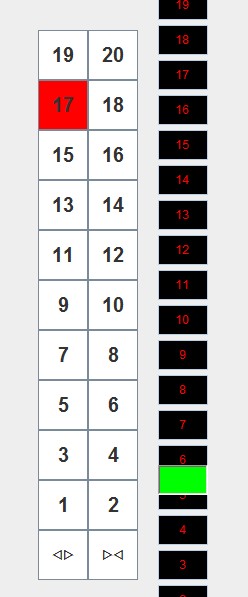
**电梯调度设计报告**

1152745 邱峰

1. **电梯说明**

电梯内部视图 1~20为相应的楼层。 按下即可响应相应楼层



左边的1~20为电梯内部按钮，按下后变成红色即被响应。

右边黑色的框为楼层示意图，绿色的为关门状态电梯，此刻电梯正在5~6楼之间向上行走

当电梯到达相应楼层后，会变粉红色，表示电梯门被打开

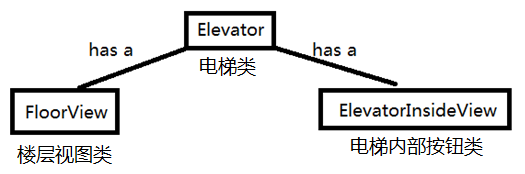
C:\Users\飘飘乎居士\AppData\Roaming\Tencent\Users\454620481\QQ\WinTemp\RichOle\@V`LSVLA8W%@Z{]QB2{_%CD.jpg这两个按钮为开门和关门，开门键只有在电梯停下后，或者电梯正在开门时（延长开门时间）响应，关门键只有在开门以后，按下可以立刻关门，并继续上下移动。

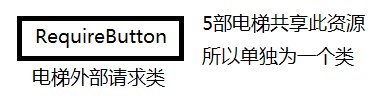
外部的38个按钮（5部电梯共享此按钮），△表示向上请求，▽表示向下请求

1. **程序设计图**

**电梯状态转移图**：见附件，由于图片太大，没有处理好，插入进word可能看不清楚。

**类的设计图**





**类方法设计图**

**Elevator**类：

run： 启动电梯

upFloor： 让电梯上楼

downFloor： 让电梯下楼

upOrDown （核心函数）： 判断下一步电梯运行状态，返回给state

setFloor： 设置电梯楼层

getFloor： 返回电梯目前所在楼层

setArriveFloor： 响应电梯按钮，标记该楼层为需要到达的楼层

openDoor： 开门

closeDoor： 关门

reopenDoor： 延长开门时间，即重新执行开门

search： 当电梯处于静止状态时，检查是否有外部按钮被相应

add： 将Elevator内所有的视图加入到panel当中

**FloorView**类：

FloorView： 构造方法，画出楼层视图

**ElevatorInsideView**类：

ElevatorInsideView： 构造方法，画出内部图

buttonFloorListener： 电梯内部按钮监听器，调用setArriveFloor

openDoorListenner： 开门键监听器

closeDoorListener： 关门键监听器

**RequireButton**类

RequireButton： 构造方法，画出电梯外部按钮，绑定监听器

requireButtonListenr： 外部按钮监听器，响应外部按钮

getRequire: 按照先到先得的顺序，服务外部未被响应按钮（电梯顺路时除外）

isShortset： 判断此电梯是否为所有静态电梯中路程最短的电梯，分配外部相

应按钮给电梯

**重要参数及变量**

**Elevator**类

**boolean arrive[40];**   
//用于记录电梯内部的按钮，也就是电梯需要到达的楼层

arrive是用于存放电梯请求的信息。0~18表示1~19楼的一个向上走的请求，21~38表示20楼到2楼向下走的请求。

Hint ：arrive[19]与arrive[39]没有使用是因为在20楼不存在向上的请求以及在1楼不存在向下的请求

之所以不给arrive设置20个，而设置40个的原因是因为，电梯分上下行走，当5楼有2个人，一个按往下行走，一个按往上行走，那么，应该响应的是不同的按钮。

**int state**

//电梯的状态

将电梯的状态分为5个状态

0 静止 （等待状态）

1 向上走（去接一个向上走的请求） 比如 电梯在1楼； 3楼按下一个往上走的请求

2 向上走（去接一个向下走的请求） 比如 电梯在1楼； 3楼按下一个往下走的请求

-1 向下走（去接一个向下走的请求） 比如 电梯在5楼； 3楼按下一个向下走的请求

-2 向下走 （去接一个向上走的请求） 比如 电梯在5楼； 3楼按下一个向上走的请求

**int willArrive** //电梯沿着这个方向运行最终的楼层

**int floor**  //目前电梯在哪一个楼层

**RequireButton**类

boolean require[40] //电梯外部的按钮请求状态

LinkedList<Integer> requrieLink 外部请求按钮的队列，之所以用链表实现是方便在电梯顺路的情况下进行删除。

1. **实现思想**
2. 从最简单的一部电梯的内部按钮开始

只有一部电梯时，电梯内部的按钮必须优先实现，除非是在“恰好路过的情况”下才响应外部按钮，否则，一律不考虑外部按钮。

**在run方法中，每次调用upOrDown（内部调度核心函数）函数。**

upOrDown函数会根据上一步的state按照不同的顺序将arrive数组扫描一遍，并将返回值返回给新的state。

例如，当state=1或者state=2时，upOrDown函数会从20楼扫描到floor，一旦遇到arrive[i] || arrive[40-i]=true时（arrive[40-i]表示高层向下的请求，之所以从20楼开始扫描是因为电梯目前状态向上，willArrive为一个需要到达的最高处需求），修改willArrive，并且返回新的state

当state=-1和state=-2时，相同。

当state=0时，

如果电梯所在楼层有开门相应（即arrive[floor]=true实际上这是不可能的情况，除非那个人睡着了，然后电梯一直停在那边，等自己醒了后按下了个floor）这时候执行开门操作。

否则，将arrive[i]扫描一遍，一旦发现有一个楼层需要请求，立刻修改willArrive，并返回新的电梯状态给state。

如果电梯内部没有请求了（arrive为false），调用search函数，看是否有外部请求（第二次修改代码后这里用了先到先响应的方法），如果有的话，需要修改arrive函数（在search会实现），之后可以在下一次调用upOrDown函数中在修改后的arrive函数中改变电梯状态。

对于返回的新的state

如果返回值为1或者2执行upFloor函数；

如果返回值为-1或者-2，执行downFloor函数；

upFloor函数：

执行电梯上楼动画，在上完一楼以后，将floor++，并且判断arrive[floor-1]和require[floor-1]，如果arrive[floor-1] || require[floor-1]=true以及，表示该楼需要开门，则停止电梯，进行开门。之后返回run函数，再次调用upOrDown查看电梯下一步操作。否则无需开门，直接返回到run函数。

在这种情况下会响应向下的按钮。

state==2（电梯本来就是想上接向下的乘客），并且同楼层外面的人只按下了向下的按钮（否则会先相应想上的按钮），并且floor=wiiArrive（即到达了电梯需要到达的最高楼层）

downFloor函数与upFloor类似。

**关于电梯内部按钮按下newFloor与arrive数组的响应**

当newFloor>floor（电梯目前所在的楼层），setArriveFloor(i-1)标记该楼为上楼的请求

当按下楼层<floor，setArriveFloor(40-i)标记该楼为下楼请求

当按下的楼层=floor 分下面三种情况 （如果你不做滑块可以省掉这个）

当电梯处于开门状态，延长开门时间

当电梯正在向上走（由于=floor，保证经过了floor楼层），标记floor向下的请求

当电梯正在向下走，标记floor向上请求

**关于开门关门**

在响应openDoorButton时，首先判断只有在doorOpen的情况下才会响应，否则不给予按钮响应（防止电梯还在上下楼，有用户恶作剧按下开门。。然后门就被打开了）

默认情况，在电梯到达arrive[i]的情况下，电梯自动打开2s的门，如果有进一步需求，可以按下开门键，此时，会重新执行一次openDoor，即再开2秒的门。

关门按钮，在门是打开的情况下，按下关门键可以立刻关上门，然后出发去下一层。（这个按钮的作用是节约时间，比如电梯默认打开2s，但是在1s的时候，用户就不想等了，这时候按下关门键可以立刻关门然后出发）

至此，完成了所有关于一部电梯仅响应内部按钮的需求。

1. 响应仅一部电梯时的外部按钮

响应外部按钮只有在电梯静止状态下响应，除非是电梯“顺路”的情况，顺路的情况在upFloor和downFloor函数中已经解决。

剩下state=0的情况，在upOrDown函数中，首先搜索了arrive数组，如果全为false，表示电梯内部没有需要到达的楼层，这样从search函数中，按照先到先得的队列思想，将require[i]赋值给arrive[i]，同时关闭require的响应（将require[i]=false，不改变require[i]的颜色，因为只有到达楼层的时候，按钮颜色才会被改变），这样在下次调用upOrDown函数时，就实现了将外部按钮转化为内部按钮的功能。

至此，完成了所有关于一部电梯的功能

1. 响应多部电梯（search函数，协调5部电梯管理外部按钮）

5部电梯是5个线程，线程之间相互工作，互不干涉。因此，内部原理完全相同，要在

创建的时候多创建4个Elevator类即可。

唯一不同之处是外部按钮。在使用队列取出一个最早按下button的require楼层之后，将require分配给arrive的情况有略微差别。这里，我采用的是最短调度法，即对于一个外部响应的电梯，在所有的静止电梯中，找到一个线路最短的电梯分配过去。

至此，完成了一部电梯到多部电梯的过度，也完成了5部电梯的调度工作。

**四、可以改进的地方**

在电梯调度的时候，由于我是先讲外部按钮分配到arrive中，故存在一个很明显的缺陷，即，当电梯1处于静止状态，此时20楼的电梯正在往下走，18层有人按下向下的按钮。显然应该是20楼的电梯顺路可以把他带下来，但是我却会派1楼的那个静止的电梯。当然，如果5部电梯都在运行时，我就会派20楼的电梯将他载下来，这就是因为在search函数中优先将require分配给了处于静止状态的电梯的原因

在电梯调度的时候，我是按照先到先得的顺序，实际中个人觉得这种情况尽管可以接受，但有时也会带来一些不方便的地方，比如：某些楼层人比较多，解决方法可以设定优先级并加上老化现象解决。

1. **心得体会**

本次电梯调度是在崔昊学长讲完他的电梯调度之后完成的，借用了崔昊学长电梯内、外

都用40个boolean类型表示请求和从一部电梯入手的建议。

另外，感谢崔昊学长在我完成之后帮我测试出一个bug。

在做电梯上下楼的动画时，遇到了不少的问题，尤其是程序莫名其妙自动调用了interrupt，至今都不知道为什么，但我通过了其他的补丁避免了这些情况的发生。

还有一点就是在电梯开关门的细节中，有许多的细节问题处理的都很复杂，导致代码不好看。

对于操作系统的理解，我的理解是，用线程模拟进程，每一部电梯是一个进程，而所有的请求是资源，我们做这个电梯的目的是模拟进程调度的问题，也就是模拟所有的请求资源如何分配给电梯，电梯应该先完成哪些资源的情况。

在完成了电梯之后，第二次对电梯进行了优化，即再search函数中，响应外部函数增加了一个队列，即先到先得的响应（开始是直接用for循环响应，后来觉得可能存在饿死现象，如果楼下有人一直按按电梯，然后20楼有人按了，20楼可能被饿死），增加了队列后避免了这种情况。

5部电梯之间的协调在search函数中，如果有更好的方法可以重载search函数。

每一步电梯的调度在upOrdown函数中，可以修改upOrDown函数的核心算法思想

最后，再次感谢崔昊学长的帮助，还有java助教的帮助。