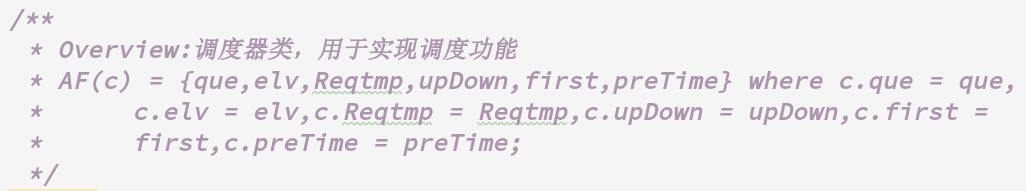
正确性论证

1. Scheduler类
2. 抽象对象得到有效实现论证



表示对象为que,elv,Reqtmp,upDown,first,preTime，通过抽象函数映射为带电梯、请求队列等属性的虚拟电梯调度系统。

1. 对象有效性论证
2. Scheduler构造方法，它初始化所有的rep，因此repOK的结果显然为真。
3. checkFloor方法，此方法中，被修改的rep有first，preTime,que,Reqtmp,upDown等，其中first只会被改成1,或0，preTime来自于请求的时间，而请求的时间会被检查是否大于或等于0，因此preTime一定大于或等于0，que只会添加新请求，不会为Null，upDown只会在1,0之间取值，因此均不会使repOK变为无效，因此此方法不违背表示不变式。
4. checkElevator方法，此方法中被修改的rep有first，preTime,que,Reqtmp，其中first只会被改成1,或0，preTime来自于请求的时间，而请求的时间会被检查是否大于或等于0，因此preTime一定大于或等于0，que只会添加新请求，不会为Null，因此均不会使repOK变为无效，因此此方法不违背表示不变式。
5. 方法实现正确性论证
6. Scheduler构造方法，其规格的划分为

<initialize the class> with <any input>

由于方法不需要任何参数，一旦进入即可初始化相应成员，因此满足其规格。

1. repOK方法，其规格的划分为

<\result=true> with <(que!=null) && (elv!=null) && (upDown==1 || upDown==0 || upDown==-1) && (first==1 || first==0) && preTime>=0>

<\result=false> with <other conditions>

而repOK直接返回的是(que!=null) && (elv!=null) && (upDown==1 || upDown==0 || upDown==-1) &&(first==1 || first==0) && preTime>=0;因此满足规格。

1. checkFloor方法，其规格的划分为(为简化起见省略了字符串向数字的转换过程以及字符串相等判别)

<\result = 0> with <type==null || floor==null || Direction==null || time==null>

<\result = 0> with <!(type==”FR” && floor<11 && floor > 0 && (Direction== “UP” || Direction == “DOWN”) && time>=0 && time < Integer.MAX\_VALUE) >

<\result = 0> with <(type==”FR” && floor<11 && floor > 0 && (Direction== “UP” || Direction == “DOWN”) && time>=0 && time < Integer.MAX\_VALUE) && first==1 && !(time==0 && floor==1 && Direction==”UP”)>

<\result = 0> with <(type==”FR” && floor<11 && floor > 0 && (Direction== “UP” || Direction == “DOWN”) && time>=0 && time < Integer.MAX\_VALUE) && first!=1 && time<preTime >

<\result = 0> with <(type==”FR” && floor<11 && floor > 0 && (Direction== “UP” || Direction == “DOWN”) && time>=0 && time < Integer.MAX\_VALUE) && first!=1 && time>=preTime && floor==1 && Direction==”DOWN”>

<\result = 0> with <(type==”FR” && floor<11 && floor > 0 && (Direction== “UP” || Direction == “DOWN”) && time>=0 && time < Integer.MAX\_VALUE) && first!=1 && time>=preTime && floor==10 && Direction==”UP”>

<\result = 1> with <(type==”FR” && floor<11 && floor > 0 && (Direction== “UP” || Direction == “DOWN”) && time>=0 && time < Integer.MAX\_VALUE) && first!=1 && time>=preTime && !(floor==1 && Direction==”DOWN”) && !(floor==10 && Direction==”UP”)>

<\result = 0) with <any exception occurs>

首先，如果type、floor、Direction、time有一个为null，那么中途会产生异常而被try捕捉到，因此也返回0。然后方法检查如果一条请求不是符合楼层请求的标准形式(标识符为FR，楼层号在[1,10]之间，方向为UP或者DOWN，时间在[0,Integer.MAX\_VALUE]之间)，那么返回0。其次，如果该请求是第一条请求但不是[FR,1,UP,0]，那么也返回0。如果该请求比前一条请求时间早，那么是乱序请求， 也返回0。如果该请求是1层但是方向向下，或者是10层方向向上，那么也返回0。如果中途产生了异常，捕获异常之后也返回0。最后当上述返回0的请求都不满足的时候，返回1，并完成相应操作。综上，checkFloor方法正确，满足相应规格要求。

1. checkElevator方法，其规格的划分为

<\result = 0> with <type==null || floor==null || time==null>

<\result = 0> with <!(type==”ER” && floor<11 && floor > 0 && time>=0 && time < Integer.MAX\_VALUE) >

<\result = 0> with <(type==”ER” && floor<11 && floor > 0 && time>=0 && time < Integer.MAX\_VALUE) && first==1>

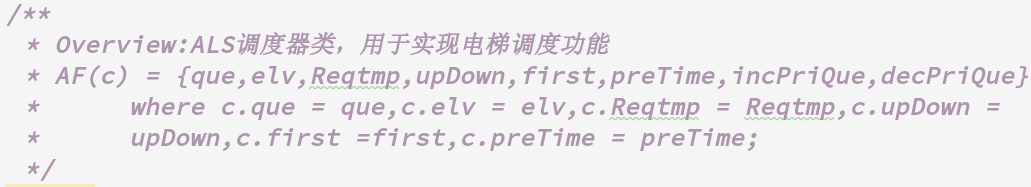
<\result = 0> with <(type==”ER” && floor<11 && floor > 0 && time>=0 && time < Integer.MAX\_VALUE) && first!=1 && time<preTime>

<\result = 1> with <(type==”ER” && floor<11 && floor > 0 && time>=0 && time < Integer.MAX\_VALUE) && first!=1 && time>=preTime>

<\result = 0) with <any exception occurs>

首先，如果type、floor、time有一个为null，那么中途会产生异常而被try捕捉到，因此也返回0。方法检查该请求是否为标准电梯请求的形式(标识符为ER，楼层在[1,10]之间，时间在[0,Integer.MAX\_VALUE]之间)，如果不是，返回0，然后，如果first依然等于1，那么也返回0，因为第一条指令被要求是[FR,1,UP,0]，其次检查请求的时间是否早于前一条请求的时间，是则返回0 ，如果中途出现异常，也返回0。如果以上返回0的请求都不满足，则返回1。综上，该方法满足自身规格。

1. ALSScheduler类
2. 抽象对象得到有效实现论证



表示对象为que,elv,Reqtmp,upDown,first,preTime,incPriQue,decPriQue，相比于父类多了incPriQue以及decPriQue，因此通过抽象函数映射为比父类调度系统多了优先请求队列属性的虚拟ALS电梯调度系统。

1. 对象有效性论证
2. ALSScheduler方法，它初始化相应的成员变量，repOK结果显然为真。
3. compute方法，修改的属性包括Reqtmp ，elv，upDown，incPriQue，decPriQue;其中elv方法只是调用其方法，不会使其为null，而upDown只会在0,1,-1中取值，而incPriQue,decPriQue只会向其中添加请求或者清空请求，也不会造成null，因此repOK为真，不违背表示不变式。
4. checkBulb方法，不修改任何属性，因此不会让repOK变为假。
5. checkPickUp方法，同样不修改任何属性，因此不会让repOK变为假。
6. 方法新增或者重写方法正确性论证（由于父类方法已经进行过正确性论证，因此在子类中不再继续论证）
7. ALSScheduler构造方法，其规格可以划分为

<initialize the class> with <any>

而此方法直接调用了super()，同时初始化了incPriQue以及decPriQue，完成了规格所述任务。

1. repOK方法，其规格可以划分为

<\result=true> with <super.repOK() && incPriQue!=null && decPriQue!=null>

<\result=false> with <other conditions>

由于其直接返回了super.repOK() && incPriQue!=null && decPriQue!=null;因此满足相应规格。

1. compute方法

此方法的规格可以被划分成：

<schedule all requests> with <que.size()>0>

<do nothing> with <que.size()==0>

首先此方法用一个for语句开头，如果que.size()为0，那么该for语句无法执行，直接退出。

如果que.size()大于0，那么开始逐个遍历que中的请求，进行调度。每次从que中取出一条指令，令它为主请求，更新电梯状态，如果电梯状态是向上或者向下，遍历当前指令之后的所有指令，如果能捎带就添加进捎带的优先队列，随后运行电梯执行完队列中所有指令。如果电梯状态是STILL，那么再遍历que，找出其中可以算无效的请求，随后运行电梯并执行请求。

综上，此方法可以满足自身规格。

1. checkBulb方法，此方法的规格可以划分为

<\result=false> with <r1==null || r2==null>

<\result=true> with <r1!=null && r2!=null && r1.getType.equals(“FR”) && r2.getType.equals(“FR”) && r1.getFloor==r2.getFloor && r1.getUpDown==r2.getUpDown>

<\result=true> with <r1!=null && r2!=null && r1.getType.equals(“ER”) && r2.getType.equals(“ER”) && r1.getFloor==r2.getFloor>

<\result=false> with <other conditions>

由于此方法一上来就判断了是否(r1==null || r2==null)，如果是则返回false。然后判断r1与r2是否都是FR请求并且二者楼层相等且方向相同，是则返回true。然后判断r1与r2是否都是ER请求并且二者楼层相同，是则返回true。最后，该方法返回false，即对于其他情况，直接返回false。因此该方法满足规格要求。

1. checkPickUp方法

此方法的规格可以被划分成

<\result=false> with <r==null || q==null>

<\result=false> with <当前输入的请求r在当前系统方向为upDown的条件下不能捎带到q中>

<\result=true> with <当前输入的请求r在当前系统方向为upDown的条件下可以被捎带到q中>

方法首先判断r或者q是否为null，如果是则返回false，然后判断请求所在楼层是否在电梯方向上，如果是则将局部变量sameDir设置为true，否则设置为false，由于最终返回的表达式包含&&sameDir，因此如果请求楼层不在电梯方向上则返回false，否则进行下一步判断。

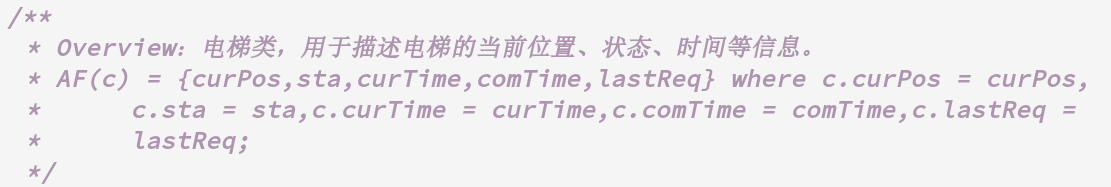
方法随后计算响应时间，也就是电梯能够响应r的最早时间resTime，该时间来自于电梯到r的距离加上电梯还要经过的楼层等待时间，如果该时间<=r产生时间，那么说明不可捎带，因为这表示当请求产生的时候，电梯已经过去了或者正在请求所在楼层，根据指导书该情况不捎带，如果resTime>r产生时间，则进行下一步。

如果当前队列中已经有与r楼层相同的请求，那么r也不能捎带，因为这表示出现了重复请求或者无效请求，返回的也是false(因为sameReq.size()大于0，那么最后return的结果中包含&&sameReq.size()==0)，如果没有相同的请求，那么执行下一步。

由于能捎带的请求要么是电梯类请求，要么是FR请求但是楼层不会超过当前任务集中的最远楼层。因此只有满足以上两点以及前面所有判断都能通过，才能返回true表示可以捎带。

综上，此方法能满足自身规格。

1. ALSElevator类
2. 抽象对象得到有效实现论证



表示对象为cusPos,sta,curTime,comTime,lastReq，通过抽象函数映射为带当前位置，状态，当前时间，完成时间以及能记住上一条指令的电梯。

1. 对象有效性论证
2. ALSElevator方法，将原本是0的cusPos赋值为1，不会导致repOK为假。
3. toString方法，不会修改任何成员变量，而且由于string的immutable特性，返回的结果不会让外界可以借此修改lastReq，因此不会导致repOK为假
4. update方法，修改的成员变量有curPos,curTime,comTime,sta，其中curPos来自于请求r，而请求r在传入update方法之前经过了检查确保其楼层在1-10楼之间，因此不会影响repOK；而curTime = |r.getFloor-curPos|+comTime，只要之前comTime>=0(repOK为真则可以保证)，那么curTime也必定>=0；而comTime = curTime+1由于上文curTime>=0，因此comTime必定>=0；而sta经过判断只会在1,-1,0之间取值，因此也不会使repOK为假。总之，该方法不会让repOK为假。
5. getCurPos方法，不修改任何成员变量，不会导致repOK为假。
6. getComTime方法，不修改任何成员变量，不会导致repOK为假。
7. getState方法，修改的只有sta，而sta必定在1,-1,0之间取值，因此不会导致repOK为假。
8. getLastReq方法，不修改任何成员变量，不会导致repOK为假。
9. updateComTime方法，只有当t>comTime的时候才更新comTime为t，因此只要在comTime>=0的前提下，那么更新后的comTime也必定>=0，因此不会导致repOK为假。
10. 方法正确性论证
11. ALSElevator方法，其规格划分为

<curPos = 1> with <any>

显然该方法直接赋值curPos为1，满足规格。

1. repOK方法，其规格划分为

<\result = true> with <(1<=curPos) && (curPos<=10) && ((sta==1) || (sta==-1) || (sta==0)) && (curTime>=0) && (comTime>=0)==true>

<\result = false> with <other conditions>

显然，该方法直接返回了布尔表达式(1<=curPos) && (curPos<=10) && ((sta==1) || (sta==-1) || (sta==0)) && (curTime>=0) && (comTime>=0)，因此任何不满足该表达式的都会返回false，任何满足该表达式的都会返回true。

1. toString方法，规格可以划分为

<\result=lastReq+"/("+curPos+",UP,"+df.format(curTime)+")"> with <sta==1>

<\result==lastReq+"/("+curPos+",DOWN,"+df.format(curTime)+")"> with <sta==-1>

<\result==lastReq+"/("+curPos+",STILL,"+df.format(comTime)+")"> with <sta==0>

首先，方法判断了sta，如果sta为0，则返回lastReq+"/("+curPos+",STILL,"+df.format(comTime)+")"，如果sta不为0，则根据sta的具体值，sta为1则返回lastReq+"/("+curPos+",UP,"+df.format(curTime)+")"，sta为-1则返回lastReq+"/("+curPos+",DOWN,"+df.format(curTime)+")"。

1. update方法

其规格可以划分为

<do nothing> with <r==null>

< lastReq == "["+r.getType()+","+r.getFloor()+","+r.getTime()+"]"> with < r.getLen()!=4>

< (lastReq == "["+r.getType()+","+r.getFloor()+","+"UP"+","+r.getTime()+"]")> with < (r.getLen()==4) && (r.getUpDown()==1)>

< (lastReq == "["+r.getType()+","+r.getFloor()+","+"DOWN"+","+r.getTime()+"]")> with < (r.getLen()==4) && (r.getUpDown()!=1)>

<(curTime == Math.abs(r.getFloor() - curPos) \* 0.5 + comTime) && (comTime = curTime+1) && (r.getFloor() < curPos ==> sta==-1) && (r.getFloor() > curPos ==> sta==1) && (r.getFloor()==curPos ==> sta==0) && (curPos = r.getFloor())> with <r.getAlreadyOut()==1>

首先，方法判断了r是否为null，如果是直接返回，然后判断r的alreadyOut是否为1，如果是1，则更新curTime,comTime,sta,curPos等变量，表示对电梯状态的更新，否则不对电梯进行更新。这与最后一条划分相吻合。随后，方法判断r的长度，如果r的长度为4，并且r是一条向上的请求，就将lastReq更新成("["+r.getType()+","+r.getFloor()+","+"UP"+","+r.getTime()+"]")，如果r不是一条向上的请求，则将lastReq更新成("["+r.getType()+","+r.getFloor()+","+"DOWN"+","+r.getTime()+"]")，这与第2,3条划分相吻合。最后如果r的长度不为4，则将lastReq更新成"["+r.getType()+","+r.getFloor()+","+r.getTime()+"]"，表示上一条请求是电梯累请求。总之，该方法能完成规格中所描述的任务。

1. getCurPos方法

该方法的规格划分为<\result = curPos> with <any>，由于是直接返回的curPos，因此显然满足规格。

1. getComTime方法

该方法的规格划分为<\result=comTime> with <any>，由于是直接返回的comTime，因此显然满足规格。

1. getState方法

该方法的规格划分为

<\result = sta> with <r==null>

<\result = sta && sta = 1> with <(curPos<r.getFloor() && r.getFloor()<=10)>

<\result = sta && sta = -1> with <(curPos>r.getFloor() && r.getFloor()>=1)>

<\result = sta && sta = 0> with <other conditions>

由于方法首先判断了r是否为null，如果是则返回未经过修改的sta，然后判断(curPos<r.getFloor() && r.getFloor()<=10)是否成立，若成立则让sta=1，然后判断(curPos>r.getFloor() && r.getFloor()>=1)是否成立，若成立则让sta=-1，最后如果上述条件都不成立，则让sta=0，然后返回sta。因此显然该方法满足规格的实现。

1. getLastReq方法

该方法的规格划分为

<\result=null> with <lastReq==null>

<\result=new Request(ss[1],Integer.parseInt(ss[2]),Long.parseLong(ss[3])> with <lastReq!=null && lastReq.split("[\\]\\[\\,]").length==4>

< \result=new Request(ss[1],Integer.parseInt(ss[2]),1,Long.parseLong(ss[4])> with < lastReq!=null && lastReq.split("[\\]\\[\\,]").length!=4 && ss[3].equals("UP")>

< \result=new Request(ss[1],Integer.parseInt(ss[2]),0,Long.parseLong(ss[4])> with < lastReq!=null && lastReq.split("[\\]\\[\\,]").length!=4 && ! ss[3].equals("UP")>

首先，该方法判断lastReq是否为null，如果是null就返回null，然后将lastReq以 “[],”为分界符拆分成若干段，如果拆分后段长为4，则返回的是一个Request对象Request(ss[1],Integer.parseInt(ss[2]),Long.parseLong(ss[3])，如果拆分之后段长不为4，说明是楼层请求，如果该楼层是向上的请求，则返回的Request对象是Request(ss[1],Integer.parseInt(ss[2]),1,Long.parseLong(ss[4])，否则返回的Request对象是Request(ss[1],Integer.parseInt(ss[2]),0,Long.parseLong(ss[4])。因此，该方法满足其规格的划分。

1. updateComTime方法

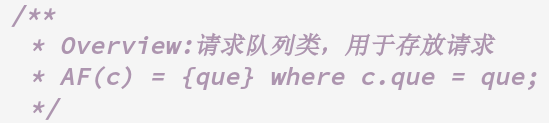
该方法的规格划分为

< (comTime = t && \result=true)> with <t>comTime>

<\result = false> with < t<=comTime>

首先该方法判断了t是否大于comTime，如果是则将comTime更新为t，然后返回true，否则不更新comTime，然后返回false。综上，该方法能满足其规格的划分。

1. ReqQue类
2. 抽象对象得到有效实现论证



表示对象为que，通过抽象函数映射为存放请求的队列。

1. 对象有效性论证
2. ReqQue方法，将que对象初始化，不会导致repOK为假
3. addReq方法，向que对象中添加请求，不会导致que为null，也不会导致repOK为假。
4. size方法，返回que对象的大小，不会导致que为null，也不会导致repOK为假。
5. get方法，获取下标为i的请求，不会导致que为null，也不会导致repOK为假。
6. 方法正确性论证
7. ReqQue方法，

此方法的规格可以划分为

< \this.que = new ArrayList<Request>()> with <any>

由于方法中直接将que赋值为new ArrayList<Request>()，因此满足该规格的划分。

1. repOK方法

此方法的规格可以划分为

<\result=false> with <que==null>

<\result=true> with <que!=null>

由于方法直接返回了que!=null，因此如果que为null则自然返回false,如果que不为null自然返回true，满足规格划分。

1. addReq

该规格的划分为

<que.size()+=1 && que.contains(r)> with <any>

由于方法直接调用que的add方法，因此r被放进que中，que的大小相比于之前增长了1，同时que也会包含r，因此满足规格的划分。

1. size

该方法的规格划分是

<\result = que.size> with <any>

由于方法直接返回了que.size()，因此满足规格划分。

1. get

该方法的规格划分是

<\result = que.get(i)> with <0<=i <que.size()>

<\result = null> with <i<0 || i>=que.size()>

由于方法一开始就判断了如果i<0或者i>=que.size成立，则直接返回null，如果不成立，则返回que.get(i)，因此该方法满足相应的规格划分。