CS410/510
Advanced Programming

Mark P. Jones and Andrew P. Black

Your Instructors

- Mark P. Jones
  - email mpj, 'phone 503 725 3206
  - office hours: Mon 1-3pm or by apt

- Andrew P. Black
  - email black, 'phone 503 725 2411
  - office hours: Mon 4-5.30pm,
    Thu 10:30-11:50am

What is a Good Program?

What is this course about?

- Teaching you to become better programmers
  - Programs have two purposes:
    - To instruct a computer
      - So the program must be executable
    - To communicate with people
      - So the program must be an object of study
        - for tools, and
        - for programmers
  - Many students never learn the second purpose!
    - So we are going to focus on it

"After completion of this course, students will be able to:

1. Select appropriate names for their functions and methods.
2. Decompose methods and functions into components at the same level of abstraction.
3. Analyze a problem and determine what problem elements to represent as functions or objects.
4. Deal with complex data objects as whole entities, rather than by twiddling with their elements.
5. Effectively use parameterization and inheritance to promote reuse.

... continued:

6. Use closures to encapsulate computations.
7. Build recursive data structures and recursive operations over those data structures.
8. Use tools such as type systems, unit tests and random testing to guarantee the integrity of their programs.
9. Compose more complex programs from simpler parts.
10. Write the simplest possible program that solves a given problem while explaining to the reader how it solves that problem."

Key Ideas in Programming

- Whole Object
  - The reason to use high-level data structures is to manipulate them at a high level
- Composition
  - Build complex objects and processes from simpler parts;
  - apply this idea recursively
- Recursive Data Structures and their operations
- First class functions
- First class continuations
- Parameterization and Inheritance

Key Tools

- Test-driven Development (TDD)
  - SUnit, HUnit and Quickcheck
- Refactoring
  - Embrace change!
  - Don't plan for generality; you ain't gona need it
- Profiling
  - First get it right, and then make it fast
- Metaprogramming
- Literate Programming

Strive for Simplicity

What Is Simplest?

So, the definition of the best design is the simplest design that runs all the test cases. The effectiveness of this definition turns on: what do we mean by simplest?

Is the simplest design the one with the fewest classes? This would lead to objects that were too big to be effective. Is the simplest design the one with the fewest methods? This would lead to big methods and duplication. Is the simplest design the one with the fewest lines of code? This would lead to compression for compression's sake and a loss of communication.

Here is what I mean by simplest—four constraints, in priority order.

1. The system (code and tests together) must communicate everything you want to communicate.
2. The system must contain no duplicate code. (1 and 2 together constitute the Once and Only Once rule).
3. The system should have the fewest possible classes.
4. The system should have the fewest possible methods.

– Kent Beck: *Extreme Programming explained,* p. 109

What about the Programming Language?

- We believe that unnecessary complexity in the language gets in the way of writing elegant, simple, programs
- For most of the course, we will be using two specific languages, both rather small and elegant:
  - Haskell — pure, functional
  - Smalltalk — effectful, object-oriented
- Most programming techniques will be applicable to both languages!
- What you learn from programming in these languages will transfer to other languages

  ... even those not invented yet

Course Organization

- This is an experimental course (eXtreme Teaching)
  - *Don't* expect to find everything well-prepared far in advance
  - *Do* expect us to respond to feedback as the course evolves
- Rapid Feedback is important
  - Small assignments every week, sometimes every class meeting
  - Programming in class
- We emphasize code that "speaks" to us
  - Come to class prepared to talk about your code
  - Be open to criticism: almost all code can be improved upon

Provisional Schedule:

Week	Dates	Topics
1	January 6, 8	Introductions: Instructors, Course, Haskell, Smalltalk
2	13, 15	Test Driven Development, Unit Testing, QuickCheck
3	20, 22	Composition, Whole Object Programming
4	27, 29	Recursive Data Structures, Polymorphism, Inheritance
5	February 3, 5	Extended Example: Finite State Machines
6	10, 12	... continued
7	17, 19	Metaprogramming, Aspect-oriented Programming
8	24, 26	Higher-order Functions, Blocks, Parsers, Continuations
9	March 3, 5	Profiling
10	10, 12	Additional Topics

Assessment

- Weekly homework assignments (sometimes in groups)
- Project, starting around the end of week 6, in small groups, presentations in week 10 and finals week
- No midterm, no final!