San Francisco State University
CSC 220: Data Structures
Dawson Zhou (zhoud@sfsu.edu)

Assignment 1: Collections

Overview

Welcome to Java Collections! In this assignment, you'll use what you've learned about the various Java collections to solve four small problems: one involving each of the stack, the queue, the set, and the map. There is no dedicated problem for the ArrayList, because several of the other problems involve an ArrayList in some way.

Submission

Assignment 1 is due on **September 20, 2021** at **11:59 pm PT**.

Late submissions will be accepted with a penalty:

- September 21, 2021 12:00 am PT to September 21, 2021 11:59 pm PT will receive 75% credit
- September 22, 2021 12:00 am PT to September 22, 2021 11:59 pm PT will receive 50% credit

Submit your solution through iLearn by uploading the **Program.java** file.

Grading

100 pts total:

- 30 pts: `cancelingWords`
- 15 pts: `shuffleLine`
- 15 pts: `printExtraNames`
- 30 pts: `populateNameMap`
- 10 pts: Miscellaneous (abiding by various style rules, etc.)

Assignment Setup

Download the folder named “Collections Starter Code” from iLearn. It will download as a .zip file which can be extracted to whatever location you want on your computer. Set up an IntelliJ project in the unzipped directory. For step-by-step instructions on setting up an IntelliJ project, see the “IntelliJ Setup Guide” handout on iLearn (at the top of the iLearn course page).

Detailed Requirements

The four methods you will need to implement are **`cancelingWords`**, **`shuffleLine`**, **`printExtraNames`**, and **`populateNameMap`**. Each of these methods has its signature provided (name, return type, and parameter list), with an empty body marked by a comment (“TODO: Implement.”). Your task is to write the code for the bodies of those four methods.

The **`main`** method is already provided for you. It calls four test methods: one for each of the four methods you’re implementing. Those are also already provided for you. You are welcome, and encouraged, to modify and add to the test examples to make sure that your implementation is complete.

In order to receive full credit for the assignment you:

- Cannot modify any of the provided method signatures (you can only change the code in their bodies)
- Cannot use any instance or global variables
- Must account for edge cases (see more details below).

`cancelingWords`

`cancelingWords` is a method which takes a parameter called `words` which is an **`ArrayList<String>`**. Your task is to process the strings in `words` from beginning to end, canceling out and eliminating any adjacent pairs of identical words. When a pair of adjacent words is eliminated, we continue applying this process to the remaining words. Once all adjacent pairs have been eliminated, you should print out the leftover words, one on each line.

In the example provided in the starter code, the `words` parameter contains the following:

- `happy`
- `sad`
- `cat`
- `cat`
- `dog`
- `dog`
- `sad`

Two copies of **`"cat"`** appear adjacent to one another, as do two copies of **`"dog"`**, so we would eliminate those, resulting in:

- `happy`
- `sad`
- `sad`

`"sad"` now has two copies adjacent to one another, so we would eliminate those as well. As a result, only **`"happy"`** would be printed.

To receive full credit, you will need to follow a few restrictions:

- You cannot modify the `words` parameter in any way -- no adding, removing, or updating any elements. You also cannot make a copy of `words` and modify that instead.
- You must solve the problem with a single "pass" through `words`. Meaning, you will need to solve it without using nested loops.

If you think about the problem carefully, it has a lot in common with the balanced parentheses demo we went over in class, making the Stack an ideal collection. As you iterate through `words`, you can store the strings you encounter until you find a case where the string you're looking at matches whatever was last pushed onto the Stack.

Remember that when comparing strings for equality, you'll need to use the **`.equals`**`(...)` method rather than the `==` operator.

Finally, to print out all of the contents of a stack in order, you can use the for-each loop we talked about in class:

```
for (String word : wordStack) {
    System.out.println(word);
}
```

shuffleLine

`shuffleLine` is a method which takes a first parameter called `line` which is a **`Queue<String>`** containing the names of people, and a second parameter called `newPersonInFront` which is a single String name. You can assume that `newPersonInFront` is a name that also appears somewhere in the queue. Your task is to shuffle the order of the names in the queue by repeatedly taking the name of the person at the front of the line and putting them at the back, until `newPersonInFront` is at the front of the queue.

In the example provided in the starter code, the queue contains the following names:

- `Troy`
- `Annie`
- `Britta`
- `Abed`
- `Shirley`
- `Jeff`
- `Pierce`

The second parameter, `newPersonInFront`, is `"Abed"`. After shuffleLine is called with those two parameters, the queue should look as follows:

- `Abed`
- `Shirley`
- `Jeff`
- `Pierce`
- `Troy`
- `Annie`
- `Britta`

The contents of the queue will be printed out by testShuffleLine, so you can verify whether your solution is making the correct changes.

printExtraNames

`printExtraNames` is a method which takes a parameter called `names` which is an **`ArrayList<String>`** containing names, including some duplicates. Your task is to print out every name in `names` which is a duplicate. If a particular name appears 3 times, for example, that name should be printed out twice -- the first appearance is an original, but the next two appearances are duplicates.

In the example provided in the starter code, the list contains the following names:

- `Troy`
- `Annie`
- `Britta`
- `Jeff`
- `Abed`
- `Shirley`
- `Abed`
- `Jeff`
- `Pierce`
- `Abed`
- `Troy`

Your solution should print out these names, each on their own line, in this specific order:

```
Abed
Jeff
Abed
Troy
```

In order to receive full credit for this problem, you must make use of a set collection. You can use either a **`TreeSet`** or a **`HashSet`**.

populateNameMap

populateNameMap is a method which takes a first parameter called `names` which is an **`ArrayList<String>`** consisting of a list of String names, and a second parameter called `nameMap` which is a **`Map<String, String>`**. `nameMap` will start off as an empty map. Your task is to populate that map with name entries based on what is provided in `names`.

For this problem, name entries are defined as mappings between the first letter of a name and the full name itself. For example, if we see the name `"Troy"` in the list of names, we would want to add a corresponding key-value pair to the map, with `"T"` as the key and `"Troy"` as the value.

One caveat is that there are multiple names which start with the same first letter. In those cases, we should combine the names, separated by a comma. When we first see the name "Annie" in the names list, we'll add a key-value pair with `"A"` as the key and `"Annie"` as the value. When we later see `"Abed"` in the names list, we'll need to recognize that the key `"A"` is already in the map. In that scenario, we'll add a comma `","` and `"Abed"` to the end of the existing value. Now, the key-value pair will have `"A"` as the key and `"Annie,Abed"` as the value.

In the example provided in the starter code, the list contains the following names:

- `Troy`
- `Annie`
- `Britta`
- `Abed`
- `Shirley`
- `Jeff`
- `Pierce`
- `Ben`
- `Alf`

The resulting test printout should read:

```
A: Annie,Abed,Alf
B: Britta,Ben
J: Jeff
P: Pierce
S: Shirley
T: Troy
```

You may find the following logic useful. In order to find the String representing the first letter of another String named `str`, you can use the following code:

```
String firstLetter = str.substring(0, 1);
```

Consider how you can replace `str` in this example with the appropriate String when you are implementing your solution.

You will also need to combine multiple Strings to form a single String. To do so, you can use + (plus) to combine Strings, as has been demonstrated in class and in various demos.