**Video Link: https://youtu.be/PYbIe8V0qRc**

**List of Use Cases**

1. **UC1: Setup Simulation**
2. **UC2: Start/Pause/Resume/Stop Simulation**
3. **UC3: Request Elevator from a Floor**
4. **UC4: Request Floor from Inside an Elevator**
5. **UC5: Press Open/Close Door Button**
6. **UC6: Press Help Button**
7. **UC7: Trigger Safety Event**
    - Fire, Overload, Door Obstacle, Power Out.

**UC1: Setup Simulation**

- **Name:** Setup Simulation
- **Primary Actor(s):** Administrator
- **Stakeholders:**
    - **Administrator** – wants to configure the simulation.
    - **Raven Elevators Inc.** – ensures the simulator is flexible to different building/elevator/passenger configurations.
- **Pre-condition(s):**

1. The simulation application is launched and in an initial "idle" or "not configured" state.

2. No existing configuration is active or the user is in "setup mode."

- **Success Guarantee (Post-conditions):**

1. The system is configured with the specified number of floors (N) and elevators (M).

2. Passenger action scripts (who does what at which time step) are stored.

3. Safety event scripts (which safety event occurs at which time step) are also recorded.

- **Main Success Scenario (Typical Flow):**

1. Administrator selects “Setup” mode in the GUI.

2. System prompts for the number of floors (N) and elevators (M).

3. Administrator enters valid N and M values.

4. System prompts for passenger behaviors (e.g., “At time t=20, Passenger #1 presses Down button on Floor 3”).

5. Administrator enters or edits the list of passenger actions.

6. System prompts for safety events (e.g., “At time t=50, Fire alarm triggers”).

7. Administrator configures the safety events as needed.

8. Administrator reviews the entire configuration and confirms.

9. System validates the configuration and displays a success message.

- **Extensions (Alternate Flows / Exceptions):**
    - **E1:** Invalid input (e.g., negative or zero floors/elevators)
        - **Action:** System displays an error message and re-prompts for valid input.
    - **E2:** Administrator cancels setup midway
        - **Action:** System discards changes and remains in the idle or previously configured state.

---

**UC2: Start/Pause/Resume/Stop Simulation**

- **Name:** Start/Pause/Resume/Stop Simulation
- **Primary Actor(s):** Administrator
- **Stakeholders:**
    - **Administrator** – wants to control the simulation timeline.
- **Pre-condition(s):**

1. Simulation has been configured (UC1 completed) or loaded with a valid scenario.

2. The simulation is in a state that allows the requested action (e.g., you cannot “resume” if it has never been “started”).

- **Success Guarantee (Post-conditions):**

1. On “Start,” the simulation begins time-step execution from the beginning.

2. On “Pause,” the simulation halts time-step progression but retains all state.

3. On “Resume,” the simulation continues from the paused state.

4. On “Stop,” the simulation either fully terminates or resets to an idle state (implementation-specific).

- **Main Success Scenario (Typical Flow):**

1. Administrator clicks “Start” from the GUI.

2. System transitions from “Configured/Idle” to “Running,” time steps advance, events get processed.

3. Administrator may click “Pause” at any point, freezing time-step advancement.

4. Administrator may click “Resume” to continue.

5. Administrator may click “Stop,” ending the simulation run.

6. System logs the user action (start, pause, resume, or stop) in the console or event log.

- **Extensions (Alternate Flows / Exceptions):**
    - **E1:** Attempting to start without a valid configuration.
        - **Action:** System displays an error that no valid configuration exists.
    - **E2:** Administrator tries to pause or resume when the simulation is already in that state.
        - **Action:** System ignores or displays “Invalid action” message.

---

**UC3: Request Elevator from a Floor**

- **Name:** Request Elevator from a Floor
- **Primary Actor(s):** Passenger (on a particular floor)
- **Stakeholders:**
    - **Passenger** – wishes to be transported to another floor.

- **Elevator Control System** – must manage and respond to floor requests.

- **Pre-condition(s):**

1. The simulation is running (not paused or stopped).

2. Passenger is physically at a floor where an up/down button is available.

- **Success Guarantee (Post-conditions):**

1. The system registers that an elevator has been requested to go up or down from that floor.

2. The corresponding floor button (Up or Down) is lit until an elevator arrives.

3. An elevator is eventually dispatched to that floor (if not interrupted by safety events).

- **Main Success Scenario (Typical Flow):**

1. Passenger presses “Up” (or “Down”) button on the floor.

2. The floor button lights up.

3. The system (via its elevator controller) logs the request.

4. The scheduler assigns an elevator (could be immediate or waits if all are busy).

5. Elevator arrives, the floor button light turns off, the bell rings, and the doors open for boarding.

- **Extensions (Alternate Flows / Exceptions):**
  - **E1:** Passenger presses the wrong button (e.g., “Up” instead of “Down”).
    - **Action:** System treats it as a valid request. If passenger ignores it, that’s passenger’s choice.
  - **E2:** System is overridden by a safety event (Fire, Power Out, etc.).
    - **Action:** The request may be delayed or canceled; the system logs a safety event message.

---

**UC4: Request Floor from Inside an Elevator**

- **Name:** Request Floor from Inside an Elevator
- **Primary Actor(s):** Passenger (already inside the elevator)

- **Stakeholders:**
    - **Passenger** – wants to travel to a particular destination floor.
    - **Elevator Control System** – must queue up or schedule the requested destination.
- **Pre-condition(s):**

1. Passenger is inside an elevator that has its doors closed or open.
2. Elevator control panel is active (no immediate safety override).

- **Success Guarantee (Post-conditions):**

1. The elevator logs the requested floor in its internal queue.
2. The passenger eventually arrives at the requested floor, provided no safety event overrides it.

- **Main Success Scenario (Typical Flow):**

1. Passenger presses the floor selection button (e.g., Floor 4).
2. The elevator acknowledges the request (button may light up).
3. Elevator either continues its current path or adjusts route to include the new floor.
4. When the elevator reaches the requested floor, it stops, opens doors, and the passenger can disembark.

- **Extensions (Alternate Flows / Exceptions):**
    - **E1:** Passenger presses multiple floor buttons in succession.
        - **Action:** Elevator adds all those floors to its "service queue" and visits them in an optimized or in-sequence route.
    - **E2:** Elevator is in emergency or overload mode.
        - **Action:** Floor requests are temporarily ignored or queued until the condition is resolved.

---

**UC5: Press Open/Close Door Button**

- **Name:** Press Open/Close Door Button

- **Primary Actor(s):** Passenger (inside the elevator)
- **Stakeholders:**
    - **Passenger** – wants to keep doors open longer or close them sooner.
- **Pre-condition(s):**

1. Elevator is at a floor (doors may be opening, open, or closing).
2. Passenger is inside the elevator and has access to the control panel.

- **Success Guarantee (Post-conditions):**

1. Doors open longer if "Door Open" is held or pressed.
2. Doors close sooner (if safe) if "Door Close" is pressed.
3. The elevator's door logic respects safety conditions (e.g., obstacle sensor).

- **Main Success Scenario (Typical Flow):**

1. Elevator arrives at a floor and starts its default door operation (open for 10 seconds).
2. Passenger presses (and possibly holds) the "Door Open" button.
3. Elevator's door remains open beyond the default time as long as the button is held.
4. Passenger releases the button; elevator closes door after a short grace period.
5. (Alternatively) passenger presses "Door Close" button to close doors before the default time ends.
6. Elevator closes doors, provided no obstacle or override.

- **Extensions (Alternate Flows / Exceptions):**
    - **E1:** Door obstacle is detected while trying to close.
        - **Action:** Elevator re-opens doors automatically (see UC7 for details if it's repeated).

---

**UC6: Press Help Button**

- **Name:** Press Help Button
- **Primary Actor(s):** Passenger (inside the elevator)
- **Stakeholders:**

- **Passenger** – may need emergency assistance or is in distress.
- **Building Safety Service** – must respond to help requests.
- **Emergency Services (911)** – fallback if no response from passenger or building safety.

- **Pre-condition(s):**

1. Passenger is inside the elevator.

2. Elevator is operational (not entirely powered down, though it can be in an emergency state).

- **Success Guarantee (Post-conditions):**

1. A help alarm signal is sent to the control system and building safety.

2. A voice connection attempt is made.

3. If no response from building safety or passenger for 5 seconds, system auto-dials 911.

- **Main Success Scenario (Typical Flow):**

1. Passenger presses the “Help” button.

2. Elevator sends a “Help Alarm” signal to the control system.

3. The system connects passenger to building safety.

4. Building safety responds; passenger clarifies the situation.

5. The system logs the event, and the elevator continues normal operation if no other issue is found.

- **Extensions (Alternate Flows / Exceptions):**
  - **E1:** No response from building safety or passenger.
    - **Action:** After 5 seconds, elevator system dials 911 automatically and logs a “911 call placed” event.

---

**UC7: Trigger Safety Event**

(*This covers Fire, Overload, Door Obstacle Repeated, Power Out.*)

- **Name:** Trigger Safety Event
- **Primary Actor(s):** Building Safety System (for Fire, Power Out), Elevator Sensors (for Overload, Door Obstacle), Elevator Control System (for Door Obstacle, Power Out), Passenger(for Door Obstacle)
- **Stakeholders:**
    - **Passengers** – may be asked to disembark or reduce load, or must be evacuated.
    - **Elevator Control System** – must respond immediately, overriding normal operation.
- **Pre-condition(s):**

1. Simulation is running or in a state where events can still occur.

2. Specific condition is detected (e.g., Fire alarm triggers, weight sensor triggers Overload, door sensor triggers repeated obstacle, etc.).

- **Success Guarantee (Post-conditions):**

1. The system overrides normal elevator operations as needed (move to safe floor, hold doors, alarm messages, etc.).

2. Passengers are notified (audio + text display).

3. The system remains in emergency mode until the condition is cleared or the simulation stops.

- **Main Success Scenario (Typical Flow)** — *Fire Example*:

1. Fire alarm is triggered at time t in the building or elevator.

2. Control system receives the “Fire” event signal.

3. All elevators are commanded to go to a designated safe floor.

4. Audio and text messages instruct passengers to disembark upon arrival.

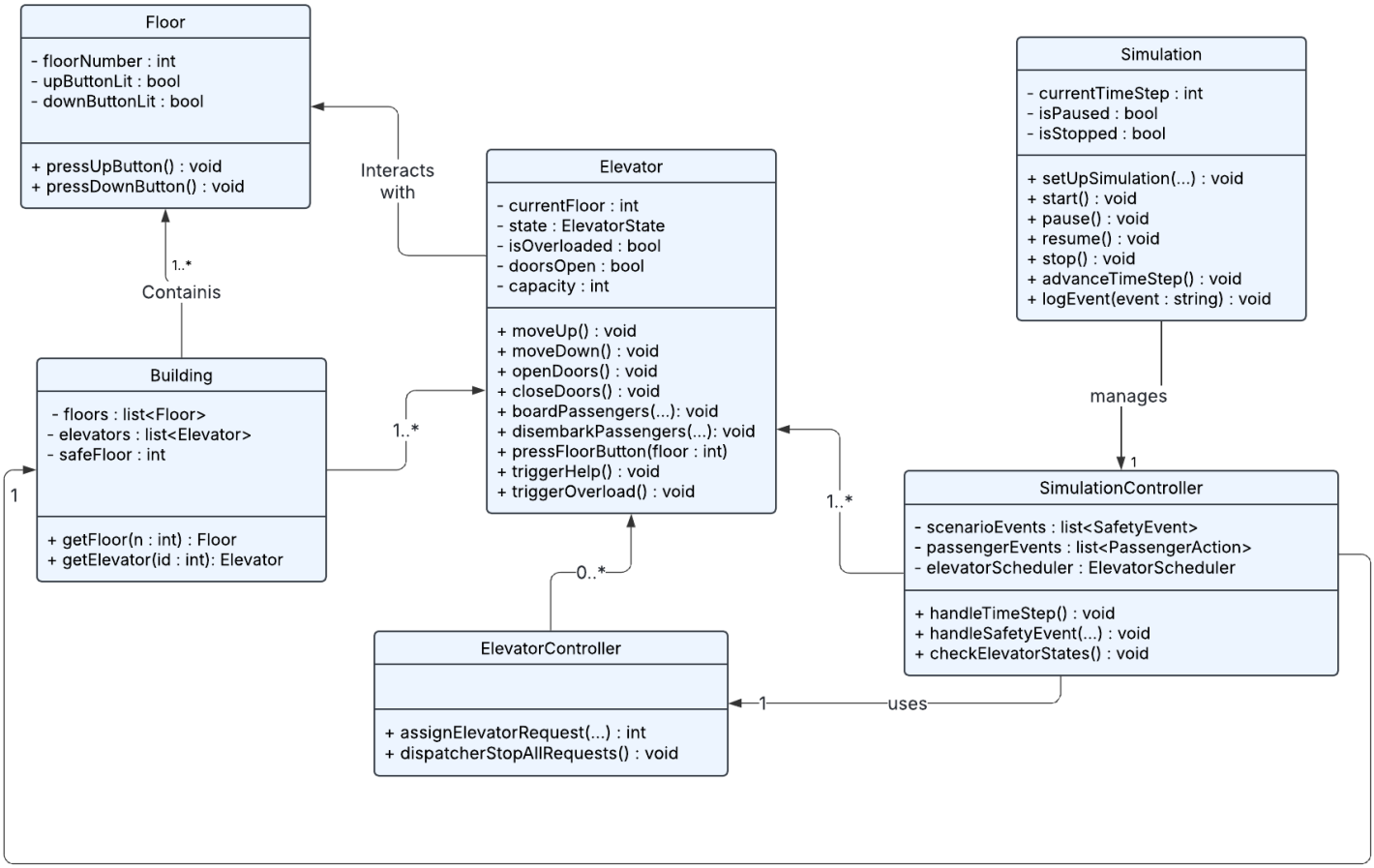
5. Elevators remain idle at the safe floor until the alarm is cleared.

- **Extensions (Alternate Flows / Exceptions)**:
    - **E1 (Overload)**: Overload alarm triggers inside elevator.

        - **Action:** Elevator does not move; an audio/text message asks passengers to reduce load. Notify Building safety if alarm prolongs.
    - **E2 (Power Out)**: Power Out event triggers.
        - **Action:** System logs “power outage,” uses backup power to move elevator(s) to safe floor, then instructs all passengers to exit.
    - **E3 (Door Obstacle)**: Repeated door obstacle events cause warning messages and door remains open.
        - **Action:** Elevator does not move or close; audio/text message asks passenger to remove obstacle. Notify Building safety if alarm prolongs.

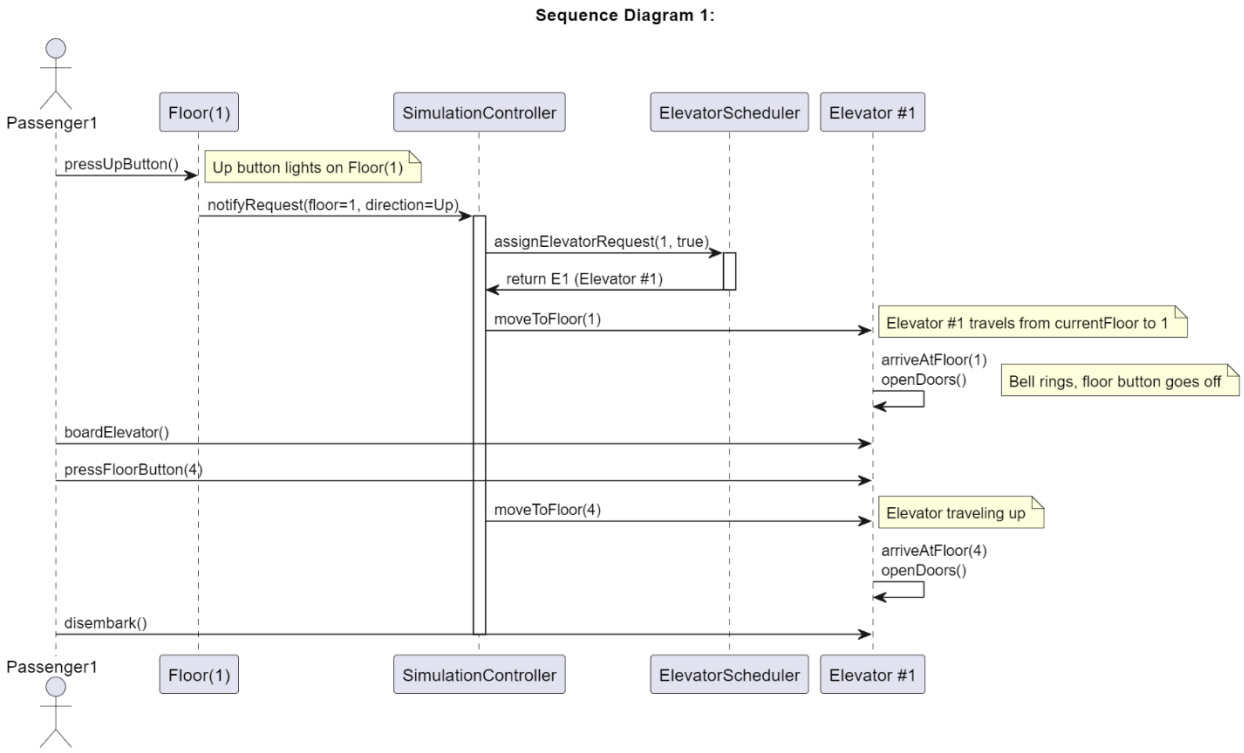
**Use case Diagram:**

## UML Class Diagram

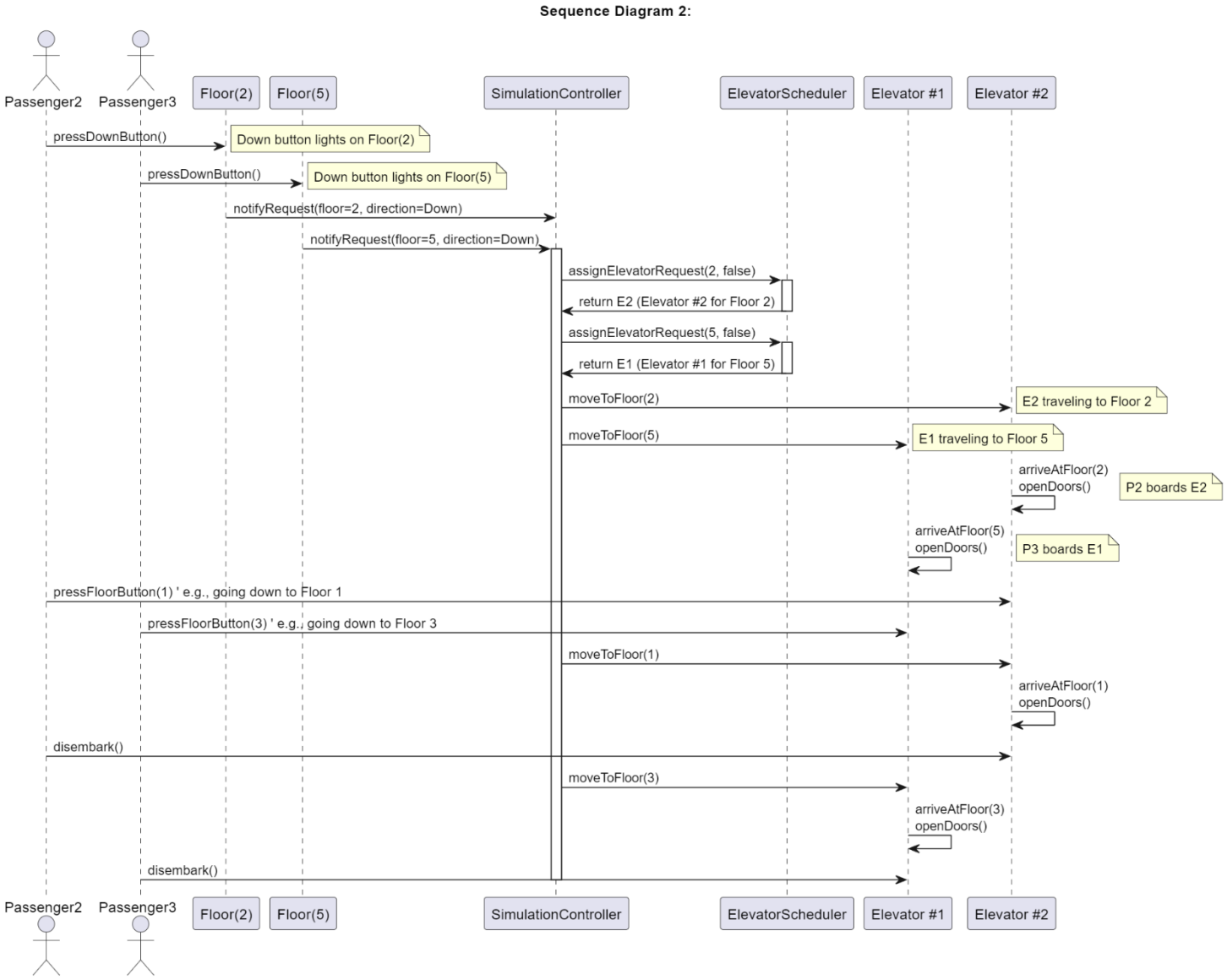


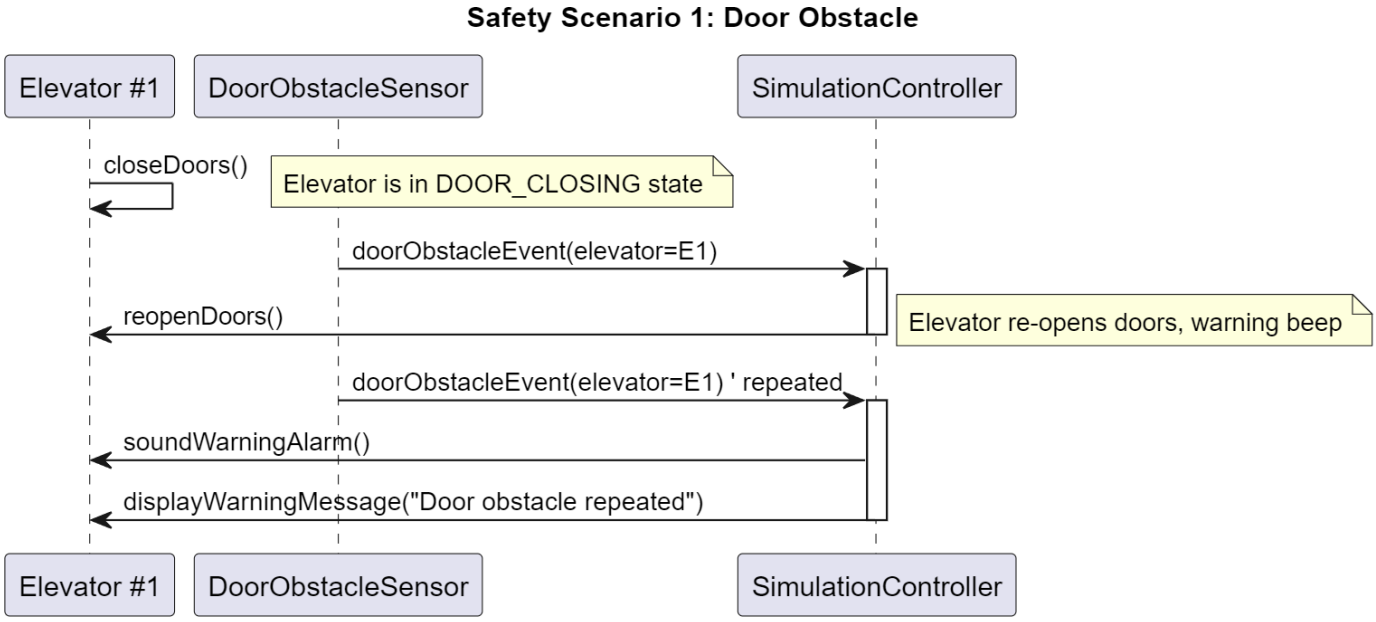
## UML Sequence diagrams

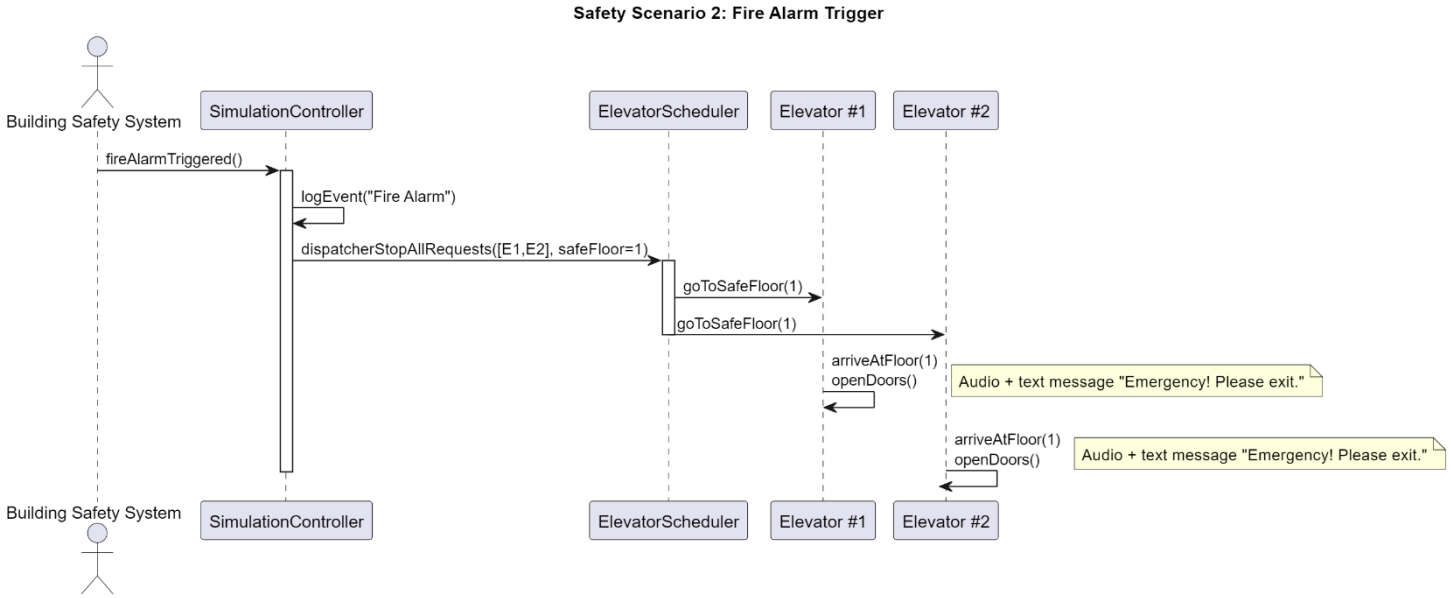
Single Passenger



## Two Passengers, Two Floors, Two Elevators

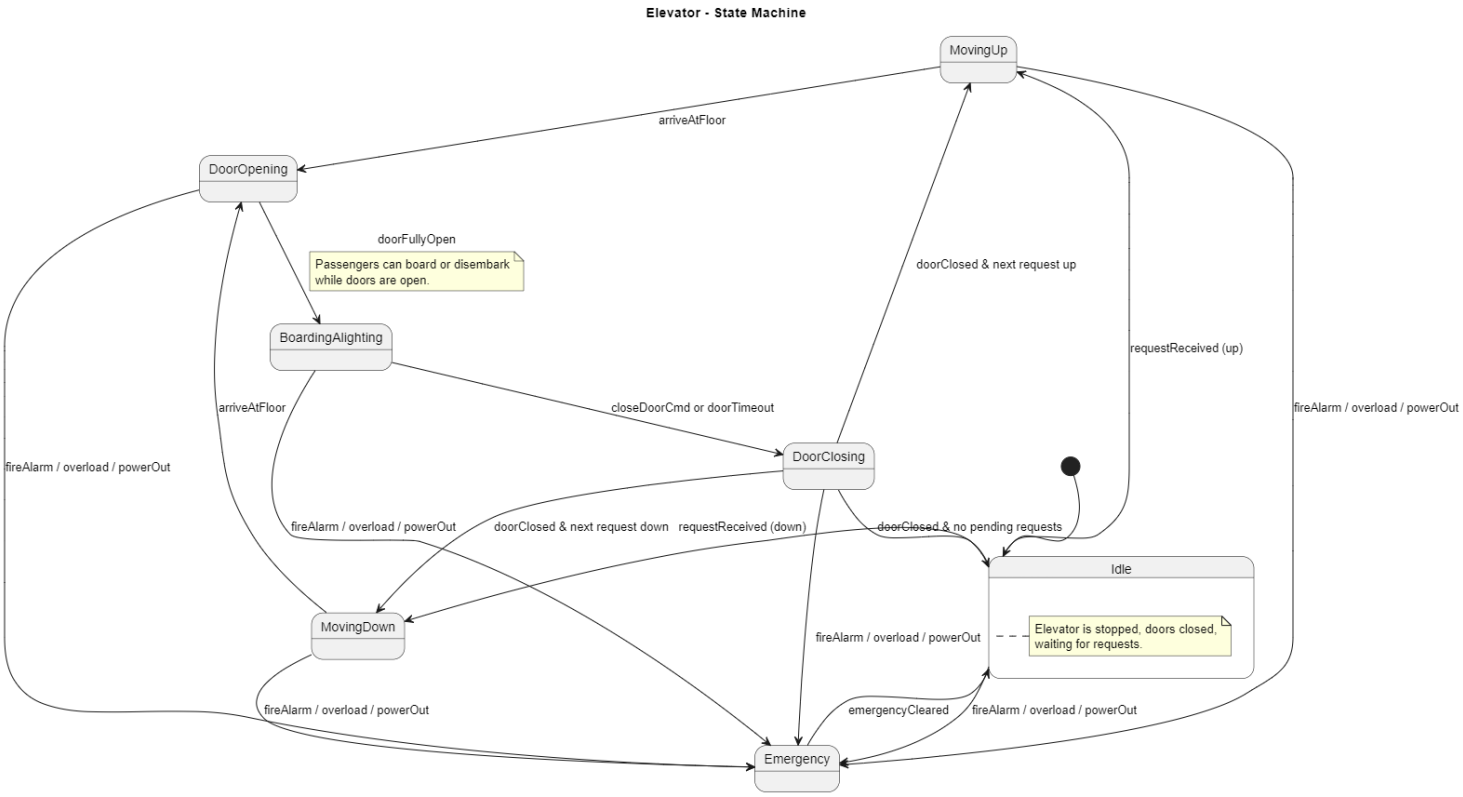



Safety Scenario 1: Door Obstacle
Elevator #1
DoorObstacleSensor
SimulationController
closeDoors()
Elevator is in DOOR_CLOSING state
doorObstacleEvent(elevator=E1)
reopenDoors()
Elevator re-opens doors, warning beep
doorObstacleEvent(elevator=E1) ' repeated
soundWarningAlarm()
displayWarningMessage("Door obstacle repeated")
Elevator #1
DoorObstacleSensor
SimulationController

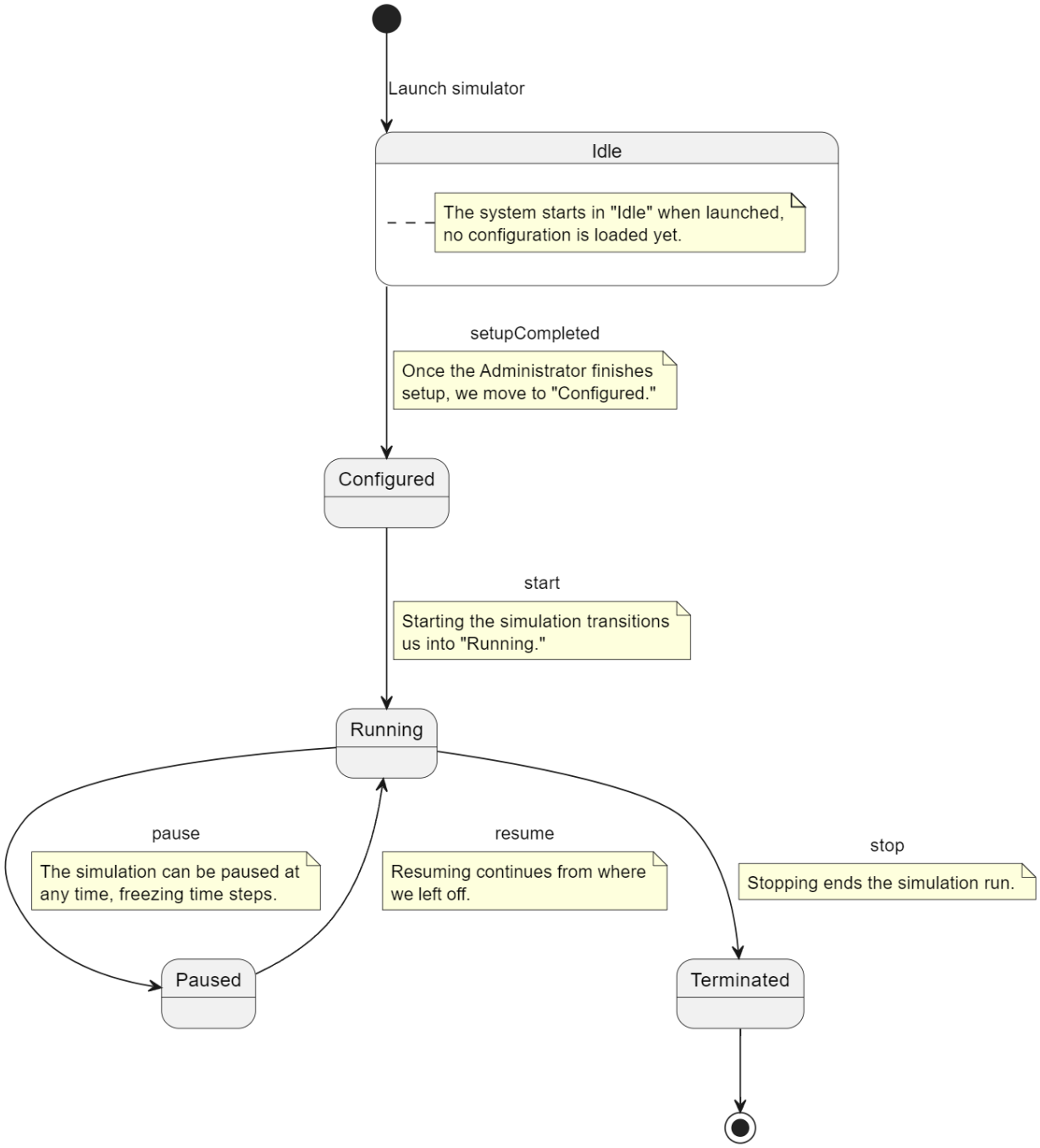

Safety Scenario 2: Fire Alarm Trigger
Building Safety System
SimulationController
ElevatorScheduler
Elevator #1
Elevator #2
fireAlarmTriggered()
logEvent("Fire Alarm")
dispatcherStopAllRequests([E1,E2], safeFloor=1)
goToSafeFloor(1)
goToSafeFloor(1)
arriveAtFloor(1)
openDoors()
Audio + text message "Emergency! Please exit."
arriveAtFloor(1)
openDoors()
Audio + text message "Emergency! Please exit."
Building Safety System
SimulationController
ElevatorScheduler
Elevator #1
Elevator #2

## UML state machine diagrams

See machine state elevator and machine state sim con for better quality image.



**Simulation Controller - State Machine**



**Requirements Traceability Matrix**

Below is an example of how to connect requirements (from the assignment specification) to use cases and to design elements (classes, methods, etc.).

PLEASE SEE EXCEL SHEET IF YOU ACTUALLY WANT TO LOOK AT IT. ITS MUCH EASIER TO LOOK AT I PROMISE

A3 Tracebility Matrix.xlsx

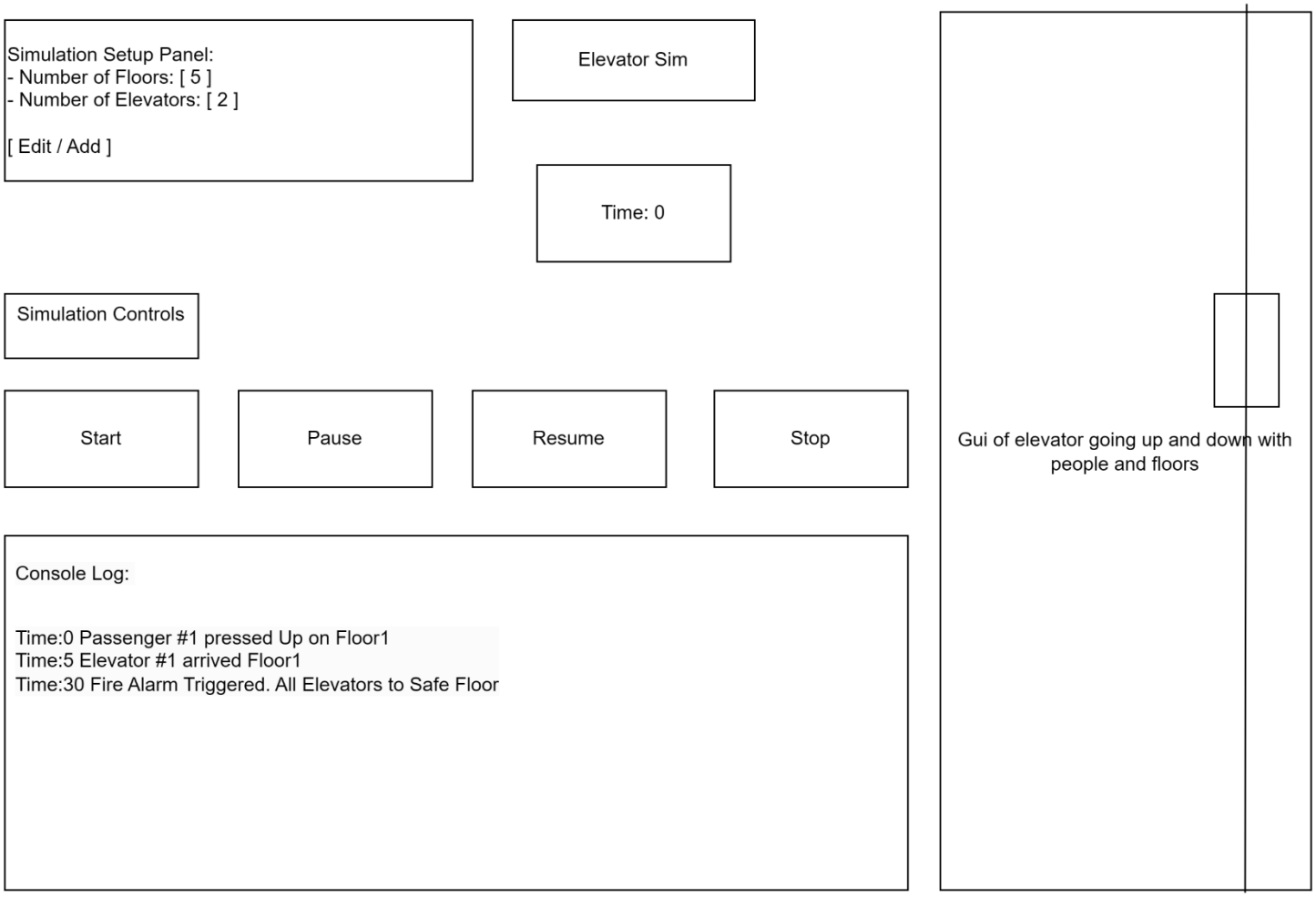
| Req. ID | Requirement | Use Case(s) | Design Element(s) | Implemented By | Tested By |
|---|---|---|---|---|---|
| **R 1** | Admin can configure N floors, M elevators before running | UC1: Setup Simulation | Simulation::setUpSimulation() SimulationController::initialize() | Simulation class SimulationController class | ElevatorSimWindow (manual UI test) **Unit**: SimTest.cpp |
| **R 2** | Start, pause, resume, stop simulation | UC2: Start/Pause/Resume.. | Simulation::start(), pause(), resume(), stop() | ElevatorSimWindow (buttons) Simulation methods | **Integration**: manual UI scenario test |
| **R3** | Log console messages for passenger actions, system responses | All UCs | Simulation::logEvent(...) GUI label/log | Simulation + ElevatorSimWindow | **Integration**: check console logs |

| | | | | | | |
|---|---|---|---|---|---|---|
| **R4** | Passengers request up/down from floors | UC3 | Floor::pressUpButton() ElevatorScheduler::assignElevatorRequest() | Floor, ElevatorScheduler | **Unit**: FloorTest.cpp | |
| **R5** | Passengers on board can select a destination floor (control panel) | UC4 | Elevator::pressFloorButton(...) | Elevator | **Unit**: ElevatorTest.cpp | |
| **R6** | Override door open/close with Open/Close Door buttons | UC5 | Elevator::openDoors(), closeDoors() | Elevator | **Unit**: ElevatorTest.cpp | |
| **R7** | Elevator help button triggers building safety; if no response => dial 911 | UC6 | Elevator::triggerHelp() SimulationController::handleSafetyEvent() | Elevator + SimulationController | **Integration**: scenario test with help button | |
| **R8** | Fire, Overload, Door Obstacle, Power Out are recognized and handled | UC7 | SimulationController::handleSafetyEvent(...) Elevator::triggerOverload() | SimulationController + Elevator | **Integration**: run safety scenarios | |

| | | | | | | |
|---|---|---|---|---|---|---|
| **R9** | Fire => all elevators move to safe floor with audio/text messages | UC7 (Fire) | ElevatorScheduler::dispatcherStopAllRequests(...)<br>Elevator::goToSafeFloor() | | SimulationController + ElevatorScheduler | **Integration**: scenario test (Fire event) |
| **R10** | Repeated door obstacles => warning audio/message | UC7 (Door obstacle) | Elevator door logic, sensor integration | | Elevator + SimulationController | **Integration**: scenario test (Door obstacle repeated) |
| **R11** | Power Out => backup power moves elevator(s) to safe floor, ask passengers to exit | UC7 (Power Out) | SimulationController::handleSafetyEvent(...)<br>Elevator::goToSafeFloor() | | SimulationController + Elevator | **Integration**: scenario test (Power Out) |
| **R12** | Simulation runs until events done & all elevators idle, or admin stops | UC2 | Simulation::advanceTimeStep()<br>SimulationController::checkElevatorStates() | | Simulation + SimulationController | **Integration**: run multiple time steps, see if stops automatically |

simulatio
n

**Sketch of GUI**



Summary of key design decisions and use of design patterns:

1. **Separation of Concerns**:
    - Simulation vs. SimulationController: The Simulation class deals with overall simulation state (time step, paused/stopped flags) while the SimulationController orchestrates the building's details, passengers' actions, and safety events.
2. **Elevator as a core class**:
    - Encapsulates the state machine for an elevator's movement, door operations, and safety handling like overloads or other emergency.
3. **Use of an ElevatorScheduler**:
    - Centralized scheduling logic to decide which elevator responds to a new floor request. This allows you to change the strategy like "Nearest Car," "Collective Control," "Traffic-based Priority," etc., without modifying the core Elevator logic.
4. **Observer or Publisher/Subscriber Pattern** (Potential):
    - Could be used to notify the GUI or log console whenever an event occurs (time step advanced, elevator moved, door obstacle triggered, etc.).
5. **State Pattern** (Potential):
    - The elevator state transitions (Idle, Moving, Boarding, etc.) can be implemented with the State design pattern, where each state is an object encapsulating its own behavior.
6. **Singleton Pattern**
    - The simulation could be run as a single instance, though it is not strictly necessary.
7. **Extension Points**:
    - The design allows for different safety events to be added without rewriting the entire system logic. The SimulationController::handleSafetyEvent() can handle new event types gracefully.

I tried to designed it so it keeps the system modular, with Building holding floors and elevators, SimulationController orchestrating the scenario, and ElevatorScheduler focusing on dispatch logic. This should make it so that changes like more floors, a new

scheduling policy, additional safety features can be introduced with minimal impact on the rest of the system.