Class 20: Design Principles

Held: Thursday, April 15, 2010

Summary: We consider some basic approaches to the design of classes. We also visit the current status of the project.

Related Pages:
- EBoard.

Notes:
- Reading for Tuesday: Chapter 9.
- Goal for next Thursday: Nine hours of work per person.
- I left out one of the types of problems for the final: There will also be a “write a test” problem.
- As you requested, I’ve removed the unused parts of the course web. Let me know if that seems better.

Overview:
- Exploring Logs.
- Design Principles.
- Alternatives to Inheritance.
- Small-group Exercise.
- Project Reflection.
- Integration.

Exploring Logs

Get out your logs and let’s talk.

- What were the primary tasks you spent your time on?
- How much time did you spend on each?
- How many classes did you write?
  - Simple classes
  - Complex classes
- How many lines of code did you write?
  - And how should we measure lines of code?
Design Principles

Principle 1: The Open-Closed Principle (OCP)

- What did you write for the “Sharpen Your Pencil” question on p. 381?
- What are reasons you might want to violate this principle (e.g., change the code in your base class)

Principle 2: The Don’t Repeat Yourself (DRY)

- Sam considers this one of the most important aspects of refactoring.
- Sam considers this one of the key reasons we have higher-order procedures.
- However, in the object model, we often use objects to encapsulate the repetition.
- How does the description of DRY on p. 385 differ from the model in XP?

Principle 3: The Single Responsibility Principle (SRP)

- The “do I do this to myself or do others do it to me” question seems to be the core idea.
- I’m not sure that I completely buy their model.

Principle 4: The Liskov Substitution Principle (LSP)

- Yay! Type theory intrudes on OO design
- Who is Liskov?

Some Other Techniques

- Inheritance isn’t the solution to every problem (duh).
- What are some other techniques for sharing behavior
  - Delegation: An object contains another object, and delegates the behavior to that object
  - Composition: Delegation plus inheritance (you can vary the behavior by varying what subclass you include)
  - Aggregation: Composition with independence
- Note: I consider the ”is there one reference or more than one reference to this object” a relatively minor matter.
  - I believe that it stems from a time in which you had to be careful in thinking about what to dispose of when you dispose of “an object”
  - In most modern OO systems, the garbage collector handles the issue
- Exercise: Draw a UML diagram for the five-minute mystery on p. 413.

Striking a Balance

- A number of questions about the past few readings have asked about the balance between Head-First “Generalize it All in Advance” and more XP-ish “Let the design emerge from the code”.
- I can only give you my own perspective.
- I have traditionally been a generalize person.
I believe that generalizations often emerge, so you might as well predict them.
I find generalizations beautiful.
I find generalizations intellectually challenging.
I know that a library of general functions helps me in the long term.

As time has passed, I’ve found that I see the value in a more XP-like approach.
For most of us, the time taken to generalize is not particularly helpful to the task at hand.
For many of us, the attempt to generalize can lead to anxiety.

So, I encourage you to generalize when it’s obvious and to use more XP-style programming when generalizations are not obvious.
But you must refactor mercilessly.

**Project Exercise**

- Three groups of four people. Each group needs to have a member of each of teams 1, 2, and 3 (or is that 0, 1, and 2)?
- Spend fifteen minutes discussing what you’ve accomplished.
- *Do not let any member/team dominate!*
- If you were forced to combine, what would you take from where, and why?

**Big Picture Issues**

- I’ve heard some comments that you are sharing code between groups between the weekly integration meetings.
  - My goal had been that you work independently between meetings.
  - What would you prefer?
  - If you’re going to share more regularly, then we should probably divide the project differently (e.g., a UI group, a Network group, and a Model group)
- I know that you had questions about the LocalWorld vs. NetworkedWorld models. We should go over my model for that and your alternatives.
- It’s probably time to work out the network protocol.