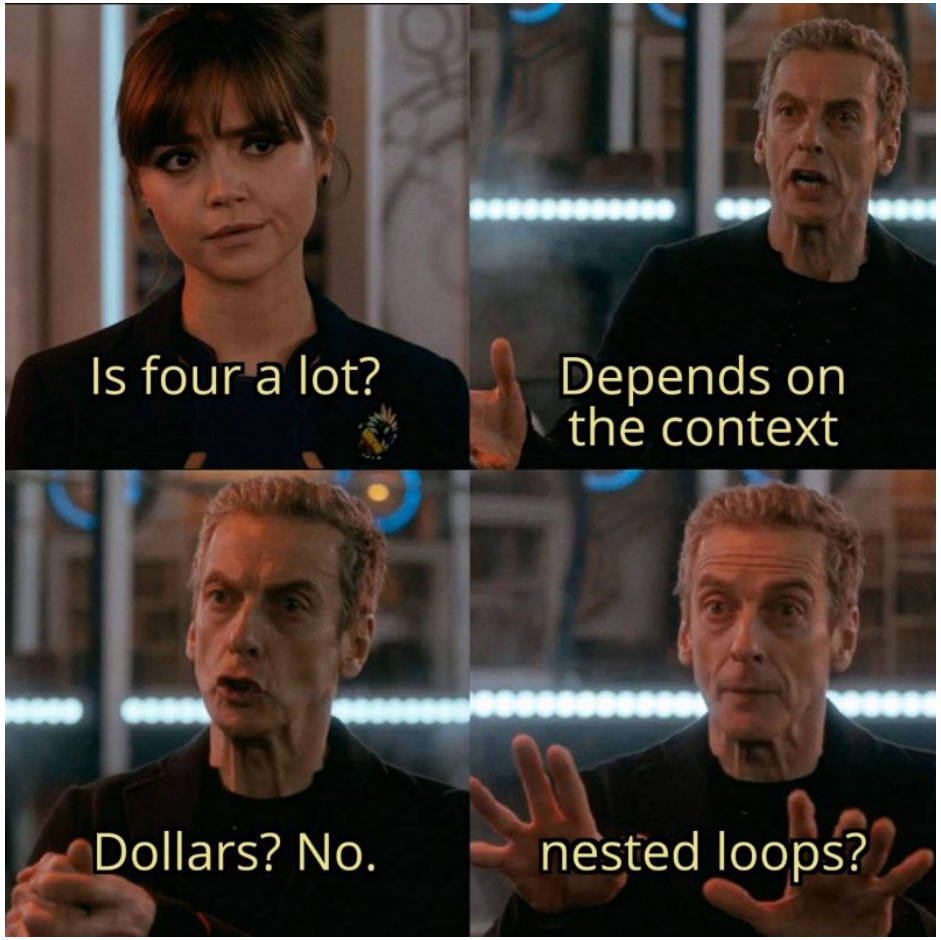

CS 170: Errors, Debugging, Testing, Comments, & Documentation

Announcements & Reminders

- Lab 4 on Friday (Lab 3 due this Friday)
- Problem Set 3 released (due next Wednesday)
- Checkpoint 3 Monday

Comments & Documentation

- Every method should have a comment describing what the method does and what any inputs or outputs are.
- Significant blocks of code should be commented to improve readability.
  - E.g. for loops/ while loops, if conditions, etc.

Errors: Syntax Errors

- Examples: missing semicolons, misspelled keywords, uppercase instead of lowercase letters, mismatched brackets, etc.
- These errors are caught by the compiler (javac).
- They are the easiest to fix and come with a specific error message.
- Also called compile-time errors.

Errors: Logical Errors

- The program compiles fine (javac gives no error), but when you run it you get the wrong result.
- These errors can be trickier to detect and correct.
- It's very useful to trace the program, i.e., manually simulate the execution of the program without having the machine execute it for you. While you trace the program, compare what the program is doing with what it is supposed to do.
- The program may crash, have an infinite loop, or given an incorrect result.
- Also called runtime-errors.

Finding Logical Errors

- It's very difficult to prove mathematically that a program works as expected
  - There are advanced formal techniques used in very sensitive applications (flight controllers, medical devices, …)
- Instead we try the program with a representative set of inputs and check the results: testing

Testing

- Try all the examples that are given to you with the problem
- Think of edge cases or extreme cases: zero, smallest or biggest possible values, boundaries of conditional statements, beginning and ending values in loops, etc.
- It is helpful to write your test cases even before writing the code
  - It helps you understand the problem better
- Test your code often while you program
  - Incrementally test and fix partial programs

Debugging

- Reproduce the problem
  - Find the input values that trigger the problem
  - What is the program supposed to do? (expected output)
  - What is the program doing instead? (real output)
- Examine the program behavior for that input
  - Code tracing
  - Visualize the internal state of the program with temporary print statements
- Fix only one thing at a time
  - Changing many things at the same time will get you out of control
- Retest all your test cases
  - Your fix might have broken something else
- Repeat the process until all bugs are fixed
- Clean up your code
  - Remove all temporary print statements, temporary variables, breaks, etc.

Peer Debugging Activity

- Team up with a classmate
- Write a correct program, or take a correct program from somewhere else
- Introduce some errors in your program, without revealing them to your teammate. Try to include errors of different types: syntax errors and logical errors. Logical errors are more difficult to catch
- Give it to your teammate for them to debug it. Ask them to verbalize their thought process so you can follow it
- Help them a little bit if they get stuck. Eventually all the mistakes should be found and fixed
- Switch roles, and repeat
- Optional: time each other and make it a challenge ;-)
- Reward yourselves for the good job!
- **For in-class activity credit: submit your buggy code to #lectures on Slack**