**SI 354 Theory Revision**

**Ch 9: Objects and Object-Oriented Programming**

An object is a logically related collection of data and functionality. It’s designed to reflect our natural understanding of the world. A car is an object that has data (make, model, number of doors, VIN, etc.) and functionality (accelerate, shift, open doors, turn on headlights, etc.).

Basic vocabulary of OOP.

* A **class** refers to a generic thing (a car). An instance (or object instance) refers to a specific thing (a specific car, such as “My Car”).
* A piece of functionality (accelerate) is called a **method**. (**instance methods**)
* A piece of functionality that is related to the class, but doesn’t refer to a specific instance, is called a **class method** (for example, “create new VIN” might be a class method: it doesn’t yet refer to a specific new car, and certainly we don’t expect a specific car to have the knowledge or ability to create a new, valid VIN). (**Static Methods**)
* When you refer to methods that are available on instances of a class, you are referring to **prototype methods**.
* When an **instance is first created**, its **constructor runs**. The **constructor initializes** the **object instance**.
* OOP also gives us a framework for **hierarchically categorizing classes**.

For example, we say that vehicle is a superclass of car, and that car is a subclass of vehicle. The vehicle class may have multiple subclasses: cars, boats, planes, motorcycles, bicycles, and so on. And subclasses may, in turn, have additional subclasses.

* **Inheritance**
  + When you create an instance of a class, it inherits whatever functionality is in the class’s prototype.
  + If a method isn’t found on an object’s prototype, it checks the prototype’s prototype. In this way, a prototype chain is established.
  + Java‐Script will walk up the prototype chain until it finds a prototype that satisfies the request.
  + If it can find no such prototype, it will finally error out.
  + Comes in handy when trying to create class hierarchies.
* **Polymorphism** is a word used to refer to how an**instance** is not treated as a member of just its own class, but also any super-classes.
* **Multiple Inheritance** is where one class can have two direct super-classes
* Languages that don’t support multiple inheritance often introduce the concept of an **interface**.
* **Mixin** is the primary mechanism for the problem of multiple inheritance in JavaScript

**Ch 11: Exceptions and Error Handling**

Exception handling is a mechanism that came about to deal with errors in a controlled fashion. It’s called exception handling (as opposed to error handling) because it’s meant to deal with exceptional circumstances—that is, not the errors you anticipate, but the ones you don’t.

Exception Handling with try and catch

* The idea is that you “try” things, and if there were any exceptions, they are “caught.”
* Because we caught the error, our program will not halt—we log the error and continue.

Throwing Errors

* You can also “throw” (or “raise”) errors yourself, which initiates the exception handling mechanism.

**Asynchronous vs Synchronous**

Synchronous program execution is somewhat similar to the above. Your program is executed line by line, one line at a time. Each time a function is called, program execution waits until that function returns before continuing to the next line of code.

This method of execution can have undesirable ramifications. Suppose a function is called to start a time consuming process. What if you want to stop the lengthy process? With synchronous execution, your program is “stuck,” waiting for the process to end, with no way out.

Asynchronous execution avoids this bottleneck. You are essentially saying, “I know this function call is going to take a great deal of time, but my program doesn’t want to wait around while it executes.”

**Ch 14: Asynchronous Programming**

* Callbacks
  + A callback is simply a function that you write that will be invoked at some point in the future.
  + Typically, you provide these callback functions to other functions, or set them as properties on objects

**Notes**

* Only functions that works on data that is already there
* Pull templates off GitHub
* Just use console.log()
* Asynchronous programming
  + Html files with script tags. Just write the functions in provided space
  + Requires you to connect
  + You have to connect to an API
  + Use “axious” to get result from API (LINE: “axious get”), no authentication.
  + Deal with API library using call backs and promises
  + **N.B. async/await (promises)**
* React
  + Classes and components are already set up but are empty
  + Fill in logic for components (Methods render methods)
  + Inside script tags in html files
  + Render methods, constructors, properties, updating states
  + Might be a fully functional game