CITS1001 Project 1, Semester 1 2020

Version 1.00, 10am 27 March 2020.
Check the LMS to ensure that you have the latest version.
If you have any questions about any aspect of the project, submit them to *help1001*.

Project Rules

- Submission deadline: **5pm Friday 24 April 2020**
- Submit via ***cssubmit***
- Value: **15% of CITS1001**
- Project work to be done **individually**
- Project published: 27 March 2020

The project task is to construct a Java program containing your solution to the following problem. You must submit your program via *cssubmit*. No other method of submission is allowed.

You are expected to have read and understood the UWA Policy on Academic Conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort.

You must submit your project before the submission deadline above. UWA's Policy on Late Submission will apply.

Project Overview

The penguins of Antarctica hold elections every twenty years. They use an unusual voting process where if there are *n* candidates in an electorate, each voter has *n* votes to distribute amongst them in any pattern they want. A voting paper is *formal* iff it contains exactly *n* marks (corresponding to the *n* candidates) each in the range *0..n* inclusive and adding up to no more than *n*. So for an election with four candidates, all of the following would be formal papers:

4,0,0,0

2,0,0,2

0,0,3,0

0,2,1,0

1,1,1,1

Obviously many other formal votes are possible. All of the following would be informal papers:

3,0,2,0 - total too high

3,0,2,-1 - negative votes not permitted

3,0,0 - too few marks

3,0,0,0,0 - too many marks

To determine the winner of the election, first of all the informal papers are discarded, then two totals are calculated for each candidate.

- The number of votes for a candidate is the sum of the votes given to that candidate on each formal paper. So given the nine papers above, the total vote for the first candidate would be 4+2+0+0+1 = 7. The total votes for the other candidates would be 3, 5, 3 respectively.
- The number of wins for a candidate is the number of formal papers on which they receive the highest number of votes, each one divided by the number of candidates who receive that highest number. So given the nine papers above, the total wins for the first candidate would be 1.0+0.5+0.0+0.0+0.25 = 1.75. The total wins for the other candidates would be 1.25, 1.25, 0.75 respectively.

The winner of the election is the candidate who has the highest number of votes; if there is a tie, the winner is the candidate amongst those tied who has the highest number of wins. So given the nine papers above, the first candidate would be the winner.

You are required to write a program that reads in a file of candidates' names and voting papers, conducts the election count, and determines the winner of the election.

If you have any questions about any aspect of the problem description, submit them to *help1001*.

Reading in files of election info

In this project, candidates' names and the voting papers will be provided in a text file. Have a look at *election1.txt* as an example. Every election file will have

- the candidates' names, one per line, followed by

- exactly one blank line which should be discarded, followed by
- the voting papers, one per line.

You can read in these files very easily using the Java class `FileIO` which is provided. Simply declare a variable belonging to this class, and create an object with the name of the election file as the argument to the constructor. You can then access the contents of the file as separate lines using the appropriate accessor method. The project FileIOexample on the *Lecture Material* page shows a simple example of this in use.

Project Materials

Download the folder *Project1.zip* from the LMS Project 1 page. This folder contains the following.

1. Skeletons for three Java classes.
    - `VotingPaper` represents one voting paper.
    - `Candidate` represents one candidate in the election.
    - `Election` represents the entire election process.

    Collectively, these classes implement the election process. You are required to complete the constructors and methods for all three classes. Where the body of a constructor or method contains a comment `TODO`, delete the body and replace it with code which implements the required functionality, as described in the associated program comment. The numbers 1-23 in these comments suggest an order for you to perform these tasks. (Don't worry, many of these methods are very simple!)
2. A fully-written class `FileIO`, which will help you to perform file input operations. You do not need to change or submit this class.
3. Three JUnit test classes, which are provided for you to check your code. Note that the test cases are not complete; a method that gets all green ticks is not guaranteed to be completely correct. Additional test cases will be used for marking and it is your responsibility to thoroughly test your code.
4. Three sample input files *election*.txt*, and three associated output files *election*.out.txt*.

Project Management Tips

Before starting the project, students are expected to have

- studied the lectures and Chapters 1-4 of the text,
- completed the assigned labs during Weeks 2-5,

- read and understood the whole of this project description,
- read and understood all relevant UWA policies.

Submit any questions about any of this to *help1001*.

It is recommended that you tackle the project tasks in the order indicated; that you compile frequently; and that you test and run the code after completing each method, to ensure your code behaves as you think it ought, and does not fail any tests. If you are stuck on a method, it is often a good idea to look in the lecture material or in the text for an example method which is similar in some way to the one that is confusing you.

You can gain good marks even if you do not complete all the methods, so long as the code you have written compiles and runs correctly. But if you submit a large body of code that does not compile or that crashes, then few marks can be awarded.

Hints and tips about the various methods may be uploaded here from time to time. Whenever that happens, the document version number will be updated.

1. For getStanding, there's a helpful method in the library class Math for rounding doubles.
2. In processFile, each line of the file is either a candidate's name or a voting paper, except the blank line that divides the two sections. So when you are looking at a particular line, how do you know whether it is a name or a paper?

Project Submission

Submit your three completed **.java source files** by the submission deadline above via *cssubmit*. Do not submit anything else. No other method of submission is allowed.

The file names, class names, and method signatures in your submitted code must match the original specifications exactly. The marking process is partly automated, and any changes you make which complicate this process will be penalised. If your code cannot be compiled with the original JUnit test cases provided, your submission will be penalised. It is ok to add other methods if you like, so long as you do not change the signatures of existing methods.

Common mistakes are to submit .class files in place of .java files, or to submit the test classes in place of your code. If you do one of these (or something similar), you will be notified by email as soon as we become aware, but you will be due for any applicable late penalty up to the time you re-submit. *cssubmit* makes it easy to check your files after you have submitted them - do it!

Project Assessment

Your submission will be assessed on

- completeness: how many of the methods you have written;
- correctness: whether your methods implement the specifications exactly;
- clarity: whether your code is clear to read and well-constructed.

JUnit testing classes are provided to help you check your program before submission, but the correctness of your program remains your responsibility.

Completeness and Correctness guidelines (/44)

The marks available for each constructor and method are as follows. These numbers give some indication of how difficult we believe each component is.

`VotingPaper`:	`getMarks` (1), `isCorrectLength` (1), `isLegalTotal` (2), anyNegativeMarks (2), isFormal (2), updateVoteCounts (3), highestVote (4), updateWinCounts (3)
`Candidate`:	constructor (1), `getName` (1), `getNoOfVotes` (1), `getNoOfWins` (1), `addToCount` (1), `addToWins` (1), getStanding (2)
`Election`:	`constructor` (2), `getCandidates` (1), getPapers (1), getFile (1), processFile (4), `conductCount` (3), getStandings (3), winner (3)

Methods will be assessed independently, so e.g. even if your `updateVoteCounts` is faulty, other methods that use `updateVoteCounts` could still get full marks.

Completeness and correctness will be evaluated (partly) automatically using JUnit tests. You should, therefore, try to ensure the following.

- The code passes the tests as per the specification.
- Validation should be **exactly** what is required by the assignment specification.
- Incorrectly rejecting a parameter is incorrect and will be penalized, so do not add extra validation that is not asked for.
- Do not print things to `System.out` or `System.err` that are not asked for. Use the BlueJ debugger rather than `print/ln` statements for debugging your code.
- Do not change the given file names or class names, or modify the signatures of existing methods.

Clarity guidelines (/11)

- Ensure you fill in the @author and @version fields in the header comments with your details for each submitted class.
- All code should be neatly laid out and indented, with lines no longer than 80 characters.
- Do not add fields to the classes.

- Variables should be given appropriate names.
- Select appropriate programming constructs for the method implementations.
- Keep code as simple as possible for the job it is required to do.
- If the logic you have used in a method is particularly complex, a brief comment should be added explaining the strategy you have adopted; but otherwise, code should not be commented unnecessarily.

Help!

The quickest way to get help with the project (unless you are sitting in a lab session!) is via *help1001*. You can ask questions, browse previous questions and answers, and discuss all sorts of topics with both staff and other students.

Please read and follow these guidelines.

- Do not post project code on *help1001*. For obvious reasons, this behaviour undermines the entire assessment process: as such you will be liable for punishment under the University's Policy on Academic Conduct, even if the behaviour is unintentional.
- Before you start a new topic, check first to see if your question has already been asked and answered in an earlier topic. This will be the fastest way to get an answer!
- When you do start a new topic, give it a meaningful subject. This will help all students to search and to make use of information posted on the page.
- Feel free to post answers to other students' questions. And don't be afraid to rely on other students' answers: we read everything on *help1001* and we will correct anything inaccurate or incomplete that may be posted from time to time.
- Be civil. Speak to others as you would want them to speak to you. Remember that when you post anonymously, this just means that your name is not displayed on the page; your identity is still recorded on our systems. Poor behaviour will not be tolerated.
- Do not post project code on *help1001*, or anywhere else online. This one is worth saying twice.

If you have a question of a personal nature, do not post to *help1001*: please email Lyndon instead.

The project will also be a big topic for discussion in the online lab sessions via Zoom. This is fine of course, but again please be careful not to share code accidentally with other students. Some things to look out for include the following.

- Do not share your desktop or files in a Zoom session.

- Do not include code in any chat message.