# 一、运行环境和运行指令规范

## a) jdk版本：jdk1.8.0\_111

## b) 平台：macOS / Windows

# 二、输入说明

## a) 输入样例

**乘客请求：**

[CR,(1,2),(3,4)]

**开关边操作：**

OPEN,(78,78),(78,79)

CLOSE,(78,78),(78,79)

## b) 说明

**总约束：**

1. 坐标支持前导零与正号，大小处于0~79（包含0与79）之间，任何超出此范围的坐标的输入视为无效输入。

2. 输入支持空格。

**乘客请求的约束：**

1. 乘客请求格式为

[CR,src,dst]

其中CR为标识符，src和dst均为(i,j)形式的坐标位置，表示乘客请求的发出地和目的地。

2. 一行仅读入一个乘客请求。

**开关边操作的约束：**

1. 开边操作格式为

OPEN,(x1,y1),(x2,y2)

关边操作格式为

CLOSE,(x1,y1),(x2,y2)

其中(x1,y1)和(x2,y2)均为坐标位置。

2. 一行仅读入一个开边操作或者关边操作。

3. 开启初始地图不存在的边视为无效输入。

4. 开启已开启的边或者关闭已关闭的边，会有相应提示。

（友情提示：若选择复制粘贴的方式输入，请注意可能在复制时被忽略的换行符）

# 三、输出规格

## a) 输出样例

Output.txt:

Request {location=[20, 20], destination=[3, 4], time=1492527168944}

Taxis that can get order at beginning:

Taxi No.42 {state = wandering, credit = 0, location = [18, 20]}

Taxi to get order:

Taxi No.42 {state = picking up customer, credit = 2, location = [21, 22]}

Path.txt:

Taxi 42 picking up customer path:

[22, 22]

[21, 22]

[21, 21]

[20, 21]

[20, 20]

Taxi 42 transporting customer path:

[20, 20]

[19, 20]

[18, 20]

[17, 20]

[16, 20]

[16, 19]

[15, 19]

[15, 18]

[15, 17]

[14, 17]

[14, 16]

[13, 16]

[12, 16]

[12, 15]

[12, 14]

[12, 13]

[12, 12]

[11, 12]

[11, 11]

[10, 11]

[9, 11]

[9, 10]

[8, 10]

[7, 10]

[6, 10]

[5, 10]

[5, 9]

[5, 8]

[5, 7]

[4, 7]

[3, 7]

[3, 6]

[3, 5]

[3, 4]

## b) 说明

1.Output.txt输出的是:

请求信息（所在地、目的地、发出时的系统时间）；

请求发出时，处于以请求为中心的4×4区域中的所有出租车状态、信用信息；（区域内无出租车则提示”No taxi available at beginning.”）

在抢单时间窗内所有抢单的出租车；（无抢单车则提示”No taxi can get order now“）

系统选择响应相应请求的出租车。（无抢单车则提示”No taxi can get order now“）

2.Path.txt输出的是出租车响应相应请求过程中的实际行驶路径。

提示语句“Taxi *ID* picking up customer path:”以下输出的是编号为ID的出租车的接客所经过的坐标点。

提示语句“Taxi *ID* transporting customer path:”以下输出的是编号为ID的出租车的送客所经过的坐标点。

# 四、接口类TaxiAPI的方法说明

1. public TaxiAPI(Taxi[] taxis)

构造器方法，传入的参数为程序中所有出现的taxi对象。

2. public synchronized void getTaxiInfo(int taxiID)

通过出租车号taxiID查找出租车信息:

1.当前系统时刻；

2.出租车当前坐标。

其中taxiID是1~100（包含1和100）的数字，超过范围则查询失败并有相应提示。

3. public synchronized ArrayList<Taxi> getTaxisInState(int state)

返回处于state状态的所有出租车对象集合，集合为ArrayList<Taxi>类型。

state的值对应的状态：

0是等待服务，

1是服务状态，

2是接单状态，

3是停止运行。

（具体的实现细节在TaxiAPI.java文件中呈现）

# 五、initializeTaxis()方法说明

1.该方法负责创建出租车对象与出租车进程，并返回创建的100个出租车对象组成的数组。

2.方法中已写好样例代码，以供参考。

具体内容为：构造前70号车为普通出租车，后30号车为可追踪出租车。

为避免违背指导书旨意，若测试者希望看到一个空方法，请将样例代码删除或注释。

3.创建普通出租车对象请以此格式为准：

Taxi taxi = new Taxi(direction, x, y, taxiNo, map, fileWriter);

direction为出租车起始朝向，可随意设定；

x和y是出租车起始坐标，可随意设定；

map位请设置为传入initializeTaxis()方法的map参数；

fileWriter位请设置为传入initializeTaxis()方法的pathWriter参数；

4.创建可追踪出租车对象请以此格式为准：

TraceableTaxi taxi = new TraceableTaxi(direction, x, y, taxiNo, map, fileWriter);

参数要求同3中要求。

5.创建完出租车对象后，请以这种形式将对象放入taxis数组中：

taxis[i] = taxi;

6.创建出租车对象后务必创建出租车线程，两类出租车创建线程的方式一样，都是

TaxiThread taxiThread = new TaxiThread(taxi);

创建线程后记得以taxiThread.start();的形式启动线程。

7.若需要gui显示，请务必在出租车对象创建后，以以下形式设置gui

gui.SetTaxiType(i + 1, taxi.type);

gui.SetTaxiStatus(taxi.getTaxiNo(), new Point(taxi.getLocation()[0], taxi.getLocation()[1]), 2);

此设置对两类出租车都适用。

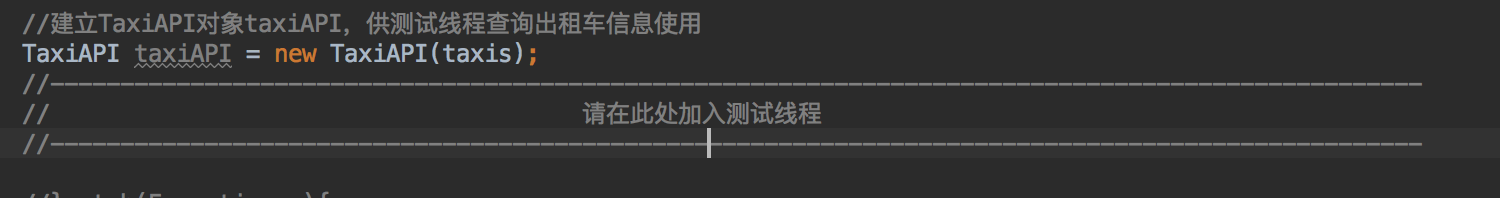
8.方法最后请返回taxis数组;

# 六、上次作业Bug

十几个JSF的问题。（摊手

# 七、其他

1. 测试线程的添加位置已在TaxiTest类的main方法中用注释标出，如下图所示。



2. 如上图所示，测试接口已经实例化，可直接通过taxiAPI对象调用接口方法。

3. 地图文件请命名为map.txt，并置于本程序的工程目录下。

4. Output.txt和Path.txt文件存储在工程目录下。

5. 程序运行中提供道路临时关闭或打开功能，在控制台中输入相应请求即可实现，详细输入格式请见输入格式说明中的“开关边操作的约束”。

6. 红绿灯文件请命名为LightDistribution.txt，并置于本程序的工程目录下。

7. initializeTaxis()方法位于TaxiTest.java的TaxiTest类中，具体使用方法请见本文第六部分的说明。

8.双向迭代器在TraceableTaxi.java的TraceableTaxi类中实现。