# Notes

## Chapter 1

Info: Strategy pattern was the first learnt: Build classes in a way that certain behaviors can be interchangeable for a main object. Lets the algorithm vary independently from clients that use it.

* pg 52/681: I thought all the behaviors are in one specific class "FlyBehavior" - or is there a class for each behavior?
* pg 63/681: HAS-A arrow?
* pg 50/681: When to use an abstract class vs inference?
  + Use abstract class if you have commonalities and only need to override a certain method. if you need to start overwriting all methods it is probably best to use interface.
* When is encapsulation?
  + private vs public, only gives access to what you allow. the encapsulation is within that class. add setter/getting for things allowed to be changed. encapsulation is the private stuff you're keeping together/hidden
* pg 61/681: Why favor composition over inherit?
  + composition you can change at runtime

## Chapter 2

Info:

* pg 84/681: Why do we need a Subject Interface?
  + Interfaces help give specific implementations of methods based on class
* pg 97/681: How does Subject weatherData work when subject is an interface?
* Online: "One of its cons is that Observable isn't an interface but a class, that's why subclasses can't be used as observables." - isn't the subclass getting the Observable from the superclass?
  + a subclass can only inherit one superclass, hm still doesn't answer question

## Project: Chapter 1 & Chapter 2

* flying/quack -> number of times a duck does it
* push/pull or both
* is the weather good for flying?
* example of a subclass trying to use observable ... what is a subclass???

### Comments:

* I originally put getName in the Subject so that I can pass the Subject as reference and still have the ability to use getName, but then I changed it to put the getName in the Duck class then type casted Subject to Duck. Is this bad practice? Is the Subject interface supposed to only hold methods for Observable?
* Can an Observable be made up of two different Subjects? Ie. be notified or not notified by different subjects?
  + Is this how I should be implementing the Migrate and the Quacks?
  + Current problem: if one changes, the update displays both -> where/how should I fix this?

## Chapter 3

Info: Decorate Pattern

* Pg 137/681: Can’t you issue a discount with another wrapper?
* You need to have the object as a component in the decorator

## Project: Chapter 3

Do some type of project using Decorator Pattern

## Chapter 4

Info: Factory Pattern

* Pg 154/681: So, the purpose is just the move the ‘changing’ code elsewhere? Because the ‘new’ operator is still being performed within the createPizza method.
  + I don’t see how this is helping eliminate new instantiations
* Pg 163/681: This is the general format when using the factory Method?
* Can we say: put changes in subclasses?

## Mini Project

Use the Factory Method to use the decorator