**CS210X 2016 B-Term -- Project 0 -- Facebuk**

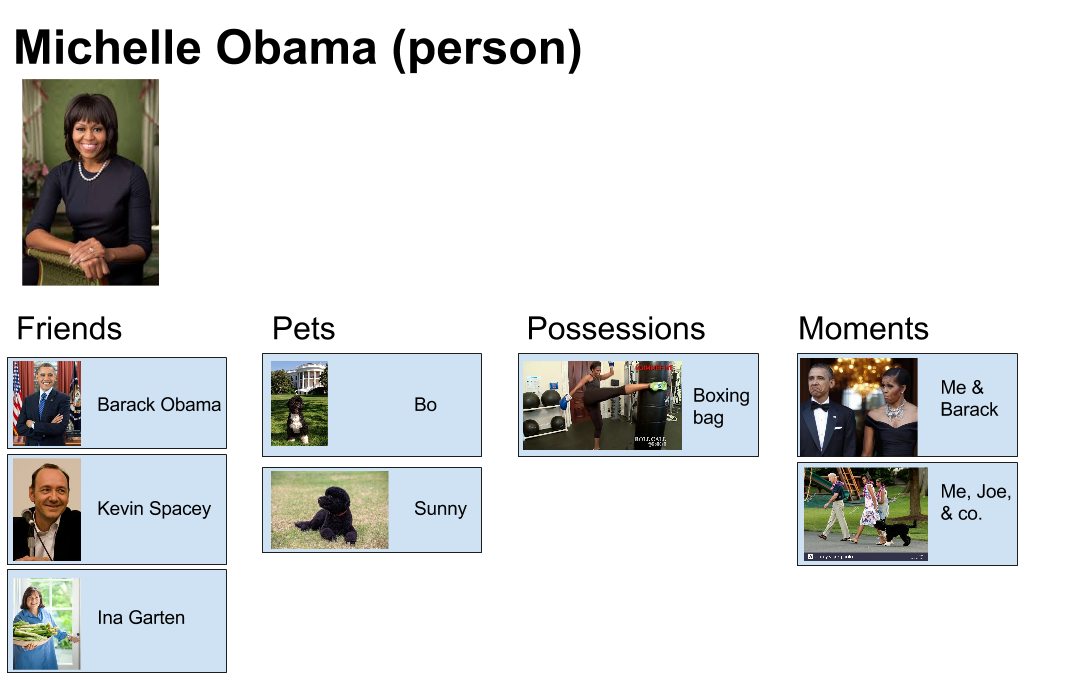
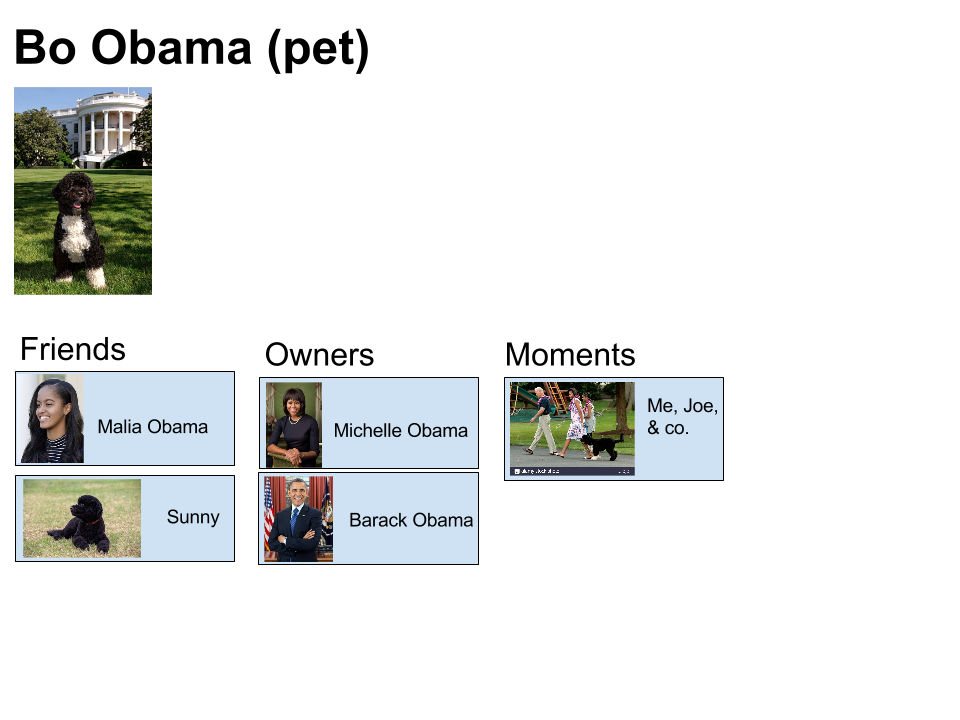
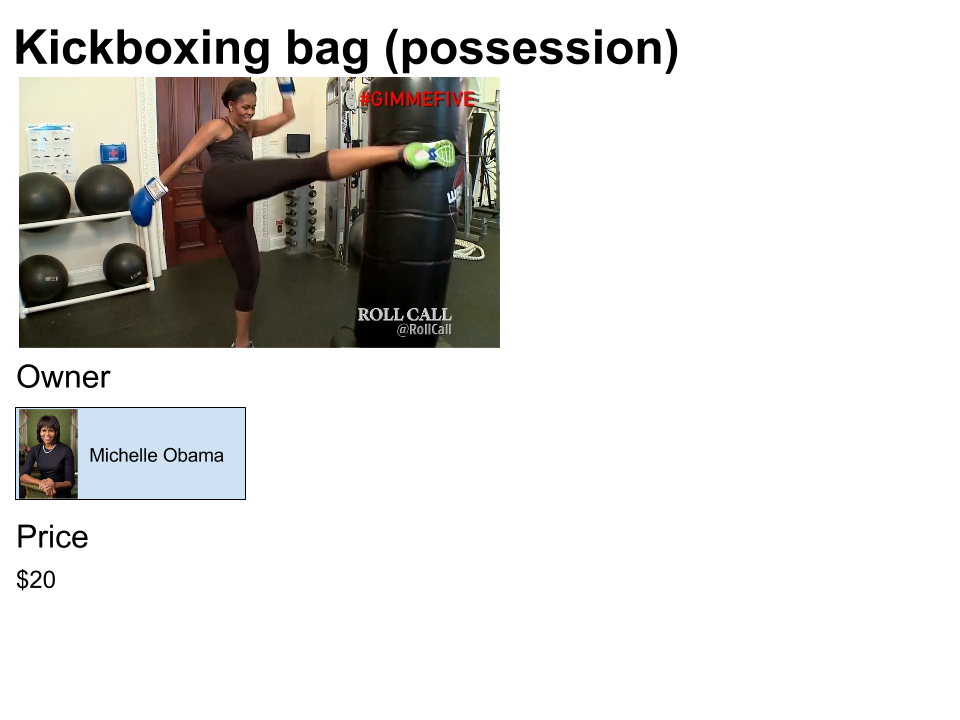
Prof. Jacob Whitehill

**Introduction**

The goal of this assignment is to practice thinking about designing software using object-oriented programming techniques in Java. Somewhat similar to our discussion during class of match.com, the theme of this programming project will also be social networks. In particular, this project asks you to think about how you would build a site called Facebuk, described in the next section.

**Facebuk**

**People, Pets, & Possessions**

Facebuk is a social networking site, somewhat akin to Facebook, that supports profiles not only for **people**, but also people's **pets** and **possessions**. People love their pets and love posting photos of their pets. Moreover, people sometime want to sell their possessions online. As an example of how Facebuk might look, suppose that Michelle Obama decided to join Facebuk (and I know she would). Her friends consist of Barack Obama, Kevin Spacey, and Ina Garten. She has two pets: Bo and Sunny. She also owns a kickboxing bag. Her dog Bo, in turn, has two friends of his own: Sunny (his sibling) as well as Malia Obama. Finally, the kickboxing bag is for sale; Michelle Obama is willing to part with it for just $20. The three pages of Facebuk which represent (respectively) Michelle Obama, Bo Obama, and Ms. Obama's kickboxing bag are shown below:  
  
  


**Moments**

In addition to linking people, pets, and possessions, Facebuk also enables users to capture **moments** in which multiple **people** and **pets** -- but **not** possessions -- participated and are captured in a photograph. See one of Ms. Obama's moments below, in which a list of the participants are displayed below the photo:  


**Happiness**

Facebuk wants its users to be happy (isn't that nice?). For this reason, Facebuk provides its users with the ability to tag each moment with how happy each participant -- which can be either a person or a pet -- appears during that moment's image. These "happiness" tags could be provided by an automatic smile detector (as we discussed in class), but in this project, for simplicity, they will just be input manually.

Using this "joy" information, Facebuk can offer its users two special search functions:

1. For a specific person *p*, find the friend *f* with whom person *p* appears the **most happy on average**, over all the moments in which both *p* and her/his friend *f* participate. If person *p* has no friends with whom she/he appears in the same moment, then return null.
2. For a specific person *p*, find the moment *m* (in which *p* participated) in which the average happiness value -- over all participants in *m* (including both people and pets) -- is highest. If the person does not participate in any moments, then return null.

By "average" we mean the [arithmetic mean](https://en.wikipedia.org/wiki/Arithmetic_mean).

**Requirements**

This project consists of both *data modeling* and *algorithmic* challenges:

**Data modeling**

Your job is to implement a set of classes and/or interfaces to support the fundamental objects that constitute Facebuk, as well as the relationships that connect the different objects together. In particular, you must support the creation of objects of type:

* Person: Each person can have pets, friends, possessions. Also, she/he can participate in moments.
* Pet: Each pet has an owner and can have friends.
* Possession: Each possesion has a sole owner; it has no friends :-(.
* Moment: Each moment is associated with a list of participants along with their respective *happiness* values.

In addition, **all** objects have a name and an image. To keep things simple, we pretend to "load" an Image from disk by specifying a hypothetical filename, e.g., new Image("Barack.png").

No matter how you design your set of classes and interfaces, you are required to include the following constructors:

* Person (String name, Image image)
* Pet (String name, Image image)
* Possession (String name, Image image, float price)
* Moment (String name, Image image, ArrayList participants, ArrayList smileValues), in which the array of participants should contain objects of type Person and the array of smileValues should contain objects of type Float that correspond to the levels of happiness of each person.

In addition, your code is required to implement the following methods (all of which should return void):

* Person.setFriends (ArrayList friends) -- sets the list of friends for the target person
* Person.setMoments (ArrayList moments) -- sets the list of moments for the target person
* Pet.setOwners (ArrayList owners) -- sets the list of owners for the target pet
* Possession.setOwner (Person owner) -- sets the sole owner for the target object

In addition to writing Java code that models the objects and relationships described above, you also need to describe your *rationale* for how and why you created the classes and interfaces that you chose. In particular, in a README file, for **every** class (abstract or concrete) and interface that you use, describe **why** you use it and **what** purpose it serves within the greater project.

**Style**

Make sure your code adheres to the style guidelines given during Lecture 1. You additionally need to make sure that your use of **whitespace** is consistent -- no randomly ragged indentation is permitted.

**ArrayList**

In this project you will be using the [ArrayList](http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) class. ArrayList is a data structure used to manage a **variable-length list** -- similar in purpose to an array, but you don't have to resize it every time the array reaches capacity. ArrayList is an extremely useful class in many applications, and we will talk about several other useful data structures over the coming 1-2 weeks.

The proper way of using an ArrayList is to specify the **type** of object you want to store in it when you declare and instantiate it. **However**, this gets surprisingly complicated when you want to store objects that can derive from a hierarchy of classes and/or implement one or more interfaces. We will cover this later in this course. **For now**, you should use ArrayList in a "type-unsafe" fashion: just declare, instantiate, and use it in the following way:

ArrayList list = new ArrayList();  
list.add(person);

where person is an object of type Person. Because this is "type-unsafe", you will get a compiler warning:

make Note: FacebukTester.java uses unchecked or unsafe operations.  
Note: Recompile with -Xlint:unchecked for details. It's ok to ignore this warning for this assignment.

To later retrieve an object from the array, use: list.get(idx); where idx is the index of the element you want to retrieve. Note that, when you use ArrayList in this manner, the return value of list.get(idx)will be an Object; hence, to cast it back to a Person, use: Person p = (Person) list.get(idx); The cast will become unnecessary once we talk about *generic types* later in the course.

**Happiness computation**

Your code should implement the following methods:

* Person.getFriendWithWhomIAmHappiest () which returns float
* Person.getOverallHappiestMoment () which returns Moment

For a description of what these should do, see above section on [happiness](https://ia.wpi.edu/cs210x/real_files.php?page=show_project&id=13#happiness). We will test your code automatically and/or by manual inspection.

**Teamwork**

You may work as a team on this project; the maximum team size is 3.

**Getting started**

Download this [zip file](https://www.cs.wpi.edu/~cs210x/b16/Project0.zip) that contains FacebukTester.java and Image.java. You should not need to modify the Image class, but you likely will want to add more tests to the tester.

**Unit testing**

We have given you starter code (FacebukTester.java) to help you with testing. We will use a larger version of this file to test your code; hence, **it is absolutely imperative that your code can compile against this tester**. If your code does not compile against our test code, you may unfortunately get 0 credit.

The only algorithmically difficult parts of this assignment are about [happiness](https://ia.wpi.edu/cs210x/real_files.php?page=show_project&id=13#happiness); make sure that you use sufficient unit testing to verify the correctness of your implementation!

**Grading**

Grading will be based on (1) your code compiling against our test code; (2) your written description (in the README you write) and justification of the design choices you made; (3) your code passing our test cases; and (4) your adherence to consistent and appropriate style.

**What to Submit**

Create a Zip file containing all of your .java files (no class files), as well as a README file describing your object-oriented design rationale. Submit the Zip file you created to InstructAssist. **Submission deadline**: Tuesday, November 1, at 11:59pm EDT.