Homework no. 1

The world viewed through the glasses of a public transport enthusiast[1]

Object-Oriented Programming, Computer Science

Version 1.02

Overall description

Design a set of classes in Java for (very simplified) urban traffic simulation. Take care of the object-oriented nature of your solution. In particular, the possibility of extending the simulation with new elements.

The simulation is to include, among others: the following elements:

- passengers,
- vehicles (in this simulation only trams),
- tram lines,
- stops .

*Vehicles*they may be different, but each has a side number and its own line. The lines have their routes described in the data (see data). The route also describes the stopping time at the loop. One vehicle always travels the same route. The vehicle travels its route in one direction, stops at the loop, then returns (via the same route) and stops at the loop again. Trams are vehicles. The first trams leave the route at 6 a.m. After 11 p.m. they finish their journeys (which in the case of the unlucky tram driver who left the loop at 11 p.m. means having to make a full circle, trams do not stay overnight on the loop at the other end of the route). Trams of one line are released in the morning on both sides of the route: half from one end, half from the other (if their number is odd, one more from the first end). Trams leave the route in the morning at equal intervals, calculated as the round trip time with stops at the loop, divided by the number of trams on this route). Each tram has its own capacity, the same for all trams (task parameter).

The tram route consists of stops. Stops have their own names and a set capacity (common to all stops). We do not distinguish between stops in one and the other direction.

Everyone*passenger*lives near a bus stop. Every morning, at a random time between 6 a.m. and 12 p.m. (the time is randomized anew each day), he goes to his stop. If there are no more seats at the stop, he or she gives up traveling that day. Wpp. waiting at the stop for a tram (any line, in any direction). After the tram arrives, he tries to get on. If he succeeds, he randomly selects one of the remaining stops on the route (in one direction), goes there and tries to get off. If this fails, he continues driving, hoping that at some point he will finally succeed. Once he gets to the stop, he starts the same procedure: he waits for the tram and tries to get on. If it fails, he waits for the next one. (For the sake of passengers' well-being, we assume that they take a thermos and a supply of sandwiches from home

[1]Inspired"Words" by Tadeusz Boy-Żeleński .

or buy something unhealthy from a vending machine at the stop while waiting for the tram - however, in the implementation of this task we ignore all these issues).

When a tram arrives at a stop, it first lets out passengers who wanted to get off there (some may not be able to do so when there are no more free seats at the stop). The order in which passengers are let off is arbitrary. Then the waiting passengers try to board the tram (again, some of them may fail when there are no more seats on the tram).

The simulation lasts a given (see data) number of days. It starts on Monday (but it doesn't matter in this version of the task).

When the simulation day ends, all passengers return to their homes (e.g. on foot or by taxi, we do not simulate this anymore, i.e. we do not have a taxi vehicle in the simulation, and in the morning everyone is home). Trams arrive at the correct ends of their routes at night (we don't simulate this anymore, they are simply where they should be in the morning).

Additional assumptions:

- Vehicles do not run after midnight (we assume that the route takes less than an hour, and after 11 p.m. vehicles no longer set off with passengers, they can only return to the depot).

Program result

The program should print three things:

1. Loaded parameters.
2. Detailed simulation record, one line for each event. Each line should contain at the beginning the time of the event (simulation day and hour) and a colon, then a description of the event (e.g. 46, 15:23: Passenger 213 boarded tram line 1 (side no. 14) with the intention of reaching the Centrum stop.).
3. Simulation statistics. At least:
    - the total number of passenger trips (each trip of one passenger in one vehicle is counted separately),
    - average waiting time at the stop (how long on average did someone wait at the stop),
    - for each day of simulation:
        - the total number of journeys (calculated as above),
        - total waiting time at stops (sum after all waiting people).

The results should be written to standard output (usingSystem.out).

Organization of simulations

The simulation is performed using an event queue. From the program's point of view, it is a timeline --- a list of events sorted in non-descending order, e.g. the time of their occurrence. Each object, knowing that it has to do something in the future, puts an appropriate event into the queue with itself and with the time when the event occurs

take place. The simulation consists in downloading the first event (i.e. with the lowest occurrence time) from the queue and asking the retrieved object to react. This happens for the given simulation time (not: real time) or until the queue becomes empty. In our simulation, since nothing runs at night, the event queue should eventually empty every day.

We measure time in our simulation in virtual (not: real) minutes counted from the beginning of the day or the beginning of the simulation (whatever is more convenient for you, in this task there are no events that pass from day to day). We assume the Earth's rounded length of the day (60*24 minutes).

An event queue can be implemented in various ways. Possible implementations include:

- A sorted array that remembers where it is filled. Its size should be doubled when there is no space for further events.
- A sorted list of events (a list in the sense of the first semester at C, i.e. a set of elements, each of which remembers its successor).
- Heap, known e.g. from ASD.

In this task, we expect any of these implementations (all will be scored the same) to provide the operations of inserting a new event into the queue, fetching the first one, and checking whether the queue is empty. When inserting several events at the same time, later events should be removed from the queue later.

We also require that the program includes an interface for the event queue (implemented by your queue, of course), so that you can easily replace one implementation with another (in this task, we only require one implementation).

An attempt to retrieve an event from an empty queue should be captured with an assertion.

Events can create a hierarchy.

Of course, there are classes available in the Java standard library that implement the event queue, but for this task we require your own implementation (one of the ones provided).

Data description

Data for simulation should be read from standard input (see additional requirements).

Data form (texts in angle brackets describe subsequent data, and texts in round brackets describe their types):

<Number of simulation days (int)>
<Stop capacity (int)> <Number of stops (int)> Now for each stop

<Stop name (String)> <Number of passengers (int)>

<Tram capacity (int)> <Number of tram lines (int)>
Now for each line (lines are numbered with consecutive natural numbers): :

<Number of trams on this line, route length, (int, int)> then the route in the form of pairs:

<stop name, arrival time (string, int)>.

The last time in the line description is the stopping time on the loop, we assume it is the same for both loops. For example:

[P1, P2, P3], [1, 2, 3]

means travel times P1<->P2 1 min, P2<->P3 2 min and a stop at loops P1 and P3 3 minutes each.

Comments:

- We assume that all strings are strings of characters other than whitespace (i.e. the place name *MIMUW* and MIM-UW are allowed, but *MIM UW* not any more).
- The presented form of data assumes division into rows --- then the data is much easier for us (protein beings) to create or edit. However, your program does not have to analyze/enforce line division, i.e. it is enough to simply read subsequent numbers or strings without checking their position in the line. However, if someone wants to use the line division suggested in the task in their program, then maybe the test data will be (as in the example below) divided into lines.

Sample (minimum) data:

```
0
0
0
0
0
0
```

Sample (slightly larger) data:

```
3
10
3
Banach
Krakowskie
Center
15
5
1
2 3 Banacha 3 Centrum 2 Krakowskie 10
```

Additional requirements

Down**everyone**draws in simulation should be used**Just**classdraw,having the following methods defined:

public static int draw(int lower, int upper) -gives a random integer z

a lower...upper compartment closed on both sides.

This class should be in the same package as the main program class (the one withmain).

The data for simulation should be read using a classjava.util.Scanner.An example of using this class is presented in the method below. It reads the number of words from the standard input, and then the words themselves. It writes the loaded words to the output after reading is finished.

```
public static void example() {
        Scanner sc = new Scanner(System.in); int count =
        sc.nextInt();
        String[] strings = new String[how many]; for (int k = 0;
        k < subtitles.length; ++k)
                strings[k] = sc.next();
        for (int k = 0; k < subtitles.length; ++k)
                System.out.println(strings[k]);
}
```

Additional remarks

- Please remember that this is an object-oriented programming task, you should definitely not be afraid of using classes, objects, constructors, visibility ranges, etc. For example, a perfectly working program with only one class will be read as unlimited sympathy for round even numbers having infinitely many divisors .

- This is a larger task, please solve it in an incremental manner. For example, reaching the version that reads the first data above is a good stage in creating a solution. Moving on to reading the second data from this task may be one of the next steps (not necessarily the next). Running the simulation with only trams (without passengers yet) also seems to be a reasonable stage of work.
- The task has been designed (more precisely: shortened :)) so that it can be easily done using tables. Since we don't yet know the containers from the Java standard library, we can't require them for this task. And since everyone should have the same task to do, then**you can't**use containers other than arrays for this task.

- Possible implementations of the priority queue have been indicated earlier, if a quick search is needed, you can use array sorting (already known from the class) (or you can write your own) and binary search.
- We accept numbering of everything (lines, side numbers, ...) from zero.

Good luck

Change history:

- 1.0: April 26, 2024, first version.
- 1/01: April 27, 2024: typo corrections.
- 1/02: April 27, 2024: added subtitle, minor corrections and additional explanations (example explaining the meaning of line description elements, assumption of numbering from 0).