Playing With Logging

1 Background

The goal of this lab is to see how logging can be used for an application that runs with little/no UI. The situation is this: We have a file that has sequential data that we have read from three sensors. The first three lines of the file have two integers that are the min and max expected values for each of the three sensors. The rest of the file contains readings from those three sensors. One reading is taken every four minutes and the readings are grouped by hours. Each reading is stored as one line in the file. That line contains a single character that is the time slot ID followed by three integer values. The timeslots IDs start at 'A' and will reset to 'A' every hour.

1.1 Things that Could Help

Here are a few clues for things you might need:

Scanner Java gives us the Scanner class which we can use to read files.

- You can create one with `Scanner inputFile = new Scanner("fileName.txt");` using the correct file name, of course. If you put the file in the root folder of your project, you don't need to specify any path information.
- You should research the getNextInt() and getNext() methods.
- There's no way to back up in the file. If you need a previous record, you'll have to hold it yourself.

String splitting If you have a variable x that contains a string, you can say

```
String[] parts = x.split("␣");
```

(that u-like thing denotes a space) to tear it into substrings at the spaces. For example, if x is "This is a string", then parts will be an array containing these strings: "This","is","a","string" in that order.

You are free Even though I'm giving you some code to start, you can create any other methods that you want to.

2 What Could Go Wrong

These files have been around for a while and have been transmitted over unreliable networks a number of times, so they aren't really pretty. We want to build a reader that is robust (doesn't crash), reliable (gives the best information it can), and informative (tells us what it has done to recover from errors). Because expect the programs using this data to run

without a UI, we want to log the necessary information so people can look at it later if they are interested.

Note: these files could be VERY long, so we need to add two rules:

1. You cannot read the whole file into memory - it might be too big. You have to process each line as you read it
2. You can't use recursion to get to the next line. Since the file could be big, that could make your runtime stack exceed the space it is given.

2.1 Types of Errors to Expect

You can rely on the first three lines not having any errors. For the rest of the file (the lines containing readings), the types of errors we expect fall into two categories: formatting errors and bad values.

2.1.1 Formatting errors:

- An incorrect number of values in a reading.
- Any incorrect characters in a reading.
- Any value more than 150% of the max value that reading is supposed to report

For formatting errors, we are going to make a "Severe" entry in the log and throw the record away.

2.1.2 Bad value errors:

- If any of the values is out of range, substitute the extreme value closest to the reading and make an informational entry in the log
- The timeSlotID should be a repeating pattern that is 15 readings long. If it isn't the expected value, determine if there is a row missing.
  - If a single row is missing, return all of the values in the previous row with the timeSlotID column corrected and hold this row to return in the next request.
  - If the timeslotID is wrong by more than one position, make a severe entry in the log and go forward as if that entry is correct.
  - Since the logic for determining what has occurred is nontrivial, make debug entries (using log level "Finest") in the log to document the choice and the queueing of the row if only one row was missing

3 Logs Don't Just Record Bad Things

We don't just log bad things. We can log things that are interesting or just activity. For example, many systems log when people log in and log out so that they can infer who was online at any given moment. Think about something interesting people might want to log from this data.

4 What I'm Giving You To Start With

To get you started, I'm giving you a bunch of files. They are in the workspace in the class repository. Here's a summary:

ReadingSet.java Holds one legal set of data (one row of the file) - nothing for you to change here

SensorReadingParser.java This is the class that you have to make work. I've given you a framework that contains these pieces:

- All of the public methods you need (returning default values for now)
- The logger is already set up. Here's what you know about it:
  - It's name is logger (go figure!)
  - It has a FileHandler attached to it, so it will write to that file.
  - The name of the file that it will create is the name of the file being read with ".log" tacked onto the end.
  - It will log everything that is at least as interesting as informational log items. That's what you need to make the tests pass. If you want to look at the debugging log items in the file, you'll need to change that to Level.FINEST.
  - All log entries will also be output to the console.

BasicLogParser This is used by the tests to verify that you have the write types of entries in your log - you don't have to do anything to this file.

ReadingSetTests ReadingSet was pretty trivial, so it's tests are, too. Nothing for you to worry about here.

SensorReadingParserTests This is the tests that the auto grader will run. DO NOT CHANGE THEM! (you can mark them to be ignored while you are working, but, in the end, you want my tests to pass)

TestFiles This folder contains the data files that are used by the tests. You can look at them to see what is happening, but DO NOT CHANGE THEM! My autograder will use the files as they were given to you, so you want to make things work without changing the data.

5 Resources

So, basically, your job is to get all of the tests to pass. In addition to parsing the file, you'll have to figure out how to make appropriate log entries. We are using the logger that is built into Java (java.util.logging.logger). Here's a good tutorial to help: https://www.vogella.com/tutorials/Logging/article.html

Clearly, you have a bunch of my code. If you have questions about it, or about this lab, you can find me in all of the usual places: email, Discord, etc.

I've also created a channel in our Discord class where you all can ask questions. It's a good idea to ask there because, someone else may be able to answer faster than I can, and, that way everyone gets to see my answer, too. I promise to keep an eye on the channel most of the time . . .