Eric Roberts and Jerry Cain Handout #43
CS 106J May 24, 2017

Section Handout #7: Objects and Maps

Portions of this handout by Eric Roberts

It's week 8, which means you have at least 8 weeks of amazing CS education under your belt! You have become strong coders, and it's time to put your skills to the test. Who needs Google when you can code your own?

This week's section will be a combination of open-ended design and practicing specific skills. As you go through this handout, keep in mind these concepts you should be getting familiar with this week:

- Iterating through a map and populating a map
- Adding methods and fields to objects
- Writing getters, setters, and factory functions
- Understanding references, encapsulation, and information hiding
- Reading in input from the user from the console / reading from a file

1. Maps

For this problem, we will be given the following data file:

```
roads.txt
Los Angeles - San Diego
Portland - Seattle
Reno - Salt Lake City
Sacramento - Reno
San Francisco - Sacramento
San Francisco - Los Angeles
...
```

If there is a road connecting two cities, it will be listed in this file. Note that the roads are bidirectional. *Given this file, your task is to read this file into your program and store the information internally in a way that makes it easy for you to do the second task*. An example of "storing internally" would be reading the file and making a variable that contains one giant string, but that wouldn't be very helpful. Most likely, you will want to do a process known as parsing the data.

The second part of your task is to write a function that returns a boolean of whether two cities can be reached by taking at most two roads (gas is expensive). With the roads listed above:

```
function isReachableInTwo(currentCity, destinationCity)
isReachableInTwo ("San Francisco", "Reno"); //true
isReachableInTwo ("San Diego", "Los Angeles"); //true
isReachableInTwo ("San Francisco", "Salt Lake City"); //false
```

We may not get to this in section, but it is important to also think about how you can upgrade your cities to contain more information. Suppose you wanted to include information like a city's population, average income, and number of restauarnts. How would you do that?

This way, instead of just reaching a city within 2 steps, you could be more specific and query "Can I reach this city within 2 steps AND pass through cities with at least 1000 restaurants?"

2. Search

When you type a search query into a search engine, how exactly is it able to find the sites you are looking for? First, search engines visit as many websites as they can using something called a spider program that just repeatedly follows all the links out of each page. Each website gets processed and added to an index of the web. Finally, when you input search queries, they are able to look through their index of the web to return your results that match your query.

To those interested: https://www.google.com/search/howsearchworks/

We will build our own search functionality in this problem. You are given an array of filenames.

```
var filenames = ["TheCatAndTheHat.txt", "HowAreGummyWormsMade.txt",
"EricRobertsWiki.txt", "JerryCainCS110Homepage.txt", "IntroPhotography.txt"]
```

There may be many more files, but each of these files contains the text you would find on a website. *Your first task is to read each file and find the most frequent word in each text.* That will be our very simple tag of the website. You need to store the tag with the original document in a structure that makes sense to you. As an aside, recall when we learned about using arrays to help count things. How is this similar or different?

Your second task is to provide the user with a search function. They will input search queries, and if the query entered matches one of your tags, you should return the filename tagged with that tag. If there are none, you can return nothing. If there are multiple, return them all.

For example, if "cat" shows up the most in "TheCatInTheHat.txt",

```
    function search(search_query)
    search("cat"); //returns "TheCatInTheHat.txt"
```

3. Polyjuice Potion

And what handout would be complete without Harry Potter. (Hi Professor Roberts!)

In J. K. Rowling's *Harry Potter* series, the students at Hogwarts School of Witchcraft and Wizardry study many forms of magic. One of the most difficult fields of study is potions, which is taught by Harry's least favorite teacher, Professor Snape (played in the movies by the late Alan Rickman). Mastery of potions requires students to memorize complex lists of ingredients for creating the desired magical concoctions. Presumably to protect those of us in the Muggle world, Rowling does not give us a complete ingredient list for most of the potions used in the series, but we do learn about a few, including those shown in the following data file:

Potions.txt

```
Polyjuice Potion
shredded boomslang skin
lacewing flies
leeches
knotgrass
powdered bicorn horn
fluxweed
a bit of the person one wants to become

Wit-Sharpening Potion
ground scarab beetle
ginger root
armadillo bile

Shrinking Solution
chopped daisy roots
skinned shrivelfig
sliced caterpillar
rat spleen
leech juice

Draught of Living Death
asphodel in an infusion of wormwood
valerian roots
sopophorous bean
```

As the example illustrates, the format of the data file is a sequence of individual potions, each of which consists of the name of the potion, followed by a list of ingredients, one per line. Each of the potions is separated from the next by a blank line. Your task in this problem is to design the data structures necessary to represent the information about potions contained in this file, which requires you to complete the following tasks:

a) *Design a class to represent an individual potion*. Your first step in this process is to define a class called **`Potion`** that represents a single potion, which has a name and a list of ingredients. Your class should implement the following public interface:

- A factory method that takes the name of the potion and creates a **`Potion`** object with that name and an empty list of ingredients.
- A **`getName`** method that returns the name of the potion.
- An **`addIngredient`** method that adds an ingredient to the **`Potion`**.
- A **`getIngredients`** method that returns an array of the ingredients required for the potion listed in the order in which they were added.

b) *Design a class to represent a collection of potions*. Your next step is to define a **`PotionCollection`** class that represents the entire collection of potions. This class should export the following entries:

- A factory method that takes the name of a file formatted in the style described on the previous page. If the file cannot be read, the factory method should return **`null`**.

- A **`getPotion`** method that takes the name of a potion and returns the **`Potion`** object that has that name, or **`null`** if there is no potion with that name.

- A **`getPotionNames`** method that returns an array list of the potion names in the order in which they appeared in the file.

c) *Write a main program that tests your implementation of the data structure*. Your program should prompt the user to enter the name of a potion and then list all the necessary ingredients on the console.

p.s. I apologize that there are no pictures and so much text. Word was being especially cranky with formatting… ☹