CSE 331
Software Design & Implementation

Kevin Zatloukal

Summer 2016

Lecture 1 – Introduction & Overview

(Based on slides by Mike Ernst, Dan Grossman, David Notkin, Hal Perkins, Zach Tatlock)

What is the goal of CSE 331?

In short: to help you become better programmers

Specifically, to teach you how to write code of

- higher **quality**
- increased **complexity**

We will discuss *tools* and *techniques* to help with these

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?

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

1. **Tools**
   - e.g., type checking compiler
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 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 defect OS could sink the *whole fleet*
- And more reasons we will see shortly…

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

In summary, we want our code to be:

1. correct
2. easy to change
3. easy to understand
4. modular

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% are errors are btw modules [Seaman, '08]
  - corner cases are more important with more users
- Comm. costs dominate schedules [Brooks, '75]

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

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, Eclipse, 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

Course staff

- Lecturer:
  - Kevin Zatloukal (kevinz@cs, zat@uw)

- TAs:
  - Justin Bare (jbare@cs)
  - Vincent Liew (vliew@cs)

- Office Hours:

Monday	Tuesday	Wednesday
2:30 – 3:30pm	2:30 – 3:30pm	2:30 – 3:30pm
CSE 218	CSE 006	CSE 006
Kevin	Vincent	Justin

Staying in touch

- Course email list: **cse331a_su16@u.washington.edu**
  - for class announcements
  - students and staff already subscribed
  - fairly low traffic
- Message Board
  - for class discussion (staff will monitor and participate)
  - help each other out and stay in touch outside of class
- Course staff: **cse331-staff@cs.washington.edu**
  - for things that don't make sense to post on message board

Prerequisites

Only prerequisite is Java knowledge

- we assume you have mastered CSE142 and CSE143

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` (interfaces)

Lecture and section

- Both are required

- All materials posted, but they are visual aids
  - arrive punctually and pay attention
  - if doing so doesn't save you time, one of us is messing up (!)

- Section will often be more tools- and homework-focused
  - especially next week: preparing for projects

- Will post other handouts related to class material on web site
  http://courses.cs.washington.edu/courses/cse331/16su/

Homework

- Homework assignments will
  1. give you more practice
  2. require you to apply the techniques learned in class
     - Pro Tip: think about which techniques are intended

- Four (**4**) late days for the quarter: save for **emergencies**
  - **max 2** per homework, save them for later
  - email staff if you need to use 2 (may have started grading)

- We will have 10 homework assignments
  - first 3 are on paper, then all coding
  - early assignments come faster in summer quarter…

Homework (cont.)

Not as bad as it looks on the calendar…

June				
Monday	Tuesday	Wednesday	Thursday	Friday
20 13:10–14:10 Lecture EEB 037 *Course overview & introduction* 14:30–15:30 Office hours (kevinz) Location: TBD	21 14:30–15:30 Office hours (vliew) CSE 006	22 13:00 HW0 due 13:10–14:10 Lecture EEB 037 *Reasoning about straight-line code* 14:30–15:30 Office hours (jbare) CSE 006	23 13:10–14:10 Section EEB 037 *Reasoning about code*	24 13:10–14:10 Lecture EEB 037 *Reasoning about loops* 17:00 HW1 due
27 13:10–14:10 Lecture EEB 037 *Writing loops* 14:30–15:30 Office hours (kevinz) Location: TBD 23:59 Quiz 1 due	28 14:30–15:30 Office hours (vliew) CSE 006	29 13:10–14:10 Lecture EEB 037 *Specifications (pt 1)* 14:30–15:30 Office hours (jbare) CSE 006 23:00 HW2 due	30 13:10–14:10 Section EEB 037 *Git & Java tools & HW3*	01 13:10–14:10 Lecture EEB 037 *Specifications (pt 2)* 17:00 HW3 due

Academic Integrity

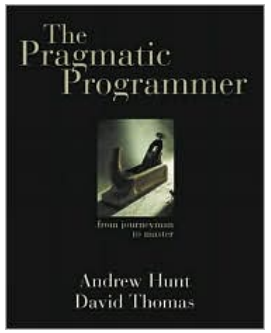
"The code you write must be your own."

- Read the course policy **carefully**
  - collaboration is encouraged, but…
  - do not share your HW code with others
- When in doubt, document your collaboration in your HW
  - at worst, you will lose a few points
- Cheating disrespects your colleagues and yourself

Books

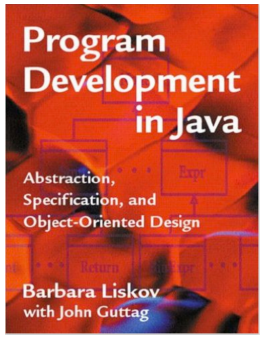
Required textbook

- *Effective Java* 2nd ed, Bloch (EJ)



Other useful books:

- *Pragmatic Programmer*, Hunt & Thomas (PP)
  - **recommended** (usually required)
- *Program Development in Java*, Liskov & Guttag
  - would be the textbook if not from 2001
- *Core Java* Vol I, Horstmann
  - good reference on language & libraries



Books? In the 21st century?

- Why not just use Google, Stack Overflow, Reddit, Quora, …?

- Web articles can
  - be out of date (without any indication this is so)
    - even 2014 is like 1960 in Internet years
  - rely on context that is not apparent on that page

- Books usually give better presentation of high level ideas
  - the purpose of a language feature or library
  - key reasons for its design

- Do use the **Java 8 APIs** (link on web site)

Readings & Quizzes

- We will have readings from first 2 (or 3) books
  - if not in EJ, then photocopies will be provided in class
  - these books are also on reserve at the library

- These are "real" books about software, approachable in 331
  - occasionally slight reach: accept the challenge

- Quizzes to make sure you don't skip the readings
  - short: 2-6 questions, usually multiple choice
  - take as many times as you want

Exams

- Midterm in class on Friday, July 22nd
- Final in class on Friday, August 19th
- Exams will be
  - focused on concepts learned in class
  - shorter than in normal quarters (1 hour each)

Grading

Approximate weighting (subject to change):

50%	Homework
5%	Homework readability review
5%	Reading quizzes
20%	Midterm exam
20%	Final exam

Readability review: make sure your code is **understandable**

- ungraded readability review on either HW5 or HW 6
- graded readability review on either HW 7 or HW 8 or HW 9

Acknowledgments

- Course designed/created/evolved/edited by others
  - Michael D. Ernst
  - Dan Grossman
  - David Notkin
  - Hal Perkins
  - Zach Tatlock (newcomer last quarter)
  - A couple dozen amazing TAs

- Hoping my own perspective offers benefits

- [Because you are unlikely to care, I won't carefully attribute authorship of course materials]

CSE 331 can be challenging

- Past experience tells us CSE 331 is **hard**
  - not my intention to make it difficult!
- Big change to move
  - from programming by brute-force, trial & error
  - to programming by careful design, reasoning, and testing
- 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
- Learning to program well is worth the effort
  - create solely with the power of your imagination
  - create software that positively affects the lives of many people

Questions?

Reasoning about code

A Problem

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

```
int indexOfMaximum(int[] arr, int n) {
    ...
}
```

Take a minute to think about how you'd write this…

A Solution?

Is this solution **correct**?

```
int indexOfMaximum(int[] arr, int n) {
    int maxValue = arr[0];
    int maxIndex = 0;
    for (int i = 1; i < n; i++) {
      if (arr[i] > maxValue) {
        maxValue = arr[i];
        maxIndex = i;
      }
    }
    return maxIndex;
}
```

A Solution?

Is this solution **correct**?

```
int indexOfMaximum(int[] arr, int n) {
    int maxValue = arr[0];
    int maxIndex = 0;
    for (int i = 1; i < n; i++) {
      if (arr[i] > maxValue) {
        maxValue = arr[i];
        maxIndex = i;
      }
    }
    return maxIndex;
}
```

Corner cases:
- What if there are ties?
- What if `n` is 0?

Error cases:
- What if `arr.length < n`?
- What if `arr` is null?

Morals

- You can all write the code!

- Takes work to show that the code is correct
- **Step 1**: what does it mean to be correct?
  - that is called the "specification" for the function
  - can't argue correctness if we don't know what is correct

- Specifications are hard to write
  - there can be many corner cases and error cases
  - do we even want to specify behavior for all of these?
    - depends on the situation
    - will discuss stronger vs weaker specs next time…

You have homework!

- Homework 0, due in dropbox by 1pm Wednesday
  - **write** an algorithm to rearrange array elements as described
  - **argue** in concise, convincing English that it is correct!
  - should run in O(n) time
    - challenge: can you do it in a single pass?
  - do not actually run your code!

- Start learning to reason about the code you write
  - this is the one homework that is *intentionally* difficult
  - stop after 2 hours (write up what you tried)
    - this HW grade is for participation not results
  - this will be easy in a week or so

To-Do List

Before the next class…

1. Familiarize yourself with website:

   http://courses.cs.washington.edu/courses/cse331/16su/

   – read the syllabus (esp. the advice section)
   – read the academic integrity policy
   – find the homework list

2. Do HW0 by 1 pm Wednesday!
   – limit this to 2 hours
   – submit a PDF into the dropbox