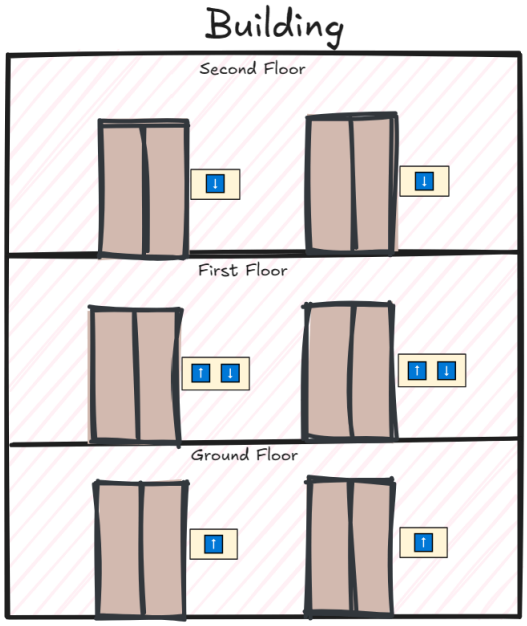
Design Elevator System

Resources

- 8. Elevator System, Low Level Design (Hindi) | SDE LLD interview question | Design Elevator System
- Code Repo → src/main/java/com/conceptcoding/interviewquestions/elevator · main · shrayansh jain / LLD-LowLevelDesign · GitLab

Problem Statement

Design a detailed low-level model for an Elevator System, including all relevant classes, interactions and design patterns covered earlier.



Elevator System in a Building

Overview

An Elevator System is used to transport people or goods across different floors in a building. We often see it in multi-storey buildings, malls, gated society apartments, and multi-floor parking lots. Elevators are used for efficient and easy vertical transport. Hence, choosing an optimised and efficient algorithm is essential. Here, we dive deeper into designing the Elevator System using all the design principles, patterns, guidelines, and best practices learned before. We will also discuss various algorithms that minimise user wait time, which serve the requirements in the most efficient and optimised way.

Working Flow

Here's how an Elevator works:

1. The user presses the `ExternalButton` - Up Arrow or Down Arrow.
2. Observes the movement of the `ElevatorCar` through the `Display` board at the user's `Floor`.
3. The `ElevatorCar` arrives at the Floor.
4. The `Door` opens.
5. The user gets in and presses the desired `Floor` number from the `InternalButton` panel inside the `ElevatorCar`.
6. The `ElevatorCar` moves in that direction to serve that request.

7. When the destination floor is reached, the `Doors` open and the user gets out.
8. The `ElevatorCar` listens for new requests from `ExternalButtons` and `InternalButtons`. And keeps moving in the direction to serve new requests.

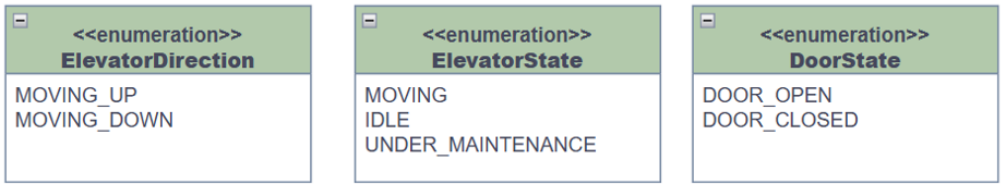
Requirement Classification

Design an Elevator System with the following requirements:

- Number of lifts/elevators = 2
- Maximum number of Floors = 10
- Lift dispatch algorithm → Odd/Even algorithm and SCAN-LOOK algorithm

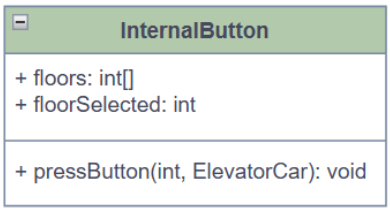
Components

1. Enums: `ElevatorState`, `ElevatorDirection` and `DoorState`



- `ElevatorDirection` denotes the direction in which the `ElevatorCar` is currently moving.
- `ElevatorState` represents the current status of the `ElevatorCar`. A change in its state will require appropriate actions to be performed.
- `DoorState` stores whether the `Door` is Open or Closed.

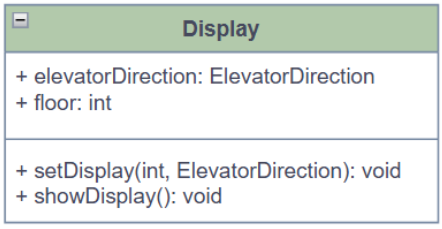
2. `InternalButton`



- `InternalButton` is the panel of buttons and display we see inside an `ElevatorCar`.

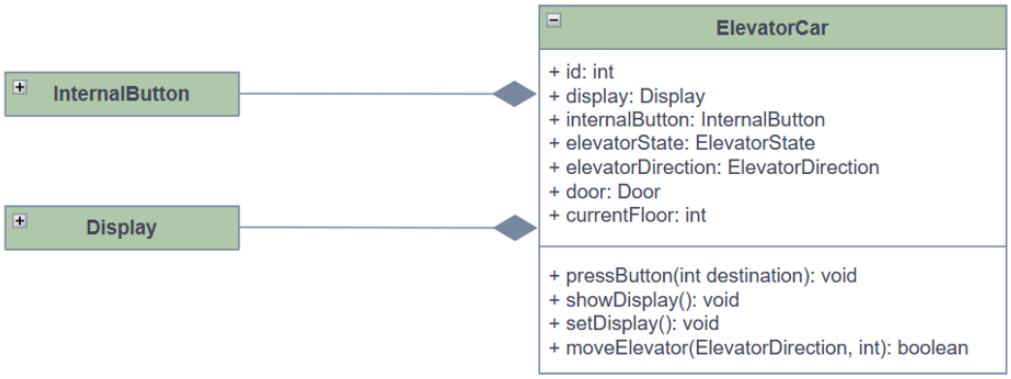
- Contains an array of floors in the building and allows the passengers to select the destination floor.
- The destination floor requested by the passenger is stored in the `floorSelected` variable.
- Upon pressing an internal button, the `InternalDispatcher` is called to handle the internal request.

3. `Display`



- A `Display` board is used inside the elevator and on every floor to show the status of the `ElevatorCar`.
- It displays the direction in which the `ElevatorCar` is currently moving, represented by an enum `ElevatorDirection`.
- Also displays the floor number the `ElevatorCar` is currently at.

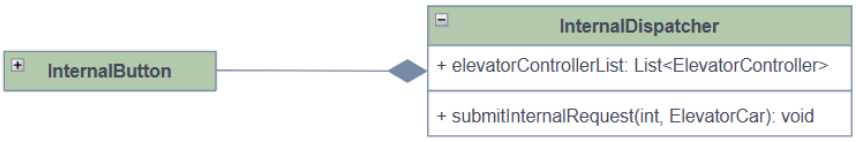
4. `ElevatorCar`



- The fundamental and essential part of the entire Elevator System is the `ElevatorCar`.
- `ElevatorCar` is composed of a list of Floor Buttons, i.e., `InternalButton` to choose from and a `Display` board to show status, and hence it holds a reference to both these objects and updates the display board on the go.

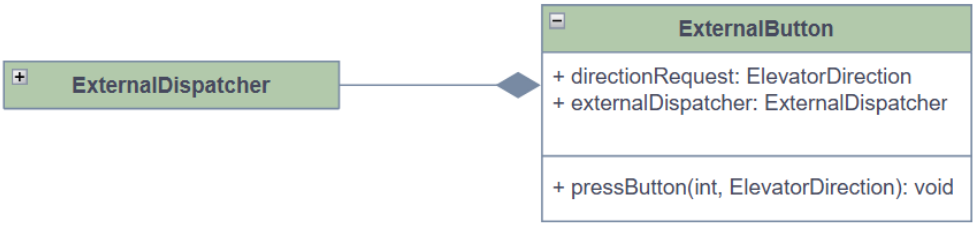
- Contains other state variables and update methods to ensure the appropriate functioning of `ElevatorCar`.

5. `InternalDispatcher`



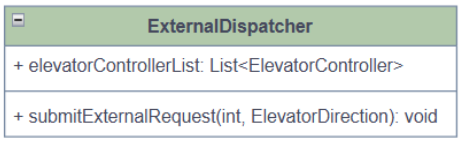
- `InternalDispatcher` receives the passenger's destination floor request and performs the necessary action of either ignoring it (if it's moving in the opposite direction), accepting and adding it to the queue.
- Responsible for the efficient and appropriate functioning of the `ElevatorCar` as per passenger input.

6. `ExternalButton`



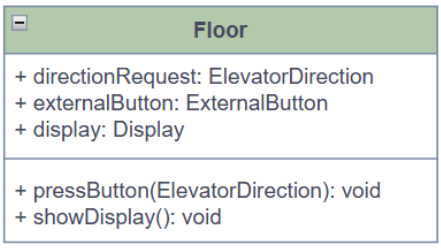
- `ExternalButton` is a button that is used to allow passengers to request for `ElevatorCar` at their floor.
- Holds a reference to the `ExternalDispatcher` that will be called to serve the nearest possible `ElevatorCar` with the least wait time.

7. `ExternalDispatcher`



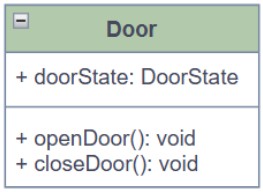
- `ExternalDispatcher` is called upon the button of an `ExternalButton`.
- Implements the `ElevatorCar` dispatch algorithm as per the requirement.

7. `Floor`



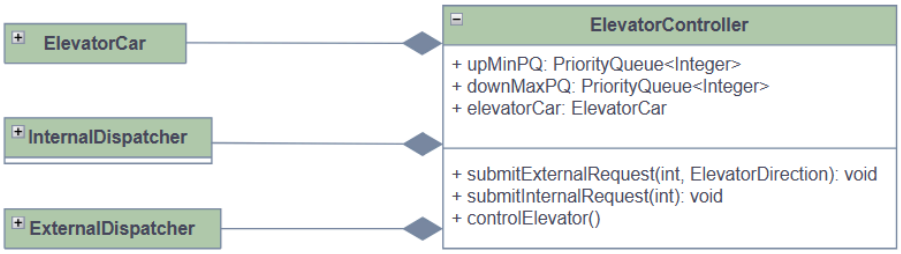
- `ElevatorDirection` holds the direction in which the `ElevatorCar` is moving.
- `ExternalButton` is used to receive the passenger's request.
- `ExternalDispatcher` is internally called to apply the chosen algorithm to dispatch `ElevatorCar` with the least wait time to the Passenger's Floor.
- A `Display` board is used on each `Floor` to show the status of the `ElevatorCar`, the floor it is currently at and in which direction it is moving.

9. `Door`



- `Door` holds the current status of the Door and has behaviours to carry out different door functions upon state change.

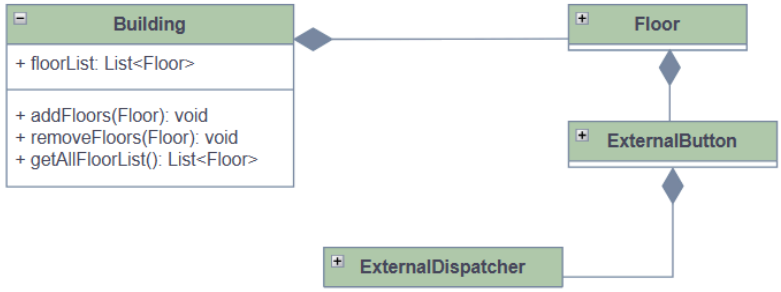
10. `ElevatorController`



- `ElevatorController` controls the working of an Elevator System in a building.
- It receives passengers' input from inside and outside the `ElevatorCar`.

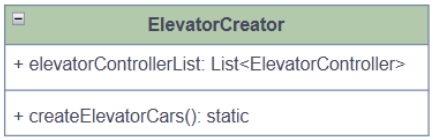
- Upon receiving an internal or external request, it invokes the appropriate dispatcher to respond to the passenger's request.
- Ensures smooth, efficient and optimised working for the Elevator System.

11. `Building`



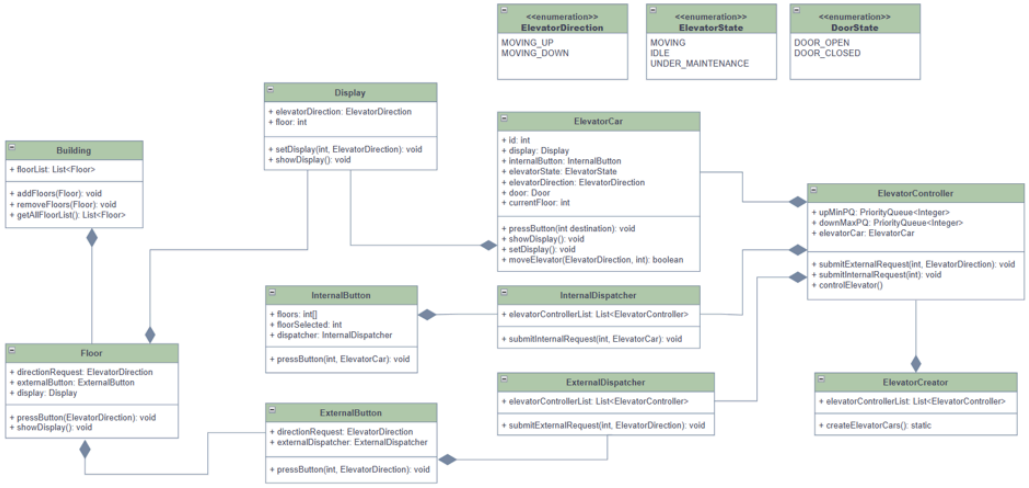
- A majority entity that has an Elevator System installed.
- A Building has `Floors`, and each passenger who wants to use the Elevator Service has access to use the `ExternalButton` to send a request to the floor they are currently at.

12. `ElevatorCreator`



- `ElevatorCreator` has a static method to instantiate all `ElevatorCars` in the Elevator System installed in a Building.

Class Diagram



Elevator System: Class Diagram

Algorithm

Here are a few algorithms used to dispatch the Elevator upon a passenger's request.

Even & Odd Algorithm

To keep the logic simple and maintain the focus on the Elevator System's design, this logic has been implemented in the code.

- If the passenger requests Odd-Numbered Floor, the Odd-Numbered `ElevatorCar` is responsible for serving that request.
- If the passenger requests an Even-Numbered floor, the Even-numbered `ElevatorCar` is responsible for serving that request.

SCAN Algorithm

- It is also known as the Elevator Algorithm because it moves like an elevator, up and down, serving requests.
- Start at the current floor position.
- Decide a direction based on user input.
- Move in that direction, serving all pending requests **until it reaches the end**.
- Once it reaches the end, reverse the direction and start serving requests in the opposite direction.
- Repeat this back-and-forth process until all requests are completed.

⛔ **Drawback:** May travel to the end even if there are no requests. Its bidirectional movement from end-to-end comes with additional cost(electricity & time).

LOOK Algorithm

- A slight improvement over the SCAN algorithm.
- Here, the `ElevatorCar` only goes as far as the last request in each direction (doesn't go all the way to its physical end).
- Start from the current floor position.
- Move in one direction, serving requests in order.
- Stops when there are no more requests in that direction (don't go to the end) and waits until it receives a new passenger request.
- Proceed in the direction of the new passenger request and serve the remaining requests.
- Continue until all are done.

> **Advantage:** Saves time compared to SCAN(no unnecessary travel), which is very fair, and an optimised approach. Saves electricity & time. "Looks" ahead to decide where to stop. Hence the name, LOOK algorithm.

Implementation

Refer Code Repository for executable code → src/main/java/com/conceptcoding/interviewquestions/elevator · main · shrayansh jain / LLD-LowLevelDesign · GitLab