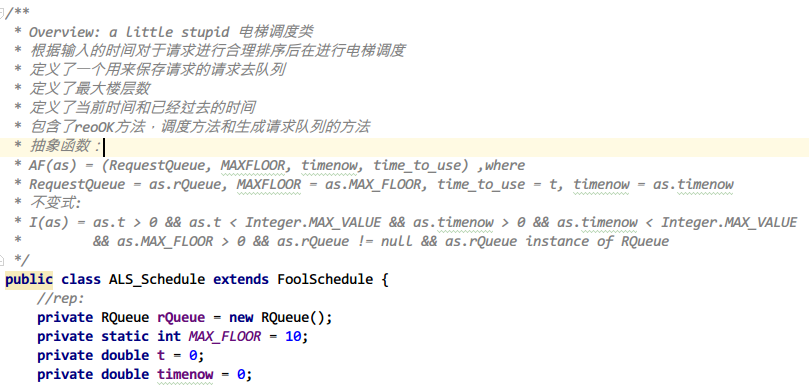
类实现正确性的论证

# ALS\_Schedule类

## 抽象对象得到了有效实现论证

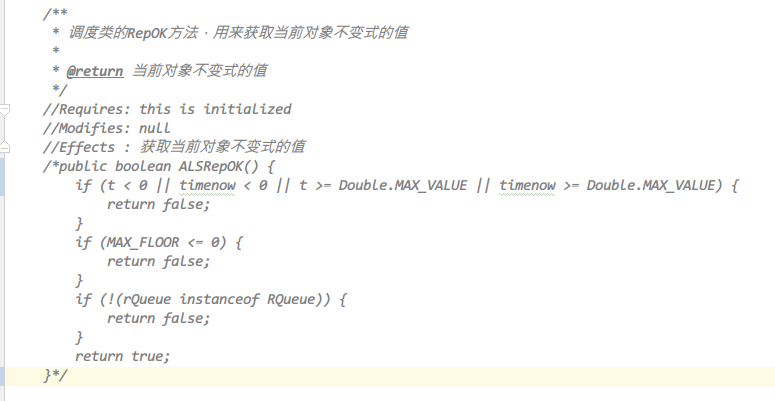
Overview+rep信息如下：



从图中可以看出，Overview明确了抽象对象，且从Overview中可以看出类的所有rep成员都在抽象函数中得以体现

## 对象有效性论证

ALS调度类的RepOK方法如下图所示：



接下来我将对于逐个方法论述其对于repOK方法的影响。

1. 构造方法

此类采用隐式构造方法，初始化将rQueue新建为RQueue类型的对象，Max\_FLOOR 为10，t 和timenow的初始值设为0，repOK返回值显然为true。

1. **public boolean** dispatch()

这个方法仅会改变rep中的t和timenow变量，由于两个变量的初始值为0，且方法中对于这两个变量仅有增加的操作，因此可以满足这两个变量永远大于0。Double变量的取值范围非常大，因此可以近似考虑为不可能超出上界，至此可以确保dispatch类不会对电梯的运行产生影响，即可以保证repOK永真。

1. **public boolean** bulidQueue()

这个方法会对rep中的rQueue产生影响，但是rQueue在构造时就新建为RQueue类型的对象，因此不变式中的(rQueue instanceof RQueue)将不会受到影响，即可以保证repOK永真。

## 方法实现正确性论证

1. dispatch()

//Requires: rQueue is built correctly  
//Modifies: t timenow  
//Effects : 对电梯进行A Little Stupid调度

根据上述过程规格，获得如下的划分：

<do nothing> with <rQueue is empty or not corrently initialized>

<pick a request from rQueue and dispatch the elevator to finish the primary request and update the follow queue> with <rQueue is not empty and available>

<change the primary rQueue> with <the elder primary request is finished and the carrying order is not finished, then the first one in carrying order becomes the new carrying request>

方法由循环开始，结束条件是**rQueue**.getRindex() != 0 || follow.getRindex() != 0。因此，当输入的队列为空的时候，直接返回结束，即满足<do nothing> with <rQueue is empty or not corrently initialized>

当循环过程中，调度会根据电梯的运动状态完成。如果follow为空，则将会把相应的元素放进去作为carry请求，在建立玩请求的执行队列之后，调度器会根据当前的primary方法跟新电梯的运动状态，同时会根据电梯的当天状态来判断follow队列中是否存在已经完成的请求，如果存在，则会将其从follow队列中清除出来。至此可以看出第二个划分为真

当primary完成且follow队列不为空时，follow队列中的第一个请求将会成为新的primary队列，并继续循环完成电梯调度，因此第三个划分也为真。

至此，所有三个划分都可论证为真。

1. buildQueue()

//Requires: 控制台输入

//Modifies: rQueue bf

//Effects : 读入请求并将其放入请求队列

根据上述过程规格，获得如下的划分：

<do nothing> with <no input in console or input is incorrect>

<add new request to rQueue> with <right input format and rQueue is not full>

方法主要负责读取控制台输入，并通过格式匹配来进行指令的添加。当没有输入或者输入无效的时候，将不会有新的指令添加到rQueue中，因此<do nothing> with <no input in console or input is incorrect>成立。

当有符合格式的指令通过控制台添加到rQueue中的时候，首先应该确保rQueue未满，之后即可享rQueue中添加新的request，即第二个划分成立。

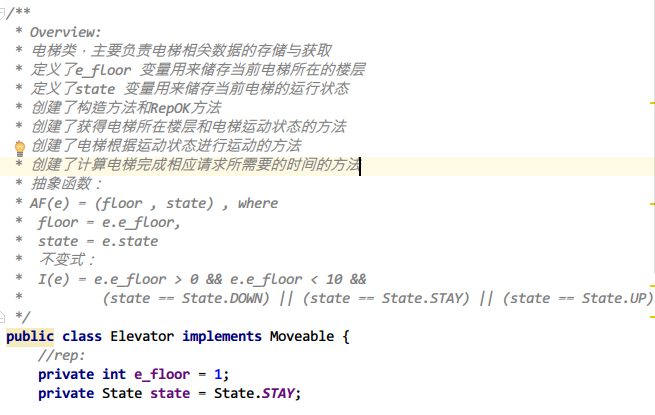
综合上述两点，我们可以看出该方法正确实现了Effect中的内容。

综上所述，所有方法的实现都满足规格。从而可以推断，ALS\_Schedule的实现是正确的，即满足其规格要求。

# Elevator类

## 抽象对象得到了有效实现论证

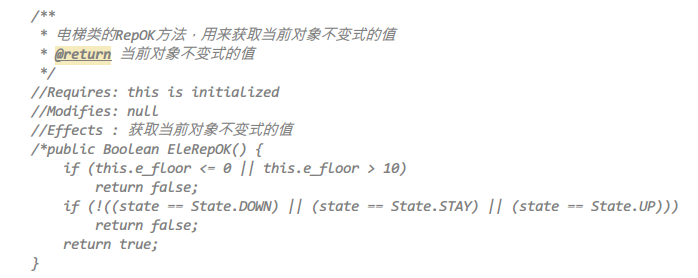
Overview信息如下：



从图中可以看出，Overview明确了抽象对象，且从Overview中可以看出类的所有rep成员都在抽象函数中得以体现，说明抽象对象得到了有效的实现

## 对象有效性论证

Elevator类的repOK方法如下：



接下来我将对于逐个方法论述其对于repOK方法的影响。

1. 构造方法

此类采用显式构造方法，初始化将e\_floor 设置为1，并将this.state设置为STAY。这显然可以使repOK返回true。

1. **public boolean move**(State st)

这个方法会改变电梯对象的state属性和e\_floor属性，根据输入的state的不同会采取相应的楼层加减工作。当e\_floor小于等于0的时候会返回false，说明操作不合法。因此在合法范围内的所有操作均可以保证满足repOK为true。

1. **public double timeSpent**(Request rq)

这个方法会对电梯对象的e\_floor进行更改，根据输入的rq的不同，方法会计算电梯移动到该层数所需要话费的时间，进而改变自己所处的层数。此方法只用于傻瓜调度。因为Request的floor都是合法的，因此e\_floor = destination之后仍然合法，即可以满足repOK为true。

该类的其他方法由于不涉及修改rep变量，因此不会对repOK的真伪产生影响，在此不做赘述。

## 方法实现正确性论证

1. Move(State st)

//Requires: st!= null st类型为State

//Modifies: this.e\_floor this.state

//Effects : 电梯的运动方法，实现自Moveable接口,

// 主要用来根据电梯当前的运动状态实现楼层的转移和运动状态的变化

// 正确完成返回true，否则返回false

根据上述规格，我们可以得到以下划分：

<do nothing> with <incorrect input>

<modify this.e\_floor and this.state，and return true> with <right input>

<modify this.e\_floor and this.state, but return false> with <illegal floor>

当没有进行正确输入的时候，方法不能够正常执行，自然不会有返回值

当输入正确，且楼层数正确的时候，更改楼层和状态属性，返回真值

当楼层<0时，说明操作非法，此时返回false,说明操作未能够正常完成

至此，说明此方法和Effects中的论述一致。

1. toString()

//Requires: this is initialized

//Modifies: null

//Effects : 获取当前电梯状态

根据上述的规格，可以看出这个类就是简单地获取电梯状态用的方法。其中返回值为一个带有电梯当前楼层和属性的字符串。因为此程序为单线程操作，因此不会出现信息错误，此方法显然和Effects中的论述一致。

1. getE\_floor()

//Requires: null

//Modifies: null

//Effects : 获得电梯当前的楼层

根据代码，我们可以看出这个方法返回的是e\_floor的值，显然和Effects中的论述一致

1. getState()

//Requires: null

//Modifies: null

//Effects : 获得电梯当前的运行状态

根据代码，我们可以看出这个方法返回的是state的值，显然和Effects中论述的一致。

1. timeSpent()

//Requires: 需要完成的请求rq

//Modifies: timeneed

//Effects : 完成请求所需要的时间，成功则返回true

根据上述规格，我们可以得到以下划分：

<do nothing> with <no input or wrong input>

<calculate the time needed to get the destination and return true> with <rq which is correctly creaeted>

当输入不合法或者没有输入时，方法无法执行，显然do nothing

当输入的rq是经过正确的方法创建的时候，由于Request类性质的保证，该方法总是能够正常的执行，因此会返回true。至此，说明此方法跟Effects中的论述一样

综上所述，所有方法的实现都满足规格。从而可以推断，Elevator的实现是正确的，即满足其规格要求。

# RQueue类

## 抽象对象得到了有效实现论证

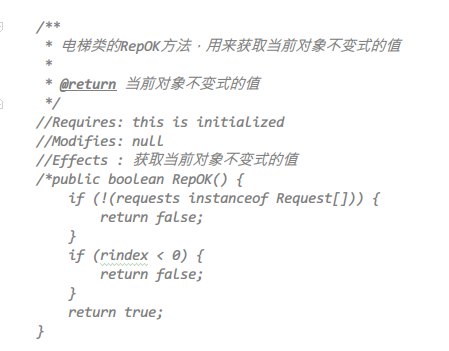
Overview信息如下图：



从图中可以看出，Overview明确了抽象对象，且从Overview中可以看出类的所有rep成员都在抽象函数中得以体现，说明抽象对象得到了有效的实现

## 对象有效性论证

RQueue类的repOK方法如下：



接下来我将对于逐个方法论述其对于repOK方法的影响。

1. 构造方法

此类采用隐式构造方法，初始化将request初始化为一个新的Request数组，并将rindex初始化为0。这显然可以使repOK返回true。

1. **public boolean addReques(Request request**)

这个方法会同时改变请求队列的requests和rindex两个属性。当rindex<100000时，会成功将传入的request添加到requests队列的队尾，并返回true；当操作不合法时，将会返回false，说明执行了错误操作。因此，在进行正确操作的时候，总是能够保证repOK返回true。

1. **public double timeSpent**(int index)

这个方法同样会同时改变请求队列的requests和rindex两个属性。当index > rindex的时候，会抛出异常，说明操作不合法。当输入的index为合法数字的时候，该方法将会删除队列中的第i个请求，并将它之后的请求都向前移动一个序号，最终染回true。因此，在能够正确执行的前提下，总能保证repOK返回true。

1. **public Boolean setQueue(int index)**

这个方法和setDone方法较为类似，这个方法仅仅会改变请求队列的requests属性。当index > rindex时，说明队列中不存在该请求，此时将会返回false，说明操作不成功。当输入合法是，将会改变requests中第i个元素的属性，但是这并不影响requests自身的属性，因此可以保证repOK返回true。

1. **public Boolean setDone(int index)**

这个方法仅仅会改变请求队列的requests属性。当index > rindex时，说明队列中不存在该请求，此时将会返回false，说明操作不成功。当输入合法是，将会改变requests中第i个元素的属性，但是这并不影响requests自身的属性，因此可以保证repOK返回true。

该类的其他方法由于不涉及修改rep变量，因此不会对repOK的真伪产生影响，在此不做赘述。

## 方法实现正确性论证

1. getRequest(int i)

//Requires: 需要取出的请求的序号i  
//Modifies: null  
//Effects : 获取第i个队列用的方法

简单地返回队列中第i个请求的方法，只有一个return，不加以赘述。

1. addRequest(Request request)

//Requires: 需要添加的请求request

//Modifies: request rindex

//Effects : 向队列中添加请求request，成功添加返回true，否则返回false

根据过程规格，我们可以得到如下划分：

<do nothing> with <no correct input >

<add request to requests and return true> with <right input and requests is not full>

<not input and return false> with <requests is full>

当没有输入，或者输入错误的时候，方法不能够正常执行，显然不会有任何输出

当输入的请求合法，并且队列未满的时候，将此请求加入到队列之中

当队列已经满了时，不执行任何操作并直接返回false。

因此我们可以说，此方法的实现和Effects中的叙述一致。

1. delRequest(int index)

//Requires: 需要删除的请求序号

//Modifies: rindex request

//Effects : 删除请求队列中第index个请求,成功删除返回true

根据过程规格，我们可以做出如下划分：

<do nothing > with <no input>

<del request from the requests and return true> with <right index input>

<throw Exception> with < index bigger than rindex>

当没有输入或者输出错误的时候，显然方法不能够成长执行，do nothing

当输入合法，并且队列中的请求可以被删除的时候，执行删除操作，并返回true

当输入的index不合法的时候，返回false；

因此我们可以说，此方法的实现和Effects中的叙述一致。

1. getRindex()

//Requires: 需要取出的请求的序号i

//Modifies: null

//Effects : 获取第i个队列用的方法

一个简单的return方法，在此不再赘述。

1. setDone(int index)

//Requires: 需要设置的请求的编号

//Modifies: request[index]

//Effects : 将request中的第index个请求设置为已完成,成功完成返回true，否则返回false

根据过程规格，我们可以得到以下划分：

<do nothing > with <no correct input>

<set requests[index].done to be true, and return true> with <correct input>

<return false> with <index > rindex>

当没有正确输入的时候，方法将不能够正确执行，此时显然do nothing

当输入合法的时候，能够执行set操作，这时候执行操作并返回true

当输入不合法的时候，由于队列中不存在第index个元素，因此将返回false，并且不执行任何操作

至此可以看出，该类的相关操作符合其过程规格。

1. setQueue(int index)

//Requires: 需要设置的请求的编号

//Modifies: request[index]

//Effects : 将request中的第index个请求设置为已入队

根据上述过程规格，我们可以得到如下划分

<do nothing> with <no correct input>

<set requests[index].queue to be true, and return true> with <correct input>

<return false> with <index > rindex>

当没有正确输入的时候，方法将不能够正确执行，此时显然do nothing

当输入合法的时候，能够执行set操作，这时候执行操作并返回true

当输入不合法的时候，由于队列中不存在第index个元素，因此将返回false，并且不执行任何操作

至此可以看出，该类的相关操作符合其过程规格。

1. info()

//Requires: null

//Modifies: null

//Effects : 打印队列中的请求的详细信息

从规格中我们可以看出这是一个比较简单的方法，直接打印当前队列中的请求，但是并不对对象的属性进行直接的修改，因此可以保证此方法总能够正常执行。

综上所述，所有方法的实现都满足规格。从而可以推断，Elevator的实现是正确的，即满足其规格要求。