Individual Assignment & Automation Tools

Individual Assignment - What's the point?

- 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 - How it works

- 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 - Workflow Tips

- 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`)
  - Be weary of non-standard libraries or accidental imports
  - Make sure your implementation goes in the CORRECT folder
  - Make sure your application code does NOT depend on the testing code

  - ***Passing TravisCI does not mean you did not go against the above tips. Be diligent!***

Individual Assignment - Workflow Tips

- Try to give us an insight into your work:
  - Make small commits, with concise and meaningful messages
    - Especially important if you choose to start late or request a remark due to a bad mistake
  - Use issues, if needed

- And ... Try to have fun

Individual Assignment - IDEs are useful

- 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 - Automation

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 - Maven

- 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 - Continuous Integration

- 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 - Continuous Integration

- 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.

Automating Tests - Tying it all together

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.

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

Team Project - What is it?

- A project evaluating your ability to develop an Agile process and solve a problem as software engineers
- Identify a problem, the people it affects, and how you intend to solve it
- Define a proof-of-concept or MVP by the end of the semester in teams of 6-7 using whatever tech stack you choose
- Facilitated by the TAs during tutorials

Team Project - Process

- Document your own pseudo-Agile methodologies: Iterations, Reviews, Meetings
    - In reality - Sprints, Retrospectives, Planning/Refinement/Grooming and end-of-sprint Review
- Artifacts demonstrate your process - meeting minutes, project-management tool snapshots, etc
- Improve your process over the course of the semester to become a more efficient team

Team Project - Product

- Building an application that solves the problem
- Use any tools you choose
- Browser extension, mobile apps, scripts, IOT devices
- Artifacts - designs, mockups, prototypes, semi-functioning screens, etc
- Try to avoid games, sharing economy and CRUD applications

Team Project - Next steps and tips

- This tutorial we'll facilitate the creation of teams
  - Have a solid understanding of the tools you're comfortable with as well as your learning goals
  - Know your schedule
  - Pitch and listen to ideas
  - Create a team, pick a time slot and attend tutorials (i.e. actively participate)
- Follow your TA's advice and feedback