Documenting Your Code

Code Documentation

- Like testing, documentation has three properties:
  - It is hard
  - I hate doing it
  - I have probably not done it enough
- Developers find writing documentation tedious and difficult; it is easier to communicate with the computer (sometimes?) than it is humans.
- Documentation is NOT like pizza
  - No documentation is probably preferable to bad documentation

Documentation Purposes

- Refresher in how to use a package/library
- Introduction to using a package/library for the first time
- Warn developers of any little “gotchas” in the method
    - “What do you *mean* it deletes the object from the list after it returns it?”

Documentation Style

- Any development community (be it around a language, a professional community, or just a group working on a project) will have its own set of standards/documentation style
  - It is usually the responsibility of a developer to conform to those coding standards as writing code
  - Examples:
    - Google: `https://google.github.io/styleguide/javaguide.html`
    - Javascript: `https://github.com/standard/standard`
- Java has a standardized tool named Javadoc to enforce common documentation style and elements
  - This gives benefits; it allows user-facing documentation like the Java API docs to be automatically generated.

Javadoc Tags

- A Javadoc comment block starts with a `/**`, then contains tags that are preceded by a `@` symbol. Some of these tags include:
  - `@author`: indicates the author of a file
  - `@return`: describes the return value of a function
  - `@throws`: describes an exception a method may throw
  - `@param`: describes the purpose of a given parameter
  - And more (`https://binfalse.de/2015/10/05/javadoc-cheats-sheet/`)

Javadoc Example: method

http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#examples

Preconditions and Postconditions

- Formally, preconditions and postconditions specify a "contract" between the developer and the method
  - If the preconditions are followed, the postconditions are guaranteed to happen.
- Javadoc does not have built-in tags to specify pre and post (though you can define your own). However...

Postconditions

- Postconditions specify the results of running the method
    - The return value can be specified by the `@return` tag
    - What about side effects?
        - Postcondition: Element is added to the list
        - Postcondition: User is prompted for input
        - Postcondition: Pod doors opened
        - Postcondition: Skynet will take over the world
    - These might be appropriate in the description of the method
- It is important to only specify things your method *will* do
    - It is impossible to list everything your method does not do; there is an infinite number of things that it does not do
    - The only time that it is acceptable is if, for some reason, your method operates outside what would be expected.
        - Example: method to delete a file, but the file is not deleted at that time, but deferred until a later time...

Preconditions

- The things that must be true (about the inputs or global state) for the postconditions to happen.
- Beginning programmers tend to struggle with these
- Consider a method called `runOption(int choice)` that will run different things based on the value of `choice`. There are 3 different options.
- Consider that, for the original program, a method called `getUserMenuChoice` exists, that will get the user's choice from a menu. It is intended for the original use that the return value of `getUserMenuChoice` will be passed to `runOption`.
- Let's investigate what might make bad or good preconditions for these methods

Bad Preconditions: `runOption(int choice)`

- `getUserMenuChoice` has been called
  - What if we wanted to switch this to a GUI? Why can't I get the choice another way?
- User has inputted a number between 1 and 3, inclusive
  - Users are unreliable at best; never have a precondition rely on the user getting it right
- `choice` must be a valid int
  - Two problems with this one:
    1. No need to specify it is an int; we see that in the method signature
    2. What does valid mean? If it means it must be an integer, the compiler takes care of that for us. If it means it must be between 1 and 3 (inclusive), why are you making me guess? Valid is too vague.

Good Preconditions: `runOption(int choice)`

- `1 <= choice <= 3`

But how do we document this?

- If we are being smart coders, we would wrap this check up around an exception call (and throw an `InvalidArgumentException`).
- As such, we could represent this with an `@throws` Javadoc tag

There is a formalized way to determine the preconditions for a method/function/program based on the postconditions; if you are interested, check out the slides on dynamic semantics on the course website.

Design Lessons from Tests and Docs

- Testing and Documenting are tedious
  - I hate doing them, I have no shame in admitting
- They are necessary to help ensure that our software is of high quality, and is reusable
- We can design our code to make both testing and documentation easier, which will in turn make our code more maintainable and reusable

Modularity and Granularity

A method should do *one* thing (a class should represent one thing, etc).

- Unit testing becomes easier
  - Smaller methods means a smaller set of possible inputs and outputs
  - Identifying edge cases becomes simpler (instead of having two sets of edge cases I need to test all combinations of, I just have one set of edge cases)
- Documentation becomes simpler
  - Pres and posts are easier to identify
  - Easier to identify all of the behavior of a method
- Code is more reusable
  - It is easier to reuse less specialized code for given problems than more specialized code

Globals are Scary/Practice Good Encapsulation

Keep side effects to a minimum when possible (aka avoid global variables)

- Not always possible/desirable
  - Adding things to a list has side effects
  - Mutators (setters) have side effects)
- Side effects can make unit testing difficult
  - Side effects in private instance variables can make determining correct behavior difficult
- Documentation gets trickier
  - Javadoc has no built-in tag for defining side effects

This is less of a big deal in Java, where everything is encapsulated. However, unless the entire point of a method is to change the state of a class, avoid side effects when possible.

Practice Safe... Coding

Defining pres/posts can help us to determine appropriate exceptions and error handling.

- When dealing with a user, assume the user is either malicious or stupid
  - Assume they either cannot or will not follow directions
- Use preconditions to help decide what exceptions to throw

Testing and Docs

- Testing and documentation are activities not every developer likes, but they are important.
- In order to properly test and document our code, we must fully understand what it is doing.
    - Breaking down problems into small steps helps us to get a good grasp on what needs to happen to solve the problem.
- By thoroughly testing and documenting our code, we can help ensure that our code is robust, correct, and works as intended (and fails as intended). Doing this in addition to selecting intelligent algorithms and understanding the effects our solutions has on runtime can lead us to write high quality, performant code that is easy to maintain and reuse.