Elevator System
SYSC 3303 A1
Group 6

Amanda Piazza 101143004

Juanita Rodelo 101141857

Nathan MacDiarmid 101098993

Michael Kyrollos 101183521

Matthew Belanger 101144323

Table of Contents

Responsibilities 3
Iteration 0 3
Iteration 1 3
Iteration 2 4
Iteration 3 4
Iteration 4 5
Iteration 5 6
UML Class Diagram 7
Sequence Diagram 8
State Machine Diagrams 9
Elevator State Machine 9
Scheduler State Machine 10
Timing Diagrams 11
Door Stuck Open Fault 11
Door Stuck Closed Fault 11
Elevator Stuck Fault 12
Set Up/Test Instructions 13
If JUnit tests are not working after importing the project 17
Measurement Results 18
Design Reflection 19

Responsibilities

Iteration 0

Amanda Piazza 101143004
- Measuring Elevators (time between each floor)
- Input Data

Juanita Rodelo 101141857
- Measuring Elevators (bottom floor to top floor)
- Input Data

Matthew Belanger 101144323
- Measuring Elevators (stop at every floor)
- Input Data

Michael Kyrollos 101183521
- Measuring Elevators (opening and close of doors)
- Input Data

Nathan MacDiarmid 101098993
- Measuring Elevators (stop at every floor)
- Input Data

Iteration 1

Amanda Piazza 101143004
- UML Class Diagram
- UML Sequence Diagram
- README

Juanita Rodelo 101141857
- UML Class Diagram
- UML Sequence Diagram

Matthew Belanger 101144323
- Unit Tests

Michael Kyrollos 101183521
- Parsing

- Import functions
- README

Nathan MacDiarmid 101098993
- Communication between floor, elevator and scheduler using Java Threads

Iteration 2

Amanda Piazza 101143004
- Floor subsystem
- Elevator subsystem
- Sequence diagram
- State Machine diagram

Juanita Rodelo 101141857
- Floor subsystem
- Elevator subsystem
- Implementation of moving elevator

Matthew Belanger 101144323
- UML diagrams

Michael Kyrollos 101183521
- Unit testing
- README

Nathan MacDiarmid 101098993
- Implementation of state machine

Iteration 3

Amanda Piazza 101143004
- Floor UDP port communication
- Elevator UDP port communication
- Scheduler UDP port communication

Juanita Rodelo 101141857
- Floor UDP port communication
- Elevator UDP port communication
- Scheduler UDP port communication

Matthew Belanger 101144323
- Unit testing
- Implementation of choosing elevator logic

Michael Kyrollos 101183521
- Unit testing
- UML class diagrams

Nathan MacDiarmid 101098993
- Floor UDP port communication
- Elevator UDP port communication
- Scheduler UDP port communication
- Implementation of choosing elevator logic
- UML Sequence Diagram
- README

Iteration 4

Amanda Piazza 101143004
- Elevator stuck timing diagram
- Implementation of door stuck open/close error event

Juanita Rodelo 101141857
- Door stuck open/close error timing diagrams
- Implementation of door open/close stuck error event

Matthew Belanger 101144323
- Elevator stuck timeout error

Michael Kyrollos 101183521
- Unit Testing
- UML class diagrams
- Fault injection
- Fault detection parsing

Nathan MacDiarmid 101098993
- UML Sequence Diagrams
- Elevator stuck timeout error
- README

Iteration 5

Amanda Piazza 101143004
- Implementation of Output Class (UI)

Juanita Rodelo 101141857
- Measurement Analysis
- Output class (UI)

Matthew Belanger 101144323
- Bug Fixes

Michael Kyrollos 101183521
- Unit Testing
- UML Class Diagram

Nathan MacDiarmid 101098993
- UML Sequence Diagram
- Scheduler State Machine
- Bug Fixes

UML Class Diagram

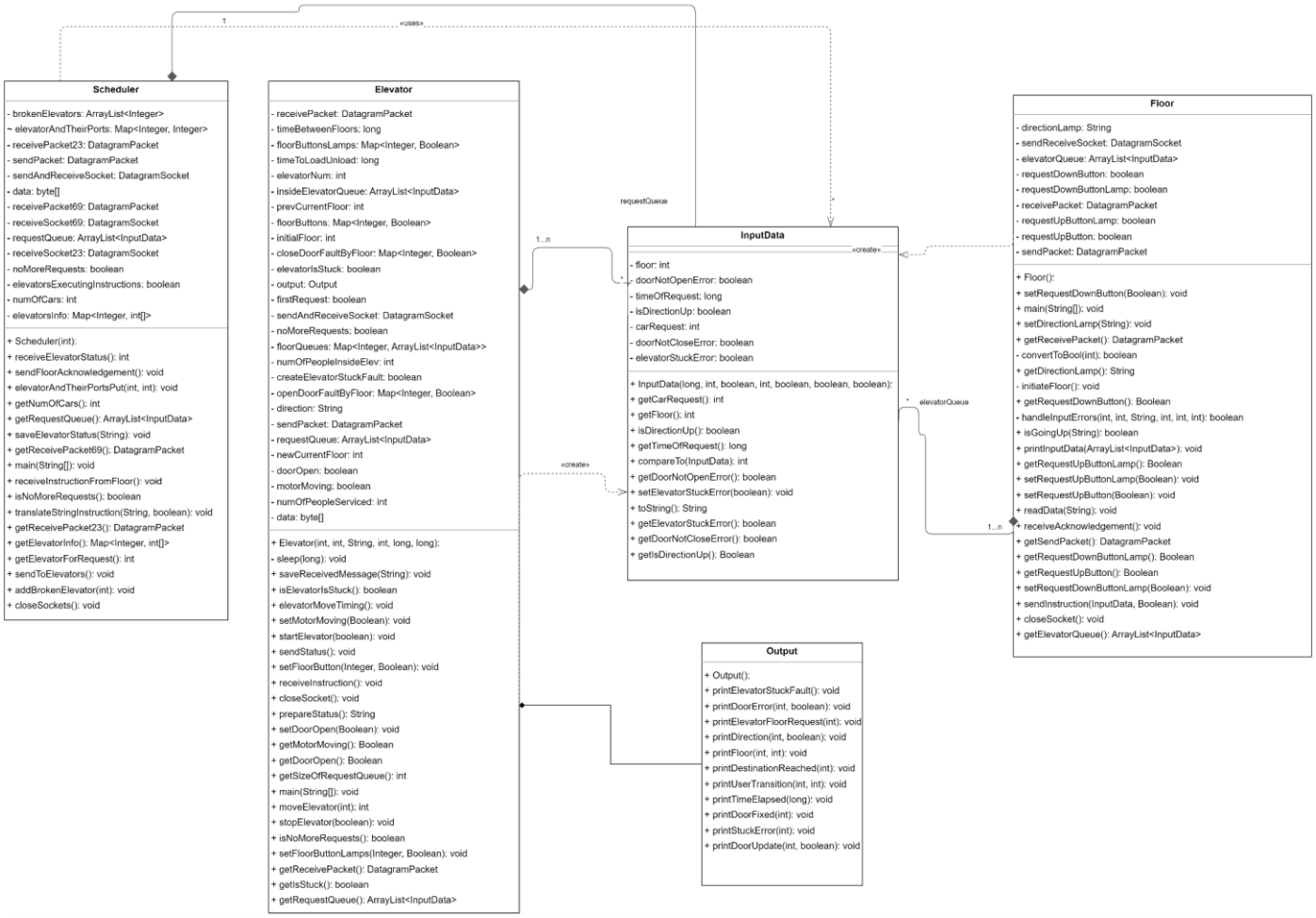


Figure 1: UML Class Diagram for System

Sequence Diagram

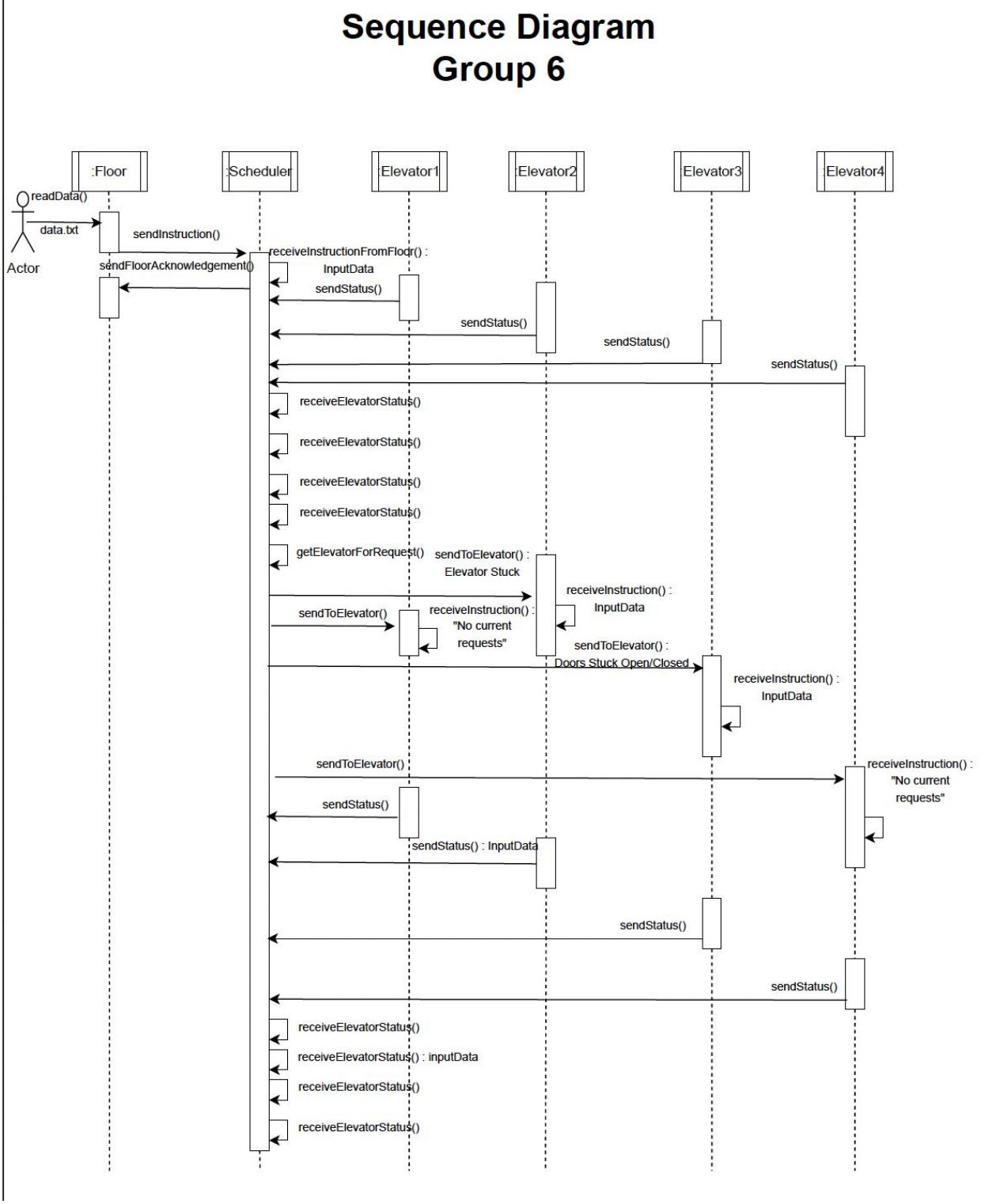


Figure 2: Sequence Diagram When Running with Four Elevators and Errors

State Machine Diagrams

Elevator State Machine

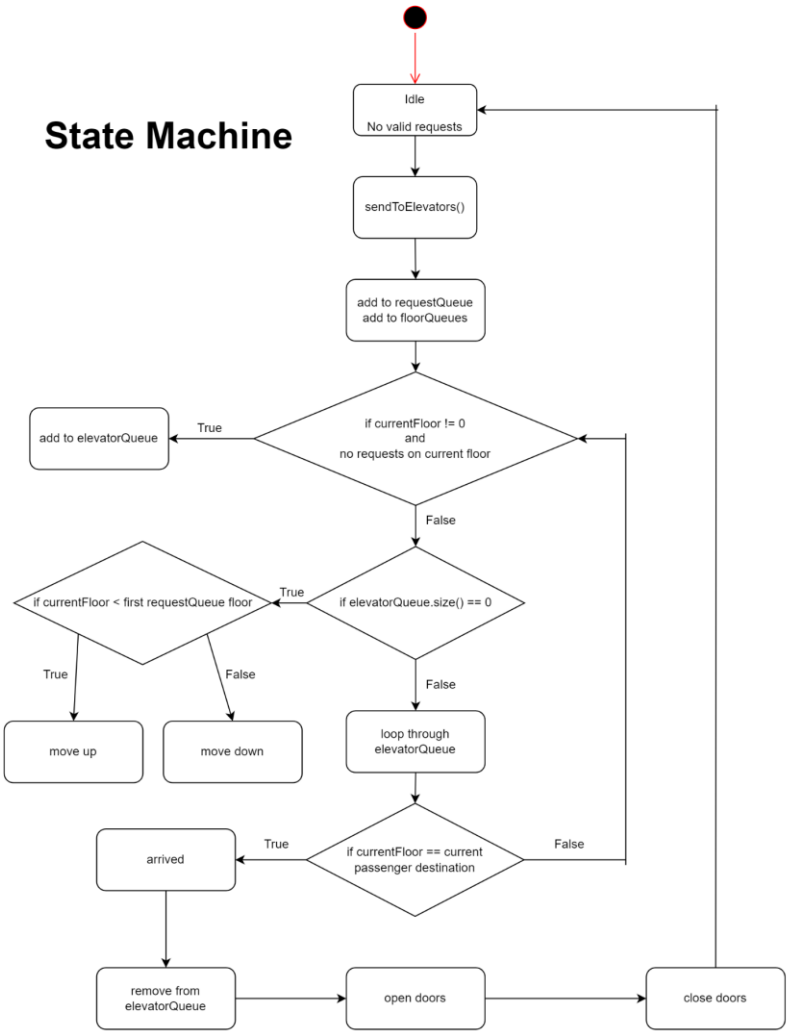


Figure 3: Elevator State Machine

Scheduler State Machine

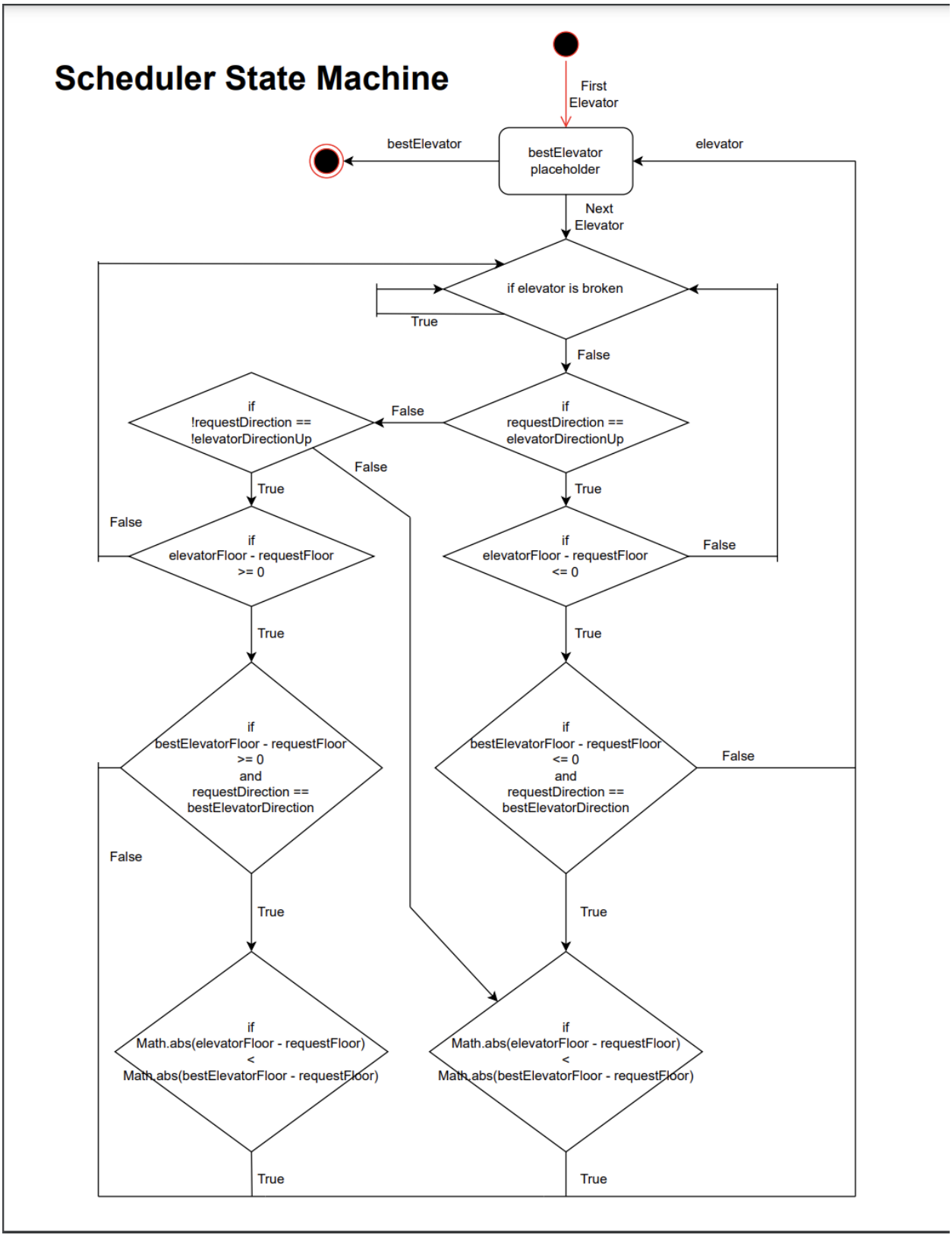


Figure 4: Scheduler State Machine

Timing Diagrams

Door Stuck Open Fault

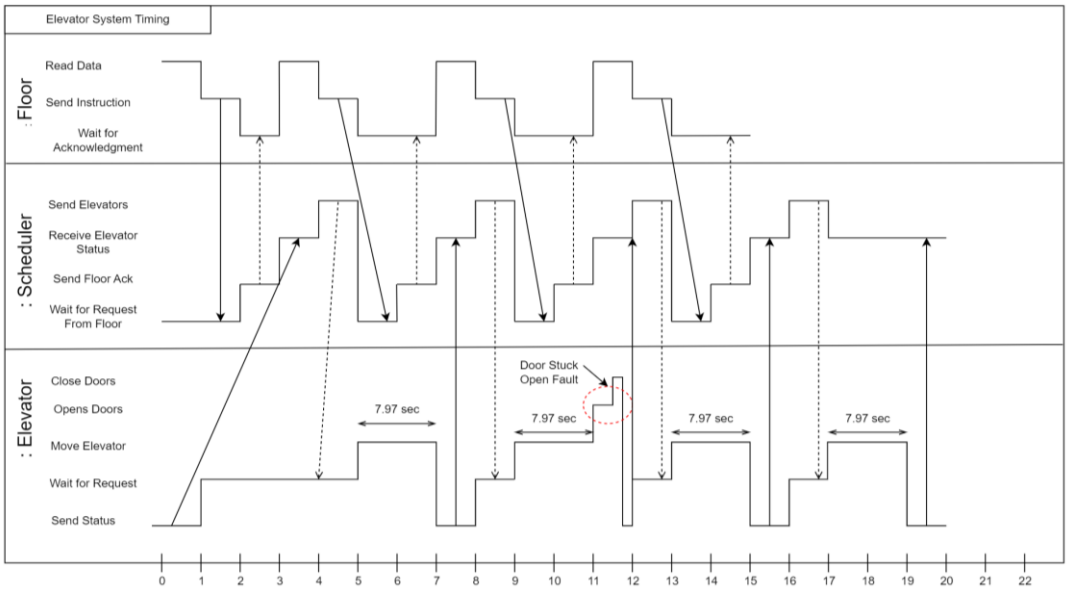


Figure 5: Door Stuck Open Fault

Door Stuck Closed Fault

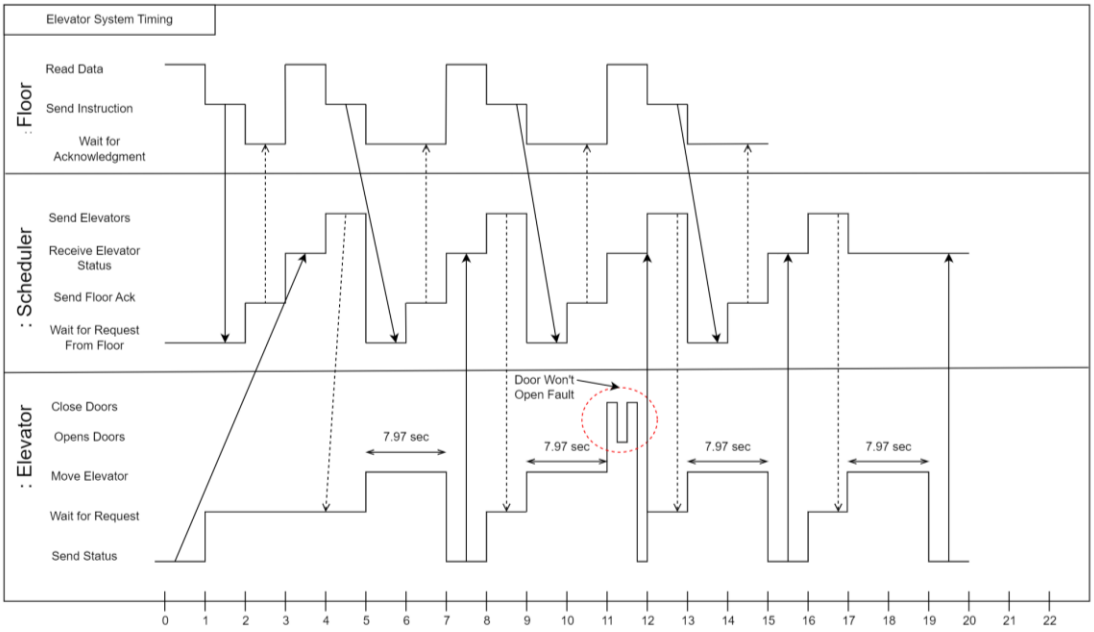


Figure 6: Door Stuck Open Fault

Elevator Stuck Fault

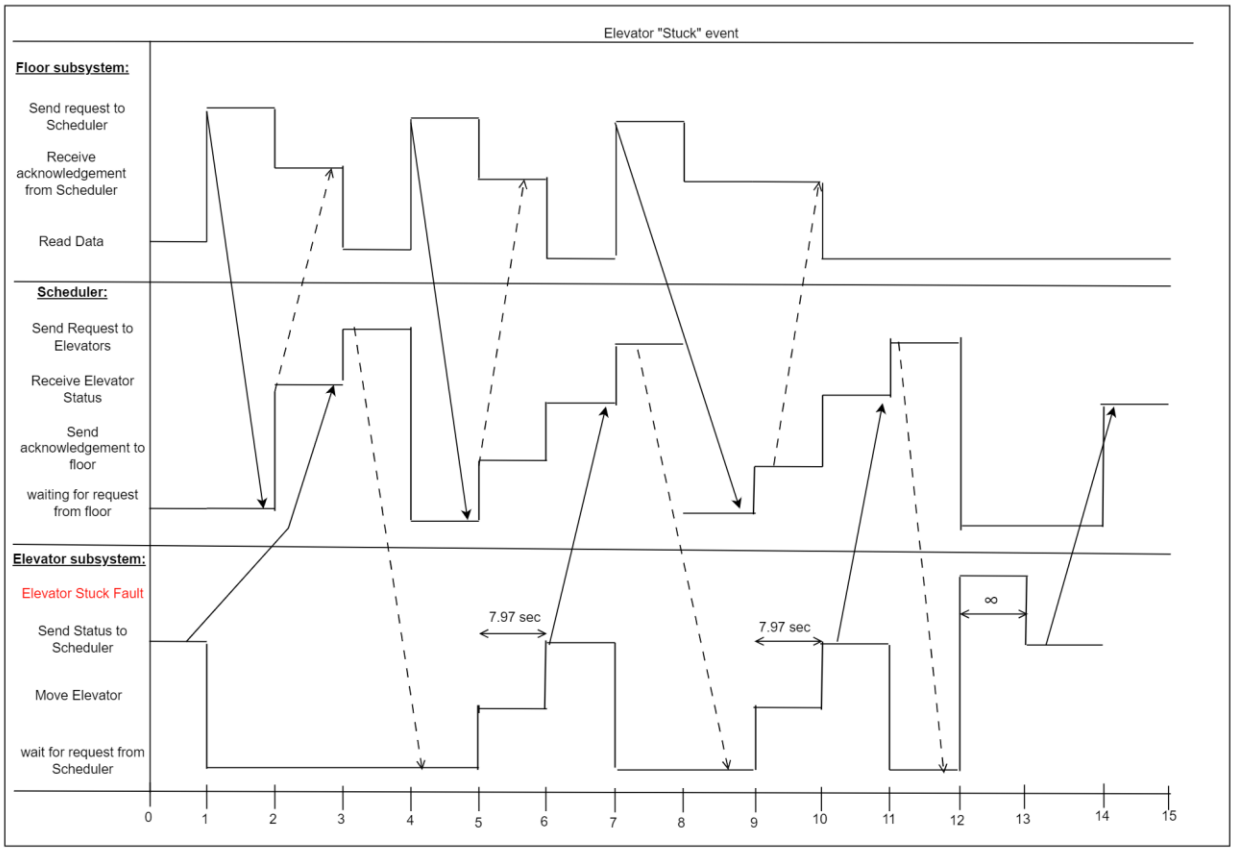
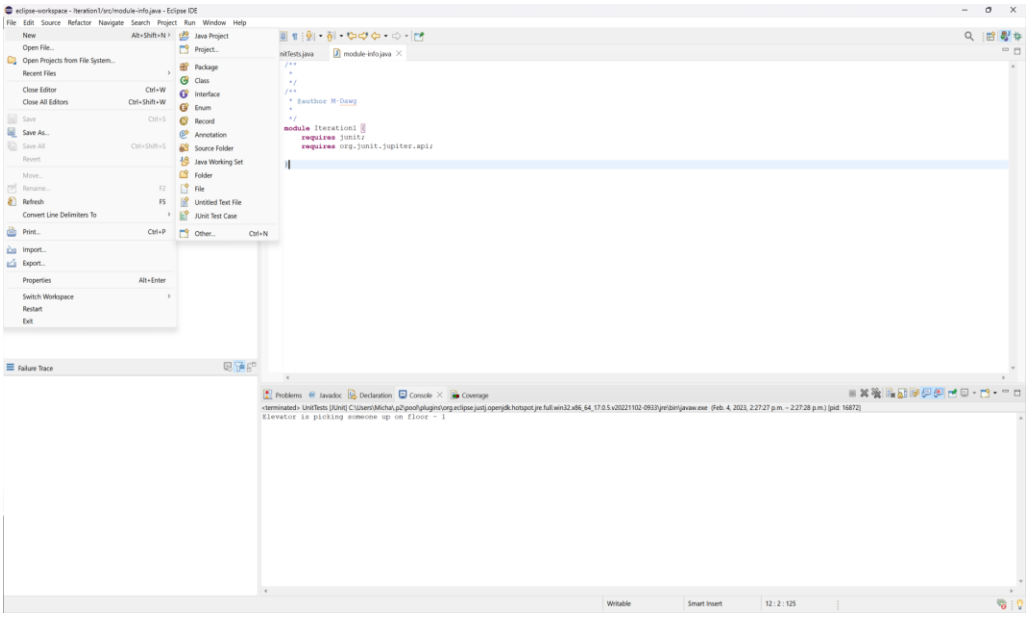


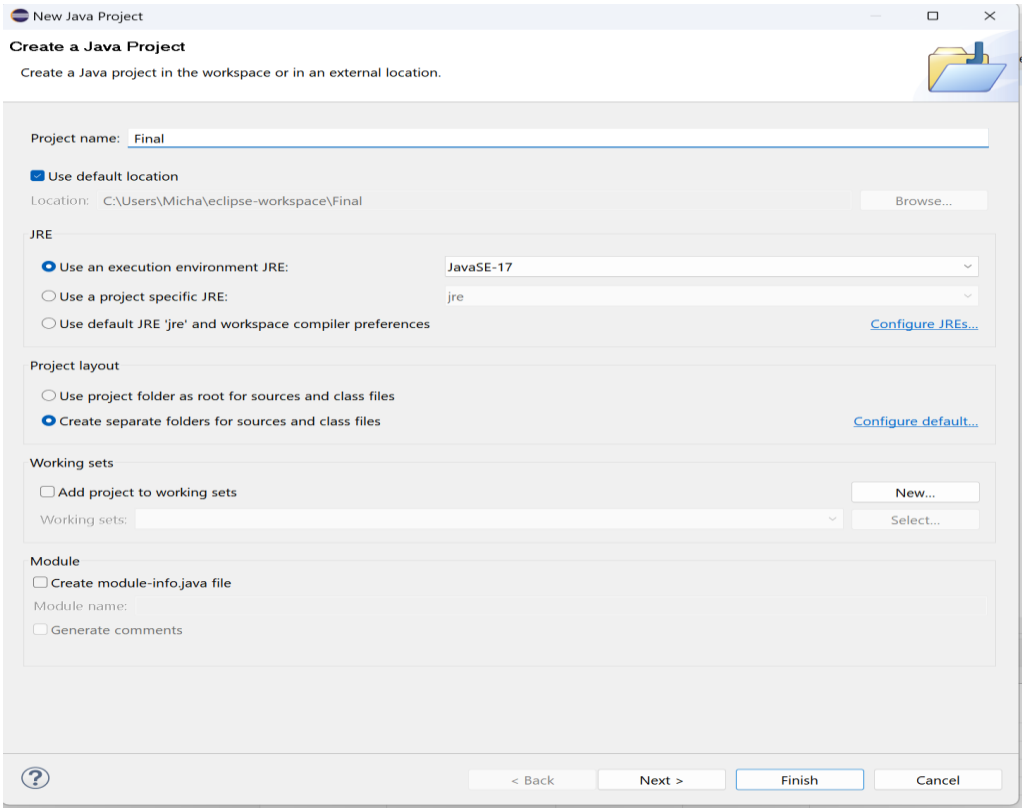
Figure 7: Elevator Stuck Fault

Set Up/Test Instructions

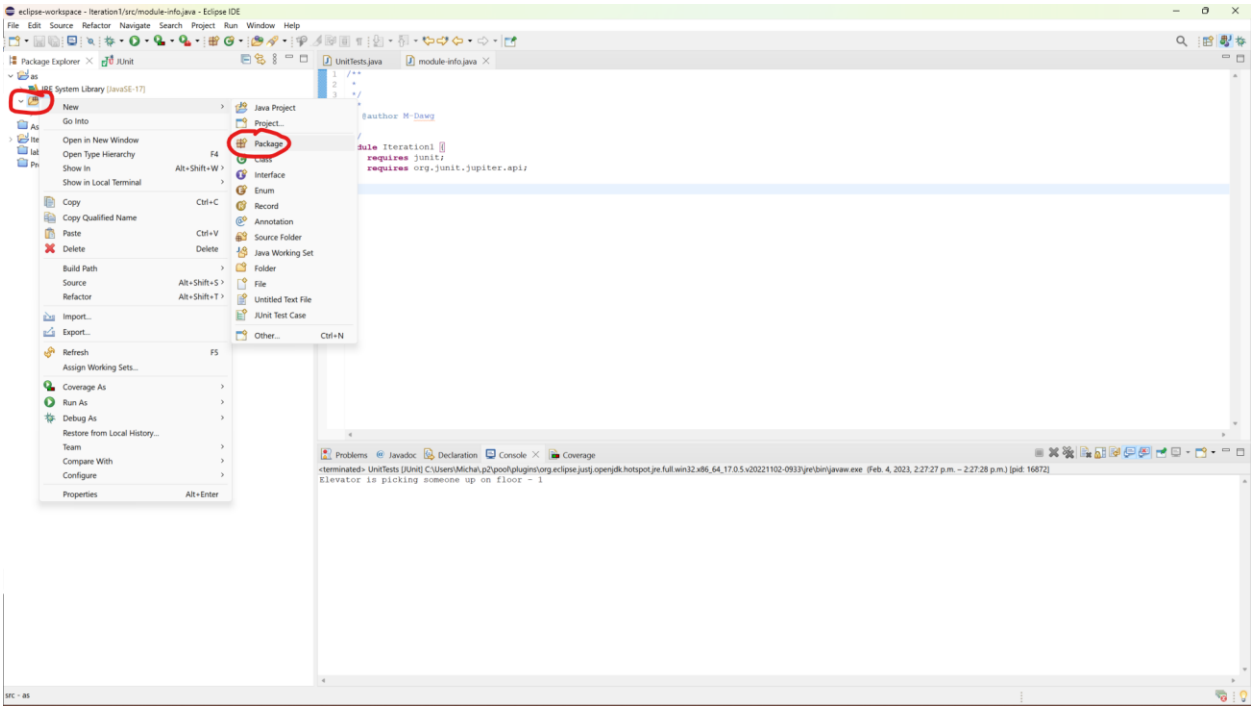
1. Open Eclipse and create a new project (File -> New -> Java Project)



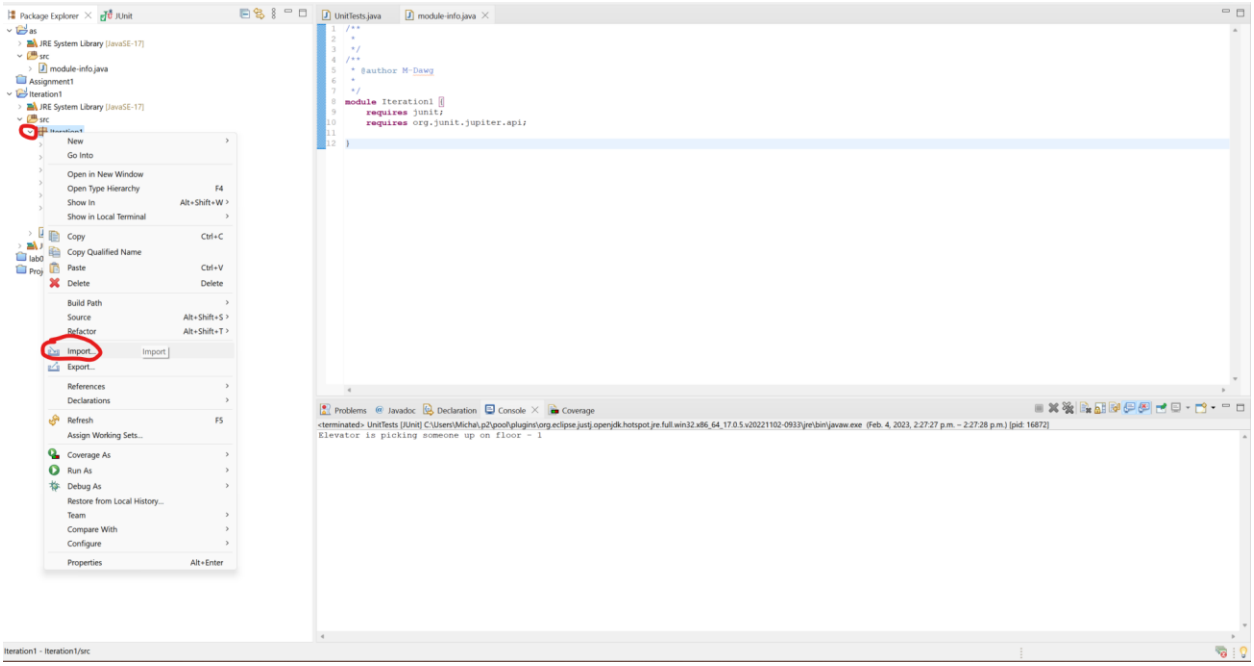
2. Name the project Final and click Finish



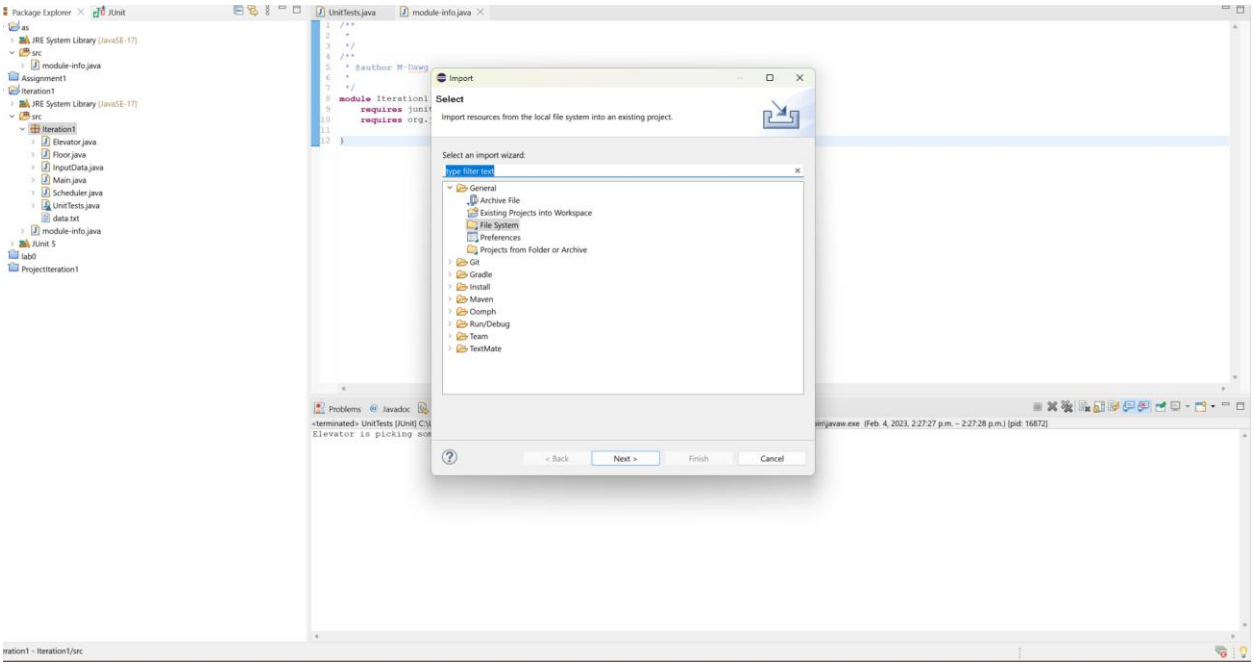
3. Create a package by right clicking on the /src directory within the project folder in Eclipse, name it Final



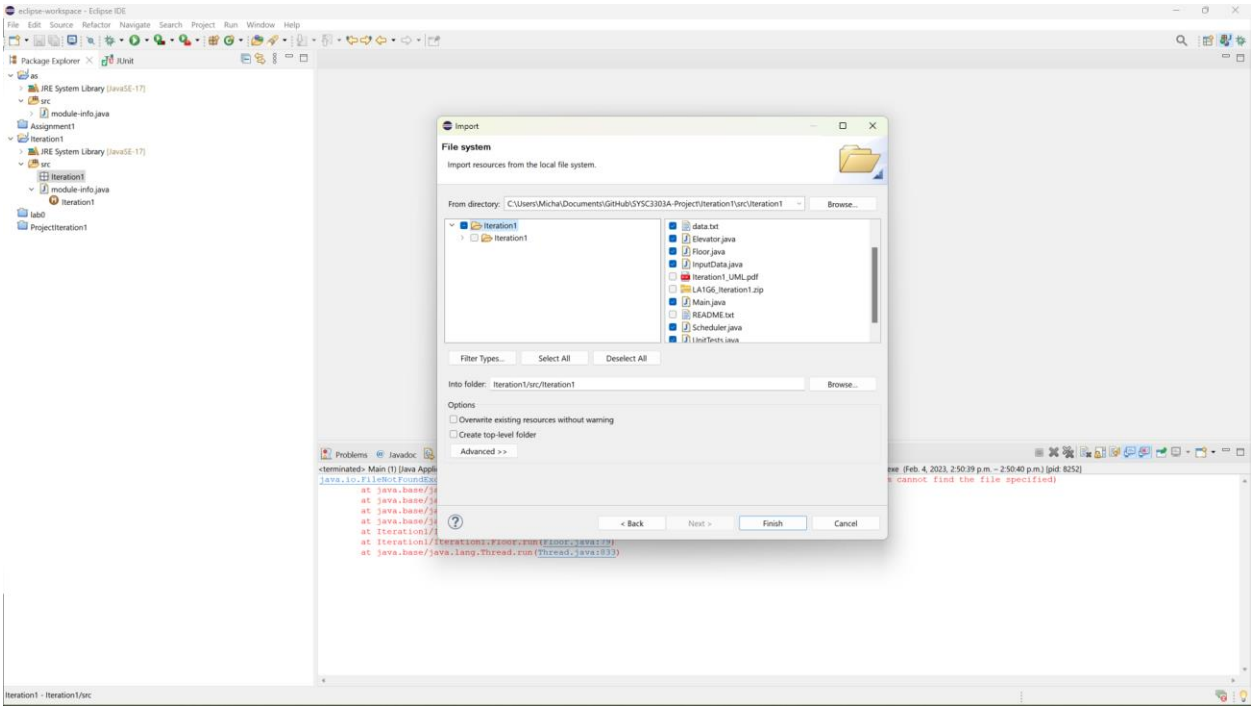
4. Unzip LA1G6_Final.zip
5. Right click on the package and click import.



6. Import as *File Systems*

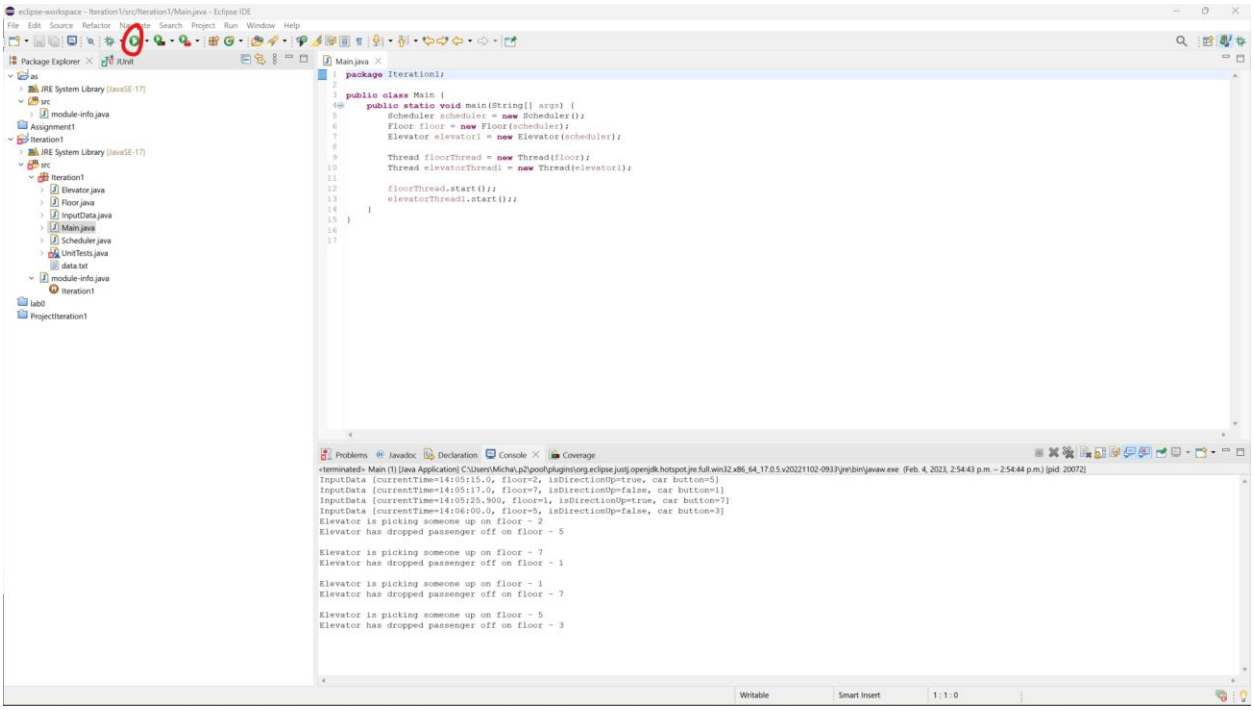


7. Locate the directory in which the Java src files can be found. (From the Extracted zip in Step 4). From the GitHub Repo, it would be *\SYSC3303A-Project\Final\src\Final*



8. Include the following files: Elevator.java, Floor.java, InputData.java, Scheduler.java, ElevatorTest.java, Floor.java, SchedulerTest.java,

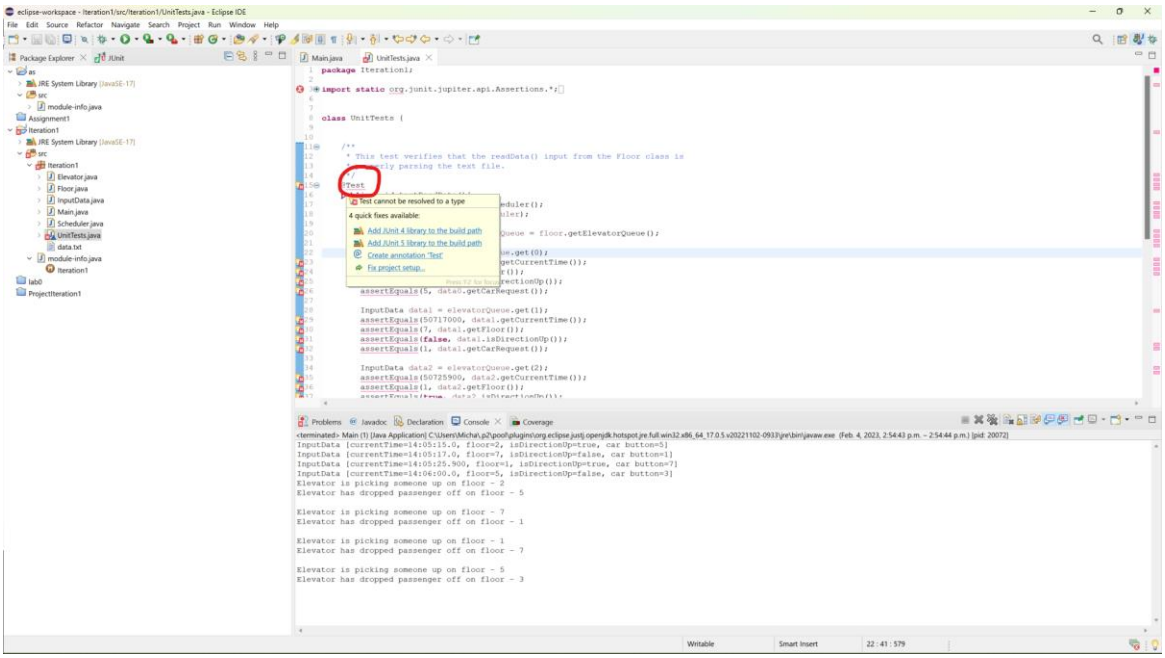
9. Run the program with the green play button at the top



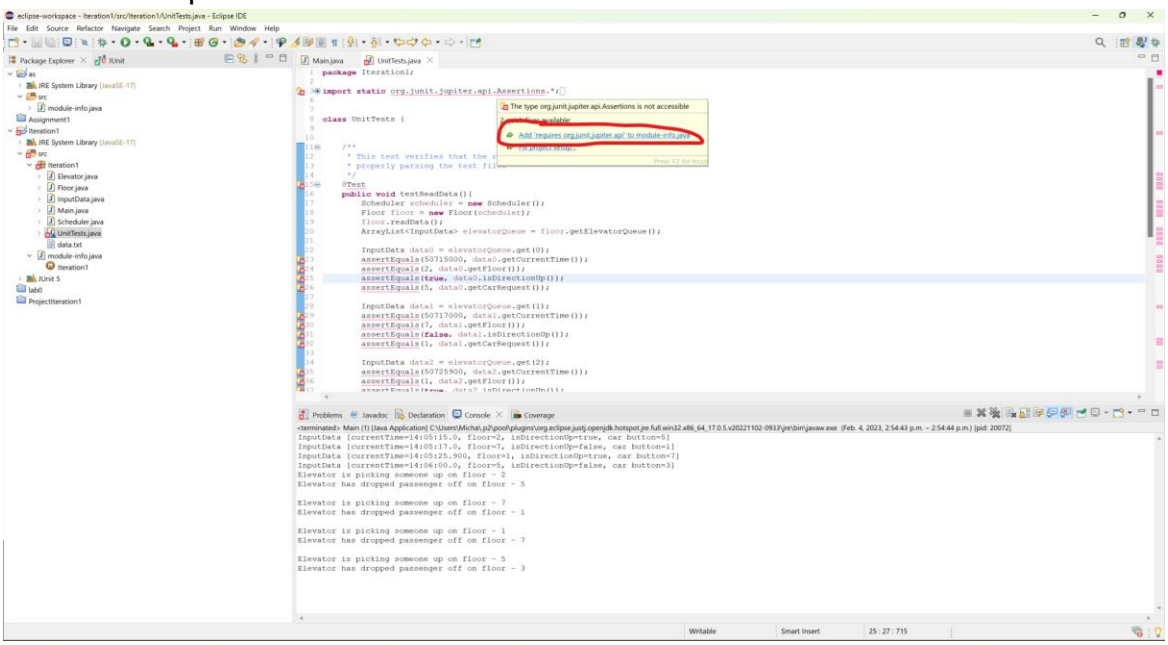
If JUnit tests are not working after importing the project

If there is an *x* symbol beside any of the testing files, follow the instructions below.

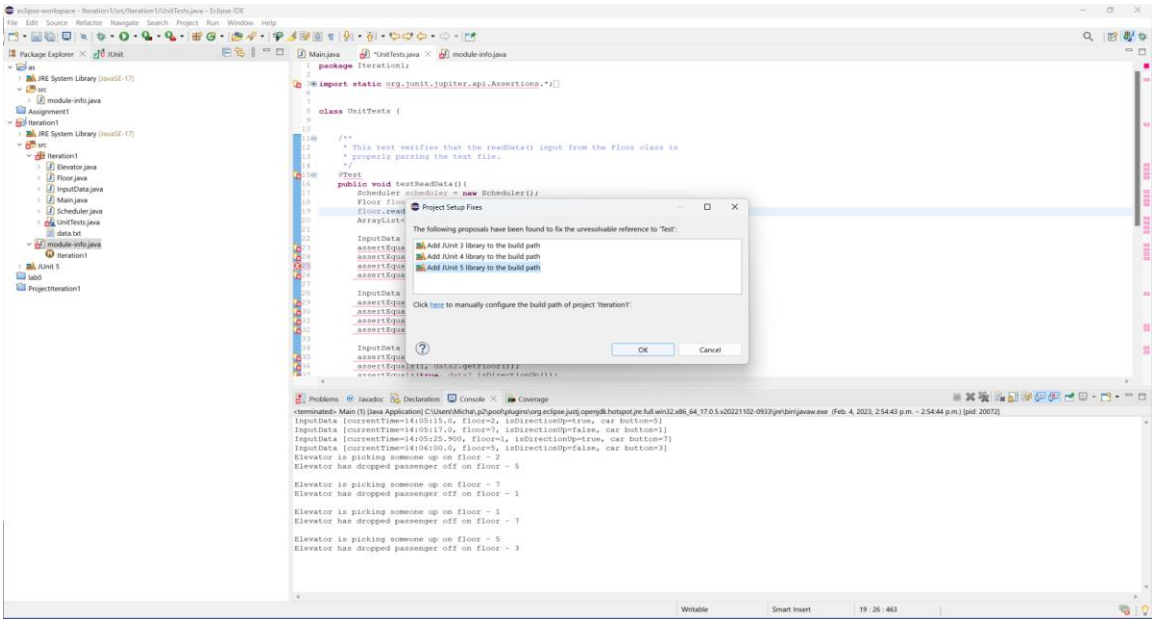
1. Open a testing file and hover the mouse over the @Test code that is underlined red. Click on *Add JUnit 5 library to the build path*



2. Hover over the import statement at the top of the file and click on the circled button in the picture below.



3. Hover over the @Test code that is underlined red. Click on *Fix Project Setup*. Click on *Add JUnit 5 library to the build path* -> choose *org.junit.jupiter.api.Test*



4. Navigate to the *module-info.java* file and open it. Hover over the red underlined code and click on the option to change the build path. The tests should be working now.

Measurement Results

We have implemented several optimizations in our elevator system to accurately measure its performance. We added a timer in our system that starts as soon as the first input request is received and stops when the last request has been brought to its destination floor. This allows us to measure the time it takes for the Floor subsystem to read, save, and send requests to the scheduler subsystem. The scheduler then replies to the Floor subsystem and saves these requests in a map. All the elevators send a message to the scheduler to notify them that they are ready to accept requests and the scheduler replies with either a request or some other message. This process continues for as many requests as the scheduler receives from the floor. The elevators then save their requests in multiple data types and start to move accordingly.

In Iteration 0, we conducted measurements of different aspects of the elevator system, including the time it takes for elevators to move between each floor and the time it takes to load and unload passengers. These measurements, with an average of 7.90 seconds and 2.70 seconds respectively, were considered during the system design to ensure accurate timing when implementing delays. Every time the elevator moves between floors, the elevator calls the sleep() method for 7.9 seconds. Every time we

stop the elevator to pick up or drop off a passenger, the sleep() method is called for 2.7 seconds. Additionally, we have considered additional delays that may occur due to faults, such as doors getting stuck open or closed. To account for this, we have added a time delay of 4.00 seconds in case of such faults. When the elevator gets stuck between floors, we have a timeout() method that detects this and goes off if the elevator takes more time than the time specified for the elevator to move between floors. If this occurs, then the elevator will be considered out of service and will stop serving requests. The rest of the requests will be sent to all other elevators in a timely and efficient manner.

The test case that we will be using for the final iteration includes 7 requests that span 22 floors with 4 elevators servicing these requests. We intentionally injected faults to measure the program's performance. There are four requests that have an open- or closed-door fault and one request that will signal an elevator stuck fault. With these requests, the system takes a total of approximately 534 seconds to service. With just one request to service (from floor 1 to 6), the program takes 45 seconds to run. With just one request to service (from floor 1 to 22), the program takes 171 seconds to service which aligns with our measurements as it takes 21 floor movements for the elevator (21 x 7.90 = 165.90 seconds). The rest of the time represents how long it takes the subsystems to communicate to each other via UDP and how it takes the subsystems to process the information.

Throughout the design and implementation of our elevator system, we have attempted to prioritize efficiency and reliability by reducing cyclomatic complexity of methods, choosing appropriate data structures, and optimizing system resources. We also tried to make optimal use of the system resources such as CPU, memory and disk I/O as well as parallelizing the subsystem's execution. Since we chose a simplistic I/O method by keeping the output in the display console, this resulted in our program being quicker since there are fewer operations within the output of our program. With all of this kept in mind as we designed and implemented our program, these timing measurements show that our system is efficient in the way that it handles and services requests for the elevators.

Design Reflection

The team decided to add only essential information to the UDP messages between the floor and the elevator subsystems to the scheduler. These essentials include the elevator's status, which tracks the number of passengers it has serviced, allowing the scheduler to distribute the passengers evenly across all elevators. Additionally, the elevator's current direction is sent to the scheduler to help decide which

elevator will be sent to service a request in the most efficient way. Also, a Boolean value is appended to each instruction sent from the floor subsystem to the scheduler, signaling whether the instruction is the final request, ensuring the UDP sockets are terminated appropriately when the requests have been serviced. The team found that after developing the communication using threads and instances, UDP messages is the most efficient method for our goals and requirements of the project.

The team agreed to handle faults using integers representing Boolean values inside each request in data.txt, 1 representing the occurrence of a fault and 0 representing a normal request. Floor.java parses this information, along with the rest of the request and adds it into the elevator's request. This method is highly effective and concise as it adds to the parameters of a request and does not require a substantial amount of code. In a future iteration, the team may consider adding a separate method for each fault to randomly generate elevator errors at different points in the elevators' execution.

The team deliberated and opted to add the functionality of the elevator the display console separates from the floor subsystem in our implementation of Iteration 5, which implies using an instance of our display console class Output.java. In future iterations we would explore adding this output functionality to the floor subsystem to allow for all information to be passed using UDP messages. Currently, we close the sockets for the UDP port messages based on a fixed list of requests stored in data.txt, allowing us to know when the last request is received. However, in future iterations, if requests are received in real-time, we would modify our approach to keep sockets open until the user ends the program, as we would not know when the last request will occur.

Overall, given the design and requirement changes throughout iterations, the total time allowed, and the resources provided, the team developed an effective and clear approach to the elevator subsystem. The team used collaboration and communication skills to work together to brainstorm, develop and test the project. We also used technical skills to complete areas and requirements individually to allow the project to rapidly progress.