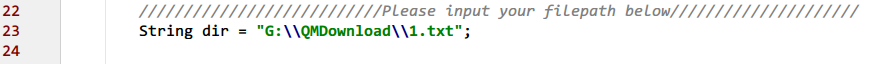
OO第八次作业——出租调度系统说明文档

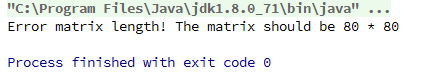
# 输入规范

1. **地图输入**

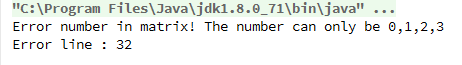
程序的地图数据需要从文件中读取。请于第一次运行程序之前在Main函数的如下位置填写你的地图文件。



地图文件应为txt格式，其内容应为80行字符串，每行字符串恰有80个字符。如果没有符合输入规范，程序将会报错并退出，如下图：



字符串中只能出现数字0,1,2,3,其意义与指导书中相同，如果输入的地图文件中有非法字符，程序将会报错并提示错误的行号，之后退出，如下图：



地图文件的内容应严格执行指导书标准。

测试者应保证自己的地图连通且边界点不可连接到80\*80以外的方格，由于地图不符合规范导致的运行错误本程序概不负责。

具体的格式也可以参照附件中的地图。

1. **请求输入**

本程序的输入采用编写测试线程的方法输入。需要用户自行在Customer中完成想要的操作。具体的操作函数即方法如下。

inputOrder (int x, int y, int endx, int endy, Boolean track)

其中各参数的意义为：

\* @param x 请求起始点的 x 坐标  
\* @param y 请求起始点的 y 坐标  
\* @param endx 请求终点的 x 坐标  
\* @param endy 请求终点的 y 坐标  
\* @param track 是否追踪该请求

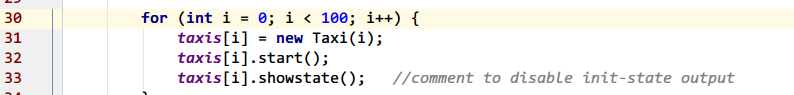
例如：inputOrder(0, 0, 79, 79, **false**);

表示输入一条从(0,0) 到(79，79)的请求，并且不跟踪请求。

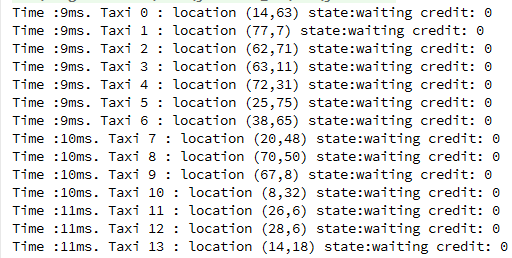
至于输出的作用我们会放到后文说明。

# 输出

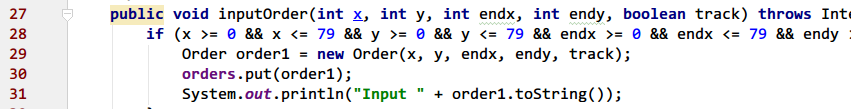
由于本次作业的信息量较大，如果全部输出的话可能不利于测试者测试，因此，本节将会介绍程序的输出规范。

1. **出租车初始状态输出**修改位置: Main函数 line33，注释掉可以取消输出初始状态  
     
   作用：在每辆车生成的时候打印车辆的初始位置，方便了解车辆的大体分布。

效果：就是挨个输出，简单粗暴

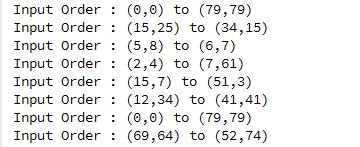


1. **Order成功添加提示信息**修改位置: Customer.java line31，注释掉可以取消添加成功的提示



作用: 证明请求被成功加载到相应队列中

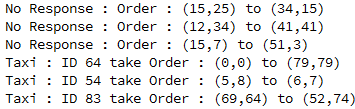
效果：



1. **Order被响应/无人响应**

**修改意见： 不建议修改，涉及很多设计相关的代码，擅自修改容易出错**

**作用：输出当前请求是否被车辆相应了，如果相应了输出车辆的ID，否则输出No Response ,效果如下图：**



1. **Taxi沿途输出**

**修改意见： 不建议修改，涉及核心功能代码**

**作用：显示某辆车相应了请求的车现在的运动状态，并在结束后输出现在的时间和车辆信息。效果如下：**







**coming to passenger 表示车辆正在赶往请求发生的地点，后面的第一个坐标表示从车辆的现在位置，第二个表示请求发生的地点**

**serving 表示车辆现在正在完成请求，后面的坐标自然就是请求的起点和重点**

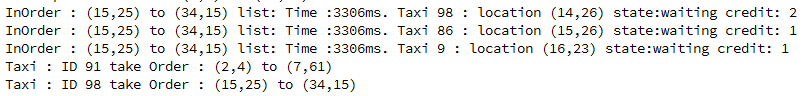
**moving 表示现在车辆移动到了哪个坐标**

**serving finished 表明车辆完成了任务，在等待1s后将会继续进入waiting状态**

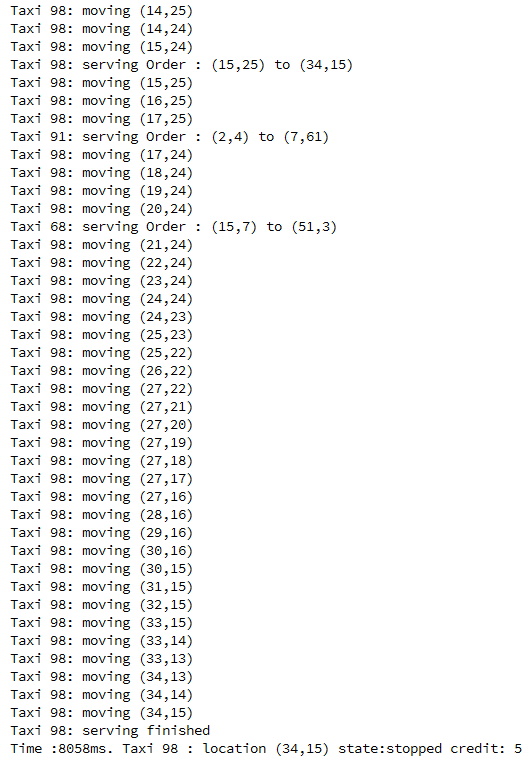
**最后输出的是当前的时间和车辆状态**

1. **请求追踪**

**当tracked的值为true时，程序将会跟踪输出所有与此请求相关的所有信息。包括，有哪些车辆曾经在3s内经过窗口:**



**车辆的具体行走路径**



**个人认为这样足以让测试者判别是否为最短路径了。**

# 测试方法

本程序的测试方法均在Customer.java中，具体如下：

Public void showAll()

用来输出现在所有出租车的状况，挨个输出，简单粗暴

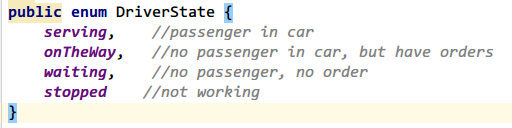
Public void showAll(DriverState driverState)

用来打印所有driverstate状态的车辆的状况

@参数 driverState 输入想要查看的车辆的状态，详细的类型请参见DriverState.java

用法例如: showAll(DriverState.***stopped***);

Driverstate的所有选项可以在DriverState.java中找到



Public void showOne(int id)

用来查看id号为id的车辆的信息，此处id应为0~99之间的整数，否则不会有输出

Public void block(int x, int y, Direction dir)

用来阻塞道路，可以实现道路的实时阻塞，当阻塞道路的总数超过5的时候将不会执行操作并提示错误。如果输入的点的坐标不合法将会输出错误信息，不执行操作。

@param x 想要阻塞点的x坐标，应在0~79之间

@param y 需要阻塞的点的y坐标，应在0~79之间

@param dir 需要阻塞的道路相对于点的坐标，详细类型请参见Direction.java

@see Direction.java

Public void rebuild(int x, int y , Direction dir)

来实时恢复被阻塞的道路，如果原地图中并不含有该道路，即道路并不是因为block()方法阻断的,那么将不会执行操作并提示错误信息。

@param x 需要恢复道路的一个端点的x坐标

@param y 需要恢复道路的同一个端点的y坐标

@param dir 需要恢复的道路相对于这个端点的位置，详细类型请参见Direction.java

@see Direction

Public void getFlow(int x, int y ,int anox, int anoy)

用来打印道路车流量的方法，输入道路的两个端点，返回道路的车流量

@param x 道路的一个端点的x坐标

@param y 道路的同一个端点的y坐标

@param anox 道路的另一个端点的x坐标

@param anoy 道理的另一个端点的y坐标

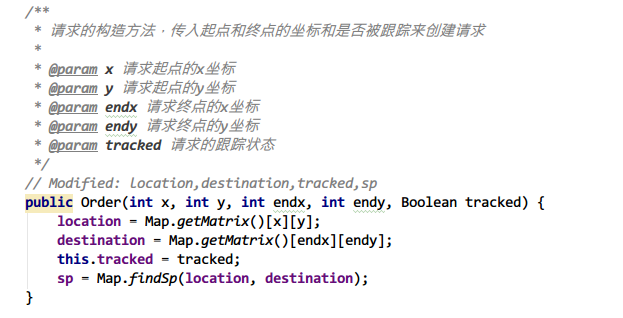
测试的时候在Customer.java中的run里面填上相应的操作，然后执行Main.java就可以实现测试了。如果把输出合理利用上，检查车辆的运行状况还是很容易的一件事儿。当然也有可能出现因为调试信息太多找不到输出的情况，所以具体怎么使用还是见仁见智吧

为了和谐六系，还请少侠手下留情



# 关于规格说明

代码的规格说明全部标注在类/方法的上方，由于原规格要求过于繁琐且不美观，本程序的规格采用Javadoc标注方法进行撰写，并辅以注释说明。如下图：



我在Javadoc注释部分，书写了Effect和Require，并详细说明了各参数的作用。由于Javadoc没有写Modified变量的tag，因此我在注释部分补写了Modified注释，方便测试者查看。此外，为了方便测试者了解各方法的用途，本人还生成了Javadoc文档，请打开javadoc/index.html来进行阅读。由于本人撰写文档功力尚浅，加之时间紧迫，故该文档可能会不尽完善，还请测试者能够给出宝贵的意见。

# 附出租车状态转换图

此状态图仅供测试者理解Taxi类的状态转换，细节条件可以有所缺省，还请谅解。



# 其他说明

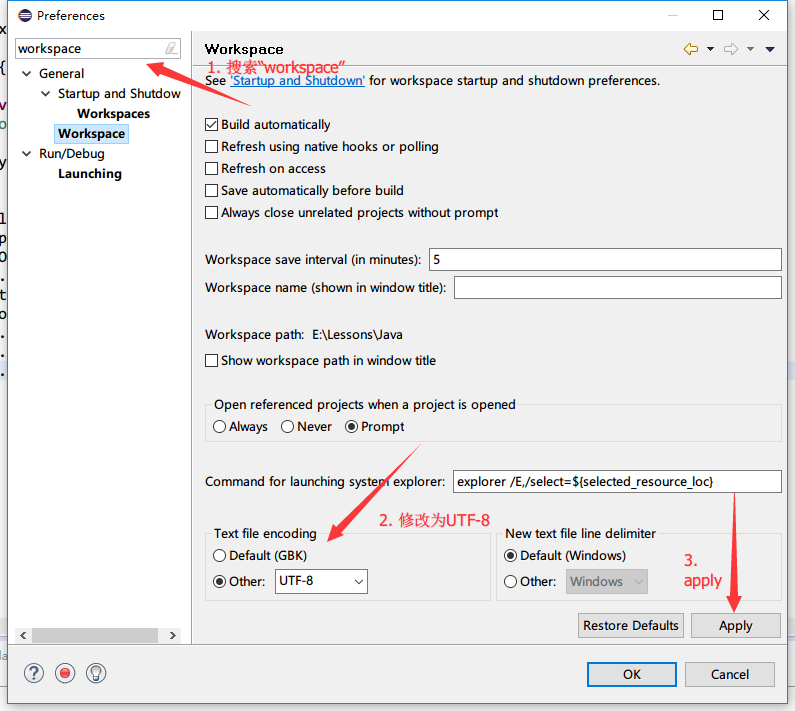
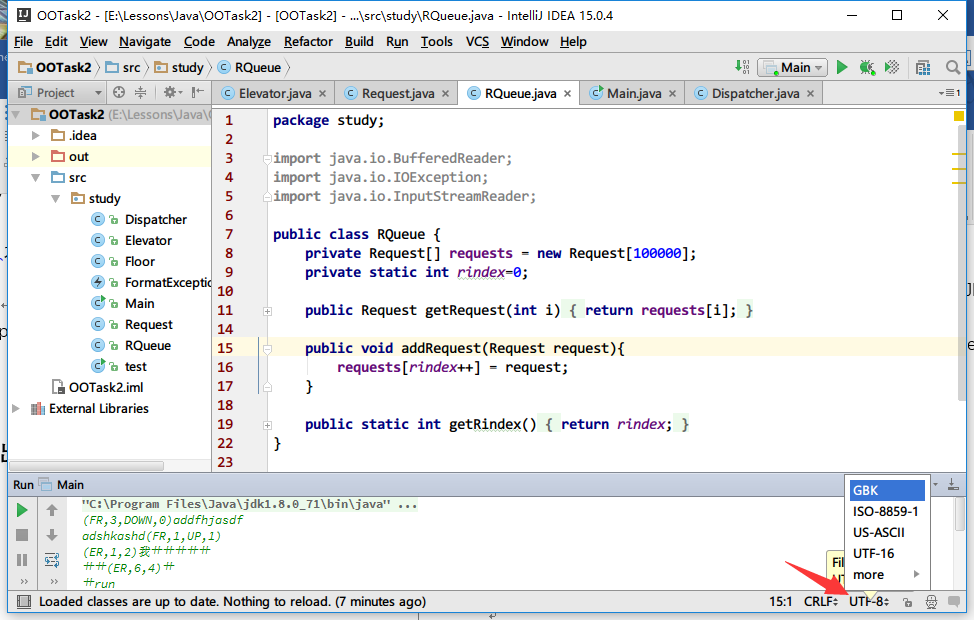
本程序的编码格式为UTF-8格式，所用IDE为Intellij IDEA，JDK版本为1.8。

IDEA导入程序只需将src整个文件夹拖入工程即可，

Eclipse导入工程需要新建一个名为study的包，并import所有.java文件即可。

如果您的JDK版本低于1.8，建议更新JAVA版本。

遇到中文乱码的情况，请调整编码设置~参照如下二图，上为eclipse，下为IDEA

**最后、感谢你对我的程序所做出的付出，恭祝学业有成！**