00 Design
Onerous “Rules”

Do not allow onerous rules to destroy your enthusiasm and creativity.

Rules That Count:
1) Does it work?
2) Do people use it? (requires rule #1 so...)

Rules are for strict “SOLID” engineering. Go rogue when rules may stand in the way of creativity.
A Tale of Three Writers

Three authors:

1) Associates (2-year), General Studies
2) BA, Creative Writing
3) PhD, French Literature

Which one(s) is/are successful writers?
Which one(s) are no longer writing?
Rules are AWESOME

Extremes of Transportation:

● Railroads
  ○ force strict adherence to the rails
  ○ built using known engineering principles.

● Off-roading
  ○ an exhilarating sport
  ○ creative problem solving.

*Engineering is a bit of both.
Design

• Design is a bit of an art
• Large/infinite design space, not enumerable
  o Requirements
  o Constraints
  o Trade-offs
  o Priorities

• You get better with
  o Practice and experience
  o Seeing good design
  o Hearing critiques of designs
If Design is an art then...?

We can follow some principles that will help us avoid “bad designs”!

What would constitute a “bad design”? 
“None of these things help me get my code working!”

- “style”
- Proper data types
- Naming
- Commenting
- Formatting
- “Rules, rules, and more rules! UGH!!!”

*they can DEFINITELY help with the right perspective*
Some Concrete Rules

1. Do NOT put all of your code in one file - each class should have its own file
2. Refactor your code
   a. Too many small methods
   b. Very large methods/classes
3. Work on small pieces - “baby steps up the mountain”
4. DO commit your code when you make significant changes
5. Don’t obsess over rules. Don’t let them paralyze you.
   a. Make it work
   b. Improve it
   c. Repeat
Design Tip 1

“protect your inner workings from “bad” input”

1) **Validate data** (Length of string)
2) **Standardize data** (simplify input)
   a) All “other” chars -> standard placeholder (space, ‘.’, etc.)
   b) All player moves -> standardized case (upper or lower)
   c) Fancy: neutral chars for X and O (if O goes first)
Design Tip 1

What is/are the benefits of Design Tip 1?
Design Tip 1

What is/are the benefits of Design Tip 1?

Streamlined logic (simplicity)
One Rule to Rule Them All

The goal of all software-design techniques is to break complicated problems into simple problems.

A.k.a - divide and conquer

*Prof Carl Evans
Virtues of Good Design

- Manages complexity
- Ease of maintenance
- Standard techniques
- Reusability
- Extensibility
Well-Defined Class*

● Has only one purpose that is clearly defined
● Does it well
  ○ Does everything it needs to do
  ○ Does nothing more than it needs
● Loosely coupled
● High cohesion

*this looks a lot like “single-responsibility principle”
Coupling and Cohesion

What are coupling and cohesion?

- **Coupling** is the degree of reliance an object has on the specific inner workings of other objects (less abstraction)
- **Cohesion** is the degree to which your elements (both methods and classes) “do only one thing and do it well”
Question

1. Which is a property of a well-defined class?
   a. Loose coupling
   b. Tight coupling
Complexity

Miller’s Law

The number of objects an average person can hold in working memory is about seven.

aka - “The Magical Number Seven, Plus or Minus Two”.
Too Many Rules!!!

If you are confused, confounded, overwhelmed, oppressed by onerous rules that you can’t understand or that crush your soul, start with these:

- Use **Abstract Data Types**
  - Treat even simple items as ADTs
  - Abstract the interactions between objects on
- Make Your Classes **Well-Defined Classes**
Question

1. Which of the following is NOT a property of a well-defined OO class:
   a. Can be easily change when we want to add new feature
   b. Less frequently changed during the development
   c. Easy to write
   d. Has many kinds of functionality so that it can be used for many different tasks (versatile)
D: Has many kinds of functionality so that it can be used for many different tasks (versatile)

*This is an example of poor cohesion. (It violates the Single Responsibility Principle.)
SOLID Principles*

- S - Single-responsibility Principle
- O - Open-closed Principle
- L - Liskov Substitution Principle
- I - Interface Segregation Principle
- D - Dependency Inversion Principle

*no, i don’t think you need to memorize this. Just remember that it exists for when you are ready for it.
Single Responsibility Principle

A class/object should:
● Do only one thing and do it well.
● Hide implementation details (focus on function rather than form)
● Manage complexity
Question

1. Which of the following grading rubrics of the assignment is NOT covered by the single responsibility principle:
   a. Each module should perform one distinct task
   b. No redundancy / storing multiple copies of the same data in the same class
   c. It should be easy to read through each method and follow its control flow
   d. Each test case should only serve one coherent purpose
Answer

C: it should be easy to read through each method and follow its control flow

While easy-to-follow code is a virtue, it is not part of the Single Responsibility Principle.
Open-Closed Principle

This means that a class should be extensible without modifying the class itself.

*think abstract classes and interfaces*
Liskov Substitution Principle

This means that every subclass or derived class should be substitutable for their base or parent class.

*we will look at this more closely another day
Interface Segregation Principle

- A client should never be forced to implement an interface that it doesn’t use
- Clients shouldn’t be forced to depend on methods they do not use

*We will look at this more closely another day*
Dependency Inversion Principle

- Entities must depend on abstractions, not on concretions
- High-level modules must not depend on low-level modules, instead they should depend on abstractions

String versus TicTacToeBoard
Too Many Rules!!!

If you are confused, confounded, overwhelmed, oppressed by onerous rules that you can’t understand or that crush your soul, start with these:

- Use **Abstract Data Types**
  - Treat even simple items as ADTs
  - Abstract the interactions between objects on
- Make Your Classes **Well-Defined Classes**
Abstraction vs Concretion

Which is the abstraction?

1. String
2. TicTacToeBoard
Defining Classes

- Define classes based around conceptual structures (abstractions)
  - Encapsulation / information-hiding
  - Make interfaces more informative (self-documenting)
  - Easier to reason about correctness
How Do I Know WHAT Classes to Design?
Types of Objects

To help you get started with a design, think in terms of these:

1. Entity Object
2. Control Object
3. Boundary Object
Bonus: Value Object
1. Entity Object
This object generally corresponds to some real-world entity in the problem space.

*tic tac toe board
2. Control Object

Control objects (sometimes also called Manager objects) are responsible for the coordination of other objects. These are objects that control and make use of other objects.

*game loop
3. Boundary Object

These are objects which sit at the boundary of your system. Any object which takes input from or produces output to another system — regardless if that system is a User, the internet or a database — can be classified as a boundary object.

(without these, your application or algorithm is operating but you see no evidence of it)
Value Object

**Bonus: Value Object**

Value objects represent a simple value in your domain. They are immutable and have no identity.
Question

1. A class that implements a web socket is a/an ______ object:
   a. Entity
   b. Control
   c. Boundary
   d. Transfer