420-406-AB Data Structures
Winter 2020

January 27th
Worth 4%

Assignment I: Streams

(Due on Check on Léa)

In this assignment, you will use Java's stream IO classes to implement a *merge* of two datasets stored in secondary storage and store the resulting dataset in a third file.

1 Merge

A merge takes two *sorted* datasets and combines them, keeping the resulting data sorted. It can be implemented efficiently without the need to use a sorting algorithm on the entire output set.

For example, a merge of two number streams: [1 4 6 7] and [2 3 5 8 9 10] would be:

input 1	input 2	output
[1 4 6 7]	[2 3 5 8 9 10]	.
[4 6 7]	[2 3 5 8 9 10]	[1]
[4 6 7]	[3 5 8 9 10]	[1 2]
[4 6 7]	[5 8 9 10]	[1 2 3]
[6 7]	[5 8 9 10]	[1 2 3 4]
[6 7]	[8 9 10]	[1 2 3 4 5]
[7]	[8 9 10]	[1 2 3 4 5 6]
.	[8 9 10]	[1 2 3 4 5 6 7]
.	[9 10]	[1 2 3 4 5 6 7 8]
.	[10]	[1 2 3 4 5 6 7 8 9]
.	.	[1 2 3 4 5 6 7 8 9 10]

2 Log Files

The input files contain log entries of network communication. Each entry includes the timestamp (date and time) of the communication, the IP address of the requesting machine, the service name of the request and the length of the communication in number of bytes.

2.1 Log entries

The log entries are in a whitespace delimited format. Each line contains the following information:

1. IP address,
2. network service name,
3. timestamp, in the format `yyyy-MM-dd hh:mm:ss.SSS`,
4. length (in bytes).

To match whitespace use this regex: `"\\s+"`

2.2 Comparing

These log files are ordered by the following criteria:

- first by IP address, numerically ascending,
- second by service name, alphabetically ascending,
- lastly by timestamp, chronologically ascending.

For example:

```
198.2.186.109  http 2018-01-28T04:39:31.965 1095
198.2.186.109  http 2018-01-28T04:39:30.607 1556
203.112.93.219 http 2018-01-28T04:39:31.512 1202
203.112.93.219 http 2018-01-28T04:39:32.503 3203
203.112.93.219 ssh  2018-01-28T04:39:30.850 4093
```

2.3 Representing Log Entries

The provided class `Log` contains all the fields necessary to store log entries. Implement the following constructor and methods:

Signature	`Log(String line)`
Description	Create a log using the line information.
Preconditions	The line is in the format described in Section 2.1.
Postconditions	The log is created, otherwise a `ParseException` is thrown.

Signature	`toString() : String`
Description	Generate a string representation of the log.
Preconditions	None.
Postconditions	A string representation is returned in the format described in Section 2.1.

Signature	`compareTo(Log rhs) : int`
Description	Compare the current log with the log `rhs`
Preconditions	None.
Postconditions	Returns a negative integer if this log comes before the log `rhs` in the sorted order, returns a positive integer if this log comes after the log `rhs` in the sorted order, or else returns 0 (log the same). Use the sorting order described in Section 2.2.

3 Merge implementation

Using Java's IO classes, merge two log files into an output log file. Implement the merge *without loading the entire file contents into memory*. Assume that the two input log files are sorted (according to the order defined in Section 2.2). The resulting merged output log file will also be sorted.

Implement the merge operation as the following *static* method:

```
/**
 * Merges the files `in1` and `in2` into `out`, maintaining the sorted order.
 * @param in1 the first sorted input file.
 * @param in2 the second sorted input file.
 * @param out the destination of the merged files.
 * @throws IOException
 */
public static void merge(String in1, String in2, String out) throws IOException { .. }
```

4 Error handling

If any log entry in the input files is not parseable, then report the error to `System.err` and abort the merge. Write the following information:

```
Error: <message> (File: <file>, Line: <lineNo>)
```

with the fields `<message>`, `<file>` and `<lineNo>` filled out. The message should . Ex:

```
Error: Bad IP address 10.39.246.foo (File: 'in1.txt', Line: 50)
```

5 Java Library Classes

In your program, you should make use of the following classes:

- `String` as seen in class.
  `https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/String.html`
- `Scanner` as seen in class.
  `https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Scanner.html`
- `PrintWriter` as seen in class.
  `https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/io/PrintWriter.html`
- `Date`. Datatype for dates that stores both date and time. Use the formatter to create instances of these.
  `https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Date.html`
- `SimpleDateFormat`. You can use this class to convert from `String` to `Date` and back. It uses a date format string. For example, here is how you would parse a date in month/day/year:

  ```
  SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
  Date date = formatter.parse("1/29/2018");
  ```

  `https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/text/SimpleDateFormat.html`

6 Sample Data

I've included some sample data in the `data/` folder. Each subfolder contains two files `in1.txt` and `in2.txt` that you should merge into `out.txt`.

7 Requirements

Your program *must* meet the following requirements:

1. Your program should be clear and well commented. It must follow the "420-406 Style Checklist" (in assignments directory on the GitHub repository).
2. Use the provided classes `Log` and `IPAddress`. The `Log` must implement the API methods defined in Section 2.3.
3. When merging, do not read the entire contents of the log files file into memory.
4. Keep all data files in the `data` folder of your project.
5. Optional: include a simple `main` method test your merge. Ex:

```
public static void main(..) {
  merge("data/test1/in1.txt", "data/test1/in2.txt", "data/test1/out.txt");
}
```

6. Submit using Git by following the instructions (in assignments directory on the GitHub repository).