**Elevator Control System**

**Requirement**

Design and implement a basic elevator control system for a N-story (N>1) building with only 1 elevator, and a simulator that can accept requests from passengers and change states (idle, up, down, open, close) based on decisions made by the control system.

Note: There could be multiple users trying to use the elevator at any given time, e.g. Elevator is in transit to floor X with passenger A when passenger B pushes the button on floor Y.  Please make sure such cases are covered by your implementation.

**Design**

**Algorithm**

The **elevator algorithm** (also **SCAN**) is named after the behavior of a building elevator where the elevator continues to travel in its current direction (up or down) until empty, stopping only to let individuals off or to pick up new individuals heading in the same direction. As the name states the same algorithm is used to design this elevator system.

To implement the algorithm; elevator maintains two sorted queues upQueue (sorted ascending) and downQueue (sorted descending) . Elevator Controller let elevator consume the queue by moving in one direction ( up or down ) until all requests are served . Once it reaches the highest floor in upQueue or lowest floor in downQueue, it processes the other queue or goes in idle state.

**Design considerations**

**Scalability**

The design can be scaled up to work with multiple elevators. The request handler required to be extended to work with multiple elevators though with ability to decide which elevator is most suitable to serve the request.

**Extensibility**

Loosely coupled design makes it easier to add / deprecate existing component i.e. Elevators / Control System to replace with the new. To avoid direct dependencies on implementation; the interface is exposed to controllers, elevators.

Ability to connect publishers to Elevator and floor controls; one possible use is for display.

**Concurrency**

The system uses thread safe data structure, ConcurrentLinkedQueue; to queue incoming requests ensure capability to handle concurrency.

**Design patterns**

Observer pattern using native JDK features is used to notify button pressed events from elevators and floor controls.

**Known limitations**

It’s defined but the Elevator doesn’t support MAINTAINANCE mode as of now.

The maximum floor supported by system are **2,147,483,647** i.e. java Integer limit.

The elevator doesn’t support real life states like MAINTAINANCE, and ALARM.

The current system doesn’t take elevator capacity in consideration.

Unlike real life Floor Control doesn’t limit UP and DOWN button on top and ground floor respectively

System needs updated concurrency changes to handle multiple elevators.

**Possible improvements in design, implementation and testing**

Spring for dependency injection of observer and looser decoupling.

System initialization using Spring or properties file.

Create granular system properties besides floors

Create multiple controller strategies for saving power VS reduced wait time latency.

Enable Ability to modify the submitted request if not processed yet to support CRUD operations on request.

Improved unit test coverage.

Better testing using mocking libraries i.e. JMOCK and others that supports observer testing

Test cases for concurrency management can be added.

**Usage**:

**Prerequisite:**

Java 1.5 or above, Maven 3.1 or above, OS that supports java 1.5 or above

**How to build and run**

1. Download the project form the github. The eclipse Project is at <https://github.com/sunilosunil/elevator-system/tree/master/elevator>

Github URL: <https://github.com/sunilosunil/elevator-system.git>

1. Go to the home directory elevator of the project (one with pom.xml) and built using command “mvn –clean –install”
2. The simulation can be run using ElevatorApplication.java file’s main method or using Runnable jar file. Use “mvn clean compile assembly:single “

Execute jar file using command

java –jar elevator-0.0.1-jar-with-dependencies.jar

Follow console instructions when running the simulation.

The output of the simulator prints the state of the elevator on every floor in the following format.

e.g.

Elevator [currentFloor=3, upQueue=[4, 7, 8], downQueue=[4], state=UP, Door [state=CLOSE]