# 类实现正确性的论证

写在前面：

- get 和 set 以及功能类似的方法根据 Require 显然正确。
- 感谢同学花费时间看这个蛋疼的文档，考试加油~~~

一、Elevator

1. 抽象对象得到了有效实现论证

   抽象函数为：AF(c) = (direction, time, floor), where direction = c.direction, time = c.time_now, floor = c,floor_now

   在类中可以发现定义了以下变量：

   ```
   private String direction;
   private double time_now;
   private int floor_now;

   private static final String UP = "UP";
   private static final String DOWN = "DOWN";
   ```

   可以发现对象类型能够表示抽象对象的所有可能取值并且对象能够存储抽象对象所有可能的数据量。

2. 对象有效性论证

   a) 由于在构造函数中，初始化了以下变量

   direction = ***UP***;

   time_now = 0;

   floor_now = 1;

   repOK 的运行结果显然返回 true。

   b) 在 runElevator 和 runElevator_first 方法中，direction 只能被改变为上面的两个静态变量中的一种，time_now 只能增加，floor_now 只能为 1-10 之间的整数，所以无论如何是满足 rep 的。

   其他方法的 Requires 保证了变量的正确赋值。

   综上：所有方法都不会导致不变式为假

3. 方法实现的正确性论证

   set 与 get 方法根据其 Requires 显然满足正确性。

   在 runElevator 和 runElevator_first 方法中。

   根据过程规格，获得如下的划分：

   <do running the elevator> with <target_floor>

   方法根据传入的目标楼层，根据电梯的当前楼层运行电梯，改变方向，增加时间，所以 time_now 只能增加，floor_now 只能为 1-10 之间的整数, direction 只能被改变为上面的两个静态变量中的一种。满足 repOK。

   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。

二、RequestQueue

1. 抽象对象得到了有效实现论证

   抽象函数为：AF(c) = (request queue, latest_time), where request queue = c.request,

latest_time = c.latest_time

在类中可以发现定义了以下变量：

```
private LinkedList<Request> request = new LinkedList<Request>();
private int latest_time;
private static final String UP = "UP";
private static final String DOWN = "DOWN";
```

可以发现对象类型能够表示抽象对象的所有可能取值并且对象能够存储抽象对象所有可能的数据量。

2. 对象有效性论证

在定义此类时，初始化了静态变量 LinkedList<Request> request = **new** LinkedList<Request>();

所以 request 不可能为 null。

a) 由于在构造函数中，初始化了以下变量

latest_time = 0;

repOK 的运行结果显然返回 true。

b) 在 addQueue 方法中。

假设 public void addQueue (String str) throws ElevatorException 运行前 repOK 为真。

该方法根据输入的字符串，判断是否合法，如果合法，则将其转为一个请求加入请求队列中，并修改最新时间 latest_time,否则根据错误类型报出错误提示。

由于对于 LinkedList<Request> request 的修改只可能存在增加一个元素，而 latest_time 修改前必须小于修改后，所以 latest_time 必定大于等于 0。

所以执行完之后必定满足 repOK。

c) 在 mergeRequest 方法中。

假设 public void mergeRequest() 运行前 repOK 为真。

该方法遍历 LinkedList<Request> request 中的元素，将重复的删除。

由于对于 LinkedList<Request> request 的修改只可能存在改变其中元素。LinkedList<Request> request 不可能变成 null。

所以执行完之后必定满足 repOK。

d) 其他的方法为单纯的队列操作，由于对于 LinkedList<Request> request 的修改只可能存在改变其中元素。LinkedList<Request> request 不可能变成 null。所以执行完之后必定满足 repOK。

综上：所有方法都不会导致不变式为假

3. 方法实现的正确性论证

队列操作方法根据其代码显然满足正确性。

a) 在 addQueue 方法中，

根据过程规格，获得如下的划分：

<do nothing> with <wrong str>

<do add a request into the RequestQueue> with <str>

如果输入的 str 不是标准的格式，方法会根据错误类型报出错误，否则加入队列。

b) 在 mergeRequest 方法中

根据过程规格，获得如下的划分：

<do merge the same request> with <nothing>

由于对于 LinkedList<Request> request 的修改只可能存在改变其中元素。LinkedList<Request> request 不可能变成 null。

所以执行完之后必定满足 repOK。

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

三、Schedule

1. 抽象对象得到了有效实现论证

抽象函数为：AF(c) = (elevator, requestQueue), where elevator = c.elevator, requestQueue = c.requestQueue

在类中可以发现定义了以下变量：

**protected Elevator elevator;**

**protected RequestQueue requestQueue;**

可以发现对象类型能够表示抽象对象的所有可能取值并且对象能够存储抽象对象所有可能的数据量。

2. 对象有效性论证

a) 由于在构造函数中，初始化了以下变量

elevator = **new** Elevator();

requestQueue = **new** RequestQueue();

repOK 的运行结果显然返回 true。

b) 在 addQueue 方法中。

假设 public void addQueue (String str) throws ElevatorException 运行前 repOK 为真。

该方法仅仅调用 requestQueue 类中的方法，根据之前的论证，显然正确。

所以执行完之后必定满足 repOK。

c) 在 sche 方法中。

假设 public void sche(Request request) 运行前 repOK 为真。

该方法根据传入的 request 类型调用 elevator 类中的方法进行调度。

均不能将定义的变量变成 null。

所以执行完之后必定满足 repOK。

d) 在 startSchedule 方法中。

假设 public void startSchedule() 运行前 repOK 为真。

该方法取出 requestQueue 一个 request 用 sche 进行调度。根据前面的证明，均不能将定义的变量变成 null。

所以执行完之后必定满足 repOK。

综上：所有方法都不会导致不变式为假

3. 方法实现的正确性论证

队列操作方法根据其代码显然满足正确性。

a) 在 addQueue 方法中，

根据过程规格，获得如下的划分：

<do use requestQueue.addQueue(str)> with <str>

代码为 requestQueue.addQueue(str);

显然正确

b) 在 sche 方法中

根据过程规格，获得如下的划分：

<do schedule the elevator> with <request>

该方法根据传入的 request 类型调用 elevator 类中的方法进行调度。

均不能将定义的变量变成 null。

所以执行完之后必定满足 repOK。

c) 在 startSchedule 方法中。

根据过程规格，获得如下的划分：

<do schedule the elevator> with <nothing>

该方法取出 requestQueue 一个 request 用 sche 进行调度。根据前面的证明，均不能将定义的变量变成 null。

所以执行完之后必定满足 repOK。

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

四、ASL_Schedule **extends** Schedule

1. 抽象对象得到了有效实现论证

抽象函数为：AF(c) = (elevator, requestQueue, mainRequest), where elevator = c.elevator, requestQueue = c.requestQueue  mainRequest = c.mainRequest

在类中可以发现**增加**定义了以下变量：

```
private Request mainRequest;
private boolean hasmain;
boolean firstrun;
private static final String FR = "FR";
private static final String ER = "ER";
private static final String UP = "UP";
private static final String DOWN = "DOWN";
private static final String STOP = "STOP";
```

可以发现对象类型能够表示抽象对象的所有可能取值并且对象能够存储抽象对象所有可能的数据量。

2. 对象有效性论证

下面指针对 Override 的方法进行论证。

a) 在 sche 方法中。

假设 public void sche(Request request) 运行前 repOK 为真。

该方法根据 firstrun 的值调用 elevator 类中的方法进行调度。

均不能将定义的变量变成 null。

所以执行完之后必定满足 repOK。

b) 在 startSchedule 方法中。

假设 public void startSchedule() 运行前 repOK 为真。

该方法根据 requestQueue 中的请求调用 sche 方法进行调度，根据 a 的论证均不能将 repOK 中需求的变量变成 null。

所以执行完之后必定满足 repOK。

综上：所有方法都不会导致不变式为假

3. 方法实现的正确性论证

下面指针对 Override 的方法进行论证。

a) 在 sche 方法中

根据过程规格，获得如下的划分：

<do schedule the elevator> with <request>

该方法根据 firstrun 的值调用 elevator 类中的方法进行调度。均不能将定义的变量变成 null。

所以执行完之后必定满足 repOK。

b) 在 startSchedule 方法中。

根据过程规格，获得如下的划分：

<do schedule the elevator> with <nothing>

该方法根据 requestQueue 中的请求，判断请求的输入是否符合时间要求

```
if (requestQueue.getRequest(0).getRequest_time() != 0) {
        requestQueue.clean();
        System.out.println("初始时间必须为 0。");
        return;
    }
```

接下来模拟电梯的运行来设置主请求和判断是否有需要捎带的请求。然后进行电梯的调度

所以执行完之后必定满足 repOK。

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