多线程电梯正确性论证

Elevator – Multi (Project 5)

# 抽象对象得到了有效实现论证

## FloorQueue

overview: FloorQueue is a queue that records all the floor requests created into the system.

## ElevatorQueue

overview: ElevatorQueue is a queue that records all the elevator requests created into the system.

## InputHandler

overview: InputHandler basically is just a class to handle inputs from user. It will validate the input and see if it matches the regular expression, if yes then it will parse that request into a request form, the FR request into FloorRequest, and the ER request into ElevatorRequest.

## Floor

overview: The Floor class is just like any floor that could be found in everyday life. It's got up and down buttons and also a place to display it's ID (which floor is it). The floor will be initialized (created) on the system init, it will automatically create floors from 1 - max along with the buttons first set up to false (not turned on). When there's a new request, the request will first call the floor and set the button (up/down) of that floor to true, meaning that it's turned on. That button will then send a request to the scheduler telling it that there's a new request waiting to be processed.

## Scheduler

overview: Scheduler is the base scheduler. This class only initialize things like creating the floors along with their buttons and elevators along with their buttons too. Scheduler also initialize the floor queue that will be taken by the ALS Scheduler for later use.

## ALS

overview: ALS Scheduler is the one that actually runs the system. ALS (A Little Smart) will determine whether a request could be processed now or not, which elevator will take it, is it a pickup request. The request itself is taken from the floor request queue initialized in the Scheduler class. To see which elevator could take the request, it will see which elevator is available to process that request as a pickup as this is first priority. If many can do that, then it will choose the one with less movement points. If none is available then it will see which elevator is IDLE. If none is idle then the request will have to wait until at least one is available. If the request is taken to be processed then the floor request queue of the Scheduler will have to remove that request from the queue, marking it as done/processed.

## Elevator

overview: Elevator is a moving machine that is used to process passenger request in elevator system. There will be two types of request that will be processed by an elevator, the floor request, that is a request made on certain floor, and elevator request, the request made in the elevator itself. Therefore, each elevator will have its own queues to record the requests that it needs to process, preventing to process other elevator's request or have its request processed by other elevator. When a request is done being processed, that request will be removed from the queue, meaning that it's done being processed and the system will call the OutputHandler to print out the record. The elevator will know which request should be processed first, considering there exist pickup requests that could be made after a main request is made and processed.

## OutputHandler

overview: OutputHandler is a class that manages all things related to printing out the output result to a file. When a request is done being processed, the system will call the OutputHandler to print the result, this class will print out the result to a file called result.txt located in the root folder.

# 构造方法实现正确性论证

public Floor(int floornum) {

/\*\*

\* @REQUIRES:floornum;

\* @MODIFIES:this;

\* @EFFECTS:

\* (\result = \this) && (\this.isEmpty());

\* (\this.floornum == floornum);

\*/

}

Floor的构造方法只被调用一次， 是在程序刚被执行的时候。目的是如同现实当中一般，构造楼层（由此题目可知有20个楼层）。由规格可知，此方法引用floornum，是为了记录楼层的层数（第一层，第二层。。。）。

public ALS(Scheduler sched) {

/\*\*

\* @REQUIRES:sched;

\* @MODIFIES:this;

\* @EFFECTS:

\* (\result = \this) && (\this.isEmpty());

\* (\this.sched == sched);

\*/

}

此方法是用来构造ALS类，引用Scheduler （sched）为了保持各个类用的Scheduler。ALS被构造后，就可以那同一个电梯List和楼层List。

public Elevator(int elevnum) {

/\*\*

\* @REQUIRES:elevnum;

\* @MODIFIES:this;

\* @EFFECTS:

\* (\result = \this) && (\this.isEmpty());

\* (\this.elevnum == elevnum);

\*/

}

此构造方法是用来构造电梯（由此题目可知有三台电梯需要被构造），如同楼层构造方法，电梯的构造方法也只是被调用一次（在程序刚被执行的时候）。此方法引用elevnum用来记录电梯数（第一台，第二台，。。。）

public ElevButton(int buttonnum) {

/\*\*

\* @REQUIRES:buttonnum;

\* @MODIFIES:this;

\* @EFFECTS:

\* (\result = \this) && (\this.isEmpty());

\* (\this.buttonnum == buttonnum);

\*/

}

此构造方法是用来构造电梯里的楼层按钮（由此题目可知每台电梯有20个楼层按钮需要被构造），如同楼层构造方法，楼层按钮的构造方法也只是被调用一次（在程序刚被执行的时候）。此方法引用buttonnum用来记录楼层按钮数（第一层，第二层，。。。）

# 方法实现正确性论证

## 请求队列

首先，请求队列分为两类：

* FloorQueue是楼层请求 🡺 里边的请求是楼层发出的请求队列
* ElevatorQueue是电梯请求 🡺 里边是从某个电梯里发出的请求队列

两个请求队列类都有同样的方法与实现：

* AddQueue

public synchronized boolean addQueue (Request req) {

/\*\*

\* @REQUIRES: req (Request);

\* @MODIFIES: this.queue;

\* @EFFECTS:

\* (\result==false)==>(\old(\this).queue.contains(req)==true)

\* (\this.queue.size == \old(\this).queue.size+1) && (\this.queue.contains(req)==true)&&(\result==true);

\*/

}

由规格可看出此方法的具体实现。首先需要引用要被加入的请求（若是楼层请求队列那便是FloorRequest，若是电梯请求类那便是ElevatorRequest）。此方法对对应的请求队列进行改变（电梯或楼层）。被执行后，若不是无效请求或同性请求，那对应的请求队列中会有这个被引用的请求，请求队列的size也自然加1。

* RemoveQueue

public synchronized boolean removeQueue (int i) throws CannotRemoveException {

/\*\*

\* @REQUIRES: i (int);

\* @MODIFIES: this.floorqueue;

\* @EFFECTS:

\* (this.floorqueue.size == \old(\this.floorqueue).size-1);

\* (\forall int e; e != i; this.floorqueue.contains(e) == \old(this.floorqueue).contains(e));

\* exceptional\_behaviour(CannotRemoveException("Remove index is out of array bounds!"));

\*/

}

由规格可看出此方法的具体实现。首先需要引用要被删除的请求index（i）。此方法对对应的请求队列进行改变（电梯或楼层）。被执行后，队列中有此index，那对应的请求队列中，具有对应index的请求会被删掉，请求队列中不会再有这个请求，请求队列的size也自然减1。

public synchronized boolean contains(int tarFloor, String dir) {

/\*\*

\* @REQUIRES: tarFloor, dir;

\* @MODIFIES: None;

\* @EFFECTS:

\* (\forall int i; this.floorqueue.contains(tarFloor, dir) ==> (\result == true);

\* (\result == false);

\*/

}

此方法是检查对应的请求队列中有没有被找的请求。具体实现是先引用请求中的楼层目（若是在楼层请求队列中寻找，那再加一个Direction的引用）。用loop逐个找，请求队列中的req(i)是否符合被找的那个楼层目的（和方向），若是，返回true。

## 楼层上/下按钮

我们知道真正的电梯系统中，每个楼层里会有上/下行的按钮，代表此楼层要求电梯去哪儿。实际上此按钮并不会限制乘客到电梯里时可选的楼层请求，只会影响电梯方向和捎带情况。比如说电梯处于IDLE情况，然后第二层的上行按钮亮了，电梯不论是在第二层的上面或下面都会去第二层载乘客。乘客到里边时，虽然刚才被按的是上行按钮，但第一层的按钮也是可以按的。若没其他请求，那电梯也可以去处理这个到第一层的请求。除非有捎带，那就先处理一个方向的请求，比如从第二层到第10层，所有会导致电梯上行的请求会被处理而需要电梯下行的请求需要先等没有其他上行请求可处理时才处理。

这个按按钮的方法实现，不论上行按钮还是下行按钮其实是一样的：

public synchronized void setUpStatus(boolean upbutton, Scheduler sched) {

/\*\*

\* @REQUIRES:upbutton, sched;

\* @MODIFIES:this.upbutton;

\* @EFFECTS:

\* (\this.upbutton == true) ==> sched.addFloorReq(new FloorRequest(\this.floornum, \this.uptime, UP);

\*/

}

例子上写的是按上行按钮的方法实现（都一样）。此方法会引用按钮的boolean情况（true代表按钮被按下）和Scheduler sched（只是为了保持一样的Scheduler被用。此方法会对对应的按钮进行改变（被按的话，按钮会亮，不然就是暗）。若是变暗，那是代表请求已被处理，不需要做任何事了。而若是变亮，代表有新的楼层请求，自然要构造新的楼层请求放到sched的楼层请求队列中。

## 输出处理

输出处理有两种，一个是输出楼层请求处理过程和电梯请求处理过程。每种均需要引用请求信息和当前电梯情况信息。然后被输出到被指定的文件中。

public void writeFloor(int target, String dir, double time, int elevid, int floor, String elevdir, int point) {

/\*\*

\* @REQUIRES: this.filename != NULL;

\* @MODIFIES: this.filename;

\* @EFFECTS: \this.filename.contains(content);

\*/

}

## 输入验证

public void validateInput(double time) {

/\*\*

\* @REQUIRES: time (double);

\* @MODIFIES: m, input;

\* @EFFECTS:

\* input.contains("(") == false && input.contains(")") == false;

\* (m.matches(input) == false ==> exceptional\_behavior(Exception("Invalid Format!")

\* (m.matches(input) == true ==> input.contains(FR) == true ==> parseFloor(time);

\* (m.matches(input) == true ==> input.contains(ER) == true ==> parseElevator(time);

\*/

}

此方法引用的time是为了传时间数据，保持在处理请求过程中时间的一样行。然后会查看请求中是有FR还是ER。FR代表楼层请求，ER代表电梯请求。这两种不一样的请求具有不一样的regex。表明清楚请求类型后会验证请求格式。若请求已符合对应的真值表达式，然后进行对应的请求解析（parseElevator或parseFloor）。

public void parseFloor(double time) throws Exception {

/\*\*

\* @REQUIRES: time (double);

\* @MODIFIES: req\_floor, req\_time;

\* @EFFECTS:

\* req\_floor == temp[1] && req\_time == System.time;

\* (\this.temp[2] == "UP") ==> \this.floors.floor(req\_floor-1).upbutton == true && \this.floors.floor(req\_floor-1).uptime = req\_time;

\* (\this.temp[2] == "DOWN") ==> \this.floors.floor(req\_floor-1).downbutton == true && \this.floors.floor(req\_floor-1).downtime = req\_time;

\*/

}

楼层请求解析也会引用时间。然后，使用split，把请求内容分配到对应的属性。之后把对应楼层（第几层）对应的按钮（上行还是下行）按下（变亮）。

public void parseElevator(double time) throws Exception {

/\*\*

\* @REQUIRES: time (double);

\* @MODIFIES: elevid, req\_floor, req\_time;

\* @EFFECTS:

\* elevid == temp[1] && req\_floor == temp[2] && req\_time == System.time;

\* \this.elevs.elev(elevid-1).buttons.button(req\_floor-1) == true;

\*/

}

电梯请求解析一会引用时间。然后也使用split，把请求内容分配到对应的属性。之后把对应电梯（第几台）中的楼层按钮（第几层的按钮）按下。

## addReq

对所有队列（不论是楼层请求队列，电梯请求队列，电梯队列，电梯按钮队列，楼层队列）都有addReq方法。此方法在所有队列中都具有一样的实现。

例子：

public synchronized boolean addFloorReq(FloorRequest req) {

/\*\*

\* @REQUIRES: req (FloorRequest);

\* @MODIFIES: this.floorqueue;

\* @EFFECTS: floorqueue.addQueue(req);

\*/

}

此例子调用的是楼层请求队列的addReq。会引用要被加进去的内容（请求队列就引用对应的请求类；电梯，楼层，电梯楼层按钮，就引用整数index）。然后调用Java提供实际的queue.add方法。

public synchronized void addPoint(int point) {

/\*\*

\* @REQUIRES: point (int);

\* @MODIFIES: this.point;

\* @EFFECTS: \this.point == \old(\this.point) + 1;

\*/

}

此方法只是简单地对电梯加运动量。被加的数目便是被引用的整数point。被执行后运动量会被加。

## moveUp() 和moveDown()

电梯上行和下行基本上是一样的方法实现。

public boolean moveUp() {

/\*\*

\* @REQUIRES: None;

\* @MODIFIES: this;

\* @EFFECTS:

\* (\result == true) ==> (\this.curFloor == \old.curFloor+1) && (\this.direction == UP);

\* (\result == true) ==> (\this.point == \old.point+1);

\* (\this.curFloor == \this.tarFloor) ==>

\* (\this.tarFloor.upbutton == false && \this.floorqueue.remove(0)) &&

\* (\this.button(tarFloor) == false && \this.elevqueue.remove(0));

\*/

}

例子中是上行的方法，被执行后，只要电梯还未到目的楼层，电梯会上行一层，方向自然变为UP。每次上一层运动量会加1。到了之后若是有对应的按钮被按（不论楼层中的上行按钮还是电梯中那个楼层的按钮），都会被关掉，然后此请求会被删掉，代表已经被处理，不会再被处理。

public void pickUp() throws CannotRemoveException {

/\*\*

\* @REQUIRES:None;

\* @MODIFIES:this;

\* @EFFECTS:

\* (\this.curFloor.canPickUpFloor) ==> (\this.curFloor.button == false && \this.isPickUp == true);

\* (\this.curFloor.canPickUpElev) ==> (\this.button(curFloor) == false && \this.isPickUp == true);

\*/

}

此方法是在电梯上行或下行是每次变一层会被调用的方法。是用来检查那个楼层中是否有可以捎带的请求。如果有，就要把电梯的isPickUp Boolean变为true，对应楼层的按钮也要被关掉，所以不会再被处理（因为已处理过）。