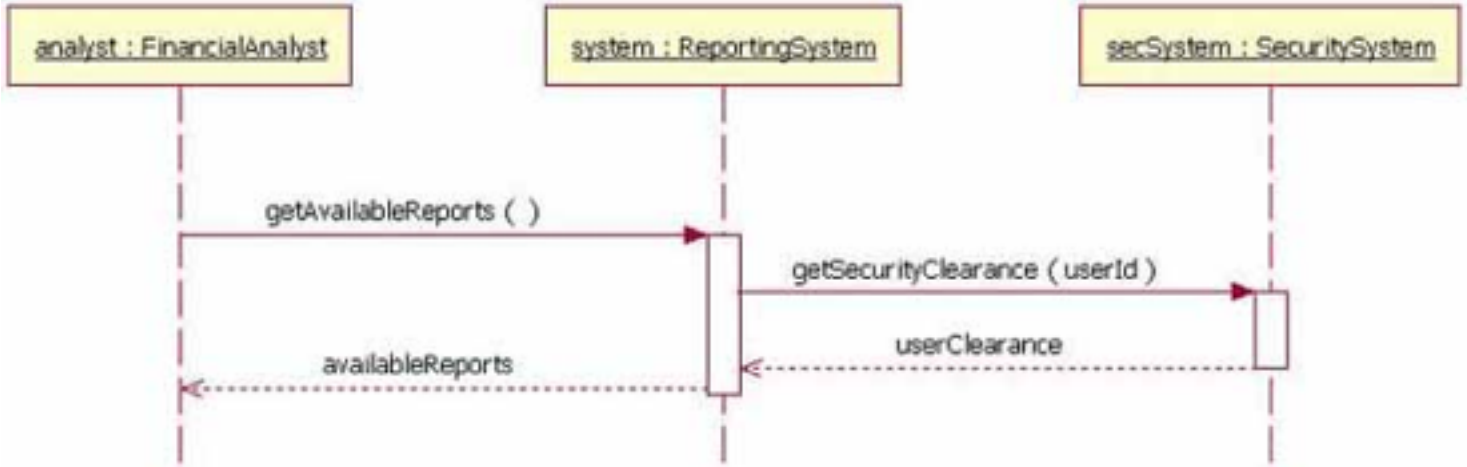
- Group classes together



Sequence Diagrams

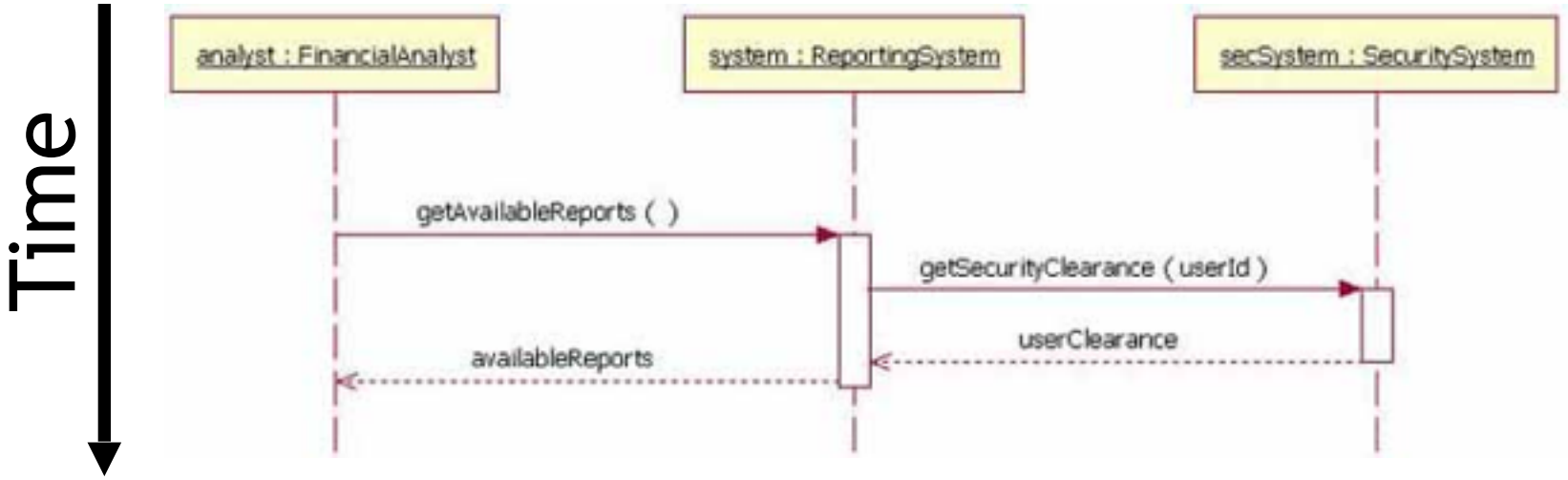
- Used to describe sequences of invocations between the objects that comprise the system
  - Focus less on *type of messages*, more on the *sequence* in which they are received
- Elements of UML sequence diagrams:
  - Lifelines
  - Messages

Sequence Diagrams: Lifeline



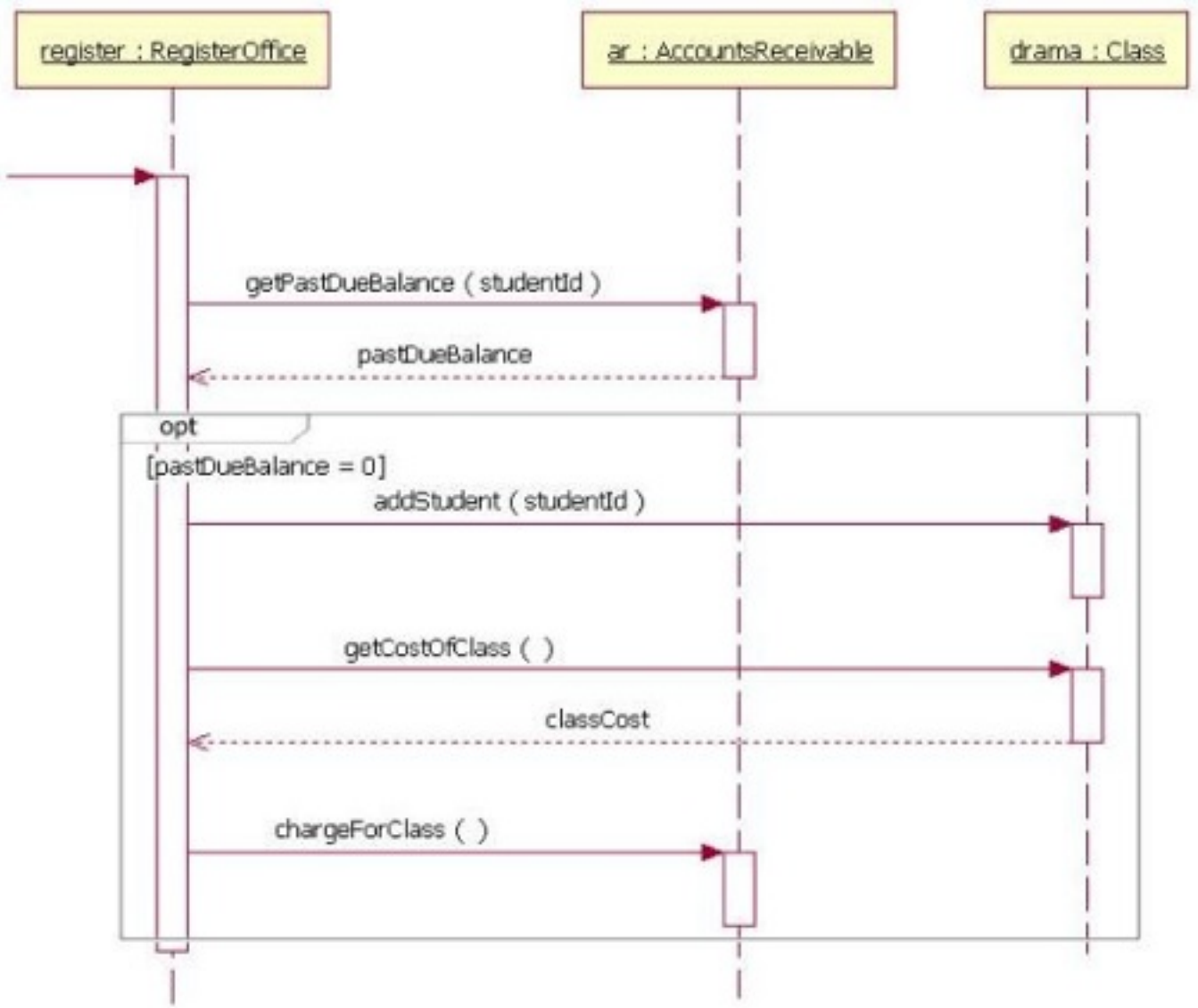
- Roles or object instances
- Participate in the sequence being modeled

Sequence Diagrams: Messages

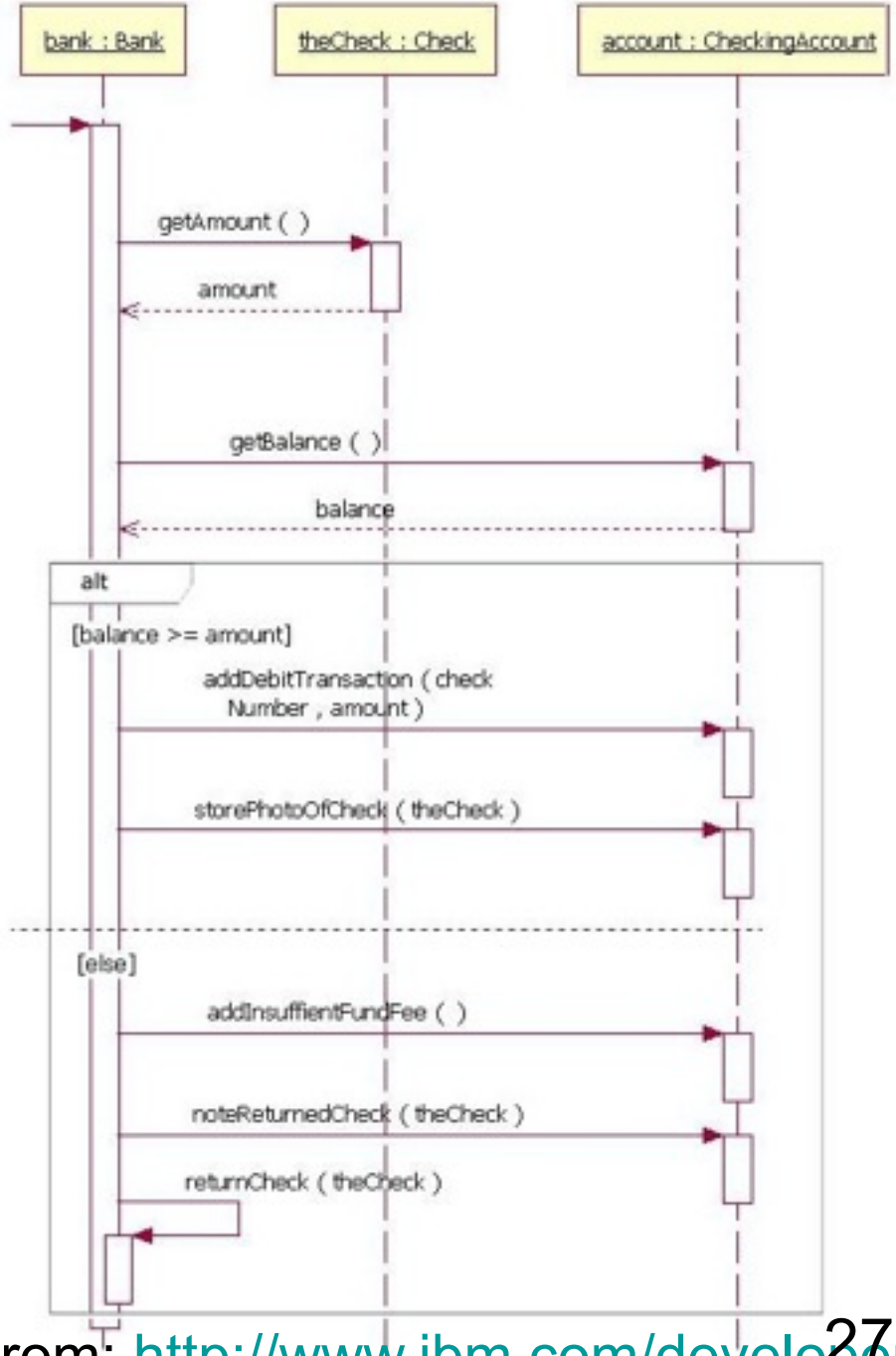


- Includes method name
- A box in the receiver's lifeline indicates activation (object's method is on the stack)
- Full arrow: synchronous (blocking)
- Optionally: information returned

Sequence diagram for conditionals



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

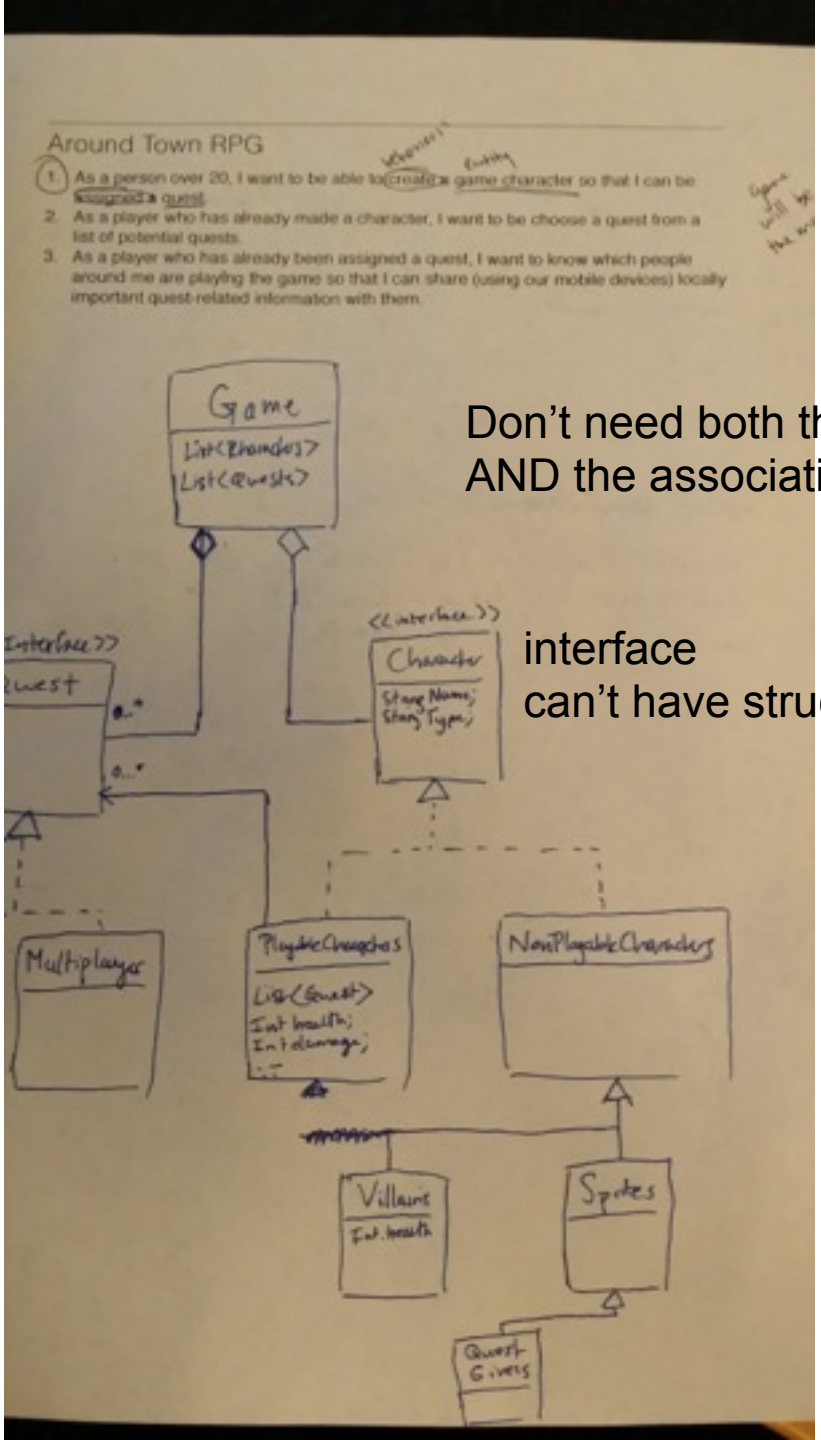


Sequence diagram when some actions are inside an if/else

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

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

Let's look at one…



Don't need both the list AND the association

interface
can't have structure

This becomes a "MultiplayerQuest"
Is that really what's intended?

Initial Design Very General Guidelines Summed Up

- **OO Generally:**
  - Classes ALWAYS start with a capital letter
  - Variables/methods NEVER start with a capital letter

- **Class Diagrams:**
  - Classes should ALWAYS be SINGULAR nouns or "doers"
  - Interfaces should ALMOST ALWAYS be roles
  - Interfaces never have fields, just method signatures.
  - Methods should ALWAYS be verbs or actions (unless they are constructors)
  - Abstract classes should ALMOST ALWAYS be abstract classifications of things
  - SubCLASSES should be true is-a's (try the concatenation test)
  - Concrete classes for interfaces (implementers) should pass the "acts as a" test.
  - Remember you don't need lists IN your classes in UML - just use an association
  - Aggregations/Compositions are reserved for whole-part relationships.

- **Sequence Diagrams:**
  - Looped/Conditional behaviour in is put in a box with the condition in the corner
  - Duration bars for methods last as long as the method is running
    (they don't break when the method calls away and restart when execution returns)
  - Duration bars stack if there is already a duration bar on that object at that timeslot