COMPUTER SCIENCE 12B (FALL, 2020)
ADVANCED PROGRAMMING TECHNIQUES IN JAVA

PROGRAMMING ASSIGNMENT 8

Program Description

This assignment will provide practice with `Hashmap`, `Hashset`, and `ArrayList`.

Detecting Fraudulent TA Timesheet Submissions

Background

At the Moon's Computer Science Bootcamp, they rely heavily on teaching assistants (TAs) to ensure that all of the students get the help needed to succeed in the field of computer science. Because of the large number of students, they need the TAs to be able to help each student as quickly and as efficiently as possible. To ensure that the TAs are being helpful and fast at the same time, Moon's Bootcamp requires that the TAs submit a timesheet entry for each student they help. Recently, after looking at the TA timesheet submissions, they suspect that some of the TAs are under-reporting the amount of time taken to help a student. Moon's Bootcamp doesn't want the TAs to feel as if they are being evaluated by how fast they help each student – but it's an important metric to understand for planning purposes. Moon's Bootcamp needs your help to automatically identify similar cases of "fraudulent" reports, to make sure the students receive adequate help without wasting funds.

Billing & Reporting Process

Moon's Computer Science Bootcamp has separate systems for job tracking and billing:

- TAs indicate when they start helping a student by submitting a job start notice to the job tracking system.
- At any time while helping a student, the TA enters a single invoice into the billing system. The billing system then gives back a unique, increasing numeric invoice ID.
- Upon completion, TAs submit their invoice IDs to job tracking, which records that the jobs were completed. A TA must complete and submit all open jobs together (other students could've showed up while a TA is helping another student, and they must finish helping all students present before submitting their invoice IDs)

Input

You will receive strings in the form of .txt files. Each string will be on its own line and will correspond to one event each. These events are already sorted in terms of time, and take one of two forms:
1. Job start events take the form <TA_NAME>;START

- The TA_NAME is a unique identifier for the TA, and is guaranteed not to contain a semicolon.

2. Job completion events take the form
<CONTRACTOR_NAME>;<INVOICE_ID>(,<INVOICE_ID>)*

- The TA_NAME is the same unique identifier for the TA as before.
- INVOICE_ID 's are integer values, guaranteed to fit within the value of an integer
- If a TA has multiple job started, then they will complete and submit invoice IDs for all started jobs as a single job completion event. These invoice IDs will be comma-delimited, one invoice ID per job start event. These are referred to as "batch job completions". For example, if a TA has started helping three students, then the next job completion from that TA is guaranteed to consist of three distinct invoice IDs. We do not know which invoice number corresponds to which job start event. You may assume that the input will always be well-formed and contain no extra characters or whitespace.

Detecting Fraud

There are two kinds of fraudulent submissions you need to detect: shortened jobs and suspicious batches. It is recommended that you first work on detecting shortened jobs before tackling suspicious batches. In addition, you should also be able to detect unstarted jobs, which are less serious than a fraudulent submission.

Unstarted Jobs

These are job finish events that have never been started in the first place. For example, consider the following event log:

```
Kelly;START
Theo;START
Theo;10
Kelly;12
Danny;20
```

In the above case, Danny could not have submitted an invoice ID if he never submitted a job start notice.

The output should be:

```
5;Danny;UNSTARTED_JOB
```

Shortened Jobs

These are job start events which, given their later submitted invoice ID, must have been submitted artificially late -- that is, jobs with an invoice ID smaller than any invoice ID submitted before their start. For example, consider this event log:

```
Nick;START
Nick;24
Dan;START
Dan;18
```

According to the log, first Nick starts a job, gets an invoice ID of 24, and then submits a job completion with invoice ID 24. Dan later does the same, with a invoice ID of 18. However, Dan could not have started his job after Nick finished his, because Dan's smaller invoice ID implies he must have gotten his invoice ID before Nick, and one cannot obtain invoice IDs before the job starts. Therefore, Dan's job start event on the third line is a shortened job.

The output should be:

```
3;Dan;SHORTENED_JOB
```

Suspicious Batches

These are batch job completions which, given their invoice IDs and their associated start events, must contain at least one shortened job. For example, consider this event log:

```
Leah;START
Leah;10
Alice;START
Alice;START
Alice;8,14
```

According to the work log, first Leah started and finished helping a student with an invoice ID of 10. Then Alice starts 2 jobs, gets invoice IDs 8 and 14 for the two jobs, and finally submits a batch job completion with those invoice IDs. Except Alice must have gotten an invoice for one of her jobs before Leah, because Alice’s invoice ID of 8 is smaller than Leah’s invoice ID of 10. Therefore, Alice's batch job completion on the fifth line is a suspicious batch, because at least one of the start events should have been before Leah's job completion. We can't identify which of Alice's two job start events is the one that was shortened, so we cannot mark either of them as shortened jobs. If both IDs were below 10, then we would mark both start events as shortened jobs, NOT as a suspicious batch.

The output should be:

```
5;Alice;SUSPICIOUS_BATCH
```

Output

Your algorithm should return a string array that indicates the set of submissions that are possibly fraudulent. For each suspicious behavior identified, the string array you return should contain a line of the form <LINE_NUMBER>;<TA_NAME>;<VIOLATION_TYPE>.

- LINE_NUMBER is a (one-indexed) line number from the input.
  - For SUSPICIOUS_BATCH, this is the line on which the batch job completion occurred.
  - For SHORTENED_JOB, this is the line on which the job start occurred.
  - For UNSTARTED_JOB, this is the line on which the job completion occurred.

- TA_NAME is the unique identifier of the offending TA.
- VIOLATION_TYPE is a string that matches either SHORTENED_JOB, SUSPICIOUS_BATCH, or UNSARTED_JOB. This indicates which pattern we have identified, following the rules above. You may present violations in any order.
- There are four possible outputs for any test case, In order to pass the test, make sure the output syntax match the PDF exactly.
  No output: when every TA reports correct hours
  SHORTENED_JOB
  SUSPICIOUS_BATCH
  UNSTARTED_JOB

You must use the following prompt to ask the user for a work log:

```
Enter a work log:
testCase3.txt
3;Evil;SHORTENED_JOB
```

Implementation Details

Before you attempt to modify the given .java files, take some time to think of how you would go about solving the problem first.

TARecord

This class is used to store start and end records (with their associated invoice IDs) for an individual TA after reading in the .txt file passed in. This class has been provided for you with a simple constructor that takes in a String, and constructs a new TARecord for the specific TA whose name is passed in. This class is incomplete - you will have to do the following:

1. Determine how best to store each start and end record (which data structure(s) to use) - keep in mind that you will have to retrieve those records later in order to determine whether fraud has been committed.

TAChecker

This is the main class that checks all the TA worklogs for instances of “fraud”. You can treat this as a simulator for your code. Below are the methods that you are to implement; make sure you name the methods exactly as specified:

1. void sortWorkLog() - this method scans through the .txt file and creates a TARecord for each unique TA
2. void checkValidity() - this method should go through each TARecord (essentially go through each TA) and detect whether the TA has committed fraud. If fraud has been committed, a message should be printed out. The algorithm you devise should handle all four of the possible cases that could occur.

Submission

Create a folder containing your Java source code named FirstLast_PA8. Compress (zip) the folder and upload it to Latte by the day it is due. For late policy check the syllabus.

Grading

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines linked in LATTE. Also, remember to include the comment header at the beginning of your program.