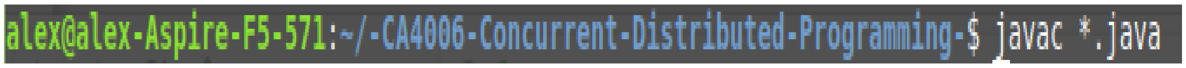
CA4006: Concurrent Java Elevator Simulation

Alex Randles: 15302766
Kealan Thompson: 12417338

How to run and compile?

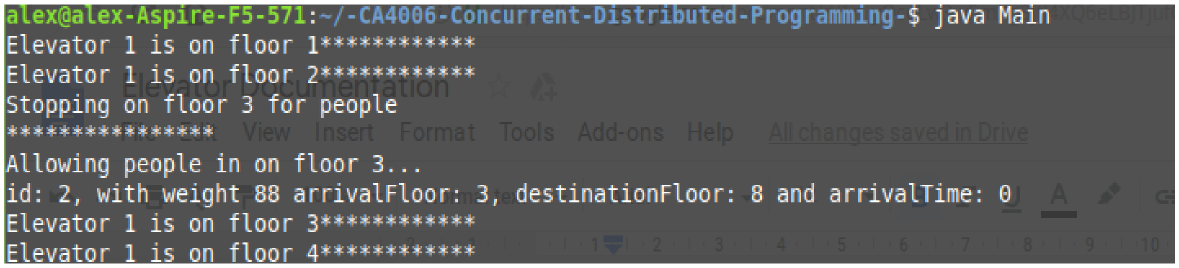
- Compiling



```
alex@alex-Aspire-F5-571:~/-CA4006-Concurrent-Distributed-Programming-$ javac *.java
```

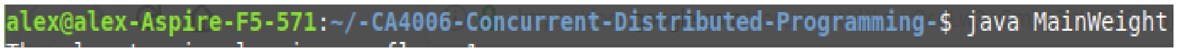
- Running
  There are 5 different main methods available that stimulate different scenarios within our system.

-Stimulate a normal run with 100 random people (Main.java)



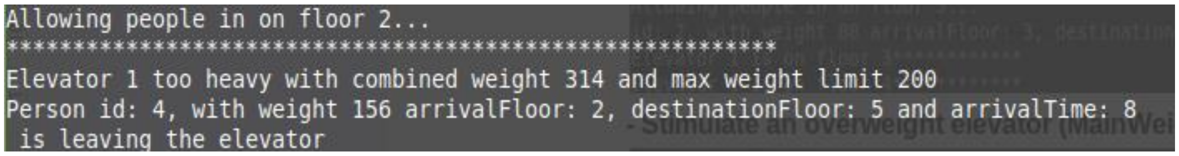
```
alex@alex-Aspire-F5-571:~/-CA4006-Concurrent-Distributed-Programming-$ java Main
Elevator 1 is on floor 1************
Elevator 1 is on floor 2************
Stopping on floor 3 for people
****************
Allowing people in on floor 3...
id: 2, with weight 88 arrivalFloor: 3, destinationFloor: 8 and arrivalTime: 0
Elevator 1 is on floor 3************
Elevator 1 is on floor 4************
```

- Stimulate an overweight elevator (MainWeight.java)



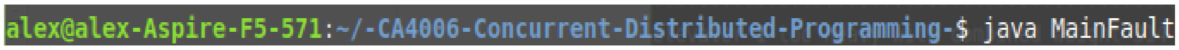
```
alex@alex-Aspire-F5-571:~/-CA4006-Concurrent-Distributed-Programming-$ java MainWeight
```

When the elevator becomes overweight the following will be printed:



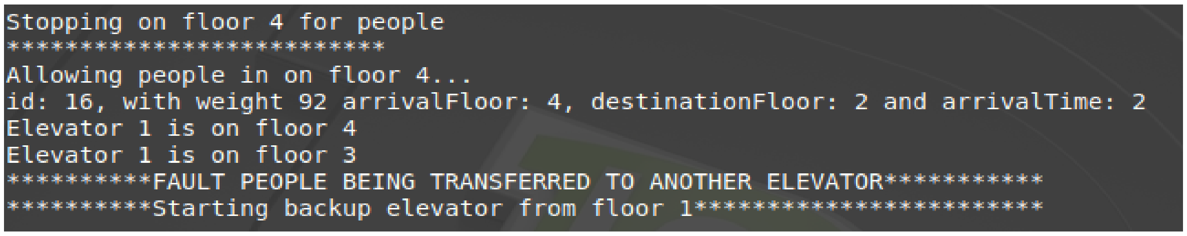
```
Allowing people in on floor 2...
***********************************************************
Elevator 1 too heavy with combined weight 314 and max weight limit 200
Person id: 4, with weight 156 arrivalFloor: 2, destinationFloor: 5 and arrivalTime: 8
 is leaving the elevator
```

- Stimulate a fault (MainFault.java)



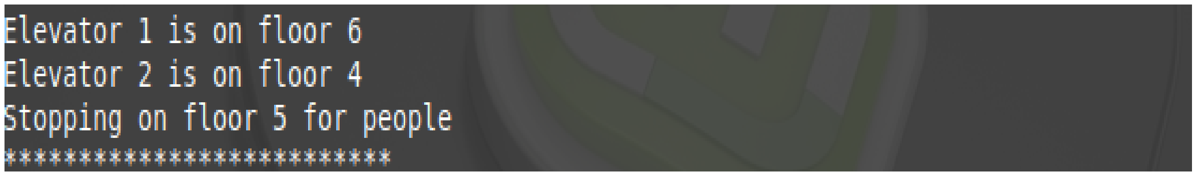
```
alex@alex-Aspire-F5-571:~/-CA4006-Concurrent-Distributed-Programming-$ java MainFault
```

When a fault occurs the following will be printed and all the people currently waiting or in the elevator will be transferred to another elevator.



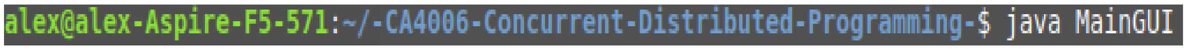
```
Stopping on floor 4 for people
**************************
Allowing people in on floor 4...
id: 16, with weight 92 arrivalFloor: 4, destinationFloor: 2 and arrivalTime: 2
Elevator 1 is on floor 4
Elevator 1 is on floor 3
**********FAULT PEOPLE BEING TRANSFERRED TO ANOTHER ELEVATOR***********
**********Starting backup elevator from floor 1*************************
```

-Multiple elevators (MainMultipleElevators.java)



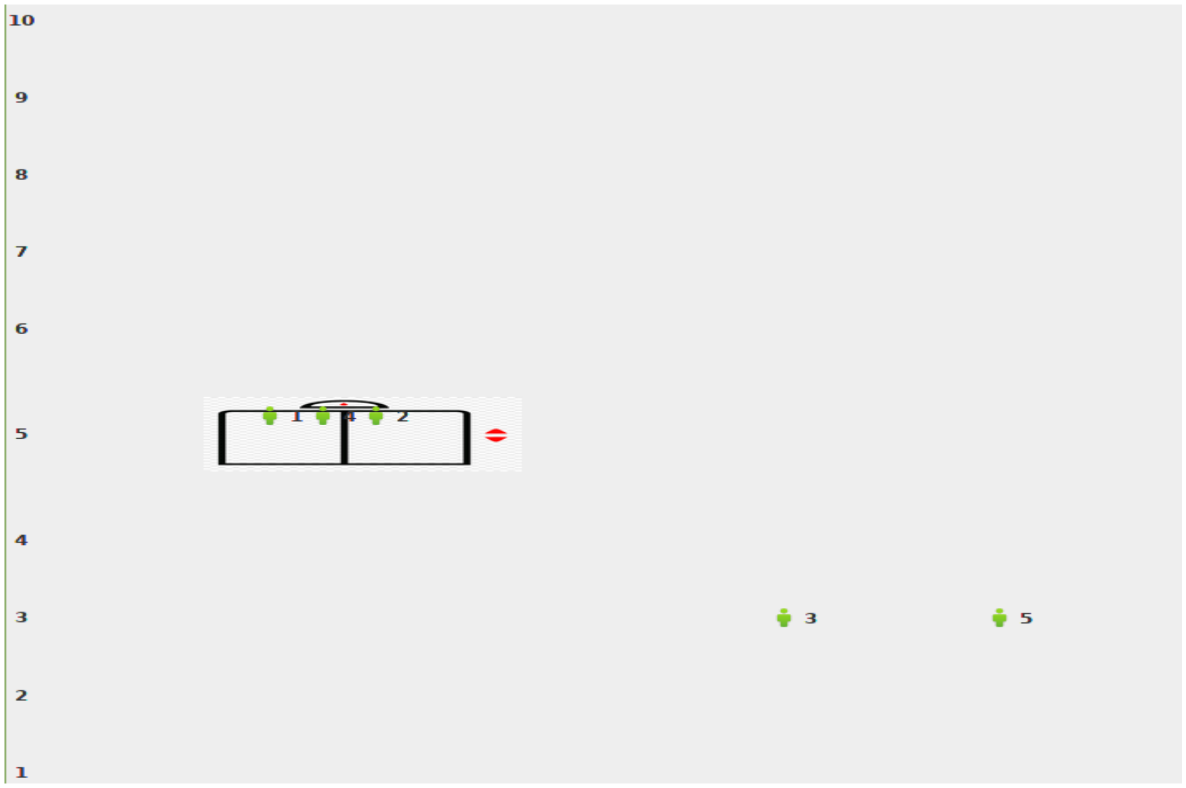
```
Elevator 1 is on floor 6
Elevator 2 is on floor 4
Stopping on floor 5 for people
**************************
```

-Using the elevator simulator GUI (MainGUI.java)



```
alex@alex-Aspire-F5-571:~/-CA4006-Concurrent-Distributed-Programming-$ java MainGUI
```

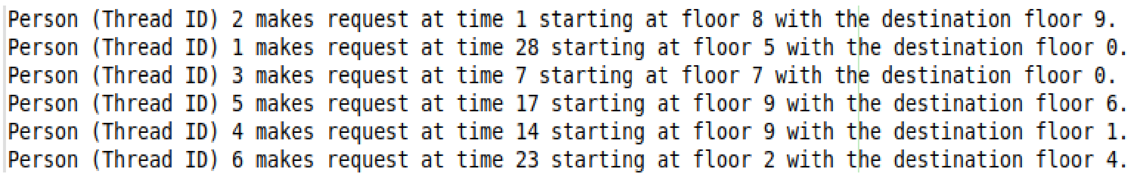
This will create a GUI that will illustrate the functionality of our code. People are will be moved in and out of the elevator accordingly.



- Output Files

-Request.dat

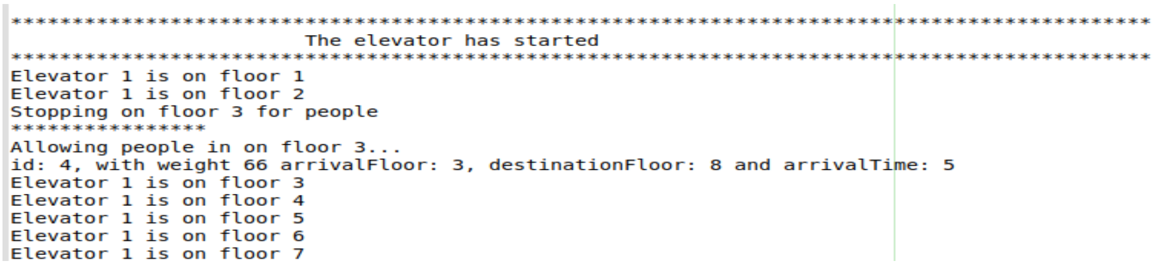
This file contains all the request the elevator has received and includes all the information relating to themselves and their request.



```
Person (Thread ID) 2 makes request at time 1 starting at floor 8 with the destination floor 9.
Person (Thread ID) 1 makes request at time 28 starting at floor 5 with the destination floor 0.
Person (Thread ID) 3 makes request at time 7 starting at floor 7 with the destination floor 0.
Person (Thread ID) 5 makes request at time 17 starting at floor 9 with the destination floor 6.
Person (Thread ID) 4 makes request at time 14 starting at floor 9 with the destination floor 1.
Person (Thread ID) 6 makes request at time 23 starting at floor 2 with the destination floor 4.
```

-Output.dat

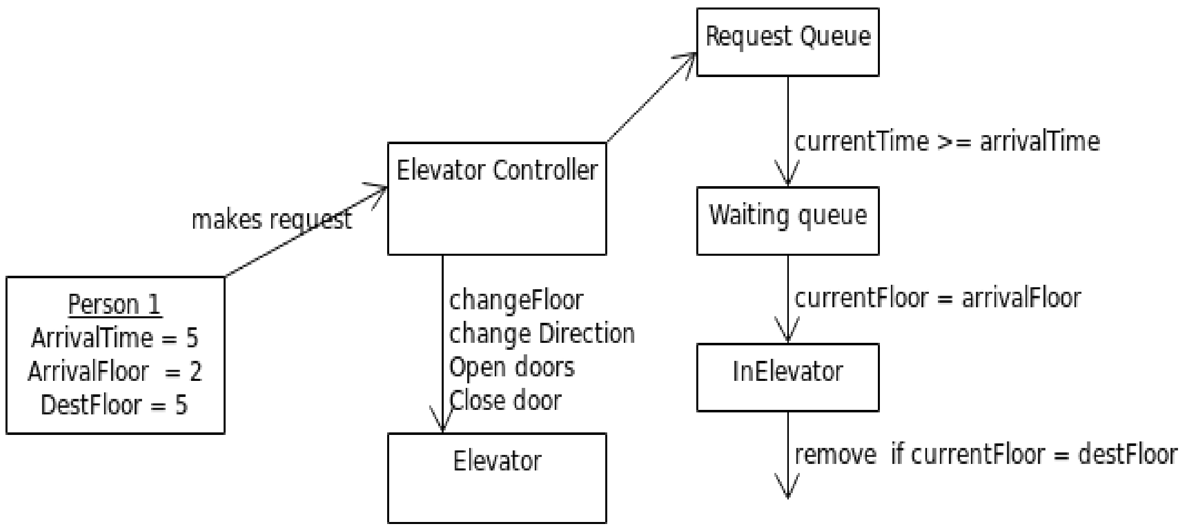
This contains all the output from the actions of the elevator such as which floor it's on, it's going to and who is entering and leaving the elevator.



```
*********************************************************************************************
                        The elevator has started
*********************************************************************************************
Elevator 1 is on floor 1
Elevator 1 is on floor 2
Stopping on floor 3 for people
****************
Allowing people in on floor 3...
id: 4, with weight 66 arrivalFloor: 3, destinationFloor: 8 and arrivalTime: 5
Elevator 1 is on floor 3
Elevator 1 is on floor 4
Elevator 1 is on floor 5
Elevator 1 is on floor 6
Elevator 1 is on floor 7
```

The design

- **Person thread ->** Person.java contains all the variables relating to the person and when the thread is run it will make a request to the elevator controller
- **Elevator Thread ->** Elevator.java runs through the actions of the elevator controller such as changeFloor, stop at floors and allow people in and out.
- **ElevatorController class ->** ElevatorController.java will control the actions of the elevator such as which direction is goes when it lets people in and out. This class has three queues: request (people who have requested the elevator), waiting (people who are currently waiting on their arrival floor) and inElevator (people who are currently in the elevator).Through the use of synchronized methods and push and pop methods the people and elevator are moved around.
- **ElevatorControllerGUI ->**. ElevatorControllerGUI.java is an elevator controller with a GUI.
- **Main Method ->** Main.java this controls the interaction between the person and elevator threads. This will create 100 random people and pass them through our system.
- **Main with fault ->** MainFault.java, this will create a fault within the a random time between 1 - 10 and stop the current elevator thread, a new elevator will be started and all people in the old elevators queues will be transferred over in order.
- **Main with heavy people ->** MainWeight.java will create heavier than normal people to show that the elevator registers weight. The person that makes it overweight will exit the elevator at the floor they go on and will be put at the front of that queue.
- **Main with multiple elevators ->** MainMultipleElevators.java this will stimulate two elevators working at the same time.
- **Main with GUI ->** MainGUI.java will initialize a ElevatorControllerGUI instance and a display will pop up showing the people and their corresponding ID within



the program moving within the elevator system.

- The above diagram outlines the basic functionality of our solution. A Main class orchestrates the program, creating person objects which are assigned an arrival time, floor, destination floor and weight randomly. The person object then makes a request to the elevator controller which monitors the functions of the elevator. Transferring people who have yet to arrive to a requests queue, those who have arrived to a waiting queue and picking up the passengers from the waiting queue of the current floor.

  Our solution uses a combination of concurrent hash maps and concurrent linked queues to implement this design concurrently. With a concurrent hash map representing the request, waiting and in elevator queues. Mapping an integer to a concurrent linked queue of person objects.

Addressing issues

- **Fairness ->** Person threads with a smaller arrival arrival time and closer to the elevators current floor will get chosen first. If two people have the same arrival time whoever is closer to the elevators current floor and in the same direction that its currently going will be picked first similar to a normal elevator.
- **Prevention of starvation ->** The elevator will not sleep until all current request are fulfilled, when this is achieved the elevator will sleep until further request are made.

Assumptions

- If the elevator direction is up but there are no current request from floors above it will begin to go down.
- The elevator will print what floor its on but it'll skip any floors that don't have people seeking to enter or exit the elevator

Dividing Workload

- In order to work on this project simultaneously we set up a git repository, planned out our approach and divided up the workload. With both of us working on seperate basic functionality, then combining our work and working together to get the simulator to behave as expected.