CSE210 Notes

**Classes**

Classes are used in OOP. They don’t exist in Procedural languages

Procedural languages store data separately. No ability to store data within functions

OOP objects are classes with data and methods within them.

Functions become methods when in a class

Data and methods can be called and used in other classes

Instances can be created with objects. Ford = Car(), honda = Car() etc

Variables and methods can be created in instance with .method. ford.color = ‘red’

The pass keyword creates an empty class or method.

The def \_\_init\_\_(self): function acts LIKE a constructor, but it isn’t a constructor because it has automatic garbage collection

The \_\_init will run for every instance

Arguments can be passed into the class and then passed through in the instance

Class Car:

Def \_\_init\_\_(self, speed, color):

print(speed)

print(color)

The \_\_init function will print the passed through arguments in the functions but not create a variable to store the value. Self.attribute(variable) will create the variable/attribute

self.speed = speed

self.color = color

ford = Car(200, ‘blue’

Class attributes and instance attributes are separate entities and stored separately

**Self.variables in the init method are like global variables for that class**

Class Car():

Class\_var = 1

Def \_\_init\_\_(self, var\_):

Self.var\_ = var\_

Class\_var is the same for all instances, but var\_ is different according to instances.

**Object-Role Stereotypes**

**CRC**

* Modeling technique known as CRC cards **Class or Candidate/Responsibility/Collaborators**
* Use 3x5 card with simple sentence for purpose of class
* Divide lined side into two columns, left lists responsibility of class, right lists other class that must be interacted with to fulfill duties of the left side

**Responsibility-Driven Design**

* Breaking down to various actions and activities
* Consider which objects will be needed and how the responsibilities be distributed between objects

| **Stereotype** | **Description** |
| --- | --- |
| Information Holder | Knows things and provides information. May make calculations from the data that it holds. |
| Structurer | Knows the relationships between other objects. |
| Controller | Controls and directs the actions of other objects. Decides what other objects should do. |
| Coordinator | Reacts to events and relays the events to other objects. |
| Service Provider | Does a service for other objects upon request. |
| Interfacer | Objects that provide a means to communicate with other parts of the system, external systems or infrastructure, or end users. |

**Design around a behavior rather than designing from a data-centric viewpoint**

**Avoid the primitive obsession. Don’t be afraid to create small objects instead of wallowing in the mud of primitive variables**

**Structurer –** track, store, and maintain relationships between objects

* In many-to-many relationships amoung entities, will often need a structurer between everything
* Will be able to replace structurer if data grows in scope, rather than modifying other classes

**Service Provider** is usually passive – performs a task on behalf of another object

**Coordinator** is useful when you have a process that is event driven

**Laboratory for Teaching Object Orientation**

 We settled on three dimensions which identify the role of an object in a design: class name, responsibilities, and collaborators.

**Class names** – Important for design. Find the right words that are internally consistent with the design

**Responsibilites** identify problems to be solved. Solutions will exist in many versions and refinements

* Expressed in a handful of short verb phrases

**Collaborators –** No class or object is an island. Will send or be sent messages in the course of satisfying responsibilities