Oo第十次作业设计分析文档

PS:关于不变式部分，我也写过类似PPT上那种的，但是感觉很不方便看，我就把程序中写的repOK方法粘上来了，没再改变它的格式，原理上都相同，当满足该类的所有属性条件的时候才会返回true，否则会返回false。

**Center类：**

**Overview：**

center类是整个工程的主类，负责在程序的最外层进行读入map文件和light文件，以及启动taxi（出租车）、taxisearch(乘客搜索出租车)、test（测试）、trafficlight（交通信号）、inputhandler(路径查询)线程的操作。

**过程规格：**

**public** Center()

{

//EFFECTS: 初始化center中保存的数据

**public** **void** setLight()

{

//REQUIRES：读入的templight文件仅由0,1,2,3组成

//MODIFIES: 红绿灯数组light

//EFFECTS: 将读入的templight进行处理，初始化红绿灯数组；

**public** **void** setMap()

{

//REQUIRES：读入的tempmap文件联通，且仅由0,1,2,3组成

//MODIFIES: 地图文件map

//EFFECTS: 将读入的tempmap进行处理，变为各条边的联通状态，联通为true否则为false；

**public** **int** getLen(Point start,Point end)

{

//REQUIRES：读入的起点和终点坐标在0-79范围内

//EFFECTS: 计算起点和终点之间的最短路径距离，如果不存在返回99999；

**public** ArrayList<Point> getChildren(Point p)

{

//REQUIRES：读入的Point p坐标在范围内

//EFFECTS: 获取读入周边四个节点中与之联通的节点，并返回一个节点的ArrayList

**public** Point getNext(Point p,Point fp)

{

//REQUIRES：读入的当前位置p和前一步位置fp在范围内，且p和fp之间距离为1

//MODIFIES: 车流统计taxiflow

//EFFECTS: 当出租车处于waiting状态时，利用该方法获取下一步位置，满足指导书要求中的选择车流最小的边行驶

**public** Point getShortNext(Point p,Point fp,Point end)

{

//REQUIRES：读入的当前位置p和前一步位置fp还有最终位置end在范围内，且p和fp之间距离为1

//MODIFIES: 车流统计taxiflow

//EFFECTS: 当出租车处于serving和picking状态时，利用该方法获取下一步位置，满足指导书要求中的选择

//路径最短中车流最小的边行驶的要求

**public** **int** getLenAdvance(Point start,Point end)

{

//REQUIRES：读入的起点和终点坐标在0-79范围内

//EFFECTS: 计算起点和终点之间的最短路径距离，如果不存在返回99999；

**public** ArrayList<Point> getChildrenAdvance(Point p)

{

//REQUIRES：读入的Point p坐标在范围内

//EFFECTS: 获取读入周边四个节点中与之联通的节点，并返回一个节点的ArrayList

**public** Point getNextAdvance(Point p,Point fp)

{

//REQUIRES：读入的当前位置p和前一步位置fp在范围内，且p和fp之间距离为1

//MODIFIES: 车流统计taxiflow

//EFFECTS: 当出租车处于waiting状态时，利用该方法获取下一步位置，满足指导书要求中的选择车流最小的边行驶

**public** Point getShortNextAdvance(Point p,Point fp,Point end)

{

//REQUIRES：读入的当前位置p和前一步位置fp还有最终位置end在范围内，且p和fp之间距离为1

//MODIFIES: 车流统计taxiflow

//EFFECTS: 当出租车处于serving和picking状态时，利用该方法获取下一步位置，满足指导书要求中的选择

//路径最短中车流最小的边行驶的要求

**public** **synchronized** **void** addFlow(Point f,Point l)

{

//REQUIRES：读入的起始位置f和目标位置l在范围内,且两点之间距离为1

//MODIFIES: 车流统计taxiflow

//EFFECTS: 将起始位置和目标位置之间的边车流统计+1

**public** **synchronized** **void** removeFlow(Point f,Point l)

{

//REQUIRES：读入的起始位置f和目标边位置l在范围内,且两点之间距离为1

//MODIFIES: 车流统计taxiflow

//EFFECTS: 将起始位置和目标位置之间的边车流统计-1

**public** **int** getFlow(Point f,Point l)

{

//REQUIRES：读入的起始位置f和目标边位置l在范围内,且两点之间距离为1

//EFFECTS: 获取起始位置和目标位置之间的边车流统计

**public** **void** updatePosition(**int** no,Point p)

{

//REQUIRES：读入的位置p在范围内,且100>no>=0

//MODIFIES: 车辆位置统计taxi\_position

//EFFECTS: 更新center中存储的车辆位置信息

**public** **void** updateCredit(**int** no,**int** credit)

{

//REQUIRES：读入的位置p在范围内,且100>no>=0

//MODIFIES: 车辆位置统计taxi\_credit

//EFFECTS: 更新center中存储的车辆信用信息

**public** **void** updateMode(**int** no,**int** mode)

{

//REQUIRES：读入的位置p在范围内,且100>no>=0

//MODIFIES: 车辆状态统计taxi\_mode

//EFFECTS: 更新center中存储的车辆状态信息

**public** **boolean** checkContains(ArrayList<Point> list,Point p)

{

//REQUIRES：p非空

//EFFECTS: 查找list中是否含有p

**public** **int** getCredit(**int** no)

{

//REQUIRES：100>no>=0

//EFFECTS: 获取指定车辆的信用信息

**public** **int** getMode(**int** no)

{

//REQUIRES：100>no>=0

//EFFECTS: 获取指定车辆的状态信息

**public** Point getPosition(**int** no)

{

//REQUIRES：100>no>=0

//EFFECTS: 获取指定车辆的位置信息

**public** **void** closeRoad(Point f,Point l)

{

//REQUIRES：道路的起始位置f和结束位置l在范围内,且两点之间距离为1

//MODIFIES: 地图数据map,红绿灯数据light

//EFFECTS: 关闭两点之间的道路，重新设置红绿灯

**public** **void** openRoad(Point f,Point l)

{

//REQUIRES：道路的起始位置f和结束位置l在范围内,且两点之间距离为1

//MODIFIES: 地图数据map,红绿灯数据light

//EFFECTS: 打开两点之间的道路,重新设置红绿灯

**表示对象：**

**private** **static** **boolean** *map*[][][]=**new** **boolean** [80][80][4];

**private** **static** **boolean** *originalmap*[][][]=**new** **boolean** [80][80][4];

**private** **static** **int** *taxiflow*[][][]=**new** **int** [80][80][4];

**private** **static** String[][] *tempmap*=**new** String [80][80];

**private** **static** String[][] *templight*=**new** String [80][80];

**private** **static** TrafficLight[][] *light*=**new** TrafficLight [80][80];

**public** **static** **final** **int** ***UP***=0;

**public** **static** **final** **int** ***RIGHT***=1;

**public** **static** **final** **int** ***DOWN***=2;

**public** **static** **final** **int** ***LEFT***=3;

**public** **static** **final** **int** ***MAX\_TAXI***=100;

**private** **static** **final** **int** ***SERVING***=0;

**private** **static** **final** **int** ***PICKING***=1;

**private** **static** **final** **int** ***WAITING***=2;

**private** **static** **final** **int** ***RESTING***=3;

**private** **static** **int** *taxi\_credit*[]=**new** **int** [***MAX\_TAXI***];

**private** **static** Point *taxi\_position*[]=**new** Point[***MAX\_TAXI***];

**private** **static** **int** *taxi\_mode*[]=**new** **int** [***MAX\_TAXI***];

**抽象函数：**

AF(c) = {map,taxiflow,tempmap,templight,light,taxi\_credit,taxi\_position,taxi\_mode}

Where map = c.map, taxiflow=c.taxiflow, tempmap=c. tempmap,templight=c. templight,light=c. light,taxi\_credit=c. taxi\_credit,taxi\_position=c. taxi\_position,taxi\_mode=c. taxi\_mode

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**for**(**int** i=0;i<80;i++)

{

**for**(**int** j=0;j<80;j++)

{

**for**(**int** k=0;k<4;k++)

{

**if**(*taxiflow*[i][j][k]<0)

**return** **false**;

}

**if**(!(*tempmap*[i][j].equals("0")||*tempmap*[i][j].equals("1")||*tempmap*[i][j].equals("2")||*tempmap*[i][j].equals("3")))

**return** **false**;

**if**(!(*templight*[i][j].equals("0")||*templight*[i][j].equals("1")))

**return** **false**;

**if**(*light*[i][j]==**null**)

**return** **false**;

**if**(!*light*[i][j].repOK())

**return** **false**;

}

}

**for**(**int** i=0;i<100;i++)

{

**if**(*taxi\_credit*[i]<0)

**return** **false**;

**if**(*taxi\_position*[i]==**null**)

**return** **false**;

**if**(!*taxi\_position*[i].repOK())

**return** **false**;

**if**(*taxi\_mode*[i]>3||*taxi\_mode*[i]<0)

**return** **false**;

}

**return** **true**;

}

**InputHandler类：**

**Overview：**

Inputhandler是输入处理类，处理输入的查询请求

**过程规格：**

**public** **void** parse(String s,**int** n)

{

//REQUIRES：读入的String非空，n>0

//MODIFIES: 字符串s

//EFFECTS: 将读入的字符串进行处理，返回查询结果；

**表示对象：**

**Private LinkedList<Taxi> taxis**

**抽象函数：**

AF(c) = { taxis }

Where taxis=c. taxis

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(taxis==**null**)

**return** **false**;

**return** **true**;

}

**Passenger类：**

**Overview：**

passenger类是乘客类，负责通过构造器传入的坐标生成乘客，同时记录当前系统时间，同时本类中也保存了抢到该乘客单的出租车。

**过程规格：**

**public** Passenger(**int** no,Point c,Point d,Center center)

{

//REQUIRES：c,d在地图范围内

//EFFECTS: 根据给出的位置初始化乘客

**public** **void** addTaxi(Taxi t)

{

//REQUIRES：t非空

//MODIFIES: 抢单出租车列表availabletaxi，出租车信用credit

//EFFECTS: 向乘客对象中保存抢单成功的出租车，并将抢单成功的出租车信用加1

**public** Taxi getTaxi()

{

//MODIFIES: 抢单出租车列表availabletaxi

//EFFECTS: 从抢单成功的出租车中按照指导书要求选择最终接载乘客的出租车

**public** Point getCallPoint()

{

//EFFECTS: 返回乘客的叫车位置

**public** Point getDesPoint()

{

//EFFECTS: 返回乘客的目的位置

**public** **long** getTime()

{

//EFFECTS: 返回乘客的叫车时间

**表示对象：**

**private** **static** **final** **int** ***SERVING***=0;

**private** **static** **final** **int** ***PICKING***=1;

**private** **static** **final** **int** ***WAITING***=2;

**private** **static** **final** **int** ***RESTING***=3;

**private** Point call\_point;

**private** Point des\_point;

**private** Center center;

**private** **int** no;

**private** **long** time;

**private** HashMap<Integer,Taxi> availabletaxi=**new** HashMap<Integer,Taxi>();

**抽象函数：**

AF(c) = {call\_point,des\_point,center,no,time,availabletaxi }

Where call\_point=c. call\_point,des\_point=c. des\_point,center=c.center,no=c.no,time=c.time,availabletaxi=c. availabletaxi

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(call\_point==**null**)

**return** **false**;

**if**(!call\_point.repOK())

**return** **false**;

**if**(des\_point==**null**)

**return** **false**;

**if**(!des\_point.repOK())

**return** **false**;

**if**(center==**null**)

**return** **false**;

**if**(!center.repOK)

**return** **false**;

**if**(no<0)

**return** **false**;

**if**(time<0)

**return** **false**;

**if**(availabletaxi==**null**)

**return** **false**;

**return** **true**;

}

**PassengeQueue类：**

**Overview：**

passengequeue类是乘客队列类，新生成的乘客将会被加入队列

**过程规格：**

**public** **synchronized** **void** add(Passenger p)

{

//REQUIRES：p非空

//MODIFIES: 乘客列表passenger\_list

//EFFECTS: 向乘客列表中添加乘客

**public** **synchronized** **int** getSize()

{

//EFFECTS: 返回乘客列表的长度

**return** **this**.passenger\_list.size();

}

**public** **synchronized** Passenger get(**int** n)

{

//REQUIRES：乘客列表中有第n个成员

//EFFECTS: 返回乘客列表中的第n个成员

**public** **synchronized** Passenger remove(**int** n)

{

//REQUIRES：乘客列表中有第n个成员

//MODIFIES: 乘客列表passenger\_list

//EFFECTS: 返回乘客列表中的第n个成员并删除之

**public** **synchronized** **void** clear()

{

//MODIFIES: 乘客列表passenger\_list

//EFFECTS: 清空乘客列表

**public** **synchronized** **boolean** isEmpty()

{

//EFFECTS: 返回乘客列表是否为空

**表示对象：**

**LinkedList<Passenger> passenger\_list=new LinkedList<Passenger>();**

**抽象函数：**

AF(c) = {passenger\_list }

Where passenger\_list=c. passenger\_list

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(passenger\_list==**null**)

**return** **false**;

**for**(Passenger p:passenger\_list)

{

**if**(!p.repOK())

**return** **false**;

}

**return** **true**;

}

**Point类：**

**Overview：**

Point类是抽象表示的地图上的路口，存储了坐标信息

**过程规格：**

**public** Point(**int** x,**int** y)

{

//REQUIRES：x，y都大于等于0小于80

//EFFECTS: 根据输入的x，y值初始化

**public** String toString()

{

//EFFECTS:将该点坐标返回为字符串形式

**public** **boolean** equals(Point a)

{

//REQUIRES：a非空

//EFFECTS: 如果两点相同，返回true，反之返回false

**public** **int** getX()

{

//EFFECTS: 返回该点x坐标

**public** **int** getY()

{

//EFFECTS: 返回该点y坐标

**表示对象：**

**private** **int** x;

**private** **int** y;

**抽象函数：**

AF(c) = {x,y}

Where x=c.x,y=y.y;

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(x>=80||x<0)

**return** **false**;

**if**(y>=80||y<0)

**return** **false**;

**return** **true**;

}

**PathPoint类：**

**Overview：**

PathPoint类是封装了一个point点和一个int数，代表服务次数

**过程规格：**

**public** PathPoint(**int** no,**int** count,Point p)

{

//REQUIRES：no大于等于0小于100，count大于0，p非空

//EFFECTS: 根据输入的值初始化

**public** **int** getServeCount()

{

//EFFECTS: 返回当前节点存储的服务编号

**public** Point getPoint()

{

//EFFECTS: 返回当前节点存储的路径节点

**public** String toString()

{

//EFFECTS: 返回当前节点的字符串表示

**表示对象：**

**private** Point p;

**private** **int** CarNo;

**private** **int** ServeCount;

**抽象函数：**

AF(c) = {p,CarNo,ServeCount}

Where p=c.p,CarNo=c.CarNo,ServeCount=c.ServeCount

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(p==**null**)

**return** **false**;

**if**(CarNo<0||CarNo>=100)

**return** **false**;

**if**(ServeCount<0)

**return** **false**;

**return** **true**;

}

**Taxi类：**

**Overview：**

Taxi类是出租车类，保存了出租车的信息，将会根据不同的模式进行运动

**过程规格：**

**public** Taxi (**int** number,Center center)

{

//REQUIRES：number大于等于0小于100

//EFFECTS: 初始化出租车，并随机选择一个初始位置

**public** **void** update()

{

//REQUIRES：当前出租车号大于等于0小于100

//MODIFIES: 信用统计taxi\_credit,状态统计taxi\_mode,位置taxi\_position

//EFFECTS: 更新center中储存的出租车信息

**public** Point getPosition()

{

//EFFECTS: 返回出租车当前位置

**public** **int** getNum()

{

//EFFECTS: 返回出租车编号**public** **boolean** equals(Point a)

{

//REQUIRES：a非空

//EFFECTS: 如果两点相同，返回true，反之返回false

**public** **int** getCredit()

{

//EFFECTS: 返回出租车信用

**public** **int** getMode()

{

//EFFECTS: 返回出租车运行模式

**public** **void** setMode(**int** mode)

{

//REQUIRES：模式是0-3之间的整数

//MODIFIES: 出租车模式mode

//EFFECTS: 设置当前出租车运行模式

**public** **void** setStart(Point p)

{

//REQUIRES：p在地图范围内

//MODIFIES: 乘客位置start

//EFFECTS: 设置当前出租车接载乘客位置

**public** **void** setEnd(Point p)

{

//REQUIRES：p在地图范围内

//MODIFIES: 乘客目的地end

//EFFECTS: 设置当前出租车目的地

**public** **void** addCredit(**int** c)

{

//REQUIRES：c为1或3

//MODIFIES: 出租车信用credit

//EFFECTS: 设置当前出租车信用

**表示对象：**

**private** Point position;

**private** Point formerposition;

**private** Point start;

**private** Point end;

**private** **int** no;

**private** **int** mode;

**private** **int** credit;

**private** Center center;

**private** **static** **final** **int** ***SIZE***=80;

**private** **static** **final** **int** ***SERVING***=0;

**private** **static** **final** **int** ***PICKING***=1;

**private** **static** **final** **int** ***WAITING***=2;

**private** **static** **final** **int** ***RESTING***=3;

**private** **int** waiting\_count;**抽象函数：**

AF(c) = {position,formerposition,start,end,no,mode,credit,center,waiting\_count}

Where position=c.position,formerposition=c. formerposition,start=c.start,end=c.end,no.c=no,mode.c,mode,credit=c.credit,center=c.center,waiting\_count=c. waiting\_count

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(position==**null**)

**return** **false**;

**if**(!position.repOK())

**return** **false**;

**if**(formerposition==**null**)

**return** **false**;

**if**(!formerposition.repOK())

**return** **false**;

**if**(!start.repOK())

**return** **false**;

**if**(!end.repOK())

**return** **false**;

**if**(no<0||no>=100)

**return** **false**;

**if**(mode<0||mode>3)

**return** **false**;

**if**(credit<0)

**return** **false**;

**if**(!center.repOK)

**return** **false**;

**if**(waiting\_count<0||waiting\_count>20)

**return** **false**;

**return** **true**;

}

**TaxiAdvance类：**

**Overview：**

Taxi类是超级出租车类，保存了出租车的信息，将会根据不同的模式进行运动

**过程规格：**

**public** TaxiAdvance(**int** number, Center center) {

//REQUIRES：number大于等于0小于100

//EFFECTS: 初始化出租车，并随机选择一个初始位置

**public** LinkedList<PathPoint> getHistoryPath()

{

//EFFECTS: 返回当前出租车服务路径

**public** **int** getServeCount()

{

//EFFECTS: 设置当前出租车服务次数

**public** TwoWayIterator PathAtIndex(**int** index)

{

//EFFECTS: 返回迭代器

**public** **boolean** isAdvance()

{

//EFFECTS: 返回当前出租车是否是超级出租

TaxiGen(TaxiAdvance it,**int** serveno)

{

//REQUIRES：读入的it非空，serveno在范围内

//EFFECTS: 生成器；

**public** **boolean** hasNext()

{

//EFFECTS: 判断是否有下一个节点

**public** **boolean** hasPrevious()

{

//EFFECTS: 判断是否有上一个节点

**public** Object next() **throws** Exception

{

//EFFECTS: 返回下一个节点

**public** Object previous() **throws** Exception

{

//EFFECTS: 返回上一个节点

**表示对象：**

**private** **int** serve\_count;

**private** LinkedList<PathPoint> HistoryPath=**new** LinkedList<PathPoint>();**抽象函数：**

AF(c) = {serve\_ count,HistoryPatj}

Where serve\_count=c. serve\_count HistoryPath=c.HistoryPath

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(!**super**.repOK())

**return** **false**;

**if**(serve\_count<0)

**return** **false**;

**if**(HistoryPath==**null**)

**return** **false**;

**return** **true**;

}

**TaxSearch类：**

**Overview：**

Taxisearch类是乘客搜索出租车了，模拟了抢单时间窗，并在时间窗结束后根据指导书规则选择出租车

**过程规格：**

**public** TaxiSearch(PassengerQueue List,LinkedList<Taxi>taxis)

{

//REQUIRES：乘客列表，出租车列表非空

//EFFECTS: 初始化对象内保存的乘客和出租车列表

**表示对象：**

**private** **static** **final** **int** ***SERVING***=0;

**private** **static** **final** **int** ***PICKING***=1;

**private** **static** **final** **int** ***WAITING***=2;

**private** **static** **final** **int** ***RESTING***=3;

**private** PassengerQueue passengerList;

**private** LinkedList<Taxi> taxis;

**抽象函数：**

AF(c) = {passengerList,taxis }

Where passengerList=c. passengerList ,taxis=c.taxis

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(passengerList==**null**)

**return** **false**;

**if**(!passengerList.repOK())

**return** **false**;

**if**(taxis==**null**)

**return** **false**;

**if**(taxis.size()!=100)

**return** **false**;

**for**(Taxi t:taxis)

{

**if**(!t.repOK())

**return** **false**;

}

**return** **true**;

}

**Test类：**

**Overview：**

Test类是生成乘客类，模拟随机生成乘客

**过程规格：**

**public** Test(PassengerQueue passengerList,Center center)

{

//REQUIRES：乘客队列非空，center存在

//EFFECTS: 初始化乘客队列

**表示对象：**

**private** **static** **final** **int** ***SLEEP\_INTERVAL***=500;

**private** **static** **final** **int** ***SIZE***=80;

**private** **static** **final** **int** ***PassengerAmount***=100;

**private** PassengerQueue passengerList;

**private** Center center;

**抽象函数：**

AF(c) = {passengerList,center }

Where passengerList=c. passengerList ,center=c.center

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(passengerList==**null**)

**return** **false**;

**if**(!passengerList.repOK())

**return** **false**;

**if**(center==**null**)

**return** **false**;

**if**(!center.repOK)

**return** **false**;

**return** **true**;

}

**TrafficLight类：**

**Overview：**

TrafficLight类是红绿灯类，每隔指定时间变换一次颜色，并且承担了阻塞交通的作用

**过程规格：**

**public** TrafficLight(**int** x,**int** y,**boolean** enable)

{

//REQUIRES：x,y大于等于0小于80

//EFFECTS: 初始化红绿灯，将初始灯设置为南北绿灯

**boolean** checkInOut(**int** in,**int** out)

{

//REQUIRES：读入的in和out是0-3之间的整数

//EFFECTS: 返回当前路径是否被红绿灯阻塞

**boolean** chechLight(Point former,Point now,Point next)

{

//REQUIRES：读入的当前位置now和前一步位置former还有下一步位置next在范围内，且former,now和next之间距离为1

//且now位置与红绿灯位置相等

//EFFECTS: 返回当前路径是否被红绿灯阻塞

**public** **void** setEnable(**boolean** enable)

{

//MODIFIES: 红绿灯开启使能enable

//EFFECTS: 开关红绿灯

**表示对象：**

**private** Point position;

**private** **int** mode;

**private** **boolean** enable;

**private** **static** **final** **int** ***UP\_DOWN***=0;

**private** **static** **final** **int** ***LEFT\_RIGHT***=1;

**public** **static** **final** **int** ***UP***=0;

**public** **static** **final** **int** ***RIGHT***=1;

**public** **static** **final** **int** ***DOWN***=2;

**public** **static** **final** **int** ***LEFT***=3;

**抽象函数：**

AF(c) = {position,mode,enable}

Where position=c.position,mode=c.mode,enable=c.enable

**不变式：**

**public** **boolean** repOK()

{

//Effects: returns true if the rep variant holds for this, otherwise returns false

**if**(position==**null**)

**return** **false**;

**if**(!position.repOK())

**return** **false**;

**if**(mode>1||mode<0)

**return** **false**;

**return** **true**;

}

关于LSP原则

在本次程序中，只有超级出租车类TaxiAdvance继承了出租车类Taxi的全部,其中仅重写了两个方法run()和isAdvance(),并新增了一些方法，可以很容易的看出，对于run()和isAdvance（）的重写并没有加强父类型方法规定的requires，并且新增的方法加强了effects，使得所有父类型对象出现的地方都可以使用子类型对象来替换。

在程序的运行中，也很好的证明了这一点，在有些地方虽然调用的是TaxiAdvance，但是使用Taxi的类名来调用也没有出现问题。

因此，本程序满足LSP替换原则。