The UNIVERSITY of VERMONT

Benjamin Fenster
CS-100 (Damon)
17 April 2003

Notes – Design

I. Object Oriented Design
   a. Identify what classes you need
      i. What will they do?
      ii. What will they be called?
      iii. List them with summaries
   b. How will they be implemented
      i. Deciding on an implementation may require defining more classes.
      ii. Just add those to the list
   c. The Tradeoff
      i. Could spend tons of time designing at one extreme, or could completely wing it at the other extreme.
      ii. Want to design enough that you can comfortably sit down and write the code.
      iii. If you sit down and say, "What am I doing?" you haven't designed enough yet.
      iv. If you can't get to that point, try writing ap rototype first.
   d. Finding Classes
      i. There's a set that's "given" from a system standpoint.
      ii. Responsibility Driven Design
         1. Driven not by implementation but by the real problem.
         2. Don't base the design on "how it should be built."
         3. Identify Four Things
            a. Name Classes
            b. Responsibilities
               i. Methods
               ii. Fields
               iii. This is the most important step.
            c. Hierarchy
            d. Collaboration

II. Classes
   a. Highlight every noun phrase in the description of the problem.
   b. Be Wary of Adjective Phrases:
      i. May describe two different things.
      ii. May describe two uses of the same thing.
      iii. May mean nothing – linguistic fluff.
   c. Be Wary of Passive Voice
      i. It can mean that there's something missing from the sentence.
      ii. Read between the lines.
   d. Be Wary of External Objects
      i. Does the user, for example, need to be represented as a class?
      ii. Consider what objects are implied by the text.
   e. Based on that list, what should really become classes?
      i. Physical objects, for sure.
      ii. Conceptual entities
         1. Eg: A "borrowing" from a library.
         2. The relationship between things, maybe
      iii. Categories of things.
      iv. Model values of attributes, not the attributes themselves. (If you highlighted 1972, you want Year not 1972.)
      v. Be suspicious of everything else.
      vi. Combine classes that are really the same thing into one name.
   f. Make Index Cards
      i. Make one index card for each class chosen.
      ii. Put the class name at the top.

iii. Tape it to the wall. (Markerboard works great)

III. Responsibility
   a. What does it do?
   b. If nothing, kill it.
   c. The job may not be obvious but there must be something.
   d. Every VERB in the text can indicate a responsibility.
      i. "Remembers," "stores"
      ii. "Has" (a book HAS a ____)
   e. Write responsibility on the left side of the index card.
   f. If responsibilities are so numerous that they don't fit, there's something wrong.
   g. Be as general as possible.
   h. What class gets what responsibilities?
      i. Keep related responsibilities together.
      ii. Split up responsibility
      iii. Make classes work on themselves as much as possible, rather than on other classes.
   i. Some responsibilities may not get assigned. That may indicate that some new classes need to be added.

IV. Collaborations
   a. Almost no class is an island.
   b. Collaborations are just classes that implement some of the behavior of another class.
   c. Ex: Think of Geometry for a Robot.
   d. If one of your classes looks like it's too complicated to do as a single function, consider a collaboration.
   e. To the right of each responsibility, write the names of any classes that will help implement that particular responsibility.

V. Need to Choose Superclasses
   a. May already have a parent-child pair that's obvious.
   b. May have several classes that need a common superclass.
   c. Identify abstract classes.
   d. Try to avoid multiple inheritance as much as possible.
   e. Eliminate whatever isn't useful in the end. What's redundant?