Code Design Fundamentals

Thanks to Nick Parlante for much of this handout

We're concentrating on OOP design ideas, but I wanted to take a moment to revisit the fundamental design ideas that underlay OOP.

Divide and Conquer

- aka "decomposition"
- Decomposition is one of those techniques that's so trivial that you sort of forget about it. Nonetheless, decomposing out a helper method actually works really well.
- For example, I wonder how many tetris `clearRows()` I saw would have been finished sooner if they decomposed out a `deleteOneRow(int row)` method. Compared to one huge loop, it's easier if a few logical parts of the problem are in their own method. Your eye can only take in some much logic in one chunk.
- It's not about cutting a long method into arbitrary parts -- it's about decomposing out a `deleteRow(int row)`, or `moveRow(int src, int dst)` -- operations that make some intuitive sense.

Bad Variable Names Can Kill You

- The tetris code has an angle of complexity that you can call you vars, `row/col` or `x/y` or `h/v`. I saw code that used `x` in some places, and `i,j,k` in many places. The code was a lot easier to see if it consistently used, say `x/y` everywhere ... so you could follow the pattern that the access always looked like `grid[x][y]`, instead of `grid[k][i]`, where you then look up to see what sort of thing `i` is. The variable name is there to help you -- use it!
- Don't write a comment explaining what the variable is, just give it a good name and no comment!
- Bad: `int i;  // note: is is the length`
- Good: `int len;`

Don't Duplicate Code -- "bottleneck" strategy

- Don't have multiple copies of code that does the same thing.
- aka "Don't Repeat Yourself", the DRY rule
- It's a bad sign if you find yourself copy/pasting
- This is not because we care that it takes up more space, it's about bugs
- With two copies of, say, the method that determines if an email address if valid, ... you are subject to subtle bugs where the 2 copies get out of sync.
- That's a classic bug in a system -- somebody comes along and edits one copy but not the other
- Have an official "bottleneck" function, e.g. `isValidEmail(String)`, and everyone who wants to test that condition goes through that one function.
- Exception: lots of duplication in test code is ok -- it it's wrong, you'll find out immediately. Test code is fairly repetitious by its nature. You can factor out a helper method in your test class to avoid code duplication -- see `ChunkListSuperTest`.

Don't Be So Darn Clever

- aka known as Keep It Simple, Stupid (KISS)
- It can be satisfying to reduce an algorithm down to some super-dense expression. It's satisfying in the same way as fitting in the perfect tetris piece.
- However, this can get out of hand .. using and abusing language features for the satisfaction of it, but actually producing terrible code.

- I think this sometimes shows up with language enthusiasts (cough, perl, cough), who find using and abusing the features of their favorite language to make the most dense code possible to a kind of badge of honor. It all comes from insecurity .. a need to show off a bit.
- The reality is that code should be readable. A bug is when your code doesn't do what you think it does -- a breakdown in readability of you looking at the code you just wrote!
- Be clever in your algorithms, not saving lines of text.
- This is also known as the "simplicity" movement -- avoid making things complex if possible. Appreciate something that is simple and works.

makeChocolate() Example

- The javabat.com site has this problem `makeChocolate(int small, int big, int goal)`:" We want make a package of goal kilos of chocolate. We have small bars (1 kilo each) and big bars (5 kilos each). Return the number of small bars to use, assuming we always use big bars before small bars. Return -1 if it can't be done."
- Here's a student solution:

```
public int makeChocolate(int small, int big, int goal) {
   return (big*5 + small >= goal) ? goal-5: (small == goal) ? 3: -1;
}
```

- This person certainly put high value on condensing the code as much as possible, showing off their understanding of ?:. In any case, the code's actually wrong! And worse still, it's crap, unreadable code. I suspect that the dominant thought in the author as they typed it was "I am soooo clever, I have no time for separate lines!". Even if the code were correct, what are the odds that the above code is debugged or maintained or re-used correctly by someone? I would prefer a solution with some actual if statements, maybe a couple comments, working through the cases of the problem.

Brian Kernighan Quote

- Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it? (Brian Kernighan)