**If you are talking about it could you put the word “the” in front of it = OBJECTS**

**VERBS = Behaviors of Objects**

* **Class (name)**
  + **Attributes (nouns/data)**
    - **Behaviors (verbs)**
* **Objects = Instance of a Class**
  + **Instantiation**

**//////////////// Objects ///////////////////////////////////////////////**

* Object-Orientation
  + Programming paradigm
    - Set of ideas supported by different languages
  + Bringing programming closer to thinking about the real world
* Object Oriented Design
  + Write out the programs design before any coding is started
  + Not meant to be perfect
* Waterfall programming
  + Old way of thinking about program design
    - Not responsive
    - Not used anymore
    - Can hinder the development process
* Continual programming
  + Best practice
  + Responsive
  + Continual analysis and design
  + Agile/Iterative Approach
    - Involve several incremental cycles
  + Don’t have to know everything upfront
  + Expect initial iterations to be inaccurate and incomplete
    - Will improve on them as it is being developed
  + Meant to be good enough
* What is an Object
  + A thing
    - Has attributes
    - Objects are separate from one another
    - Has own identity that is independent from other objects
    - Have its own behavior
* Attributes
  + Describe current state of an object
    - i.e. color, size, etc
* Nouns
  + Not just physical things
  + Can be
    - People
    - Places
    - Ideas
    - Concepts
  + If you can put the word “the” in front of it
* Verbs
  + Are Behaviors of Objects
    - i.e. explosions, saving, printing

///////// Class ///////////////////////////////////////////////////////////

* Classes
  + Classes and Objects go hand in hand
    - Whole point of OOP is use of classes
    - Use classes to create objects
  + Describes what an object will be
    - A blue print
    - Detailed description
    - Definition
  + Can define a class once and use thousands of objects based on the class
    - Class comes first
  + Name (aka type)
    - Describes what it is
* Describes two things
  + Attributes (aka properties, data)
    - Pieces of information that describe each object
  + Behavior (aka method, operation)
    - What can the object do?
    - **Methods are functions that belong to a class**
    - Blocks of code that can be
      * Called
      * Perform actions
      * Return values
* Frameworks and Libraries (.Net Framework)
  + Created to store thousands of classes to be used by developer
    - Benefits
      * Don’t have to write the same classes over and over again
* 4 Fundamental Ideas to keep in mind when creating classes **(APIE)**
  + **Abstraction** = is what is done when a Class is made
    - Focus on the essential qualities of something rather then one specific example
    - Automatically discard what is unimportant or irrelevant
      * i.e. table (didn’t say what kind of table: just table)
    - Have an idea or concept that is separate from any specific incident
    - Do it all the time in conversations
    - Focus on the essential qualities of the idea
    - i.e. “What should a bank account class look like for this application, under these circumstances at this time, focusing always on the essentials”
    - Abstraction supports the foundation of OOP
  + **Encapsulation = Enclose an Objects Attributes and Methods and Hide everything about the Object, except what is absolutely necessary to expose**
    - Very useful when creating other classes
    - Think space capsule, medication capsule, food container
    - Surround and keep the contents together
      * Taking Attributes and Behaviors and bundling them together in the same unit (class)
    - Protect the contents
      * Restrict access to the inner workings of that class or objects of that class
        + Called “Data Hiding” or “Information Hiding”
      * Object should not reveal anything about itself except what is absolutely necessary for the other parts of the application to work
        + “Black Boxing”
    - Not about being secretive
      * Its about reducing dependencies between different parts of the application
      * A change in one place wont cascade down and require multiple changes elsewhere
    - Rule is: **Hide as much as possible**
  + Polymorphism
  + Inheritance