Object Oriented Programming:

Why?

Today’s Goals:

- Introduce OOP philosophy
- Explain why OOP is useful
- Show OOP in action
- Explain OOP terminology

Review

- Classes are blueprints for data and methods that act on that data.
  - Think architectural designs for a house.
- Objects are instantiations of classes
  - Think the house they build from the designs
- Methods and Data are wrapped inside the class and their instantiations

What is an Class, Really?

Two layers:

- Interface
  - Visible
  - Gives us external behaviors to rely on.
    - Pre-/Post-conditions
- Implementation
  - Hidden
  - Produces the behavior specified by the interface

Whats the Difference?

- An interface is a list of methods, their behaviors and conditions
  - Describes
- An implementation is code, or pseudo-code, that performs specified behavior
  - Does

Whats the Difference?

Interface for String:

- int length()
  - Returns number of characters in the string
- char charAt(int index)
  - Pre-condition: index < length()
  - Returns the char at *index*
- int indexOf(char ch)
  - Returns the index of the character *ch* in the string, -1 if not found
- Etc.

What's the Difference?

Possible Implementation for String:

```
char[ ] charString;

public int length(){
   return charString.length;
}

public char charAt( int index ){
   if( index >= this.length() ) throw new Exception(index + " is out of bounds");
   return this.charString[ index ];
}

public int indexOf( char ch ){
   for( int i = 0; i < this.length(); ++i){
      if( this.charAt( i ) == ch ) return i;
     }
     return -1;
}
// etc.
```

Abstraction & Encapsulation

These layers create:

- Abstraction
  - Creating a higher level view of an idea
  - The interface
- Encapsulation
  - Binding data and implementation within one thing
  - The implementation

Enough with the Vocabulary

Cool, Abstraction, Encapsulation, so what?

- Abstraction allows us to *use* other classes without worrying *how* they work.
  - The interface tells us what they do.
- Encapsulation allows us to *write* classes that perform tasks:
  a) without having to divulge how it works
  b) making the task appear simpler to the outside

Which does...?

- Ultimately, when using an object we only care about *what* the object does, rather than *how* the object does it.
- This makes code:
  - Reusable
  - Easy to modify
  - Simple to use, despite its internal complexity

Abstraction and Interfaces

Even with abstraction, there may be implementation details to consider:

- Speed
- Memory consumption

But an interface allows for multiple implementations.

Additionally, an interface is now a type, can we can instantiate objects of that type

Interface Syntax in Java

Describes the methods that a class will implement:

```
public interface IntegerList {     // our interface
    public int remove(); // Defines that there is this method signature
    public int size();
    public void add( int element );
    // ...etc
}

public class IntegerArrayList implements IntegerList {
    public int remove(){   // implements this method
        // remove() method implementation code
    }
    // …etc
}
```

Summary

- Classes and Objects are powerful constructs that allow for simplification of data-structures, increasing their usability and rebuildability by abstraction and encapsulation.
- Interfaces allow us to be unconcerned with implementation details when using a class; and alternatively, not be concerned with the use of the class when performing the implementation.