CSE 331
Software Design & Implementation

Hal Perkins

Spring 2014

Lecture 1 – Introduction & Overview

Welcome!

We have 10 weeks to move well beyond novice programmer:

- Larger programs
  - Small programs are easy: "code it up"
  - complexity changes everything: "design an artifact"
  - Analogy: using hammers and saws vs. making cabinets (but not yet building houses)
- Principled, systematic software: What does "it's right" mean? How do we know "it's right"? What are best practices for "getting it right"?
- Effective use of languages and tools: Java, IDEs, debuggers, JUnit, JavaDoc, Subversion, …
  - Principles are ultimately more important than details
    - You will forever learn details of new tools/versions

Concise to-do list

By tomorrow night:

1. Trying to add? Sign sheet before leaving today
2. Familiarize yourself with website:
   http://courses.cs.washington.edu/courses/cse331/14sp/
3. Read syllabus and academic-integrity policy
4. Fill in office hours doodle
5. Post something to the discussion board ("welcome" followup)
6. Do Homework 0 (see homework calendar)

Who: Course staff

- Lecturer:
  - Hal Perkins: CSE faculty since sometime during the last millennium, fifth time teaching CSE331
- TAs:
  - Karina Jain
  - Alex Mariakakis: 331-staff veteran, sections
  - Vinod Rathman: 331-student veteran
  - Sarah Wei: another 331-student veteran
- Office hours will be figured out ASAP

Get to know us!

- Make sure this *feels like* a 40-person class with 90 students
- We're here to help you succeed!

Acknowledgments

- Course designed/created/evolved/edited by others
  - Michael D. Ernst
  - David Notkin
  - Dan Grossman
  - A couple dozen amazing TAs

Goals

- CSE 331 will teach you to how to write correct programs
- What does it mean for a program to be correct?
  - Specifications
- What are ways to achieve correctness?
  - Principled design and development
  - Abstraction and modularity
  - Documentation
- What are ways to verify correctness?
  - Testing
  - Reasoning and verification

Main topic: Managing complexity

- Abstraction and specification
  - Procedural, data, and control flow abstractions
  - Why they are useful and how to use them
- Writing, understanding, and reasoning about code
  - Will use Java, but the issues apply everywhere
  - Some focus on object-oriented programming
- Program design and documentation
  - What makes a design good or bad (example: modularity)
  - Design processes and tools
- Pragmatic considerations
  - Testing
  - Debugging and defensive programming
  - Managing software projects (more in CSE 403, not here)

The goal of system building

- To create a correctly functioning artifact
- All other matters are secondary
  - Many of them are ***essential*** to producing a correct system
- We insist that you learn to create correct systems
  - This is hard (but fun and rewarding!)

Related skill: *communication*

- Can you convince yourself *and others* something is correct via precise, coherent explanations?

Why is building good software hard?

- Large software systems are enormously complex
  - Millions of "moving parts"
- People expect software to be malleable
  - After all, it's "only software"
- We are always trying to do new things with software
  - Relevant experience often missing

- Software engineering is about:
  - Managing complexity
  - Managing change
  - Coping with potential defects
    - Customers, developers, environment, software

Programming is hard

- It is surprisingly difficult to specify, design, implement, test, debug, and maintain even a simple program
- CSE 331 will challenge you
- If you are having trouble, *think* before you act
  - Then, look for help
- We strive to create assignments that are reasonable if you apply the techniques taught in class…
  … but likely hard to do in a brute-force manner
  … and almost certainly impossible to finish if you put them off until a few days before they're due

Prerequisites

- Knowing Java is a prerequisite
  - We assume you have mastered 142 and 143

Examples:

- Sharing:
  - Distinction between == and equals()
  - Aliasing (multiple references to the same object)
- Object-oriented dispatch
  - Inheritance and overriding
  - Objects/values have a run-time type
- Subtyping
  - Expressions have a compile-time type
  - Subtyping via `extends` (classes) and `implements` (intefaces)

Lectures and section

- 3 lectures + 1 section each week
  - All required

- Website: http://www.cs.washington.edu/331
  - Most course materials posted but these are visual aids
  - Arrive punctually and pay attention – take notes(!)
  - If doing so doesn't save you time, one of us is messing up

- Section will often be more tools and homework details
  - Particularly next few weeks preparing for projects

Staying in touch

- Message (discussion board)
  - Keep in touch with colleagues and us
  - Post a reply now to the first message & it will keep track of new messages for you

- Course staff: cse331-staff@cs.washington.edu

- Mailing list: messages from course staff to everyone (you are subscribed if you are enrolled; you are responsible for messages sent to this list)

Requirements

- Primarily programming assignments but some written problem sets, approximately weekly (55%)
- 1 midterm (15%), 1 final (25%)
- 5% online quizzes, exercises, citizenship, etc.
- Collaboration: individual work unless announced otherwise; *never* look at or show your code to others
  - But talk to people, bounce ideas, sketch designs, …
- Extra credit: when available, small effect on your grade if you do it – no effect if you don't

- We reserve the right to adjust percentages as the quarter evolves to reflect the workload

Homeworks

- Biggest misconception about CSE331 (?)

  "Homework was programming projects that seemed disconnected from lecture"

  – If you think so, you are making them harder!
    - Reconsider
    - Seek out the connections by thinking-before-typing
    - Approaching them as CSE143 homework won't work well
    - Don't keep cutting with a dull blade

- First couple assignments are "more on paper", followed by software development that is increasingly substantial

Deadlines

- Turn things in on time!
- But things happen, so …
  - You have 4 late days for the quarter for assignments (not quizzes, exercises)
  - No more than 2 per assignment
  - Counted in 24 hour chunks (5 min = 24 hours late)
- That's it. No other extensions (but contact instructor if you are hospitalized)
- Advice: Save late days for the end of quarter when you (might) really need them

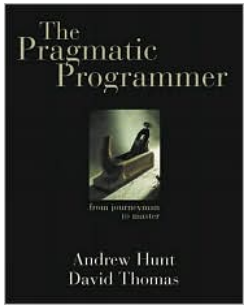
Academic Integrity

- Policy on the course web. **Read it!**
- Do your own work – always explain any unconventional action on your part
- I trust you completely
- I have no sympathy for trust violations – nor should you
- Honest work is the most important feature of a university (or engineering, or business). It shows respect for your colleagues *and yourself.*

Resources – Books

Required (assigned readings, short quizzes)

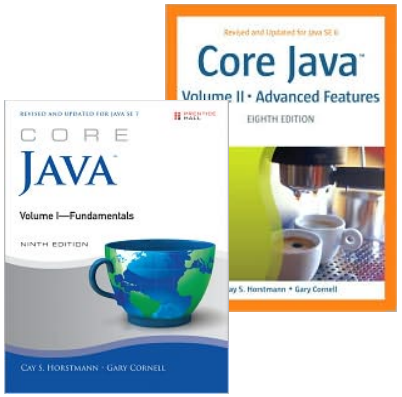
- *Pragmatic Programmer*, Hunt & Thomas
- *Effective Java* 2nd ed, Bloch



Every serious programmer should study both of these

Decent "Java book" if you want one

- *Core Java* Vol I, Horstmann



And use the Java API Docs

Readings (and quizzes)

- These are “real” books about software, approachable in 331
  - Occasionally slight reach: accept the challenge
- Overlap only partial with lectures
- Want to make sure you “do it”
  - Reading and thinking about software design is essential
    - Books seem expensive given your budget, but very cheap as a time-constrained professional
  - Will have some simple online reading quizzes
    - Frequency and schedule to-be-determined; no late days
  - Material is fair-game for exams

Books? In 2014?

- Why not just use Google, Stack Overflow, Reddit, Quora, …?
- Web-search good for:
  - Quick reference (What is the name of the function that does …? What are its parameters?)
  - Links to a good reference
- (can be) Bad for
  - Why does it work this way?
  - What is the intended use?
  - How does my issue fit into the bigger picture?
- Beware:
  - Random code blobs cut-and-paste into your code (why does it work? what does it do?)
  - This inscrutable incantation solved my problem on an unstated version for no known reason

Exams

- Midterm: date announced soon, in class
- Final: Tuesday June 10, 2:30-4:20 pm
- All the concepts, different format than homework
  - Will post old exams from prior quarters later

You have homework!

- Exercise 0, due online by 10 am Wednesday
  - Links went live right before class
  - No late submissions
  - Write (don't run!) an algorithm to rearrange (swap) the elements of an array
  - And argue (prove) in concise, convincing English that your solution is correct!
- Why?
  - Great practice
  - Surprisingly difficult
  - Might find some of the things we learn helpful for this – as well as for much larger projects ☺

CSE 331 is hard!

- You will learn a lot!
- Be prepared to work and to think
- The staff will help you learn
  - And will be working hard, too

- So let's get going…
  - Before we create masterpieces we need to hone our ability to reason very precisely about code….

Example

"Complete this method such that it returns the index of the max of the first **`n`** elements of the array **`arr`**."

```
int index_of_max(int[] arr, int n) {
    …
}
```

Example

"Complete this method such that it returns the index of the max of the first **`n`** elements of the array **`arr`**."

```
int index_of_max(int[] arr, int n) {
    …
}
```

What questions do you have about the *specification*?

Given a (better) specification, is there 1 *implementation*?

Moral

- You can all write the code

- More interesting in CSE331:
  - What if n is 0?
  - What if n is less than 0?
  - What if n is greater than array length
  - What if there are “ties”?
  - Ways to indicate errors: exceptions, return value, …
  - Weaker versus stronger specifications?
  - Hard to write English specifications (n vs. n-1)