Controller

1. 抽象对象得到了有效实现论证

论证类的overview明确了抽象对象，且类的rep能够通过抽象函数映射到相应的抽象对象。在有继承层次的情况下，要注意区分子类抽象函数与父类抽象函数的关系，并能够有效重用父类的相应论证结果。

static变量first表示是否为第一条指令，第一次过后变为false，只被judgefr修改，可以起到其作用

1. 对象有效性论证

论证在任意时刻，一个类的任意对象都能保证repOK为真，即任意一个方法的执行都不会导致rep为假。如果类的属性声明为public，则无法进行有效论证。在有继承层次的情况下，如果父类所有的rep都通过private进行保护，则无需进行额外论证，否则（即声明为protected）必须针对每个对父类rep状态进行修改的方法进行论证，证明不会导致父类对象的repOK为假。

*注：首先假设repOK的实现是正确的。在方法论证部分对repOK的正确性进行论证。*

1. 针对构造方法，论证初始态对象的repOK为真。

Controller没有构造方法，初始条件下sign，first均等于true，此时repOK值为真

1. 逐个论证每个对象状态更改方法的执行都不会导致repOK的返回值为false。

judgefr方法

传入一条FR指令，对于first的修改表示着是否为第一条指令，只存在只存在false=true和first=false两种语句，所以（fist!=true&first!=false）不存在。!((first!=true&first!=false))永真，repOK为真。

judgeer方法

是FR方法的ER版本，理由同上。

1. 该类不存在其他方法。
2. 综上，对该类任意对象的任意调用都不会改变其repOK为true的特性。因此该类任意对象始终保持对象有效性。
3. 方法实现正确性论证

目标：在方法的前置条件和后置条件没有逻辑矛盾的前提下，根据前置条件和后置条件分析整理出完整的{<effect Y> with <input partition X>}，然后对照方法代码实现来论证给定每个<partition X>，方法执行结果都满足<effect Y>。

(a)**staticboolean**judgefr(String s0,String s1,String s2,String s3,String[][] Queue,**double**Max){

/\*

@Require:String s0,s1,s2,s3; s0.equals("FR"|"ER"),1=<s1&&s1<=Floor.floormax,s2.equals("UP"|"DOWN"),s3>=0;

@Modified:first;

@Effects:\result==!((Double.parseDouble(s3) <Max(Request.getLast()))||

\all int m;0<=m&m<=1000;\exist m,s0.equals(Queue.getQueue()[m][0])&&s1.equals(Queue.getQueue()[m][1])&&s2.equals(Queue.getQueue()[m][2])&&s3.equals(Queue.getQueue()[m][3]))||

(first&&!(Double.parseDouble(s1)==1&&s2.equals("UP")&&Double.parseDouble(s3)==0))；

if sign first==false;

\*/

根据上述过程规格，获得如下的划分：

<sign==false>with< (Double.*parseDouble*(s3) < Max)>

<sign==false>with<\all int m;0<=m&m<=1000;\exist m,s0.equals(Queue.getQueue()[m][0])&&s1.equals(Queue.getQueue()[m][1])&&s2.equals(Queue.getQueue()[m][2])&&s3.equals(Queue.getQueue()[m][3]))>

<sign==false>with< (first&&!(Double.parseDouble(s1)==1&&s2.equals("UP")&&Double.parseDouble(s3)==0)>

<first==false>with< (sign==ture)>

* 方法检查目前时间是否小于目前已经到达的最大时间，成立时返回false
* 方法检查整个队列是否有重复请求，成立时返回false
* 方法检查当目前是第一条指令时是否为（FR,1,UP,0）,否则返回false
* 方法检查是否指令有效，若有效将first改为false标记已经不是第一条指令
* 没有前置条件，所以操作符合前置条件

(b)**staticboolean**judgeer(String s0,String s1,String s2,String[][] Queue,**double**Max){

/\*

@Require:String s0,s1,s2,s3; s0.equals("FR"|"ER"),1=<s1&&s1<=Floor.floormax,s2.equals("UP"|"DOWN"),s3>=0;

@Modified:first;

@Effects:\result==!((Double.parseDouble(s2) < Max)||

\all int m;0<=m&m<=1000;\exist m,s0.equals(s0.equals(Queue.getQueue()[m][0])&&s1.equals(Queue.getQueue()[m][1])&&s2.equals(Queue.getQueue()[m][2]))||

(first);

\*/

根据上述过程规格，获得如下的划分：

<sign==false>with< (Double.*parseDouble*(s2) < Max)>

<sign==false>with<\all int m;0<=m&m<=1000;\exist m,s0.equals(s0.equals(Queue.getQueue()[m][0])&&s1.equals(Queue.getQueue()[m][1])&&s2.equals(Queue.getQueue()[m][2]))>

<sign==false>with< (first)>

* 方法检查目前时间是否小于目前已经到达的最大时间，成立时返回false
* 方法检查整个队列是否有重复请求，成立时返回false
* 方法检查当目前是第一条指令,如果是返回false（ER不能做第一条）
* 没有前置条件，所以操作符合前置条件

Queue

1.抽象对象得到了有效实现论证

论证类的overview明确了抽象对象，且类的rep能够通过抽象函数映射到相应的抽象对象。在有继承层次的情况下，要注意区分子类抽象函数与父类抽象函数的关系，并能够有效重用父类的相应论证结果。

Static对象Queue[][]存放所有的正确的指令，通过insetfrinseter添加，拿走指令的时候直接删除，可以起到其作用

staticn记录目前指令总个数不断往后添加

2.对象有效性论证

论证在任意时刻，一个类的任意对象都能保证repOK为真，即任意一个方法的执行都不会导致rep为假。如果类的属性声明为public，则无法进行有效论证。在有继承层次的情况下，如果父类所有的rep都通过private进行保护，则无需进行额外论证，否则（即声明为protected）必须针对每个对父类rep状态进行修改的方法进行论证，证明不会导致父类对象的repOK为假。

*注：首先假设repOK的实现是正确的。在方法论证部分对repOK的正确性进行论证。*

(a)针对构造方法，论证初始态对象的repOK为真。

Queue没有构造方法，初始条件下n=0，此时repOK值为真

(b)逐个论证每个对象状态更改方法的执行都不会导致repOK的返回值为false。

insetfr方法只在存入一条指令后++，n只能增加，repOK为true

inseter是insetfr方法的ER指令版本，原因相同

(c)该类的其他方法的执行皆不改变对象状态，因此这些方法执行前和执行后的repOK都为true。

(d)综上，对该类任意对象的任意调用都不会改变其repOK为true的特性。因此该类任意对象始终保持对象有效性。

3.方法实现正确性论证

目标：在方法的前置条件和后置条件没有逻辑矛盾的前提下，根据前置条件和后置条件分析整理出完整的{<effect Y> with <input partition X>}，然后对照方法代码实现来论证给定每个<partition X>，方法执行结果都满足<effect Y>。

(a)**staticint** insetfr(**boolean**sign,String s0,String s1,String s2,String s3,String[][] Queue){

/\*

\* @Require:boolean sign,String s0,s1,s2,s3;

\* @Modified:Queue;

\* @Effects:if sign,getQueue()[n][0] == s0,getQueue()[n][1] == s1,getQueue()[n][2] == s2,getQueue()[n][3] == s3;\result==n;

\* \*/

根据上述过程规格，获得如下的划分：

<donothing>with<sign==false>

< getQueue()[n][0] == s0,getQueue()[n][1] == s1,getQueue()[n][2] == s2,getQueue()[n][3] == s3;\result==n;>with< sign == **true**>

* 方法检查目前sign是否为true,否则直接返回结束
* 方法sign为true时往Queue中插入指令，并且将n++
* 没有前置条件，所以操作符合前置条件

(b)**staticint**inseter(**boolean**sign,String s0,String s1,String s2,String[][] Queue){

/\*

\* @Require:boolean sign,String s0,s1,s2;

\* @Modified:Queue;

\* @Effects:if sign,getQueue()[n][0] == s0,getQueue()[n][1] == s1,getQueue()[n][2] == s2;\result==n;

\* \*/

根据上述过程规格，获得如下的划分：

<donothing>with<sign==false>

< getQueue()[n][0] == s0,getQueue()[n][1] == s1,getQueue()[n][2] == s2;\result==n;>with< sign == **true**>

* 方法检查目前sign是否为true,否则直接返回结束
* 方法sign为true时往Queue中插入指令，并且将n++
* 没有前置条件，所以操作符合前置条件

NewElevator

1.抽象对象得到了有效实现论证

论证类的overview明确了抽象对象，且类的rep能够通过抽象函数映射到相应的抽象对象。在有继承层次的情况下，要注意区分子类抽象函数与父类抽象函数的关系，并能够有效重用父类的相应论证结果。

Static对象waiting[][]存放能够这一次主请求下捎带的指令，可以查询删除添加

static对象 headingfloor在主请求时确定

static对象floor根据每次时间改变和运动方向改变，可以表达楼层

static对象direction表示电梯方向

以上对象均能起到其作用

2.对象有效性论证

论证在任意时刻，一个类的任意对象都能保证repOK为真，即任意一个方法的执行都不会导致rep为假。如果类的属性声明为public，则无法进行有效论证。在有继承层次的情况下，如果父类所有的rep都通过private进行保护，则无需进行额外论证，否则（即声明为protected）必须针对每个对父类rep状态进行修改的方法进行论证，证明不会导致父类对象的repOK为假。

*注：首先假设repOK的实现是正确的。在方法论证部分对repOK的正确性进行论证。*

(a)针对构造方法，论证初始态对象的repOK为真。

NewElevator没有构造方法，初始条件下floor=1，headingfloor=0,direction=NULL此时repOK值为真

(b)逐个论证每个对象状态更改方法的执行都不会导致repOK的返回值为false。

run方法只会执行到请求到达的楼层和途中楼层，请求楼层在1-10之内，headingfloor只会等于请求楼层， direction只有direction=UP direction=DOWN direction=STLL三种赋值语句，repOK为ture

(c)该类的其他方法的执行皆不改变对象状态，因此这些方法执行前和执行后的repOK都为true。

(d)综上，对该类任意对象的任意调用都不会改变其repOK为true的特性。因此该类任意对象始终保持对象有效性。

3.方法实现正确性论证

目标：在方法的前置条件和后置条件没有逻辑矛盾的前提下，根据前置条件和后置条件分析整理出完整的{<effect Y> with <input partition X>}，然后对照方法代码实现来论证给定每个<partition X>，方法执行结果都满足<effect Y>。

(a)**staticint** run(String[][] Queue){

/\*

\* @Require:String[][] Queue;

\* @Modified:Queue.queue,int floor,String diection,double headinglfloor,String[][] Waitinglist;

\* @Effects:move the elevator according to the requires;

if(FR&!running&UP) move the elevator up;

if(FR&!running&DOWN)move the elevator down;

if(FR&!running&STILL) stop the elevator down;

if(FR&running&UP&canpick)add the elevator list;

if(FR&running&UP&!canpick)do nothing;

if(FR&running&DOWN&canpick)add the elevator list;

if(FR&running&DOWN&!canpick)do nothing;

if(ER&!running&UP) move the elevator up;

if(ER&!running&DOWN)move the elevator down;

if(ER&!running&STILL) stop the elevator down;

if(ER&running&UP&canpick)add the elevator list;

if(ER&running&UP&!canpick)do nothing;

if(ER&running&DOWN&canpick)add the elevator list;

if(ER&running&DOWN&!canpick)do nothing;

if(DOWN&have require)floor--,stop the elevator;

if(UP&have require)floor++,stop the elevator;

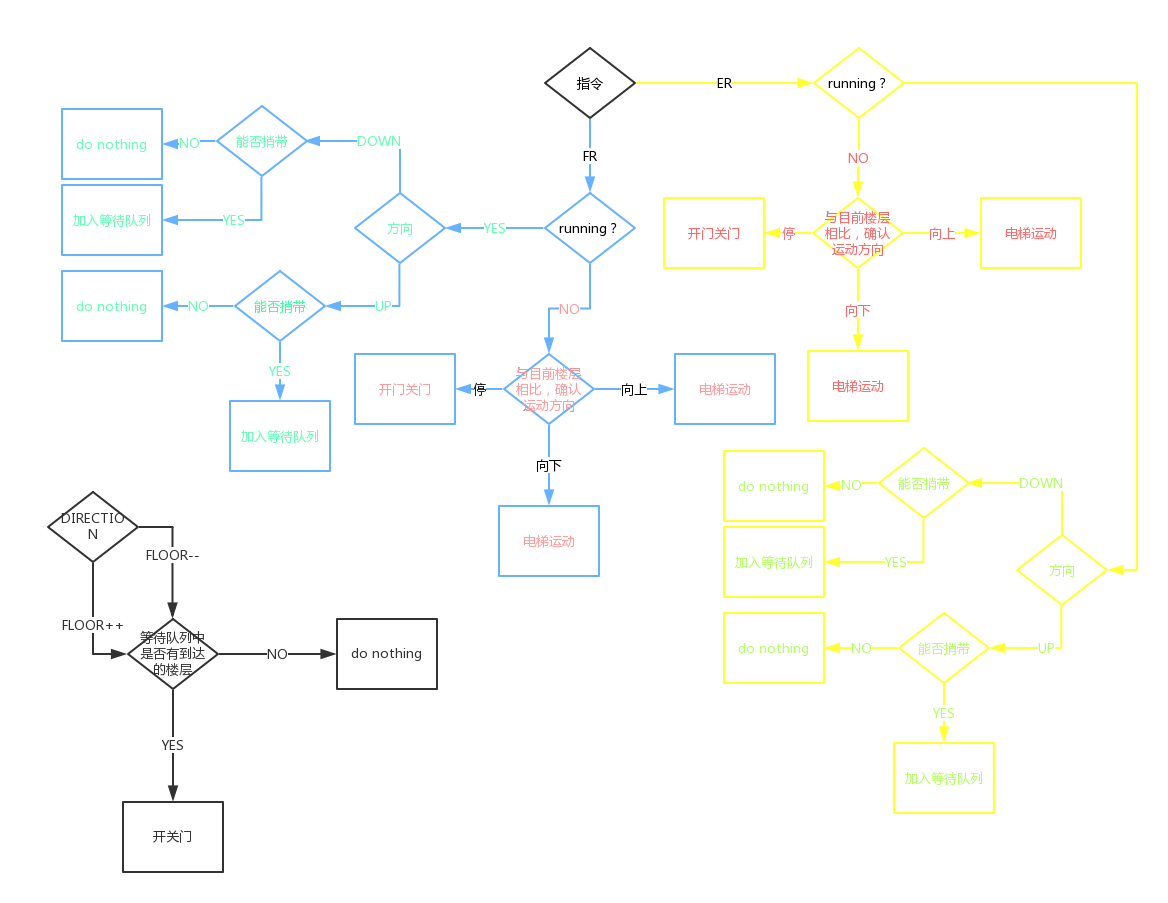
if(DOWN&!have require)floor--;

if(UP&!have require)floor++;

\* \*/

这个方法比较大，Effect很难用规格语言描述，底下我具体的讲一下

如下的划分：



蓝色和黄色代表两大分支FR和ER，红色绿区域表示FR ER中处理方式相同，只不过按照指令区别有微调

<move the elevator up>with<FR&!running&UP>

<move the elevator down>with<FR&!running&DOWN>

<stop the elevator down>with<FR&!running&STILL>

<add the elevator list>with<FR&running&UP&canpick>

<do nothing>with<FR&running&UP&!canpick>

<add the elevator list>with<FR&running&DOWN&canpick>

<do nothing>with<FR&running&DOWN&!canpick>

<move the elevator up>with<ER&!running&UP>

<move the elevator down>with<ER&!running&DOWN>

<stop the elevator down>with<ER&!running&STILL>

<add the elevator list>with<ER&running&UP&canpick>

<do nothing>with<ER&running&UP&!canpick>

<add the elevator list>with<ER&running&DOWN&canpick>

<do nothing>with<ER&running&DOWN&!canpick>

<floor--&stop>with<DOWN&haverequire>

<floor++&stop>with<UP&haverequire>

<floor++>with<UP&!haverequire>

<floor-->with<DOWN&!haverequire>

* 整体函数按照上述分支构造
* 指令为FR，没有在跑指令时按照上行下行停下选择主指令，在跑的时候根据方向判断此时能否捎带，捎带加入小队列
* 指令为ER，没有在跑指令时按照上行下行停下选择主指令，在跑的时候根据方向判断此时能否捎带，捎带加入小队列
* 每0.5s扫描队列，看是否有已经到达的请求
* 没有前置条件，所以操作符合前置条件