

---

# OO 第九次作业——出租调度系统说明文档

## 一、 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。其中 **dir** 对应着之前的结点邻接图，**verticalFile** 对应着各结点的交叉方式。修改位置如下：

```
22 //////////////////////////////////////////////////Please input your filepath below////////////////////////////////////  
23 String dir = "G:\\QMDownload\\1.txt";  
24 String verticalDir = "G:\\QMDownload\\2.txt";  
--
```

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：

```
"C:\Program Files\Java\jdk1.8.0_71\bin\java" ...  
Error matrix length! The matrix should be 80 * 80  
  
Process finished with exit code 0
```

对于邻接图，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：

```
"C:\Program Files\Java\jdk1.8.0_71\bin\java" ...  
Error number in matrix! The number can only be 0,1,2,3  
Error line : 32
```

对于交叉图，字符串中只能出现数字 0,1 其意义与指导书中相同。如果输入的地图中有非法字符，程序将会报错并提示错误的行号，之后结束运行，如下图：

```
"C:\Program Files\Java\jdk1.8.0_71\bin\java" ...  
Error number in vertical matrix. The number can only be 0 or 1!  
Error line : 2  
  
Process finished with exit code 0
```

地图文件的内容及更多细节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。

### 2. 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 请求起始点的横坐标
- \* @param y 请求起始点的纵坐标
- \* @param endx 请求终点的横坐标

\* @param endy 请求终点的纵坐标

\* @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。

至于跟踪的作用我们会放到后文说明。

## 二、 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 line44, 注释掉可以取消输出初始状态

```
41         for (int i = 0; i < 100; i++) {
42             taxis[i] = new Taxi(i);
43             taxis[i].start();
44             taxis[i].showstate(); //comment to disable init-state output
45         }
```

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

```
Time :18ms. Taxi 0 : location (32,56) state:waiting credit: 0
Time :19ms. Taxi 1 : location (19,62) state:waiting credit: 0
Time :19ms. Taxi 2 : location (17,50) state:waiting credit: 0
Time :19ms. Taxi 3 : location (52,24) state:waiting credit: 0
Time :19ms. Taxi 4 : location (37,17) state:waiting credit: 0
Time :19ms. Taxi 5 : location (42,54) state:waiting credit: 0
Time :19ms. Taxi 6 : location (44,47) state:waiting credit: 0
Time :19ms. Taxi 7 : location (56,51) state:waiting credit: 0
Time :19ms. Taxi 8 : location (19,73) state:waiting credit: 0
Time :19ms. Taxi 9 : location (24,18) state:waiting credit: 0
Time :20ms. Taxi 10 : location (38,45) state:waiting credit: 0
Time :20ms. Taxi 11 : location (69,13) state:waiting credit: 0
Time :20ms. Taxi 12 : location (24,7) state:waiting credit: 0
Time :20ms. Taxi 13 : location (29,29) state:waiting credit: 0
Time :20ms. Taxi 14 : location (28,12) state:waiting credit: 0
```

### 2. Order 成功添加提示信息

修改位置: Customer.java line58, 注释掉可以取消添加成功的提示

```
54     public void inputOrder(int x, int y, int endx, int endy, boolean track) throws InterruptedException {
55         if (x >= 0 && x <= 79 && y >= 0 && y <= 79 && endx >= 0 && endx <= 79 && endy >= 0 && endy <= 79) {
56             Order order1 = new Order(x, y, endx, endy, track);
57             orders.put(order1);
58             System.out.println("Input " + order1.toString());
59         } else {
60             System.out.println("Error Order input!");
61         }
```

作用: 证明请求被成功加载到相应队列中

效果：

```
Input Order : (0,0) to (79,79)
Input Order : (15,25) to (34,15)
Input Order : (5,8) to (6,7)
Input Order : (2,4) to (7,61)
Input Order : (15,7) to (51,3)
Input Order : (12,34) to (41,41)
Input Order : (0,0) to (79,79)
Input Order : (69,64) to (52,74)
```

### 3. 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：

```
No Response : Order : (15,25) to (34,15)
No Response : Order : (12,34) to (41,41)
No Response : Order : (15,7) to (51,3)
Taxi : ID 64 take Order : (0,0) to (79,79)
Taxi : ID 54 take Order : (5,8) to (6,7)
Taxi : ID 83 take Order : (69,64) to (52,74)
```

### 4. 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：

```
Taxi 60: coming to passenger Order : (70,62) to (69,64)
```

```
Taxi 60: serving Order : (69,64) to (52,74)
```

```
Taxi 97: moving (66,15) @16670ms
```

```
Taxi 97: moving (65,15) @16770ms
```

```
Taxi 97: serving finished
```

```
Time :16770ms. Taxi 97 : location (65,15) state:stopped credit: 4
```

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当前位置，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

### 5. 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：

```
InOrder : (15,25) to (34,15) list: Time :3306ms. Taxi 98 : location (14,26) state:waiting credit: 2
InOrder : (15,25) to (34,15) list: Time :3306ms. Taxi 86 : location (15,26) state:waiting credit: 1
InOrder : (15,25) to (34,15) list: Time :3306ms. Taxi 9 : location (16,23) state:waiting credit: 1
Taxi : ID 91 take Order : (2,4) to (7,61)
Taxi : ID 98 take Order : (15,25) to (34,15)
```

车辆的具体行走路径

---

```
Taxi 97: moving (16,53) @3548ms
Taxi 23: coming to passenger Order : (2,2) to (0,5)
Taxi 23: moving (3,3) @3601ms
Taxi 97: moving (15,53) @3652ms
Taxi 23: moving (3,4) @3701ms
Taxi 23: moving (3,5) @3802ms
Taxi 23: moving (3,6) @3903ms
Taxi 23: moving (3,7) @4003ms
Taxi 23: moving (3,8) @4283ms
Taxi 23: moving (2,8) @4383ms
Taxi 23: moving (2,9) @4484ms
Taxi 23: moving (2,10) @4584ms
Taxi 23: moving (1,10) @4684ms
Taxi 97: serving Order : (15,52) to (65,15)
Taxi 97: moving (16,52) @4849ms
Taxi 23: moving (1,9) @4850ms
Taxi 97: moving (16,51) @4989ms
Taxi 23: moving (1,8) @4989ms
Taxi 97: moving (16,50) @5176ms
Taxi 23: moving (1,7) @5176ms
Taxi 23: moving (1,6) @5331ms
Taxi 23: moving (1,5) @5431ms
Taxi 97: moving (16,49) @5551ms
Taxi 97: moving (17,49) @5781ms
Taxi 97: moving (18,49) @6002ms
Taxi 97: moving (19,49) @6215ms
Taxi 97: moving (20,49) @6413ms
Taxi 23: serving Order : (0,5) to (0,0)
Taxi 23: moving (1,5) @6586ms
Taxi 97: moving (20,50) @6618ms
Taxi 23: moving (1,6) @6686ms
Taxi 97: moving (21,50) @6768ms
Taxi 23: moving (1,7) @6786ms
Taxi 97: moving (21,49) @6905ms
Taxi 23: moving (1,8) @6905ms
```

个人认为这样足以让测试者判别是否为最短路径了。

### 三、 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 enum DriverState {  
    serving,      //passenger in car  
    onTheWay,     //no passenger in car, but have orders  
    waiting,      //no passenger, no order  
    stopped      //not working  
}
```

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 道路的同个端点的 y 坐标

@param anox 道路的另一个端点的 x 坐标

@param anoy 道路的另一个端点的 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



---

## 四、 红绿灯说明

红绿灯控制在TrafficLight类中通过单独线程实现,所有路口的红绿灯都由此线程控制。起始状态所有路口都是南北通行,每过 300ms 就会变化一次通行方向。本质上是所有路口共用一个红绿灯。

## 五、 关于规格说明

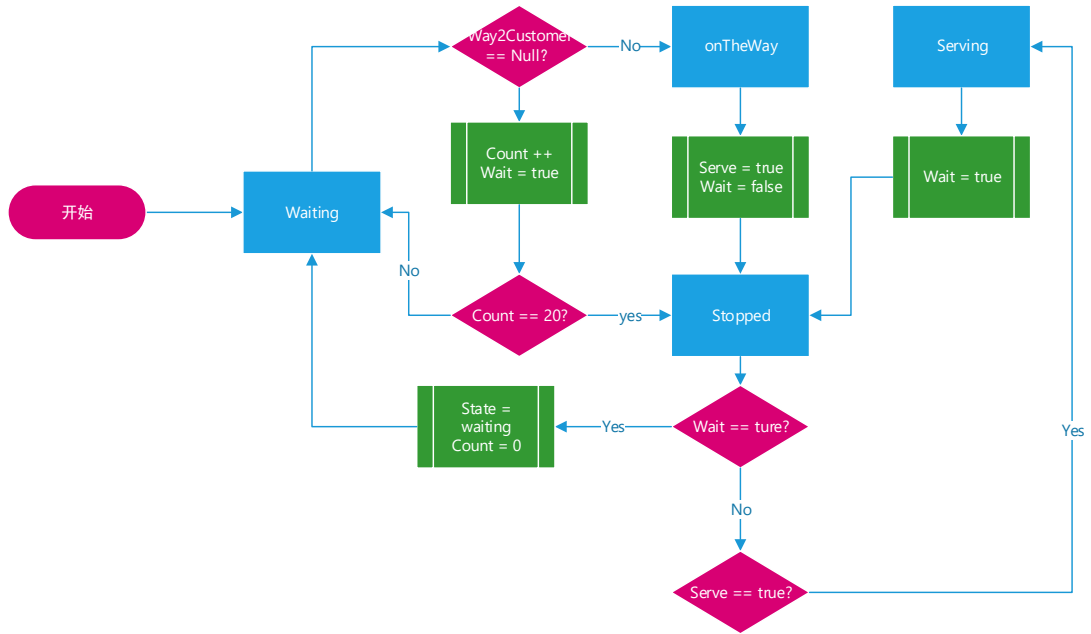
代码的规格说明全部标注在类/方法的上方,由于原规格要求过于繁琐且不美观,本程序的规格采用 Javadoc 标注方法进行撰写,并辅以注释说明。如下图:

```
/**
 * 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
 *
 * @param x 请求起点的x坐标
 * @param y 请求起点的y坐标
 * @param endx 请求终点的x坐标
 * @param endy 请求终点的y坐标
 * @param tracked 请求的跟踪状态
 */
// Modified: Location,destination,tracked,sp
public Order(int x, int y, int endx, int endy, Boolean tracked) {
    location = Map.getMatrix()[x][y];
    destination = Map.getMatrix()[endx][endy];
    this.tracked = tracked;
    sp = Map.findSp(location, destination);
}
```

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。

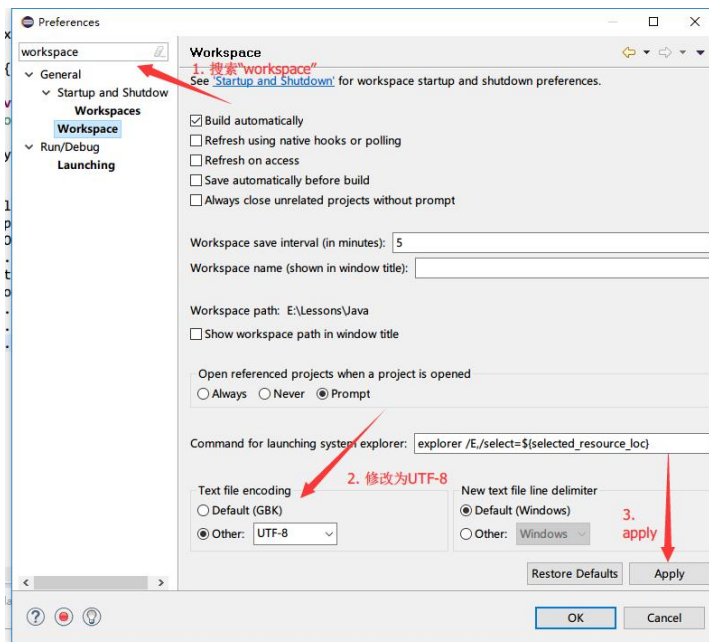
## 六、 附出租车状态转换图

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。

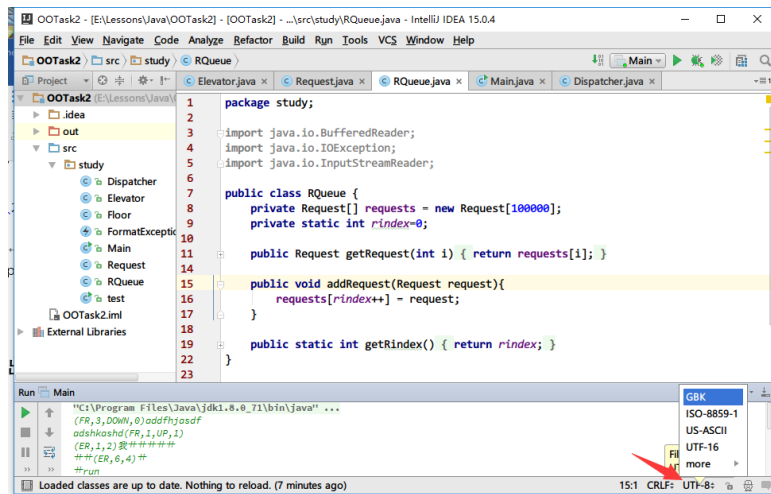


## 七、 其他说明

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







## 附、Javadoc 生成教程

总觉得 OO 课缺少彼此交流的方式，本着能够相互学习的原则，我还是希望能够在这一节总结一下我所学到的东西。(如果你已经是大神了请忽略此节=w=)

个人感觉 javadoc 是用来介绍程序的最好方式，也是 java 自带的用来生成文档的工具。一般 IDE 通过输入 “/\*\*” + “回车” 可以生成 javadoc 代码块

```
/**
 * 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
 * 当输入不合法时将不会正常工作
 *
 * @param x 请求起始点的横坐标
 * @param y 请求起始点的纵坐标
 * @param endx 请求终点的横坐标
 * @param endy 请求终点的纵坐标
 * @param track 是否跟踪该请求
 * @throws InterruptedException 只要少使输入的请求合法就不会出错·嗯=w=
 */
```

其中有几个典型的 tag 可以用来进行参数的标识，常用的有如下标签

- @author 作者
- @version 版本
- @see 参考转向
- @param 参数说明
- @return 返回值说明
- @exception 异常说明

最上面无标签的就是方法的概要(Overview)，会出现在文档的方法概要中



所有方法	实例方法	具体方法
限定符和类型		方法和说明
void		<b>block</b> (int x, int y, <b>Direction</b> 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。
boolean		<b>customerRepOk</b> () customer类的不变式
void		<b>getFlow</b> (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
void		<b>inputOrder</b> (int x, int y, int endx, int endy, boolean track) 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
void		<b>rebuild</b> (int x, int y, <b>Direction</b> dir) 用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。
void		<b>run</b> () 测试的主线程
void		<b>showAll</b> () 用来查看现在所有出租车的状况，挨个输出，简单粗暴
void		<b>showAll</b> ( <b>DriverState</b> driverState) 用来打印所有driverstate状态的车辆的情况
void		<b>showOne</b> (int id) 用来打印某一辆出租车的信息，注意出租车编号的合法性

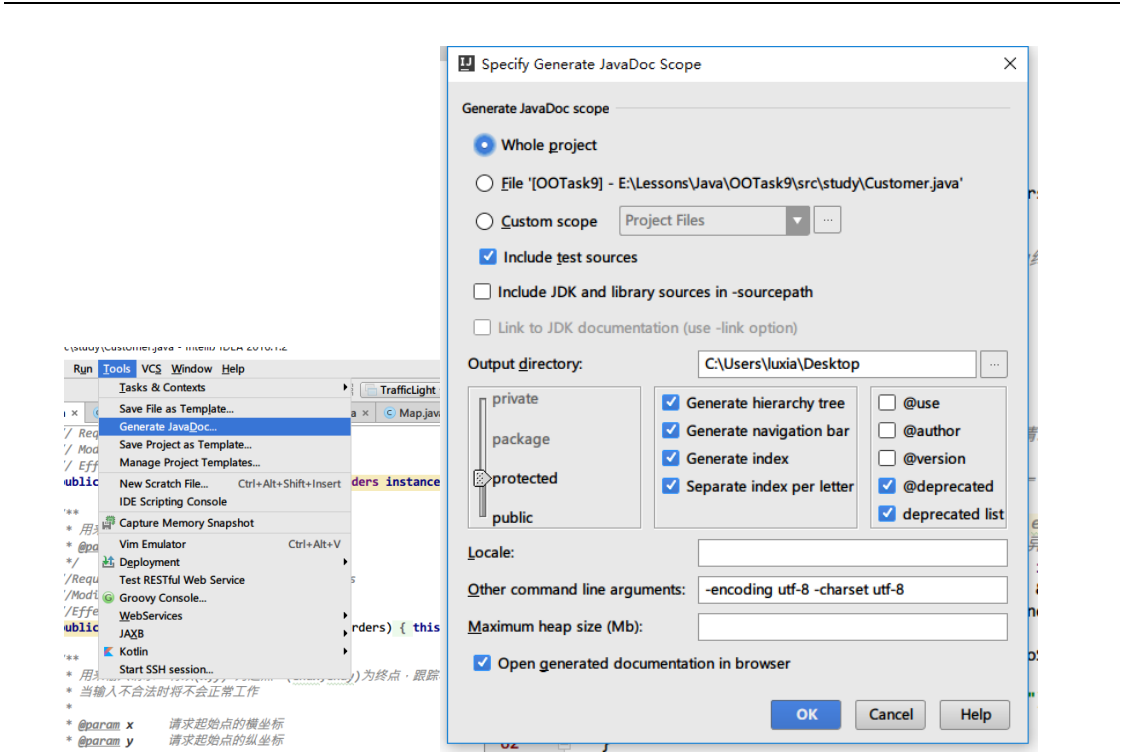
点进去便能够看到更加具体的方法说明

inputOrder
<pre>public void inputOrder(int x,                       int y,                       int endx,                       int endy,                       boolean track)     throws java.lang.InterruptedException</pre>
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
<b>参数:</b> x - 请求起始点的横坐标 y - 请求起始点的纵坐标 endx - 请求终点的横坐标 endy - 请求终点的纵坐标 track - 是否追踪该请求
<b>抛出:</b> java.lang.InterruptedException - 只要少快输入的请求合法就不会出错，嗯=w=

生成文档的方法也很简单，我使用的是 IDEA，只需要在写好文档注释之后点击 Tools-Generate Javadoc,之后选择路径即可.eclipse 平台应该也大同小异

需要注意的是, 如果注释中有中文, 则需要添加额外的指令, 否则可能会出现错误, 指令如下：

-encoding utf-8 -charset utf-8



之后通过打开 index.html 可以打开文档，如果注释写的比较全的话真的很 nice



如果你对 javadoc 有更多的了解，或者有其他值得分享的姿势，也欢迎在申诉区与我交流~（怎么感觉怪怪的 2333

最后、感谢你对我的程序所做出的付出，恭祝学业有成！