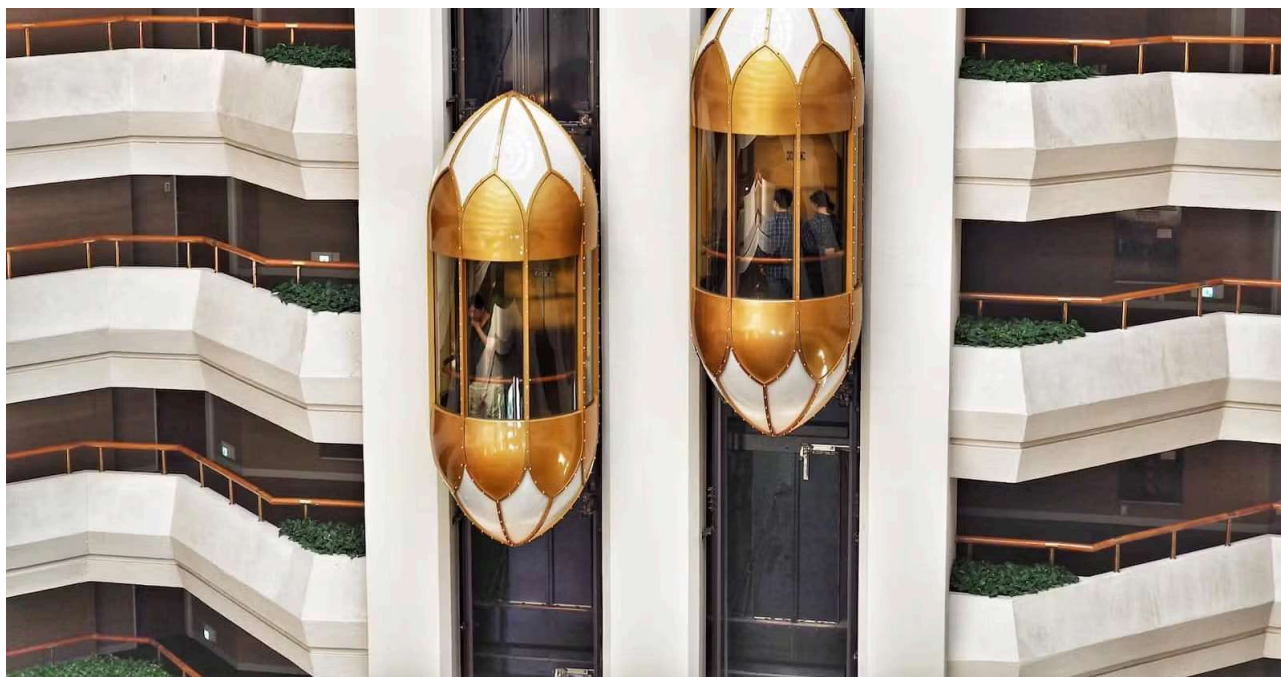


swati jha's Blog

Follow



Elevator System (Low-Level-Design)



swati jha

May 8, 2023 • 📖 7 min read



+2



Table of contents

Problem Definition

Requirements

Objects

UML Class Diagram

Code

Problem Definition

Elevators are an essential component of many buildings, especially those with multiple floors or high-rise structures, where stairs may not be a practical option. The objective of an elevator system is to provide an alternative means of vertical transportation that is convenient, accessible, and safe for all building occupants.

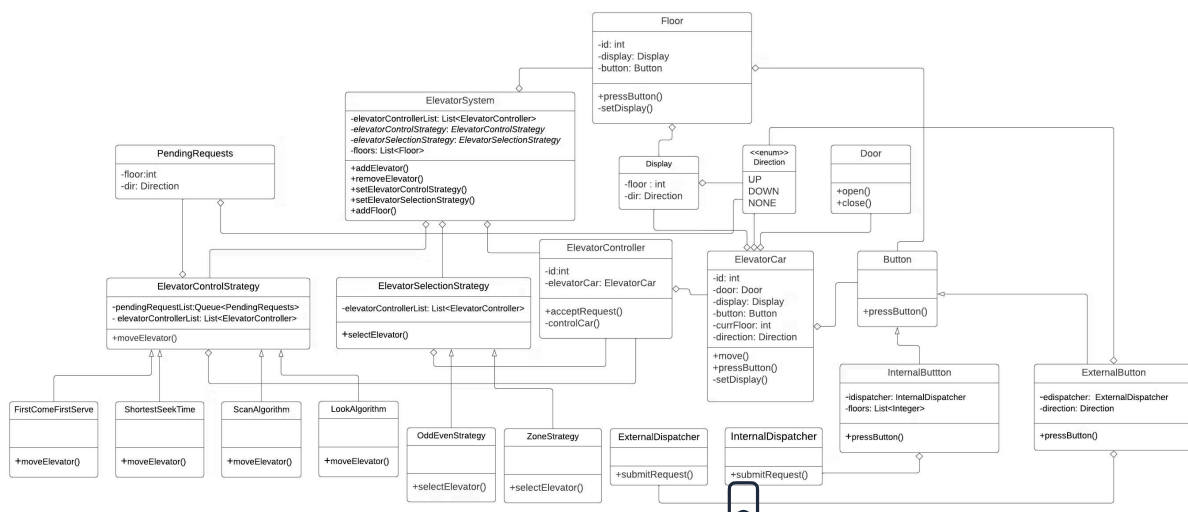
Requirements

1. There can be multiple floors
2. There can be more than one elevator
3. Elevators can be added or removed(for maintenance) as needed.
4. Two types of requests to be considered-
 - i. Person on the floor pressing the UP/DOWN button to call the elevator
 - ii. Person in the elevator pressing the floor number button to reach a destination
5. The algorithm for calling the elevator should be dynamic.
6. The functional algorithm for the elevator should also be dynamic

Objects

1. ElevatorSystem
2. ElevatorCar
3. ElevatorController
4. Floor
5. Button- InternalButton, ExternalButton
6. Direction- UP, DOWN, NONE
7. Display
8. Door
9. Dispatcher- InternalDispatcher, ExternalDispatcher
10. ElevatorSelectionStrategy- OddEvenStrategy, ZoneStrategy
11. ElevatorControlStrategy- FirstComeFirstServe, ShortestSeekTime, ScanAlgorithm, LookAlgorithm

UML Class Diagram



Code

Full implementation on Github

COPY

```
public class Button {  
    //for External Button  
    public void pressButton(int floor, Direction dir){}  
  
    //for Internal Button  
    public void pressButton(int floor, Direction dir, int elevatorId)  
}
```



COPY

```
public enum Direction {  
    UP, DOWN, NONE;  
}
```

COPY

```
@Getter  
@Setter  
public class Display {  
    private int floor;  
    private Direction dir;  
}
```

COPY

```
public class Door {  
    public void open(int id)
```



```

{
    System.out.println("Door opens for elevator "+ id);
}
public void close(int id)
{
    System.out.println("Door closes for elevator "+ id);
}
}

```

COPY

@Getter

```
public class ElevatorCar {
```

```
    private int id;
```

```
    private Door door;
```

```
    private Display display;
```

```
    private Button button;
```

@Setter

```
    private int currentFloor; //updated while elevator moves to ea
```

```
    private Direction dir; //updated every time elevator hanges dir
```

```
public ElevatorCar(int id)
```

```
{
```

```
    this.id= id;
```

```
    door= new Door();
```

```
    display= new Display();
```

```
    currentFloor= 0;
```

```
    dir= Direction.NONE;
```

```
    button= new InternalButton();
```

```
}
```



```
public void move(Direction dir, int floor)
```

```
{
    System.out.println("Elevator " + id + "moving " + dir);
    System.out.println("Elevator " + id + "stops at floor " + floor);
    door.open(id);
    door.close(id);

    //called everytime when currFloor value changes
    setDisplay();
}

public void pressButton(int floor)
{
    Direction dir= Direction.NONE;
    if(floor>currentFloor)
        dir= Direction.UP;
    else if(floor<currentFloor)
        dir= Direction.DOWN;
    button.pressButton(floor, dir, id);
}

private void setDisplay()
{
    display.setFloor(currentFloor);
    display.setDir(dir);
}
}
```



`@Getter`

```

public class ElevatorController {

    private int id;
    private ElevatorCar elevatorCar;

    public ElevatorController(int id)
    {
        this.id= id;
        elevatorCar= new ElevatorCar(id);
    }

    public void acceptRequest(int floor, Direction dir)
    {
        ElevatorSystem.elevatorControlStrategy.getPendingRequestList()
        .add(new ElevatorRequest(floor, dir));

        controlCar();
    }
    private void controlCar()
    {
        ElevatorSystem.elevatorControlStrategy.moveElevator(this);
        System.out.println("Elevator moving...");
    }
}

```



COPY

`@Getter`

```

public class ElevatorSystem {
    private List<ElevatorController> elevatorControllerList= new ArrayList<>();
}

```



```
public static ElevatorControlStrategy elevatorControlStrategy;
public static ElevatorSelectionStrategy elevatorSelectionStrate
public List<Floor> floors= new ArrayList<Floor>();

public static ElevatorSystem INSTANCE= new ElevatorSystem();

private ElevatorSystem()
{

}

public void addElevator(ElevatorController e)
{
    elevatorControllerList.add(e);
}
public void removeElevator(ElevatorController e)
{
    elevatorControllerList.remove(e);
}
public void setElevatorControlStrategy(ElevatorControlStrategy
{
    this.elevatorControlStrategy= elevatorControlStrategy;
}
public void setElevatorSelectionStrategy(ElevatorSelectionStrat
{
    this.elevatorSelectionStrategy= elevatorSelectionStrategy;
}
public void addFloor(Floor floor)
{
    floors.add(floor);
}

}
```



COPY

```
public class ExternalButton extends Button{
    private ExternalDispatcher edispatcher= ExternalDispatcher.INSTANCE
    @Getter
    private Direction direction;

    public void pressButton(int floor, Direction dir)
    {
        direction= dir;
        System.out.println("Pressed " + direction + " from floor "
            + floor);
        edispatcher.submitRequest(floor, dir);
    }
}
```



COPY

```
public class InternalButton extends Button{
    private InternalDispatcher idispatcher;
    @Getter
    private List<Integer> floors= new ArrayList<Integer>();

    public InternalButton()
    {
        idispatcher = new InternalDispatcher();
    }

    public void pressButton(int floor, Direction dir, int elevatorId)
    {
        floors.add(floor);
    }
}
```



```
        System.out.println("Pressed floor "+ floor + " from elevator "+ elevatorId);
        idispatcher.submitRequest(floor, dir, elevatorId);
    }
}
```

COPY

@Getter

@Setter

```
public class Floor {
    private int id;
    private Display display;
    private Button button;

    public Floor(int id)
    {
        this.id= id;
        button= new ExternalButton();
    }

    public void pressButton(Direction dir)
    {
        button.pressButton(id, dir);
    }

    //called everytime selected elevator moves each floor
    private void setDisplay(int floor, Direction dir)
    {
        display.setDir(dir);
        display.setFloor(floor);
    }
}
```



}

COPY

```

public class InternalDispatcher {

    public void submitRequest(int floor, Direction dir, int elevatorId)
    {
        for(ElevatorController eController: ElevatorSystem.INSTANCE.getElevatorControllers())
        {
            if(eController.getId() == elevatorId)
            {
                eController.acceptRequest(floor, dir);
            }
        }
    }
}

```



COPY

```

public class ExternalDispatcher {

    public static ExternalDispatcher INSTANCE = new ExternalDispatcher();
    private ExternalDispatcher()
    {

    }

    public void submitRequest(int floor, Direction dir)
    {
        int elevatorId = ElevatorSystem.getElevatorSelectionStrategy().selectElevator(floor, dir);
        System.out.println("Selected elevator " + elevatorId);
    }
}

```

```

for(ElevatorController eController: ElevatorSystem.INSTANCE
{
    if(eController.getId()== elevatorId)
    {
        eController.acceptRequest(floor, dir);
    }
}
}
}

```

COPY

```

public class ElevatorSelectionStrategy {
    protected List<ElevatorController> elevatorControllerList = Ele

    public int selectElevator(int floor, Direction dir) {
        return 0;
    }
}

```



COPY

```

public class OddEvenStrategy extends ElevatorSelectionStrategy {

    @Override
    public int selectElevator(int floor, Direction dir) {
        for(ElevatorController eController: elevatorControllerList)
        {
            //old elevator for odd floors and even elevators for ev
            //select elevator which is moving in same direction whi

```

```

//          if(floor%2 == eController.getId()%2)
//          {
//              int currFloor= eController.getElevatorCar().getCu
//              Direction currDir= eController.getElevatorCar().g
//              if(floor>currFloor && currDir==Direction.UP)
//                  return eController.getId();
//              else if(floor<currFloor && currDir==Direction.DOWN)
//                  return eController.getId();
//              else if(currDir==Direction.NONE)
//                  return eController.getId();
//          }
//      }
//      return ThreadLocalRandom.current().nextInt(1, elevatorContr
}
}

```

COPY

```

public class ZoneStrategy extends ElevatorSelectionStrategy {
    @Override
    public int selectElevator(int floor, Direction dir) {
        for(ElevatorController eController: elevatorControllerList)
        {
            //assign elevators according to zones in building
            //out of these elevators select the elevator which is g
        }
        return ThreadLocalRandom.current().nextInt(1, elevatorContr
    }
}

```



COPY

@Getter

```
public class PendingRequests {  
    private int floor;  
    private Direction dir;  
  
    public PendingRequests(int floor, Direction dir)  
    {  
        this.floor= floor;  
        this.dir= dir;  
    }  
}
```

COPY

@Getter

```
public class ElevatorControlStrategy {  
    //queue storing pending requests in form of  
    @Setter private Queue<PendingRequests> pendingRequestList= new  
    private List<ElevatorController> elevatorControllerList = Eleva  
  
    public void moveElevator(ElevatorController elevatorController)  
    {  
  
    }  
}
```

COPY

```
public class FirstComeFirstServe extends ElevatorControlStrategy {
```



```

public void moveElevator(ElevatorController elevatorController)
{
    //poll each requests out of queue one by one
    //move elevator according to each request
    //Disadvantage: frequent change of direction of elevator, h
    // long waiting time for users

}
}

```

COPY

```

public class ShortestSeekTime extends ElevatorControlStrategy {
    public void moveElevator(ElevatorController elevatorController)
    {
        //implemented using min heap which is sorted according to
        //min distance of requested floor from the current floor of

        //this min heap is updated everytime a new request is added
        // when elevator moves to another floor

        //Disadvantage: starvation of distant floor when maximum re

    }
}

```



COPY

```

public class ScanAlgorithm extends ElevatorControlStrategy {
    public void moveElevator(ElevatorController elevatorController)
    {

```

```
//      In this algorithm, elevator starts from one end of the di
//      towards the other end, servicing requests in between one
//      Then the direction of the elevator is reversed and the pr

//      Implemented using two array
//      All floors with UP requests are marked in the UP array
//      and all floors with DOWN request are marked in the DOWN a
//      and the elevator scans UP array while moving up and DOWN
//      and it stops at the requested floors

//      Advantage:
//      1. not frequent change of floor for every request
//      2. no starvation

//      Disadvantage: If there are 100 floors, and last requestec
//      is 15, then also the elevator will move till the 100th fl
//      This is improved by LOOK Algorithm

    }
}
```

COPY

```
public class LookAlgorithm extends ElevatorControlStrategy {
    public void moveElevator(ElevatorController elevatorController)
    {
//      In this algorithm, the elevator moves in a specific direc
//      but instead of going all the way to the end of the buildi
//      like the SCAN algorithm, it reverses direction as soon as
//      request in the current direction.

//      Implemented using a min heap, a max heap and a queue
//      Min heap: all requests that can be served in UP direction
```



```
//      (eg. Requested floor > currFloor, requested UP, elevator n
//      Max heap: all requests that can be served in DOWN directi
//      (eg. Requested floor < currFloor, requested DOWN, elevator
//      Queue: all requests that cannot be served in current dire
//      eg. elevator moving up and currFloor is 3, now someone at


//      While moving up, all requests from min heap will be taken
//      when min heap is empty, elevator reverses direction and a
//      will be put in min heap.

//      While moving down, all requests from max heap will be tak
//      when max heap is empty, elevator reverses direction and a
//      will be put in max heap.

//      Advantage:
//      1. not frequent change of floor for every request
//      2. no starvation of requests
//      3. efficient in terms of usage because it moves only the

//      Disadvantage: it does not prioritize requests based on th

    }
}
```

In this blog, we have explored the importance of Elevator System Low-Level Design and discussed some of the key requirements that need to be considered in the design process. By addressing the technical aspects of the elevator system, the low-level design helps to ensure that the system operates smoothly, is user-friendly, and meets the needs of the building's occupants. 

Subscribe to my newsletter

Read articles from directly inside your inbox. Subscribe to the newsletter, and don't miss out.

SUBSCRIBE

#elevator

low level design

low level programming

high level to low level

elevator-system

Written by

**swati jha**

As a software developer, I find immense joy in crafting elegant code and bringing ideas to life through programming. And when I'm not busy coding, you'll often find me creating content that showcases my passion for this fascinating world of software development.

[Follow](#)

ARTICLE SERIES

Low Level Design Problems**1****Vehicle Rental System(Low-Level-Design)**

Problem Definition The Vehicle Rental System serves as a bridge between customers and rental agencies...

2

Elevator System (Low-Level-Design)

Problem Definition Elevators are an essential component of many buildings, especially those with mul...

3



Snakes and Ladders (Low-Level Design)

Problem Definition Snakes and Ladders is a popular board game enjoyed by people of all ages around t...

4

Parking Lot (Low-Level Design)

Problem definition A parking lot is a designated area for parking vehicles and is a feature found in...

©2024 swati jha's Blog

[Archive](#) • [Privacy policy](#) • [Terms](#)



Powered by Hashnode - Build your developer hub.

Start your blog

Create docs

