CS Lecture 4

- On Friday, you guys had the opportunity to see some Java in action
- Today, we'll be looking at something called "Object Oriented Programming", which I did not make explicit to you
    - OOP is a "paradigm". A paradigm in CS is a "way of programming", or a "way of thinking" in a programming language. There are many different languages, each with their central and sub paradigms. Examples include imperative, procedural, and reflective
    - What does this mean? OOP is concerned with representing data.
        - Everything is an object.
            - Objects have properties; qualitative and quantitative
                - I have an age, a list of favorite colors, etc.
            - Objects also know how to do things
                - These are called "methods", and they look like functions
                - I know how to goFrom(location one, location two)
        - Objects are instances of a class
            - We are all humans. I know my age, I know my favorite colors, I know how to do things, and you know how to do the same things too
                - We could call humans a class. We are all instances of the human class
            - (Checkpoint: understanding Java and representing the world)
            - A class defines the behavior for objects. So a human class might look like...
                - Remember in Java when we want a variable we give the type variable_name? We could say Human Chris = new Human();
            - Classes can build off each other
                - A property of a school class might be a list of students, which we would denote as Students[]
    - Excercizes
        - Write a class for a book
        - Class for a store item
        - Class for a store
    - Methods and Functions
        - You guys know what functions are. You have seen that methods are like functions.
            - Function/method header/signature
                - consists of the return type and arguments (sometimes name)
                - A computer can identify the version to use based on the signature
                - You'll see this used later in overloading
            - Difference
                - A function performs a task on its own
                - A method is a function of an object; something an object knows how to do
    - Encapsulation
        - Once you make a class, you can use it right away. As you saw, you can even use your own classes to build other classes. So your classes have

their data and methods, and there's this idea of "scope" or "information hiding" to be specific.

    - The main idea is, you don't want to touch stuff inside the class if you don't have to. Someone builds a computer, puts it in a box, you use the computer, you're happy
    - In programming, your class might have data fields you don't want someone to touch. For example, you don't want someone changing your clock to 70 seconds
        - Instead, you have a method that will set the time, and it checks to make sure that the time is valid
    - You may have "helper functions"
        - Remember each function deals with only one task? Some tasks might be complicated, and will call helper functions
        - wakeUp() might contain shower(), brushTeeth(), eat(), and so on.
        - These helper functions should be invisible; you don't want a human to go brushTeeth() in the middle of the day
- Extending classes
    - From basic biology, you know that there are those 7 things that go from Animal Kingdom to like Homo Sapien or something. A dog is not a human. But a dog is a mammal. We are mammals. A snake is not a mammal. But a snake is an animal. We are all animals. You get the idea. All animals might know how to eat(). All mammals might have something(). All humans might know how to doMath(), well not all so that's why we're going to have a subclass
    - Human class
        - Age
        - Name
        - Colors
        - Talk
    - Asian class
        - doMath()
- Polymorphism
    - A snake is an animal, so it's valid to do this: Animal a = new Snake()
    - If a class extends another class, it means that it can do everything the old class did, so you can use it like a plain old animal, but it can do extra stuff
    - Animal a = new Snake() can eat because animals know how to eat
        - the snake cannot shedSkin() in this case because animals do not know how to shed skin. Snake s = new Snake() can shed skin
    - You may have a list of animals, and they're all different types and they're running in a program and you want them to eat. And because all animals at least know how to eat, everything's fine
    - A computer is not totally dumb though; it does know what something is behind the scenes and can do something about it
        - Sometimes you want to change how something works. A GoodEmployee may do more when he work()s
        - This is called overriding a method. You say, the old way isn't accurate for this class I'm making. A GoodEmployee can still work(), but he work()s differently
        - Now, if you had a list of employees and you told them all to work, they would all work because employees can work, but each employee would work their own way

            - Employee a = new GoodEmployee()
            - Employee b = new BadEmployee()
            - a.work() might get 10 things done, and b.work might only get 5
        - We have OverzealousAsian which not only doesMath(), but talks about math. We will override his talk class so that when you want him to talk, he just does math
            - We will overload the method so that he can be a bit more useful to us