4 Lab Assignment 4

You are to write a program to use an `ArrayList` and an insertionsort to permit breakpoint processing.

Background on the data

First, you have three raw data files labelled

- `datasubset.021`
- `datasubset.305bluff`
- `datasubset.378woodlands`

These are the raw ballot image files for precincts 21, 305, and 378 in Richland County. (My own precinct is 378; somewhere in that file is my own vote from last November 2.)

Don't worry about parsing these files. I have done it for you. In a later assignment you will use my parsing code to parse raw data, but for this lab you can use some of the intermediate output of my program.

The one thing you need to know about the nature of the ballot data is this: the candidate name and contest as a text string at the right side of the raw data file is generated data, not embedded data in the vote record stored in the voting machine. In the voting machine, what gets stored (I think) is

- precinct number
- iVotronic number (the voting machine is called an iVotronic)
- B/I number
- sequence number

There can be multiple ballot styles inside a given machine, and each style inside a given machine has a different B/I number. The sequence number is the number assigned to a candidate in a given B/I.

Therefore, in order to uniquely identify a candidate by name, it is necessary to know all four of these numbers.

After some parsing and totalling on my part, I have produced the three "unsorted" files. (You will get code later that will do this, and you will write some code later to produce something like this.)

Note that the "unsorted" files are actually sorted first by precinct number, then by iVotronic number, and then mostly sorted on the fields that follow.

There are also three "sorted" files. These have the same data as the "unsorted" files, but the data is now sorted so that common candidate/contest lines appear together.

Your lab assignment is to create an insertionsort that will read in the unsorted list `zunsorted.305` (it's the smallest of the three) and produce the sorted list `zsorted.305`

For reference, the data in a given line of both the sorted and unsorted lists is:

- precinct number
- iVotronic number (the voting machine is called an iVotronic)
- B/I number
- sequence number
- vote total
- textual candidate/contest information

For example, the first line of data in the unsorted 305 file reads

```
305  5121076  2  10  3  Nikki R Haley GOVERNOR
```

and means that in precinct 305, on iVotronic 5121076, on B/I number 2, Haley was candidate sequence number 10, received three votes, and is labelled `Nikki R Haley GOVERNOR` in the table of candidates.

(What I have done in creating the unsorted list from the raw data is to total up votes in a given precinct, machine, B/I, to produce numbers like the 3 votes totalled.)

Note that the first four items in any row are all numbers, so your `Scanner` can use `nextInt` to read them, and then you can just use `nextLine` to read all the way to the rest of the line. This leaves open the possibility that there may be textual differences in the way names and candidates are written, but you won't have to worry about that for this assignment. If the text is different, you can output results that assume it's different. Note that this does happen in the write-in candidates, whose names begin with the tag label "W/I".

For testing purposes, I have also taken the first 35 lines of the sorted 305 data and created an unsorted version of this, with obvious file names. This way you can test your code on a much smaller input file.

(And you should for amusement value look at the write-in names...)

Insertionsort

You will need an `ArrayList<OneLine>` for each line, and each `OneLine` entry in that will have five instance variables–four integers and a string. I have given you accessors and mutators. You will also need to write a method to read one line of data from a `Scanner` and a `toString` method.

The insertionsort algorithm is this.

1. Read two lines of data into an instance of `OneLine` and add them to the `ArrayList`.
2. Compare the two `String` instance variables of `candidate` and exchange the two entries so they are in sorted order.
3. Now, until you run out of data, read in an instance of `OneLine` and add it to the (end of the) `ArrayList`. This is the outer loop. Then run an inner loop to pull the `OneLine` instance forward through the `ArrayList` until you find the appropriate position for it.

The inner loop is thus exactly the inner loop in Chapter 4 and in the powerpoint. The outer loop is a read of data and an add of the data to the end of the `ArrayList`.

DO NOT WORRY ABOUT MAINTAINING THE REST OF THE LIST IN SUBSORTED ORDER BY MACHINE NUMBER, ETC. You only need to gather all the lines together in blocks by candidate/contest name.

Feel free to steal and reuse driver programs and file utilities programs from previous exercises.

The program you turn in for the lab will use only the 305 data. The other is provided as examples and for later use. Consider this an assignment to try to understand the data without looking at the manual for how it is created. In doing that kind of forensics, having more data means you can get a better idea of what "generally" happens.