出租车调度系统设计文档

[出租车调度系统设计文档 1](#_Toc450984098)

[一、 Customer.java 3](#_Toc450984099)

[1. Overview 3](#_Toc450984100)

[2. 过程规格 3](#_Toc450984101)

[3. 表示对象 4](#_Toc450984102)

[4. 抽象函数 4](#_Toc450984103)

[5. 不变式 4](#_Toc450984104)

[二、 Flow.java 5](#_Toc450984105)

[1. Overview 5](#_Toc450984106)

[2. 过程规格 5](#_Toc450984107)

[3. 表示对象 5](#_Toc450984108)

[4. 抽象函数 5](#_Toc450984109)

[5. 不变式 6](#_Toc450984110)

[三、 Map.java 6](#_Toc450984111)

[1. Overview 6](#_Toc450984112)

[2. 过程规格 6](#_Toc450984113)

[3. 表示对象 8](#_Toc450984114)

[4. 抽象函数 8](#_Toc450984115)

[5. 不变式 8](#_Toc450984116)

[四、 Node.java 8](#_Toc450984117)

[1. Overview 8](#_Toc450984118)

[2. 过程规格 9](#_Toc450984119)

[3. 表示对象 13](#_Toc450984120)

[4. 抽象函数 13](#_Toc450984121)

[5. 不变式 13](#_Toc450984122)

[五、 NodePair.java 13](#_Toc450984123)

[1. Overview 13](#_Toc450984124)

[2. 过程规格 14](#_Toc450984125)

[3. 表示对象 14](#_Toc450984126)

[4. 抽象函数 14](#_Toc450984127)

[5. 不变式 14](#_Toc450984128)

[六、 Order.java 14](#_Toc450984129)

[1. Overview 14](#_Toc450984130)

[2. 过程规格 15](#_Toc450984131)

[3. 表示对象 16](#_Toc450984132)

[4. 抽象函数 16](#_Toc450984133)

[5. 不变式 16](#_Toc450984134)

[七、 Pair.java 17](#_Toc450984135)

[1. Overview 17](#_Toc450984136)

[2. 过程规格 17](#_Toc450984137)

[3. 表示对象 17](#_Toc450984138)

[4. 抽象函数 18](#_Toc450984139)

[5. 不变式 18](#_Toc450984140)

[八、 Taxi.java 18](#_Toc450984141)

[1. Overview 18](#_Toc450984142)

[2. 过程规格 19](#_Toc450984143)

[3. 表示对象 21](#_Toc450984144)

[4. 抽象函数 21](#_Toc450984145)

[5. 不变式 21](#_Toc450984146)

[九、 TrafficLight.java 21](#_Toc450984147)

[1. Overview 21](#_Toc450984148)

[2. 过程规格 22](#_Toc450984149)

[3. 表示对象 22](#_Toc450984150)

[4. 抽象函数 22](#_Toc450984151)

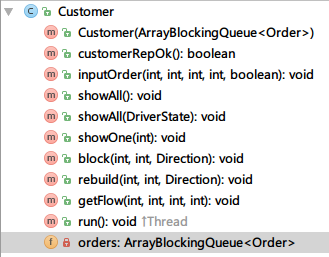
[5. 不变式 22](#_Toc450984152)

# Customer.java

## Overview

测试线程类，包含有构造传入的请求队列orders，主要的修改方法和测试线程均在此类中。

包含构造方法和repok方法，包含了输入请求的方法、显示车辆信息的方法、阻塞和修复道路的方法，获得道路流量的方法



## 过程规格

|  |
| --- |
| // Require : null // Modified : null // Effect ： 不变式的值**public boolean** customerRepOk()  //Require : ArrayBlockingQueue 格式的 orders //Modified : this.orders //Effect : 构造传入orders**public** Customer(ArrayBlockingQueue<Order> orders)  //Require : 0 <= x,y,endx,endy <= 79, tracked = false or true //Modified： orders //Effect : 用来输入请求,将以(x,y) 为起点，(endx,endy)为终点，跟踪状态为track的请求加入到请求队列中  //当输入不合法时将不会正常工作并抛出异常 **public void** inputOrder(**int** x, **int** y, **int** endx, **int** endy, **boolean** track)  //Require: null //Modified： null //Effect : 用来查看现在所有出租车的状况，挨个输出，简单粗暴 **public void** showAll()  //Require : 想要查询的出租车状态 driverState,类型是DriverState //Modified：null //Effect : 用来打印所有driverstate状态的车辆的状况**public void** showAll(DriverState driverState)  //Require : 想要查询车辆的id号, 0<=id <= 100 //Modified：null //Effect : 查询车辆号为id的车辆的运动状态**public void** showOne(**int** id)  //Require : 0<=x,y<=79 , Direction 类型的dir //Modified：Map //Effect : 用来实时阻塞道路，阻塞的道路为以(x,y)为端点，指向dir方向的道路**public void** block(**int** x, **int** y, Direction dir)  //Require : 0<=x,y<=79 , Direction 类型的dir //Modified：Map //Effect : 用来实时恢复被阻塞道路，恢复的道路为以(x,y)为端点，指向dir方向的道路**public void** rebuild(**int** x, **int** y, Direction dir)  //Require : 0<=x,y,anox,anoy<=79 //Modified: n1 n2 //Effect : 用来打印以(x,y),(anox,anoy)为端点的道路的车流量，非正确坐标将会输出错误提示**public void** getFlow(**int** x, **int** y, **int** anox,**int** anoy) |

## 表示对象

传入的请求队列orders

## 抽象函数

AF(c) = （请求队列） where

请求队列 = c.orders

## 不变式

Orders is kind of ArrayBlockingQueue

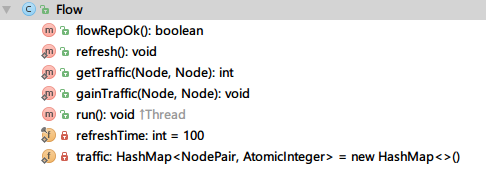
# Flow.java

## Overview

用来统计道路流量相关的类。定义了流量统计时间窗的长度refreshTime，通过保存以点对信息为key，流量为value的Hashmap traffic来保存当前的流量信息

包含了构造方法和repOK方法

包含了刷新，增加，获得流量的方法



## 过程规格

|  |
| --- |
| // Require : null // Modified : null // Effect : 获得当前Flow对象不变式的值**public boolean** flowRepOk()  // Require : null // Modified: traffic // Effect : 用来清空所有在上一个时间窗口内被记录的数据, traffic将被清空**public static synchronized void** refresh()  // Require : n1 != null && n2 != null // Modified : np // Effect : 用来获得以n1 n2 为端点的边的交通流量**public static synchronized int** getTraffic(Node n1, Node n2)  // Require : n1 != null && n2 != null // Modified : traffic // Effect : 用来增加以n1 n2 为端点的道路的流量**public static synchronized void** gainTraffic(Node n1, Node n2) |

## 表示对象

刷新时间窗长度freshTime，保存用量用的Hashmap traffic

## 抽象函数

AF(c) = (刷新时间，结点对[i]，与结点对[i]对应的流量) where

刷新时间 = c.freshTime

结点对 = traffic.getKey()

结点对所对应的流量 = traffic.get(结点对)

## 不变式

FreshTime > 0 && freshTime < 100 && traffic is kind of Hashmap

# Map.java

## Overview

用来储存地图信息的类，其中有修改地图以及计算最短路径的相关方法

file和dir分别为保存地图结点连接关系的地图的文件及其路径

verticalFile和verticaldir分别为保存了交叉关系的文件及其路径

bf用来读取文件，并将读取后的信息存入readinStr中，并分割为char的二维数组存入pointset中

定义了地图的规格，应为MAXCOUNT\*MAXCOUNT的方阵

matrix保存了结点的矩阵

changes记录了当前共有几次不重复的道路阻断

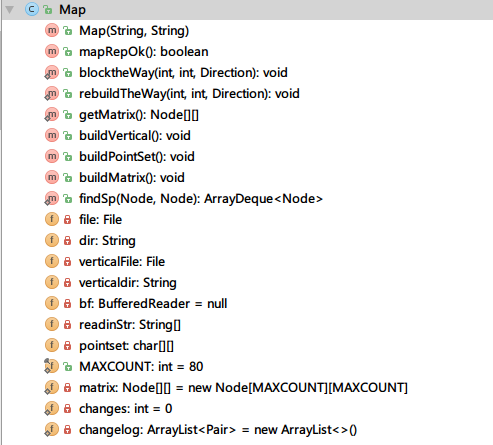
changelog记录了经过改变的结点对

包含构造方法和RepOk方法

包含了阻塞及修复道路的方法

包含了从文件构造地图的方法

包含了使用A\*算法寻找最短路的算法



## 过程规格

|  |
| --- |
| // Require : Map is initialized // Modified : null // Effect : 获得Map不变式的值**public boolean** mapRepOk()  // Require : dir,verticaldir != null // Modified：matrix,file,verticalFile,this.verticaldir,this.dir,pointset,readinStr // Effect : 构造以dir路径的file为地图文件，verticaldir路径的file文件为交叉地图文件的地图，并对一些值进行初始化**public** Map(String dir, String verticaldir)  //Require : 0<=x,y<MAXCOUNT dir类型为DIrection且不为null //Modified：Map,n,another // Effect : 阻断道路的方法，输入想要阻断道路的一个端点和其对应的方向即可打断已有的道路**public static synchronized void** blocktheWay(**int** x, **int** y, Direction dir) {  // Require : 0<=x,y<MAXCOUNT dir类型为DIrection且不为null // Modified： Map, n , ano // Effect : 重新修复已经被阻断的道路,如果该道路没有被blocktheWay方法截断，则无法正常工作并打印错误信息**public static synchronized void** rebuildTheWay(**int** x, **int** y, Direction dir)  // Require : null //Modified：null // Effect : 获得matrix **public static** Node[][] getMatrix()  // Require ： null // Modified : matrix // Effect : 通过交叉文件来建设红绿灯**public void** buildVertical()  // Require : null //Modified：bf,temp,line,readinStr,pointset // Effect : 将传入的字符串转化为点阵**public void** buildPointSet()  // Require : null // Modified: matrix,related Nodes // effect : 将点阵转化为Node结点**public void** buildMatrix()  // Require : 起点和终点结点start和end // Modified: openlist,sp,closelist,check,route // Effect : 计算从start到end的最短路径,使用A\*算法**public static synchronized** ArrayDeque<Node> findSp(Node start, Node end) |

## 表示对象

邻接图和交叉图的文件，file和verticalFile，以及二者所对应的路径dir和verticaldir.

用来读取文件的BufferedReader bf

用来缓存文件数据的readinStr和pointset

定义了地图尺寸的MAXCOUNT

用来保存地图结点信息的matrix

用来记录地图改动次数的changes，改动相同的边不重复计数

用来记录哪些边曾经被修改过的changelog

## 抽象函数

AF(c) = (邻接图路径，邻接图文件，交叉图路径，交叉图文件，读取数据用的BR，字符串缓存,字符缓存,地图尺寸，结点矩阵，地图变化次数，修改信息) where

邻接图路径 = c.dir

邻接图文件 = c.file

交叉图路径 = c. verticaldir

读取数据用的BR = c.bf

字符串缓存 = c.readinStr

字符缓存 = c.pointset

地图尺寸= c.MAXCOUNT

结点矩阵= c.matrix

地图变化次数 = c.changes

修改信息 = c.changelog

## 不变式

Dir != null && verticaldir != null && File.exist() && verticalFile.exist() && matrix !=null && 0<changes<5 && changelog !=null && readinStr.length == MAXCOUNT &&pointset.length == MAXCOUNT && 0 <MAXCOUNT<Integer.MAX\_VALUE

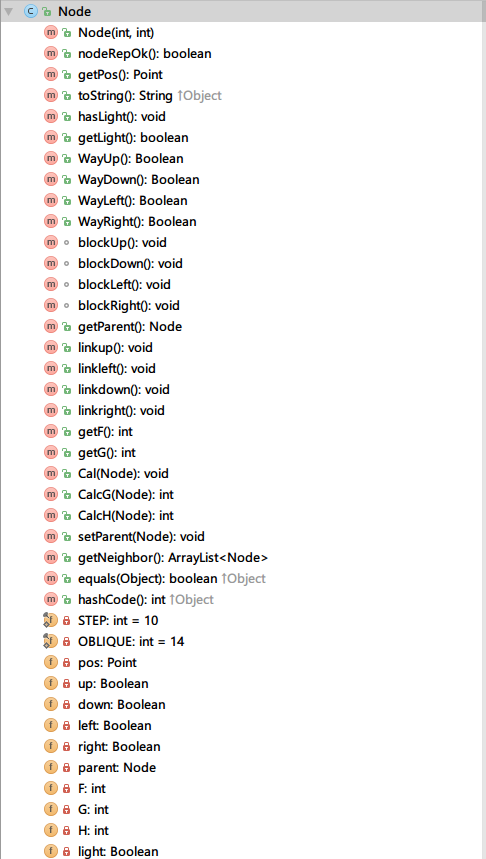
# Node.java

## Overview

结点类，用来保存地图信息。定义了用于A-star计算的STEP和OBLIQUE静态常量。  
每个结点保存有其位置信息pos，其上下左右的邻接关系up, down, left, right  
记录其父亲结点parent, 记录了A-star F，G，H估算值  
记录是否有灯的属性light

包含了构造方法和RepOK方法

包含获得结点位置的方法，获得节点信息的方法，获得结点是否有灯，设置节点有灯属性，连接上下左右结点，封闭上下左右道路，是否与周围结点临街，获取父亲结点，获取或计算FGH的值，获得邻接节点，判断是否相等的方法



## 过程规格

|  |
| --- |
| // Require : Node is initialized // Modified : null // Effect : 获得当前Node对象的不变式的值**public boolean** nodeRepOk()  // Require : pos != null // Modified: null // Effect : 获得当前结点的位置**public** Point getPos()  // Require : pos != null // Modified: null // Effect : 返回当前结点的信息@Override **public** String toString()  // Require : x,y != null // Modified : pos,up,down,left,right,light // Effect : 构造位置为(x,y)结点，并对相应的变量进行初始化**public** Node(**int** x, **int** y)  //Require : null //Modified : light //Effect : 若该结点有大于等于3个邻接节点，则将light置为true **public void** hasLight()  // Require : this != null // Modified : null // Effect : 获得当前路口是否有灯**public boolean** getLight()  // Require : null // Effect : 判定该结点是否有向上延伸的道路 // Modified: null**public** Boolean WayUp()  // Effect : 判定该结点是否有向下延伸的道路 // Modified: null**public** Boolean WayDown()  // Modified: null // Effect : 判定该结点是否有向左延伸的道路**public** Boolean WayLeft()  // Modified: null // Effect : 判定该结点是否有向右延伸的道路**public** Boolean WayRight()  // Modified: up // Effect : 将向上连接道路的标示标记为false，即表示道路被阻断**void** blockUp()  // Modified: down // Effect : 将向下连接道路的标示标记为false，即表示道路被阻断**void** blockDown()  // Modified: left // Effect : 将向左连接道路的标示标记为false，即表示道路被阻断**void** blockLeft()  // Modified: right // Effect : 将向右连接道路的标示标记为false，即表示道路被阻断**void** blockRight()  // Modified: null // Effect : 获得结点的父亲结点**public** Node getParent()  // Modified: up // Effect : 将向上有道路连接的标示标记为true，即表示与向上的点之间存在通路**public void** linkup()  // Modified: left // Effect : 将向左有道路连接的标示标记为true，即表示与左边的点之间存在通路**public void** linkleft()  // Modified: down // Effect : 将向下有道路连接的标示标记为true，即表示与下边的点之间存在通路**public void** linkdown()  // Modified: right // Effect : 将向右有道路连接的标示标记为true，即表示与右边的点之间存在通路**public void** linkright()  // Modified: null // Effect :　获得该点现在的F权值，用于A-star寻路算法**public int** getF()  // Modified: null // Effect : 获得该点现在的G估计值，用于A-star寻路算法**public int** getG()  // Require : 终点节点end // Modified: G,H,F // Effect : 计算更新所有与A-star算法相关的值，共涉及G,H,F三个变量**public void** Cal(Node end)  // Require : 结点node , node需要与this相邻 // Modified: temp,dis,parentG,G // Effect : 用来计算A-star中G的值**public int** CalcG(Node node)  // Require : 终点Node // Modified: step,H // Effect : 计算A-star中H的值**public int** CalcH(Node end)  // Require : Node v， v需要是this的邻接节点 // Modified: this.parent // Effect : 将this.parent 置为 v**public void** setParent(Node v)  // Require : null // Modified: neighbor // Effect : 来获知当前结点有几个邻接结点,返回包含有this邻接节点的Arraylist**public** ArrayList<Node> getNeighbor()  // Require : 传入对象o // Modified : null // Effect : 用来判断this与o两个对象是否相同@Override **public boolean** equals(Object o)  // Require : null // Modified : this.hashcode // Effect : 简单地通过位置来计算this的ashCode@Override **public int** hashCode() |

## 表示对象

定义了A-star算法所用的常量STEP和OBLIQUE

位置pos，邻接关系up,down,left,right

父亲结点parent

1. star用的值F,G,H

是否有灯light

## 抽象函数

AF(c) = (邻接节点权值，对角结点权值，位置，有上方邻接节点，有下方邻接节点，有左方邻接节点，有右方邻接节点，父亲结点，综合估值，起点估值，终点估值，红绿灯存在) where

邻接节点权值= c.STEP

对角结点权值= c.OBLIQUE

位置 = c.pos

有上方邻接节点 = c.up

有下方邻接节点 = c.down

有左方邻接节点 = c.left

有右方邻接节点 = c.right

父亲结点 = c.parent

综合估值 = c.F

起点估值 = c.G

终点估值 = c.H

红绿灯存在 = c.light

## 不变式

STEP == 10 && OBLIQUE == 14 && pos != null && up !=null && down !=null && left != null && right != null && (parent == null || parent instanceof Node) && light != null

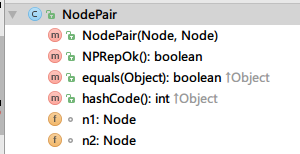
# NodePair.java

## Overview

点对类型，用来保存两个点，主要用于RoadCounter类

点对通过保存两个端点来实现，n1, n2均为Node类型

包含构造方法，RepOk方法，判断相等的方法



## 过程规格

|  |
| --- |
| // Require : NodePair is initialized // Modified : null // Effect : 获得当前NodePair对象的不变式的值**public boolean** NPRepOk()  // Require ： n1 != null n2 != null // Modified : this.n1 this.n2 // Effect : 构造NodePair**public** NodePair(Node n1, Node n2)  // Require ： 另一个对象O // Modified: null // Effect ： 判断this和o是否相同,其中只需要满足包含的结点相同即可判定为相等,与结点的顺序无关。@Override **public boolean** equals(Object o)  // Require : this!=null; // Modified : small,big,result // Effect : 计算当前点对的hashcode@Override **public int** hashCode() |

## 表示对象

两个结点n1,n2

## 抽象函数

AF(c) = (一个结点，另一个结点) where

一个结点= c.n1

另一个结点 = c.n2

## 不变式

n1 != null && n2 != null

# Order.java

## Overview

请求类，用来储存请求信息

location和destination变量用来保存请求的起点和终点

tracked用来保存该请求的跟踪状态

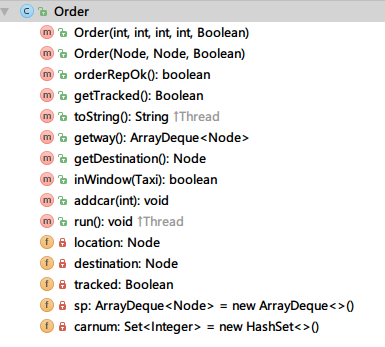
sp用来记录起点到终点的最短路径

carnum用来记录在3s内经过窗口的车辆的车牌号

包括了构造和RepOK方法

包括了设定追踪状态的方法，判断窗口内车辆以及添加车辆的方法

包括了获得请求信息的方法，获得最短路径的方法，获得目的地的方法



## 过程规格

|  |
| --- |
| // Require : Order is initialized, Map is initialized // Modified : null // Effect : 获得当前对象的不变式的值**public boolean** orderRepOk()  // Require : null // Modified: null // Effect : 用来获得改指令的跟踪状态**public** Boolean getTracked()  // Require : null // Modified: null // Effect :用来返回指令的打印信息@Override **public** String toString()  // Require : 0<=x,y,enx,enxy <MAXCOUNT , tracked != null // Modified: location,destination,tracked,sp // Effect : 请求的构造方法，传入起点和终点的坐标和是否被跟踪来创建请求 **public** Order(**int** x, **int** y, **int** endx, **int** endy, Boolean tracked)  // Require : 起点和终点结点location和destination 跟踪状态tracked // Modified: destination,location,tracked,sp // Effect : 请求的另一个种构造方法，通过直接传入起点和终点的结点来创建请求**public** Order(Node location, Node destination, Boolean tracked)  // Require : null // Modified: null // Effect : 获得最短路径sp**public** ArrayDeque<Node> getway()  // Require: null // Modified: null // Effect : 获得该请求的终点**public** Node getDestination()  // Require : 出租者t ！= null // Modified: null // Effect : 判断出租车t是否在响应窗口里**public boolean** inWindow(Taxi t)  // Require : 0<=i<100 // Modified: carnum // Effect : 将车辆的编号添加到carnum中**public void** addcar(**int** i) |

## 表示对象

结点的起点location终点destination，跟踪状态tracked，最短路径sp，经过窗口的车辆编号集carnum

## 抽象函数

AF(c) = （起点，终点，跟踪状态，最短路径，响应车辆集）where

起点 = c.location

终点 = c.destination

跟踪状态 = c.tracked

最短路径 = c.sp

相应车辆集 = c.carnum

## 不变式

Location is kind of Node && destination is kind of Node && tracked != null && sp is kind of ArrayDeque && carnum is kind of Set

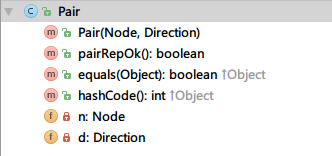
# Pair.java

## Overview

通过节点和方向来记录边的工具类，比较简单

n表示结点，d表示该道路相对于结点的方向

包含构造方法、repOK方法，判断是否相等的方法



## 过程规格

|  |
| --- |
| // Require: Pair is initialized // Modified:null // Effect : 测试Pair对象的不变式的值**public boolean** pairRepOk()  // Require : n !=null d !=null // Modified : // Effect : 工具类的构造方法，传入结点n和方向d来表示道路**public** Pair(Node n, Direction d)  // Require : 另一个对象o // Modified : null // Effect : 判断this和o是否相等的方法@Override **public boolean** equals(Object o)  // Require : null // Modified : this.hashcode // Effect :　获得this的hashcode的方法@Override **public int** hashCode() |

## 表示对象

结点n，方向d

## 抽象函数

AF(c) = （结点，方向） where

结点 = c.n

方向 = c.d

## 不变式

N is kind of Node && d is kind of Direction

# Taxi.java

## Overview

出租车类，用来保存出租车的信息及相关方法

\* 其中id为出租车的编号

\* location和destination分别为当前出租车的位置和将要去的位置

\* moveTo为将要更新的location地址

\* driverState为当前出租车的位置

\* credit为当前出租车的信誉度

\* way2Customer和realOrder分别表示从接到请求到请求起点与从请求起点到请求终点的请求

\* count用来记录当前waiting状态维持时间的counter

\* tracked 表示自己有没有拿到跟踪的请求

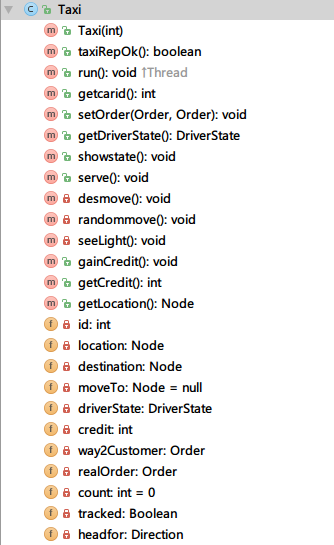
\* headfor是现在车头的方向

\* 包括了构造方法和repOk方法

\* 包括了获得车辆id，运动状态，信誉，位置的方法

\* 包括了给车设定请求的方法

\* 包括车辆随机和有目的移动，等灯，增加信用，服务的方法



## 过程规格

|  |
| --- |
| // Require : Taxi is initialized // Modified : null // Effect : 获得当前对象的不变式的值**public boolean** taxiRepOk()  // Require : 传入车辆的id // Modified: id,location,driverState,destination,credit,way2Customer,realOrder,tracked // Effect : 构造编号为id的车辆**public** Taxi(**int** id)  // Require : null // Modified: null // Effect : 用来查看当前车辆的id**public int** getcarid()  // Require : way2Customer！=null realOrder != null // Modified: way2Customer,realOrder,driverState,count,tracked // Effect : 用来设置请求**public synchronized void** setOrder(Order way2Customer, Order realOrder)  // Require : null //Modified: null // Effect : 用来获得当前车辆的行驶状况**public** DriverState getDriverState()  // Require : null //Modified: null // Effect ： 用来打印车辆的信息，格式为“时间+车号+车辆位置+车辆信用”**public void** showstate()  //Require : null //Modified: wait serv driverState count nei moveTo min samedis finished dis RoadCounter //Effect： 出租车的运行函数。通过参数来实现状态切换**public void** serve()  // Require : location != null deastination != null // Modified : nei,min,finished,samedis,flow,RoadCounter // Effect : 有目标地移动方式**private void** desmove()  // Require : location != null //Modified: neibor moveTo minflow flow RoadCounter // Effect :　实现车辆的随机移动**private void** randommove()  // Require : null // Modified : dx dy // Effect : 根据车车辆的前进方向和红绿灯状态来决定如何行进**private void** seeLight()  // Require : null // Modified : credit // Effect : 当前车辆的信誉值+1，用于车辆存在于扫描队列中时**public synchronized void** gainCredit()  // Require : null // Modified : null // Effect : 获取当前车辆的信誉**public int** getCredit()  // Require : null // Modified : null // Effect : 获取当前车辆的位置**public** Node getLocation() |

## 表示对象

车辆编号，车辆现在的位置，车辆目标终点，车辆的下一个行驶点，车辆状态，车辆信用，车辆所需请求，车辆waiting时间，车辆跟踪状态，车头方向

## 抽象函数

AF(c) = （编号，位置，终点，下一个行驶点，状态，信用，至起点请求，至终点请求，waiting次数，跟踪状态，车头方向） where

编号 = c.id

位置 = c.location

终点 = c.destination

下一个行驶点 c.moveTo

状态 = c.driverState

信用 =c.credit

至起点请求 = c. way2Customer

至终点请求=c.realOrder

waiting次数 = c.count

跟踪状态 =c.tracked

车头方向 = c.headfor

## 不变式

0<=id<100 && location is kind of Node && destination is kind of Node && moveTo is kind of Node && driverState != null && credit > Integer.MAX\_VALUE && tracked != null && headfor != null

# TrafficLight.java

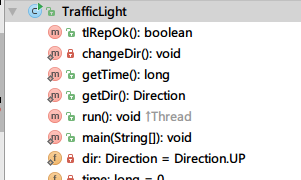
## Overview

红绿灯类。负责整个地图的红绿灯控制

\* dir用来定义现在能够通行的方向

\* time记录上次变灯的时间

\* 包含repOk方法，获得时间和方向的方法，变换方法的方法



## 过程规格

|  |
| --- |
| //Require : null //Modified : null //Effect : 测试对象不变式的值**public boolean** tlRepOk()  // Requrie ： null // Modified ： this.dir // Effect :　用来改变通行的方向**private static void** changeDir()  // Require :　null // Modified : null // Effect :　获得下次变灯时间的方法**public static long** getTime()  //Require : null // Modified : null // Effect : 获得当前允许同行方向**public static** Direction getDir() |

## 表示对象

通行方向，变灯已经过时间

## 抽象函数

AF(c) = （方向，变灯时间） where

方向 = c.dir

变灯时间 = C.time

## 不变式

Dir != null && dir == RIGHT && dir == DOWN && 0<=time < Integer.MAX\_VALUE