Objects

Thursday, May 19, 2016 1:21 PM

I. Why objects, instance variables and instance methods
   - Learning goals
      - Objects are a mechanism to encapsulate functionality and memory about ones self
         - We want a way to group these methods together, and only give certain objects access to certain methods
         - We also want a way to give the same type of entities the same abilities
         - What if I want certain attributes from being changed? No option to encapsulate state
   1. Review
      - What if there were no objects?
         1. Build out three different types of cars with hashes. A car should have a color, year, a top speed, and an engine.
         2. Write methods that a car can do. A car can change it's position.
      - Now imagine that we want to build methods for a racecar drivers
         - So create hashes for racecar drivers
         - And racecar drivers should have methods, that allow them to celebrate, and declare themselves as conquerors of the track
   2. Concepts
      - What is the problem with the application we built?
         - Main learning goal
            - We want a way to group these methods together, and only give certain objects access to certain methods
            - We also want a way to give the same type of entities the same abilities
            - What if I want certain attributes from being changed? No option to encapsulate state
      - So this is objects

1. Objects
    1. Classes as factories
        - Class keyword
        - Baby - this is a constant
        - Baby -> represents the entire class
        - Baby.class -> Class
    2. Make instances of a class
        - Make an object
        - When we make one, it is a specific baby, and is particular
        - The variable, points to the instance
        - And now my_car.methods(false) -> cannot do anything because in our blueprint we have not taught babies to do anything
        - Just note that this has an object id just like anything else in ruby…
        - We use variables to reference instances of classes
    3. Lifecycle of objects
        - So now create another object, special_car
            - So teach it that every time a car is born, it should say vroom
            - So then we need to hook into this moment in time
            - That when they are born, they have immediately a procedure that they need to eecute
            - And then what code should I place in here if I want a car to go vrooom
        - Def initialize
            - Puts 'vroom'
        - End
        - Local variable for volume
            - Def initialize
                - Puts 'vroom'
                - Loudness = rand(100)
            - End
        - This is scoped to an instance of the object
            - So dogs do not cry when its born
        - Note that when I reference an object again, initialize does not get called again
- Give them a couple minutes

- ◻ Make an object baby that cries when it is born

1. Instance Methods
    - ◻ We want to be able to call car.color=()
    - ◻ Need to write a setter method
    - ◻ But problem is that this is a property of the car
    - ◻ Want to be able to write a piece of data to the object

    - ◻ And now need the other method to retrieve that data about the object

1. Instance variables
    1. What is the color…
    - ◻ Problem -
        - Is that they are not in scope
        - So cannot read a local variable from another method. Instead need a different kind of variable.
        - How many names are we going to need - We need one variable for each object instance
        - And need way so that each object has its own loudness, but this attribute can be shared across the entire object
        - So then the scope is the instance…every individual copy is
        - a) Instance variable
        - ◆ Show that from inside another method we can get this information about the loudness
2. To access data, from inside an object,
    - ◻ Must have a method
3. Initialize with arguments
    - ◻ Note that initalize is the interface for the object
4. Concrete Practice
    - ◻ Set an attribute called manufacture date
        - ◆ It should be called on initialize
        - ◆ You should not be able to change it
    - ◻ Then write two methods
        - ◆ One called age
        - ◆ Another called is_old?
            - ◆ Returns true if age is over 5 years
        - ◆ End
        - ◆ Have another where you can ask for the color, and another where you can change the color

5. Labs
6. Conclusion
    1. What are the purpose of objects
    2. Explain how methods relate to data encapsulation in objects
    3. How are classes like a blueprint and a factory?

II. Class variables and Class Scope
1. Class variables
- Problem: We want to keep track of all of the cars that were made
- So who should we ask that question of…?
- So we store that on the factory, on the class
    - So just like we store state on an object, by using an instance variable in an instance method
    - We store state on an object with a class variable, so @@count
    - So show how we can set the count to be zero
    - Notice that the class variable is not on the instance itself but on all cars in general
- Student task:
    - Change our car class, such that it increases the count each time a car is created
2. How do we retrieve the data, need a class method
    - Def Car.count
    - End
3. Scope of class variables
    - Like instance variables, they are accessible throughout the class
4. Class methods
    - So how do we make a class method
        - Because self is just a way to refer to the class
        - So you can also do Baby.count
        - So this is how you make a class method, you explicitly set the receiver to be the class itself
        - So this reads a class variable, and its called a class reader - it is exposing the contents of @@all
    - So now I want to do
    - Perhaps start off with
        - Print all of the names

III. Self - tease this apart
    - Write Baby.all

,

1. Problem
    - □ So how do we store the car itself?
    - □ Take a minute, make some attempts, and try to indicate the problem we have
2. Self
    1. **So self is just the object you are currently inside of**
        - ◆ It is the object to the left of the dot.
        - ◆ To do this, we must keep track of the method call that we are inside of
    2. Also, you can look at the class in the method you are currently

- ▪ Baby.find_by_name("Avi")
    - □ So teaching the class, how to find babies by a property of theirs
    - □ So if trying to find all particular babies by name, must have access to all babies; and I get access to all babies with @@all
    - □ @@all - the data structure it is, is an array
        - ◆ And it responds to find
        - ◆ So an instance of a baby can never equal a string
    - □ (People were very confused by this)

        - ◆ We need to go through ever individual baby, and ask it for its name, and then compare it against the name passed through
        - ◆ Note that @@all is filled with instances of babies, not just their name, and their blood type but all of their qualities