类实现正确性的吐血论证

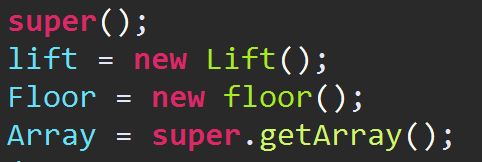
AlsScheduler

1. 抽象

ALSscheduler this lift can pick up requests on its way to the destination.

描述了类的功能，实现可以进行捎带请求的电梯调度

1. 对象有效性
   1. 初始化



后所有的对象均不为空指针，repOK成立

* 1. 假设deal（）开始运行时repOK为true

在deal函数中，首先检查是否存在更远的捎带，决定接下来将那一条指令设置为主指令。若不存在更远的捎带则将当前请求队列的首指令设为电梯的主指令。若存在则将那个更远的指令设为主指令。此时修改Array并不会导致Array中出现异常的指令，也不会致使其变成空指针。然后检查当前主指令是否有效，若无效则返回获取主指令，有效则继续运行。接下来刷新电梯内部记录的信息，包括出发位置，目的地位置，预计到达时间，方向。然后电梯进入逐层运行，每一层刷新时间，检查新的指令并加入指令队列。若当前楼层存在可捎带指令，则输出相应信息，更新当前系统时间，读取新指令，并更新电梯预计到达时间。读取lift的数据，并不会在类外部进行修改，所以保证liift有效，且不会为空指针。修改Array并不会导致Array中出现异常的指令，也不会致使其变成空指针。电梯到达目的楼层后检查当前楼层是否存在捎带指令，若当前楼层存在可捎带指令，则输出相应信息，更新当前系统时间，读取新指令。修改Array并不会导致Array中出现异常的指令，也不会致使其变成空指针。输出主指令执行结果，然后判断是否存在更远的捎带请求，更改furO，提示下一个主指令应为furtherOrder保存的特定指令。

因此该方法的执行不会导致repOK为假，不违背表示不变式

* 1. 假设hasFurther（）开始运行时repOK为true

在hasFurther函数中我们从lift.getTerminal()开始，根据电梯当前运行方向依次检查所有楼层，是否存在可捎带的指令。若存在则返回该指令，否则返回null，过程中并未修改类中的内容，只是访问并判断。

因此该方法的执行不会导致repOK为假，不违背表示不变式

* 1. 假设check（）开始运行时repOK为true

检查从当前时间，止到指定时间点t的指令。判断其是否有效，若为ER类型，则和电梯内保存的指令比较，若为FR则和楼内各层保存的指令比较。出现重复，则输出SAME并标记该指令无效。判断过程中只会修改楼层和电梯内按钮的信息，且加入的数据类型均为Order，不会导致Floor或lift指针变为null。

因此该方法的执行不会导致repOK为假，不违背表示不变式

1. 方法有效性
   1. Deal

/\*

@REQUIRES: None

@MODIFIES: \this.lift, \this.Floor, \this.Array, System.out

@EFFECTS: \all i; 0<i<Array.size;i++:

!Array(i).isEnable ==> continue;

futO=true ==> Array(i)=furtherOrder

Array(i).isEnable&&Array(i).getTime<=lift.getTime+10 ==> lift.StatusRefreash(Array(i).getTarget,Array(i).getTime)

Array(i).isEnable&&Array(i).getTime>lift.getTime+10 ==> lift.StatusRefreash(Array(i).getTarget,lift.getTime)

\all j in lift.position to lift.target:

lift.pickup(j) ==> System.out("("+n+","+lift.toString()+","+((TA+10+5\*t)/10)+"."+((TA+10+5\*t)%10)+")\n"&&check(t+2)

floor.pickup(j) ==> System.out("("+n+","+lift.toString()+","+((TA+10+5\*t)/10)+"."+((TA+10+5\*t)%10)+")\n")&&check(t+2)

lift.tostring!=Still&&lift.pickup(j) ==> System.out("("+n+","+lift.toString()+","+((TA+10+5\*t)/10)+"."+((TA+10+5\*t)%10)+")\n"&&check(t+2)

lift.tostring!=Still&&floor.pickup(j) ==> System.out("("+n+","+lift.toString()+","+((TA+10+5\*t)/10)+"."+((TA+10+5\*t)%10)+")\n")&&check(t+2)

lift.tostring==Still==> System.out("["+tmp.getReq()+"]/("+lift.getTerminal()+","+lift.toString()+","+(lift.getTime()/10+1)+"."+lift.getTime()%10+")");

futO=false

furtherOrder = this.hasfurther()!=null ==> futO=true

\*/

方法先检查是否存在更远的捎带，决定接下来将那一条指令设置为主指令。若不存在更远的捎带则将当前请求队列的首指令设为电梯的主指令。若存在则将那个更远的指令设为主指令。

然后检查当前主指令是否有效，若无效则返回获取主指令，有效则继续运行。接下来刷新电梯内部记录的信息，包括出发位置，目的地位置，预计到达时间，方向。然后电梯进入逐层运行，每一层刷新时间，检查新的指令并加入指令队列。若当前楼层存在可捎带指令，则输出相应信息，更新当前系统时间，读取新指令，并更新电梯预计到达时间。读取lift的数据，并不会在类外部进行修改，所以保证liift有效，且不会为空指针。修改Array并不会导致Array中出现异常的指令，也不会致使其变成空指针。电梯到达目的楼层后检查当前楼层是否存在捎带指令，若当前楼层存在可捎带指令，则输出相应信息，更新当前系统时间，读取新指令。修改Array并不会导致Array中出现异常的指令，也不会致使其变成空指针。输出主指令执行结果，然后判断是否存在更远的捎带请求，更改furO，提示下一个主指令应为furtherOrder保存的特定指令。

运行结果全部为指令队列为空，输出运行结果。

* 1. hasfurther()

/\*

@REQUIRES:None

@MODIFIES:None

@EFFECTS: lift.toString()==UP&&\any i=lift.getTerminal();i<11;i++:lift.isfloor(i)&&i>lift.getTerminal()==>\result = lift.getOatF(i)

lift.toString()==DOWN&&\any i=lift.getTerminal();i>0;i--:lift.isfloor(i)&&i<lift.getTerminal()==>\result = = lift.getOatF(i)

\*/

有以下分支

\result = lift.getOatF(i) with this.lift.toString().equals("UP") && lift.isfloor(i) && i>lift.getTerminal()

\result = lift.getOatF(i) with this.lift.toString().equals("DOWN") && lift.isfloor(i) && i<lift.getTerminal()

经方法测试，当电梯运行方向为向上时，当前电梯主指令目标楼层为3，5层有请求，则返回5层的指令，满足后置条件，满足\result = lift.getOatF(i) with this.lift.toString().equals("UP") && lift.isfloor(i) && i>lift.getTerminal()

经方法测试，当电梯运行方向为向下时，当前电梯主指令目标楼层为6，2层有请求，则返回5层的指令，满足后置条件，满足\result = lift.getOatF(i) with this.lift.toString().equals("DOWN") && lift.isfloor(i) && i<lift.getTerminal()

* 1. check(long t)

/\*

@REQUIRES:None

@MODIFIES:\this.Array

@EFFECTS: \all tmp2.getTime()<=t;j++:

Array(j).getType().equals("FR")&&!Floor.judge(tmp2)==>System.out.println("SAME["+tmp2.getReq()+"]");

Array(j).getType().equals("ER")&&!lift.judge(tmp2)==>System.out.println("SAME["+tmp2.getReq()+"]");

j++

\*/

有以下分支

System.out.println("SAME["+tmp2.getReq()+"]") with tmp2.getType().equals("FR") && !Floor.judge(tmp2)

System.out.println("SAME["+tmp2.getReq()+"]") with tmp2.getType().equals("ER") && !lift.judge(tmp2)

Do nothing with tmp2.getType().equals("ER") && lift.judge(tmp2)

Do nothing with tmp2.getType().equals("FR") && Floor.judge(tmp2)

通过方法测试，（ER,7,3）(ER,7,4),报same，满足后置条件，满足System.out.println("SAME["+tmp2.getReq()+"]") with tmp2.getType().equals("ER") && !lift.judge(tmp2)

通过方法测试，(FR,5,UP,3)(FR,5,UP,4),报错same，满足后置条件，满足System.out.println("SAME["+tmp2.getReq()+"]") with tmp2.getType().equals("FR") && !Floor.judge(tmp2)

通过方法测试，(FR,5,DOWN,3)，正常返回，满足后置条件，满足Do nothing with tmp2.getType().equals("FR") && Floor.judge(tmp2)

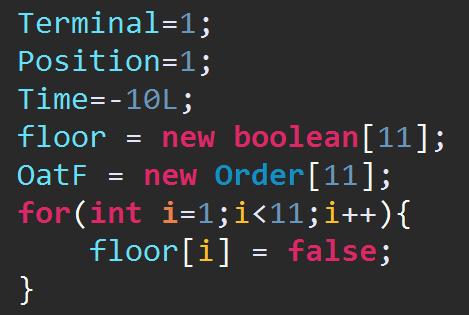
通过方法测试，(ER,5,3)，正常返回，满足后置条件，满足Do nothing with tmp2.getType().equals("ER") && lift.judge(tmp2)

Lift

1. 抽象

电梯类，实现电梯信息的存储与获取

1. 对象有效性
   1. 初始化时



所以保证repOK为true

* 1. 假设setFloor(Order tmp)开始运行时repOK为true

设置指定楼层的按钮点亮以及保存相应指令，建立指令时已经保证了有效性，所以不会影响有效性

方法执行结束，因此该方法执行不会导致repOK为假。

* 1. 假设setFloor(int n) 开始运行时repOK为true

将指定楼层的按钮设置为熄灭

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设judge(Order tmp) 开始运行时repOK为true

判断获取的新指令所在楼层是否已经存在指令，若存在，则新指令无效，若不存在，将新指令存入

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设pickup(int n) 开始运行时repOK为true

如果楼层为1或10不允许捎带，其他楼层则访问按钮是否点亮，若已经点亮则输出指令信息，并将按钮状态设置为false。

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设getPosition()开始运行时repOK为true

获取this.Position，不修改信息

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设getTerminal() 开始运行时repOK为true

获取Terminal，不修改信息

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设getTime() 开始运行时repOK为true

获取Time不修改信息

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设getOatF(int i) 开始运行时repOK为true

获取指定楼层的指令

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设n\_pos(long t) 开始运行时repOK为true

计算当前楼层，不修改信息

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设StatusRefreash(int term, long time) 开始运行时repOK为true

刷新状态， \this.Position = \this.Terminal; \this.Terminal = term; \this.Time = time;\this.Time = getArrivalTime();，

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设toString() 开始运行时repOK为true

获取当前状态，不会修改信息

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设getArrivalTime() 开始运行时repOK为true

获取ArrivalTIme，不修改信息

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设adjust() 开始运行时repOK为true

将当前时间加1

方法执行结束，因此该方法执行不会导致repOK为假

* 1. 假设isfloor(int n)开始运行时repOK为true

获取当前电梯内按钮点亮情况，不修改信息

方法执行结束，因此该方法执行不会导致repOK为假

1. 方法有效性
   1. setFloor(Order tmp)

有以下划分

floor[tmp.getTarget()] = false with tmp.getType().equals("ER")

do nothing with tmp.getType().equals("FR")

方法检查指令tmp的类型，若为ER则将指定的楼层按钮设置为false，满足后置条件，满足floor[tmp.getTarget()] = false with tmp.getType().equals("ER")

检查tmp的类型，若为FR指令则直接返回，满足do nothing with tmp.getType().equals("FR")

* 1. setFloor(int n)

无分支，将指定楼层点亮设置为false，满足后置条件

* 1. judge(Order tmp)

/\*

@REQUIRES:None

@MODIFIES:\this.floor

@EFFECTS: floor[tmp.getTarget()]==>tmp.setEnable(false);

!floor[tmp.getTarget()]==>OatF[tmp.getTarget()] = tmp;&&tmp.setEnable(true);

\*/

有以下分支

OatF[tmp.getTarget()] = tmp;&&tmp.setEnable(true); with ！floor[tmp.getTarget()]

tmp.setEnable(false) with floor[tmp.getTarget()]

通过方法检查指定楼层有指令时，将tmp设置为无效，满足后置条件，满足tmp.setEnable(false) with floor[tmp.getTarget()]

通过方法检查指定楼层没有指令时，将指令设置为有效，并将指令存入楼层的指令保存区，满足后置条件，满足OatF[tmp.getTarget()] = tmp;&&tmp.setEnable(true); with ！floor[tmp.getTarget()]

* 1. pickup(int n)

/\*

@REQUIRES:None

@MODIFIES:System.out,OatF

@EFFECTS: n==1||n==10==>\result=false

floor[n]==>System.out.print("["+OatF[n].getReq()+"]/");&&OatF[n].setEnable(false)&&\result = false;

\result = false;

\*/

有以下分支

\result=false with n==1||n==10

System.out.print("["+OatF[n].getReq()+"]/");&&OatF[n].setEnable(false)&&\result = false; with floor[n]

\result=false with ！floor[n]

通过方法检查，检查的楼层为1或10时，不允许捎带，直接返回false，满足后置条件，满足\result=false with n==1||n==10

通过方法检查，楼层有可捎带指令时，输出指令信息，并将该指令设置为false，满足后置条件，满足System.out.print("["+OatF[n].getReq()+"]/"); &&OatF[n].setEnable(false)&&\result = false; with floor[n]

通过方法检查，楼层没有捎带指令时，直接返回false，满足后置条件，满足result=false with ！floor[n]

* 1. getPosition()

不存在分支，返回position，满足后置条件

* 1. getTerminal()

不存在分支，返回terminal，满足后置条件

* 1. getTime()

不存在分支，返回time，满足后置条件

* 1. getOatF(int i)

不存在分支，返回OatF[i]，满足后置条件

* 1. n\_pos(long t)

/\*

@REQUIRES: None

@MODIFIES: None

@EFFECTS: \this.Position>\this.Terminal==>\result = \this.Terminal+(int)(\this.Time-t)/5;

\this.Terminal>\this.Position==>\result = \this.Terminal-(int)(\this.Time-t)/5;

return \this.Position

\*/

有以下分支

\this.Terminal+(int)(\this.Time-t)/5; with \this.Position>\this.Terminal

\this.Terminal-(int)(\this.Time-t)/5; with \this.Position<\this.Terminal

\this.Position with \this.Terminal==\this.Position

通过方法测试，position = 1， terminal = 1， time = 0， t = 0，返回1，满足后置条件，满足\this.Position with \this.Terminal==\this.Position

通过方法测试，position = 1， terminal = 2， time = 0， t = 0，返回2，满足后置条件，满足\this.Terminal-(int)(\this.Time-t)/5; with \this.Position<\this.Terminal

通过方法测试，position =2， terminal = 1， time = 0， t = 0，返回1，满足后置条件，满足\this.Terminal+(int)(\this.Time-t)/5; with \this.Position>\this.Terminal

* 1. StatusRefreash(int term, long time)

/\*

@REQUIRES:None

@MODIFIES:None

@EFFECTS: \this.Position = \this.Terminal;

\this.Terminal = term;

\this.Time = time;//新出发时间

\this.Time = getArrivalTime();

\*/

无分支，只刷新信息，满足后置条件

* 1. toString()

/\*

@REQUIRES:None

@MODIFIES:None

@EFFECTS: this.Terminal>this.Position==>\result = "UP"

this.Terminal<this.Position==>\result = "DOWN"

this.Terminal==this.Position==>\result = "STILL"

\*/

有以下分支

\result = "UP" with this.Terminal>this.Position

\result = "DOWN" with this.Terminal<this.Position

\result = "STILL" with this.Terminal==this.Position

通过方法测试，position = 1， terminal = 2，返回UP，满足后置条件，满足this.Terminal>this.Position==>\result = "UP"

通过方法测试，position = 2， terminal = 2，返回STILL，满足后置条件，满足this.Terminal==this.Position==>\result = "STILL"

通过方法测试，position = 2， terminal = 1，返回DOWN，满足后置条件，满足this.Terminal<this.Position==>\result = "DOWN"

* 1. getArrivalTime()

/\*

@REQUIRES:None

@MODIFIES:None

@EFFECTS: \this.Terminal>=\this.Position==>\result = \this.Time+10+(\this.Terminal-\this.Position)\*5;

\this.Terminal<\this.Position ==>、result = \this.Time+10+(\this.Position-\this.Terminal)\*5;

\*/

有以下分支

\result = \this.Time+10+(\this.Terminal-\this.Position)\*5; with \this.Terminal>=\this.Position

result = \this.Time+10+(\this.Position-\this.Terminal)\*5; with \this.Terminal<\this.Position

通过方法测试，terminal>position时，返回Time+10+(Terminal-Position)\*5，满足后置条件，满足\result = \this.Time+10+(\this.Terminal-\this.Position)\*5; with \this.Terminal>=\this.Position

通过方法测试，terminal < position时返回Time+10+(Position-Terminal)\*5，满足后置条件，满足result = \this.Time+10+(\this.Position-\this.Terminal)\*5; with \this.Terminal<\this.Position

* 1. adjust()

无分支，调整预计到达时间，满足后置条件

* 1. isfloor(int n)

无分支，返回按钮状态，满足后置条件

OrderArray

1. 抽象

指令队列，存贮所有输入的指令信息

1. 对象有效性
   1. 初始化时，实例化指令队列链表，该方法的执行不会导致repOK为假，不违背表示不变式。
   2. 假设add()开始执行时repOK为true

方法首先检查指令是否有效，无效直接返回，然后检查是否为第一个指令，第一个指令必须为（FR，1，UP，0），否则输出错误信息并返回。之后检查时间是否按照从小到大排列，若不满足顺序，则报错并返回，满足则加入指令队列Array。

因此，该方法的执行不会导致repOK为假，不违背表示不变式

* 1. 假设getArray()开始执行时repOK为true

获取Array，不改变信息，满足不变式。

因此，该方法的执行不会导致repOK为假，不违背表示不变式

1. 方法有效性
   1. Add(Order a)

/\*

@REQUIRES:None

@MODIFIES:System.out

@EFFECTS: i==-1&&a.getTime()==0 && a.getType().equals("FR") && a.getTarget()==1 && a.getUpDown().equals("UP")==>Array.add(a)&&i++

i==-1&&a.getTime()!=0 || !a.getType().equals("FR") || a.getTarget()!=1 || !a.getUpDown().equals("UP")==>System.out.println("INVALID["+a.getReq()+"]")

i!=-1&&a.getTime()>=Array.get(i).getTime()==>Array.add(a)&&i++;

i!=1&&a.getTime()<Array.get(i).getTime()==>System.out.println("INVALID["+a.getReq()+"]")

\*/

有以下分支

Do nothing with !a.isright

Array.add(a)&&i++ with i==-1&& a.getTime()==0 && a.getType().equals("FR") && a.getTarget()==1 && a.getUpDown().equals("UP")

System.out.println("INVALID["+a.getReq()+"]") with i==-1&&( a.getTime()!=0 || !a.getType().equals("FR") || a.getTarget()!=1 || !a.getUpDown().equals("UP"))

Array.add(a)&&i++ with i!=-1&& a.getTime()>=Array.get(i).getTime()

System.out.println("INVALID["+a.getReq()+"]") with i!=-1&& a.getTime() < Array.get(i).getTime()

通过方法测试，"abc"，不符合要求，a.isright返回false，满足后置条件，满足Do nothing with !a.isright

通过方法测试，"(FR,2,UP,0)"，不符合第一条为，“（FR，1，UP，0）”，报错并返回，满足后置条件，满足System.out.println("INVALID["+a.getReq()+"]") with i==-1&&( a.getTime()!=0 || !a.getType().equals("FR") || a.getTarget()!=1 || !a.getUpDown().equals("UP"))

通过方法测试，"(FR,1,UP,0)"，满足第一条指令要求，满足后置条件，满足Array.add(a)&&i++ with i==-1&& a.getTime()==0 && a.getType().equals("FR") && a.getTarget()==1 && a.getUpDown().equals("UP")

通过方法测试，"(FR,3,UP,3)"，满足正常指令要求，被加入指令队列，满足后置条件，满足Array.add(a)&&i++ with i!=-1&& a.getTime()>=Array.get(i).getTime()

通过方法测试，"(FR,1,UP,1)"，不满足时间顺序要求，报错并返回，满足后置条件，满足System.out.println("INVALID["+a.getReq()+"]") with i!=-1&& a.getTime() < Array.get(i).getTime()

* 1. getArray()

不存在分支，获取指令队列，满足后置条件