P2 Advanced Design Lab

- Intro
  - we will analyze what is good design
  - 95% people in industry don't know how to program
  - What is a program?
    - piece of code
    - executes a task
    - representation
    - design
    - solution to a problem
    - executable model of a solution to a problem
    - sequence of statements
    - computational simulation of a domain
    - why not "a bunch of objects"?
  - Programmer sees problem domain from a viewpoint
    - translates to a computational model
    - want to minimize the semantic gap
      - when things change in the real world, you need to make a similar change in your model
  - What is a good program?
    - fast
    - runs flawlessly
    - easy to use
    - fulfil requirements
    - flexible
      - easy to change or evolve
      - easy to maintain
  - Why objects? Why OOP?
    - easier to reduce semantic gap between program and reality
  - What is an OO program?
    - set of objects that collaborate by sending messages
- Telco modeling example
  - goal to come up with a maintainable design
  - need to find the objects
    - will use UML "object graphs"
  - Objects
    - aCall
      - Naming is important
        - Model an object's responsibility
          - What messages does it respond to?
        - For each responsibility, why do we need it?
      - price
        - Does this really belong here?
          - Perhaps a call should not know its price
      - caller
      - receiver
      - beginning
        - need to know if the call is at peak hour
      - duration
        - to compute cost
      - type
        - international
        - national
        - local
        - Later we will make this a responsibility of another class
    - aTelcoManager
      - "I worked in a company where, if you wrote a class with Manager in its name, you got fired."
        - End up with "a bag of cats"
        - name does not evoke its responsibility
    - aTelcoInvoiceSystem
      - To avoid the "bag of cats" problem, we model "the system" as a composition of other objects
      - Scenario
        - 1 International call
        - 2 Find price
        - 3 Find caller
        - 4 Prepare invoice
        - 5 Print it
      - processInvoices
    - aPriceCalculator
      - priceFor(aCall)
      - Need to know the type of the call to compute the price
      - If we make the price calculator an object, we can replace it by a different price calculator (!)
    - aCallClassifier
      - typeOf(aCall)
        - Call Types
          - anInternationalCallType
          - aNationalCallType
          - aLocalCallType
        - Instead of returning the type, we could return a classified call
          - more flexible
      - classify(aCall)
    - aClassifiedCall
      - combines a call and a call type
      - type
      - call
    - aPhoneNumber
      - country code
      - area code
        - problem: sometimes you can keep your number even if you move
          - a cell phone number does not tell you where you are
      - local number
    - to do: model the customer
    - final step is to print invoices
      - Should we model invoices as objects?
        - Yes: may need to correct one
          - Well, actually you get a new invoice!
          - should be immutable
          - be careful to reflect the domain in your model
        - Need to model it
        - Has responsibilities (contains calls ...)
    - what if a customer has no calls?
      - model as a null object