CSE 11 Fall 2020 PA4 - Covid Transmission-2

Due date: Wed, Nov 4th @ 11:59PM PST (Thurs, Nov 5th @ 11:59PM PST with slip day)

No submission will be accepted after Thurs, Nov 5th @ 11:59PM PST

Total Points: 100

(Note: If you submit your assignment late, the autograder will automatically use your slip day if you have any remaining.)

Files to Submit:

`InfectionTracking.java`
In order to receive points for this assignment, your file names **must** match exactly as specified.

Goal:

Programming Assignment 4 is an introduction to one dimensional Arrays in Java. In this PA, you will use Arrays, and many of the other programming techniques you learned from previous PAs to complete the assignment.
Please read the entire write-up before getting started

Background

Disclaimer: These are sample statistics that do not accurately reflect covid transmission. Please follow the CDC guidelines to stay up to date on the coronavirus pandemic: https://www.cdc.gov/

UC San Diego is taking many actions to slow the spread of COVID-19, but they don't have a way of assessing how successful their preventative measures are. Given the skills you have demonstrated so far in CSE 11, they have tasked you with running transmission simulations and gathering data regarding how the virus would spread if all students went about their lives without taking any preventative measures. UCSD will then be able to use your results as a baseline and successfully assess how effective their preventative measures have been.

For this task, you will perform operations based on a file containing important information regarding the students involved in the simulation: their names, their locations at the start of the simulation, their movement patterns, and their COVID-19 infection statuses. We'll model the world as a one-dimensional array, so a student's location is their index in this array and their movement pattern is the number of indices that the student moves per day.

You will then be able to use our understanding of the virus to model how the virus would spread for the given scenario, and track important statistics about that spread.

After being briefed by Chancellor Khosla himself on the overall goal and why you were recruited, he assigns you several tasks to quickly get started with the problem at hand.

Part 1: Reading the Input File and Populating Arrays

You are asked to bring each student's information into your program from a file to create 4 **Parallel Arrays** describing the students involved in the simulation. Parallel Arrays are essentially multiple arrays of the same size such that the i-th element across all of the arrays when combined represent one object, which is a student in our case.

This file, if it exists, will have the following format:

- Each line of the file will represent a single student
- Each line is comma-separated and has the format `Name,Location,Movement,Infection`.
  - `Name` will be the student's first name and is a `String` (the file will not contain quotation marks though).
  - `Location` describes where the student currently is in our one dimensional world and is an `int`.
  - `Movement` describes the amount of distance the student will move each day.
  - `Infection` is an `int` with value `0` if the student does not have COVID-19 and `1` otherwise.
- You can assume that the file is formatted properly and follows the following format, each line will always be a `String` with no spaces or symbols, a comma (`','`), an `int`-parsable number, a comma (`','`), an `int`-parsable number, a comma (`','`), and either `0` or `1`.

Throughout this assignment, we will be referencing specific variables by their name in each method.

- `names` represents the array of student names.
- `locations` represents the array of student locations in the 1-dimensional world.
- `movements` represents the movement amounts of students.
- `infections` represents the infection status of students.
- `worldSize` (which will be introduced in Part 2) represents the size/length of the 1-dimensional world. Note that `worldSize` can be larger than the length of the parallel arrays.

Furthermore, we will **always** need to do validity checks for our inputs. Some methods will have extra checks needed, but for every method, the following constitute invalid input:

- Any of the input arrays (`names`, `locations`, `movements`, `infections`) does not match any of the others in length.
- Any of the input arrays is `null`.
- Any of the values in the input array `infections` is not either `0` or `1`.
- Any of the values in the input array `locations` is not in the range [0, `worldSize`-1]. When `worldSize` is not provided as an argument, there is no upper bound on the values in the input array `locations`.
- Non-positive input `worldSize`.

TODO: Method to Implement for Parsing Input

```
public static int populateArrays(String pathToFile, String[] names, int[]
locations, int[] movements, int[] infections) throws IOException;
```

`public static int populateArrays(String pathToFile, String[] names, int[] locations, int[] movements, int[] infections) throws IOException`

- Read from the input file found at pathToFile, and parse its data into the respective parallel arrays.
- The student order (index in the array) should be as they are found in the file.
- The four arrays (`names`, `locations`, `movements`, and `infections`) will be passed in as parameters. Populate these arrays directly.
- If the arrays are populated successfully, return the maximum location value of the students + 1, so if the largest location value is 315 we return 316, else return `-1`. Note if the file IO results in an exception, it is acceptable to not return -1 and throw an `IOException` instead.
  - The arrays cannot be populated successfully if the file at `pathToFile` is inaccessible or does not exist or if `pathToFile` is `null`.
  - The arrays also cannot be populated successfully if any of their lengths does not exactly match the number of students in the file or if any array is `null`.
  - In any case where you would return `-1`, you do not need to worry about restoring the input arrays to their original states.

Example of populateArrays()

Suppose the file `Students.csv` has the following content:

```
Greg,5,2,1
Paul,2,-3,0
Bob,1,1,1
Doug,0,-2,0
```

If `names` is a `String` array of size 4 and each of `locations`,`movements`, and `infections` is a separate `int` array of size 4, then a call to `populateArrays("Students.csv", names, locations, movements, infections)` should return the value `6` (largest location value + 1) and the arrays after the call should have the elements:
`names = ["Greg", "Paul", "Bob", "Doug"]`
`locations = [5, 2, 1, 0]`
`infections = [1, 0, 1, 0]`
`movements = [2, -3, 1, -2]`

All of Greg's information is at index 0, all of Paul's information is at index 1, etc.

Another Example of populateArrays()

A file containing

```
Jim,7,3,0
Greg,5,2,1
Sally,1,-2,0
Paul,1,-5,1
Mary,8,4,0
Fred,5,3,0
Sara,1,1,1
Sam,3,5,0
Rob,9,-6,0
```

will populate the arrays to look like

Name:

"Jim"	"Greg"	"Sally"	"Paul"	"Mary"	"Fred"	"Sara"	"Sam"	"Rob"

Location:

7	5	1	1	8	5	1	3	9

Movement:

3	2	-2	-5	4	3	1	5	-6

Infection:

0	1	0	1	0	0	1	0	0

Actual World:

	"Sally" "Paul" "Sara"	"Mary"	"Sam"		"Greg" "Fred"		"Jim"		"Rob"

Note that the "Actual World" can be larger in length than the Parallel Arrays

Part 2: Update Locations and Infections

Now that you have a convenient way of representing the students in your program, Chancellor Khosla wants you to simulate students moving around and how infections spread when students come into contact with another student infected with COVID-19.

TODO: Methods to Implement for Movement and Infection Spreading

```
public static void updateLocations(int worldSize, int[] locations, int[]
movements);
public static int[] updateInfections(int worldSize, int[] infections, int[]
locations);
```

`public static void updateLocations(int worldSize, int[] locations, int[] movements)`

- Given the locations and movements arrays, which hold the students' current locations and their movement values, move all students' locations by their respective movement amount.
- An update to a student at index i should be made as follows:
  - $locations_i = locations_i + movements_i$
  - if $locations_i + movements_i < 0$ or $locations_i + movements_i$ >= `worldSize` (the length of the world we are modeling), use the modulo operator (%) to wrap around in the arrays. For example, if the world is 20 units long (`worldSize` is `20`) and a student has location 2 and their movement value -5, their updated location should be 17. Note that in Java, the result of a%b is not always non-negative (it will depend on the sign of the operand). You will need to account for this difference when dealing with negative locations.
  - Important edge cases to test include:

        - $locations_i + movements_i$ >= worldSize
        - $locations_i + movements_i$ < 0
        - $|locations_i + movements_i|$ > worldSize when $locations_i + movements_i$ < 0.
- Do not change any values in `movements`.
- On invalid inputs, do not change any values in `locations`, i.e., this method should return without doing anything.
- Invalid inputs include the normal invalid inputs mentioned in Part 1.

Example Input:

`worldSize = 4`

`locations = [3, 2, 1, 0]`

`movements = [2, -3, 1, -2]`

Updated `locations` Array:

`locations = [1, 3, 2, 2]`

`public static int[] updateInfections (int worldSize, int[] infections, int[] locations)`

- This function will serve two purposes. Firstly, it will update the infection status of students who come in contact with infected students. Secondly, it will create and return a new array that stores the number of infections caused by each of the students.
- Updating Infection Status:
    - Given the `infections` array and the `locations` array, which store each student's infection status and current location respectively, update every student's infection status.
    - A student's infection value should be updated from 0 -> 1 if and only if they currently share a location with one or more students who have an infection value of 1.
    - Students who have an infection value of 1 will not experience an infection value change in this method.
    - These updates should be directly made to the `infections` array that is passed in.
- Create numStudentsInfected array:
    - Create a new array which stores the number of infections caused by each student. This array should follow the same format as the parallel arrays, so the number of infections caused by student 0 should be stored in `numStudentsInfected[0]`.
    - The number of people an infected student will infect is equal to the number of non-infected individuals this student shares a location with.
    - A non-infected student will have a `numStudentsInfected` value of 0.
    - If there are multiple infected students sharing the same location with non-infected students, all infected students will have their numStudentsInfected values increased to equal the number of non-infected students they share a location with.
- Since we know the worldSize, make sure that any positive value in `locations` is in the bounds of the world.
- Do not make any changes to the values in the `locations` array.
- On invalid inputs, return `null` (there is no need to restore the original state of the inputs).

Example Input:

`worldSize = 6`

`infections = [0, 0, 1, 1, 0, 0, 1, 0, 1]`

`locations = [2, 1, 4, 3, 3, 5, 1, 1, 3]`

All students on locations 1, 3, and 4 must have their infection status updated to 1 as there are infected students on these locations. In this example, only locations 1 and 3 have both infected and non infected students, so we must update those students accordingly.

Updated infections Array:

`[0, 1, 1, 1, 1, 0, 1, 1, 1]`

Returned numStudentsInfected Array:

`[0, 0, 0, 1, 0, 0, 2, 0, 1]`

The infected student on location 1 will have infected two individuals as they share location 1 with two non-infected students. The infected students at location 3 share the location with one non-infected student, so their numStudentsInfected values will both be one.

Part 3: Simulate Spread

Chancellor Khosla is really pleased with the progress you've made so far, and he's ready for you to fully simulate how COVID-19 will spread on campus when students go about their daily lives. He wants you to write a method that can simulate student movement and infections, and count the number of infections each student causes, as he believes this will provide powerful insight into the virus's spread.

TODO: Methods to Implement for Simulation

```java
public static int[] countInfectionsByStudent(int days, int worldSize, int[]
locations, int[] movements, int[] infections);
```

`public static int[] countInfectionsByStudent(int days, int worldSize, int[] locations, int[] movements, int[] infections)`

- Each day, all students will move once by their movement amount (as defined in and by calling `updateLocations()`) and infect or get infected by whoever comes in contact with them at their *new* location (as defined in and by calling `updateInfections()`. Note that at the beginning, even if an uninfected student is at the same location as an infected one, the uninfected student will not be infected (infections only happen after the first movement step).
- This method should update `locations` and `infections` according to the simulation and the `days` that the simulation runs for.
- This method should not update `movements`.
- This method should additionally return an array parallel to the `locations` (and `movements` and `infections`) array where each element represents the total number of other students the student infected (as defined in `updateInfections()`) for the whole simulation.
- If there are any invalid inputs, return `null`. Invalid inputs include the normal invalid inputs mentioned in Part 1 but also additionally include `days` being negative.

Example Input:

`int days = 3`

`int worldSize = 10`

`locations = [3, 2, 1, 0, 5, 2, 9]`

`movements = [2, -3, 1, -2, 3, 2, 5]`

`infections = [0, 1, 0, 0, 0, 0, 1]`

Return:

`infectionRecord = [0, 1, 0, 2, 0, 1, 3]`

Part 4: Analyze Results

The UC San Diego administration is really happy with the work you've done so far, and is excited to see how effective their preventative measures have been when compared to your model. They have two final tasks for you, one of which is to analyze the results a little more deeply, to gain a better understanding of what took place, and the other of which is to be able to interact with the program from the command line.

TODO: Methods to Implement To Analyze Results and Interact with Program

```java
public static int findRNaught(int[] infectionRecord);
public static String findSuperSpreader(int[] infectionRecord, String[] names);
```

Suggested: Method to Implement to Test Program

```java
public static void main(String[]args);
```

`public static int findRNaught(int[] infectionRecord)`

- R_0 or RNaught tells you the average number of people who will contract a contagious disease from one person with that disease.
- Given the `infectionRecord` array, which gives us details regarding how many infections were caused by each student (in the format of the returned value from `countInfectionsByStudent()`), find the average infections per student. Return the smallest integer greater than or equal to this value (the ceiling of the value)
- On invalid inputs, return `-1`. Invalid inputs include:
  - `infectionRecord` being `null` or length `0`.
  - Any value in `infectionRecord` being negative.

Example Input:

`infectionRecord = [2, 1, 0, 4, 7, 3, 1]`

Return Value:

`3`

`public static String findSuperSpreader(int[] infectionRecord, String[] names)`

- This method will take in the `infectionRecord` array, which gives us details regarding how many infections were caused by each student (in the format of the returned value from `countInfectionsByStudent()`), and the `names` array which holds the first names of the students, to return the name of the student who caused the most amount of infections.
- If there is a tie among multiple students, return the student who appears first in the names array.
- On invalid inputs, return `null`. Invalid inputs include:
  - `infectionRecord` and/or `names` being `null` or length `0`.
  - `infectionRecord` and `names` have different lengths.
  - Any value in `infectionRecord` being negative.

Example Input:

`infectionRecord = [2, 1, 0, 4, 7, 3, 1]`
`names = ["Alice", "Bob", "Mary", "Paul", "Steven", "Greg", "Charles"]`

Return Value:

`Steven`

`public static void main(String[]args)`

This method is only a suggestion to help you in debugging your code, we will not be grading your main method

- You will need a main method to test the functionality of your methods.
- Your main function should read the command line inputs (passed in as `args`), which will be of the form "pathToFile numberOfDays numberOfStudents", and call the respective functions with the necessary inputs. The order of the command line arguments should be the `pathTofile`, the `days` (to run the simulation), and `numberOfStudents` (the number of students we expect the file at `pathToFile` to contain information for).
- Initialize the four parallel arrays with length `numberOfStudents`.
- Once you have parsed the command line inputs, and have initialized your arrays, you can test your methods and make sure you have accounted for all of the edge cases.
- This method will not be graded, but you will need to write a main method to conduct your own testing for this PA.

Style

Coding style is an important part of ensuring readability and maintainability of your code. We will grade your code style in `InfectionTracking.java` thoroughly according to the style guidelines. Namely, there are a few things you must have in each file / class / method:

1. File header(s)
2. Class header(s)

3. Method headers
4. Inline comments
5. Proper indentation (do not intermingle spaces and tabs for indentation)
6. Descriptive variable names
7. No magic numbers
8. Reasonably short methods (if you have implemented each method according to specification in this write-up, you're fine). This is not enforced as strictly.
9. Lines shorter than 80 characters (note, tabs will be counted as 4 characters toward this limit. It is a good idea to set your tab width/size to be 4)
10. Javadoc conventions (@param, @return tags, /** header comments */, etc.)

A full style guide can be found here. If you need any clarifications, feel free to ask on Piazza.

README

All the questions for the README portion of this assignment can be found on Gradescope. Note that this portion of the assignment *cannot* be submitted late.

Survey (1 point):

We ask that you spend a few minutes reflecting on your experience in this course this week. Please give thoughtful and truthful answers, to the best of your ability. Your specific answers will not affect your grade in any way. You will receive credit as long as you complete it.

- Weekly Reflection 4

Submission

Turning in your code

Submit the following file to Gradescope by **Wednesday, Nov 4 @ 11:59PM PDT** (Thursday, Nov 5 @ 11:59PM PDT w/ slip day):

- InfectionTracking.java

When submitting, please wait until the autograder finishes running and read the output. **Your code must compile in the autograder in order to receive proper partial credit.**

Evaluation

1. **Correctness** (83 points)
   You will earn points based on the autograder tests that your code passes. If the autograder tests are not able to run (e.g., your code does not compile or it does not match the specifications in this writeup), you may not earn credit.
2. **Coding Style** (10 points)
3. **README** (6 points)
4. **Weekly Reflection Survey** (1 point)