The Elevator Problem

Problem Specification:

You have been hired by an airport to build an elevator controller in the 10 floor terminal building which is used by people going from floor to floor (some of whom have trolleys with baggage). The elevator(s) can be represented using threads. Each person boarding or leaving the elevator is also represented by a thread. You must implement the methods called by the arriving person (e.g. a method **`ArrivingGoingFromTo(int atFloor, int toFloor)`** should wake up an elevator (if necessary) and tell it which floor to go to). The elevator speed is fast but not instantaneous, taking only 100 ticks to go from one floor to the adjacent one. A vanilla option would be to assume the following:

- There is only one elevator and it has a maximum weight capacity; obviously this capacity **is** affected by the trolleys (as a first approximation you can take these to be of fixed weight, say equal to that of two people). When the weight of people and trolleys in the lift exceeds the maximum weight, the lift will display an error and remain at the floor until sufficient people (and their trolleys) exit.
- The elevator can measure the details (weight, ID, arrival floor, destination floor etc) of the people when they enter and does not allow more than a specified weight in the elevator.
- There is only one person entering the elevator at a floor at a time.
- The elevator ascends one floor at a time without skipping floors.
- When not in service, the elevator waits at the last floor it visited and 'sleeps'.

You have to come up with a way of storing the requests to the elevator e.g. if a person is accessing the elevator at the 3rd floor and some body is trying to get access from the 5 th floor then latter person has to wait before he could place his request. In other words (s)he has to be put in a queue. Also if the elevator is going upwards all the requests for the destinations downwards should not be accepted until it starts its downward motion and vice versa.

These specifications are for the vanilla option of the implementation. Any form of creativity that you feel like putting in that will add interest to my marking of the project (variable weight trolleys, Nice GUIs, multiple elevators, 'smart' elevators with AI, random events such as faulty lifts etc) is encouraged and marks awarded for the project will reflect this. Making use of java large-scale thread management support would be essential.

Hint: One Approach to the Problem Might be the following:

The program will contain two classes "elevator" and "person". Each "person" should be generated with a random arrival time, arrival floor and destination floor and have a unique ID for identification purposes.
In order to simulate the 100 ticks between the two consecutive floors, you can use the **`wait (100)`** construct.
You can assume 100 ticks is equivalent to your 1 unit of time (discussed above).
In order to compile your java code, I should just be able to do so with java compiler with " **javac <filename>**" and after it gets compiled execute it with " **java <filename(executable)>**". ***Please do not rely on my using an IDE for the execution of your code***.

Output file:

The format for an output file should be something like following:
"**Person (Thread ID) # makes request at time # starting at floor # with the destination floor #**".
There should be as many lines printed as are the number of requests. You should also generate an output file output.dat to store your output.
Note: You must assume that you the starting time for the first request is 0.

Deliverables and instructions for submission:

1. The assignment is a **two** person project, worth 10% of the overall module mark and **has** to be done in Java (it's up to you if you want to do it individually, but marking will be the same).
2. Please also write a short (4 page max) design documentation which may get you some partial credit if I cannot get your program to execute as it should. The report should (among other things) describe how you divided the work between you, how to compile and run the code, how your solution addresses issues like fairness, prevention of starvation etc.
3. You have to submit all your source files, documentation and also all the generated class files for the unix platform.

4. Please Mail the above to me with a completed plagiarism declaration form which is signed by **both of you**.
5. Deadline for submission is the **Friday 22nd March at 5.00pm**. Penalties will apply for late submissions.
6. There will be a short intermediate presentation phase whereby you present your approach to the class. This will probably happen in week 6 but more details will be given beforehand.

A Word on Resit Continuous Assessment:

1. *A resit is available for all components of this module.*
2. *Students who fail the module on the first sitting must resit each component (CA or exam) that they failed. In particular, students who fail the CA must resit the CA.*
3. *A student may not resit a component that has been passed and their marks for the component are carried forward*
4. *Marks for failed components will be set to zero in ITS before the resit.*

Good luck!

i.e. one that is not exotic