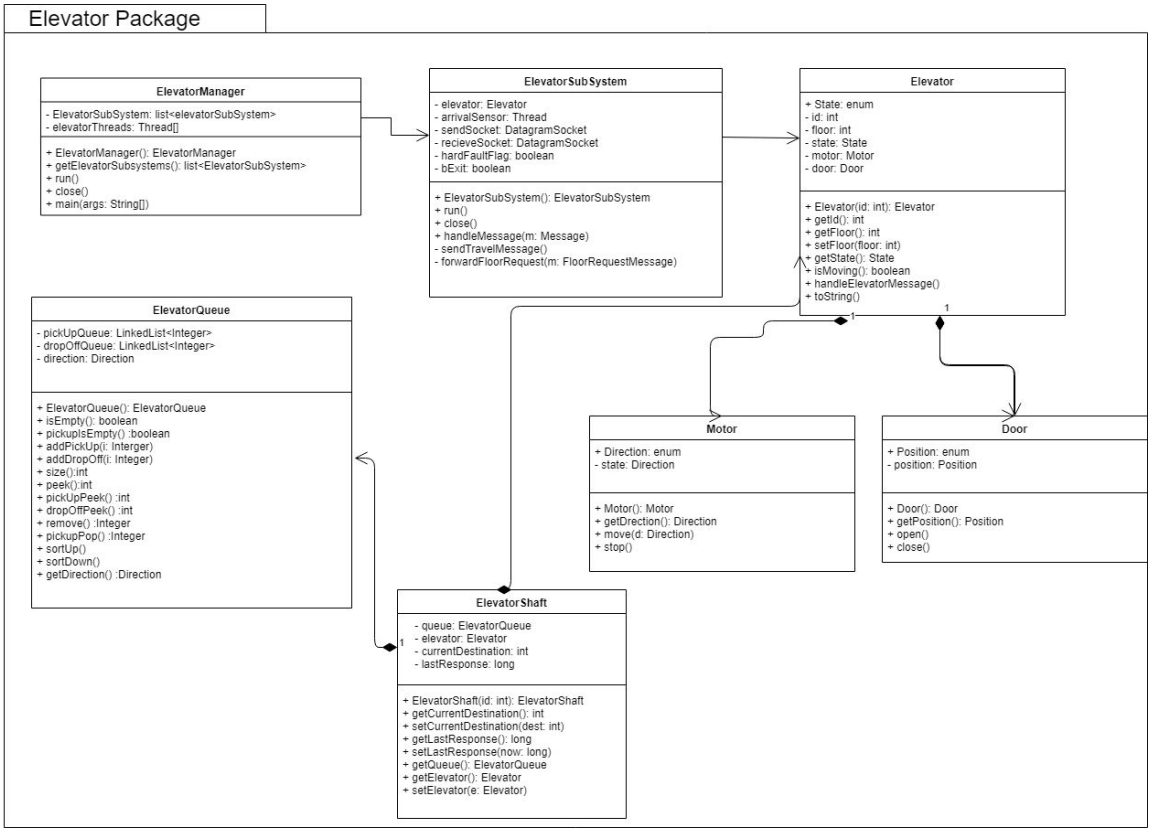
Project Report - Group 9

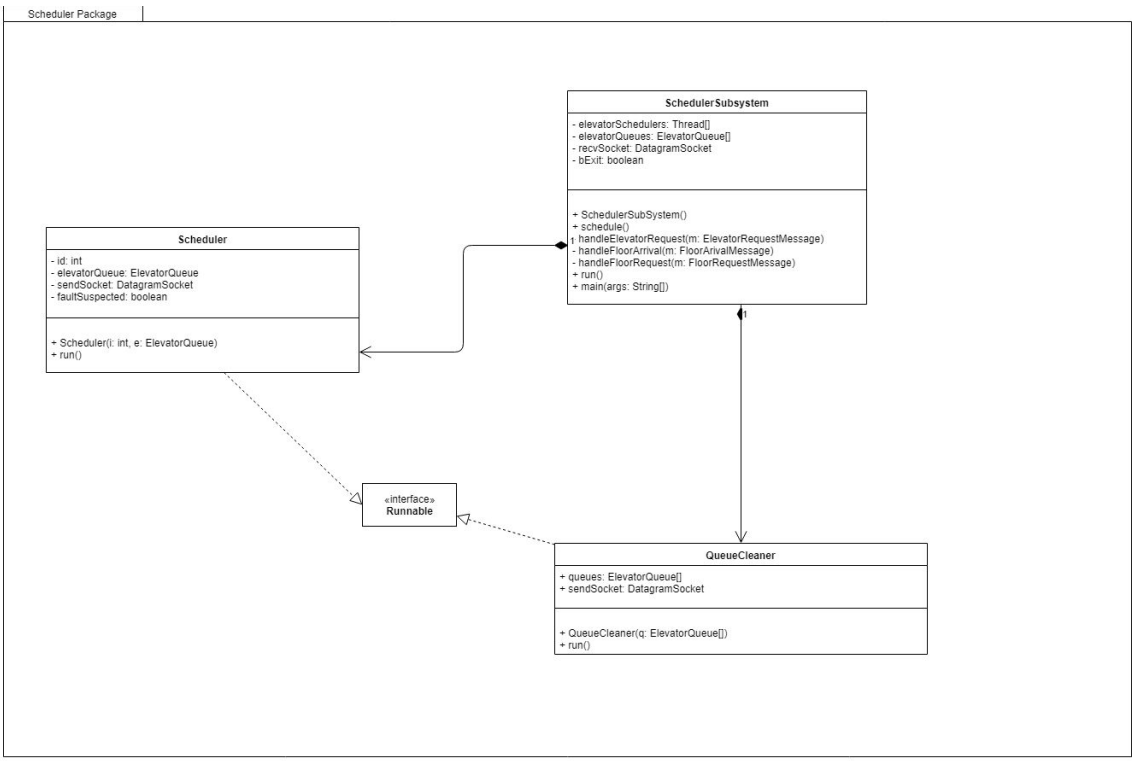
Vikram Bombhi, Jacky Chiu, Connor Poland, Kirin Rastogi

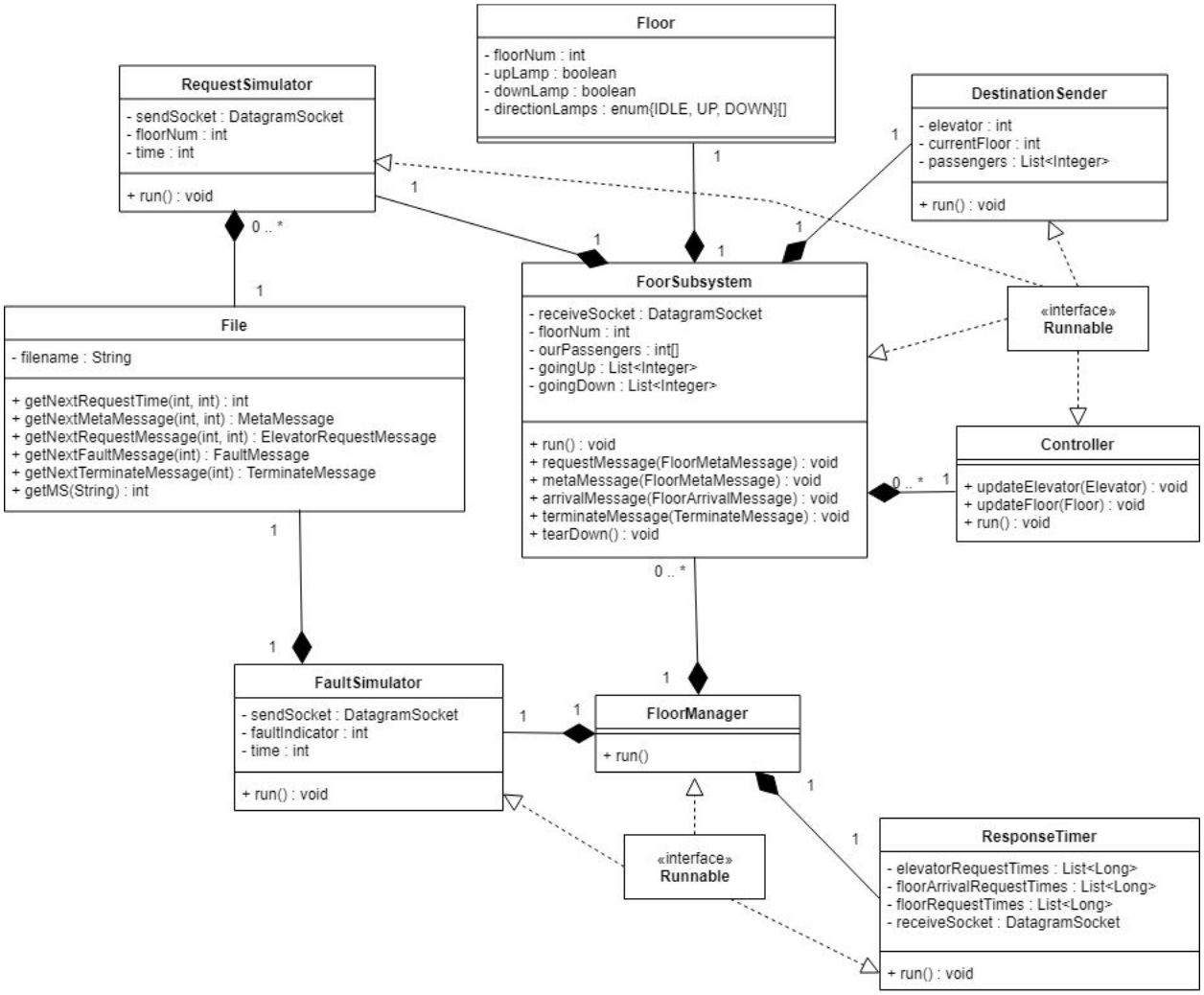
Responsibility Breakdown

	Vikram	Jacky	Connor	Kirin
Iteration 1	Basic structure and communication	Message (de)serialization and elevator subsystem	Floor subsystem	Built the scheduler algorithm
Iteration 2	Junit automation + testing, UML updates, scheduler + input file contribution	Elevator refactor, elevator junit, scheduler contributions	Implemented input file requests, timing diagram	Elevator refactor, scheduler contributions
Iteration 3	Elevator fault simulation and integration	Scheduler hard faults and tests	Refactor queuing system and simulating faults from the input file	Scheduler soft faults and tests
Iteration 4	Scheduler timing and statistics	Scheduler refactor	Diagrams and scaling up the amount of elevators	Scheduler timing and statistics
Iteration 5	UI development contributions	Controller and view for MVC	Scheduler refactor - model setup for MVC	Final diagram updates (UML, etc)

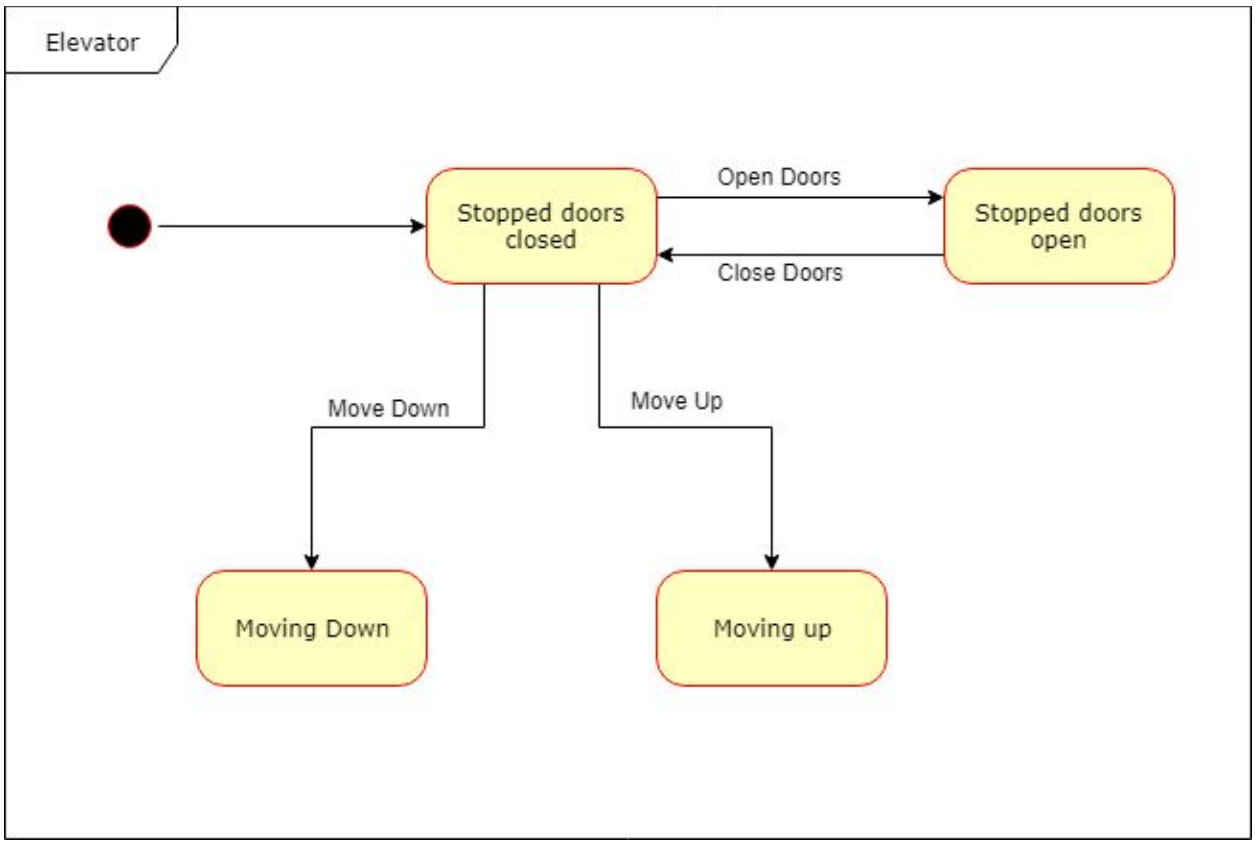
UML Class Diagrams


Elevator Package
ElevatorManager
- ElevatorSubSystem: list<elevatorSubSystem>
- elevatorThreads: Thread[]
+ ElevatorManager(): ElevatorManager
+ getElevatorSubsystems(): list<ElevatorSubSystem>
+ run()
+ close()
+ main(args: String[])
ElevatorSubSystem
- elevator: Elevator
- arrivalSensor: Thread
- sendSocket: DatagramSocket
- recieveSocket: DatagramSocket
- hardFaultFlag: boolean
- bExit: boolean
+ ElevatorSubSystem(): ElevatorSubSystem
+ run()
+ close()
+ handleMessage(m: Message)
- sendTravelMessage()
- forwardFloorRequest(m: FloorRequestMessage)
Elevator
+ State: enum
- id: int
- floor: int
- state: State
- motor: Motor
- door: Door
+ Elevator(id: int): Elevator
+ getId(): int
+ getFloor(): int
+ setFloor(floor: int)
+ getState(): State
+ isMoving(): boolean
+ handleElevatorMessage()
+ toString()
ElevatorQueue
- pickUpQueue: LinkedList<Integer>
- dropOffQueue: LinkedList<Integer>
- direction: Direction
+ ElevatorQueue(): ElevatorQueue
+ isEmpty(): boolean
+ pickupIsEmpty() :boolean
+ addPickUp(i: Interger)
+ addDropOff(i: Integer)
+ size():int
+ peek():int
+ pickUpPeek() :int
+ dropOffPeek() :int
+ remove() :Integer
+ pickupPop() :Integer
+ sortUp()
+ sortDown()
+ getDirection() :Direction
Motor
+ Direction: enum
- state: Direction
+ Motor(): Motor
+ getDirection(): Direction
+ move(d: Direction)
+ stop()
Door
+ Position: enum
- position: Position
+ Door(): Door
+ getPosition(): Position
+ open()
+ close()
ElevatorShaft
- queue: ElevatorQueue
- elevator: Elevator
- currentDestination: int
- lastResponse: long
+ ElevatorShaft(id: int): ElevatorShaft
+ getCurrentDestination(): int
+ setCurrentDestination(dest: int)
+ getLastResponse(): long
+ setLastResponse(now: long)
+ getQueue(): ElevatorQueue
+ getElevator(): Elevator
+ setElevator(e: Elevator)
1
1
1
1

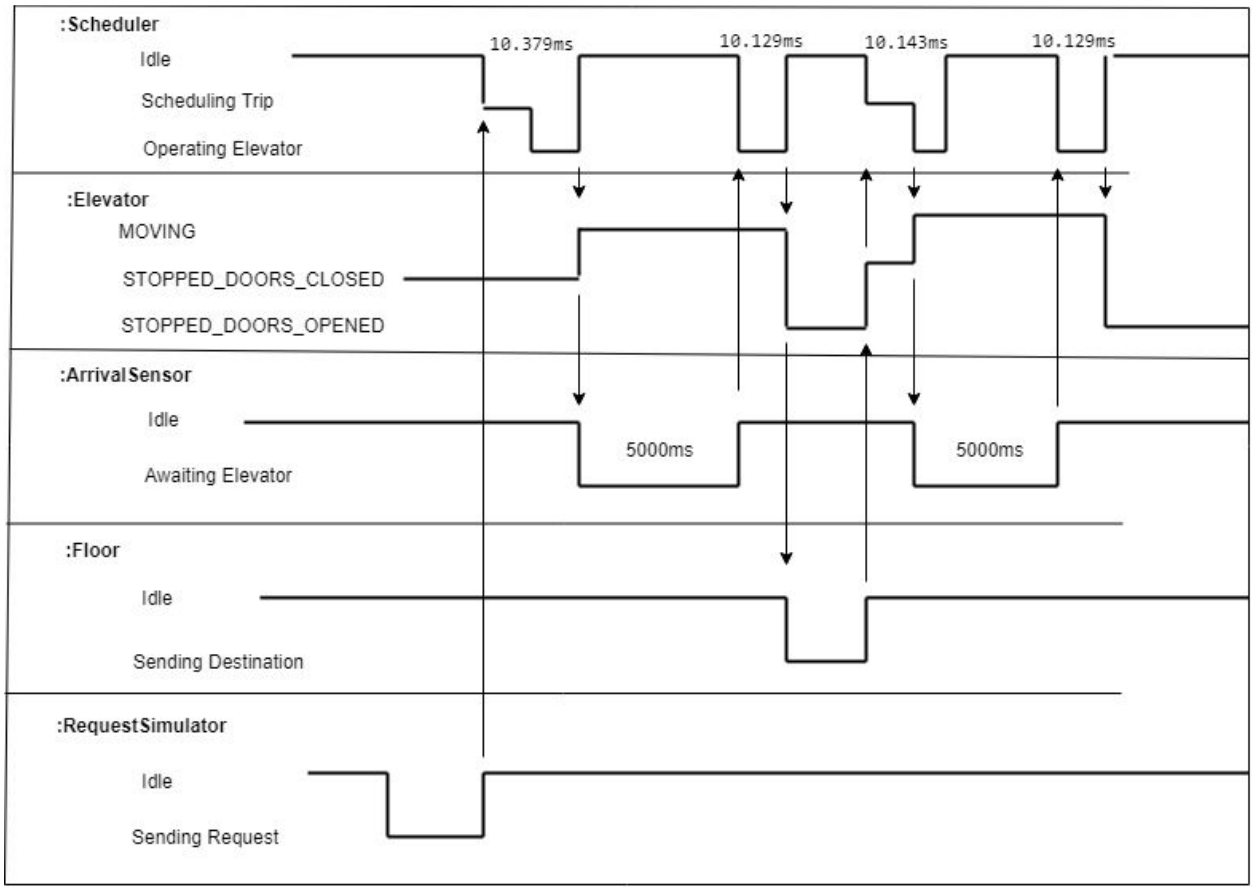

Scheduler Package
SchedulerSubsystem
- elevatorSchedulers: Thread[]
- elevatorQueues: ElevatorQueue[]
- recvSocket: DatagramSocket
- bExit: boolean
+ SchedulerSubSystem()
+ schedule()
- handleElevatorRequest(m: ElevatorRequestMessage)
- handleFloorArrival(m: FloorArivalMessage)
- handleFloorRequest(m: FloorRequestMessage)
+ run()
+ main(args: String[])
Scheduler
- id: int
- elevatorQueue: ElevatorQueue
- sendSocket: DatagramSocket
- faultSuspected: boolean
+ Scheduler(i: int, e: ElevatorQueue)
+ run()
«interface»
Runnable
QueueCleaner
+ queues: ElevatorQueue[]
+ sendSocket: DatagramSocket
+ QueueCleaner(q: ElevatorQueue[])
+ run()
1
1


Floor
- floorNum : int
- upLamp : boolean
- downLamp : boolean
- directionLamps : enum{IDLE, UP, DOWN}[]
RequestSimulator
- sendSocket : DatagramSocket
- floorNum : int
- time : int
+ run() : void
DestinationSender
- elevator : int
- currentFloor : int
- passengers : List<Integer>
+ run() : void
FoorSubsystem
- receiveSocket : DatagramSocket
- floorNum : int
- ourPassengers : int[]
- goingUp : List<Integer>
- goingDown : List<Integer>
+ run() : void
+ requestMessage(FloorMetaMessage) : void
+ metaMessage(FloorMetaMessage) : void
+ arrivalMessage(FloorArrivalMessage) : void
+ terminateMessage(TerminateMessage) : void
+ tearDown() : void
«interface»
Runnable
File
- filename : String
+ getNextRequestTime(int, int) : int
+ getNextMetaMessage(int, int) : MetaMessage
+ getNextRequestMessage(int, int) : ElevatorRequestMessage
+ getNextFaultMessage(int) : FaultMessage
+ getNextTerminateMessage(int) : TerminateMessage
+ getMS(String) : int
Controller
+ updateElevator(Elevator) : void
+ updateFloor(Floor) : void
+ run() : void
FaultSimulator
- sendSocket : DatagramSocket
- faultIndicator : int
- time : int
+ run() : void
FloorManager
+ run()
«interface»
Runnable
ResponseTimer
- elevatorRequestTimes : List<Long>
- floorArrivalRequestTimes : List<Long>
- floorRequestTimes : List<Long>
- receiveSocket : DatagramSocket
+ run() : void
1
0 .. *

State Machine Diagram



Timing Diagram



Set-up/Run Instruction

1. Import the project into eclipse
2. Open src/scheduler/SchedulerSubsystem.java, click run in eclipse
3. Open src/ui/Controller.java, click run in eclipse
4. Run (Optional) the following as junit test:
    a. java -cp build/:vendor/junit-4.10.jar: org.junit.runner.JUnitCore floor.FloorUnitTests
    b. java -cp build/:vendor/junit-4.10.jar: org.junit.runner.JUnitCore elevator.ElevatorManagerTest
    c. java -cp build/:vendor/junit-4.10.jar: org.junit.runner.JUnitCore scheduler.SchedulerTest

Results from your measurements

The results of our system test across 2 computers with significant message traffic are reflected in the following results:

Arrival Sensor Interface: Period - 50, Response Time - 10.129ms

Elevator Buttons Interface: Period - 100, Response Time - 10.143ms
Floor Buttons Interface: Period - 200, Response Time - 10.379ms

Analysis of your system for schedulability

To make a fair RMS viability analysis we can generously inflate these average response times…

Arrival Sensor Interface: Period - 50, Response Time - 20ms
Elevator Buttons Interface: Period - 100, Response Time - 20ms
Floor Buttons Interface: Period - 200, Response Time - 20ms

Using the standard RMS viability inequality for 3 processes:

(20/50) + (20/100) + (20/200) <= 0.78

0.70 < 0.78 ... therefore our system is certainly schedulable by RMS to meet its deadlines consistently

Reflection on your design - what parts do you like and what parts should be redone

The Part We Liked

One of the fundamental pieces of our project is the message package. It's role in the project was to define a message protocol and interface to be used for serialization and deserialization for message sending and receiving. We were very fond of this design because it provided a solid foundation for our subsystem communication. The message package followed principles of SOLID. It was responsible for transferring and receiving messages. The package was also open for extension, every time we needed a new message type, we created a new class that extended our base Message class. The newly created class is now usable with the existing code that accepts this Message interface, which allows for substitution. This would then allow the new class to be used with our existing messenger helpers. This allows for easy extension.

The Part We Didn't Like

We've been less happy with the scheduler package. We've had to go through multiple refactors in the package. The reason being the fact that it contains a lot of responsibilities that all depend on it's model of the elevator queues. This makes it difficult to separate concerns across multiple classes. This becomes a cascading issue once we decided to multithread the scheduler, because contention rises for the access for elevator queues. To help this we've reduced the locks to be per elevator queue but this is still a issue when deciding a which elevator a request should be sent to because all queues need to be evaluated. If it were to be redone, we'd implement a read-write

lock instead on the elevator queues since majority of the operations would be reads for evaluating which queue the request should be added in.