Anti-Patterns, Code Smells and Design Traps
Grading Challenges

- It is not uncommon for students to be confused and/or frustrated by the scores they receive in code reviews.
- First, ask for clarification. Try to determine if you have violated a design principle, code smell, etc.
- It is OK to challenge.
  - No one gets in trouble.
  - You should not try to contest every point.
  - You will gain more points in the long run by following your moderator’s guidance.
More Rules!?!?!

Reassurance:

The “rules” are not set in stone. Some things may be really bad ideas so there will be a rule against it.

I once invented a feature by not “following the rules”. (there weren’t any)

- I exploited a design “flaw”.
- It resulted in boxing in future releases.

(also: Easter Eggs)
Boundaries Are Useful

SOLID Principles
DRY - Don’t Repeat Yourself
KISS - Keep It Short and Simple
KISS

KISS - A Design Principle

“It focuses on the idea that if we can’t understand a product, we can’t use it properly and that the widest possible audience must be able to understand it...”
Ways We Can Go Wrong

Common

- Code Smells
- AntiPatterns
Code Smells

“A code smell is a surface indication that usually corresponds to a deeper problem in the system. The term was first coined by Kent Beck while helping me with my Refactoring book” - Martin Fowler

https://refactoring.guru/refactoring/smells
Code Smells

Categories

- Bloaters
- Object-Orientation Abusers
- Change Preventers
- Dispensibles
- Couplers
Common CS126 Smells

For now, concentrate on the simplest ones:

- Bloaters
- Dispensibles
- Couplers (maybe)
Bloaters

Long Method

Large Class

Long Parameter List

Primitive Obsession*
Primitive Obsession

- Use of primitives instead of small objects for simple tasks (such as currency, ranges, special strings for phone numbers, etc.)
- Use of constants for coding information (such as a constant `USER_ADMIN_ROLE = 1` for referring to users with administrator rights.)
- Use of string constants as field names for use in data arrays.

  e.g. - myItem = arrayData[INSTRUCTOR];
Dispensibles

Comments*

Duplicate Code (DRY - Don’t Repeat Yourself)

Data Class*

Speculative Generality*
Comments!?!?!?!

“A method is filled with explanatory comments.”

We have had this discussion.

If you are using any of the comments other than the “Intent of code” or some form of documentation then you have a code smell.
Data Class

“These classes don’t contain any additional functionality and can’t independently operate on the data that they own.”

Do-nothing class
Speculative Generality

“Sometimes code is created “just in case” to support anticipated future features...”

Implementing every constructor or operation you can think of!
Couplers

Feature Envy*

Inappropriate Intimacy*
Feature Envy

“A method accesses the data of another object more than its own data.”
Inappropriate Intimacy

“One class uses the internal fields and methods of another class.”
Anti-patterns

At Friday’s AMA, you may have heard Patrick use the term “anti-pattern”.

“An AntiPattern... describes a commonly occurring solution to a problem that generates decidedly negative consequences.” - SourceMaking.com
Anti-patterns

“The AntiPattern may be the result of a manager or developer:

● not knowing any better,
● not having sufficient knowledge or experience in solving a particular type of problem, or
● having applied a perfectly good pattern in the wrong context.”

- SourceMaking.com
Common Anti Patterns

The Blob

Functional Decomposition

Golden Hammer
Common Anti Patterns - The Blob

“Procedural-style design leads to one object with a lion’s share of the responsibilities, while most other objects only hold data or execute simple processes.”
Common Anti Patterns - Functional Decomposition

This is the result of developers approaching design in a procedural style. Signature is classes that are designed around put functionality steps.

This is an intuitive way to solve problems. Even in a correct OO designed, you will find procedural code WITHIN A CLASS.
Common Anti Patterns - Functional Decomposition

Classes are designed to do something specific in a specific context in a specific order.

doStepOne();

doStepTwo();
Common Anti Patterns - Golden Hammer

“A Golden Hammer is a familiar technology or concept applied obsessively to many software problems.”

Especially when we are learning, we find a new idea (OO concept) and apply it everywhere that it seems to work or solve some part of our problem.

If all you have is a hammer then everything looks like a nail.
Paradoxically...

It seems that some of these rules conflict with others. Now what?

Meta-Programming

Design Patterns
How to Avoid Design Traps

SOLID

DRY

KISS
Remember

Class Types + Single Responsibility

- Entity
- Control
- Boundary
- (Value)
Questions to Ask Yourself

- Is there an obvious type of extension to this?
  - Try not to box yourself out
- What can change?
  - Try to minimize the number of classes affected with change.
- What is most likely to change and in what way?
Amazing Adventures UI

- Is there an obvious type of extension to this?
  - Input from a variety of sources is a common type of modification
  - Ability to load different maps
  - Multiple simultaneous sessions
- What can change/What is most likely to change?
  - Source/destination of input/output
  - More features/options
- How can I minimize the impact of change?
Case Study: TTTBoard - Data Class?

A data class refers to a class that contains only fields and crude methods for accessing them.
TTTBoard - Data Class?

Why I created TTTBoard:

1) Tied (m, n, k) with the board representation string
2) Allows both TicTacToe and ConnectFour to accept the same object type
3) Streamlined UI

Was it a good idea?

What functionality would make sense to add to it?

Would TTTBoard be the best place for that functionality?

What functionality would be duplicated for TicTacToe and ConnectFour?
Duplicated Functionality

Both TicTacToe and ConnectFour need properly-formatted data before any real work can be done.

They share some needs;

1. Length of string - depends on m, n
2. Noise filter - only 2 letters are acceptable, all else are noise
3. Case standardization - toUpper or toLower
What Could Change?

Let’s say my thought was that this would be able to act as an intermediary for variety of board games that work on a grid:

- Tic Tac Toe
- Connect Four
- Sudoku - {‘1’,...’9’}
- Chess?
- Checkers?
- Battleship?
  - Empty, ship, miss, hit
Duplicated Functionality

Both TicTacToe and ConnectFour need properly-formatted data before any real work can be done. The boards are very similar and the game rules are similar.

They share some needs;

1. Length of string - depends on m, n
2. Noise filter - only 2 letters are acceptable, all else are noise
3. Case standardization - toUpper or toLower

What’s needed to keep this general?

- List of acceptable letters?
- How to handle cases.
TTTBoard Could Handle Some Tasks...

It could:

1. force the board string to always be the right length or reject it if it's wrong.
2. filter the board string by a set of acceptable characters.
3. handle case standardization. Is that a good thing?
How Many Classes Are Affected?

1) If I keep adding boards with new needs,  
   a) what happens to this class?  
   b) Could it actually accommodate all boards?  
2) If I add a new type of game board, will I have to  
   change the classes for the specific use?  
3) Am I addressing a design specification?  
   a) Only in my head
Conclusions

Reason (excuse?)

- “It simplified my life AT THE TIME”.
- Is that a good enough reason?

If there was only a call for Tic Tac Toe, this would NOT be a good solution.

The parameter list needed to pass the information wasn’t long enough to violate the “magical number 7 (plus or minus 2).

**Conclusion:** paradoxically, it was lazy.