OOP

* OOP is a specific style of programming centered around "objects" which contains data and code that acts upon that data.
* Deep dive into data types:
  + We already know some Java data types: int, double, Boolean, String, long
  + Java has two different kinds of data types: primitive types (int double Boolean long, etc), which always start with a lowercase letter, and
  + reference types, which we will learn are called classes (String). These always start with uppercase letters.
  + But the thing that unites all data types is: we always can answer two questions:
    - **What kind of values does this data type hold?**
    - **What kind of operations can we do on these values**?
  + Answer these for int:
    - What kinds of values: whole numbers, between -2^31 and 2^31 – 1.
    - What kinds of operations: plus, minus, times, etc.
  + For Boolean:
    - What kinds of values: only 2: true, false
    - operations: and, or, not.
  + String:
    - values: "sequences of characters between double quotes"
    - operations: +, .length(), look them up in java docs.
  + So the unifying idea again is we have the values (what the data looks like), and the operations (the things that we can do with the data).
* Motivation: As you start designing more and more complicated programs, what typically starts to happen is that the built-in data types to any programming language are become too simple to handle the situations that you're trying to model.
* Do example with class: (Ask if they want to do dogs or cats). They always want dogs.
* Suppose you are designing a program that would be used in a veterinarian's office. or in a place that boards dogs while their owners are out of town. What information do we need to store about each dog?  
  + Come up with attributes of a dog. name, age, weight, sex, etc.
  + Now imagine that every time you get a new dog at the vet or the boarding facility, you need to create a brand-new set of variables to store all of this information in your program. That's a lot of variables to manage.
  + Now imagine you wanted to write a function that calculated how much to feed a dog. The amount a dog should be fed probably depends upon various attributes of a dog, like its age, weight, maybe what medications its on, etc.
  + But your function has to take all of these variables as parameters to calculate how much food the dog needs. That's a lot of parameters!
  + So the first thing that OOP lets us do is ***design new data types that let us combine simple data types into more complex ones***.
    - for instance, you could make a dog data type that consists of an int for its age, a double for its weight, etc. (draw picture with primitives and a dog object).
  + But wait, there's more!
  + Remember how I said for each data type in a language, we should know what do the values of this data type look like, and what are the operations we can do with these values?
  + This also goes for this new Dog data type. We know the values that a dog can be, because we know the values of all the variables built into a dog, but what operations should we be able to do on a dog?
  + Maybe we should be able to calculate how much food the dog will eat per day. What else should a dog be able to do? Maybe in a simulation game like at a dog park? Eat sleep bark chase another dog.
  + So the **second major thing that OOP lets us do is we can design new data types that let us combine simple data types into complex ones, AND package those types up with a set of FUNCTIONS that act on the data**.
  + These new data types we design (that combine ***variables*** to represent the information about the data type [STATE], and ***functions*** that can act on the variables [BEHAVIOR]) are called **CLASSES**.

END DAY 1. Didn't get to any coding (b/c spent the first 20 mins on going over totalDistance problem).

For day 2:

* Review highlights of OOP:
* OOP lets us design out own data types to combine simple data types into complex ones.
* AND lets us package those data types with a set of functions that act on the data.
* This makes it very easy to re-use code in very flexible ways.
* Analogy – remember in 141 when you first learned about functions. And it probably took you a few days or weeks before you bought into them to see how they were really useful?
* This is the same thing in 2 ways:
  + One: Because just like functions make programs easier to write by making it easy to re-use sections of code, OOP does the same thing by letting you re-use these new data types (sections of variables+code together). So these new data types have STATE and BEHAVIOR.
  + Two: It might take a few days/weeks to really see the benefit.

* APIs: The nice thing about most classes, especially in a language like Java, is that you do not need to know everything about a class to be able to use it. This goes for classes that are part of the Java language or other ones you get from elsewhere or design yourself.   
  + Usually a class publishes what is called an API (application programming interface) that lists all the functions that are built into the class. Functions that are built into a class are called ***METHODS***.
  + Look at API for SimpleCanvas.
  + Look at the methods. Then look at the constructors. **Constructors** are special methods that are used for creating an **instance** of a class, which is called an **object**.
  + Once you have an instance of an object stored in a variable, you can call methods on that object with the syntax:  
      
     *object*.*methodName*(*param1*, *param2*, etc...)
  + **Important: Methods listed as being part of an object must be called using this syntax. You must call them with an object in front of the name of the function. They cannot be called using our "regular" function syntax methodname(param, param)...**

* SimpleCanvas things  
  + Demo 1
  + Demo 2. Emphasize the SYNTAX of calling instance methods. (object)DOT(method).
  + Draw memory diagram with 2 canvases, showing that they are separate from each other in memory.
  + Ask students to create concentric square design. They can use any colors they want. **REMIND THEM to call .show() at the end.**  
    - SimpleCanvas canvas = new SimpleCanvas(500, 500);  
      canvas.setPenColor(Color.*RED*);  
      canvas.drawFilledRectangle(100, 100, 300, 300);  
      canvas.setPenColor(Color.*BLUE*);  
      canvas.drawFilledRectangle(150, 150, 200, 200);  
      canvas.setPenColor(Color.*GREEN*);  
      canvas.drawFilledRectangle(200, 200, 100, 100);  
      canvas.show();
  + Show Color API (handout and webpage). Use to demonstrate squares.
  + Have them do the problems in CanvasFades. 🡨- barely got to this problem at end of day 2. Day 3 (lab day) spent 15 mins at beginning doing this problem.
  + go through canvas count blue
  + go through canvas brighten
  + have them write filters.