SYSC 3303B Group Project – G1

Measuring and Simulating a Real Elevator

Team Members:

Capstick, Ryan - 101239778

Chukwuma, Nkechi - 101230684

Keertani, Aditi Manjunath - 101202033

Ogidi-Gbegbaje, Jennifer -101209061

Olowookere, Oluwatobi - 101245900

Table of Contents

Breakdown of Responsibilities:

SYSC3303_Group_Project

Group 1 Members

Capstick, Ryan - 101239778

Chukwuma, Nkechi - 101230684

Keertani, Aditi Manjunath - 101202033

Ogidi-Gbegbaje, Jennifer -101209061

Olowookere, Oluwatobi - 101245900

Project Iteration 5 – Measuring Timing of Multiple Real Elevators with Error Detection and Correction and User Interface

PURPOSE

The purpose of this project is to design and implement an elevator control system and simulator using three different subsystems: Elevator, Floor and Scheduler. Each subsystem is coded as a separate 32Win process/ program that can be opened as a project in IntelliJ, and all three subsystems communicate with each user using DatagramSocket objects. This project is an insight into real-time communication between the three different subsystems in order to make the elevator work, using a scheduler. The Floor subsystem reads the input file which contains the information about arrival of passengers and all button presses and lamps and sends it across to the Scheduler. The scheduler will then communicate that the elevator has acknowledged the request. The elevator is responsible for making calls to the Scheduler while it is stationary to check for any pending requests. In case of pending requests, the elevator subsystem them handles the request and then sends and acknowledgement that the request has been received to the Scheduler to pass back to the Floor subsystem. The program terminates when the Floor subsystem has handled and completed all events in the input file and received an acknowledgement for the last event as well.

In this iteration, we accounted for more elevators being able to run through the simulation and introduced a state machine that is responsible for the different states the elevator goes into throughout its runtime. Additionally, we added code for detecting and handling faults. For example, we added timing events so that if the timer goes off before an elevator reaches a floor, then our system should assume a fault (either, the elevator is stuck between floors, or the arrival sensor at a floor has failed). In this iteration, we added a user interface to make the user experience smoother and more convenient. We used JFrame to integrate this into existing code and ensured that the simulation matched up as accurately as possible to our experimental data.

-BREAKDOWN OF RESPONSIBILITIES

The folder 'Diagrams' containing all required UML diagrams, timing diagram, sequence diagrams and state diagrams, as well as JUnit test files: ElevatorTest.java, FloorTest.java, SchedulerTest.java - Jennifer, Nkechi

Floor.java, FloorControl.java - Oluwatobi, Ryan

ElevatorGUI.java, Utilities.java - Nkechi, Jennifer

ElevatorConfig.java, Host.java, Scheduler.java, UtilTest.java - Oluwatobi, Ryan, Aditi

AllTest.java - Oluwatobi

Readme.txt and Report- Aditi

-DESCRIPTION OF FILES WITH FILE NAMES

ElevatorGUI.java – This is the class with the graphical user interface made for the user to interact with the system, without directly accessing the code or any of the threads/processes. It creates a simple user experience for the common man to easily understand and operate the simulation with.

ElevatorConfig.java - This is the main thread/class responsible for controlling the working of the elevator, by responding to the requests sent by the floor subsystem via the Scheduler. It uses DatagramPackets in order to communicate with the other two subsystems, and handles the requests sent by the user in order for the elevator to reach the destination floor selected by the user. It also initializes the Floor subsystem(client), Elevator subsystem(client) and the Scheduler(server) threads and then starts them.

ElevatorTest.java - This is a JUnit test class that tests all the methods written in the files from the Elevator Subsystem, in order to see whether the subsystem is able to function efficiently.

Floor.java - This thread is responsible for the Floor Subsystem containing the floor number, number of elevators, lamp status to indicate whether the elevator is going up or down and if the floor is an upper or lower floor.

FloorControl.java -This is the main class responsible for the control of the elevator with respect to requests sent via the scheduler, and it also processes the elevator's departure messages. It ensures that the floor subsystem waits after sending a service request to the Scheduler, in order to receive an acknowledgement.

FloorTest.java - This is a jUnit test class that tests all the methods of the Floor Subsystem.

Host.java - This class acts as the server and contains information about the ports that are not meant to be used/ which must be cleared before running the main project. It also contains the DatagramSocket information necessary for communication between the 3 different subsystems, and helps to control movements of the elevator via the scheduler by actively taking part in the request handling process.

Scheduler.java - This is the thread responsible for controlling the scheduler subsystem and the main point of contact between the Floor Subsystem and the Elevator Subsystem.

It is used for submitting new requests to the Scheduler from the Floor Subsystem, and then communicating the same to the elevator in order for the elevator to then take the necessary steps needed to reach the destination floor of the user's request. The scheduler is responsible for accepting input from all of the sensors, and sending indications (to devices such as lamps) and commands (to devices such as the motor and door). It is responsible for routing each elevator to requested floors and coordinating elevators in such a way to minimize

waiting times for people moving between floors. It also needs to handle possible faults and failures in the system, such as doors not opening/closing, packets being lost on the LAN etc.

Utilities.java – This is a class with all the variables and

Utiltest.java - This is a JUnit test class that tests the Utilities class.

AllTest.java -This is a JUnit test class that tests all three test files: FloorTest, SchedulerTest and ElevatorTest; concurrently.

input.txt - This is a text file containing dummy test cases of the elevator going from one floor to another; this input is used in order to test the efficiency of the elevator.

HandleUserInput.java - This is the class that takes the input text file and uses the user input(the elevator requests) to run the elevator using the state machine via the scheduler.

-SETUP INSTRUCTIONS

1. Unzip project folder and import into IntelliJ

2. Run Host.java, ElevatorConfig.java, and FloorControl.java. The files should only be run in this specified order and no other order in order for the system to run efficiently.

In case of port error "java.net.BindException: Address already in use: JVM_Bind":

In administrator mode command prompt: type in 'netstat -aon | findstr :<port number>' (In order to get the pid of the specific task) and 'taskkill/pid <pid> /f' will kill that task in the case that it was already running.

This will clear any ports already in use; port numbers 8008, 3137, 3237, 3337,3437 and 6520 must not be in use before running the main program.

Project Iteration 4 - Measuring Timing of Multiple Real Elevators with Error Detection and Correction

- PURPOSE

The purpose of this project is to design and implement an elevator control system and simulator using three different subsystems: Elevator, Floor and Scheduler. Each subsystem is coded as a separate 32Win process/ program that can be opened as a project in IntelliJ, and all three subsystems communicate with each user using DatagramSocket objects. This project is an insight into real-time communication between the three different subsystems in order to make the elevator work, using a scheduler. The Floor subsystem reads the input file which contains the information about arrival of passengers and all button presses and lamps and sends it across to the Scheduler. The scheduler will then communicate that the elevator has acknowledged the request. The elevator is responsible for making calls to the Scheduler while it is stationary to check for any pending requests. In case of pending requests, the elevator subsystem handles the request and then sends and acknowledgement that the request has been received to the Scheduler to pass back to the Floor subsystem. The program terminates when the Floor subsystem has handled and completed all events in the input file and received an acknowledgement for the last event as well.

In this iteration, we accounted for more elevators being able to run through the simulation and introduced a state machine that is responsible for the different states the elevator goes into throughout its runtime. Additionally, we added code for detecting and handling faults. For example we added timing events so that if the timer goes off before an elevator reaches a floor, then our system should assume a fault (either, the elevator is stuck between floors, or the arrival sensor at a floor has failed).

-BREAKDOWN OF RESPONSIBILITIES

The folder 'Diagrams' containing all required UML diagrams, timing diagram, sequence diagrams and state diagrams, as well as JUnit test files: ElevatorTest.java, FloorTest.java, SchedulerTest.java - Jennifer, Nkechi

Elevator.java, ElevatorButton.java, ElevatorStatus.java, Floor.java, FloorControl.java - Oluwatobi, Ryan

ElevatorStateMachine.java, DoorOpenState.java, DoorCloseState.java, MovingState.java, IdleState.java - Nkechi, Aditi

ElevatorState, ElevatorEvent.java - Nkechi

ElevatorConfig.java, Host.java, Scheduler.java - Oluwatobi, Ryan, Aditi

AllTest.java - Oluwatobi

Readme.txt - Aditi

-DESCRIPTION OF FILES WITH FILE NAMES

DoorOpenState.java - This class is the state of the elevator when the doors of the elevator are open, and is one of the states of the state machine.

DoorCloseState.java - This class is the state of the elevator when the doors of the elevator are close, and is one of the states of the state machine.

MovingState.java - This class is the state of the elevator when the elevator is in motion, and is one of the states of the state machine.

IdleState.java - This class is the state of the elevator when the elevator is not in use, and is one of the states of the state machine.

Elevator.java - This thread is responsible for taking requests from the scheduler, while stationary. It then handles those requests, and then passes and acknowledgement to the Scheduler in order to be passed back to the Floor subsystem.

ElevatorButton.java - This class contains the buttons(either, down or hold) which indicates to the other subsystems which direction the elevator will go in.

ElevatorState -This is an interface that connects all the different states of the state machine by a method that handles all the different events responsible for the elevator to go into those respective states.

ElevatorConfig.java - This is the main thread/class responsible for controlling the working of the elevator, by responding to the requests sent by the floor subsystem via the Scheduler. It uses DatagramPackets in order to communicate with the other two subsystems, and handles the requests sent by the user in order for the elevator to reach the destination floor selected by the user. It also initializes the Floor subsystem(client), Elevator subsystem(client) and the Scheduler(server) threads and then starts them.

ElevatorStatus.java - This thread is responsible for indicating whether the elevator is in use or empty.

ElevatorEvent.java - This is a class that enumerates events that can occur in the elevator system state machine. It includes events such as going up, going down, opening door, closing door, and stopping.

ElevatorStateMachine.java - This is a class that represents the implementation of the state machine, which contains all of the different states that the elevator goes into and transitions through. It contains information about the current state of the elevator, as well as a method to set the new state of the elevator.

ElevatorTest.java - This is a JUnit test class that tests all the methods written in the files from the Elevator Subsystem, in order to see whether the subsystem is able to function efficiently.

Floor.java - This thread is responsible for the Floor Subsystem containing the floor number, number of elevators, lamp status to indicate whether the elevator is going up or down and if the floor is an upper or lower floor.

FloorControl.java -This is the main class responsible for the control of the elevator with respect to requests sent via the scheduler, and it also processes the elevator's departure messages. It ensures that the floor subsystem waits after sending a service request to the Scheduler, in order to receive an acknowledgement.

FloorTest.java - This is a jUnit test class that tests all the methods of the Floor Subsystem.

Host.java - This class acts as the server and contains information about the ports that are not meant to be used/ which must be cleared before running the main project. It also contains the DatagramSocket information necessary for communication between the 3 different subsystems, and helps to control movements of the elevator via the scheduler by actively taking part in the request handling process.

Scheduler.java - This is the thread responsible for controlling the scheduler subsystem and the main point of contact between the Floor Subsystem and the Elevator Subsystem.

It is used for submitting new requests to the Scheduler from the Floor Subsystem, and then communicating the same to the elevator in order for the elevator to then take the necessary steps needed to reach the destination floor of the user's request. The scheduler is responsible for accepting input from all of the sensors, and sending indications (to devices such as lamps) and commands (to devices such as the motor and door). It is responsible for routing each elevator to requested floors and coordinating elevators in such a way to minimize waiting times for people moving between floors. It also needs to handle possible faults and failures in the system, such as doors not opening/closing, packets being lost on the LAN etc.

SchedulerTest.java - This is a JUnit test class that tests the methods in the Scheduler.java file.

AllTest.java -This is a JUnit test class that tests all three test files: FloorTest, SchedulerTest and ElevatorTest; concurrently.

input.txt - This is a text file containing dummy test cases of the elevator going from one floor to another; this input is used in order to test the efficiency of the elevator.

HandleUserInput.java - This is the class that takes the input text file and uses the user input(the elevator requests) to run the elevator using the state machine via the scheduler.

-SETUP INSTRUCTIONS

1. Unzip project folder and import into IntelliJ

2. Run Host.java, ElevatorConfig.java, and FloorControl.java. The files should only be run in this specified order and no other order in order for the system to run efficiently.

In case of port error "java.net.BindException: Address already in use: JVM_Bind":

In administrator mode command prompt: type in 'netstat -aon | findstr :<port number>' (In order to get the pid of the specific task) and 'taskkill/pid <pid> /f' will kill that task in the case that it was already running.

This will clear any ports already in use; port numbers 8008, 3137, 3237, 3337,3437 and 6520 must not be in use before running the main program.

Project Iteration 3 - Measuring Timing of Multiple Real Elevators

- PURPOSE

The purpose of this project is to design and implement an elevator control system and simulator using three different subsystems: Elevator, Floor and Scheduler. Each subsystem is coded as a separate 32Win process/ program that can be opened as a project in IntelliJ, and all three subsystems communicate with each user using DatagramSocket objects. This project is an insight into real-time communication between the three different subsystems in order to make the elevator work, using a scheduler. The Floor subsystem reads the input file which contains the information about arrival of passengers and all button presses and lamps, and sends it across to the Scheduler. The scheduler will then communicate that the elevator has acknowledged the request. The elevator is responsible for making calls to the Scheduler while it is stationary to check for any pending requests. In case of pending requests, the elevator subsystem them handles the reqiest and then sends and acknowledgement that the request has been received to the Scheduler to pass back to the Floor subsystem. The program terminates when the Floor subsystem has handled and completed all events in the input file and received an acknowledgement for the last event as well.

In this iteration, we accounted for more elevators being able to run through the simulation and introduced a state machine that is responsible for the different states the elevator goes into throughout its runtime.

-BREAKDOWN OF RESPONSIBILITIES

The folder 'Diagrams' containing all required UML diagrams, sequence diagrams and state diagrams, as well as JUnit test files: ElevatorTest.java, FloorTest.java, SchedulerTest.java - Jennifer, Nkechi

Elevator.java, ElevatorButton.java, ElevatorStatus.java, Floor.java, FloorControl.java - Oluwatobi, Ryan

ElevatorStateMachine.java, DoorOpenState.java, DoorCloseState.java, MovingState.java, IdleState.java - Nkechi, Aditi

ElevatorState, ElevatorEvent.java - Nkechi

ElevatorConfig.java, Host.java, Scheduler.java - Oluwatobi, Ryan, Aditi

AllTest.java - Oluwatobi

Readme.txt - Aditi

-DESCRIPTION OF FILES WITH FILE NAMES

DoorOpenState.java - This class is the state of the elevator when the doors of the elevator are open and is one of the states of the state machine.

DoorCloseState.java - This class is the state of the elevator when the doors of the elevator are close and is one of the states of the state machine.

MovingState.java - This class is the state of the elevator when the elevator is in motion and is one of the states of the state machine.

IdleState.java - This class is the state of the elevator when the elevator is not in use and is one of the states of the state machine.

Elevator.java - This thread is responsible for taking requests from the scheduler, while stationary. It then handles those requests, and then passes and acknowledgement to the Scheduler in order to be passed back to the Floor subsystem.

ElevatorButton.java - This class contains the buttons(either, down or hold) which indicates to the other subsystems which direction the elevator will go in.

ElevatorState -This is an interface that connects all the different states of the state machine by a method that handles all the different events responsible for the elevator to go into those respective states.

ElevatorConfig.java -This is the main thread/class responsible for controlling the working of the elevator, by responding to the requests sent by the floor subsystem via the Scheduler. It uses DatagramPackets in order to communicate with the other two subsystems, and handles the requests sent by the user in order for the elevator to reach the destination floor selected by the user. It also initializes the Floor subsystem(client), Elevator subsystem(client) and the Scheduler(server) threads and then starts them.

ElevatorStatus.java - This thread is responsible for indicating whether the elevator is in use or empty.

ElevatorEvent.java - This is a class that enumerates events that can occur in the elevator system state machine. It includes events such as going up, going down, opening door, closing door, and stopping.

ElevatorStateMachine.java - This is a class that represents the implementation of the state machine, which contains all of the different states that the elevator goes into and transitions through. It contains information about the current state of the elevator, as well as a method to set the new state of the elevator.

ElevatorTest.java -This is a JUnit test class that tests all the methods written in the files from the Elevator Subsystem, in order to see whether the subsystem is able to function efficiently.

Floor.java - This thread is responsible for the Floor Subsystem containing the floor number, number of elevators, lamp status to indicate whether the elevator is going up or down and if the floor is an upper or lower floor.

FloorControl.java -This is the main class responsible for the control of the elevator with respect to requests sent via the scheduler, and it also processes the elevator's departure messages. It ensures that the floor subsystem waits after sending a service request to the Scheduler, in order to receive an acknowledgement.

FloorTest.java - This is a JUnit test class that tests all the methods of the Floor Subsystem.

Host.java - This class acts as the server and contains information about the ports that are not meant to be used/ which must be cleared before running the main project. It also contains the DatagramSocket information necessary for communication between the 3 different subsystems and helps to control movements of the elevator via the scheduler by actively taking part in the request handling process.

Scheduler.java - This is the thread responsible for controlling the scheduler subsystem and the main point of contact between the Floor Subsystem and the Elevator Subsystem.

It is used for submitting new requests to the Scheduler from the Floor Subsystem, and then communicating the same to the elevator in order for the elevator to then take the necessary steps needed to reach the destination floor of the user's request. The scheduler is responsible for accepting input from all of the sensors and sending indications (to devices such as lamps) and commands (to devices such as the motor and door). It is responsible for routing each elevator to requested floors and coordinating elevators in such a way to minimize waiting times for people moving between floors. It also needs to handle possible faults and failures in the system, such as doors not opening/closing, packets being lost on the LAN etc.

SchedulerTest.java -This is a JUnit test class that tests the methods in the Scheduler.java file.

AllTest.java - This is a JUnit test class that tests all three test files: FloorTest, SchedulerTest and ElevatorTest; concurrently.

input.txt -This is a text file containing dummy test cases of the elevator going from one floor to another; this input is used, in order to test the efficiency of the elevator.

HandleUserInput.java - This is the class that takes the input text file and uses the user input(the elevator requests) to run the elevator using the state machine via the scheduler.

-SETUP INSTRUCTIONS

1. Unzip project folder and import into IntelliJ

2. Run Host.java, ElevatorConfig.java, and FloorControl.java. The files should only be run in this specified order and no other order in order for the system to run efficiently.

In case of port error "java.net.BindException: Address already in use: JVM_Bind":

In administrator mode command prompt: type in 'netstat -aon | findstr :<port number>' (In order to get the pid of the specific task) and 'taskkill/pid <pid> /f' will kill that task in the case that it was already running.

This will clear any ports already in use; port numbers 8008, 3137, 3237, 3337,3437 and 6520 must not be in use before running the main program.

Project Iteration 2 - Measuring Timing of a Real Elevator

- PURPOSE

The purpose of this project is to design and implement an elevator control system and simulator using three different subsystems: Elevator, Floor and Scheduler. Each subsystem is coded as a separate 32Win process/ program that can be opened as a project in IntelliJ, and all three subsystems communicate with each user using DatagramSocket objects. This project is an insight into real-time communication between the three different subsystems in order to make the elevator work, using a scheduler. The Floor subsystem reads the input file which contains the information about arrival of passengers and all button presses and lamps and sends it across to the Scheduler. The scheduler will then communicate that the elevator has acknowledged the request. The elevator is responsible for making calls to the Scheduler while it is stationary to check for any pending requests. In case of pending requests, the elevator subsystem them handles the request and then sends and acknowledgement that the request has been received to the Scheduler to pass back to the Floor subsystem. The program terminates when the Floor subsystem has handled and completed all events in the input file and received an acknowledgement for the last event as well.

-BREAKDOWN OF RESPONSIBILITIES

The folder 'Diagrams' containing all required UML diagrams, sequence diagrams and state diagrams, as well as JUnit test files: ElevatorTest.java, FloorTest.java, SchedulerTest.java - Jennifer, Nkechi

Elevator.java, ElevatorButton.java, ElevatorStatus.java, Floor.java, FloorControl.java - Oluwatobi, Ryan

ElevatorConfig.java, Host.java, Scheduler.java - Oluwatobi, Ryan, Aditi

Readme.txt - Aditi

-DESCRIPTION OF FILES WITH FILE NAMES

Elevator.java - This thread is responsible for taking requests from the scheduler, while stationary. It then handles those requests, and then passes and acknowledgement to the Scheduler in order to be passed back to the Floor subsystem.

ElevatorButton.java -This class contains the buttons(either, down or hold) which indicates to the other subsystems which direction the elevator will go in.

ElevatorConfig.java -This is the main thread/class responsible for controlling the working of the elevator, by responding to the requests sent by the floor subsystem via the Scheduler. It uses DatagramPackets in order to communicate with the other two subsystems, and handles the requests sent by the user in order for the

elevator to reach the destination floor selected by the user. It also initializes the Floor subsystem(client), Elevator subsystem(client) and the Scheduler(server) threads and then starts them.

ElevatorStatus.java - This thread is responsible for indicating whether the elevator is in use or empty.

ElevatorTest.java - This is a JUnit test class that tests all the methods written in the files from the Elevator Subsystem, in order to see whether the subsystem is able to function efficiently.

Floor.java - This thread is responsible for the Floor Subsystem containing the floor number, number of elevators, lamp status to indicate whether the elevator is going up or down and if the floor is an upper or lower floor.

FloorControl.java - This is the main class responsible for the control of the elevator with respect to requests sent via the scheduler, and it also processes the elevator's departure messages. It ensures that the floor subsystem waits after sending a service request to the Scheduler, in order to receive an acknowledgement.

FloorTest.java - This is a jUnit test class that tests all the methods of the Floor Subsystem.

Host.java - This class acts as the server and contains information about the ports that are not meant to be used/ which must be cleared before running the main project. It also contains the DatagramSocket information necessary for communication between the 3 different subsystems and helps to control movements of the elevator via the scheduler by actively taking part in the request handling process.

Scheduler.java - This is the thread responsible for controlling the scheduler subsystem and the main point of contact between the Floor Subsystem and the Elevator Subsystem.

It is used for submitting new requests to the Scheduler from the Floor Subsystem, and then communicating the same to the elevator in order for the elevator to then take the necessary steps needed to reach the destination floor of the user's request. The scheduler is responsible for accepting input from all of the sensors, and sending indications (to devices such as lamps) and commands (to devices such as the motor and door). It is responsible for routing each elevator to requested floors and coordinating elevators in such a way to minimize waiting times for people moving between floors. It also needs to handle possible faults and failures in the system, such as doors not opening/closing, packets being lost on the LAN etc.

SchedulerTest.java - This is a jUnit test class that tests the methods in the Scheduler.java file.

-SETUP INSTRUCTIONS

1. Unzip project folder and import into IntelliJ

2. Run Host.java, ElevatorConfig.java, and FloorControl.java. The files should only be run in this specified order and no other order in order for the system to run efficiently.

In case of port error "java.net.BindException: Address already in use: JVM_Bind":

In administrator mode command prompt: type in 'netstat -aon | findstr :<port number>' (In order to get the pid of the specific task) and 'taskkill/pid <pid> /f' will kill that task in the case that it was already running.

This will clear any ports already in use; port numbers 8008, 3137 and 6520 must not be in use before running the main program.

Project Iteration 1 - Measuring Timing of a Real Elevator

- PURPOSE

The purpose of this project is to design and implement an elevator control system and simulator using three different subsystems: Elevator, Floor and Scheduler. Each subsystem is coded as a separate 32Win process/ program that can be opened as a project in IntelliJ, and all three subsystems communicate with each user using DatagramSocket objects. This project is an insight into real-time communication between the three different subsystems in order to make the elevator work, using a scheduler. The Floor subsystem reads the input file which contains the information about arrival of passengers and all button presses and lamps and sends it across to the Scheduler. The scheduler will then communicate that the elevator has acknowledged the request. The elevator is responsible for making calls to the Scheduler while it is stationary to check for any pending requests. In case of pending requests, the elevator subsystem handles the request and then sends and acknowledgement that the request has been received to the Scheduler to pass back to the Floor subsystem. The program terminates when the Floor subsystem has handled and completed all events in the input file and received an acknowledgement for the last event as well.

-BREAKDOWN OF RESPONSIBILITIES

The folder 'UML diagrams' containing all required UML diagrams and sequence diagrams, as well as JUnit test files: ElevatorTest.java, FloorTest.java, SchedulerTest.java - Jennifer, Nkechi

Elevator.java, ElevatorButton.java, ElevatorStatus.java, Floor.java, FloorControl.java - Oluwatobi, Ryan

ElevatorConfig.java, Host.java, Scheduler.java - Oluwatobi, Ryan, Aditi

Readme.txt - Aditi

-DESCRIPTION OF FILES WITH FILE NAMES

Elevator.java - This thread is responsible for taking requests from the scheduler, while stationary. It then handles those requests, and then passes and acknowledgement to the Scheduler in order to be passed back to the Floor subsystem.

ElevatorButton.java - This class contains the buttons(either, down or hold) which indicates to the other subsystems which direction the elevator will go in.

ElevatorConfig.java -This is the main thread/class responsible for controlling the working of the elevator, by responding to the requests sent by the floor subsystem via the Scheduler. It uses DatagramPackets in order to communicate with the other two subsystems, and handles the requests sent by the user in order for the elevator to reach the destination floor selected by the user. It also initializes the Floor subsystem(client), Elevator subsystem(client) and the Scheduler(server) threads and then starts them.

ElevatorStatus.java - This thread is responsible for indicating whether the elevator is in use or empty.

ElevatorTest.java - This is a JUnit test class that tests all the methods written in the files from the Elevator Subsystem, in order to see whether the subsystem is able to function efficiently.

Floor.java - This thread is responsible for the Floor Subsystem containing the floor number, number of elevators, lamp status to indicate whether the elevator is going up or down and if the floor is an upper or lower floor.

FloorControl.java -This is the main class responsible for the control of the elevator with respect to requests sent via the scheduler, and it also processes the elevator's departure messages. It ensures that the floor subsystem waits after sending a service request to the Scheduler, in order to receive an acknowledgement.

FloorTest.java - This is a jUnit test class that tests all the methods of the Floor Subsystem.

Host.java - This class acts as the server and contains information about the ports that are not meant to be used/ which must be cleared before running the main project. It also contains the DatagramSocket information necessary for communication between the 3 different subsystems and helps to control movements of the elevator via the scheduler by actively taking part in the request handling process.

Scheduler.java - This is the thread responsible for controlling the scheduler subsystem and the main point of contact between the Floor Subsystem and the Elevator Subsystem.

It is used for submitting new requests to the Scheduler from the Floor Subsystem, and then communicating the same to the elevator in order for the elevator to then take the necessary steps needed to reach the destination floor of the user's request.The scheduler is responsible for accepting input from all of the sensors, and sending indications (to devices such as lamps) and commands (to devices such as the motor and door). It is responsible for routing each elevator to requested floors and coordinating elevators in such a way to minimize waiting times for people moving between floors. It also needs to handle possible faults and failures in the system, such as doors not opening/closing, packets being lost on the LAN etc.

SchedulerTest.java - This is a jUnit test class that tests the methods in the Scheduler.java file.

-SETUP INSTRUCTIONS

1. Unzip project folder and import into IntelliJ
2. Run Host.java, ElevatorConfig.java, and FloorControl.java

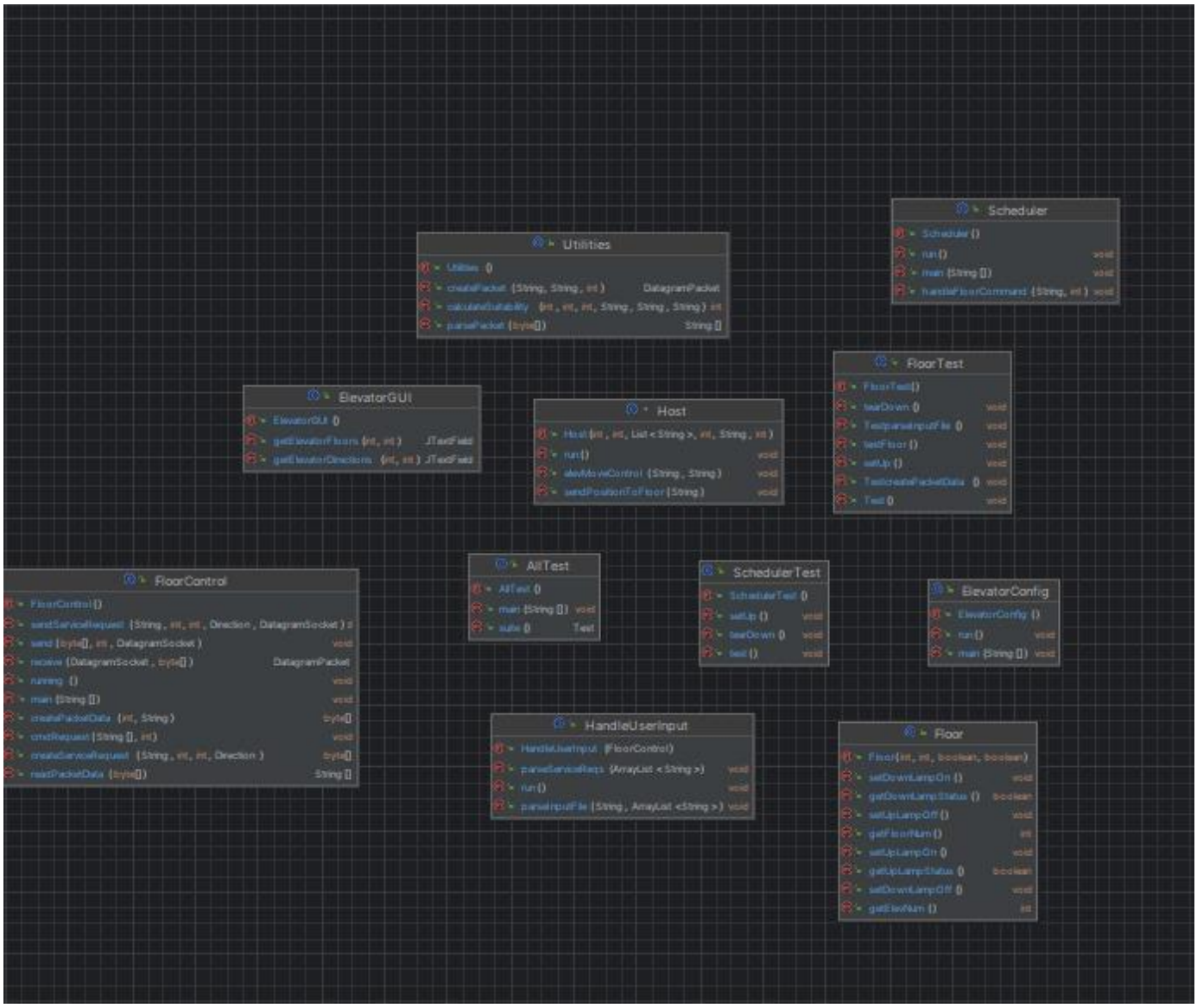
In case of port error "java.net.BindException: Address already in use: JVM_Bind":

In administrator mode command prompt: type in 'netstat -aon | findstr :<port number>' (In order to get the pid of the specific task) and 'taskkill/pid <pid> /f' will kill that task in the case that it was already running.

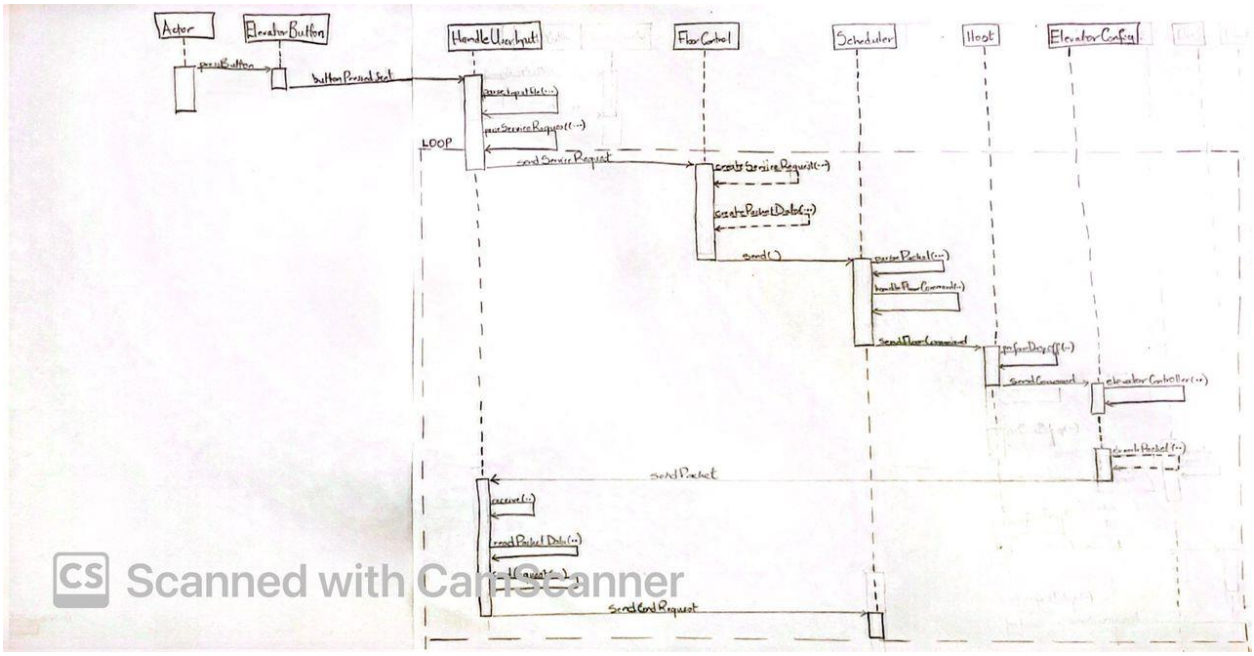
This will clear any ports already in use; port numbers 8008, 3137 and 6520 must not be in use before running the main program.

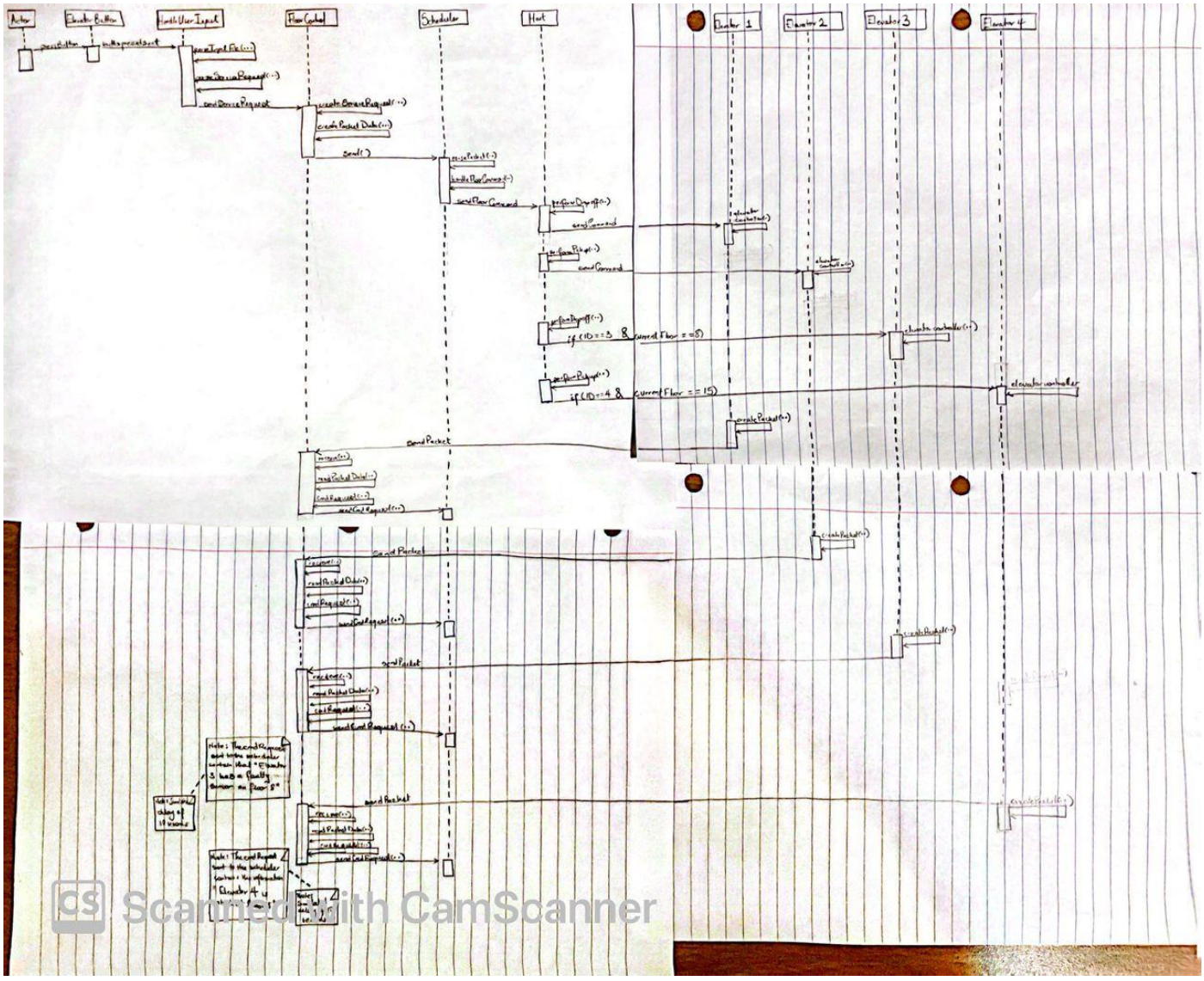
UML Diagrams:

UML diagram:

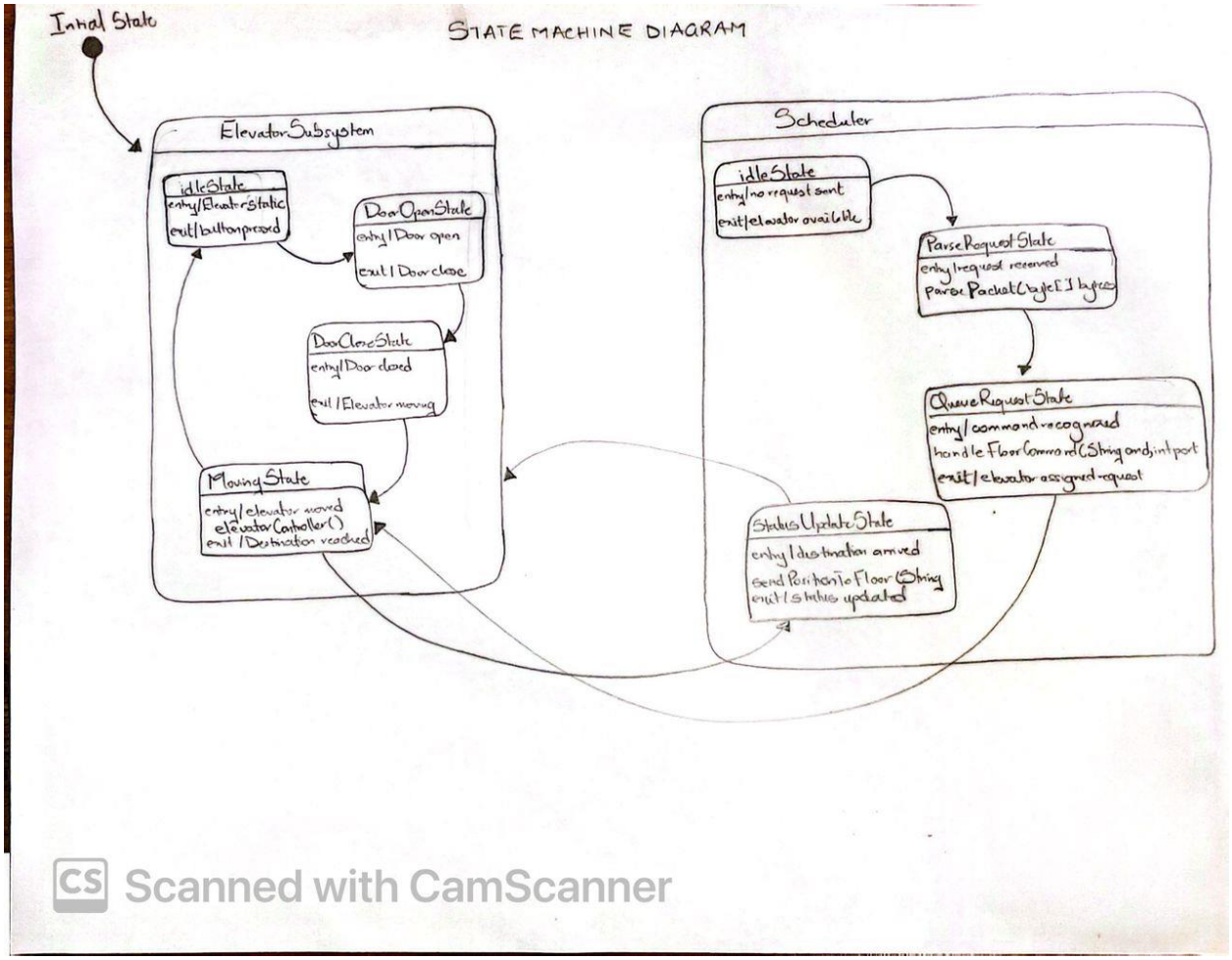


Sequence Diagrams:

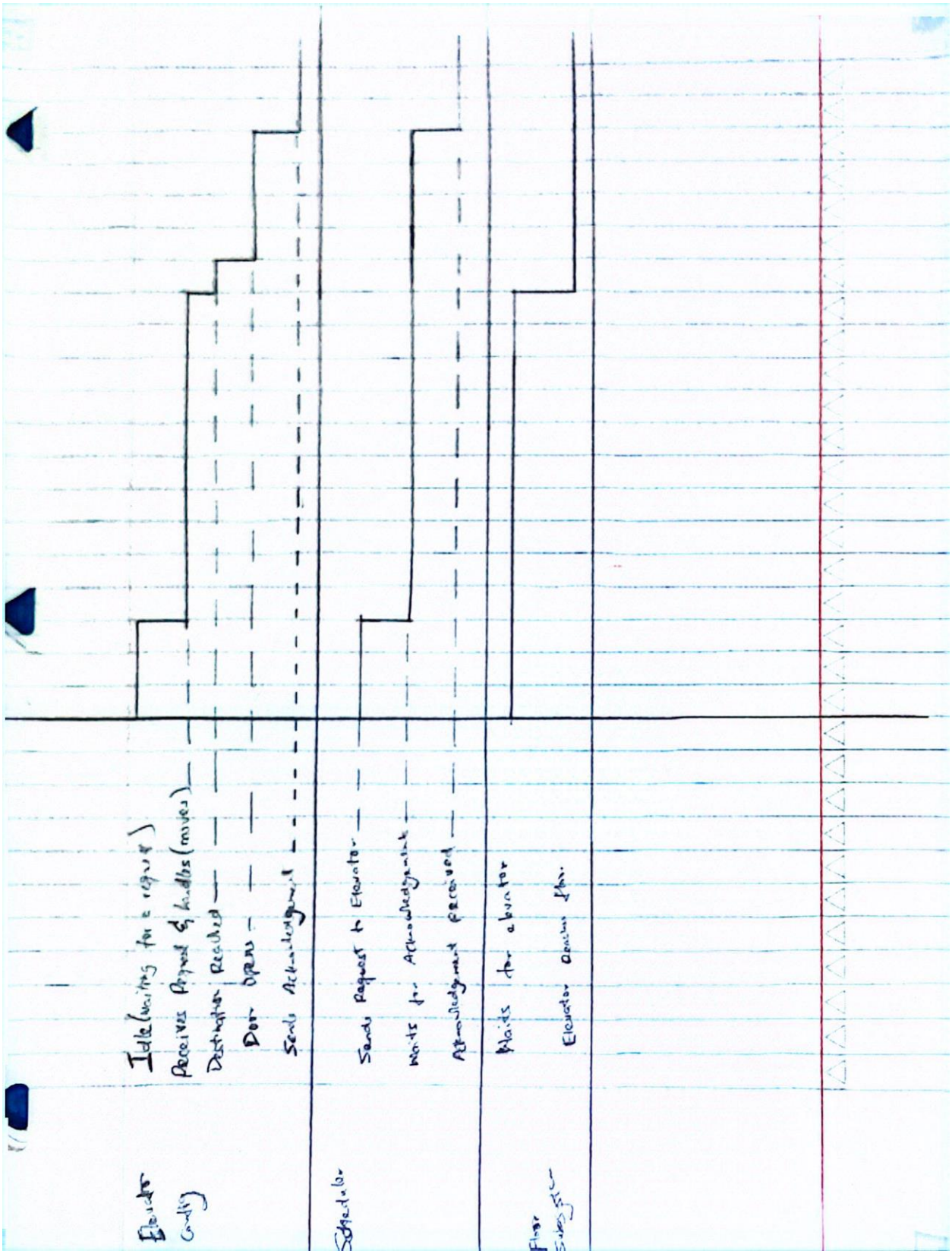




State Machine Diagram:



Timing Diagram:



Detailed instructions for set-up: running and testing.

-SETUP INSTRUCTIONS

The project folder contains code for three different subsystems: Elevator, Floor and Scheduler. For testing each subsystem, there is a jUnit test class as well as utiltest.java to test the Utilities class.

1. Unzip project folder and import into Intellij

2. Run Host.java, ElevatorConfig.java, and FloorControl.java. The files should only be run in this specified order and no other order in order for the system to run efficiently.

In case of port error "java.net.BindException: Address already in use: JVM_Bind":

In administrator mode command prompt: type in 'netstat -aon | findstr :<port number>' (In order to get the pid of the specific task) and 'taskkill/pid <pid> /f'will kill that task in the case that it was already running.

This will clear any ports already in use; port numbers 8008, 3137, 3237, 3337,3437 and 6520 must not be in use before running the main program.

Measurement results for determining door and floor movement times.

Collected Data

Team Member	Load Time	Move Time	Unload Time	Total Time
4th to 3rd Floor				
Ryan	13.06	6.55	10.81	30.43
Nkechi	13.59	7.55	10.4	31.54
Aditi	12.84	7.33	10.5	30.68
Jennifer	12.86	10.64	7.53	31.04
Tobi	12.95	6.98	10.7	30.65
Average	13.06	7.81	9.988	30.868
Variance	0.09535	2.64585	1.91407	0.18897

Team Member	Load Time	Move Time	Unload Time	Total Time
3rd to 4th Floor				
Ryan	20.7	9.15	8.64	38.5
Nkechi	19.94	11.54	7.61	39.11
Aditi	18.98	9.33	7.72	36.03
Jennifer	20.48	11.21	7.21	38.75
Tobi	21.63	9.93	7.3	38.26
Average	20.346	10.232	7.696	38.13
Variance	0.95568	1.18572	0.32303	1.47715

Team Member	Load Time	Move Time	Unload Time	Total Time
4th Floor to Tunnels				
Ryan	12.91	21.3	11.96	46.18
Nkechi	13.08	22.38	11.44	46.91
Aditi	12.78	22.52	10.93	46.23
Jennifer	12.85	21.82	11.93	46.6
Tobi	12.5	21.78	8.16	42.45
Average	12.824	21.96	10.884	45.674
Variance	0.04513	0.2444	2.49583	3.33613

Calculations

Distance between floors:
4 meters

Reflection on this project

We learnt through this project that the simulation of something as real and common in our day to day lives as elevators can be done through the help of threads/processes using code and and that everything around us is just an advanced version of what we learn/ learn to build on a small-scale. If we had had more time though, we would have added more states into the code alongside the GUI in order to make the simulation of the elevator system more accurate and as close to the measured values of the experimental data we calculated above.