# Elevator System Final Report

SYSC 3303A

**Team 4**
Alexander Bhend, 101073223
Colin Crasta, 101115422
Dharshatharan Jayatharan Aronan, 101134019
Bailey Lyster, 101115419
Quinn Sondermeyer, 101132073

April 14, 2021

# Table of Contents

## 1.0 Responsibilities

| | |
|---|---|
| Iteration 1 | **Alex:**<br>-FloorSubsystem.java<br>-RequestElevatorEvent.java<br>**Bailey:**<br>-Door.java<br>-Other classes omitted from Iteration 1 due to quick last minute restructuring of<br>the class diagram since we did more than what was asked for this iteration. His<br>classes will be used for future iterations due to this.<br>-Helped with README.txt<br>**Colin:**<br>-Scheduler.java<br>-ElevatorState.java<br>-FloorNumber.java<br>-Direction.java<br>**Dharsh:**<br>-Elevator.java<br>-ElevatorInfo.java<br>-ElevatorJob.java<br>**Quinn:**<br>-ElevatorTest.java<br>-ElevatorSystemSimulation.java |
| Iteration 2 | **Alex:**<br>-FloorSubsystem.java<br>-RequestElevatorEvent.java<br>**Bailey:**<br>-moving.java<br>-Elevator.java<br>**Colin:**<br>-idle.java<br>-UML Diagrams<br>-Elevator.java<br>**Dharsh:**<br>-ReceiveSchedulerRequest.java<br>-SchedulerState.java<br>-ProcessFault.java<br>-fault.java<br>**Quinn:**<br>-stopped.java<br>-ElevatorState.java<br>-Refactoring and merging |

| | |
|---|---|
| Iteration 3 | **Alex:**<br>-FloorSubsystem.java<br>-RequestElevatorEvent.java<br>-FloorSchedulerCommunicator.java<br>**Bailey:**<br>-moving.java<br>-Elevator.java<br>-ElevatorSchedulerCommunicator.java<br>**Colin:**<br>-idle.java<br>-UML Diagrams<br>-SchedulerFloorCommunicator.java<br>**Dharsh:**<br>-ReceiveSchedulerRequest.java<br>-Scheduler.java<br>-SchedulerElevatorCommunicator.java<br>**Quinn:**<br>-stopped.java<br>-ElevatorState.java<br>-ElevatorSubsystem.java<br>-Refactoring and merging |
| Iteration 4 | **Alex:**<br>-Set up a protocol (starting from the input.txt file) to trigger this fault in the elevator<br>-Set up a timer when both opening and closing the door<br>-Cancel the timer if the door closes in time<br>-Create a function that the timer will call in the case of a timeout<br>**Bailey:**<br>-Handle the timeout case (If the door fails to open, try closing and opening again (and vice versa) a total of 2 times excluding the fault. If the door is still not able to close, shutdown the elevator and send a notification to the scheduler)<br>-Create a protocol to communicate the door failure and shutdown<br>-Communicate the shutdown to the floor subsystem<br>**Colin:**<br>-Set up a protocol (starting from the input.txt file) to trigger this fault in the elevator<br>-Set up one timer per elevator<br>-Start the timer for the specific elevator in<br>-ProcessElevatorInfoThread class in the processElevatorInfo() function<br>**Dharsh:** |

| | |
|---|---|
| | -Cleaning up iteration 3 logic<br>-Worked on majority of the refactoring of code to be able to trigger faults in the right areas<br>-Refactored the code to change the way we were processing jobs, our previous model didn't allow for doing multiple jobs at a time and also being efficient in terms of where to stop first<br>-Made the scheduling logic more efficient<br>**Quinn:**<br>-Have the timer call a function that will deal with the fault<br>-The fault should be handled by shutting down the respective elevator (this should be done by sending a notification to the respective elevator) and also marking it as shutdown in the scheduler<br>-The elevator needs a shutdown field by getting all threads related to that elevator to check if shutdown constantly (refer to lectures)<br>-Send shutdown notification to floor |
| Iteration 5 | **Alex:**<br>- Report Structuring<br>- Data analysis<br>- Measuring time to go through input file<br>**Bailey:**<br>- GUI for elevator subsystem<br>- Refactoring GUI to remove flickering<br>- Elevator UML<br>**Colin:**<br>-  Floor UML<br>- Scheduler UML<br>**Dharsh:**<br>- Timing Diagrams<br>- State Diagram<br>- CommunicationTest.java<br>- ElevatorStateTest.java<br>- ElevatorSubsystemTest.java<br>- Refactoring for 22 floors from 9<br>**Quinn:**<br>- Sequence Diagram<br>- DoorFaultsTest.java<br>- FloorBtwFaultsTest.java |

Table 1 - Breakdown of Responsibilities by Iteration

# 2.0 Diagrams

## 2.1 UML Class Diagrams

### 2.1.1 Elevator

*Figure 1* demonstrates the UML class diagram for the Elevator package.

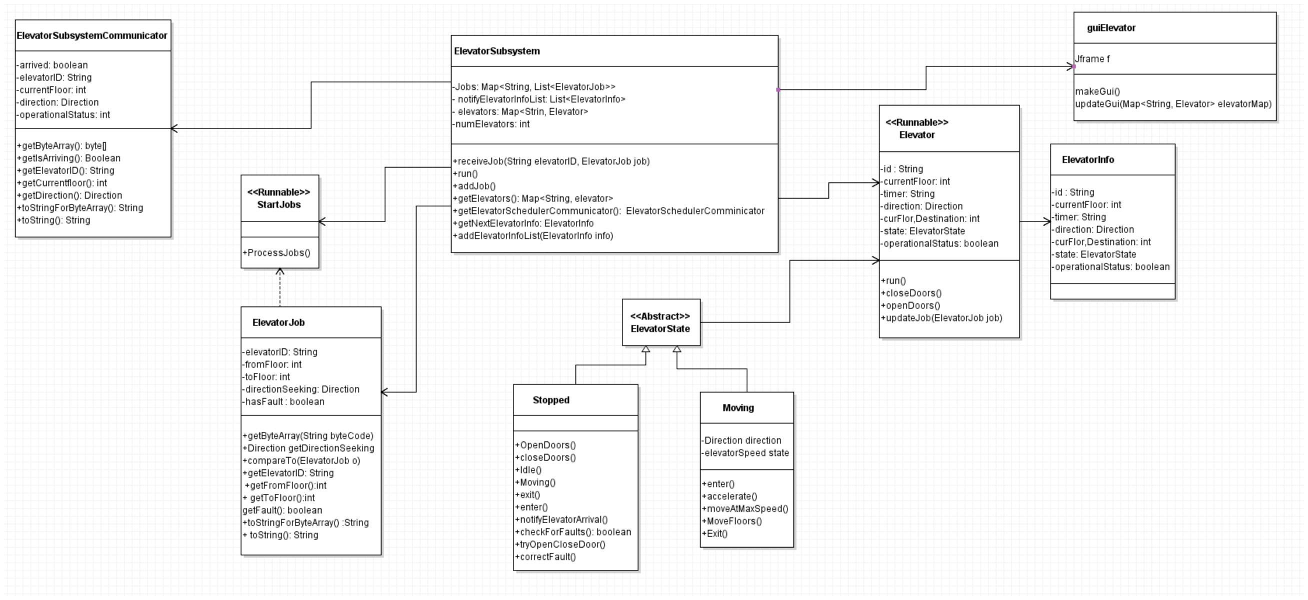


Figure 1 - Elevator Package UML Class Diagram

### 2.1.2 Floor

*Figure 2* demonstrates the UML class diagram for the Floor package.

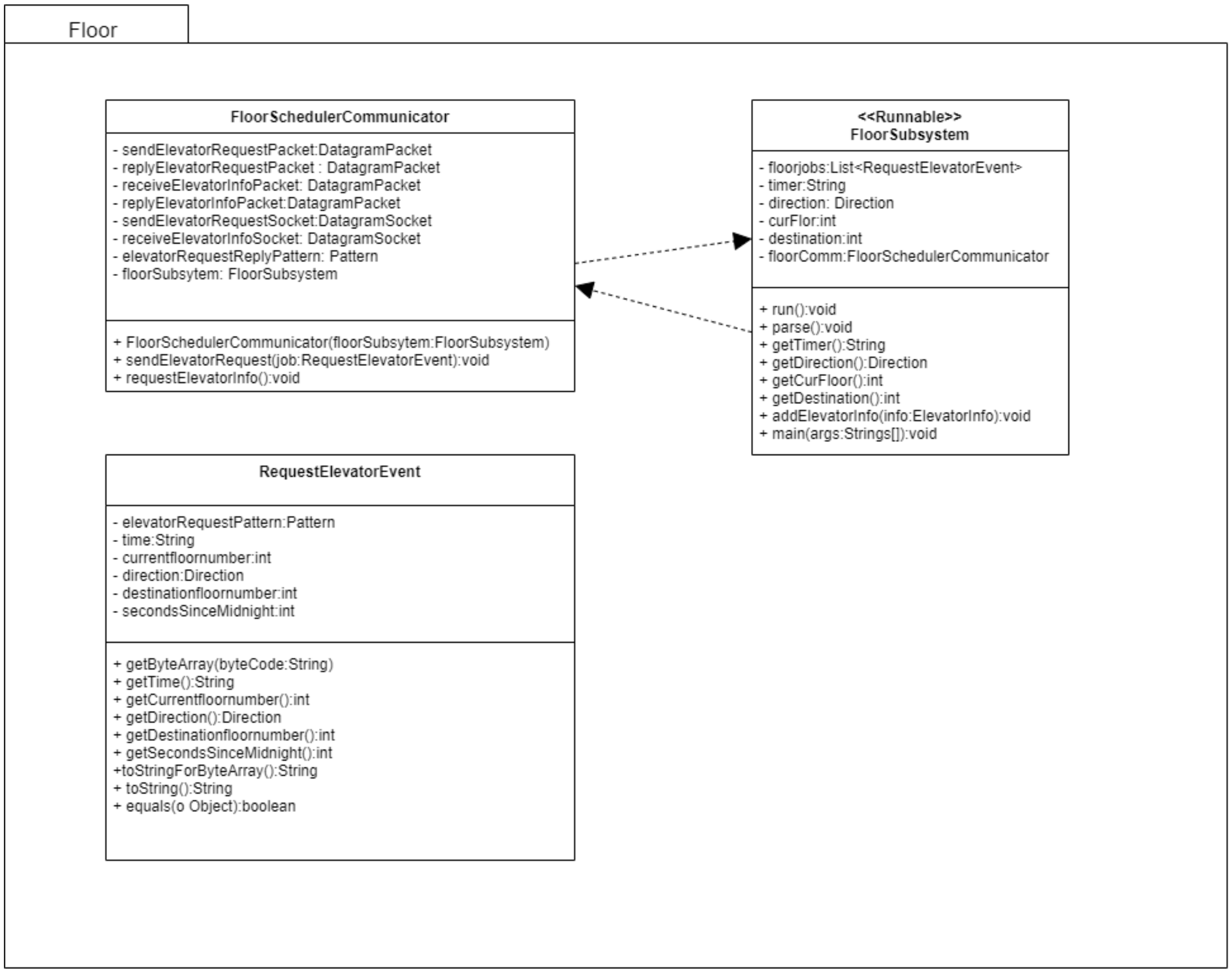

Floor
FloorSchedulerCommunicator
- sendElevatorRequestPacket:DatagramPacket
- replyElevatorRequestPacket : DatagramPacket
- receiveElevatorInfoPacket: DatagramPacket
- replyElevatorInfoPacket:DatagramPacket
- sendElevatorRequestSocket:DatagramSocket
- receiveElevatorInfoSocket: DatagramSocket
- elevatorRequestReplyPattern: Pattern
- floorSubsytem: FloorSubsystem
+ FloorSchedulerCommunicator(floorSubsytem:FloorSubsystem)
+ sendElevatorRequest(job:RequestElevatorEvent):void
+ requestElevatorInfo():void
<<Runnable>>
FloorSubsystem
- floorjobs:List<RequestElevatorEvent>
- timer:String
- direction: Direction
- curFlor:int
- destination:int
- floorComm:FloorSchedulerCommunicator
+ run():void
+ parse():void
+ getTimer():String
+ getDirection():Direction
+ getCurFloor():int
+ getDestination():int
+ addElevatorInfo(info:ElevatorInfo):void
+ main(args:Strings[]):void
RequestElevatorEvent
- elevatorRequestPattern:Pattern
- time:String
- currentfloornumber:int
- direction:Direction
- destinationfloornumber:int
- secondsSinceMidnight:int
+ getByteArray(byteCode:String)
+ getTime():String
+ getCurrentfloornumber():int
+ getDirection():Direction
+ getDestinationfloornumber():int
+ getSecondsSinceMidnight():int
+toStringForByteArray():String
+ toString():String
+ equals(o Object):boolean

Figure 2 - Floor Package UML Class Diagram

## 2.1.3 Scheduler

*Figure 3* demonstrates the UML class diagram for the Scheduler package.

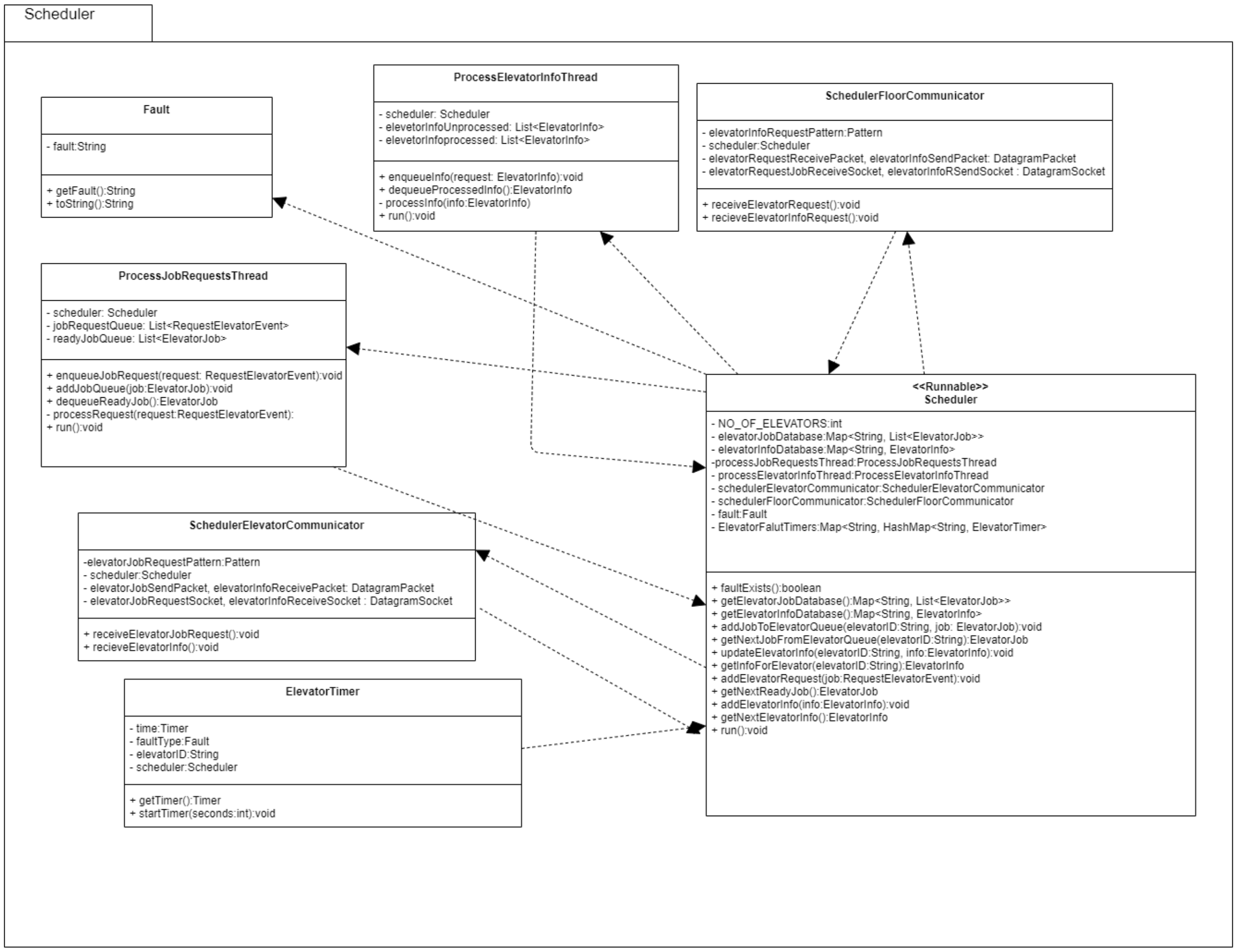

Scheduler
Fault
- fault:String
+ getFault():String
+ toString():String
ProcessElevatorInfoThread
- scheduler: Scheduler
- elevetorInfoUnprocessed: List<ElevatorInfo>
- elevetorInfoprocessed: List<ElevatorInfo>
+ enqueueInfo(request: ElevatorInfo):void
+ dequeueProcessedInfo():ElevatorInfo
- processInfo(info:ElevatorInfo)
+ run():void
SchedulerFloorCommunicator
- elevatorInfoRequestPattern:Pattern
- scheduler:Scheduler
- elevatorRequestReceivePacket, elevatorInfoSendPacket: DatagramPacket
- elevatorRequestJobReceiveSocket, elevatorInfoRSendSocket : DatagramSocket
+ receiveElevatorRequest():void
+ recieveElevatorInfoRequest():void
ProcessJobRequestsThread
- scheduler: Scheduler
- jobRequestQueue: List<RequestElevatorEvent>
- readyJobQueue: List<ElevatorJob>
+ enqueueJobRequest(request: RequestElevatorEvent):void
+ addJobQueue(job:ElevatorJob):void
+ dequeueReadyJob():ElevatorJob
- processRequest(request:RequestElevatorEvent):
+ run():void
<<Runnable>>
Scheduler
- NO_OF_ELEVATORS:int
- elevatorJobDatabase:Map<String, List<ElevatorJob>>
- elevatorInfoDatabase:Map<String, ElevatorInfo>
-processJobRequestsThread:ProcessJobRequestsThread
- processElevatorInfoThread:ProcessElevatorInfoThread
- schedulerElevatorCommunicator:SchedulerElevatorCommunicator
- schedulerFloorCommunicator:SchedulerFloorCommunicator
- fault:Fault
- ElevatorFalutTimers:Map<String, HashMap<String, ElevatorTimer>
+ faultExists():boolean
+ getElevatorJobDatabase():Map<String, List<ElevatorJob>>
+ getElevatorInfoDatabase():Map<String, ElevatorInfo>
+ addJobToElevatorQueue(elevatorID:String, job: ElevatorJob):void
+ getNextJobFromElevatorQueue(elevatorID:String):ElevatorJob
+ updateElevatorInfo(elevatorID:String, info:ElevatorInfo):void
+ getInfoForElevator(elevatorID:String):ElevatorInfo
+ addElevatorRequest(job:RequestElevatorEvent):void
+ getNextReadyJob():ElevatorJob
+ addElevatorInfo(info:ElevatorInfo):void
+ getNextElevatorInfo():ElevatorInfo
+ run():void
SchedulerElevatorCommunicator
-elevatorJobRequestPattern:Pattern
- scheduler:Scheduler
- elevatorJobSendPacket, elevatorInfoReceivePacket: DatagramPacket
- elevatorJobRequestSocket, elevatorInfoReceiveSocket : DatagramSocket
+ receiveElevatorJobRequest():void
+ recieveElevatorInfo():void
ElevatorTimer
- time:Timer
- faultType:Fault
- elevatorID:String
- scheduler:Scheduler
+ getTimer():Timer
+ startTimer(seconds:int):void

Figure 3 - Scheduler Package UML Class Diagram

## 2.2 State Machine

Our scheduler doesn't have states but our elevators do. *Figure 4* demonstrates the state diagram for the elevator system.

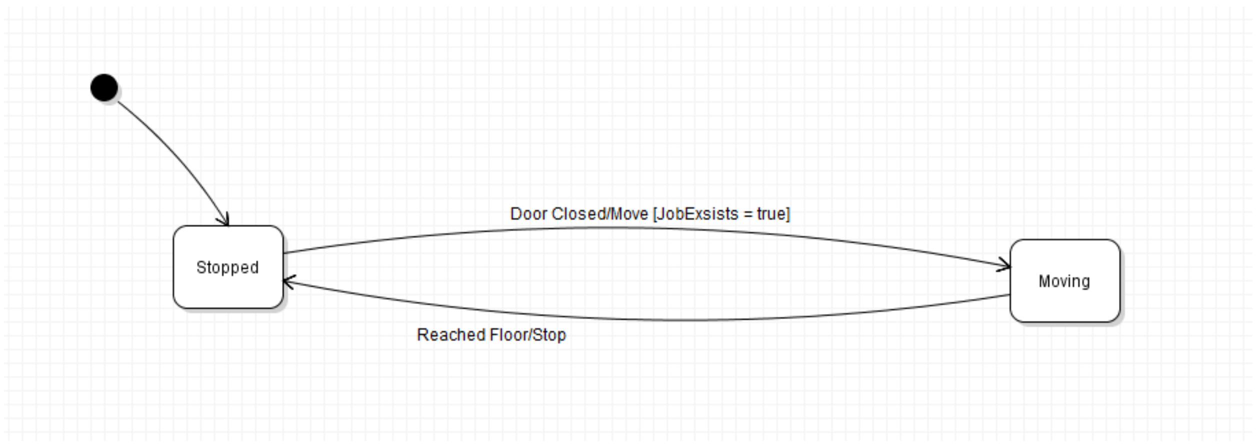


Figure 4 - Elevator State Machine

## 2.3 Sequence Diagrams

*Figure 5*, demonstrates the sequence diagram depicting all error scenarios between the three main systems.

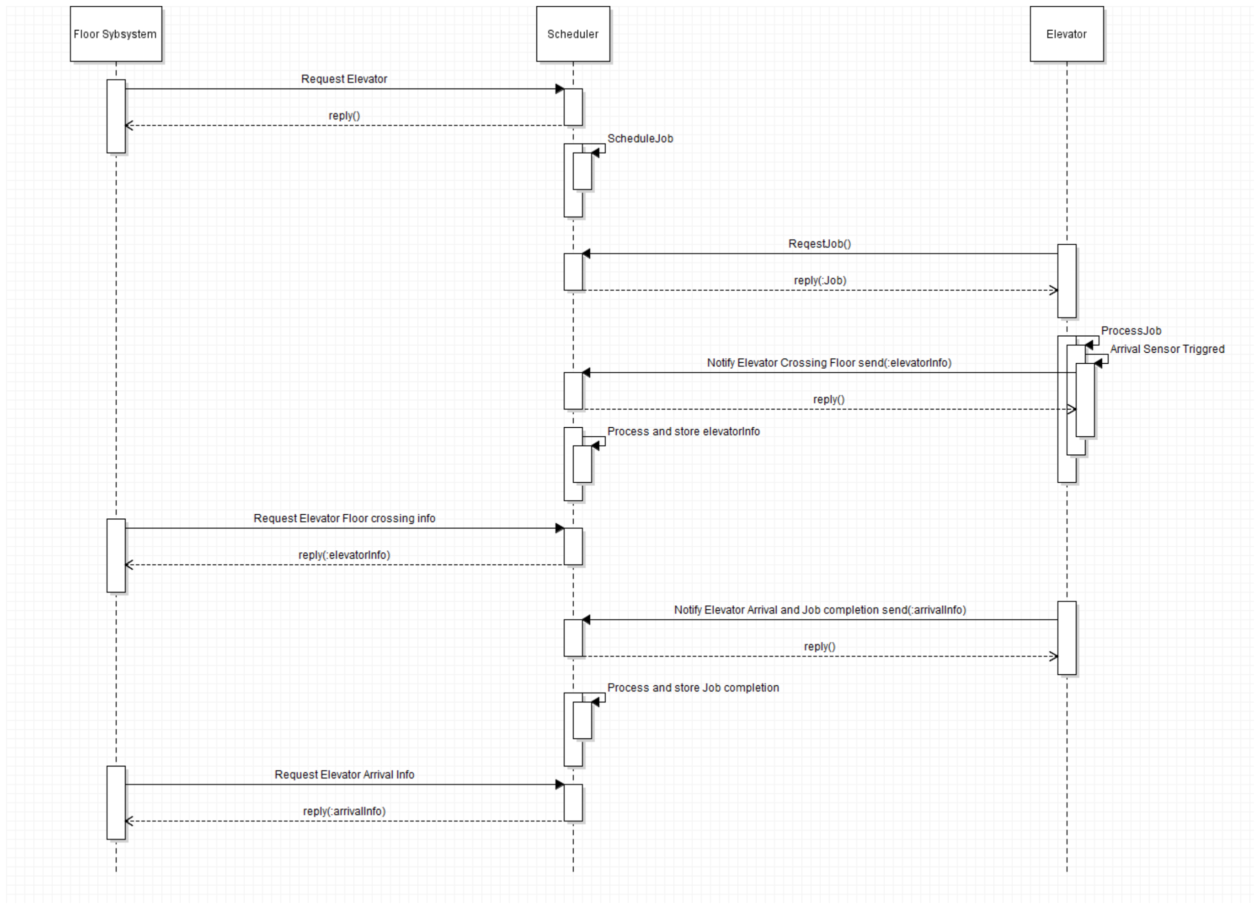

Floor Sybsystem
Scheduler
Elevator
Request Elevator
reply()
ScheduleJob
ReqestJob()
reply(:Job)
ProcessJob
Arrival Sensor Triggred
Notify Elevator Crossing Floor send(:elevatorInfo)
reply()
Process and store elevatorInfo
Request Elevator Floor crossing info
reply(:elevatorInfo)
Notify Elevator Arrival and Job completion send(:arrivalInfo)
reply()
Process and store Job completion
Request Elevator Arrival Info
reply(:arrivalInfo)

Figure 5 - Error Scenarios Sequence Diagram

## 2.4 Timing Diagrams

*Figure 6*, demonstrates the timing diagram for a door fault.

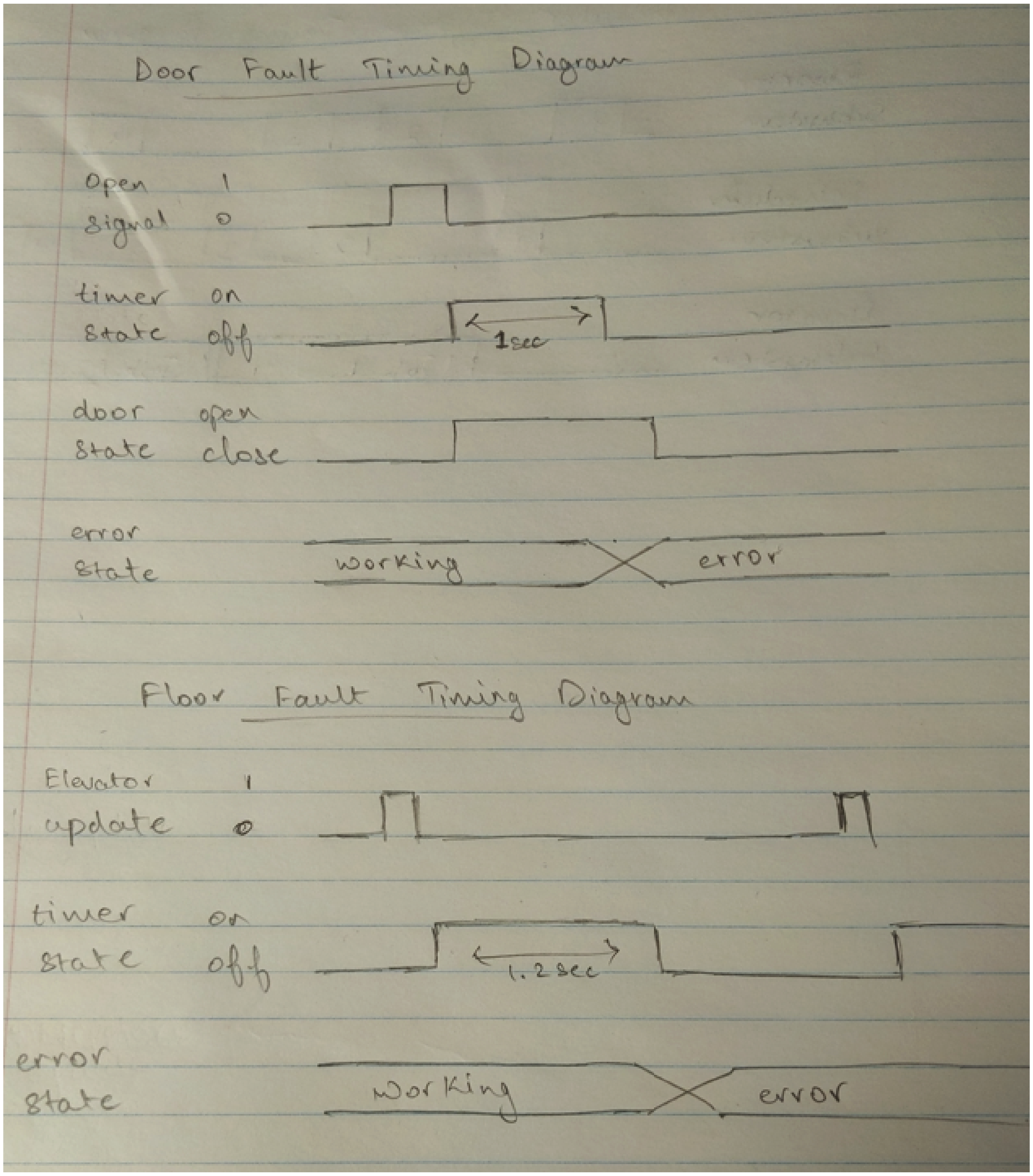


Figure 6 - Door Fault Timing Diagram

*Figure 7*, demonstrates the timing diagram for the scheduler.

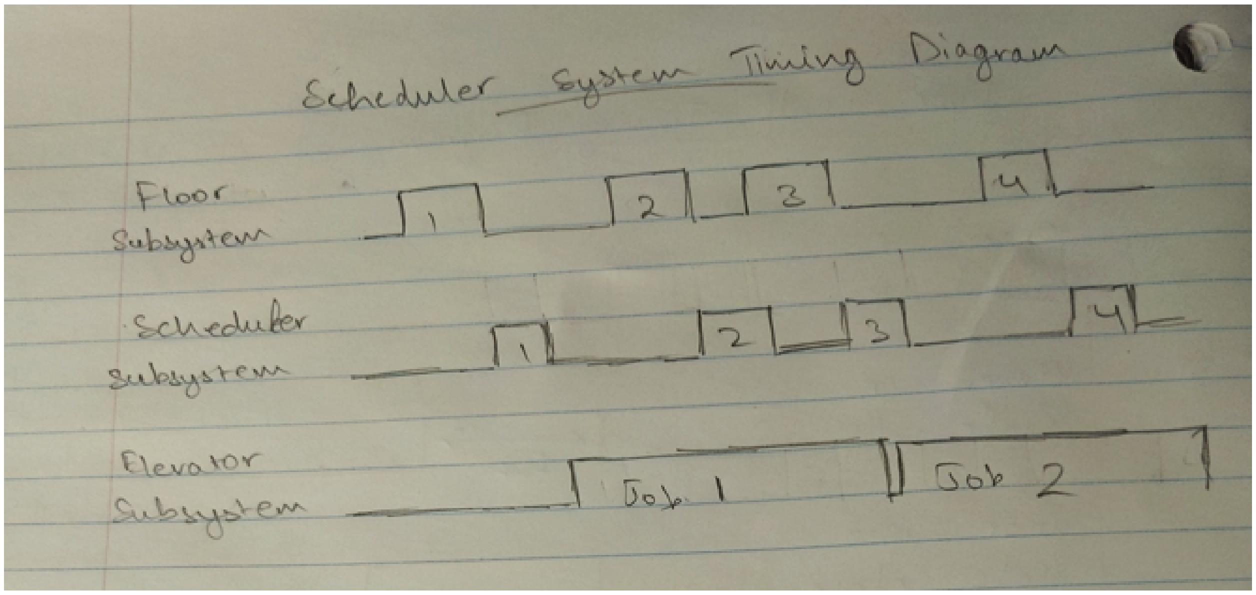


Figure 7 - Scheduler Timing Diagram

# 3.0 Instructions

## 3.1 Setup

1. Open project (ensure the project is complete, input.txt needs to be in the project root folder)
2. Run the following files in order
    a. Scheduler.java
    b. FloorSubsystem.java
    c. ElevatorSubsystem.java
3. Observe the console for the progress of the code as it executes.
4. Observe the GUI for the elevator states

## 3.2 Test

1. Faults Test, run the following independently:
    a. FaultBtwFloorsTest
    b. DoorFaultTest

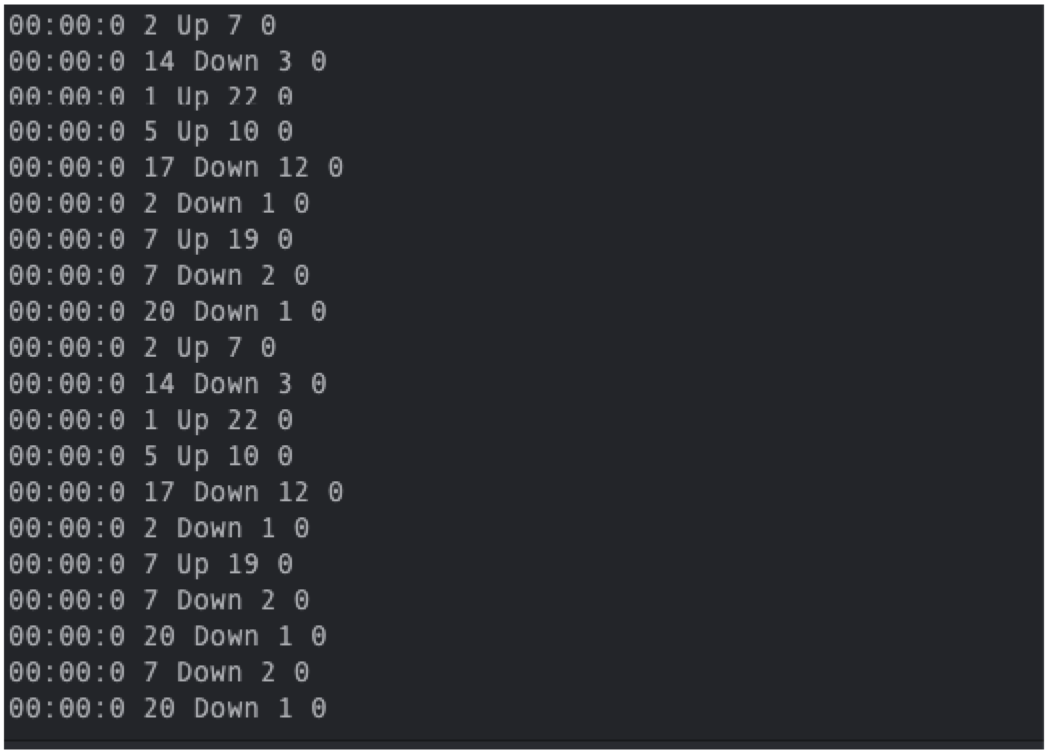
# 4.0 Measurements

*Table 2* demonstrates our selections for the different timing mechanisms.

| Event | Time (ms) |
|---|---|
| Acceleration | 1000 |
| Max Speed Between Floors | 800 |
| Deceleration | 1000 |
| Open Doors | 1000 |
| Close Doors | 1000 |

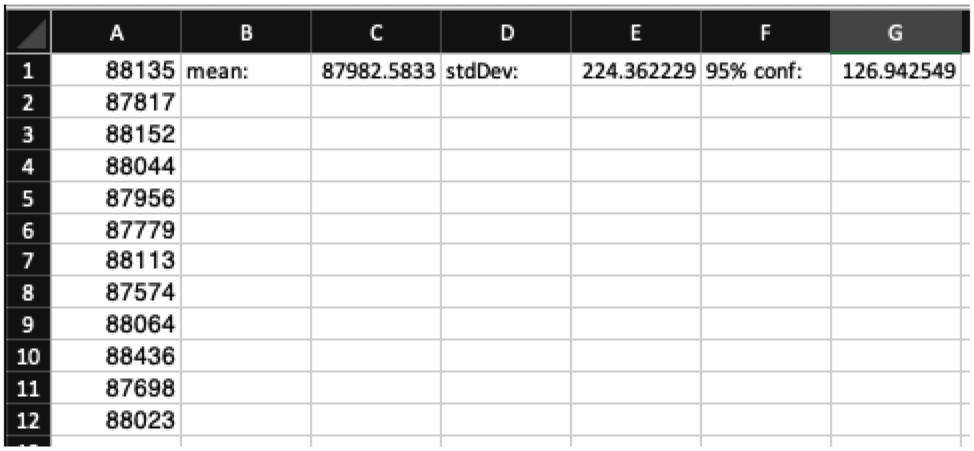
Table 2 - Timing of All Timed Events

Using these times when running the program, we ran 12 trials using the input file shown in *Figure 8* to see how long it takes to complete execution. The results are summarized in *Table 3*. We did not test timing with any errors triggered.



```
00:00:0 2 Up 7 0
00:00:0 14 Down 3 0
00:00:0 1 Up 22 0
00:00:0 5 Up 10 0
00:00:0 17 Down 12 0
00:00:0 2 Down 1 0
00:00:0 7 Up 19 0
00:00:0 7 Down 2 0
00:00:0 20 Down 1 0
00:00:0 2 Up 7 0
00:00:0 14 Down 3 0
00:00:0 1 Up 22 0
00:00:0 5 Up 10 0
00:00:0 17 Down 12 0
00:00:0 2 Down 1 0
00:00:0 7 Up 19 0
00:00:0 7 Down 2 0
00:00:0 20 Down 1 0
00:00:0 7 Down 2 0
00:00:0 20 Down 1 0
```

Figure 8 - Input File for Testing



| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | 88135 | mean: | 87982.5833 | stdDev: | 224.362229 | 95% conf: | 126.942549 |
| 2 | 87817 | | | | | | |
| 3 | 88152 | | | | | | |
| 4 | 88044 | | | | | | |
| 5 | 87956 | | | | | | |
| 6 | 87779 | | | | | | |
| 7 | 88113 | | | | | | |
| 8 | 87574 | | | | | | |
| 9 | 88064 | | | | | | |
| 10 | 88436 | | | | | | |
| 11 | 87698 | | | | | | |
| 12 | 88023 | | | | | | |

Table 3 - Input File Parsing Trials

Based on these trials, we found our system has a mean of about 87982.6 ms to run the input file. This data was then used to create a 95% confidence interval which resulted in an interval of 87855.7 - 88109.5 ms. This makes sense as the input file contains 20 jobs which all begin at the same time, and takes time to complete each job.

## 5.0 Analysis

Our current application is able to complete 20 jobs in under 90 seconds. On average each job is completed in

Another analysis to be made on our scheduler is that it is designed in a robust manner, the scheduler is capable of adapting to faults and errors in runtime. As soon as an error that renders an elevator non-operational is detected, that elevator is will not be scheduled any new jobs, which in practice actually increases the effective throughput of jobs, as  jobs scheduled to faulty elevators cannot be completed- the  throughput of jobs, and by extension amount of passengers processed is affected.

## 6.0 Conclusion

TA feedback was especially important for us when making updates to our iterations.
One thing that we could have done better was testing. We often found ourselves with little time after completing the requirements for each iteration but did not have enough time to create meaningful JUnit tests for our system. This would have saved us time for the final submission as we had to write tests for past iterations when they should have already been completed. Better time-management could have alleviated this problem.

Overall, we are satisfied with the achieved system and found this project to be very rewarding.