Carleton University Department of Systems and Computer Engineering
SYSC 3303B2 Winter 2020
Winter Project
Team 4

Tamer Ibrahim, Jordan Peterkin, Benjamin Ransom, Mario Shebib
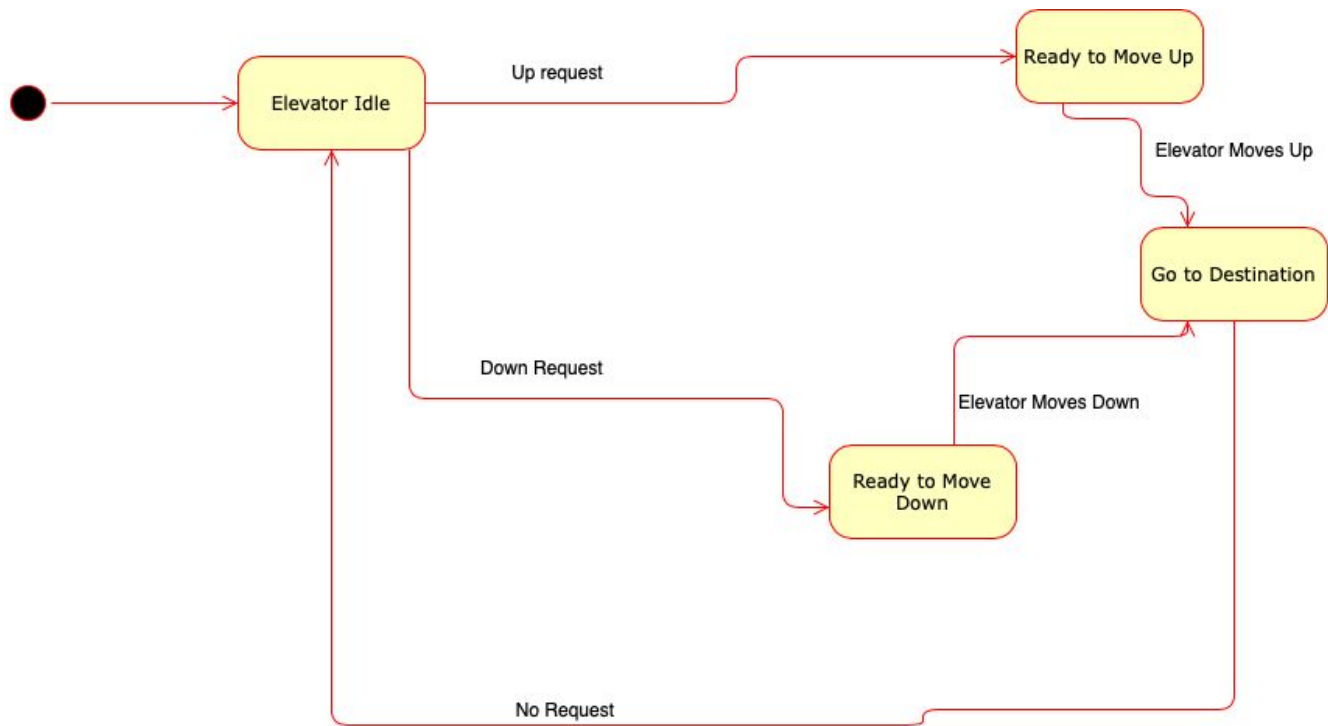
Table of Contents

Responsibilities:

Tamer: diagrams, bits of code on the floor class, as well as error handling.

Jordan: parts of the floor, scheduler and elevator classes, some of the README files, and the report.

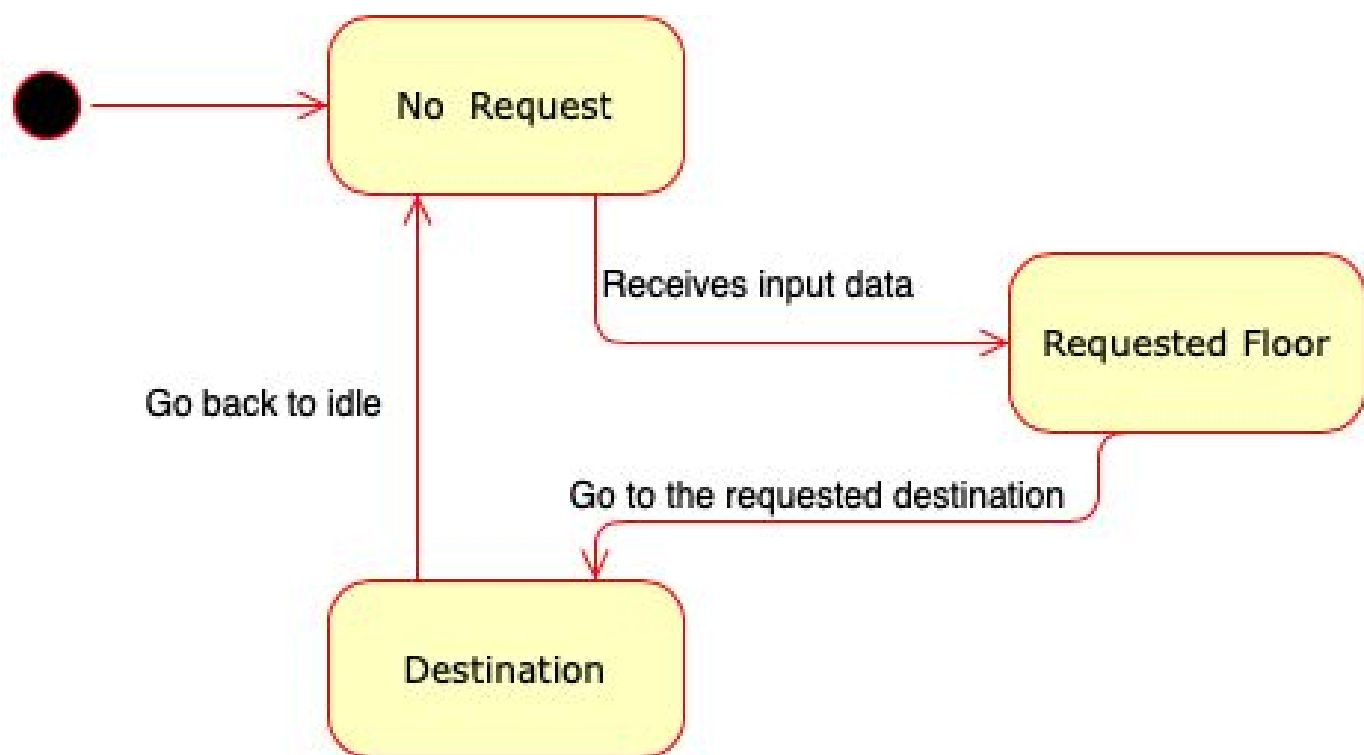Benjamin: the majority of the code in the scheduler and parts of the floor and elevator classes. Calculations and measurements.

Mario: the majority of the README files, the majority of the elevator class, as well as the JUnit tests.
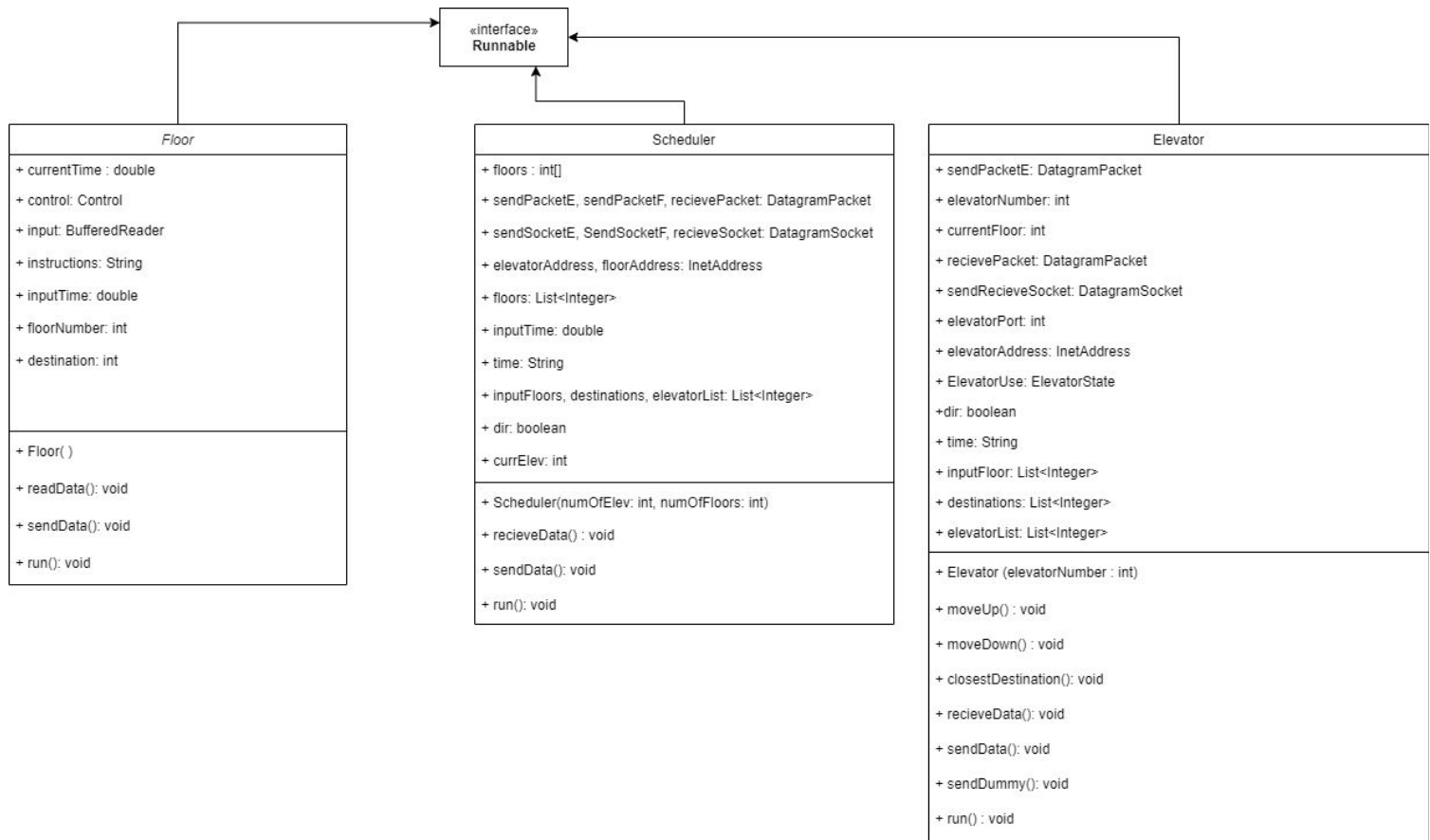
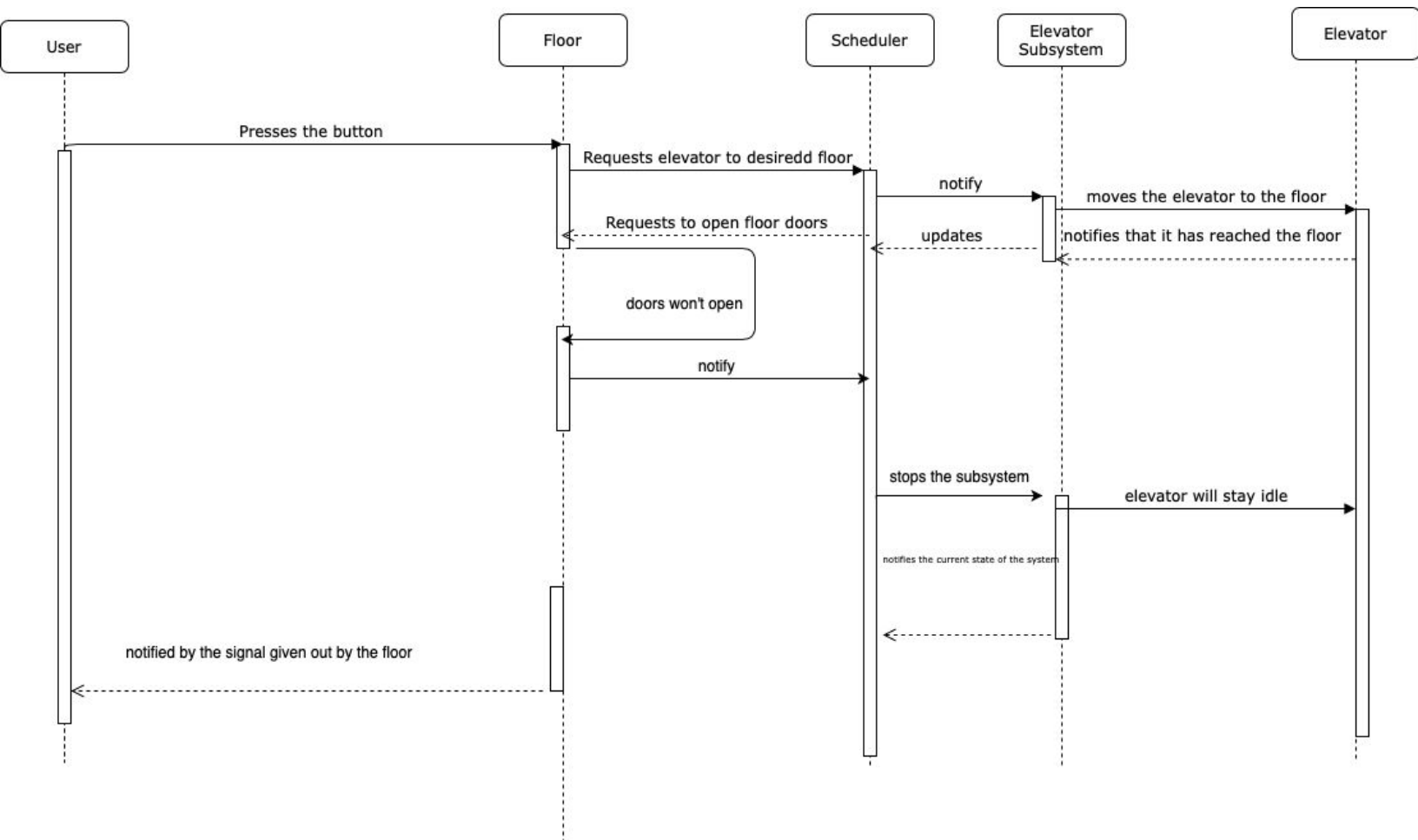Elevator State Machine Diagram:

Scheduler State Machine Diagram

No Request

Receives input data

Requested Floor

Go back to idle

Go to the requested destination

Destination

UML Diagram

```
                              «interface»
                              Runnable
```

Floor

+ currentTime : double

+ control: Control

+ input: BufferedReader

+ instructions: String

+ inputTime: double

+ floorNumber: int

+ destination: int

+ Floor( )

+ readData(): void

+ sendData(): void

+ run(): void

Scheduler

+ floors : int[]

+ sendPacketE, sendPacketF, recievePacket: DatagramPacket

+ sendSocketE, SendSocketF, recieveSocket: DatagramSocket

+ elevatorAddress, floorAddress: InetAddress

+ floors: List<Integer>

+ inputTime: double

+ time: String

+ inputFloors, destinations, elevatorList: List<Integer>

+ dir: boolean

+ currElev: int

+ Scheduler(numOfElev: int, numOfFloors: int)

+ recieveData() : void

+ sendData(): void

+ run(): void

Elevator

+ sendPacketE: DatagramPacket

+ elevatorNumber: int

+ currentFloor: int

+ recievePacket: DatagramPacket

+ sendRecieveSocket: DatagramSocket

+ elevatorPort: int

+ elevatorAddress: InetAddress

+ ElevatorUse: ElevatorState

+dir: boolean

+ time: String

+ inputFloor: List<Integer>

+ destinations: List<Integer>

+ elevatorList: List<Integer>

+ Elevator (elevatorNumber : int)

+ moveUp() : void

+ moveDown() : void

+ closestDestination(): void

+ recieveData(): void
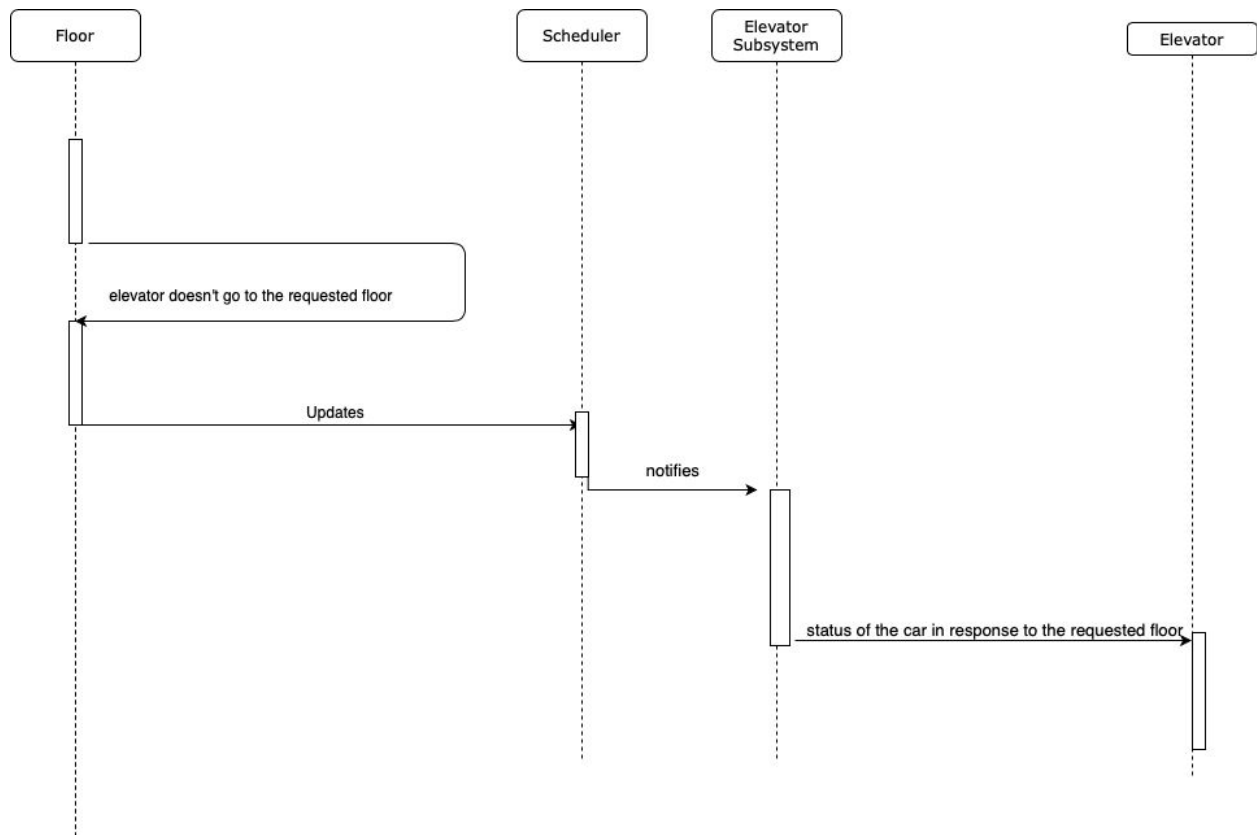
+ sendData(): void

+ sendDummy(): void

+ run() : void

Error Handling Sequence Diagram:

Doors Won't Open

Error Handling Sequence Diagram: Elevator will not go to the requested floor

Timing Diagram:

Elevator going towards requested floor		
Class name	Different methods	Timing Diagram
Elevator	Idle (waiting for request)	
	Recieves Request and moves	Error occurred
	Elevator arrived at distination	
	Door opens	
	Sends Acknowledge	Error occured
Scheduler	Sends Request to Elevation	
	Waits for Acknowledgment	
	Acknowledge Recieved	
Floor	Waits for Elevator	
	Elevator reaches floor	
		0 1 2 3 4 5 6 7 8 9 10
Time (Seconds)		
Error scenario. Elevator stuck	The error would be that the door gets stuck when it is at a door open state(	

Setup and Instructions:

Extract the folder to a single folder. Have all classes and the input file in one folder in the workspace. Right click each of the three classes and click "Run as…" , then "Java Application" for Scheduler.java, then run Elevator.java, then run Floor.java. Make sure all of these are run in separate console windows. The system will then run and the output will appear in each console window.

Calculations:

Loading/unloading time(s)		moving time (s)					
		1 floor	2 floors	3 floors	4 floors	5 floors	6 floors
7.71							
8.88		8.51	10.11	11.44	14.18	16.22	19.08
8.78		8.06	10.25	12.22	14.23	16.13	18.62
12.11		7.45	10.25	11.82	13.86	16.29	19.13
9.16		8.35	9.67	11.69	14.62	16.53	18.29
10.14		7.79	9.94	12.38	14.29	16.82	18.53
8.89		8.15	9.38	12.19	13.94	16.29	18.69
8.95		8.19	10.24	11.92	14.72	16.53	18.45
10.62		8.23	9.53	11.68	14.6	16.3	18.73
9.19		8.02	9.67	12.23	14.27	16.43	18.79
9.87		8.38	10.06	12.29	14.25	16.82	18.59
7.92		7.94	10.16	12.28	14.63	16.05	18.67
8.59		7.86	10.29	12.19	14.73	16.35	18.92
8.72		8.4	9.43	12	14.43	16.42	18.7
Average Loading/unloading time		8.21	9.38	11.86	14.29	16.9	18.96
9.252142857		8.15	10.12	11.69	14.75	16.52	19.13
		7.68	10.29	12.89	13.86	16.48	19.42
		7.86	9.86	11.69	14.26	16.2	18.25
		8.16	9.58	12.35	14.35	16.34	18.95
		7.96	10.35	12.24	14.86	16.43	18.68
		8.32	9.46	12.15	14.35	16.26	19.53
Distance between floors = 21 steps		7.69	10.2	12.03	14.03	16.34	19.32
1 step = 7 inches		8.26	9.86	12.09	14.08	16.26	18.95
21 steps = 12.25 feet		8.19	10.11	11.96	14.2	16.34	18.59
12.25feet =3.7338 m		8.36	9.59	12.43	14.62	16.44	18.68
		8.26	10.36	12.34	14.05	16.69	18.48
		7.89	10.24	12.96	14.14	16.1	19.26
		7.98	9.97	12.28	14.62	16.95	18.57
		8.65	9.68	11.86	14.29	16.76	19.68
		Average moving time(s)					
		8.105357143	9.929642857	12.1125	14.33928571	16.43535714	18.84428571
		Time between floors (s)					
(Difference between moving 1 floor and 2 floors, 2 floors and 3 floors, etc.		1.824285714	2.182857143	2.226785714	2.096071429	2.408928571	
		Average time between floors(s)					
		2.147785714					
		Standard deviation of moving times					
		0.268041603	0.324273718	0.338806993	0.275355985	0.238064849	0.358901385
		95%confidence interval					
	Average moving time +/-	0.099283999	0.120112666	0.125495866	0.101993284	0.088180454	0.132938933
		8.10535 ± 0.09928	9.92964±0.12011	12.1125±0.12550	14.33929±0.10199	16.43536±0.08818	18.84429±0.13294
		99.9%confidence interval					
	Average moving time +/-	0.166705939	0.201678971	0.210717804	0.17125505	0.148062181	0.223215321
		8.10535 ± 0.16671	9.92964±0.20167	12.1125±0.21072	14.33929±0.14126	16.43536±0.14806	18.84429±0.22322
		Maximum speed of the elevator = distance between floors/average time between floors						
		1.738441584					
		Assuming maximum speed is reached by the time the elevator gets halfway through all the floors, we use 3 floors as the distance for the acceleration equation, and that acceleration is constant						
		A=vf-vi/t					
		0.145421672	m/s^2				

What We Would Change:

We had to refactor our code several times because we did not manage it well. For example, after the first iteration, we had to go and redo the entire project. So to make the code more capable to handle UDP implementation and to implement each subsystem as a thread. The timestamps were difficult to implement because we did not keep track of when things were taking place from the beginning and needed to be implemented in the first iteration. Communication about what needs to be added and following through with what each member needs to get done and when it will get done could be improved drastically. The number of JUnit tests could be increased to account for more scenarios as well as testing all the methods in each class.

What We Did Well:

The team was able to come up with a system that was able to have three threads communicating with UDP one multiple computers (changed Inet address to localhost to have it run on one computer for the video). This is probably the only thing that is well designed and runs smoothly and well managed within this project. Outside of the core code, there are plenty of JUnit tests that work well even though they aren't able to cover all methods