CSE 331
Software Design & Implementation

Kevin Zatloukal

Spring 2022

Lecture 1 – Introduction

“Software is *eating the world*.” (2011)

— Marc Andreessen (VC)

“This isn’t the beginning of the end. It’s not even the end of the beginning.” (2021)

— Alex Rubalcava (Software VC)

“There are just not enough software engineers for everyone.
… and this is not going away. It’s getting worse, actually” (last week)

— Olivier Pomel (CEO of Datadog)

Agenda

1. Motivation
2. Administrivia
3. Advice
4. HW0

Motivation

What are the goals of CSE 331?

Learn the skills to be able to contribute to a modern software project

- move from CSE 143 problems toward what you'll see in industry and in upper-level courses

Specifically, how to write code 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 across the 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

Key Insight

1. Can’t stop people from making mistakes

Scale of Modern Software Projects

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

A
B
C
D
E
F
G
H
I
J
K
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
DIDO
EURYALUS
NAIAD
PHOEBE
SIRIUS
BONAVENTURE
HERMIONE
CHARYBDIS
CLEOPATRA
SCYLLA
EDINBURGH
BELFAST
MAURITIUS
NIGERIA
CEYLON
GAMBIA
JAMAICA
UGANDA
FIJI
KENYA
ADELAIDE
VINDICTIVE
FROBISHER
HAWKINS
EFFINGHAM
AURORA
GALATEA
PENELOPE
ARETHUSA
SYDNEY
HOBART
PERTH
ORION
NEPTUNE
ACHILLES
AJAX
LEANDER
DORSETSHIRE
NORFOLK
SHROPSHIRE
SUSSEX
DEVONSHIRE
LONDON
AUSTRALIA
KENT
CANBERRA
MANCHESTER
LIVERPOOL
GLOUCESTER
GLASGOW
BIRMINGHAM
SHEFFIELD
NEWCASTLE
SOUTHAMPTON
BERWICK
SUFFOLK
CUMBERLAND
CORNWALL
EXETER
YORK
DUKE OF YORK
PRINCE OF WALES
KING GEORGE V
JELLICOE
BEATTY
ILLUSTRIOUS
VICTORIOUS
FORMIDABLE
INDOMITABLE
IMPLACABLE
MARSHAL SOULT
ARGUS
ALBATROSS
TERROR
EREBUS
EAGLE
HERMES
GLORIOUS
COURAGEOUS
FURIOUS
ARK ROYAL
RENOWN
REPULSE
22 GUNBOATS
HOOD
3 LUPIN class
ROYAL SOVEREIGN
15 LEADERS
3 LATONA class
WHITLEY class
RAMILLIES
V and W classes
RESOLUTION
70 ACASTA type
17 HALCYON class
15 ODIN PARTHIAN RAINBOW classes
ROYAL OAK
16 TRIBAL class
23 HUNT class
16 JAVELIN class
5 EGRET
3 BITTERN classes
8 LIGHTNING class
12 HASTINGS INDUS HINDUSTAN
10 SCOUT class
10 GRIMSBY
OSCAR PARKES
15 TRITON class
TEMERAIRE
LION
QUEEN ELIZABETH
VALIANT
BARHAM
MALAYA
WARSPITE
12 SHARK class
REVENGE
3 THAMES class
RODNEY
NELSON
3 UNITY class
9 KINGFISHER class

Scale makes everything harder

Many studies showing scale makes quality harder to achieve

- Time to write N-line program grows faster than linear
  - Good estimate is O($N^{1.05}$) [Boehm, ‘81]
- Bugs grow like Θ(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

Full Scope of the Challenge

Problem facing us

- 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

1. Can’t stop people from making mistakes
2. 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

interviews focus on this (a.k.a. "reasoning")

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

How do we cope with scale?

We tackle increased software scale 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

What are the goals of CSE 331?

In summary, we want our code to be:

1. Correct
2. Easy to change
3. Easy to understand
4. Modular

These qualities also allow us to solve more complex problems

- increased complexity = larger scale and sophistication

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

Who: Course staff

- **Instructor**: Kevin Zatloukal (kevinz at cs)
  - 15 years in industry (30 years of programming)
  - 7th year teaching

- 17 great **TA**s
  - mix of new and veteran

- Office hours posted soon
  - (starting later this week)

Get to know us!

- We're here to help you succeed

Who: Students

- Assuming you have mastered CSE142 and CSE143

- Hoping (but not assuming) have you taken 311
  - will connect to 311 material where it arises

- Assuming you are in your first year of CSE major courses
  - seniors may be bored

Prerequisites

- Basic Java knowledge is a prerequisite

Examples:

- Difference between **`int`** and **`Integer`**
- Distinction between **`x == y`** and **`x.equals(y)`**
  - multiple references to the same object (aliasing)
  - 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 (link on course web page)
  - should have access already
  - best place to ask questions

- Course staff: **`cse331-staff@cs.washington.edu`**
  - for things that don't make sense to post on message board
  - also fine to email me directly for private matters

- Course email list: **`cse331{a,b}_sp22@u.washington.edu`**
  - students already subscribed (your UW email address)
  - for me to email you… do not "reply all"
  - infrequent, but important emails

Lectures

- In person lectures focused on **key ideas**

- Morning section will be recorded
  - recordings available on Canvas

- Don’t fall into the trap of skipping lectures to work on HW
  - can spiral into falling further and further behind in class

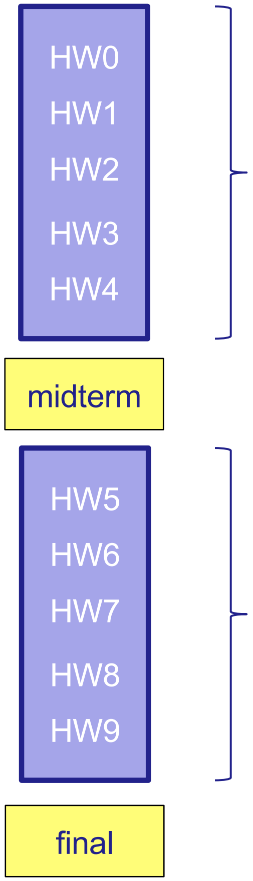
Section

- Will be focused on **helping with homework**
  - typically fall on day when a new HW is released
  - get you get you started with the work to be done
  - they should be very useful

- <u>Not recorded</u>
  - materials will be posted

Homework Assignments

- Exactly 1 assignment per week (10 total)



HW0
HW1
HW2
HW3
HW4

practice reasoning (and testing)

midterm

HW5
HW6
HW7
HW8
HW9

build an app (also practice adv Java tools)

final

Tests

- **Midterm exam** will be in class (50 min)
  - Friday, May 6th, just after HW5 is due
  - focus on reasoning and testing

- **Final exam** during finals week (110 min)
  - Tuesday, June 7th
  - time and location
    - Section B: KNE 220 at 2:30–4:20pm
    - Section A: KNE 220 at 4:30–6:20pm (**unusual**)
  - focus on reasoning and testing

Grading

- Approximate weighting (subject to change):

60%	Homework
15%	Midterm Exam
25%	Final Exam

- Very difficult to **fail** this class
  - likely need to not submit multiple assignments

- But **scores** may be lower than in other classes
  - these aren’t nearly as important as you think they are

Late Policy

- All students given free “late days”
  - Up to **4** times this quarter you can turn in a homework assignment **one** day late
  - Late days are 24-hour chunks
- Why have due dates?
  - keep you on schedule (real wor
  - finishing late means one less d
  - get feedback to you before nex

Do not use all of yours and then ask for a special extension when an emergency does arise

- Intended to handle special situations
  - plan to complete each assignment *on time*
- Any additional lateness requires special permission

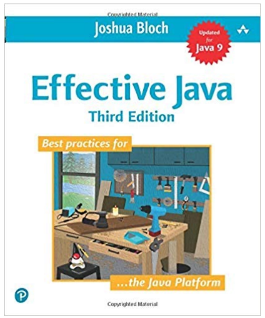
Academic Integrity

- “The code you submit must be your own”
  - no copying from other students, web pages, etc.
  - can talk to others but you must do the work yourself
- 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
  - worst result otherwise is expulsion
- Violations are unfair to other students and yourself

Books

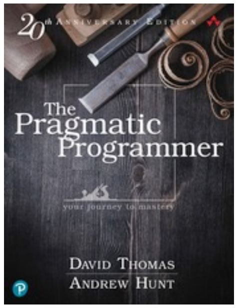
Required book

- *Effective Java* 3rd ed, Bloch (EJ)



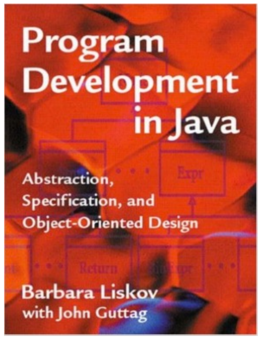
Optional book

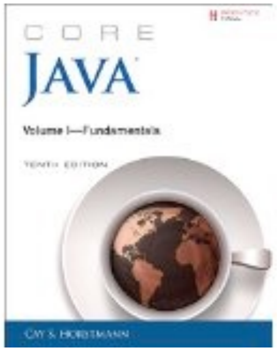
- *Pragmatic Programmer*, new 20th anniversary (2nd) edition, Hunt & Thomas (PP)



Other 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

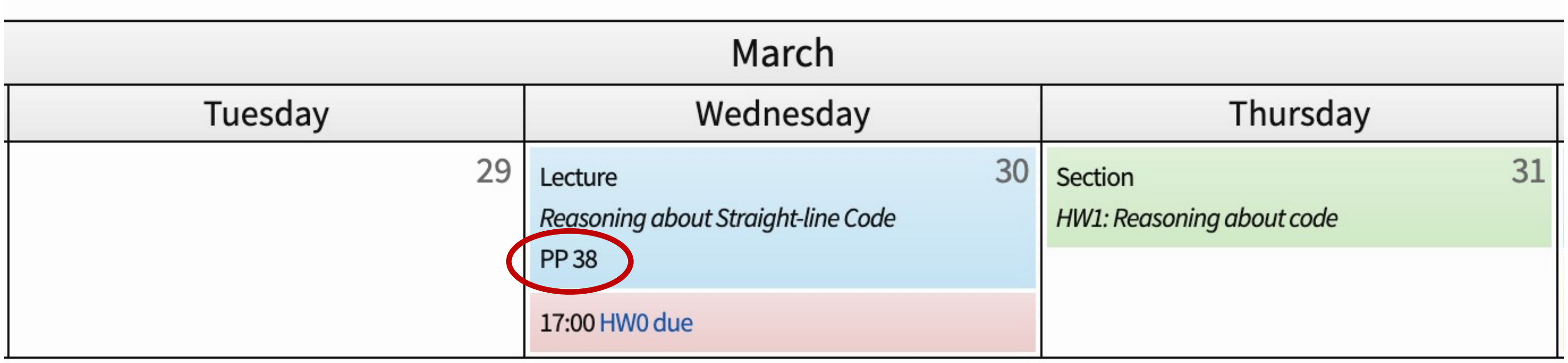

Program Development in Java
Abstraction, Specification, and Object-Oriented Design
Barbara Liskov
with John Guttag



Readings

- Calendar will include book sections for you to read
  - EJ = required, PP = optional

- Readings are fair game for exams
  - want to make sure you do it



March		
Tuesday	Wednesday	Thursday
29	30 Lecture *Reasoning about Straight-line Code* PP 38 17:00 HW0 due	31 Section *HW1: Reasoning about code*

Important Websites

- **Course website** (cs.uw.edu/331) for
  - calendar
  - assignments
  - section materials

- **Gradescope** for
  - submit assignments
  - request regrades (for obvious errors)

- **Canvas** for
  - recorded lectures
  - final scores for assignments (after regrades)
    - **ignore** percentages, totals, etc.

Advice

CSE 331 can be challenging

- Experience tells us CSE 331 can be **hard**
  - not my intention to make it difficult!
- Big change to move
  - **from** programming by trial & error
    - technique that 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 early

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

HW0

An exercise before next class

- Do HW0 (90 minutes max) before lecture on Wednesday
  - practice interview question
  - **write** an algorithm to rearrange array elements as described
  - **argue** in concise, convincing English that it is correct
    - don’t just explain *what the code does!*
  - **do not run** your code! (pretend it’s on a whiteboard)
    - know that is correct *without* running it (a necessary skill)

- This is expected to be difficult (esp. the ”argue” part)
  - participation credit, not graded for correctness
  - do not spend more than 90 minutes on it
  - want you to see that it is tricky… *without the tools coming next*

Before next class...

1. Familiarize yourself with website:

   http://courses.cs.washington.edu/courses/cse331/22sp/

   – read the syllabus
   – read the academic integrity policy
   – find the homework list
   – find the link to Canvas

2. Do HW0 before lecture on Wednesday!
   – submit a PDF on Gradescope
   – limit this to at most 90 min