README

Use `Gradle` to build the project. For more information, see `build.gradle`.

Then run the main class `VmwareCodingQuizApplication` as the simple `springboot` project.

Dependencies

- gradle 7.4.1 use gradle-wrapper
- org.springframework.boot 2.6.6
  - spring-boot-starter-web
  - spring-boot-starter-logging
- org.projectlombok:lombok:1.18.22

Algorithm

- Based on Disk Scheduling Elevator Algorithm.
- When passenger request arrive, only one elevator will respond to passenger requests.
- The elevator will keep moving util there are no more request on the direction. Then the elevator will turn around to the next task floor.
- When the elevator stop at a floor, all the waiting passengers on this floor with the same direction with the elevator will get into the elevator until the elevator is full loaded.

```
  Elevator:1  state: UP stop on 5 floor 0/20

  Waiting Passengers on 5 floor:
     user1 5-10 (UP)     direction same with the elevator
     user2 5-6  (UP)     direction same with the elevator
     user3 5-2  (DOWN)   direction opposite to the elevator

  user1 and user2 will get in the elevator regardless of whether the
elevator was originally allocated
```

- `Task` based on the passenger's startFloor and endFloor

```
  user1 5-10 (UP)    startFloor is 5 endFloor is 10
  user2 5-2  (DOWN)  startFloor is 5 endFloor is 2

  When the elevator need to pick user1,the task is (5,UP)
  When the user1 get in the elevator,the task is (10,UP)

  When the elevator need to pick user2,the task is (5,DOWN)
  When the user2 get in the elevator,the task is (2,DOWN)

  The direction of the task is determined by the direction of the passenger
```

- Task queue sort algorithm
    - the elevator will move according to the task queue
    - tasks in the queue are unique, same floor with same direction will be treated as one task
    - when a new task is added, sort the task queue
    - the algorithm is divided into four parts
    - take the elevator UP as an example

```
Elevator:1  state: UP stop on 5 floor (curFloor)

Current Task:
  (5,UP) (7,UP) (5,DOWN) (4,UP) (2,UP) (11,UP) (4,DOWN) (6,UP)

First,sort tasks in ascending order
  (2,UP) (4,UP) (4,DOWN) (5,UP) (5,DOWN) (6,UP) (7,UP) (11,UP)

Second,choose the task floor>= curFloor with the same direction
  (5,UP) (6,UP) (7,UP) (11,UP)

Third,choose the task with the opposite direction in descending order
  (5,DOWN) (4,DOWN)

Fourth,choose the task floor<curFloor with the same direction
  (2,UP) (4,UP)

Finally,the sorted tasks is
  (5,UP) (6,UP) (7,UP) (11,UP) (5,DOWN) (4,DOWN) (2,UP) (4,UP)
```

- Elevator Allocation Strategy
    - add new passenger task
    - sort task queue for each elevator
    - count the distances to reach the passenger's startFloor

    ```
    distances[0] the num of up floors
    distances[1] the num of down floors
    distances[2] the num of stop floors
    ```

    - calculate elevator cost

    ```
    timecost = distances[0] + distances[1] + distances[2]
    powercost= distances[0] * upCost + distances[1] * downCost

    cost = timecost * ratio + powercost * (1-ratio)
    ```

    - choose the elevator with the lowest cost