类实现正确性的论证

一、 ALS_Schedule 类

1. 抽象对象得到了有效实现论证

Overview+rep 信息如下：

```
/**
 * Overview: a little stupid 电梯调度类
 * 根据输入的时间对于请求进行合理排序后在进行电梯调度
 * 定义了一个用来保存请求的请求去队列
 * 定义了最大楼层数
 * 定义了当前时间和已经过去的时间
 * 包含了reoOK方法，调度方法和生成请求队列的方法
 * 抽象函数：
 * AF(as) = (RequestQueue, MAXFLOOR, timenow, time_to_use) ,where
 * RequestQueue = as.rQueue, MAXFLOOR = as.MAX_FLOOR, time_to_use = t, timenow = as.timenow
 * 不变式:
 * I(as) = as.t > 0 && as.t < Integer.MAX_VALUE && as.timenow > 0 && as.timenow < Integer.MAX_VALUE
 *         && as.MAX_FLOOR > 0 && as.rQueue != null && as.rQueue instance of RQueue
 */
public class ALS_Schedule extends FoolSchedule {
    //rep:
    private RQueue rQueue = new RQueue();
    private static int MAX_FLOOR = 10;
    private double t = 0;
    private double timenow = 0;
```

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

2. 对象有效性论证

ALS 调度类的 RepOK 方法如下图所示：

```
    /**
     * 调度类的RepOK方法，用来获取当前对象不变式的值
     *
     * @return 当前对象不变式的值
     */
    //Requires: this is initialized
    //Modifies: null
    //Effects : 获取当前对象不变式的值
    /*public boolean ALSRepOK() {
        if (t < 0 || timenow < 0 || t >= Double.MAX_VALUE || timenow >= Double.MAX_VALUE) {
            return false;
        }
        if (MAX_FLOOR <= 0) {
            return false;
        }
        if (!(rQueue instanceof RQueue)) {
            return false;
        }
        return true;
    }*/
```

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。

2）`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。

3）`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。

3. 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。

2） 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。

二、 Elevator 类

1. 抽象对象得到了有效实现论证

Overview 信息如下：

```
/**
 * Overview:
 * 电梯类，主要负责电梯相关数据的存储与获取
 * 定义了e_floor 变量用来储存当前电梯所在的楼层
 * 定义了state 变量用来储存当前电梯的运行状态
 * 创建了构造方法和RepOK方法
 * 创建了获得电梯所在楼层和电梯运动状态的方法
 * 创建了电梯根据运动状态进行运动的方法
 * 创建了计算电梯完成相应请求所需要的时间的方法
 * 抽象函数：
 * AF(e) = (floor , state) , where
 *  floor = e.e_floor,
 *  state = e.state
 *  不变式：
 *  I(e) = e.e_floor > 0 && e.e_floor < 10 &&
 *         (state == State.DOWN) || (state == State.STAY) || (state == State.UP)
 */
public class Elevator implements Moveable {
    //rep:
    private int e_floor = 1;
    private State state = State.STAY;
```

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

2. 对象有效性论证

Elevator 类的 repOK 方法如下：

```
/**
 * 电梯类的RepOK方法，用来获取当前对象不变式的值
 * @return 当前对象不变式的值
 */
//Requires: this is initialized
//Modifies: null
//Effects : 获取当前对象不变式的值
/*public Boolean EleRepOK() {
    if (this.e_floor <= 0 || this.e_floor > 10)
        return false;
    if (!((state == State.DOWN) || (state == State.STAY) || (state == State.UP)))
        return false;
    return true;
}
```

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。

2）**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。

3）**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。

3. 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。

2) 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。

3) 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

4) 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。

5) 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。

三、 RQueue 类

1. 抽象对象得到了有效实现论证

Overview 信息如下图：

```
/**
 * 请求队列类
 * 主要用于存储请求及完成请求队列的相关操作
 * 定义了一个Request类型的数组来保存请求
 * 定义了rindex来作为当前队列的指针
 * 包括ReqpOK方法
 * 包括从请求队列中获得、添加、删除请求的方法
 * 包括获得请求队列请求个数的方法
 * 包括设置请求done和inQueue属性的方法
 * 包括显示请求队列信息的方法
 * 抽象函数：
 * AF(rq) = (requests[], rindex), where
 * request = rq.requests[],
 * rindex = rq.rindex
 * 不变式：
 * I(rq) = rindex >=0 && request instance of Request[]
 */
public class RQueue {
    //rep:
    private Request[] requests = new Request[100000];
    private int rindex = 0;
```

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

2. 对象有效性论证

RQueue 类的 repOK 方法如下：

```
    /**
     * 电梯类的RepOK方法，用来获取当前对象不变式的值
     *
     * @return 当前对象不变式的值
     */
    //Requires: this is initialized
    //Modifies: null
    //Effects : 获取当前对象不变式的值
    /*public boolean RepOK() {
        if (!(requests instanceof Request[])) {
            return false;
        }
        if (rindex < 0) {
            return false;
        }
        return true;
    }
```

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。

2）**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。

3）**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。

4）**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。

5）**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。

3. 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。

2) 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。

3) 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。

4) 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。

5) 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。

6) 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。

7) 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。