SYSC 3303 Term Project Report - L4 Group 6

GitHub: https://github.com/AshtonMohns/elevatorTiming

Demo Video: https://www.youtube.com/watch?v=iBuNZKiITTE&feature=youtu.be

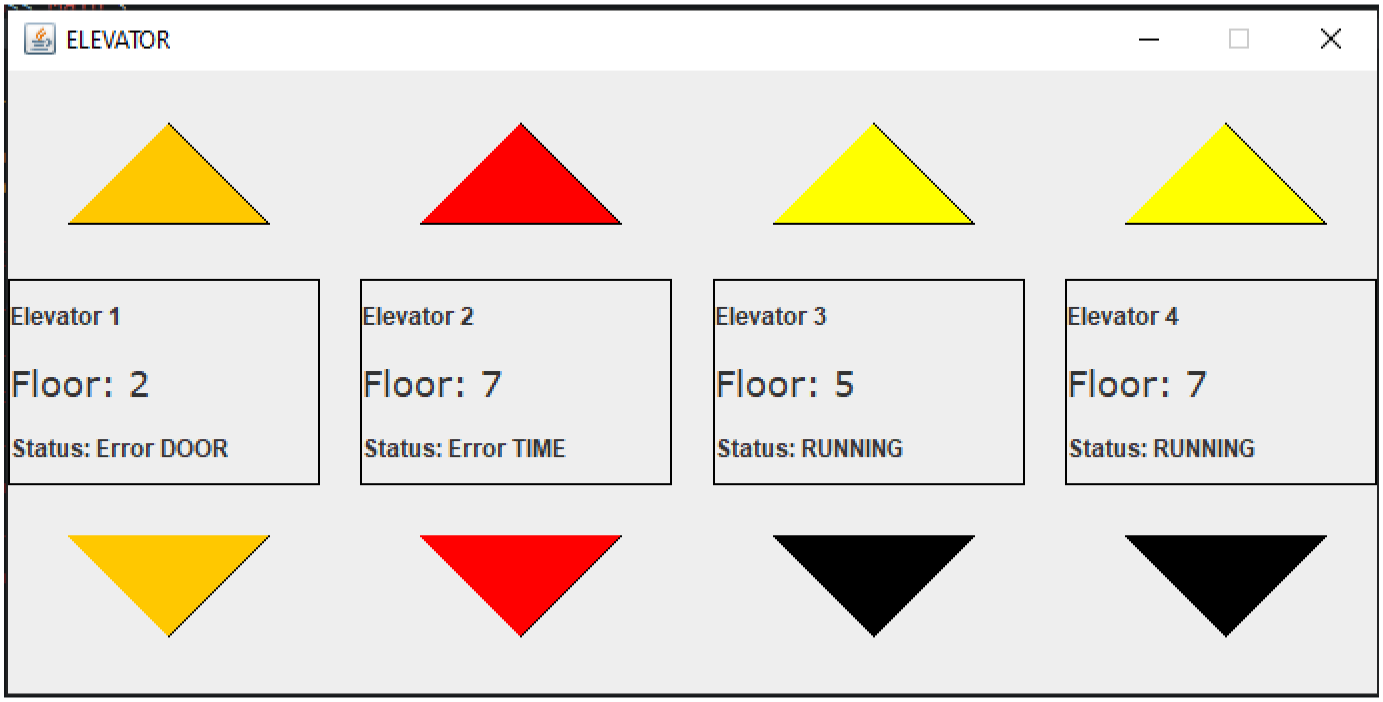
Recipient: TA for SYSC 3303

Client: Roy Franks of Department of Systems and Computer Engineering Class of SYSC 3303

Authors:

1. Ashton Mohns: 101074479
2. Edmond Chow: 100883365
3. Jyotsna Mahesh: 101084851
4. Asifur Rahman: 101069183
5. Md Aiman Sharif: 101062765

Date: April 14, 2021



Elevator 1
Floor: 2
Status: Error DOOR

Elevator 2
Floor: 7
Status: Error TIME

Elevator 3
Floor: 5
Status: RUNNING

Elevator 4
Floor: 7
Status: RUNNING

Table of Contents

1.0 BREAKDOWN OF RESPONSIBILITIES:

1.1 ITERATION 1:

Ashton Mohns:
- Floor Subsystem and their related classes

Edmond Chow:
- Scheduler and Elevator Subsystem and their related classes

Jyotsna Mahesh:
- UML and Sequence Documentation

Asifur Rahman:
- UML, README and Testing

Md Aiman Sharif:
- Testing framework

1.2 ITERATION 2:

Ashton Mohns:
- Thread3 Class Creation
- Map Class Creation

Edmond Chow:
- Thread2 Class Creation
- Map Class Creation
- ReqBox implementation for Schedular Subsystem

Jyotsna Mahesh:
- Floor Subsystem Creation
- Thread1 Class Creation
- ReqBox implementation for Floor Subsystem

Asifur Rahman:
- ElevatorSubSystem Creation
- Updating README file

Md Aiman Sharif:
- Drawing UML Class Diagram
- JUnit Testing

1.3 ITERATION 3:

Ashton Mohns:

- Drawing Sequence Diagram
- Update and refactor code for Floor subsystem

Edmond Chow:

- JUnit testing
- Update threads and code for Elevator subsystem

Jyotsna Mahesh:

- Remodeling Floor Subsystem
- Drawing State Machines Diagram

Asifur Rahman:

- Remodeling of the ElevatorSubSystem
- Updating README file

Md Aiman Sharif:

- Drawing UML Class Diagram
- Updating Javadoc
- Updating Floor Manager's Implementation

1.4 ITERATION 4:

Ashton Mohns:

- Implementing an intermediate host for forwarding door closes from floor to elevator subsystem
- Updating tests with the new request packets
- Helping with state diagrams

Edmond Chow:

- Debug testing
- Elevator Rebuild, Elevator Fault Handling
- Debug setting for console output

Jyotsna Mahesh:

- Upgrade file reader and Direction Enum
- Create Error Enum
- Update Print statements
- Documentation (sequence and state diagrams)

Asifur Rahman:

- Implementing ElevatorThread4 class for handling door close/open functionality
- Upgrading ElevatorBox for error handling
- Updating README file
- Modifying RequestHandler to meet the current criteria

Md Aiman Sharif:

- Floor door handling
- Updating packets sent by the floor subsystem to handle errors
- Updating Class diagrams

1.5 ITERATION 5:

Ashton Mohns:

- GUI
- Acceptance testing
- Reflections

Edmond Chow:

- Documentation
- Integration Testing
- Refactor code, remove bugs
- Error handling unit tests

Jyotsna Mahesh:

- Created more Test Files
- Diagrams: Sequence, State Machine and timing
- Worked on the Report

Asifur Rahman:

- Creating and maintaining the report
- Diagrams: UML, Sequence
- Updating README file

Md Aiman Sharif:

- Designed the GUI- code
- Implemented GUI endpoints
- User Interface testing
- Worked on report

2.0 DIAGRAMS:

2.1 UML Class Diagram:

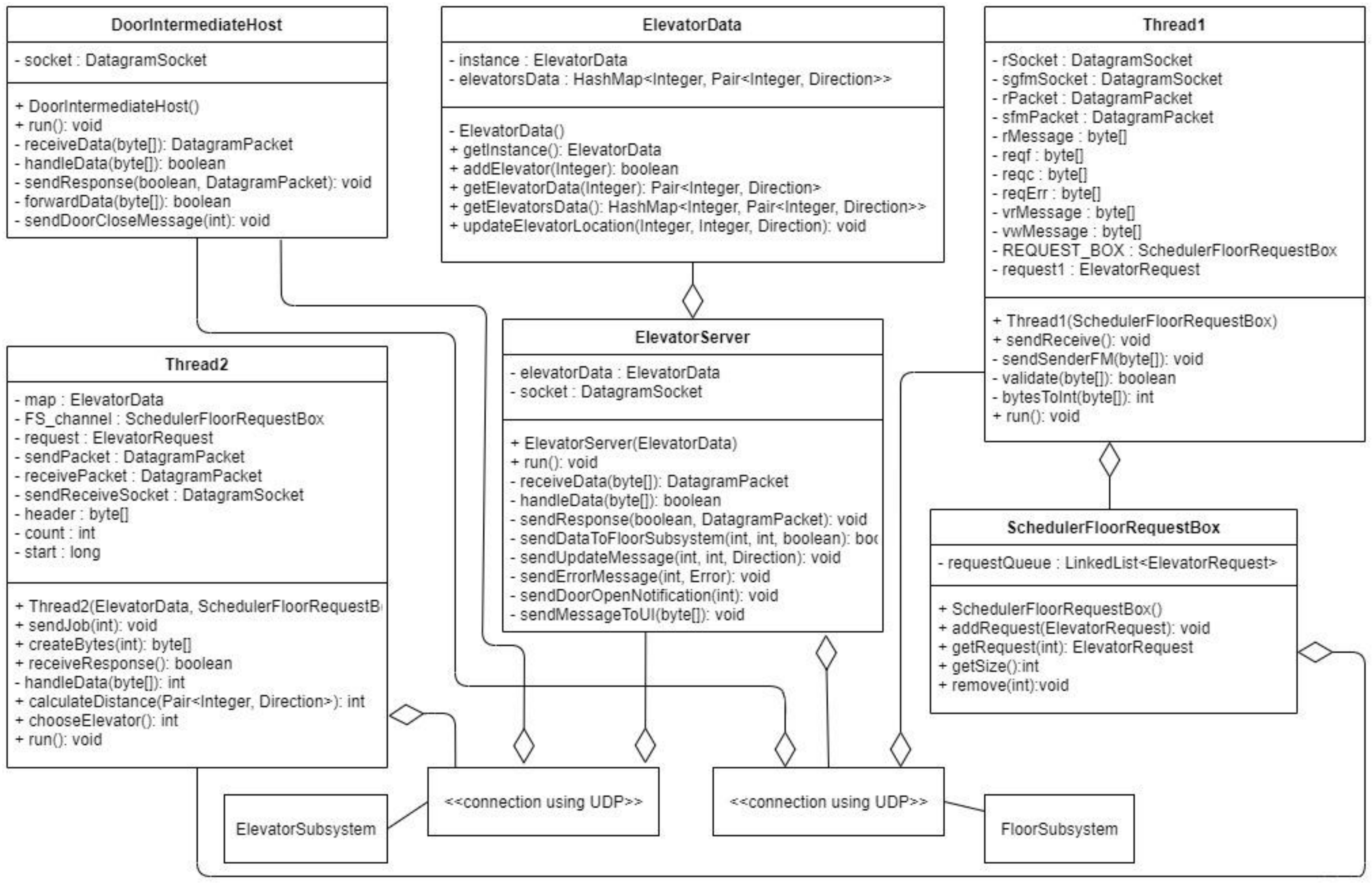

DoorIntermediateHost
- socket : DatagramSocket
+ DoorIntermediateHost()
+ run(): void
- receiveData(byte[]): DatagramPacket
- handleData(byte[]): boolean
- sendResponse(boolean, DatagramPacket): void
- forwardData(byte[]): boolean
- sendDoorCloseMessage(int): void
ElevatorData
- instance : ElevatorData
- elevatorsData : HashMap<Integer, Pair<Integer, Direction>>
- ElevatorData()
+ getInstance(): ElevatorData
+ addElevator(Integer): boolean
+ getElevatorData(Integer): Pair<Integer, Direction>
+ getElevatorsData(): HashMap<Integer, Pair<Integer, Direction>>
+ updateElevatorLocation(Integer, Integer, Direction): void
Thread1
- rSocket : DatagramSocket
- sgfmSocket : DatagramSocket
- rPacket : DatagramPacket
- sfmPacket : DatagramPacket
- rMessage : byte[]
- reqf : byte[]
- reqc : byte[]
- reqErr : byte[]
- vrMessage : byte[]
- vwMessage : byte[]
- REQUEST_BOX : SchedulerFloorRequestBox
- request1 : ElevatorRequest
+ Thread1(SchedulerFloorRequestBox)
+ sendReceive(): void
- sendSenderFM(byte[]): void
- validate(byte[]): boolean
- bytesToInt(byte[]): int
+ run(): void
ElevatorServer
- elevatorData : ElevatorData
- socket : DatagramSocket
+ ElevatorServer(ElevatorData)
+ run(): void
- receiveData(byte[]): DatagramPacket
- handleData(byte[]): boolean
- sendResponse(boolean, DatagramPacket): void
- sendDataToFloorSubsystem(int, int, boolean): boo
- sendUpdateMessage(int, int, Direction): void
- sendErrorMessage(int, Error): void
- sendDoorOpenNotification(int): void
- sendMessageToUI(byte[]): void
Thread2
- map : ElevatorData
- FS_channel : SchedulerFloorRequestBox
- request : ElevatorRequest
- sendPacket : DatagramPacket
- receivePacket : DatagramPacket
- sendReceiveSocket : DatagramSocket
- header : byte[]
- count : int
- start : long
+ Thread2(ElevatorData, SchedulerFloorRequestB
+ sendJob(int): void
+ createBytes(int): byte[]
+ receiveResponse(): boolean
- handleData(byte[]): int
+ calculateDistance(Pair<Integer, Direction>): int
+ chooseElevator(): int
+ run(): void
SchedulerFloorRequestBox
- requestQueue : LinkedList<ElevatorRequest>
+ SchedulerFloorRequestBox()
+ addRequest(ElevatorRequest): void
+ getRequest(int): ElevatorRequest
+ getSize():int
+ remove(int):void
<<connection using UDP>>
<<connection using UDP>>
ElevatorSubsystem
FloorSubsystem

Figure 1: UML Class Diagram for Scheduler Subsystem

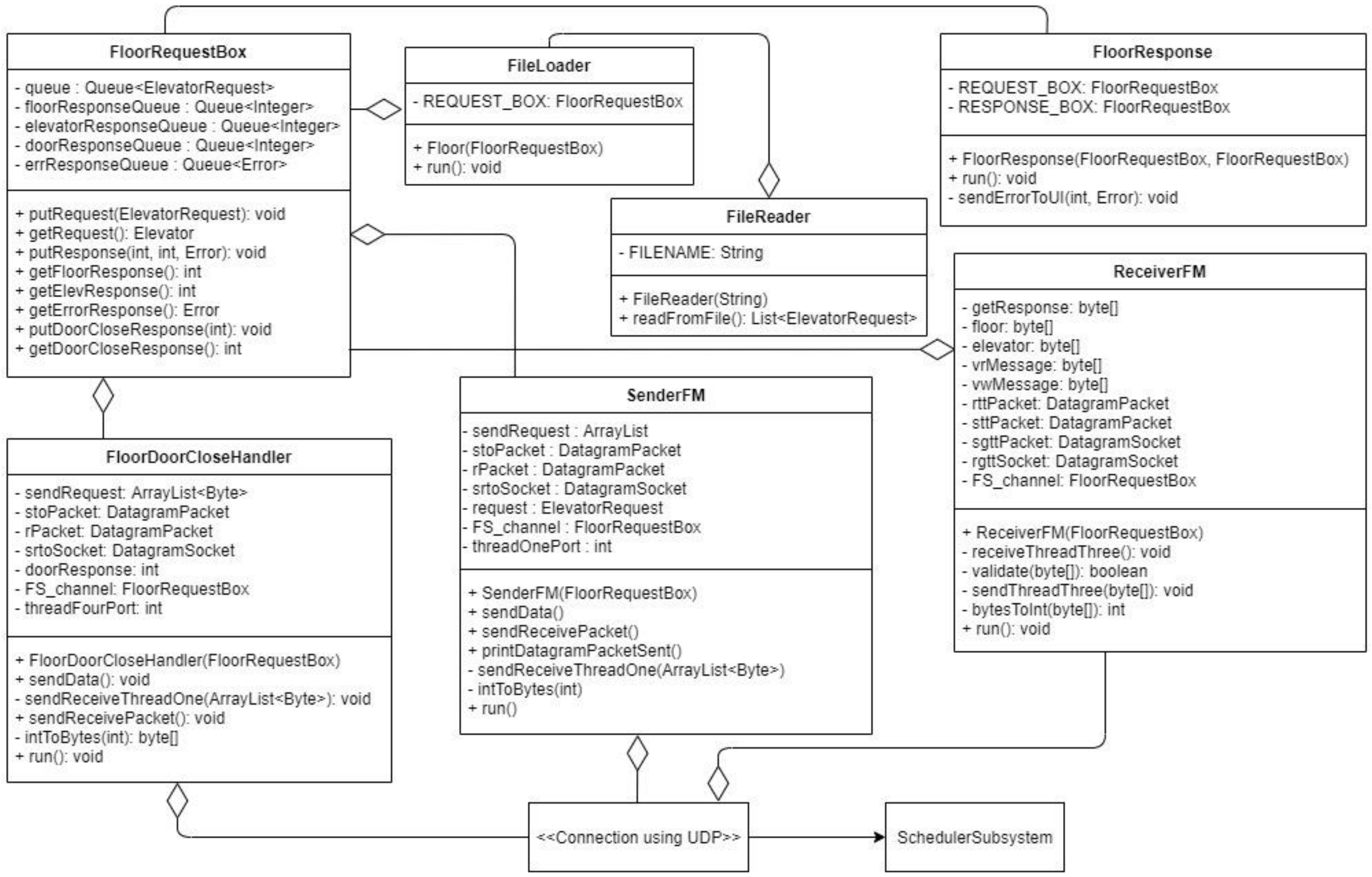

FloorRequestBox
- queue : Queue<ElevatorRequest>
- floorResponseQueue : Queue<Integer>
- elevatorResponseQueue : Queue<Integer>
- doorResponseQueue : Queue<Integer>
- errResponseQueue : Queue<Error>
+ putRequest(ElevatorRequest): void
+ getRequest(): Elevator
+ putResponse(int, int, Error): void
+ getFloorResponse(): int
+ getElevResponse(): int
+ getErrorResponse(): Error
+ putDoorCloseResponse(int): void
+ getDoorCloseResponse(): int
FileLoader
- REQUEST_BOX: FloorRequestBox
+ Floor(FloorRequestBox)
+ run(): void
FloorResponse
- REQUEST_BOX: FloorRequestBox
- RESPONSE_BOX: FloorRequestBox
+ FloorResponse(FloorRequestBox, FloorRequestBox)
+ run(): void
- sendErrorToUI(int, Error): void
FileReader
- FILENAME: String
+ FileReader(String)
+ readFromFile(): List<ElevatorRequest>
ReceiverFM
- getResponse: byte[]
- floor: byte[]
- elevator: byte[]
- vrMessage: byte[]
- vwMessage: byte[]
- rttPacket: DatagramPacket
- sttPacket: DatagramPacket
- sgttPacket: DatagramSocket
- rgttSocket: DatagramSocket
- FS_channel: FloorRequestBox
+ ReceiverFM(FloorRequestBox)
- receiveThreadThree(): void
- validate(byte[]): boolean
- sendThreadThree(byte[]): void
- bytesToInt(byte[]): int
+ run(): void
SenderFM
- sendRequest : ArrayList
- stoPacket : DatagramPacket
- rPacket : DatagramPacket
- srtoSocket : DatagramSocket
- request : ElevatorRequest
- FS_channel : FloorRequestBox
- threadOnePort : int
+ SenderFM(FloorRequestBox)
+ sendData()
+ sendReceivePacket()
+ printDatagramPacketSent()
- sendReceiveThreadOne(ArrayList<Byte>)
- intToBytes(int)
+ run()
FloorDoorCloseHandler
- sendRequest: ArrayList<Byte>
- stoPacket: DatagramPacket
- rPacket: DatagramPacket
- srtoSocket: DatagramSocket
- doorResponse: int
- FS_channel: FloorRequestBox
- threadFourPort: int
+ FloorDoorCloseHandler(FloorRequestBox)
+ sendData(): void
- sendReceiveThreadOne(ArrayList<Byte>): void
+ sendReceivePacket(): void
- intToBytes(int): byte[]
+ run(): void
<<Connection using UDP>>
SchedulerSubsystem

Figure 2: UML Class Diagram for the Floor Subsystem

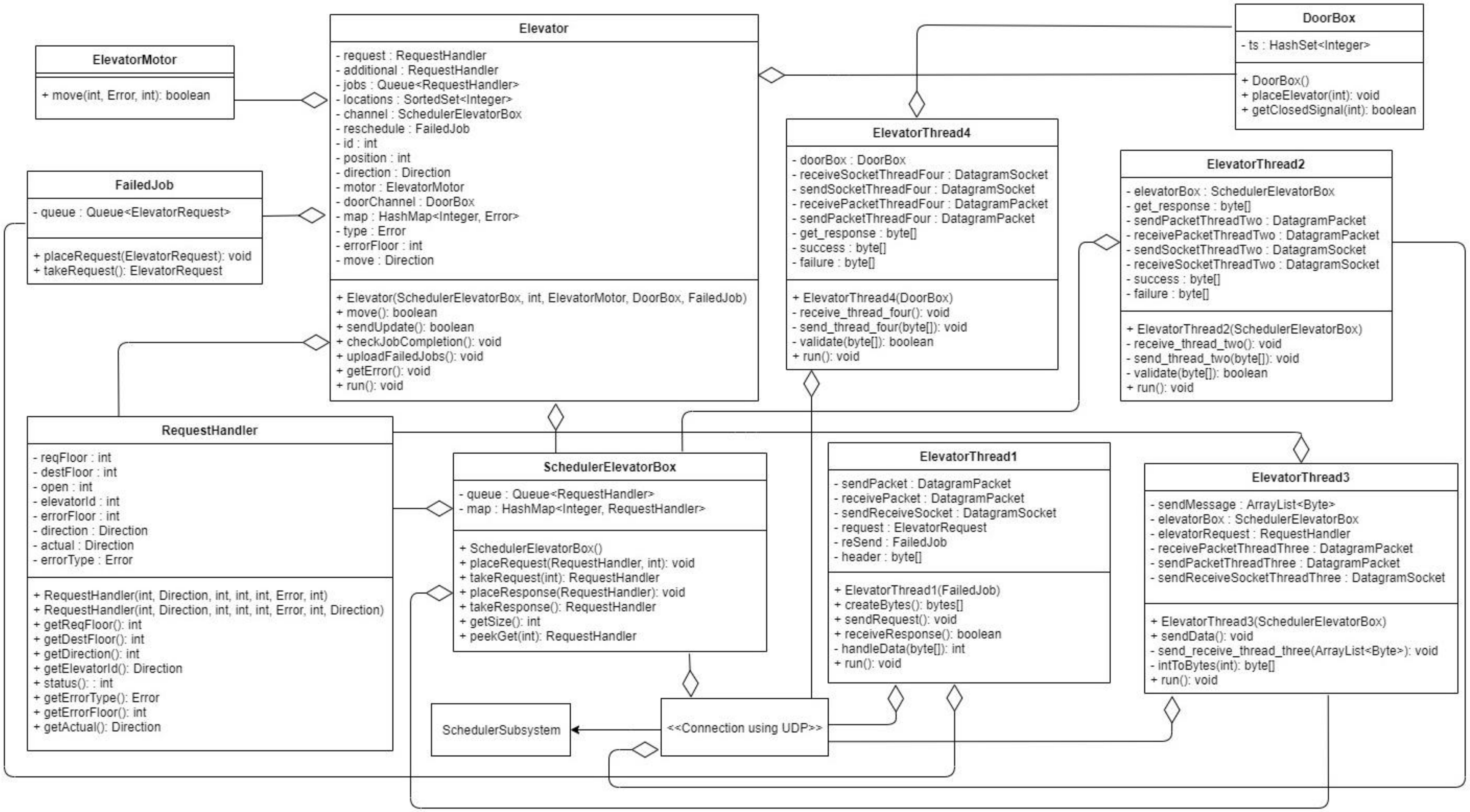

ElevatorMotor
+ move(int, Error, int): boolean
Elevator
- request : RequestHandler
- additional : RequestHandler
- jobs : Queue<RequestHandler>
- locations : SortedSet<Integer>
- channel : SchedulerElevatorBox
- reschedule : FailedJob
- id : int
- position : int
- direction : Direction
- motor : ElevatorMotor
- doorChannel : DoorBox
- map : HashMap<Integer, Error>
- type : Error
- errorFloor : int
- move : Direction
+ Elevator(SchedulerElevatorBox, int, ElevatorMotor, DoorBox, FailedJob)
+ move(): boolean
+ sendUpdate(): boolean
+ checkJobCompletion(): void
+ uploadFailedJobs(): void
+ getError(): void
+ run(): void
DoorBox
- ts : HashSet<Integer>
+ DoorBox()
+ placeElevator(int): void
+ getClosedSignal(int): boolean
FailedJob
- queue : Queue<ElevatorRequest>
+ placeRequest(ElevatorRequest): void
+ takeRequest(): ElevatorRequest
ElevatorThread4
- doorBox : DoorBox
- receiveSocketThreadFour : DatagramSocket
- sendSocketThreadFour : DatagramSocket
- receivePacketThreadFour : DatagramPacket
- sendPacketThreadFour : DatagramPacket
- get_response : byte[]
- success : byte[]
- failure : byte[]
+ ElevatorThread4(DoorBox)
- receive_thread_four(): void
- send_thread_four(byte[]): void
- validate(byte[]): boolean
+ run(): void
ElevatorThread2
- elevatorBox : SchedulerElevatorBox
- get_response : byte[]
- sendPacketThreadTwo : DatagramPacket
- receivePacketThreadTwo : DatagramPacket
- sendSocketThreadTwo : DatagramSocket
- receiveSocketThreadTwo : DatagramSocket
- success : byte[]
- failure : byte[]
+ ElevatorThread2(SchedulerElevatorBox)
- receive_thread_two(): void
- send_thread_two(byte[]): void
- validate(byte[]): boolean
+ run(): void
RequestHandler
- reqFloor : int
- destFloor : int
- open : int
- elevatorId : int
- errorFloor : int
- direction : Direction
- actual : Direction
- errorType : Error
+ RequestHandler(int, Direction, int, int, int, Error, int)
+ RequestHandler(int, Direction, int, int, int, Error, int, Direction)
+ getReqFloor(): int
+ getDestFloor(): int
+ getDirection(): int
+ getElevatorId(): Direction
+ status(): : int
+ getErrorType(): Error
+ getErrorFloor(): int
+ getActual(): Direction
SchedulerElevatorBox
- queue : Queue<RequestHandler>
- map : HashMap<Integer, RequestHandler>
+ SchedulerElevatorBox()
+ placeRequest(RequestHandler, int): void
+ takeRequest(int): RequestHandler
+ placeResponse(RequestHandler): void
+ takeResponse(): RequestHandler
+ getSize(): int
+ peekGet(int): RequestHandler
ElevatorThread1
- sendPacket : DatagramPacket
- receivePacket : DatagramPacket
- sendReceiveSocket : DatagramSocket
- request : ElevatorRequest
- reSend : FailedJob
- header : byte[]
+ ElevatorThread1(FailedJob)
+ createBytes(): bytes[]
+ sendRequest(): void
+ receiveResponse(): boolean
- handleData(byte[]): int
+ run(): void
ElevatorThread3
- sendMessage : ArrayList<Byte>
- elevatorBox : SchedulerElevatorBox
- elevatorRequest : RequestHandler
- receivePacketThreadThree : DatagramPacket
- sendPacketThreadThree : DatagramPacket
- sendReceiveSocketThreadThree : DatagramSocket
+ ElevatorThread3(SchedulerElevatorBox)
+ sendData(): void
- send_receive_thread_three(ArrayList<Byte>): void
- intToBytes(int): byte[]
+ run(): void
SchedulerSubsystem
<<Connection using UDP>>

Figure 3: UML Class Diagram for Elevator Subsystem

2.2 State Machine Diagram:

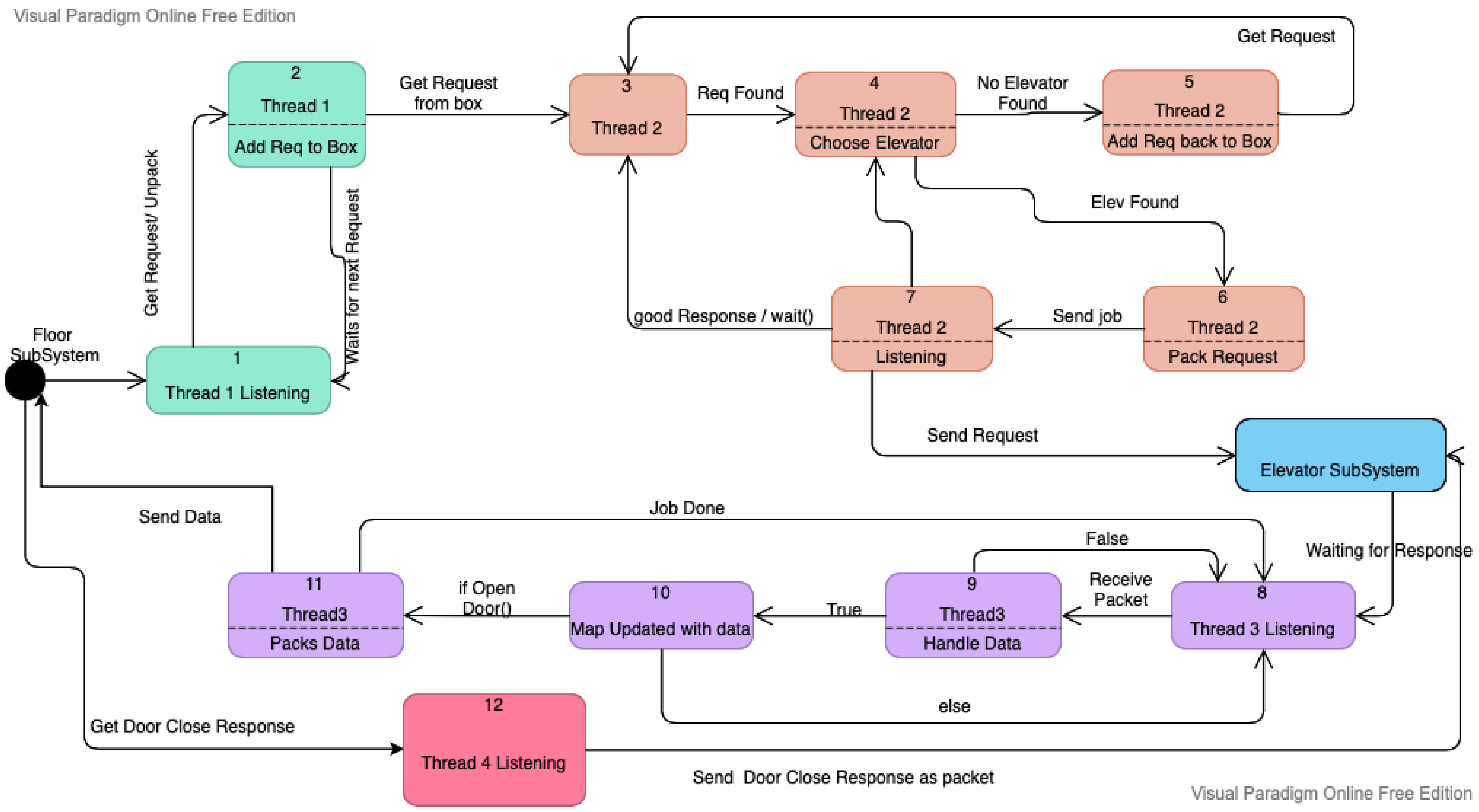
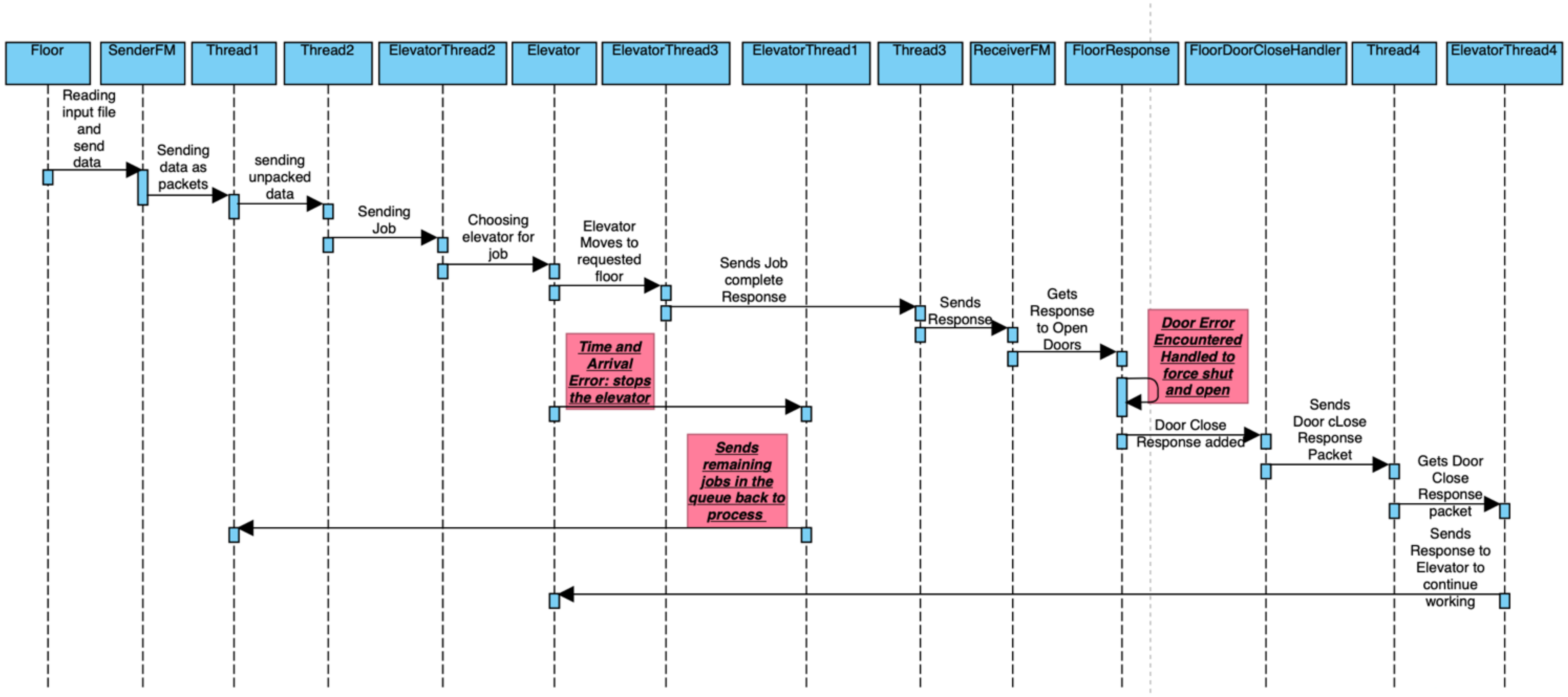


Figure 4: State Machine Diagram for Scheduler Subsystem

2.3 Sequence Diagram:



Floor
SenderFM
Thread1
Thread2
ElevatorThread2
Elevator
ElevatorThread3
ElevatorThread1
Thread3
ReceiverFM
FloorResponse
FloorDoorCloseHandler
Thread4
ElevatorThread4
Reading input file and send data
Sending data as packets
sending unpacked data
Sending Job
Choosing elevator for job
Elevator Moves to requested floor
Sends Job complete Response
Sends Response
Gets Response to Open Doors
Door Error Encountered Handled to force shut and open
Time and Arrival Error: stops the elevator
Door Close Response added
Sends Door cLose Response Packet
Gets Door Close Response packet
Sends remaining jobs in the queue back to process
Sends Response to Elevator to continue working

Figure 5: Sequence Diagram showing error handling in all three subsystems

2.4 Timing Diagram:

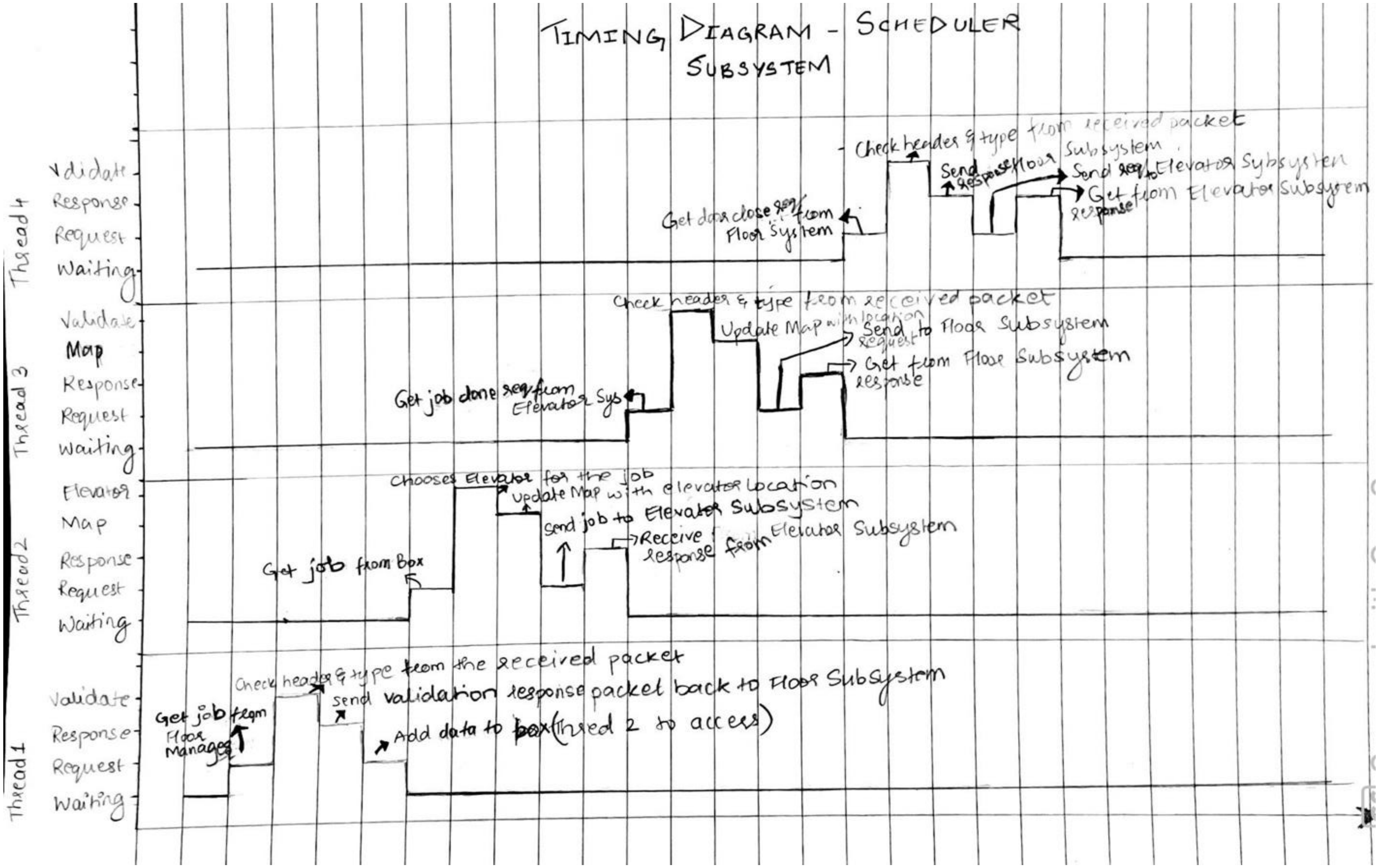

TIMING DIAGRAM - SCHEDULER SUBSYSTEM
Thread 4
Validate
Response
Request
Waiting
Get data close req from Floor System
Check headers & type from received packet
Send response Floor Subsystem
Send req to Elevator Subsystem
Get response from Elevator Subsystem
Thread 3
Validate
Map
Response
Request
Waiting
Get job done req from Elevator Sys
Check header & type from received packet
Update Map with location
Send request to Floor Subsystem
Get response from Floor Subsystem
Thread 2
Elevator
Map
Response
Request
Waiting
Get job from Box
Chooses Elevator for the job
Update Map with elevator location
Send job to Elevator Subsystem
Receive response from Elevator Subsystem
Thread 1
Validate
Response
Request
Waiting
Get job from Floor Manager
Check header & type from the received packet
Send validation response packet back to Floor Subsystem
Add data to box (thread 2 to access)

Figure 6: Timing Diagram for Scheduler Threads

[Was informed by the prof that exact timings were not required]

3.0 DETAILED SET-UP AND TEST INSTRUCTIONS:

3.1 SETUP AND FORMATS:

3.1.1 SETUP INSTRUCTIONS:

1. **Import the project archive file into Eclipse**
2. **From Eclipse**
   - ✓ Select "File"
   - ✓ Select "Import"
   - ✓ Select "General"
   - ✓ Select "Projects from File System or Archive File" under "General"
   - ✓ Select "Archive..."
   - ✓ Locate "L4G6_Milestone5.zip" as Import Source
   - ✓ Click "Finish"
3. **From within the project open "ElevatorTiming"**

Before starting the code running, if you wish to change the times for elevators to move and/or doors to open, you can change the values in the sharedObjects/Constants.java file. Both values are time in milliseconds. Please do not reduce too close to 0, as there are potential dropped packets in this case. If you encounter any errors while running the code, please 'clean' and 'refresh' the Project. In order to run the program, you can follow one of below listed set of steps:

3.1 The main files for each subsystem can be run individually and have the outputs be shown on separate consoles.
Run the files in this order:
   i. Run the UserInterfaceMain.java file from the main package.
   ii. Run the SchedulerMain.java from the main package.
   iii. Run the MainEM.java file from the main package.
   iv. Run the MainFM.java file from the main package.

3.2 The main files can be run simultaneously with all their outputs being shown on the same console.
Run the Main.java file from the main package.

3.1.2 Input.txt format:

Time	RequestedFloor	Direction	DestinationFloor	ErrorType	ErrorFloor

Time: when the request was sent

RequestedFloor: use up to 22 floors

Direction: "Up" or "Down" inputs only [Case does not matter]

DestinationFLoor: use up to 22 floors

ErrorType: "NONE", "ARRIVAL", "TIME", "DOOR" inputs only [Case does not matter]

ErrorFloor: floor where you want the error to happen

3.1.3 Using Constants.java to modify behavior:

A wide variety of factors in behavior of the system can be edited through manipulation of variables in the constants.java file as listed below

- **Boolean debug:** when debug is false, the system will output less information to the consoles to increase readability in general cases. When debug is set to true, more information will be outputted to consoles for troubleshooting.
- **Int TIMEOUT_MILLIS**: sets how long is required for a socket time out to occur, not extremely low time out values will cause system to fail
- **Int elevator:** change to adjust the number of elevators that will be running in the system
- **Int MOVE_TIME:** This value is used to adjust how long it will take an elevator to move between floors
- **Int DOOR_TIME:** This value is used to control how long a door will stay open when they open
- **String FILENAME:** This will be the name of the input file read by the system

Int FLOOR_PORT, Int ELEVATOR_SERVER_PORT, and **Int HEADER** should not be edited as they are constants used by the system for regular operations.

3.2 TEST INSTRUCTIONS

3.2.1 Unit Testing:

- The **ElevatorThreadTests**, **ElevatorServerTest**, **SenderFMTest**, **ReceiverFMTest**, **Thread1Test** are to test the threads functionalities individually.
- The **FloorRequestBoxTest**, **SchedulerElevatorBoxTest**, **SchedulerFloorRequestBoxTest**, **DoorBoxTest** are to test if the functionalities of the box classes are working as intended.
- **ElevatorFaultTest** checks if the system can handle timer and arrival sensor faults.
- **FloorResponseTest** Checks if the system can properly handle and clear door errors
- **ElevatorMotorTest** ensures that the movement of elevator is performed as expected.
- **ElevatorRequestTest** checks if the elevator request objects handle the data properly.
- **FailedJobTest** checks if the failed process can be reprocessed by the scheduler.
- **FloorDoorCloseHandlerTest** checks if the thread sending the packets properly in the right manner.

Upon writing the unit tests, minor bugs were found and patched in the code. These included inconsistencies in handling requests and slower response times than expected. We have since patched the bugs and the unit tests succeed.

3.2.2 Integration Testing:

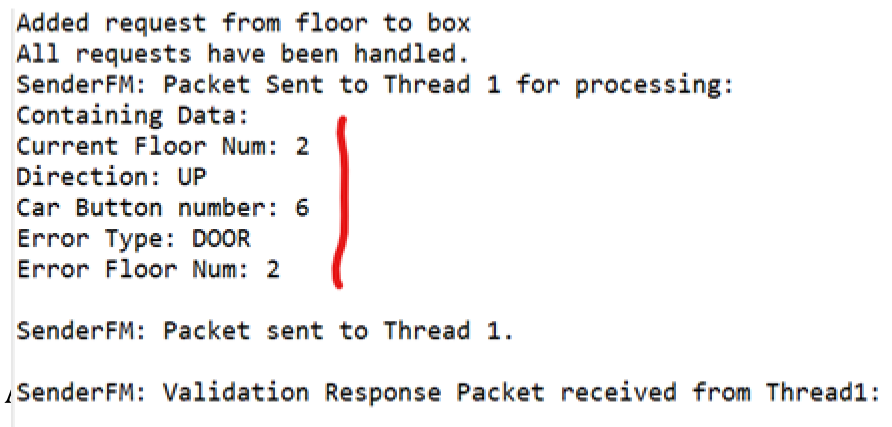
No actual testing file was made to monitor integration testing. In order to test the integration between the various subsystem components the following steps were done:

1. In the Constants.java file, set the Boolean variable debug to true, to enable
2. Run the program as normal.

3. At the end of the run of the program you can examine the console logs of the 3 subsystems.
4. To see how connections between components work, one can compare the outgoing packet logs, to the incoming packet logs for the adjacent subsystem

For example, to test that the floor subsystem is properly sending requests to the scheduler

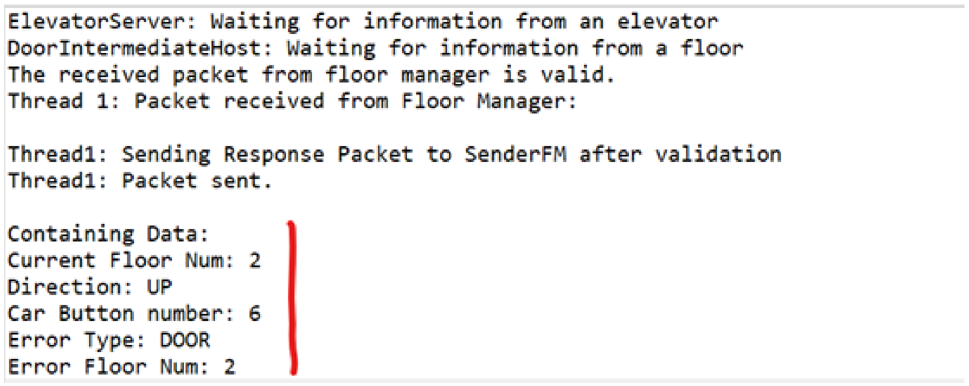
You can look at the values in the packet being sent by the floor subsystem



```
Added request from floor to box
All requests have been handled.
SenderFM: Packet Sent to Thread 1 for processing:
Containing Data:
Current Floor Num: 2
Direction: UP
Car Button number: 6
Error Type: DOOR
Error Floor Num: 2

SenderFM: Packet sent to Thread 1.

SenderFM: Validation Response Packet received from Thread1:
```



```
ElevatorServer: Waiting for information from an elevator
DoorIntermediateHost: Waiting for information from a floor
The received packet from floor manager is valid.
Thread 1: Packet received from Floor Manager:

Thread1: Sending Response Packet to SenderFM after validation
Thread1: Packet sent.

Containing Data:
Current Floor Num: 2
Direction: UP
Car Button number: 6
Error Type: DOOR
Error Floor Num: 2
```

And if the data sent out by the floor matches what the scheduler reads in, then we know that the integration between the two components is successful.

Another tool we have to use in making sure that integration between subsystems is the packet validation for every packet sent between subsystems. The packet validation allows us to see at a glance if a packet that was sent between subsystems is being constructed improperly allowing faster validation. The packet validation was done by comparing the format of the packet in matched the pre-agreed format for that type of packet.

3.2.3 Acceptance Testing:

For acceptance testing, we went and reviewed the UI and corresponding console outputs to ensure the following; first, that the elevator requests are being distributed logically; and second, that all requests are being handled correctly by both the backend and the UI.

Initially, the jobs are distributed as per table 1 below. The jobs are distributed initially to available elevators, then assigned based on the current location and direction of the elevators. The first request in ELEVATOR 1 is a door error, which can be recovered from. This error causes a recoverable fault, so jobs are not allocated, but are forced to wait.

The second elevator also has an error, but this is an elevator timer error, which is a hard fault. When this occurs, the elevator is shut down since it cannot recover. When this happens, the jobs from ELEVATOR 2 are sent back to the scheduler subsystem to be reassigned. The scheduler subsystem looks at the updated location and directions of the elevators and reassigns them as per Table 2 below.

When looking at the UI, we can see the doors open on each floor required for the elevators to handle the jobs assigned to it. We can also see the UI light up to display both the expected faults. Similarly, we can see the output logs from both the elevators and floor subsystems to match the expected results for each of the elevators. Because of these factors, the acceptance testing has proven the code works in an acceptable manner for the UI and overall functionality of the system.

Table 1: Initial job distribution

ELEVATOR 1	**ELEVATOR 2**	**ELEVATOR 3**	**ELEVATOR 4**
14:05:15.000 2 Up 20 Door 2	14:10:15.000 16 Down 2 Time 7	14:15:15.000 1 Up 19 None 0	14:20:15.000 19 Down 1 None 0
14:25:15.000 1 Up 16 None 0			
14:35:15.000 2 Up 8 None 0			

Table 2: Job redistribution after hard fault

ELEVATOR 1	**ELEVATOR 2**	**ELEVATOR 3**	**ELEVATOR 4**
14:05:15.000 2 Up 20 Door 2	FAULT - STOPPED	14:15:15.000 1 Up 19 None 0	14:20:15.000 19 Down 1 None 0
14:25:15.000 1 Up 16 None 0			14:10:15.000 16 Down 2 None 7
14:35:15.000 2 Up 8 None 0			14:30:15.000 17 Down 2 None 0
			14:40:15.000 9 Down 1 None 0

4.0 RESULTS FROM MEASUREMENT:

Door opening & closing time: $\frac{6.2+9.7+10.6+10.7+10.2+11+8.2+8+9.9+11+7.8}{11} = \frac{103.3}{11}$ = 9.39 seconds

All floors:

v1 = $\frac{28\ m}{17.6\ s}$ = 1.59 m/s

v2 = v3 = $\frac{28\ m}{19.6\ s}$ = 1.428 m/s

v4 = $\frac{28\ m}{22.5\ s}$ = 1.244 m/s

Average Velocity = 1.4225 m/s

5.0 REFLECTION ON DESIGN:

Overall, this project went well. There were portions where we attempted to get ahead of ourselves with the work. For example, in Milestone 2, we attempted to refactor the project to use UDP rather than boxes, which would have been the requirement for Milestone 3. This ended up making a lot of excess work on ourselves and we ended up having to submit without properly having the whole code working.

If we had more time to redo portions, we would likely work on redoing the elevator classes, as they became fairly jumbled over the course of a few milestones in order to get them working by the deadlines. The UI turned out decently, as it runs asynchronously through remote procedure calls from the scheduler subsystem. It properly displays the current location and status of each elevator, though is partially hard-coded due to the minimal time for getting Milestone 5 running before the demonstration. The scheduler logic turned out decently for distributing jobs, but it was tougher to create than we would have liked due to inconsistent information regarding how new jobs should be handled (i.e., if they should be automatically given to stopped elevators or just the closest elevator even if it is busy, etc.).

It has been a great group to work with, even though the many stressful weeks of getting code complete. Thankfully, we have not faced any problems with any of the group members not coordinating or cooperating to get the work done. Everyone was cooperative and worked hard to get each of the project iterations done successfully and put in their full time and effort to get it done as a group. Overall, this project could have gone better if we had more time, but we believe that we put in our absolute best effort in this project. Thank you for taking the time to read our final project report.