**October 22, 2021**

**Object Orientation Theory**

**Overview**

**Goals**

* Learn formal terms for object-oriented design

**Intro**

**What is it?**

* **Object orientation** is a way organizing your code using classes and objects
* It emphasizes the importance of storing **data** and **methods** in the same place
* Encourages developers to think in terms of objects rather than individual variables, functions, etc.
* All object store data but only some allow you to change. Class stores data. Class has change functions that you can call.
* A function is tied to a class.
* A method is a fn that lives inside of a object

**What does object orientation give you?**

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

**Abstraction**

* Hide the details
* Don’t need to understand the code behind every little detail in order to use

the class

* Can **instantiate** objects using classes without knowing all about how

those classes work

* You can **call a method** on a class without knowing how that method works
* Reduces **cognitive load** for a code-base

**let** order1 = DomesticOrder()

order1.addItem(12254)

order1.submit()

* The underlying logic and how order work is **hidden**
* We don’t need to know what’s happening behind the scenes in order to use the class

**Encapsulation**

* Store everything about dogs in **one place** - the class
* Dog logic, data, methods live together in the code
* Reduces bugs– when something about dogs needs to change, only need to edit one file
* Makes code more readable and understandable

**class** Order {

items = [];

submit(){

*// ...*

}

}

* The actual data for the order, ***items***, is in the same place as the ***submit*** logic

**Inheritance**

* Classes can inherit data/behavior from a **parent class**
* Useful for when one class is a special type of a broad category
* Ensures that you don’t copy/paste same code into similar classes

**class** **Order** {

// ...

}

**class** **DomesticOrder** extends Order {

// ...

}

* All ***Order*** functionality is shared
* ***DomesticOrder*** *inherits* ***Order***’s methods, but might have some unique ones of it’s own

**Polymorphism**

* Objects that are similar behave similarly
* Can have classes that inherit behaviors from a common parent
  + For example, ***Animal*** is parent class, ***Dog*** and ***Cat*** are child classes
* Polymorphism means that both dogs and cats share certain data and behaviors
  + They share a common **interface**
* This gets more into **object inheritance**, which is an intermediate topic covered in specializations courses