Tie Figher Patrols

Kristopher Sewell ©2018

kjs170430 CE2336.002

## Input

We will receive input from a formatted text file. The input document will always have the same filename and will describe the route pilots will take during their patrol. Patrols will always start and stop in the same location. Each “checkpoint” will be seperated by a space. Names will also have a space between them and the start of x,y pairs. X-coordinates and Y-coordinates will be seperated by a comma. Each line will begin with the pilots name. The last line of the file may or may not end with “/r /n” or “/r”. Limits have been imposed on how many pilots (20), and coordinate pairs (16) per pilot are allowed.

### Openfile

To open and process the file we must know several things. Does the file exist? Can it be opened/read (file permissions)? Is the document formatted correctly? The first two are fairly simple by checking with the .exist() and .canRead() methods of File objects. If both are true we can pull info from a scanner object. I will use while (hasNext()) and getNext to grab string info.

### Parse

We must determine if we are looking at a name or a coordinate pair. Names with not contain 0-9, or punctuations as the first character. So we can test the input array of characters first index for 0-9, period and comma. Coordinates may contain varations of numbers, periods and commas but will not contain letters or other symbols.

If the input is a name add the string to the current index of name array. Additional name fields, IE first middle last, should be concatenated onto the string already in the array. Once a coordinate is detected no further writes should be done to the name. \*Should refrain from incrementing counter at this stage, maybe use a boolean to prevent further writes.

Coordinates may have spaces between the comma and number which would cause the pairs to not be in the same strings when getNext() pulls info. Combinations would be x,y x x, y ,y where x and y could represent any integer or floating point number. The easiest case would be if both x and y were seperated from the comma by a space. If the string equals ‘,’ we can ignore the string and get next. If x or y does not contain the other number but contains the comma we can remove the comma from the string and convert to double. If x and y come as a pair with the comma the we should pull just the numbers and period (if it exists) before and after the comma to convert to double.

To check the coordinate we should look for the most common case first x,y without any other spaces. First we should find position of the comma. Then we need to make sure that y is present by checking that comma position + 1 is < str.length. (length returns size excluding zero). Additionally we should check that comma position is not 0. These checks will indicate either ‘x,’ or ‘,y’ and not ‘x,y’. Then we can convert position 0 thru commaPosition -1 as x and commaPosition +1 thru str.length as y. If we are not at coordinate index [pilot][0] then we should check if the current index equals the [0][0] and [0][1] indices. If they equal then we should expect a name for the next getNext call, so reset the coordinate index to 0 and increment the pilot index.

## Calculate

After the file has been parsed into the two arrays we can calculate polygram area for each pilot. By passing the 2D coordinate/value array into an area function we can iterate the sum and calculate the anwser. We can store the value returned in a third array foreach pilot.

## Output

To write the file we should do dilgence to make sure we are not destroying previous data. Checking if the file already exist with the File object is a good way to start. If an append mode exist for PrintWriter then that would be another way to make non-destructive writes. If one does not exist we can present a prompt if the file exist to overwrite. If yes overwrite the file, if no then append/increment an integer to the filename and restart the function with the new filename.

To write the file we just need to pass .println(name[i] + area[i]) to the printWriter object. We can use the “try (new printWriter(File) { output }” format to autoclose the file once everything is written.

### Exit

All that is left is to exit graceful from the main function System.exit(0);