Department of Computer Science and Engineering
University of Puerto Rico
Mayagüez Campus

CIIC4020 / ICOM4035 – Data Structures
Spring 2021-2022
Project #1: Instant Runoff Voting

Introduction

It's election year in the country of Poor Harbor and changes are being considered for the voting system. The current system allows for the candidate with the highest amount of votes to be declared as the winner. However, in the last two elections the winners received less than 50% of the votes, causing citizens to question whether it was legitimate to have a president that was rejected by more than half of the voters. The lawmakers of Poor Harbor have since decided to implement an alternate voting method called ***Instant Runoff Voting***.

Using this alternate voting method, if there are *n* candidates running for a particular position, voters can vote for all *n* candidates, giving each candidate a rank of preference, from 1 to *n*. In general, an election may include several elected positions, with several candidates running for each position, and several cast ballots. However, we'll focus on an election for a single position.

You will write a program that is able to receive data about the candidates for the position and votes cast in the election, then process the votes to determine the final results based on the counting rules that we are about to discuss.

Some people might claim that the new voting system exhibits higher fairness than the current one, while others might disagree. No voting system is perfect, but it is outside the scope of this project to discuss all of the pros and cons of these systems in particular.

Valid ballots

In each ballot voters can:

- vote for any number of candidates from 0 (blank ballot) up to *n*, where *n* is the number of candidates running for that particular position.
- rank each candidate according to his/her preference, assigning the value 1 to the highest preference, 2 to the second-highest preference, and so on, with a higher value indicating a less-preferred candidate.
- no ranking value can be repeated in a ballot; otherwise, the ballot is invalid.
- no candidate can have a ranking vote greater than the number of candidates: $k \leq n$, where *k* is the rank of the candidate and *n* is the number of candidates. If a ranking value is greater than *n* is given, the ballot is also considered invalid.

Counting rules

The counting process may consist of several rounds or iterations. In each round, if one candidate has more than 50% of 1’s, that candidate wins the election for that position and the counting process concludes. Otherwise, the candidate having the lowest amount of 1’s is eliminated and will not be considered in the subsequent counting rounds.

If there is a tie for lowest amount of 1’s, then the decision is based on the lowest amount of 2’s (between those that tied for the lowest amount of 1’s). If again there is a tie, then the decision is based on the lowest amount of 3’s (between those that tied for the lowest amount of 2’s), and so on. If at the end no decision can be made, then, among all of those candidates that are still tied, the onehaving the current largest ID# is removed.

When a candidate is eliminated, the counting for the next round must take into account the voters’ preferences. If the #1 candidate in a ballot is eliminated, then the higher-ranked candidates are “moved up” a position, so that the #2 candidate now becomes #1, the #3 candidate now becomes #2, and so on. In general, if the candidate being eliminated was given a rank value of *k* in a ballot, then all candidates on that ballot with a rank greater than *k* will have their rank subtracted by 1, hence giving them a lower rank (higher preference).

Election example

Consider the following election with 5 candidates and 10 ballots. Notice that no candidate has more than five (50%) 1’s.

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}		1s	2s	3s	4s	5s
Pepe	1	4	4	5	1	1	4	1	3	4		4	0	1	4	1
Juan	2	3	2	1	2	5	2	2	4	2		1	6	1	1	1
Lola	3	1	3	3	4	3	5	4	2	3		1	1	5	2	1
Pardeep	4	5	1	2	3	2	3	5	5	1		2	2	2	1	3
Pucho	5	2	5	4	5	4	1	3	1	5		2	1	1	2	4

<u>Round 1:</u> Notice that Juan and Lola both have the least amount of 1’s. Then, following the counting rules, we compare the amount of 2’s (only between Juan and Lola). Juan has six 2’sand Lola has only one, so Lola is eliminated. The ballots are updated as follows:

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}		1s	2s	3s	4s
Pepe	1	3	3	4	1	1	4	1	2	3		4	1	3	2
Juan	2	2	2	1	2	4	2	2	3	2		1	7	1	1
Pardeep	3	4	1	2	3	2	3	4	4	1		2	2	3	3
Pucho	4	1	4	3	4	3	1	3	1	4		3	0	3	4

<u>Round 2:</u> Juan has the least amount of 1's, so he is eliminated. The ballots are updated asfollows:

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}		1s	2s	3s
Pepe	1	2	2	3	1	1	3	1	2	2		4	4	2
Pardeep	2	3	1	1	2	2	2	3	3	1		3	4	3
Pucho	3	1	3	2	3	3	1	2	1	3		3	2	5

<u>Round 3:</u> No candidate has yet reached at least 50% of 1's, so we need to eliminate another candidate. Pardeep and Pucho are tied for 1's, but Pardeep has more 2's, so Pucho is eliminated. The ballots are updated as follows:

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}		1s	2s
Pepe	1	1	2	2	1	1	2	1	1	2		6	4
Pardeep	2	2	1	1	2	2	1	2	2	1		4	6

<u>Final result:</u> Pepe is the winner!

Input data

The input for the program is to be read from two data files located in the same directory as your program, you will call this directory ***inputFiles*** and it will be stored in the same level as the src folder that eclipse generates for your Java project.

1. **`candidates.csv`:** Contains the data about the candidates. Each line contains two fields of information separated by a comma: The name of the candidate and the ID# assigned to thatcandidate. *Note that the ID numbers are 1-based* (lawmakers don't know about 0-based indexing). The candidates appear in the file in ascending ofder by **ID #**, starting at 1 and in sequential order (e.g. 1, 2, 3, etc.)

2. **`ballots.csv`:** Contains the data about the ballots and the votes cast by the voters. Each line starts with the ballot number followed by the votes in that ballot (in ascending order **by rank**),in the following format:

   $$\texttt{ballot\#,}\mathbf{c_1}\texttt{:1,}\mathbf{c_2}\texttt{:2,}\dots\texttt{,}\mathbf{c_k}\texttt{:k}$$

   where every c_i:i combination represents the candidate ID (c_i) to which the voter assigned the rank value `i`. Note that you must validate each ballot (See "Valid ballots" section).

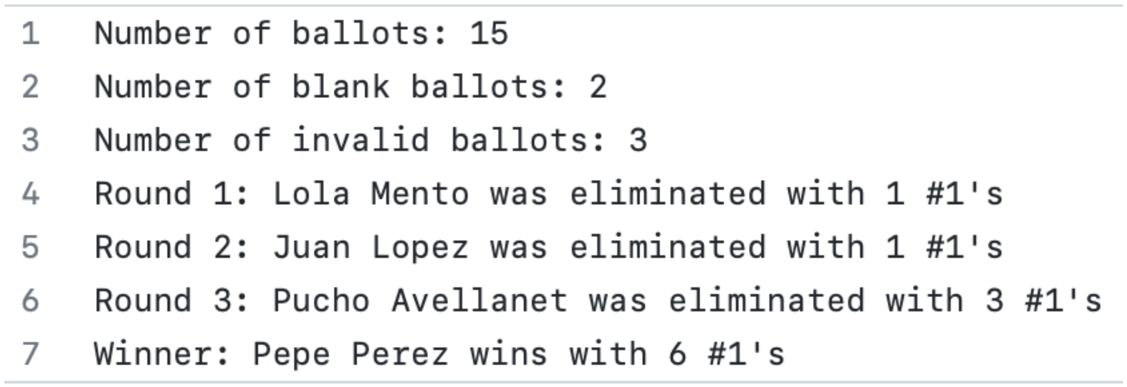
Note: For reading input files, check the `BufferedReader` class from the Java API

Note: Some parts of this project were left ambiguous (for example, the input and output) on purpose. Part of being a good Software Engineer is having the skill of researching on your own and having the curiosity of trying out new things, while still following instructions. It is the student's responsibility to research these things and try out different ways to implement solutions. Google will always be your best friend, but if you get stuck on a part of the implementation even after researching on the internet and external resources, don't hesitate to reach out to your lab instructor or the professor for help.

Output

You **must** save the results in the current directory in a file named `results.txt` (which your program must create). The results must include:

- Number of ballots received
- Number of blank ballots (no candidates were selected)
- Number of invalid ballots
- Round results. For each eliminated candidate you must store:
  - Number of 1's that the candidate received at the moment of elimination
  - Round in which the candidate was eliminated.
  - Use the following format: Round #: <Candidate name here> was eliminated with n #1's.
- Final result. Use the following format: Winner: <Candidate name here> wins with n #1's.



```
Number of ballots: 15
Number of blank ballots: 2
Number of invalid ballots: 3
Round 1: Lola Mento was eliminated with 1 #1's
Round 2: Juan Lopez was eliminated with 1 #1's
Round 3: Pucho Avellanet was eliminated with 3 #1's
Winner: Pepe Perez wins with 6 #1's
```

Example output from results.txt that your program should generate

Note: For creating and writing to files, check the `BufferedWriter`, `OutputStreamWriter` & `FileOutputStream` classes from the Java API

Sample data

You will be provided with sample input files with the corresponding output file so that you can test your program. Remember that one successful test case does NOT guarantee that your program is correct, but it should at least help you to find some of the most common errors. Additionally, the sample output file (and the screenshot above) will show you **exactly** how your output should beformatted.

Technical details

Ballots

You must create a `Ballot` class, such that an instance of this class can store all of the information regarding a single ballot. It must contain the ballot number as well as a List to contain the votes that were cast in that ballot (you can use more than one List if you wish, but technically you can do it with only one). How you use the list(s) and what you store in them is up to you. Your Ballot class must contain *at least* the following methods:

```
    public int getBallotNum(); // returns the ballot number
    public int getRankByCandidate(int candidateID); // rank for that candidate
    public int getCandidateByRank(int rank); // candidate with that rank
    public boolean eliminate(int candidateId); // eliminates a candidate
```

An interface will be provided in the initial code, which you'll have to *implement* in your `Ballot` class.
Inside the actual Ballot class, create a constructor such that you could pass the line of text from the input file and then theconstructor would take care of storing the information accordingly. You may create as many additional methods as you deem necessary and/or helpful.

Storing the ballots

The ballots' votes must be stored in *n* Lists, where *n* is the number of candidates. The best way to collect and manipulate this collection of lists is, of course, another list. The List L_i must contain all the ballots where candidate with ID *i*+1 has a rank value of 1. As you read the input file, you must create `Ballot` objects and store them in the appropriate list depending on which candidate has a rank value of 1 in that ballot (every valid ballot must have such a candidate).
<u>Note that empty ballots and invalid ballots should not be stored, only accounted for.</u>

Removing candidates (Design vs. Implementation)

Although in the explanation of the voting method a candidate disappears from the ballots once he/she is eliminated, that would require going through **all** of the ballots and performing a search for that candidate. The reason we are using separate lists for the candidates is to avoid that overhead. Instead, we're only going to modify the ballots where the candidate being removed has a rank value of 1 (the "top candidate"), so that we only have to loop over that particular list. The only downside to this strategy is that when we remove the top candidate from a ballot, it's possible that the new top candidate might have also been previously eliminated and we hadn't removed him/her from that ballot. To be able to track the eliminated candidates you have to create a `Candidate` class where you can store details about the candidate (name, age, ….), beside if the candidate is `active` or not. This way you can check if the top candidates in the ballots as you are modifying them.

Using the same example as before, suppose we start as follows:

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
Pepe	1	4	4	5	1	1	4	1	3	4
Juan	2	3	2	1	2	5	2	2	4	2
Lola	3	1	3	3	4	3	5	4	2	3
Pardeep	4	5	1	2	3	2	3	5	5	1
Pucho	5	2	5	4	5	4	1	3	1	5

Once we eliminate Lola, we have:

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
Pepe	1	3	4	5	1	1	4	1	3	4
Juan	2	2	2	1	2	5	2	2	4	2
Lola	3		3	3	4	3	5	4	2	3
Pardeep	4	4	1	2	3	2	3	5	5	1
Pucho	5	1	5	4	5	4	1	3	1	5

eliminated = { Lola }

Note that we only had to modify one ballot (b_2)!
In this round, Juan gets eliminated, but fortunately we only have to modify b_4.

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
Pepe	1	3	4	4	1	1	4	1	3	4
Juan	2	2	2		2	5	2	2	4	2
Lola	3		3	2	4	3	5	4	2	3
Pardeep	4	4	1	1	3	2	3	5	5	1
Pucho	5	1	5	3	5	4	1	3	1	5

eliminated = { Lola, Juan }

Now Pucho gets eliminated. When updating b_2, we notice that Juan moves up to be the top candidate, but Juan was already eliminated, so we modify the ballot again. The same thing happens with b_7 and b_9, and we end up with:

Candidate	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
Pepe	1	1	4	4	1	1	2	1	1	4
Juan	2		2		2	5		2	2	2
Lola	3		3	2	4	3	3	4		3
Pardeep	4	2	1	1	3	2	1	5	3	1
Pucho	5		5	3	5	4		3		5

eliminated = { Lola, Juan, Pucho }

And Pepe wins (again).

Note that *half of the ballots were never modified*, thanks to the strategy of separating the ballots by top candidate and only modifying when necessary. Otherwise, we would have had to modify all 10 ballots in every single round. Technically we now loop over the *eliminated* set instead, but that is a much smaller set. Another advantage is that now it's really easy to find out at any point the number of ballots in which a candidate has a rank value of 1 (think about it). However, we still loop over all ballots to count the amount of votes each candidate has for other ranks, but this in not different than before.

Election class

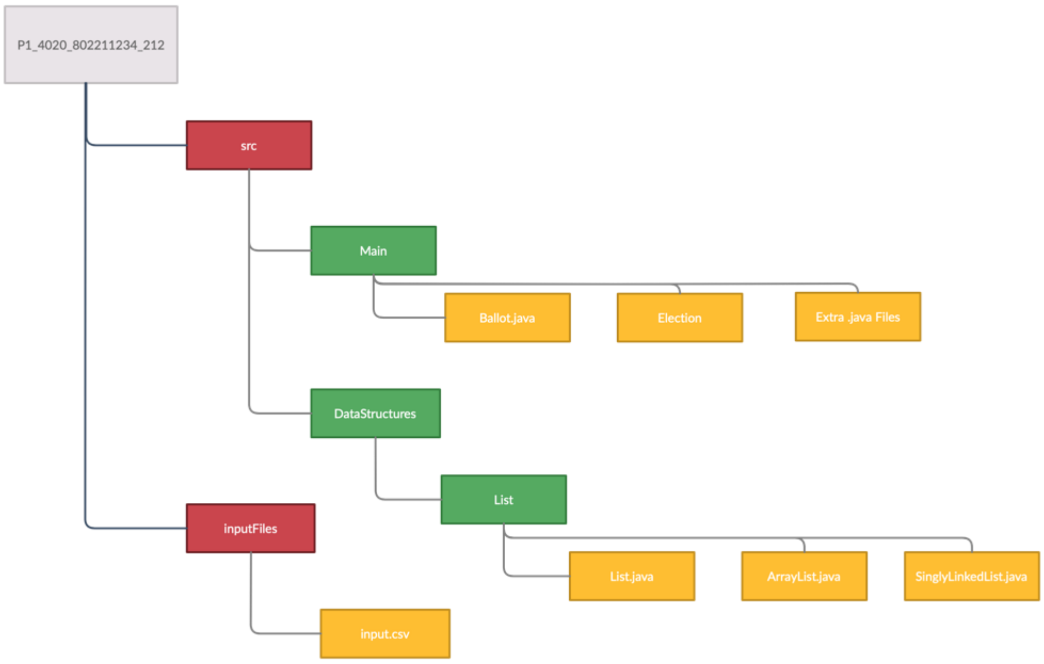
Your main program must be named Election.java, and it must implement all of the aforementioned logic using the Ballot class you created. *You must use **only locally defined data structures**; you **cannot** use Java's built-in List structures or any that are provided within the Java Collections Framework (List, Set, Stack, Queue, HashMap, etc.).* You will lose points if there is any trace of use of Java's Data Structures within the project implementation.

Documentation & Comments

You must properly document your code using the Javadoc format. Include at the top of the class some comments explaining the structure of your class, particularly what type of list(s) you are using and why. For example, "An ArrayList was used to store the votes, where the top candidate is the last element of the list, so that when that top candidate is removed...".
Different people will have different implementations, so it's important that your code is well documented and commented so that we may understand your intention. You will lose points if you don't provide comments or if we deem they are inadequate.

Project Structure & GitHub Repository

Even though you will have total control over how you implement this project, you will still need to follow some project structure. You will be required to store all of your files inside of your eclipse project in the following matter:



Note that the grey box is the root folder of your eclipse project, red boxes are source folders, green boxes are packages and yellow boxes are individual files within the project (.java, .csv, etc.). For the *DataStructures* package, you can add as many packages inside of it as you like, depending on how many Data Structures you use. For example, if the project used Lists and Stacks, you would have a *Lists* & Stack inside of the main *DataStructures* package. Failure to comply with this structure will result in subtracting points from your final project grade.

Note: You can have multiple types of files within inputFiles (ballots.csv, candidates.csv).
Note: Any type of .java file created to complement your solution (e.g., Candidate.java) ***MUST*** be inside of the Main package

You will be given an invitation to join a GitHub Classroom organization. Once accepted, you will be prompted to accept an assignment, in which GitHub will create a private repository with some initial code following the structure mentioned. You ***CANNOT*** change the structure given, only add packages and directories if needed (for example adding a *Set* package in *DataStructures* and adding the *inputFiles* folder). In this repository you will push all your changes while you make your project, as well as roll back to a previous commit of you must.

Submission

The final date to make commits to your GitHub Classroom Repository will be **Friday, March 18, 2022, at 11:59pm**. Any commits made after this deadline will NOT be considered for grading, we will only take the last commit made before the deadline into consideration. If the case is that you upload your first commit to your repository after the deadline, it will be graded but it will also result in subtracting points from your final project grade.

Academic Integrity

Do NOT share your code! You may discuss design/implementation strategies, but *if we find projects that are too similar for it to be a coincidence, all parties involved will receive a grade of 0*. Don't cheat yourself out of a learning experience; seek our help if you need it.

Final comments

The specifications of a project are the first, and arguably the most important, part of a software development project. Therefore, it's crucial that you read these specifications thoroughly so that you understand what is being asked of you. These are skills that you will need to succeed in your professional career, so it's imperative that you start applying and improving them now. If your program runs successfully, but does not adhere to the specifications, it is of no use.
Before you submit your project, review these specifications one last time and make sure youmeet all of the requirements that have been imposed.

If your code does not compile properly, your grade will be 0, NO EXCEPTIONS!