CSE 331
Software Design & Implementation

Hal Perkins

Winter 2023

Lecture 1 – Introduction & Overview

(Based on slides by Mike Ernst, Dan Grossman, Kevin Zatloukal, James Wilcox, me, many others)

What is CSE 331 about?

- It's about 10 weeks…
- It's lots of Java programming…
- It's weird Java language stuff like generic type bounds…
- It's git and gradle and Intellij and junit (and I already know all that stuff from my internship)…
- It's JavaScript and TypeScript and React, but I'm already a Full-Stack®™℠ Developer!...

- Yes, we'll do those things, but it's not (primarily) those things
  - It's the "why" behind the "what"
  - It's the rest of the course that goes with the lab (think chemistry, physics, mechanical/civil engineering, etc.)

What is the goal of CSE 331?

How to build harder-to-build software

- Move from CSE 143 problems toward what you'll see in upper-level CSE courses and in "real world" (industry, etc.)
- Move beyond coding to software design and development (and coding is actually a fairly small part of that)

Specifically, how to write programs of

- Higher **quality**
- Increased **complexity**

We will discuss *tools* and *techniques* to help with this and the *concepts* and *ideas* behind them

  - There are *timeless principles* to both
  - Widely used best practices in software industry

What is high quality?

Code is high quality when it is

1. **Correct**
   - Everything else is of secondary importance

2. Easy to **change**
   - Most work is making changes to existing systems

3. Easy to **understand**
   - Needed for 1 & 2 above

How do we ensure correctness...

... when **people** are involved?

People have been known to

- walk into windows
- drive away with a coffee cup on the roof
- drive away still tied to gas pump
- lecture wearing one brown shoe and one black shoe

What is increased complexity?

Analogy to building physical objects:

- 100 well-tested LOC = a nice cabinet
- 2,500 LOC = a room with furniture
- 2,500,000 LOC = 1000 rooms ≈

North Carolina class WW2 battleship

≈

the entire British Naval fleet in WW2

Actually, software is more complex…

- Every bit of code is unique, individually designed
  - US built 10 identical Essex carriers

  - Software equivalent would be one carrier 10 times as large:

- Defects can be even more destructive
  - A defect in one room can sink the ship
  - But a defective OS could sink the *whole fleet*

How do we cope with complexity?

We tackle complexity with **modularity**

- Split code into pieces that can be built independently
- Each must be documented so others can use it
- Also helps understandability and changeability

Scale makes everything harder

Modularity makes scale **possible** but it's still **hard**…

- Time to write N-line program grows faster than linear
  - Good estimate is $O(N^{1.05})$ [Boehm, '81]
- Bugs grow like $\Theta(N \log N)$ [Jones, '12]
  - 10% of errors are between modules [Seaman, '08]
- Communication costs dominate schedules [Brooks, '75]
- Small probability cases become high probability cases
  - Corner cases are more important with more users

Corollary: quality must be even higher, per line, in order to achieve overall quality in a *large* program

People Do Build Great Software

Full scope of the challenge:

- software is built by people, who make mistakes all the time
- surprisingly difficult to get even a small program to work
- needed to write hundreds of millions of lines of code
- each line gets harder to write as the program scale

Despite those challenges, we have lots of software that works

- hundreds of millions of lines of working programs
- products rarely fail because the software is too buggy

How do we do it?

How do we ensure correctness...

... when **people** are involved?

People have been known to

- walk into windows
- drive away with a coffee cup on the roof
- drive away still tied to gas pump
- lecture wearing one brown shoe and one black shoe

Key insights:

- Can't stop people from making mistakes
- **Can** stop mistakes from getting to users

How do we ensure correctness?

Best practice: use three techniques (we'll study each)

1. **Tools**
   - Type checkers, test runners, etc.
2. **Inspection**
   - Think through your code carefully
   - Have another person review your code
3. **Testing**
   - Usually >50% of the work in building software

Each removes ~2/3 of bugs. Together >97%

What is high quality code?

In summary, we want our code to be:

1. Correct
2. Easy to change
3. Easy to understand
4. Easy to scale (modular)

These qualities also allow for increased complexity

What we will cover in CSE 331

- Everything we cover relates to the 4 goals
- We'll use Java but the principles apply in any setting

Correctness

1. Tools
   - Git, IntelliJ, JUnit, Javadoc, …
   - Java libraries: equality & hashing
   - Adv. Java: generics, assertions, …
   - debugging
2. Inspection
   - reasoning about code
   - specifications
3. Testing
   - test design
   - coverage

Changeability

- specifications, ADTs
- listeners & callbacks

Understandability

- specifications, ADTs
- Adv. Java: exceptions
- subtypes

Modularity

- module design & design patterns
- event-driven programming, MVC, GUIs

Administrivia

See course syllabus for full details

Who: Course staff

- Instructor:
  - Hal Perkins (long-time CSE 331 veteran)
- TAs:
  - ~15 great TAs, mix of veterans and new folks

- Office hours posted now – starting today!

Get to know us!

  - We're here to help you succeed

Who: Students

- Assuming you have mastered intro (through CSE 123 or CSE 143)
  - Will review things quickly, but not re-teach them
- Some connections with CSE 311
  - But not required – we cover everything we need
- Assuming you are fairly new to the Allen School
  - Good course to take *before* that internship
  - May be more redundant if you're about to graduate, but you ~~probably~~ definitely will see new connections between things – "ah, that's why they do that…"

Prerequisites

- Knowing Java is a prerequisite

Examples:

- Difference between `int` and `Integer`
- Distinction between `==` and `equals()`
- Aliasing: multiple references to the same object, what does assignment (`x=y;`) really mean?
- Subtyping via `extends` (classes) and `implements` (interfaces)
- Method calls: inheritance and overriding; dynamic dispatch
- Difference between compile-time and run-time type

Staying in touch

- "ed" Message Board for general discussion and announcements
  - You should have already received an invitation
  - Join in! Staff will read/contribute too
  - Help each other out and stay in touch outside of class
- "ed" Message Board for debugging/assignment help
  - Use private ed messages

- Gradescope: assignment feedback and regrade requests here

- Course staff: **`cse331-staff@cs.washington.edu`**
  - For things that don't make sense to post on message board
    - Personal situations, project rerun questions, etc.
  - Please do *not* send messages to individual staff if possible – easier to route and follow up if it goes to the list

Lecture and section

- Both required and attendance assumed

- All materials posted, but they are visual aids
  - Attend regularly and pay attention (& take notes!)
  - If doing so doesn't save you time, one of us is messing up (!)
  - Lectures recorded for review, but no substitute for attending

- Section will often be more tools and homework-details focused
  - Held the day most HW is released – should help you get started
  - Demos/tutorials will be recorded or we'll have videos
  - Group work and discussions not recorded

Laptops, phones & gadgets in class?

- **No**
- Why? You'll learn more/better if you are mentally present in the room, not just physically
  - And gadgets *really* distract your neighbors
- Exception: if you *actually* use a tablet or something to take notes (but paper is better for learning!)
- Exception: some sections where we will be using laptops to set up tools / projects
- Urge to search? Ask a question!!
- You may close/silence/put away your gadgets now
- We thank you for your help

Homework Assignments

- Biggest misconception (?) about CSE 331

  "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 like intro homework won't work well
    - Don't keep cutting with a dull blade

- First few assignments are on paper "like math problems", followed by software development and related problems that are increasingly substantial

- Deadlines: **11 pm** **not** 11:59 pm.

Late Policy

- Assignments must be submitted by deadline. Full stop.
- But, stuff happens (bugs, computer crashes, …)
- So:
  - Up to **4** times this quarter you can turn in a homework assignment **one** (**1**) day late ***=>max<=***
  - Late days are 24-hour chunks
  - That's it. Not accepted for credit after that.
    - Requests for "extensions" later in the quarter because of bad time management earlier are not looked on with favor
    - Do not plan to use late days – save for unexpected things
- Why?
  - Keep you on schedule (most important)
  - Allow staff to get feedback to you before next deadline
- This is almost certainly different from what you're used to. No excuses for not knowing what the policy is.

But: Unusual Situations

- We're still not back to the pre-covid world (and may never get there) and we're all dealing with the world differently

- We will do our best to work with you, but please contact course staff or the instructor in advance if you can (unless you can't because of a true emergency)
  - Please reach out early – don't let things fester until it's late and much harder to fix
  - We have a lot of flexibility to make things work

- And a reminder: if you're sick, *stay home* ! *get well* !!
  - And get in touch so we can help you keep up

Reruns for Programming Assignment Staff Tests

- You can ask staff to rerun automated correctness tests for programmng assignments once after the original feedback is released
  - do this via email to cse331-staff[at]cs ***only***

- Aim of the policy is to limit the deductions for minor mistakes that end up causing a disproportionate / catastrophic number of test failures

- We will re-calculate the correctness (staff tests) score up to a maximum of 80% of the original max possible
  - other scores (design, code quality, etc.) are not changed

Academic Integrity

- "The code you submit must be your own"
  - no copying from other students, web homework solutions, Chegg, ChatGPT, Codex, AlphaCode etc.
- Read the full course policy carefully
  - ask questions if you are unsure
- Always explain in your HW any unconventional action
  - worst result then is some points lost
- Violations are unfair to other students and yourself

Exams

- Goal: need to review/digest what we learn
- We will have a regular midterm and final exam
  - Midterm: late afternoon, Tuesday, Feb. 7
    - One hour sometime between 4:30 and 6:30
  - Final exam: finals week, still working on day/time
  - Everyone in all sections will take exams at the same time
  - Will try to pin down exact details quickly
- May supplement this with some lightweight quizzes, especially on readings (still to be determined)

Grading

Approximate weighting (subject to change):

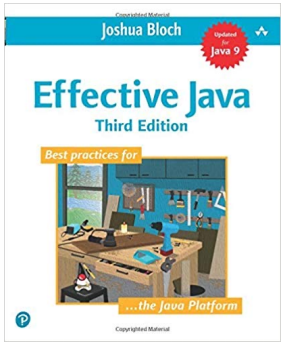
65%	Homework, projects
10%	Midterm
15%	Final
5%	Quizzes, other

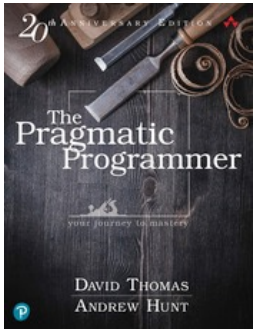
Project feedback may be different than you're used to seeing: multiple dimensions (design, documentation, testing, code quality) with qualitative feedback for each (superior; good; needs major improvement; no credit – later two of these should be rare)

Books

Required textbooks

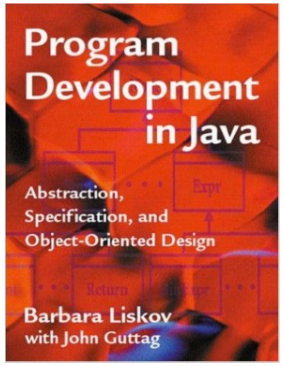
- *Effective Java* 3rd ed, Bloch (EJ)
- *Pragmatic Programmer*, 20th anniversary (2nd) edition, Hunt & Thomas (PP)

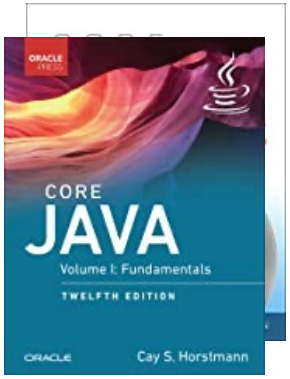




Other useful books:

- *Program Development in Java*, Liskov & Guttag
  - would be the textbook if not from 2001
- *Core Java* Vol I, Horstmann
  - good reference on language & libraries





Readings (and quizzes)

- These are “real” books about software, approachable in 331
  - Occasionally slight reach: accept the challenge
- Overlap only partial with lectures – readings related to class topics on the lecture calendar (listed by item/topic #s)
- Important to “do it”
  - Reading and thinking about software design is essential
    - Books may seem expensive given your budget, but very cheap as a time-constrained professional
    - But: you can access them free!! online (see syllabus)
  - Quizzes (if we have them) & exams ☺

Books? In the 21st century?

- Why not just use Google, Stack Overflow, Reddit, Quora, …?
- Web-search good for:
  - Quick reference (What is the name of the function that does … in Java? What are its parameters?)
- (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:
  - Answers on the web are often **quickly** out of date (especially software configuration information)
    - aim is to answer the question at the time when asked
  - "This inscrutable incantation solved my problem on an unknown version for no known reason"

CSE 331 can be challenging

- Past experience tells us CSE 331 is **hard**
  - not our intention to make it difficult!
- Big change to move
  - **from** programming by trial & error
    - this does not work for building large scale software
  - **to** programming by careful design, reasoning, and testing
- Programming itself can be hard
  - surprisingly difficult to specify, design, implement, test, debug, and maintain even a simple program

CSE 331 can be challenging

- We strive to create assignments that are reasonable if you apply the techniques taught in class…
  - … but likely hard to do in a trial & error manner
    - … and almost certainly impossible to finish if you put them off until a few days before they're due

- Assignments will take more time than you think (**start early**)
  - even professionals *routinely* underestimate by 3x
  - these assignments will be a step up in difficulty
  - aim to finish a day early and leave yourself 24 hrs. to review and fix any last-minute problems

- If you are having trouble, *think* before you act
  - then, look for help

Other Advice

- Don't be afraid to make mistakes
  - accepting that you will make mistakes is perhaps the most important lesson of this course
  - we often learn best from our mistakes
  - if you're not making mistakes, you're not challenging yourself
    - "Never promote someone who hasn't made some bad mistakes because, if you do, you are promoting someone who has never done anything" — Dr. Herbert Dow (founder of the Dow Chemical Company)

- Don't expect everything to be spelled out for you
  - real-world problems don't come that way
    - if there are detailed instructions for solving a problem, then there should already be a program that does it
  - world needs you for your intuition, creativity, & intelligence

You have homework!

- Homework 0, due online by **10AM Friday** (no late days)
  - Write (don't run!) an algorithm to rearrange (swap) the elements in an array
  - in O(*n*) time (and preferably in a single pass)
  - And argue (prove) in concise, convincing English that your solution is correct!
  - (Pretend that you are presenting this on a whiteboard to someone – you cannot run the code!)

- Purpose:
  - Great practice
  - Surprisingly difficult (and useful calibration on what's easy!)
  - Help us start thinking about how we can reason about code and build code that works as intended

Concise to-do list

Before next class (i.e., sections tomorrow):

1. Familiarize yourself with website, do readings
   - Lecture slides will be posted on web evening before class
2. Read syllabus and academic-integrity policy
3. Do Homework 0 (see web calendar), due by **10AM Friday**! (no late days on this time)
   - Use gradescope to submit pdf copy (scan or use your phone)
   - Gradescope accounts will be created for everyone later today
   - (send mail to cse331-staff with name, ID #, and UW Email address if not registered so we can add you to the gradescope course roster to turn in the assignment)
4. Go to section tomorrow! We're starting right away with key content (especially because of Wed. quarter start)