
出租车调度系统设计文档

| | |
|------------------------|----|
| 出租车调度系统设计文档 | 1 |
| 一、 Customer.java | 3 |
| 1. Overview | 3 |
| 2. 过程规格 | 3 |
| 3. 表示对象 | 4 |
| 4. 抽象函数 | 4 |
| 5. 不变式 | 4 |
| 二、 Flow.java | 5 |
| 1. Overview | 5 |
| 2. 过程规格 | 5 |
| 3. 表示对象 | 5 |
| 4. 抽象函数 | 5 |
| 5. 不变式 | 6 |
| 三、 Map.java | 6 |
| 1. Overview | 6 |
| 2. 过程规格 | 6 |
| 3. 表示对象 | 8 |
| 4. 抽象函数 | 8 |
| 5. 不变式 | 8 |
| 四、 Node.java | 8 |
| 1. Overview | 8 |
| 2. 过程规格 | 9 |
| 3. 表示对象 | 13 |
| 4. 抽象函数 | 13 |
| 5. 不变式 | 13 |
| 五、 NodePair.java | 13 |
| 1. Overview | 13 |
| 2. 过程规格 | 14 |
| 3. 表示对象 | 14 |
| 4. 抽象函数 | 14 |
| 5. 不变式 | 14 |
| 六、 Order.java | 14 |
| 1. Overview | 14 |
| 2. 过程规格 | 15 |
| 3. 表示对象 | 16 |
| 4. 抽象函数 | 16 |
| 5. 不变式 | 16 |
| 七、 Pair.java | 17 |
| 1. Overview | 17 |
| 2. 过程规格 | 17 |
| 3. 表示对象 | 17 |

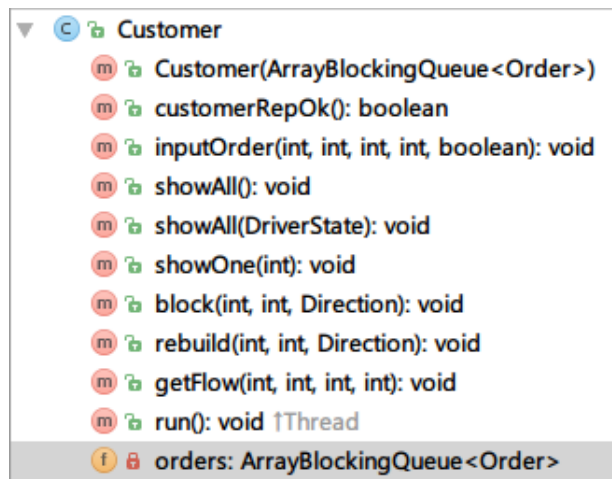
| | | |
|----|------------------------|----|
| 4. | 抽象函数..... | 18 |
| 5. | 不变式..... | 18 |
| 八、 | Taxi.java..... | 18 |
| 1. | Overview | 18 |
| 2. | 过程规格 | 19 |
| 3. | 表示对象 | 21 |
| 4. | 抽象函数..... | 21 |
| 5. | 不变式..... | 21 |
| 九、 | TrafficLight.java..... | 21 |
| 1. | Overview | 21 |
| 2. | 过程规格 | 22 |
| 3. | 表示对象 | 22 |
| 4. | 抽象函数..... | 22 |
| 5. | 不变式..... | 22 |

一、 Customer.java

1. 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



2. 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)

```

3. 表示对象

传入的请求队列 orders

4. 抽象函数

AF(c) = (请求队列) where

请求队列 = c.orders

5. 不变式

Orders is kind of ArrayBlockingQueue

二、 Flow.java

1. 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

```
Flow
  flowRepOk(): boolean
  refresh(): void
  getTraffic(Node, Node): int
  gainTraffic(Node, Node): void
  run(): void ↑Thread
  refreshTime: int = 100
  traffic: HashMap<NodePair, AtomicInteger> = new HashMap<>()
```

2. 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)
```

3. 表示对象

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

4. 抽象函数

$AF(c) = (\text{刷新时间}, \text{结点对}[i], \text{与结点对}[i]\text{对应的流量})$ 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)

5. 不变式

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

三、 Map.java

1. 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

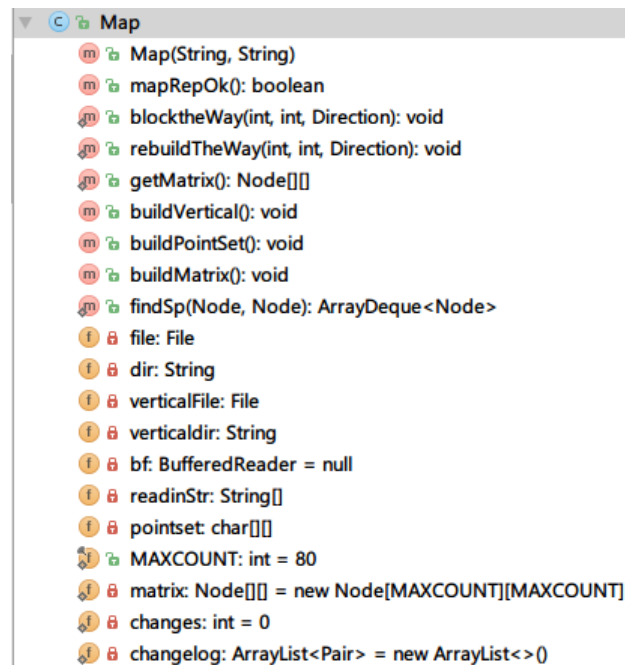
changelog 记录了经过改变的结点对

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



```
Map
  Map(String, String)
  mapRepOk(): boolean
  blocktheWay(int, int, Direction): void
  rebuildTheWay(int, int, Direction): void
  getMatrix(): Node[][]
  buildVertical(): void
  buildPointSet(): void
  buildMatrix(): void
  findSp(Node, Node): ArrayDeque<Node>
  file: File
  dir: String
  verticalFile: File
  verticaldir: String
  bf: BufferedReader = null
  readinStr: String[]
  pointset: char[][]
  MAXCOUNT: int = 80
  matrix: Node[][] = new Node[MAXCOUNT][MAXCOUNT]
  changes: int = 0
  changelog: ArrayList<Pair> = new ArrayList<>()
```

2. 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,closetlist,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)
```

3. 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

4. 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

5. 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

四、 Node.java

1. 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。

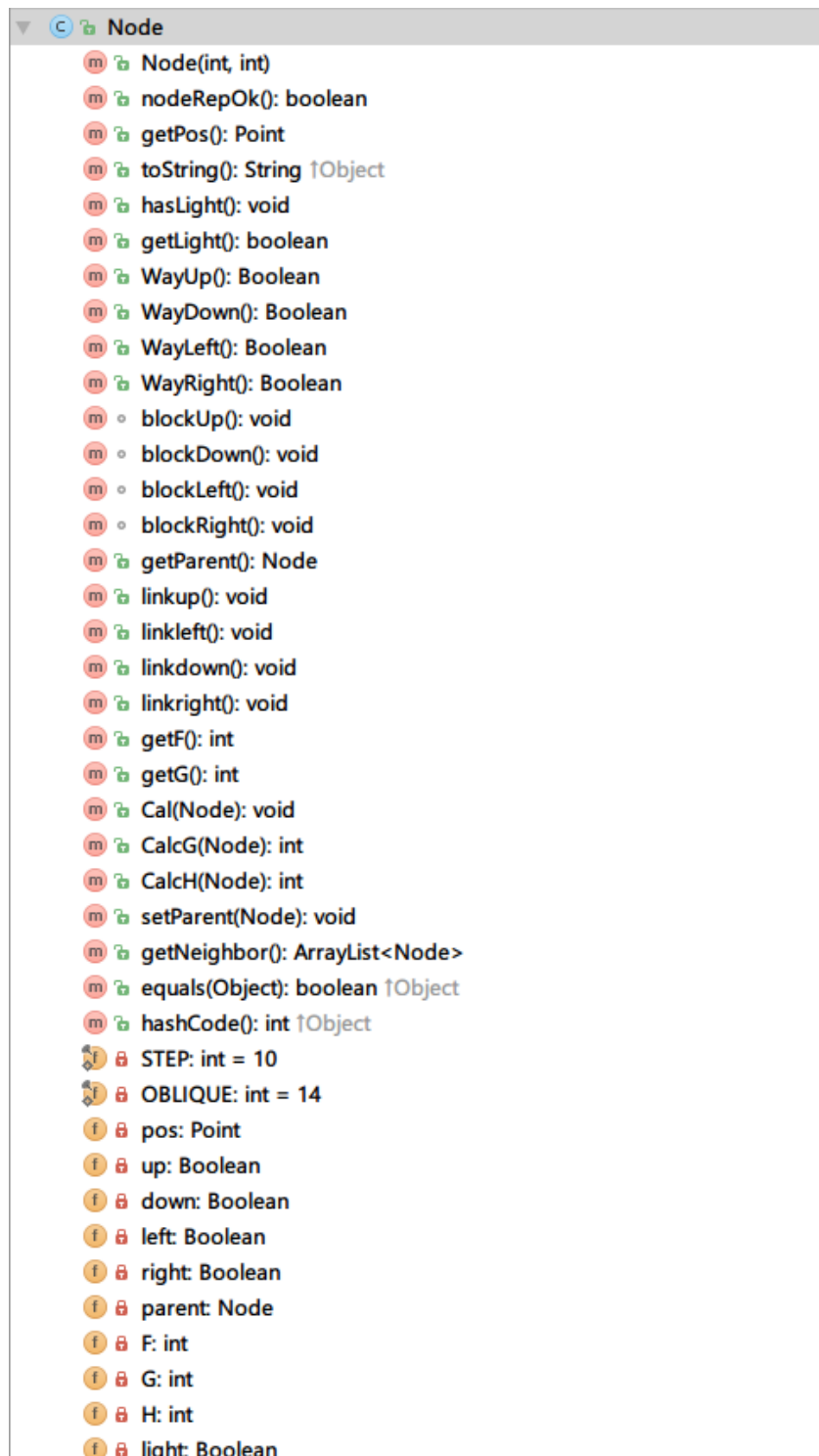
每个结点保存有其位置信息 pos, 其上下左右的邻接关系 up, down, left, right

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



2. 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()
```

3. 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

A- star 用的值 F,G,H

是否有灯 light

4. 抽象函数

AF(c) = (邻接节点权值, 对角节点权值, 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

对角节点权值 = 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

5. 不变式

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

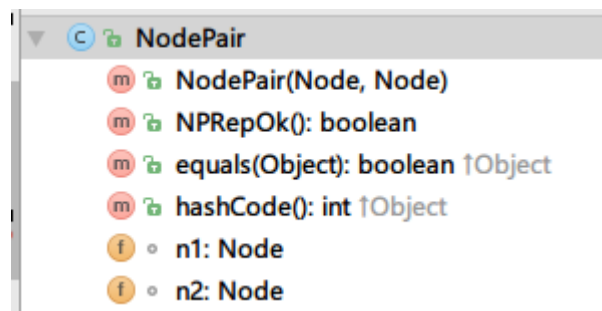
五、 NodePair.java

1. 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



2. 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()
```

3. 表示对象

两个结点 n1,n2

4. 抽象函数

$AF(c) = (\text{一个结点}, \text{另一个结点})$ where
一个结点 = c.n1
另一个结点 = c.n2

5. 不变式

$n1 \neq \text{null} \ \&\& \ n2 \neq \text{null}$

六、 Order.java

1. 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

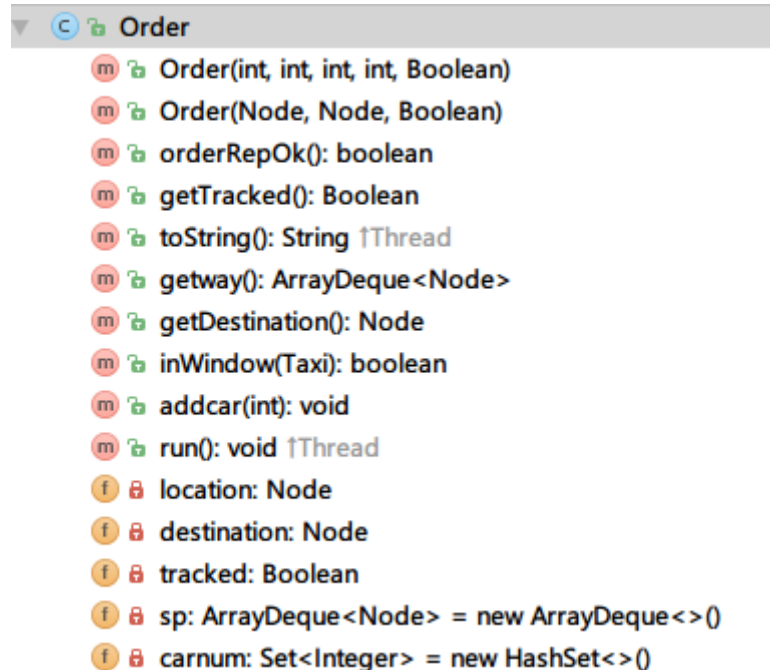
sp 用来记录起点到终点的最短路径

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



2. 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 : $0 \leq x, y, \text{enx}, \text{enxy} < \text{MAXCOUNT}$, 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)

```

3. 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

4. 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

5. 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

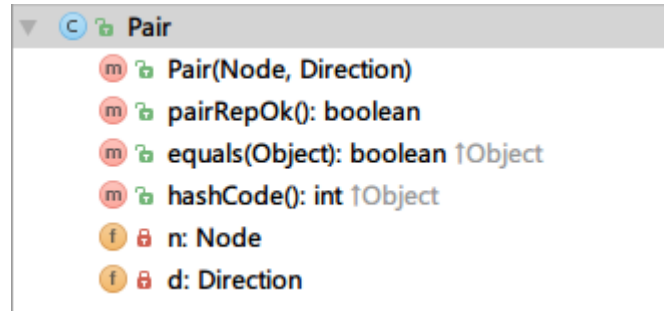
七、 Pair.java

1. 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



2. 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()
```

3. 表示对象

结点 n，方向 d

4. 抽象函数

$AF(c) = (\text{结点}, \text{方向}) \text{ where}$

结点 = c.n

方向 = c.d

5. 不变式

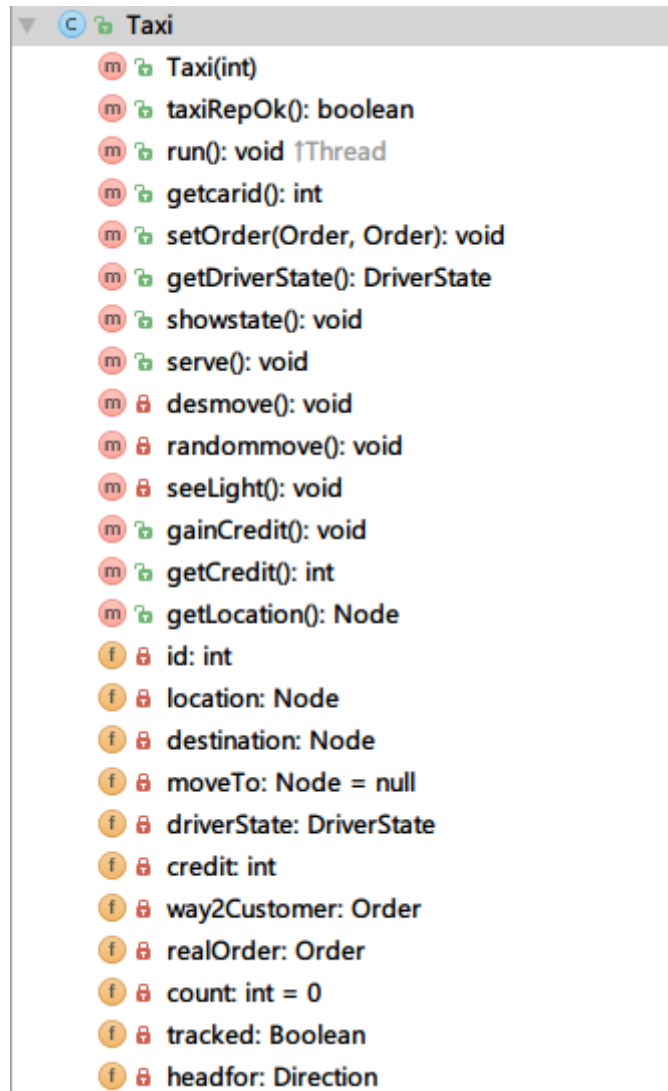
N is kind of Node && d is kind of Direction

八、 Taxi.java

1. 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



2. 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()

```

3. 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

4. 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

5. 不变式

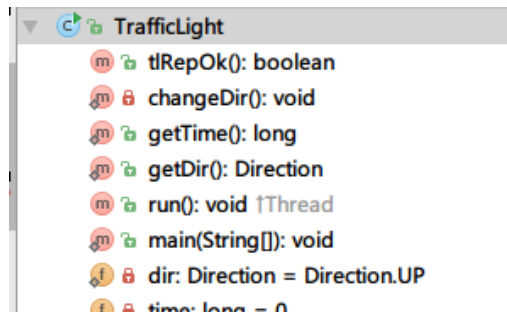
$0 \leq id < 100$ && location is kind of Node && destination is kind of Node && moveTo is kind of Node && driverState != null && credit > Integer.MAX_VALUE && tracked != null && headfor != null

九、 TrafficLight.java

1. 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



2. 过程规格

```
//Require : null
//Modified : null
//Effect : 测试对象不变式的值
public boolean tlRepOk()

// Require : 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()
```

3. 表示对象

通行方向, 变灯已经过时间

4. 抽象函数

AF(c) = (方向, 变灯时间) where

方向 = c.dir

变灯时间 = C.time

5. 不变式

Dir != null && dir == RIGHT && dir == DOWN && 0<=time < Integer.MAX_VALUE