**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
  + **Each object contains its own Data and Logic**
* 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
  + **Pre-defined classes not typically included in a class library**
    - **Names**
* 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** = Many Forms
    - Lets us work freely with objects that have been created from any classes
    - Lets us do the right thing at the right time
      * i.e. “+” sign
        + Concatenate Strings
        + Numerically Add Integers
      * Lets us automatically do the correct behavior even if what we’re working with could take one of many different forms
    - Inheriting when useful, but can override behavior when that’s useful
  + **Inheritance (Single inheritance)** = A class can inherit the properties and methods of a superclass
    - Superclass
      * Parent Class
    - Subclass
      * Child Class
      * Inherits everything super class has
        + Attributes
        + Behaviors
      * Can add its own behaviors and attributes
    - Make a change in the Superclass it changes in the Child Class too

/////////// Software Engineering Methods to Create Apps/////////////////////

* **Gather Requirements**
  + What does the app need to do?
  + What problem are you trying to solve?
    - Get Specific
    - Write it down
* **Describe the Application**
  + Build the narrative
    - Explain it in plain conversational language for how people use the app
    - Explain in small self contained pieces
  + Can use
    - Use cases
    - User stores
  + Look for the smallest set of stories that will make it a real application
* **Identify the Main Objects**
  + Starting point to identify actual classes
    - Use the stories and descriptions
    - Pick out the most important ideas and concepts
    - Discard what is irrelevant
  + Not everything will become a class but a lot of them will
* **Describe the Interactions = Create a Sequence Diagram**
  + Understand the abilities of the different objects and behaviors they have
  + When interact with other objects
    - What they do
    - What order the need to do it in
* **Create a Class Diagram = Visual representation of the classes you need**
  + Be really specific about object-oriented principles
    - APIE (Abstraction, Polymorphism, Inheritance, Encapsulation)

////////// Requirement Analysis ///////////////////////////////////////////

**Have minimum amount of requirements**

**Not dream features, Choose only What is REQUIRED!**

* Gather Requirements (Analysis)
  + What is the Application *required* to do?
    - What *MUST* it do
  + Core requirements
    - Functional Requirements
      * What does it do?
      * Capabilities of the application
      * Features
    - Non-Functional Requirements
      * Help
      * Documentation
      * Legal Requirements
        + Do you know the details?
      * Performance Requirements
        + Response time

How many people does the app need to support simultaneously

* + - * Support Requirements
        + If there is something wrong with the app at 2am is there someone to answer?
      * Security Requirements
  + Functional Requirements Examples = Create short statements
    - System **MUST** display the heart rate, temperature and blood pressure of a patient connected to the patient monitor
    - Application **MUST** allow user to search by customer’s last name, telephone number or order number
    - Program **MUST** allow receipts to be generated via email
  + Non-Functional Requirement Examples
    - System **MUST** respond to searches within 2 seconds
    - Help desk available by telephone. Mon-Fri 8am-6pm
    - Comply with all relevant HIPAA regulations
    - Be available 99.9% of time during business hours
* FURPS/FURPS+ = Checklist
  + Functional Requirements
    - Features of App
  + Usability Requirements
    - Help
    - Documentation
    - Tutorials
  + Reliability
    - Disaster Recovery
    - Acceptable failure rates
  + Performance
    - Availability
    - Capacity
    - Resources
    - Supportability
    - Could it be Internationalized
  + +
    - Design Requirements
      * IPhone app
      * Relational Database
    - Implementation Requirements
      * Language going to use?
      * Comply with certain standards or methodology?
    - Interface Requirements
      * Not the user interface
      * Any external system need to interface with
        + Need to specify this now
    - Physical Requirements
      * Physical constraint
        + Needs to run on a device with a camera
        + Must ship with 50 gigabytes of DVDs
* **UML = Unified Modeling Language**
  + Not a programming language
  + It is a graphical notation specifically for drawing diagrams of an object-oriented system such as classic class diagram
    - i.e. <http://agilemodeling.com/images/models/classDiagramInheritance.jpg>