CSC207.01 2014S, Class 01: An Introduction to the Course

Overview

- Preliminaries.
  - Admin.
  - Homework.
  - Questions.
- About the Course.
- ADTs and Data Structures.
- An Exercise: Designing a List ADT.

Preliminaries

Admin

- Attendance!
  - I'll be taking attendance every day for the first few weeks. (I have about 120 students this semester, so it will take awhile for me to learn names.)
  - Please let me know your preferred gender pronoun (he, she, zhe) or preferred title (Mr., Ms., ...)
- The course Web is in rough form. (Sorry, it was a too-short break.) Expect to see it evolve over the next few days.
- In general, there are readings for every class and homework assignments that are due every Wednesday.
- I don't believe in going over class policies on the first day of class. Please read the course web and ask me questions via email or in person.
- I'm trying to set up a Google calendar for this course. Please let me know what else you'd like to see on that calendar.
- Extra credit:
  - MLK Day Talk, tonight @ 7:30 in JRC 101.
  - Thursday extra on summer research in CS, Thursday @ 4:30 in Noyce 3821.
  - CS Table Friday at noon.

Homework

- You have readings due for tomorrow's class. See the syllabus for details.
- Your first assignment is due Tusday night at 10:30 p.m. I've distributed it in class.
  - Read and write about the course web.
  - Take the RISC survey.

- You must also make a fifteen-minute appointment to talk to me some time this week.
    - I’ve brought a sign-up sheet to class.

Questions before we begin?

_Group work?

In "the real world", you work with other people. You might as well get used to it now.

You learn better working together. (Yes, there’s research to show this.)

I mix individual and group work.

About the Course

- The course where you transition from doing small programming tasks to being a real software designer and computer scientist
- A course in *formalized problem solving*.
    - Techniques for going from a problem statement to something that we believe solves the problem
    - And ways to express it formally.
- A course in *algorithms*
    - A formalized procedure for getting something done.
    - Writing them
    - Generalized techniques
    - "Classics" - the literature of CS
- A course in *abstract data types* and *data structures*
    - Abstract organizations of data: ADT
    - Concrete organizations of data: How do we lay things out in memory
- A course in *object-oriented software design*
    - In contrast to
        - Imperative - Direct, sequenced instructions. (Think about the "imperative voice" in a natural language.) Focus on the state of the machine.
        - Functional - Problem in terms of functions. Pure: think about building new values from old, rather than replacing values in place. Order of operations is implicit, rather than explicit.
    - Object-oriented programming
        - Focus on objects, which group data and operations
        - Think about individual objects that communicate
    - Three main parts of object oriented programming
        - Encapuslation: Group together data and operations AND protect them from the client code
            - Makes it easier to ensure that the code behaves correctly
            - Makes it easier to change the code, since we know what depends on the underlying representation
        - Inheritance: Base new code on old WITHOUT rewriting
        - Polymorphism: Write code that works with multiple related objects
            - E.g., NUMBER square(NUMBER x) { return x.multiply(x); }

- And we do this in Java

A few expectations about software design

- Refactor! If you find that you are writing the same code multiple times, write a more general procedure and use that.
- Document! People should be able to learn about your code, particularly your complex code, by reading English summaries.
    - Multiple kinds of documentation
        - Client documentation: WHAT the code does, not how it accomplishes it
        - Peer documentation: HOW the code achieves its goal / what the steps are
        - System documentation: How do the pieces fit together?
- Separate interface from implementation in your code as well as your documentation
- Start simple.

Warning: We're trying to theme this course

- Computing for social good
    - Narrow focus of crowdmapping using Ushahidi
- Using Android/mobile computing

ADTs and Data Structures

ADT Questions

- Start by coming up with an overall philsophy for how you want to arrange the data for the client.
    - List: Values in a sequence that we visit one by one
    - Array/vector: A group of values indexed by integer
- Identify some potential problem domains (cases) in which the ADT would be useful.
- Identify the methods that allow us to achieve the philosophy
    - List: front, advance, get, ... (build and map)
    - Array: set[i], get[i]

Implementation questions / Data structures

- How do we lay out the data in memory?
    - Like an array
    - As a bunch of small things linked together
- How do we implement the methods from above using that layout?
- Analyze for efficiency

Copyright (c) 2013-14 Samuel A. Rebelsky.

This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/3.0/` or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.