Individual Assignment & Automation Tools
Individual Assignment

• A chance to experience some aspects of test driven development
  ○ Reading specifications in the form of unit tests
  ○ Writing code against clearly defined specifications
  ○ Different than real TDD - You get all of the tests in advance

• Get used to using GitHub (pull-requests, issues, etc.)
Individual Assignment

- Each student will
  - Get a read-only, private repo (we call this repo the *handout repo*)
  - Fork their handout repo
  - Clone the fork to their local machine, and work on the assignment
    - Write Java code to pass JUnit tests
    - Commit regularly and (optionally) use GitHub issues
  - Push their commits and submit the solution as a PR.

- After the deadline, we will merge your PR.
Individual Assignment

- Start early

- Double check your PR
  - Make sure Travis CI passes all checks
  - Make sure you did not change the given interfaces or testing code (any file under `src/test`)
  - Make sure your application code does NOT depend on the testing code

- Try to give us an insight into your work:
  - Make small commits, with concise and meaningful messages
  - Use issues, if needed

- And … Try to have fun
Individual Assignment

● Learn how to use your **IDE** (e.g. Eclipse) efficiently
  ○ Use autocomplete, *Ctrl + space*
  ○ Use auto-correct, *Ctrl + 1* (*Cmd + 1*, in OSX)
  ○ Use refactor tools (right click → refactor)
    Ex: Rename a variable/method using *Alt+Shift+R* (*Cmd+Option+R* in OSX)
  ○ Pay attention to compile warnings!
    They can save your code from breaking the auto-marker (e.g. leaving an unused JUnit import in the application code, will result in your code failing on the auto-marker)
  ○ Learn to use the debugger

● General tip: Try to pick up a new convenient shortcut every week.
Individual Assignment

One more (minor) goal for this assignment - Introduce a couple of useful automation tools (and get you to start thinking about automation as part of a software development process) ...
Automating Tests

- In Eclipse, you run unit tests by clicking through some menus, but what if we need to automate?

Ex:

  - Run unit tests every time someone submits (or updates) a pull-request. a.k.a Continuous Integration
  - Run long-running and/or resource-intensive tests during off hours
Automating Tests

- Can use **Maven** to automate JUnit test runs

- We follow some conventions
  - A specific **directory structure**
  - A **configuration file**, called pom.xml, that provides Maven with the information it needs

- Maven provides us with easy automation
  - *mvn test*
  - If we can run it in the shell (i.e. terminal) we can script (i.e. automate) it.
Automating Tests & Travis CI

In this assignment, whenever you submit (or update) a pull-request against the handout repo

1. **Travis CI** will clone your repo from GitHub
2. Use Maven to compile the code and run the tests
3. Reports the results back to GitHub (you will see them with the pull-request)

Note: It usually takes a few minutes (sometimes a bit longer) until you can see the test results.
Automating Tests & CI

● Continuous Integration is a useful tool.
  ○ Helps us avoid merging broken code into our repo
  ○ Extremely useful in open-source, where contributors may not trust each other’s code
  ○ Super convenient when need to test your code on diverse OS, CPUs, runtimes, etc.
  ○ Allows us to confidently merge code into production

● In the past, companies invested millions in server farms for CI.
  ○ Adobe Flash, Intel Android CI ran on 100’s of CPUs
  ○ now you can have it too.
Automating Tests & CI

- CI is not truly needed for your assignment, but we still wanted you to see it, because
  - It can still catch a few naive mistakes that can prevent your code from compiling. Ex: Forgot to add one of the files, before committing.
  - It can reveal other build-related bugs
  - You will most likely run into it at your first job
  - We think it’s cool, and the people at Travis-CI were generous to let us use their pro version for free.
More On Maven

- Maven can automate many tasks, not just running JUnit tests.
  - Compile the code
  - Generate Java Docs
  - Download dependencies from the Internet
  - And many more (Maven is extensible via plugins)

How is that relevant to CSC301?
Maven in CSC301

- Assignment code uses Maven to download a small, custom utility library
  - Eclipse takes care of it automatically

- The library is downloaded from JitPack’s servers
  - The pom.xml of your assignment specifies the URL of JitPack’s servers
  - The pom.xml of your assignment specifies the name (and version) of our utility library
  - Maven takes care of the rest
Maven in CSC301

● The first time someone asks JitPack for our library, the following happens:

  ○ JitPack clones the source from GitHub to one of their servers,
  ○ It then uses Maven to
    ■ Compile the code (into a Jar file)
    ■ Generate Jar files with Java Docs and source
  ○ And responds with the Jar file

● On future requests, JitPack can skip the build steps and simply serve the Jar file (that it cached on their servers).
Automation Tools

- Why are we telling you about these automation tools?
  - Chance to “show off” open-source software tools
  - Give you an idea of how modern systems are built.
    Many moving parts, even in a fairly modest operation, like the one we’re running for CSC301.
  - Heads up for PEY, internships, summer jobs