OO 第七次作业需求分析

一、 交互关系分析

本次作业的目标为开发出模拟出租车运行的系统，需要与用户的其他数据进行交互，主要有如下交互关系：

1．地图

地图是构建本程序的最基本的数据，需要从文件读入并保存在程序中。所以我们需要设计一个 Map 类用于保存地图数据，在其中完成文件的读入并将其转化为程序的所需要的结点信息。

2．用户请求

用户请求是出租车运行所需要的必不可少的数据信息。因此我们需要单独设计一个 Order 类用以保存请求信息。当有多个请求发生的时候，我们将会把这些请求存进一个 Order 的队列来对其进行相应。请求的产生方式有两种，其一就是通过测试线程 Customer.java 生成，这样生成的请求将会被放进请求队列等待相应，当有出租车能够相应的时候车辆该请求将会被从队列中删除；其二是当某个请求被分配给了某辆出租车之后，出租车自己将会自动生成一个从当前地点到请求起点的请求，这样就能够指导出租车向着目的地进发。

3. 出租车的运行状态

为了实现出租车运行状态的实时监控以及测试，我们需要在 Customer.java 中提供能够显示现在出租车位置的方法。

二、 整体架构分析

为了全面地模拟出打车的全过程，我们需要思考如下几个问题:

1. 地图如何构建？

   建立地图类 Map.java 来保存地图信息。其中 Map 的构造需要传入地图文件的地址。Map 类中的 buildPointSet()方法将会把输入的 80 个字符串分解成 80*80 的字符数组。之后使用 buildMatrix() 方法来将各字符保存为程序特定的保存地图结点信息的对象 Node。由于 Map 针对一次运行只有一个，因此特地将 matrix 设定为 static final，防止之后被修改。

2. Node 应该保存哪些信息？

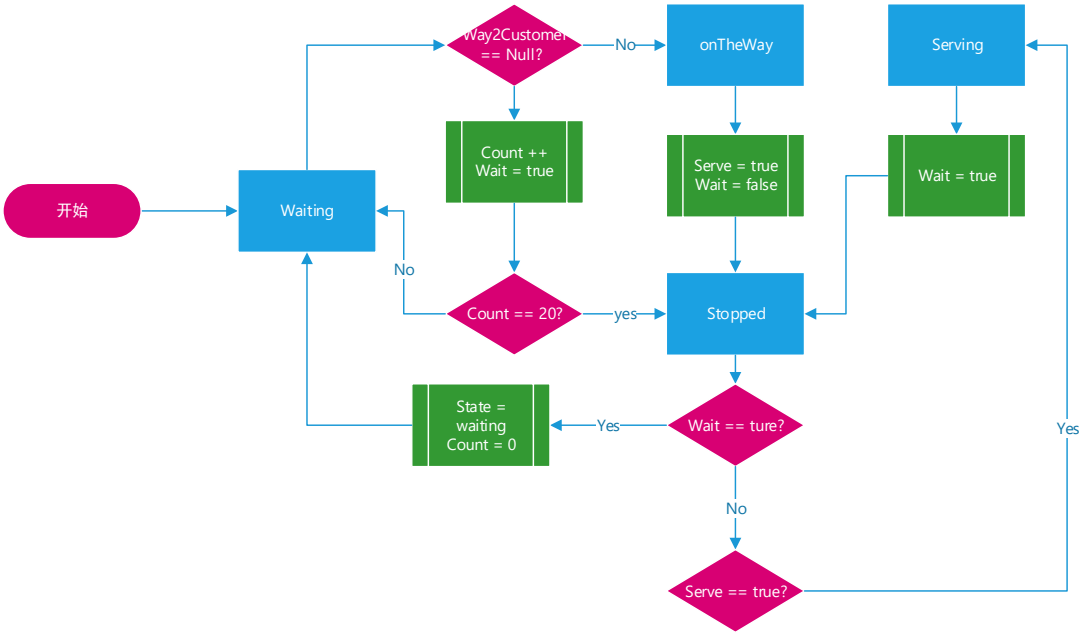
   考虑到保存邻接矩阵太过浪费空间，且不利于操作，我创建了一个 Node 类用来保存结点的信息。具体保存了一个 Point 对象用来保存结点的位置以及 4 个布尔值用来表示结点的上下左右四个方向是否与其他结点相连。.

3. 如何计算最短路径？

   其实计算最短路径的方法有很多，BFS，Dijkstra 等算法都可以求解，本人选择的是 A-Star 算法。原因是其效率较高而且在这种简单的连通的图中能够保证准确性。具

体的算法在这里不做赘述。

4. 何时计算最短路径？如何保存？
   因为本次作业的地图给定之后图的权值和形状均不会变化，因此可以在请求产生的时候就计算好从起点到终点的最短路径，并以一个 Node 队列的形式保存在 Order 里面。这样出租车在拿到请求之后只需要沿着队列中的结点行走即可，比较节省运算资源。
5. 如何初始化出租车位置？
   在构造函数中随机生成位置即可。出租车自己有一个位置属性 location 用来保存它自己现在的位置。
6. 指令如何进行分配？
   指令为单独的一个线程，在开始运行的时候他会检测自己周围 4*4 的区域内是否有出租车，如果有的话将其编号加入到队列 a 里，这种检测会持续三秒。三秒过后，指令扫描队列 a，并根据指导书的原则来找到应该分配给的车辆。并调用车辆的 setOrder()方法来将自己分配给车辆。分配后线程结束。如果三秒后队列为空，那么将会返回” No response” 表示没有车辆可以相应。
7. 出租车的运行状态如何切换？
   这个可以通过状态机来实现，所有车辆的初始状态都是 waiting，接到 Order 之后将，会把实际的请求保存在 realOrder 中，并计算从现在位置到请求起点的请求，保存为 Way2Customer 中。状态切换到 onTheWay 并开始完成 Way2Customer，完成 Way2Customer 请求后将会进入 stopped 状态并之后转入 serving 状态开始执行 realOrder。完成后输出相关信息，转入 stopped 状态并重新回到 waiting 状态。具体如下图：



三、 对象内部分析

见同文件夹下的 javadoc。

四、 类协同分析

Map 的需求关系最多，因为无论是出租车还是请求都需要地图来计算出最短路径。其次是出粗车类的需求较多，因为每个请求发生的时候都需要分析车辆现在的状态。Customer 类和 Main 类相对独立，因为测试线程只负责添加请求，和程序运行并无直接联系。

五、 并发关系分析

每个出租车都是一个线程，每个 Order 对象也是一个单独的线程。为了避免计算最短路径的时候出现冲突，将地图的计算最短路径的方法设为同步方法。为了防止同一辆车拿到两个单，在 Order 分配选择出租车的时候将会锁定 taxis 对象。

六、 时空平衡分析

在寻找最短路径的过程中，我们可能会遇到时间与空间相互取舍的问题。例如本次作业计算最小生成树有以下几种方法：

1. 车每走一步单独计算一次
2. 在构建地图的时候吧所有点对之间的最短路径都计算好，共有 6400*6400 个值。使用的时候直接取值即可。
3. 在生成请求的时候，利用请求的 3s 时间将最短路径计算出来。

本次程序最终使用的是第三种方法。

首先第一种程序的空间消耗很小，但是计算量比较大。由于要开 100+个线程，为了防止时间不足的情况，本次未能采用。第二种方法的速度很快，但是空间消耗太大，因此放弃。第三种方法能够合理的利用请求产生和需要最短路径之间的时间差，这样就能够最大化利用资源。但是缺点是如果地图的权值发生变化可能会导致结果不准确，针对此次作业暂时不会有这样的问题。