Your Object Re-Orientation

CTEC 151

Starting Out

Procedural Program

- Generally, when we first learn to program, we learn to program procedurally
  - Programs are broken down into a series of steps
  - Variables hold the state of our program
  - Functions break steps down into nicely labelled blocks of code
  - Loops allow us to repeat sections of code
- Procedural programming can be used to make just about anything, but it has limitations

Limitations of Procedural Programming

- Code is rigidly structured
  - It’s difficult to add new features to old code
- Code can be long and difficult to understand
  - Functions and comments help with this to some extent
- Changes to code can break existing features

Programming Paradigms

- Imperative:
  - Procedural
  - Object Oriented
- Declarative:
  - Functional

Look, let’s not talk about Functional programming right now...

Python and OOP

- Python supports Procedural, Object Oriented and Functional styles of programming, though the Procedural and Object-Oriented styles are the most strongly supported
- Python is an interpreted language where functions and variables can be declared in the global scope
- Python allows the definition of Classes, which can create objects

Java and OOP

- Java is strictly an Object Oriented language, though it continues to add some Functional-style features as it updates
- Variables and “Methods” can only be defined inside of class definitions
  - This leads to some of the odd “boiler-plate” code that’s required to write a Java program
- Java is statically typed: you must declare what each variable holds

Why OOP?

What is OOP?

- A way to organize your code
- A way to think about your programs
- A coding style that's supported via several languages

Object Oriented Programming doesn't allow us to create anything differently than we could in a different paradigm, it's just a tool that can help make coding easier.

You can write a book with a pencil, a pen or a typewriter. All will give you the same result, but one is faster.

How is OOP Different

- Object-Oriented: we're dealing with Objects here
- In a normal procedural program, data is stored in variables and acted upon by functions
  - In a giant program, many different functions and parts of your code manipulate your variables
  - If something goes wrong, it's difficult to find the section of code that caused the issue
- In an object-oriented program, data is stored in Objects and acted upon by Methods
  - Objects manage their own data, so issues are kept to one area of code

Object Responsibilities

- Objects have a State (the variables they hold) and Behavior (the things they can do)
- In good Object-Oriented Programming, an Object only has 1 job
  - What that means is partially up to the programmer
  - If an Object is starting to handle multiple things, it may be time to split it into multiple objects
  - The goal is to keep all your code on this one concept in one place

The Advantages of Objects

- Objects help us break our code out into smaller chunks
- Objects help abstract tasks making our code easier to read
    - It's easier to understand myCards.dealHand() than the several lines of code that likely make it up
- Objects can provide extra flexibility through some concepts we haven't gotten to yet:
    - Inheritance
    - Polymorphism

How do we think through problems with OOP?

- Break a problem into different “objects”
  - To create a Bank application, we may have a BankAccount, a Customer, a Vault, etc.
  - To create a game we might have a Player, an Enemy, a Platform and so on
- Define what those objects know and what they can do
  - State and Behavior
- Then solve the problem as you might procedurally, but with the assistance of these objects that know how to do things

How OOP?

Classes

- The Blueprint for an Object
- When we write a class, we're not writing any specific object, we're writing what an object will generally look like

Instance Variables / Fields

- The data that an object holds
- In a class, these represent the data that an object will hold

Methods

- Just a fancy term for a function that belongs to a class
- Tells us what an object can do
- Has access to the instance variables of the Object
- Like normal functions, can have input parameters and an output value

The Constructor

- A special method with the same name as the Class
- Can take in parameters, but doesn't return anything
  - Or could be said to return an instance of the Class (i.e. an Object)
- Generally used to set up all the initial values of an object

Static as a Bad Word

- The point of OOP is to define your code as Objects that work together
- You can define variables or methods as Static:
  - Static methods “belong” to the Class, not the Object
  - Static variables “belong” to the Class, not the Object
- The overuse of Static is a code-smell to be aware of
  - Code that only uses Static values is no different from procedural code, which means you can’t get any of the benefits of OOP
  - Static variables and methods have their uses, but you should ask yourself why you’re using them before you do

Static Examples

Good Use Cases:

- Constant references that should be accessible globally:
  - Math.PI
- Helper methods that operate only on their parameters and return a value
  - Math.max(int a, int b)

Bad Use Cases:

- Getting access to a value without instantiating an object
  - The value will be stored on the class and the same for any object
- Trying to run a method without instantiating an object

Class Relationships

- More complex applications can include more classes, each which depend on each other in different ways
  - You may build a User Registration system that creates and stores Users in a database
  - Then, you might build a Shopping App that contains products and a cart, but also contains your User Registration system
  - This may turn into a larger entertainment system with a store (made from your Shopping App), but also a Game Player and a Movie Watcher
    - These may all use your User Registration system
- Classes may be written with other Classes in mind

Class Relationships Cont.

- Dependency
  - An object “uses” a different object
- Aggregation
  - An object contains references to other objects, but can exist on its own
    - A classroom has students, but can exist without them
- Composition
  - An object contains references to other objects, but cannot exist on its own
    - A human has a heart, but cannot exist without one

Just Good Practice

- Method Decomposition
  - In writing out everything a method needs to do, a method may become very large and hard to read
  - It's a best practice (though not necessary) to create private methods that handle smaller concrete parts of a problem
  - That way your larger method can become smaller and easier to read

Let's Write a Class