IST488/688 Social Web Technologies, Fall 2010
Lab 3.
Instructor: Keisuke Inoue

Description:

In this lab, you will work on implementing concepts for a social web site using object-oriented programming. All the information you will need is covered in Cooper, Chapter 6. The following instructions will help you to accomplish the task, but will not tell you all the small steps. **Note: To maximize your learning experience, you are encouraged to work on it independently. You may ask questions to instructors and/or friends (but keep it quiet) and look up resources, but you are not allowed to copy-and-paste any resources other than you created.**

Scenario:

(We are going with a new scenario this time.) Knowing pizzas create a $30+ BILLION industry (A quick fact: average Americans eat 46 slices of pizza per year. Source: http://pizzaware.com/facts.htm), you decided to start a new delivery pizza shop, where customers place orders online, according to their liking. You are about to design and implement concepts for your pizza shop…

Lab Instructions:

1. Setup and basic Linux operations

1.1. First, login to your Linux system, launch a terminal application, and create a folder called "Lab3". If you don't remember how to do it, review Lab 1.

1.2. Launch your favorite editor, create a new file and save it as "<your_lastname>.rb" in the Lab3 folder you just created.

1.3. Insert the comment lines, in the top section, stating the file name, description, author, and date, as always.

2. Defining a very simple Pizza class.

Now, it is most obvious that the most important concepts for your system is pizza. By this time, you should all know how to define concepts with Ruby, namely, by defining classes. Below is a definition of a simplest class possible, named "Pizza".

```
#  Class Definition: Pizza
#
class Pizza
end
```

Go ahead and copy-and-paste the code above to your lab file. Now you can instantiate the Pizza class by calling new, e.g.:

```
#  Creating an instance of the Pizza class
#
p1 = Pizza.new
```

Copy-and-paste the code above. Remember that `p1` in the code above is a *variable*, a placeholder for the new Pizza *object*. You can name variables anyway you like, as long as you start with a lower case and use alphabets, numbers or under scores. (You can lookup the textbook for more information, but that will do for our purpose.) The `p1` above is called more specifically a *local variable*, because it is only interpreted in a

limited context (called *scope* or *variable scope*). For example, the variable p1 above cannot be used in the Pizza class definition. (You can try testing it, but it will be confusing, so let's not do it.)

3. Defining instance variables

The Pizza class we defined above is pretty useless, without any attributes or methods. So let's add some characteristics to the class, which represents the real world pizzas! Three fundamental attributes of pizzas are (Note: I am not a big pizza eater, so correct me if I'm wrong.)

- Size: Small, Medium, Large or X-Large
- Crust: Regular, Thin or Deep
- Toppings: Pepperoni, Sausage, Green peppers, Olives, Pineapple, Mushrooms, Onions, and many more.

Let's implement the three attributes, by using attr_accessor. You just need to insert one line to the class definition you previously copy-and-pasted. Now the class definition should look like this:

```
#  Class Definition: Pizza
#
class Pizza
      attr_accessor :size, :crust, :toppings
end
```

Now you can specify the attributes of p1, as bellow (**Don't copy-and-paste these!**):

```
p1.size = "Small"
pi.crust = "Regular"
p1.toppings = [ "Pepperoni", "Green peppers", "Mushrooms" ]
```

4. Defining constructors

A problem, though, is that it is pretty tedious to specify all the attributes, especially when you have multiple objects. You would want to make it so that you can specify all the attributes at the same time, when the object is created. This can be done by defining a method called *initialize*, as below:

```
#  Class Definition: Pizza
#
class Pizza
      attr_accessor :size, :crust, :toppings

      def initialize (size, crust, toppings)
            @size = size
            @crust = crust
            @toppings = toppings
      end
end
```

Remember that `size`, `toppings`, and `toppings` in the method above are called *parameters*. They are a kind of variable, which are passed to the method. The things that start with the '@' marks, @size, @crust, and @toppings, are called *instance variables*, attributes of each instance of the class. So what the initialize method is doing above is just setting the values of the instance variables.

Now you can create an instance by specifying the parameters. Change the previously typed line of "`p1 = Pizza.new`" to the following line:

```
p1 = Pizza.new("Large", "Regular", ["Pepperoni", "Mushrooms"])
```

Notice that the last parameter is using an array, because you often want multiple toppings.

5. Show method

(Here is the first lab instruction where you are typing in your own code.) Just like the introduction method we had for the User class in the previous labs, define a method "`show`" in the Pizza class, so that we can see if the Pizza class has been properly instantiated. For example p1.show should print out the following texts:

```
Size:     Large
Crust:    Thin
Toppings: Pepperroni Mushrooms
```

Printing out size and curst should be easy. If you are not sure how to print out the toppings, look up the previous lab, where you used an iteration with an array. Run the program and make sure it's working.

6. Creating more Pizzas

Now, let's practice making a pizza. Using the new initializer, create another pizza of your liking. Call them `p2`. Call the `show` method the object, once it is created.

7. Class Inheritance

For customers, creating pizzas according to their preference is important. But as a premier online pizza shop, we should provide some "specialty" pizzas with all the parameters predefined. How do we do that? We can use the ***inheritance*** here. Below is a definition of a new class, called, HonoluluHwaiannPizza.

```
#  Class Definition: HonoluluHawaiianPizza
#
#  One of the classics of "specialty" kinds… despite the name,
#  it was a German invention, according to Wikipedia.
#
class HonoluluHawaiianPizza < Pizza
    def initialize (size)
       @size = size
       @crust = "Regular"
       @toppings = ["Ham", "Bacon", "Pineapple", "Roasted Peppers"]
    end
end
```

This looks like pretty much any class definitions we have seen, except it says "`< Pizza`" after the class name. This means that the HonoluluHawaiianPizza class is **inheriting** all the attributes and methods from the `Pizza` class. This is a strong feature of Object Oriented Programming, because not only you can save a lot of typing, you can see conceptual relationship clearly. Take this example; Honolulu Hawaiian Pizza is a kind of Pizza, in a real world, right? In Ruby, HonoluluHawaiianPizza is implemented as a *subclass* of *the* Pizza class. (The Pizza class is a *superclass* of HonoluluHawaiianPizza.)

Notice that the initialize method of the HonoluluHawaiianPizza class has only one parameter, size. This is because crust and toppings are specified by the method. So you should be able to create an instance with specifying only the size, e.g.:

```
p3 = HonoluluHawaiianPizza.new("X-Large")
```

Now, go ahead and create our third pizza, p3 as the Honolulu Hawaiian Pizza, and call the `show` method, to make sure the parameters are all set.

8. It is your turn to define your unique pizza. Define a specialty pizza class, just like the HonoluluHawaiianPizza class. Be creative and make it UNIQUE! Create instance of the new pizza, and call the show method for the object. Below is an example output (Ok, a SushiPizza might not be appetizing, but you get the idea…):

```
Pizza:    HonoluluHawaiianPizza
Size:     Small
Crust:    Regular
Toppings: Ham Bacon Pineapple Roasted Peppers

Pizza:    TokyoSushiPizza
Size:     Small
Crust:    Regular
Toppings: Tuna Soy Sauce Wasabi Rice Vinegar
```

(By the way, I modified the show method to show the class name above. You can do so, too, by using `self.class.to_s`.)

9. Now, define a method in the Pizza class, `price`, that tells you how much the pizza will cost. The price should be based on the size and the number of toppings. (You can include other factors, if you would like.)

Let's say Small: $8, Medium: $10, Large: $12, and X-Large: $14, and each topping is $1.50.

You can use an if-expression or a hash to store the prices for the different sizes. Call the price method for all the pizzas that you have created, i.e., p1, p2 and p3 (and more, if you like).

10. Submission

Submit your ruby file through http://sanka.syr.edu/swt/ as for the previous labs.

11. Evaluation

This Lab will be graded by the following criteria:

Classes, objects, and initialize methods (2pt)	The code shows that the student followed the step 1 – 4 correctly.
Show Method (2pt)	The code includes a correct implementation of the show method specified in the step 5.
Creating a unique Pizza (2pt)	The code includes a creation of unique pizza as specified in the step 6.
Defining a specialty pizza using Inheritance (2pt)	The code shows that the student followed the step 7 correctly.
Defining a method with if-expressions (or hash) (2pt)	The code defines the price method correctly, as specified in the step 7.