Object Oriented Design Principles

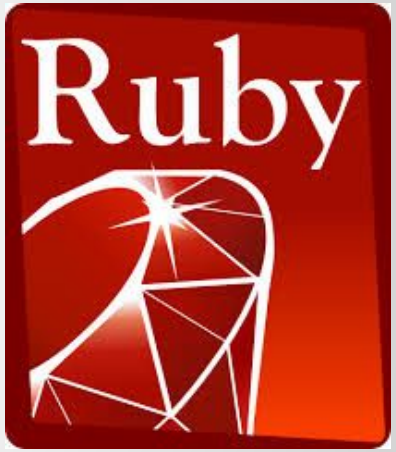
CS342 - Fall 2016

“You should never have to comb through all of the code because A changed, which required you to change B, which triggered a change in C, which rippled all the way down to Z.”

--Russ Olsen

Why Ruby?

- It is *extremely* Object Oriented
  - Everything is an object
- We'll be writing real Object Oriented code
  - Real object-oriented code should be easy to understand, modify and extend
- Ruby requires you to ***really*** understand OO



Why Object Oriented?

- Reuse, Reuse, Reuse
  - Reuse has been the cornerstone of the promotion of object-oriented programming.
- Reuse is supported and encouraged by object-oriented programming languages
  - Reuse does not come from code libraries
    - *Reuse is not about importing libraries into your code. Reuse is about writing code that can easily be 'plugged into' another program and reused without causing problems.*
    - *Not about using libraries, but how to create those libraries.*

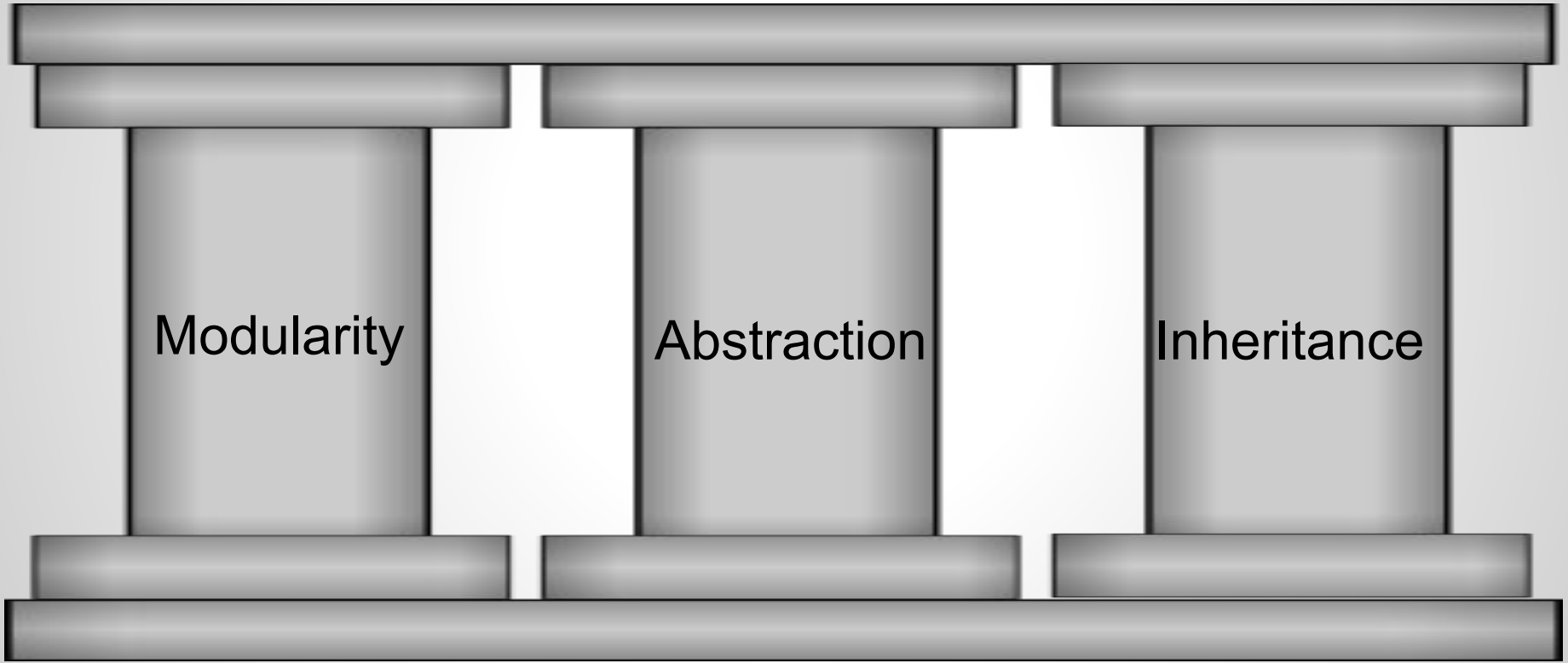
Some Object Oriented Concepts

- What is polymorphism?
  - a single interface that acts as a front end to different types.
- What does “Abstraction” mean?
  - hiding the complex details of implementation to create an interface
- What does an “Inheritance” mean?
  - When a Class is an extension or enhancement of another class, specifying implementation to maintain the same behavior and attributes as the other class.
- What is an “Interface”?
  - The specification that defines the behaviour and attributes of an object

Object Oriented Languages?

- What Languages are Object Oriented?
- Does C have interfaces?
  - Function interfaces (declarations)
- Can C, a procedural language, be Object Oriented?
  - Of course. OO is a conceptual way of programming. Any language can be Object Oriented.
- OO programming is about how you design your program, not what language you are using.

3 Pillars of Object Oriented Design



Design Principle of Reuse

- Our first design principle.
  - You will need to know these design principles for the test.
- **“Identify the aspects of your application that vary and separate them from what stays the same”**
  - Do something once, and never do it again
  - How can this be done in programming terms?
- Why do we write functions? Are they necessary?
  - You never need to write a function
  - Variables are what change, the function code is what stays the same

Classwork 1 - Part 1

No Solution

Design Principle of Information Hiding

- An interface hides implementation details. Why?
  - The underlying implementation can change, but this does not affect the users who rely only on the interface
    - *Hides design decisions that are likely to change*
- ***Design Principle:*** **Only allow access to software components necessary to the interface, not the implementation**
  - Language features keep aspects of the software from being accessible to higher level components
- Information Hiding is not Information Security

Classwork 1 - Part 2

Solution

- Common to both a satellite image and a map image are size and the ability to zoom in and out.

class Image

size : Dimension

---

zoomIn()

zoomOut()

Programming to an Interface

- ***Design Principle:*** **Program to an interface, not an implementation**
- Write a system to give everyone in the class the superpower of their choosing
  - Write separate classes for each superpower?
  - Join all classes with an interface:
    - *class SuperPowerInterface*

      @superpower = ‘ ’

      end

      - I am going to be writing in Ruby before I explain how to write in Ruby. It’s okay

Design Principle of Encapsulation

- ***Design Principle:*** **Group data and behavior into distinct classes that reveal only what is required to use the class**
- Enables validation/processing of values in a method before they are applied to the data of the object
  - preserves integrity of the data
- Encapsulation defines access levels: public, protected, private

Encapsulation vs Information Hiding

- Difference from Information Hiding is that data does NOT have to be hidden
- Encapsulation is about grouping data so that an object/function can thereafter be referred to by a single name and only reveals what it must to do its job
- Information Hiding is using an interface or definition to reveal as little as possible about its inner workings.
- **Information Hiding is an aspect of Encapsulation**

Classwork 1 - Part 3

A Possible Solution

Suppose we have a User class with username and password instance variables.

- The username should be encapsulated so changes can be verified as unique.
- The password should be hidden so that you only know if there is a match

Abstraction

- Abstraction means hiding the complexity of a system and providing a simple interface for it
  - Instead of writing the code for starting a car every time your want to go somewhere, you have car.start()
    - *For example:*
      - Do you need to know how your smartphone internally works to use it?
      - You only need to know the interface (on/off, touch screen, etc) to make it work.
- Encapsulation vs Abstraction:
  - Encapsulation focuses on the implementation (the system development)
  - Abstraction focuses on the interface (using the system)