Introductions

- **John Jannotti (jj) — Networking, Sightpath, Foodler**
  - CIT 449 (Hours: After class until 5pm.)
- **HTAs**
  - Joe Engelman (joengelm) — Full Stack, Pinterest, Snapchat
  - Joe Romano (jromano1) — Javascript, e-commerce, Startups
  - Katie Han (kwhan) — Full Stack, Fitbit, Hack@Brown
TAs (part 1)

- Adam DeHovitz (adehovit) — Back-End Developer, ALTR infosec
- Ansel Vahle (avahle) — Full Stack, Yelp
- Ben Shteinfeld (bshteinf) — Databases, MongoDB
- Ben Wesner (bwesner) — Backend & Mobile, MITRE
- Brian Lu (bl48) — Full Stack, 2D games, Twitter
- Erica Oh (edoh) — Front-end development, React
- Greg Carlin (gcarlin) — Full Stack, Databases
- Jonathan Powell (jjpowell) — Backend, BlackRock
- Lindsey Baab (lbaab) — Full Stack, Web Hosting, BookBub
TAs (part 2)

- Nick DeMarco (ndemarco) — iOS, macOS, Adobe
- Nick Fahey (nfahey) — Back-End Developer, CashStar, Portland ME
- Nina Polshakova (npolshak) — Databases & Backend, Computational Biology
- Ping Hu (ping) — Back-End/Distributed Systems, Dropbox, Twitter
- Rafael Reyes (rr33) — Full Stack, Salesforce
- Rick Miyagi (rmiyagi) — Android & iOS, Google
- Sahil Prasad (sp85) — Backend & iOS, Cloudera
- Sara Solano (ssolano) — Full Stack, Square
- Sean Segal (ss97) — Full Stack, Facebook, RetailMeNot
You’ll remember your group project

Demo of Settlers of Catan
http://cs032catan.herokuapp.com/
(TAs have a simultaneous class, so they need to demo now. You’ll learn more about project constraints later today.)
Announcements

- Mostly, this whole lecture is announcements. But also...
- IPP Infosession about internships **today** CIT 368 5:30pm
- AppDynamics Ask Me Anything and Resume Review CIT 368/Atrium, Monday 6pm
- “Stars” gear-up session, next Tuesday, 6:30pm (BERT 130)
Software Engineering

- Software Engineering ≠ Programming
  - Programming is often a surprisingly small part.
- What else?
  - Determining what to build
    - Requirements (what tasks should the Software accomplish?)
    - Specifications (exact operating behavior)
  - Determining how to build it (Design/Architecture)
  - Testing (correctness and user experience)
  - Debugging (functional and performance)
  - Maintaining the program (new APIs, platforms, minor features)
- There are dependencies here, but not purely linear.

*Hopefully*, software lifetime is mostly maintenance/enhancement.
CSCI0320 focus

- We focus mainly on *Applications* programming.
  - CS33 focuses on *Systems* programming.

- We will spend most of our time on
  - Writing correct, understandable, and extensible code.
  - Building the right thing. *You* will make these decisions.
  - Integrating and connecting subsystems.

- But we’ve designed the class to give you a feel for maintaining a code base over time.
  - Projects work together.
  - Your pair project will marry your code to another student’s.
  - Labs will evolve the Boggle project.
Programming Languages

- Among “equivalence classes” language isn’t so important
  - Should be able to pick up new ones on your own.
  - We’ll teach universal concepts.
- But, how you use the language is important
  - By necessity, we’ll talk about language specific things.
  - Others need to understand your code.
  - Idioms vary from one language to another.
  - We’ll expect (and teach) you to write Java “properly.”
- Projects and lectures will all use Java (8) at their “core”.
- But the user interfaces will be “webby” - HTML, CSS, Javascript es6.
Why Java?

- Easy to teach concepts.
  - The interface mechanism is simple and powerful (often a better abstraction mechanism than inheritance).
  - Encapsulation / Data-hiding support is excellent.
  - Garbage Collection allows cleaner interfaces than C/C++
  - Static typing catches many mistakes at compile-time.

- Strong “ecoysystem”
  - IDEs and command-line tools widely available.
  - Extensive libraries available.
  - Common enough that you gain real-world experience.

- Fast enough for almost anything.
Why “Webby”?  

- User Interfaces are always a bit grungy.
- You might as well pick up reusable skills.
- HTML, CSS, and Javascript are “reasonable”
- EcmaScript 6, or “es6”, even more so.
- You’ll learn at least half of what matters to modern web apps.
- And a fair amount applies to mobile apps as well.
- We’ll be treating them as “display technology.”
- In projects, we won’t worry about
  - Users: Logins, Passwords, Multiple users
  - Security: Cookies, XSS attacks, malicious clients
  - Browser details: “Back button” behaviour, multiple windows, browser quirks.
Assignments

**Stars**
- Nearest-neighbors in 3D
- Emphasis on testing and design

**Autocorrect**
- Typo corrections, and type-ahead suggestions.
- You’ll be asked to think about users’ needs.
- String handling; Strategy pattern; Interactive Java/Web UI.

**Bacon**
- Finding shortest paths in a graph, Dijkstra’s algorithm.
- Coping with more data than memory (Using a SQL DB).
- Use autocorrect to improve your UI.

**A Pair Project to come**
- Bigger UI work, client/server model.
- You will be reusing, revisiting, and repurposing earlier projects.
- Pair partner can not be part of your term-project group.
Term Project

- Team project — Four person teams.
- Project of your choice
  - You own it, you live it.
  - Some of you won’t stop at the end of the course.
  - You will learn the most from this, we can’t guide you step-by-step.
- Concepts
  - Pull together all your skills
  - Software engineering, project management; software design
- Handout on-line now
  - Ideas not due for a while. Pay attention in daily life.
  - We encourage “heterogeneous” teams (wrt Intro Sequence)
  - No web frameworks beyond Spark.
  - No (native) mobile apps.
Project Ideas

- League / tournament scheduler (round-robin, double elimination, matched strength)
- Anti-clickbait (or fake news) as a service and a browser plugin to use it.
- Analyze, find anomalies in large document dumps.
- Wait-in-line app (need to make it interesting — texting, history)
- Strong Go, Chess, Scrabble, Crossword, Poker players
- Real-time strategy (Ikariam, Call of War)
- Desktop Tower Defense (Mix up a classic)
- Teleconferencing, “burner numbers”, other voice app (w/ Twilio)
Point Allocations

- Labs: 10%
- Clickers: 5%
- Projects: 55%
  - Stars: 10%
  - Autocorrect: 15%
  - Bacon: 15%
  - Pair: 15%
- Term Project: 30%
  - Requirements
  - Specifications
  - Design
  - 4-way Demo
  - Adversary TA meeting
  - Final Demo
  - Final Mentor TA meeting
Collaboration

- We are loosening some restrictions this year.
- You may discuss projects orally with other (current) 32 students.
- You may not look at each others’ code, or collaborate over a medium that allows direct sharing of others’s words or code (email, IM).
- You may take your own written notes. No photos, recordings.
- Take a break before coding after oral collaboration.
- This is about helping you learn, please don’t violate the spirit of these rules (word for word written notes, for example).
- We do use MOSS. Please don’t test it.
Meetings

- Lectures (Salomon 001)
  - Emphasis on design, we’ll walk through many case studies.
  - We time lectures to help with projects, honest.
  - Tuesday & Thursday 1pm-2:20pm

- Project “gear-up” sessions.

- Labs
  - Hands-on training you’ll need for assignments.
  - Significant to your grade.

- Communication via the web-site, Piazza
  - Signup: piazza.com/brown/spring2017/cs32
  - Calendar and lectures slides will be available.
  - You are responsible for everything posted.
This course just keeps going. . .

There is very little slack in the schedule.

- Start (even if only to read) assignments when they come out.
- Labs: Due in lab hours by due date or *nothing*.
- Assignments can be turned in 1 day late. (20% off)
  - No credit at all after 24 hours, and yet. . .
  - Every assignment must be turned in and working to pass!
  - If you’re not truly done 23.9 hours after the deadline, turn *something* in.
  - Getting a zero on a project costs about one whole letter grade.
  - You still need a passing grade. “All working by semester end” is a necessary, but not sufficient, condition for credit.
Much of cs32 is learn-by-doing.

but learn-by-doing works best if you “do” correctly.

We cover relevant ideas/designs/patterns “just in time.”

I’ll tell “War stories” — Avoid my mistakes.

We’ll do code walks/reviews of projects (our code and yours).

I’ll be using clickers for the first time. I hope that’s engaging, not a time-sink.
Clicker Test

Which into sequence did you take?

A) 15/16

B) 17/18

C) 19

D) Transfer, or similar.

ithelp.brown.edu/kb/articles/358-i-clicker-for-students-obtain-and-register-your-i-clicker
What is Software Engineering?

- Building software that works
  - We know how to build other things.
  - Engineering is the discipline that teaches us the methodologies that work for building complex objects.

- Apply engineering techniques to software.
  - Know what methodologies work.
  - Understand why and how.
  - Apply them appropriately and repeatably.
Components of Software Engineering

- Knowledge
  - Fundamentals (theory — algorithms, data structures)
  - Applied sciences (practice — debuggers, profilers, version control)
  - Empirical knowledge (applications — what’s out there?)

- Skills
  - Inventiveness.
  - Good judgment and foresight.
  - Attention to detail.
  - Clear communication.
  - Objectivity.
Design as Constraints

- Design involves finding a solution that satisfies a set of constraints
  - Users’ needs and wants
  - Target platform or environment
  - Financial or temporal resources
  - Programmers’ abilities
  - Government regulations

- Set of constraints is essentially infinite
  - Find a practical subset
  - Set priorities and ignore irrelevant ones
  - Find solutions that optimize the remainder

- Surprising Final Project Constraints
  - No network effects! (Not enough time.)
  - Limited access to interesting data.
  - You need to demonstrate good design, not a neat (or profitable) idea.
  - Don’t just build a C.R.U.D app!
Software Is Different

- Set of constraints is continually changing
  - In significant and meaningful ways
  - In unknown and unknowable ways
  - Priorities are changing
- People expect software to adapt
  - More so than anything else
- Software often fails to meet expectations...
  - yet we intend to build it anyway
- So how do we go about building software?
  - This is what software engineering is about.