BIL 212:Summer 2021

Assignment 1

Assigned on Friday June 4. You should upload your program to uzak.etu.edu.tr by Friday, June 18 (any time up to midnight). When uploading do not compress your source directory, upload each file in your source directory separately, and do NOT use packages. Remember that your code should be fully documented. I once again remind you to read the academic honesty policy stated in the syllabus.

Overview:

In this assignment, you will implement an Earthquake Notification System. Your program will receive input from two input files (In real life these would have been realtime data streams, but you will implement a discrete time simulation using static files.) The first input file provides information on the time, location, and magnitude of earthquakes around the world. The second input file will be a series of requests by the users of the notification system. The users may request adding a new watcher to the system, removing a watcher, or query for earthquakes. The program can be viewed as a discrete event simulation, where events could be either earthquakes or user requests; events take place at discrete times, denoted by their timestamps; and are processed in the order that they occur. You may assume that each file contains the events in increasing time order (earliest event first). You may assume the timestamp given is the hours passed since the start of the time (time=0).

You can implement the discrete event simulation in either of the following two equivalent ways: (1) Assume that your simulation starts at time 0 (initialize time to 0). Your program should run in a loop, each iteration of which corresponds to a time slice of 1 hour. At each iteration, the simulation checks if there is an event at that time (note that events are read from two separate files), and if there is, it processes the event, otherwise it proceeds to the next iteration. (2) Simulation starts at time 0, but this time your program increments time to the next event timestamp rather than incrementing by 1.

IMPORTANT: For this assignment, you may ONLY use the Java code you have written or Java classes provided on our Piazza website. You are NOT allowed to use any of Java's built-in data structures such as Java's ArrayList, LinkedList or Stack classes.

Input Files:

- **earthquake-file:**The input file for the earthquakes is in the following format. Information in each earthquake is within earthquake tags. Each earthquake has an id number, time, location description, coordinates given as latitude, longitude and depth, and the magnitude.

```
<earthquake>
    <id>  001 </id>
    <time>  6   </time>
    <place> 4km East of San Francisco, CA </place>
    <coordinates>  -115.5808, 33.0187,  9.5 </coordinates>
    <magnitude> 3.971428571428571 </magnitude>
</earthquake>
<earthquake>
    <id>  002 </id>
    <time>  15  </time>
    <place> 21km SE of Mammoth Lakes, California </place>
    <coordinates>  -118.8353, 37.493, 1.9   </coordinates>
    <magnitude> 3.457142857142857 </magnitude>
</earthquake>
```

- **watcher-file:** The inputfile for the requests may contain one directive per line. The line starts with the timestamp of the request event. It is followed by one of *add, delete, query-largest* requests. The add request will be followed by two numbers denoting latitude and longitude of the user's location. Finally the line will end with the watcher's name. An example is below. The delete request will only be followed by the watcher's name.

  ```
  0 add -105.7 -24.3 Tom
  1 add 21.2 -38.6 Jane
  3 query-largest
  4 add -11.0 63.1 Taylor
  5 add -79.2 37.3 John
  6 add -125.1 -38.5 Henry
  8 delete Taylor
  10 query-largest
  ```

Data Structures:

For this program you will implement three data structures:

(1) The first is a doubly linked list of watchers who have registered for earthquake notifications. You will store watchers on the list in the order that they arrive, and all processing to delete watchers or generate notifications will be done by sequentially going through the list. Let's call this list watcher-list.

(2) The second data structure is a linked list based queue to store the list of recent earthquake records, in order of arrival. You will only be maintaining earthquake records for the most recent 6 hours time period. As simulation time moves on, you will delete records with old timestamps from the queue. Earthquake records are stored in order of arrival time. Let's call this queue the earthquake-queue.

(3) The third data structure also stores these same earthquake records (again, removing earthquakes records that are more than 6 hours old). This data structure is an arraylist **ordered by earthquake magnitude.** Let's call this list magnitude-ordered-list. One of the queries in the request input file is for the biggest earthquake seen in the past 6 hours. This query is answered by looking in the magnitude-ordered-list.

As earthquake records arrive or expire, they are added to or removed from both the queue and the magnitude-ordered-list. Since the queue is ordered by time, it is easy to find which records are old. But you will also need to have an efficient way to find those records in the magnitude-ordered-list. It is not sufficient to do a sequential search through the magnitude-ordered-list to find the records. Since an array for the magnitude-ordered-list needs to be allocated at the start of the program, you will use a size of 1000. We will never test with more than 1000 unique earthquakes.

Event Simulation: Your two input files (the earthquake records and the user requests) each have timestamped records. As the time progresses in your discrete time simulation, you will process the various events.

- **User Events:** When you process an *add* request, a new watcher with the given information is added to the watcher-list (and you print a suitable message to standard output such as `''Tom is added to the watchers list''`. When you process a new watcher *delete* request, the watcher is removed from the watch list (and you print a suitable message to standard output). When you process a new *query-largest* request, you will output a copy of the information for the largest earthquake currently in the magnitude-ordered-list.
- **Earthquake Events:** A new earthquake is added to the rear of the earthquake-queue. Also, you must add the new earthquake record to the magnitude-ordered-list, with its position determined by the magnitude of the earthquake. Print a message as `''Earthquake 60km E of Cape Yakataga, Alaska is inserted into the magnitude-ordered-list''` when a new earthquake happens. Finally, you need to check the timestamp for the oldest earthquake(s), and remove any that are more than six hours old. To do this, you will look at the front of the queue. For any earthquakes that must be removed, you will need to find their position in the magnitude-ordered-list. You are NOT allowed to do this using sequential search of the magnitude-ordered-list. Instead, you must store and maintain in the earthquake record the magnitude-ordered-list position (index). All magnitude-ordered-list update methods must update this magnitude-ordered-list position appropriately when records move around within the magnitude-ordered-list's underlying array. In addition to updating the earhquake-queue and the magnitude-ordered-list, you will also check against the watcher-list when you see a new earthquake. You will output notification messages for each watcher that is "close enough" to the earthquake. Each watcher record contains an (latitude, longitude) position, as does the earthquake record. A watcher should receive a notification message if it is within a certain distance of the earthquake, as defined by the following formula: $Distance < 2.Magnitude^3$. A notification is a suitable message sent to standard output such as `''Earthquake 19.0km from WSW of Mahakanadarawa, Sri Lanka is close to Tom''`.

Output Format:

You MUST follow the following output format.

- When a watcher with <name> is added to the watch list, you should print out:
  `<name> is added to the watcher-list`
  For example:
  `Tom is added to the watcher-list`

- When you process a new watcher delete request, you should print out:
  `<name> is removed from the watcher-list`
  For example:
  `Tom is removed from the watcher-list`

- When you process a *query-largest* query, you should print out:
  ```
  Largest earthquake in the past 6 hours:
  Magnitude <magnitude> at <place>
  ```
  For example:
  ```
  Largest earthquake in past 6 hours:
  Magnitude 3.999 at 60km E of Cape Yakataga, Alaska
  ```

  Note: if the magnitude-ordered-list is empty, you should print out: `No record on list`

- If the `--all` option has been given, then you should print the following message each time that an earthquake is added to the earthquake-queue/magnitude-ordered-list:
  `Earthquake <place> is inserted into the magnitude-ordered-list`
  For example:
  `Earthquake 60km E of Cape Yakataga, Alaska is inserted into the magnitude-ordered-list`

- When a new earthquake comes in and is added to the magnitude-ordered-list, you should print a single line for each watcher that is within the appropriate distance. Each such line should appear as:
  `Earthquake <place> is close to <name>`
  For example:
  `Earthquake 60km E of Cape Yakataga, Alaska is close to Tom`
  Note that these notification messages should come after the message printed for the –all command.

  Important: The program will be invoked from the command line as: `java EarthquakeNotification [--all] <watcherFile> <earthquakeFile>`
  where`--all` is an optional parameter. If the `--all` parameter is given, then a suitable message is output for every earthquake processed. Otherwise, when the `--all` option is not given, you will only be outputting notification messages (messages about earthquakes that are close enough to a watcher) and messages related to the user requests (add, delete, query-largest).

  DO NOT hardcode filenames within your Java code.