Nick Troccoli
CS 106A

Section #9
March 12, 2018

Section Handout #9: Lesson Plan

Written by Nick Troccoli

This week, students are tasked with designing a larger client-server application that combines may critical CS106A concepts from the past few weeks, including:

- Internet applications
- File reading
- HashMaps
- ArrayLists
- Classes
- Strings

This program is intentionally open-ended, to allow you and your students to discuss the best way to design the client and server programs. To help guide this discussion, we recommend the following lesson plan for discussing the program design.

1 – What does the client-server communication look like?

Discuss the job of the client program, and how the server (from a high level) helps achieve that. For instance, you could pose the question "ok, the user wants the phone numbers of all congress members for Maryland, how do we do that?" This lets you discuss why each type of request exists.

2 – How should we store the data in the server?

Discuss what the server needs to do with the data most frequently (answer: look up congress members by state) and what data structure might fit that best (answer: HashMap from state code to members). Then, discuss what the "list of members" might look like. Can we store it as strings?

Some students may suggest just storing the member information as a list of strings. The reason we cannot store it as strings is because we need many pieces of information – the name, the phone number, and the email – and we *don't know* which (email or phone) will be needed for a given request.

If we need to put information together into a logical group, a class may be a good idea. Discuss how a new variable type might help make it easier to store this information. Go through the 3-step process for designing a class: **1)** what data does it store? (instance variables) **2)** what can it do? (public methods) **3)** how do you create one? (constructor). There are multiple approaches here, from implementing string methods (like in the solution) directly in the class, or just making getters and doing string processing elsewhere. Also ask how we might indicate when a member doesn't have an email address (answer: either have email be null, empty string, etc.).

The end result is you should agree on a map from state codes to a list of congress members.

3 – How should we read data from file?

Discuss the process of getting data from file into your map. This is a great chance to go over the format of the data file:

```
NAME
STATE CODE
PHONE
EMAIL (blank if no email)

NAME
STATE CODE
PHONE
EMAIL (blank if no email)
...
```

You can mention that students may assume there are always 4 lines per member, followed by a blank line, and that the email line may sometimes be blank. How can we handle that?

4 – Writing `requestMade`

Discuss how we can use the data structure we have created and populated to respond to requests. Take one type of request at a time:

`getCongressPhonesForState` – what data do we have in this request? (answer: state code). How can we use that data to find the relevant congress members? (answer: look up in map) Once we have them, how do we make a string response containing the right information? (answer: build up string with data from each member). What happens if we have a state that is not in our map? (answer: return error message).

`getCongressEmailsForState` – this is very similar to the previous command, with the exception of emails not being present for all members. How can we detect if a member doesn't have an email address, and how can we then make the appropriate string response?

5 – Client Program

Now, let's turn to the client program. What is the main interaction loop with the user? What steps are we repeating each time? (see handout for explicit answer). How do we know what type of request to send to get the information we need? (answer: depending on email or phone). How do we output the response to the user? (answer: just print!).