第三次博客作业

规格化设计的发展历史

在20世纪50年代，软件伴随着第一台电子计算机的问世诞生了。以写软件为职业的人也开始出现，他们多是经过训练的数学家和电子工程师。1960年代美国大学里开始出现授予计算机专业的学位，教人们写软件。

在计算机系统发展的初期，由于目的的单一性，软件的通用性是很有限的。大多数软件是由使用该软件的个人或机构研制的，软件往往带有强烈的个人色彩。早期的软件开发也没有什么系统的方法可以遵循，软件设计是在某个人的头脑中完成的一个隐藏的过程。而且，除了源代码往往没有软件说明书等文档。

从60年代中期到70年代中期是计算机系统发展的第二个时期，在这一时期软件开始作为一种产品被广泛使用，出现了“软件作坊”专职应别人的需求写软件。这一软件开发的方法基本上仍然沿用早期的个体化软件开发方式，但软件的数量急剧膨胀，软件需求日趋复杂，维护的难度越来越大，开发成本令人吃惊地高，而失败的软件开发项目却屡见不鲜。“软件危机”就这样开始了！

“软件危机”使得人们开始对软件及其特性进行更深一步的研究，人们改变了早期对软件的不正确看法。早期那些被认为是优秀的程序常常很难被别人看懂，通篇充满了程序技巧。现在人们普遍认为优秀的程序除了功能正确，性能优良之外，还应该容易看懂、容易使用、容易修改和扩充。

1968年北大西洋公约组织的计算机科学家在联邦德国召开的国际学术会议上第一次提出了“软件危机”(software crisis)这个名词。 概括来说，软件危机包含两方面问题：一、如何开发软件，以满足不断增长，日趋复杂的需求；二、如何维护数量不断膨胀的软件产品。

1968年秋季，NATO（北约）的科技委员会召集了近50名一流的编程人员、计算机科学家和工业界巨头，讨论和制定摆脱“软件危机”的对策。在那次会议上第一次提出了软件工程（software engineering）这个概念。

软件工程是一门研究如何用系统化、规范化、数量化等工程原则和方法去进行软件的开发和维护的学科。通过不断努力，人们逐渐解决了软件危机，并认识到规格化设计的重要性，在此期间，一些重要的文档格式的标准被确定下来，包括变量、符号的命名规则以及源代码的规范式。后来随着发展，这些规范逐渐形成了软件开发中的规格化设计，并且由于其高效性与高可靠性，越来越受到软件开发人员的重视。

我认为规格化设计之所以能够受到软件开发人员的重视，主要是因为它能大幅度提高软件工程的质量。例如在大型软件开发中，随着时间的推移可能有越来越多的新功能被要求添加，但是随意修改代码会造成代码的不可靠，只有用抽象与规格的方法设计程序，才能保证代码的高可靠性与易维护性。规格实际上是一种契约化编程手段，将代码的功能进行抽象，使设计人员无需关注代码实现的细节，从而提高设计效率与正确性。

写的好的过程规格与数据规格

过程规格

1.

```
protected void driving(char c,int curState) {
        /**
         * @MODIFIES:
         *      \this.newLocation;
         *      \this.toCusPath;
         *      \this.toDesPath;
         * @EFFECTS:
         *      (c=='U') ==> \this.newLocation == location - SIZE
         *      (c=='D') ==> \this.newLocation == location + SIZE
         *      (c=='L') ==> \this.newLocation == location - 1
         *      (c=='R') ==> \this.newLocation == location + 1
         *      (curState==State.TOCUS) => toCusPath.append(c);
         *      (curState==State.TODES) => toDesPath.append(c);
         */
        if (curState==State.TOCUS)
            toCusPath.append(c);
        else if (curState==State.TODES)
            toDesPath.append(c);
        if (c == 'U') newLocation = location - SIZE;
        else if (c == 'D') newLocation = location + SIZE;
        else if (c == 'L') newLocation = location - 1;
        else if (c == 'R') newLocation = location + 1;
    }
```

2.

```
private boolean checkConnected(int curState) {//检测道路连通性
        /**
         * @MODIFIES:pathIndex;path;
         * @EFFECTS:
         *      roadChanged => (roadChanged==false) &&
         *           (cusState==State.TOCUS ==>
path==map.SPFA(location,customers.peek().getaStart(),taxiEdges)) &&
         *           (curState==State.TODES ==>
path==map.SPFA(location,customers.peek().getEnd(),taxiEdges)) &&
         *           (pathIndex == 0) && (\result==false);
         *      !roadChanged => \result==true;
         */
        if (roadChanged) {
            //路断了，需要重新寻路
            roadChanged = false;
            if (curState==State.TOCUS)
                path = map.SPFA(location, customers.peek().getStart(),taxiEdges);
            else if (curState==State.TODES)
                path = map.SPFA(location, customers.peek().getEnd(),taxiEdges);
            pathIndex = 0;
            return false;
        } else return true;
    }
```

3.

```
synchronized void addTorrent(int start, int end) {
        /**
         * @REQUIRES:
         *      0<=start<=6399;
         *      0<= end <=6399;
         * @MODIFIES:
         *      \this.torrent;
         * @EFFECTS:
         *      isConnected(start,end) ==> (\this.torrent[start][end]+=1)&&(\this.torrent[end]
[start]+=1)
         * @THREAD_REQUIRES:
         *      \locked(\this);
         * @THREAD_EFFECTS:
         *      \locked();
         */
        if (edges[start].getAdjust().stream().anyMatch(o->o==end)) {
            int dx = Math.abs(start / TaxiSys.SIZE - end / TaxiSys.SIZE);
            int dy = Math.abs((start - end) % TaxiSys.SIZE);
            if (dx + dy == 1) {
                torrent[start][end]++;
                torrent[end][start]++;
            }
        }
    }
```

数据规格

1.

```
class Edge {
    /**
     * @Overview:邻接点集类，代表一个点周围的边的集合。拥有开闭路以及获取邻接点的功能。
     * @invariant:(0<=i<6400 for all i belongs to adjoin);
     * @AF(C) = {0<=C.adjoin.get(i)<6400 | 0<=i<c.adjoin.size()}
     */
    private Vector<Integer> adjoin;

    Edge() {
        /**
         * @EFFECTS:
         *      初始化邻接点集使之为空;
         */
    }

    synchronized void openPath(int next) {
        /**
         * @REQUIRES:
         *      0<=next<=6399;
         * @MODIFIES:
         *      \this.adjoin;
         * @EFFECTS:在邻接点集中添加next
         * @THREAD_REQUIRES:
         *      \locked(\this);
         * @THREAD_EFFECTS:
         *      \locked();
         */
    }

    synchronized void closePath(int next) {
        /**
         * @REQUIRES:
         *      0<=next<=6399;
         * @MODIFIES:
         *      \this.adjoin;
         * @EFFECTS:在邻接点集中删除出next;
         * @THREAD_REQUIRES:
         *      \locked(\this);
         * @THREAD_EFFECTS:
         *      \locked();
         */
    }

    synchronized Vector<Integer> getAdjust() {
        /**
         * @EFFECTS:获取邻接点集;
         * @THREAD_REQUIRES:
         *      \locked(\this);
         * @THREAD_EFFECTS:
         *      \locked();
         */
    }
}
```

2.

```
class TrafficLight implements Runnable{
    /**
     * @Overview:TrafficLight表示地图上的交通信号灯，能按照固定周期变化灯光。
     * @invariant:(tfLight[i] == (0 || 1 || -1) for 0<=i<6400) && (50<=period<=100) && (cycle==
(1||-1));
     * @AF(C) = (TrafflicLight,period) where TrafficLight = C.tfLight && period = C.period;
     */
    private int period;//随机产生的变化周期
    private volatile int cycle;//周期性改变
    private TaxiGUI gui;
    private int[] tfLight;
    private ArrayList<Integer> light;
    private Thread t;

    TrafficLight(String filename,TaxiGUI taxiGUI) {
        /**
         * @REQUIRES:
         *      filename!=null && taxiGUI!=null;
         * @MODIFIES:
         *      \this.period;\this.cycle;\this.tfLight;\this.t;
         * @EFFECTS:
         *      初始化红绿灯变化周期以及红绿灯位置.
         */
    }

    int getEastWest(int index) {//如果返回值是1，那么表示为红灯，否则表示为绿灯,0表示没有灯
        /**
         * @REQUIRES:
         *      0<=index<6400;
         * @EFFECTS:
         *      \result == 索引为index的红绿灯在东西方向上的情况;
         */
    }

    int getSouthNorth(int index) {//如果返回值为1，表示为红灯，否则表示为绿灯，0表示没有灯
         /**
         * @REQUIRES:
         *      0<=index<6400;
         * @MODIFIES:None;
         * @EFFECTS:
         *      \result == 索引为index的红绿灯在南北方向上的情况;
         */
    }

    public void run() {
         /**
         * @MODIFIES:
         *      \this.cycle;;
         * @EFFECTS:
         *      \this.cycle == -\this.cycle;
         */
    }

    private boolean init(String filename) {
         /**

         * @REQUIRES:
```

```
            *       filename!=null;
            * @MODIFIES:
            *       \this.tfLight;
            * @EFFECTS:
            *       初始化红绿灯数组
            */
        }

        void start() {
            /**
            * @MODIFIES:
            *       \this.t;
            * @EFFECTS:
            *       启动红绿灯线程;
            */
        }
    }
```

3.

```
class MyPrintWriter {
    /**
     * @Overview:MyPrintWriter将所有的输出统一到一个文件中。
     * @invariant:printWriter!=null;
     * @AF(C) ==> {printWriter} where printWriter=C.printWriter;
     */
    private PrintWriter printWriter;

    MyPrintWriter(String filename) {
        /**
         * @REQUIRES:
         *      filename!=null;
         * @MODIFIES:
         *      \this.printWriter;
         * @EFFECTS:
         *      如果中途不出现异常，则初始化printWriter;
         *      如果中途出现异常，吞掉异常并且System.out.println("ERROR");
         */
    }

    synchronized void print(String s) {
        /**
         * @MODIFIES:
         *      \this.printWriter;
         * @EFFECTS:向printWriter中输出字符串s
         * @THREAD_REQUIRES:
         *      \locked(\this);
         * @THREAD_EFFECTS:
         *      \locked();
         */
    }

    void close() {
        /**
         * @REQUIRES:None;
         * @MODIFIES:
         *      \this.printWriter;
         * @EFFECTS:
         *      \this.printWriter.close();
         */
        printWriter.close();
    }
}
```

写的不好的过程规格与数据规格

写的不好的过程规格

1.

```
//缺乏synchronized的THREAD_EFFECTS语句，改正方式就是在规格中添加相应的THREAD_EFFECTS
private synchronized void listening() {
        /**
         * @MODIFIES:
         *      \this.path;
         *      \this.pathIndex;
         *      \this.state;
         * @EFFECTS:
         *      !\this.customers.isEmpty() ==> \this.path ==
map.SPFA(location,customers.peek.getStart(),taxiEdges) &&
         *                                      \this.pathIndex == 0 &&
         *                                      \this.state == State.TOCUS;
         */
        if (!customers.isEmpty()) {
            path = map.SPFA(location, customers.peek().getStart(),taxiEdges);
            pathIndex = 0;
            state = State.TOCUS;
        }
    }
```

2.

```
//缺少Exception_behavior，解决方法是添加相应的EFFECTS语句
/**
n-1>=0 ==> normal_behavior:\result==records.get(--n)
n-1<0 ==> exception_behavior(NoPreviousException);
*/
public Record previous() throws NoPreviousException{
            /***
             * @MODIFIES:\this.n;
             * @EFFECTS:
             *      \result==records.get(--n);
             */
            if (n-1>=0)
                return records.get(--n);
            else
                throw new NoPreviousException();

        }
```

3.

```
//effects中存在算法描述，修改方式是只写出方法执行之后的效果
/**
((\all i;0<=i<6400) && (\all next;next belongs to edges[i].getAdjust())) ==> torrent[i][next] =
0;
*/
public void run() {
        /**
         * @REQUIRES:None;
         * @MODIFIES:
         *      \this.torrent;
         * @EFFECTS:
         *      首先等候100ms以便与出租车线程错开，然后每隔200ms，地图自动更新
         *      将流量torrent清零.
         * @THREAD_REQUIRES:
         *      \locked(\this);
         * @THREAD_EFFECTS:
         *      \locked(\this);
         */
        try {
            Thread.sleep(100);
        } catch (Exception e) {
        }
        long comp;
        long curTime = gv.getTime();
        while (true) {
            comp = gv.getTime()-curTime;
            if (comp>0 && comp<200) comp = 200-comp;
            else comp = 200;
            try{
                Thread.sleep(comp);
            } catch (Exception e) {
            }
            curTime+=200;
            synchronized (this) {
                for (int i = 0; i < TaxiSys.SIZE2; i++) {
                    for (int next : edges[i].getAdjust()) {
                        torrent[i][next] = 0;
                    }
                }
            }
        }
    }
```

4.

```
//MODIFIES不应该是None,而应该是\this.customers,此外EFFECTS也应该写成
//\result==customers[0] && customers.size = \old(customers).size-1 &&
!customers.contains(\old(customers)[0])
synchronized Customer poll() {
        /**
         * @REQUIRES:None;
         * @MODIFIES:None;
         * @EFFECTS:
         *      \result==customers.poll();
         * @THREAD_REQUIRES:
         *      \locked(\this);
         * @THREAD_EFFECTS:
         *      \locked();
         */
        return customers.poll();
    }
```

5.

```
//依然是存在算法的描述，修改方式是将EFFECTS改成如下所示。
/**
\all customer;customers.contains(customer) && customer.getTime+3000<=t;(c.tryToGetOn() ==>
output(c+" successfully got on a  taxi.")) && (!c.tryToGetOn() ==> c.logFailure()) &&
customers.traverse(taxis);
*/
public void run() {
        /**
         * @REQUIRES:None;
         * @MODIFIES:
         *      \this.customers;
         *      \this.taxis;
         * @EFFECTS:
         *      不断查询当前用户队列，从中取出过期的用户请求并执行它，用户请求可能成功或者失败，
         *      将成功的用户请求分配给出租车，将失败的用户请求进行失败记录。随后遍历用户队列中
         *      剩余的用户请求，继续记录它们周围的出租车状态。
         */
        while (true) {
            long t = gv.getTime();
            while (!customers.isEmpty() && customers.peek().getTime() + 3000 <= t) {
                Customer c = customers.poll();
                if (c.tryToGetOn()) System.out.println(c + " successfully got on a taxi.");
                else {
                    c.logFailure();
                    System.out.println(c + " failed to get on a taxi!");
                }
            }
            //对所有时间未到的用户请求来说，现在还处于窗口期，可以继续加出租车进去
            customers.traverse(taxis);
        }
    }
```

写的不好的数据规格

1.

```
//存在的问题是数据功能太多，SPFA即寻找最短路的算法应该放在Taxi类中，而不是放在Map类中，
//此外Map也不应该放置流量，而应该将流量放在边类中。
class Map implements Runnable{
    /**
     * @Overview:地图是车辆运行环境的表示，包括结点的连通性与边的流量。
     * @invariant: \result==(edges!=null && numMap!=null && torrent!=null &&
                torrent!=null) && (edges[i]!=null for all 0<=i<TaxiSys.SIZE2) &&
                (numMap[i][j]==(0||1||2||3) for all 0<=i<80 && for all 0<=j<80) &&
                (torrent[i][j]>=0 for all 0<=i<6400 && for all 0<=j<6400);
     */
    Edge[] edges = new Edge[TaxiSys.SIZE2];
    Edge[] initEdges = new Edge[TaxiSys.SIZE2];
    int[][] numMap = new int[TaxiSys.SIZE][TaxiSys.SIZE];
    int[][] torrent = new int[TaxiSys.SIZE2][TaxiSys.SIZE2];
    private Thread t;
    private TaxiGUI taxiGUI;

    Map(String mapFile,TaxiGUI gui) {
        /**
         * @EFFECTS:
         *      \all int i;0<=i<Taxisys.SIZE2;edges[i]==new Edge();
         *      \this.numMap will be initialized int method init(filename);
         *      \this.t == new Thread(\this);
         *      \this.taxiGUI == gui;
         *      if initialization failes, this program will output "Wrong input" and quit;
         */
    }

    synchronized void addTorrent(int start, int end) {
        /**
         * @EFFECTS:
         *      如果start与end邻接,则
         *          \this.torrent[start][end]++;
         *          \this.torrent[end][start]++;
         *      否则不对torrent做任何操作
         */
    }

    public void run() {
        /**
         * @EFFECTS:
         *      首先等候100ms以便与出租车线程错开，然后每隔200ms，地图自动更新
         *      将流量torrent清零.
         */
    }

    void start() {
        /**
         * @EFFECTS:
         *      \this.t.start();
         */
    }

    void closePath(int start, int end,Taxi[] taxis) {
        /**

         * @EFFECTS:
```

```
         *      \all int i;i belongs to edges[start].getAdjust && i!=end;System.out.println("The
edge you want to remove doesn't exist at all!");
         *      \exits int i;i belongs to edges[start].getAdjust &&
i==end;edges[start].closePath(end) && edges[end].closePath(start) && return;
         *      \this.torrent[start][end] == \this.torrent[end][start] == 0 && 在gui关闭这条边 &&
         *      (\all int i;0<=i<100 && taxis[i].getState==(State.TODES ||
State.TOCUS);taxis[i].roadChanged == true;
         */
    }

    void openPath(int start, int end,Taxi[] taxis) {
        /**
         * @EFFECTS:
         *      (\all int i;i belongs to edges[start].getAdjust && i!=end; && (\exist int i;
         *      i belongs to initEdges[start].getAdjust && i==end;) =>
edges[start].openPath(end))
         *      && edges[end].openPath(start) && 在gui上连接这条边 &&
         *      (\all int i;0<=i<100 && taxis[i].getState==(State.TODES ||
State.TOCUS);taxis[i].roadChanged == true;
         *      \exits int i;i belongs to edges[start].getAdjust &&
i==end;System.out.println("The edge you want to add exits.") && return;
         */
    }

    private boolean init(String filename) {
        /**
         * @EFFECTS:
         *      \result==true <==> \this.edges and \this.numMap is initialized correctly.
         *      The input file's format isn't correct ==> \result==false;
         *      如果在处理文件过程中出现异常 ==> 吞掉该异常并且\result==false;
         */
    }

    synchronized String SPFA(int _start, int _end,Edge[] searchEdges) {
        /**
         * @EFFECTS:
         *      令字符串path表示SPFA算法找到的从_start到_end的路径最短且流量之和最小的路径.
         *      \result == path;
         */
    }
}
```

2.

```
//问题在于Taxi过于庞杂，从其中的属性数目就可以看出，而且方法数也是所有类中最多的，例如经过反思我认为，
//addCredit方法，listening等方法是多余的，完全可以省略，因此解决办法是细化分工，删除不必要的方法和属性
//，或者整合方法和属性。
class Taxi implements Runnable {
    /**
     * @Overview:出租车类根据分配任务与否选择相应的状态行驶。
     * @invariant:(0<=id<100) && (0<=location<6400) && (0<=lastLocation<6400) &&
(0<=newLocation<6400) &&
     (state == (TOCUS||READY||IDLE||TODES)) && (0<=credit) && (0<=tickTock<=100) &&
     (customers!=null && taxiGUI!=null && map!=null && trafficLight!=null && history!=null) &&
toCusPath!=null &&
     toDesPath!=null;
     */
    protected int id;//出租车编号
    protected int location;//出租车当前位置
    protected int lastLocation;
    protected int newLocation;
    protected int state;//出租车当前状态
    protected int credit = 0;//出租车信用
    protected int tickTock;//出租车计时用变量
    protected String path;//当前路径
    protected String workPath;//将顾客从起点送到终点的路径
    protected int pathIndex;//在当前路径上的位置
    protected BlockingQueue<Customer> customers = new LinkedBlockingQueue<>();//顾客队列
    protected TaxiGUI taxiGUI;
    protected Map map;
    protected Edge[] taxiEdges;
    protected TrafficLight trafficLight;
    protected Vector<Customer> history;//历史订单
    public volatile boolean roadChanged;
    protected Thread t;
    protected StringBuilder toCusPath;
    protected StringBuilder toDesPath;

    Taxi(int id, TaxiGUI taxiGUI, Map map, TrafficLight trafficLight) {
        /**
         * @EFFECTS:
         *      \this.id == id;\this.location == new Random().nextInt(SIZE2);\this.lastLocation
== location;
         *      \this.newLocation == location;\this.state == State.READY;\this.tickTock == 100;
         *      \this.history == new Vector<>();\this.taxiGUI == taxiGUI;\this.map == map;
         *      \this.trafficLight == trafficLight;\this.taxiEdges == map.edges;
         *      \this.toCusPath = new StringBuilder();\this.toDesPath = new StringBuilder();
         */
    }

    public void start() {
        /***
         * @EFFECTS:
         *      \this.t.start();
         */
    }

    synchronized void addCust(Customer c) {//将顾客分配给此出租车
        /**

         * @EFFECTS:
```

```
         *      \this.customers.offer(c);
         *      \this.workPath==map.SPFA(c.getStart(),c.getEnd(),taxiEdges);
         *      \this.state==State.READY ==> \this.path ==
map.SPFA(location,c.getStart(),taxiEdges) &&
         *                                  \this.pathIndex == 0 &&
         *                                  \this.state == STATE.TOCUS;
         *      \result==workPath;
         */
    }

    synchronized void addCredit(Customer customer) {
        /**
         * @EFFECTS:
         *      \exists Customer cust;cust belongs to history && cust.id==customer.id;return;
         *      \all Customer cust;cust belongs to history && cust.id!=customer.id;
         *          history.add(customer) && credit++;
         */
    }

    private synchronized void listening() {
        /**
         * @EFFECTS:
         *      !\this.customers.isEmpty() ==> \this.path ==
map.SPFA(location,customers.peek.getStart(),taxiEdges) &&
         *                                      \this.pathIndex == 0 &&
         *                                      \this.state == State.TOCUS;
         */
    }

    protected void driving(char c,int curState) {//有目的行驶
        /**
         * @EFFECTS:
         *      \this.newLocation == location - SIZE if c=='U':
         *                           location + SIZE if c=='D';
         *                           location - 1 if c=='L';
         *                           location + 1 if c=='R';
         *      (curState==State.TOCUS) => toCusPath.append(c);
         *      (curState==State.TODES) => toDesPath.append(c);
         */
    }

    private void driving() {//无目的行驶
        /**
         * @EFFECTS:
         *      从当前位置选出周围流量最小的边所对应的顶点并更新newLocation，如果有多条流量
         *      相同的边则随机选择一条；如果周围没有任何边那么newLocation保持当前位置location
         *      的值。
         */
    }

    public void run() {
        /**
         * @EFFECTS:
         *      此方法控制单个出租车的运行状态。不断检查state，如果state为等待服务状态，则
         *      将计时器不断减一(减为0便进入停止状态，停止1s后重新恢复计时器并进入等待服务状
         *      态)，如果在中途出现顾客上车，那么出租车状态转变为准备服务状态去接客，接客后睡
         *      1s状态变为服务，到达服务目的地后睡1s状态继续变为等待服务。在每运行一条边之后
```

```
         *       出租车均会修改边的流量信息。在每服务完一位乘客之后出租车的信用均+3。
         */
    }

    protected void finishedTask(Customer c) {
        /**
         * @EFFECTS:
         *       让c输出它这趟行程经过的所有点到文件中
         */
    }

    private void working(int curState) {
        /***
         * @EFFECTS:
         *       检测前方道路是否联通，联通则继续等红绿灯然后行驶，再更新流量以及设置GUI，否则重新计算道路并运行
         */
    }

    private boolean checkConnected(int curState) {//检测道路连通性
        /**
         * @EFFECTS:
         *       roadChanged => (roadChanged==false) &&
         *            (path==map.SPFA(location,customers.peek().getEnd(),taxiEdges) &&
         *            (pathIndex == 0) && (\result==false);
         *       !roadChanged => \result==true;
         */
    }

    private void checkTrafficLight(int prev,int cur,int next) {
        /**
         * @EFFECTS:
         *       如果前方是红灯且出租车将要直行，或者前方是绿灯且出租车将要左转，则等红绿灯直到红绿灯变色。
         */
    }

    synchronized Object[] getSnapShot(int custPos, boolean necessary) {//为了避免脏读写,同时返回5个信息
        /**
         * @EFFECTS:
         *       返回一个包含了当前出租车状态等信息的Ojbect数组，为了避免无谓的计算设置标志位necessary，
         *       仅当necessary被设置时才进行SPFA()计算，否则不进行计算。
         */
    }
}
```

3.

```
//通过检查类的数据抽象规格，发现addTaxi方法没有提出对重复出租车的判断，因此可能存在3s
//的时间窗口中同一辆出租车被加入乘客类的taxis中，解决办法是加上这个判断
class Customer {
    /**
     * @Overview:乘客类代表一个乘客请求，拥有上车以及记录的功能。
     * @invariant:(0<=start<6400) && (0<=end<6400) && (time>=0) && (out!=null) && (buffer!=null)
&& (taxis!=null);
     */
    private int start;
    private int end;
    private long time;
    private int id;
    private boolean logged;//控制首次记录
    private MyPrintWriter out;
    private StringBuilder buffer;
    private Set<Taxi> taxis = new LinkedHashSet<>();

    Customer(int start, int end, long time, MyPrintWriter out) {
        /**
         * @EFFECTS:
         *      \this.start==start;
         *      \this.end==end;
         *      \this.time==time-time%100;
         *      \this.logged==false;
         *      \this.out==out;
         *      \this.buffer==new StringBuilder();
         */
    }

    boolean equals(Customer c) {
        /**
         * @EFFECTS:
         *      \result == (c!=null && \this.time == c.time &&
         *          \this.start == c.start && \this.end == c.end);
         */
    }

    synchronized void addTaxi(Taxi taxi) {
        /**
         * @EFFECTS:
         *      \this.taxis.add(taxi);
         */
    }

    boolean tryToGetOn() {//是否能上车
        /**
         * @EFFECTS:
         *      \this.taxis.length==0 ==> \result=false;
         *      \exists Taxi t;(t in taxis)&&(t.state == READY);\result==true;
         *      \all Taxi t;(t in taxis)&&(t.state != READY);\result==false;
         */
    }

    void arrivedDes(String path) {
        /**

         * @EFFECTS:
```

```
            *       根据传入的路径将这趟行程所经过的点全部打印出文件中
            */
    }

    private void printPath(int x, int y) {
        /**
         * @EFFECTS:
         *      \this.buffer.append(">>>>("+x+","+y+")\t");
         */
    }

    void logFailure() {
        /**
         * @EFFECTS:
         *      MyPrintWriter prints the content in the buffer and flushes;
         */
    }

    void log(Object[] snapshot) {
        /**
         * @EFFECTS:
         *      \this.buffer.append("Taxi#" + id + "\tState : " + state +
         *      "\tLocation : (" + (loc / TaxiSys.SIZE + 1)
         *      + "," + (loc % TaxiSys.SIZE + 1) + ")"
         *      + "\tCredit : " + credit + "\n");
         */
    }
}
```

4.

```
//我感觉写的不好的地方是在请求队列中存在了traverse方法，如果从抽象的角度理解，请求队列只是一个存储请求
//的地方，而遍历它应该交给外部类通过迭代器或者其他方法来实现，因此这个方法有分工不明之嫌，应该放置在调
//用者Scheduler中。
class RequestQueue {
    /***
     * @Overview:请求队列中存放用户请求，并不断清理过期用户请求。
     * @invariant:
     */
    private static int cnt;
    private BlockingQueue<Customer> customers;

    RequestQueue() {
        /***
         * @EFFECTS:
         *      \this.customers = new LinkedBlockingQueue<>(300);
         */
    }

    synchronized Customer poll() {
        /**
         * @EFFECTS:
         *      \result==customers.poll();
         */
    }

    synchronized Customer peek() {
        /**
         * @EFFECTS:
         *      \result==customers.peek();
         */
    }

    synchronized boolean isEmpty() {
        /**
         * @EFFECTS:
         *      \result==customers.isEmpty();
         */
    }

    synchronized void offer(Customer customer) {
        /**
         * @EFFECTS:
         *      \all Customer c;c belongs to customers &&
!c.equals(customer);customers.offer(customer) && customer.setId(cnt++);
         *      \exists Customer c;c belongs to customers && c.equals(customer);return;
         */
    }

    void traverse(Taxi[] taxis) {
        /**
         * @EFFECTS:
         *      遍历taxis中的出租车获取它们的当前快照，再遍历customers将距离合适的出租车
         *      添加到用户的考虑队列中，再将该用户添加到该出租车的用户队列中，如果是该用户
         *      第一次记录信息则记录该用户周围出租车状态。
         */

    }
```

```
}
```

5.

```
//我感觉写得不好的地方在于此类作为一个出租车管理类只提供了几个查询所有出租车的函数，缺少
//对指定编号出租车坐标、信用值等查询的函数，因此从一个数据抽象上来说这个类型是不完备的。
class TaxiManager {
    /**
     * @Overview:出租车管理器提供查询任意出租车的能力
     * @invariant:(taxis!=null)&&(taxis[i]!=null for all 0<=i<taxis.length)
     */
    private Taxi[] taxis;

    TaxiManager(Taxi[] taxis) {
        /*
         * @EFFECTS:
         *      \this.taxis == taxis;
         */
    }

    ArrayList<Taxi> queryTaxi(int _state) {
        /*
         * @EFFECTS:
         *      ArrayList<Taxi> temp = new ArrayList<>();
         *      \all int i;0<=i<TaxiSys.TAXINUM &&
taxis[i].getState()==_state;temp.add(taxis[i]);
         *      \result==temp;
         */
    }

    //获取编号为i的出租车的状态
    int getState(int i) {
        /*
         * @EFFECTS:
         *      \result==taxis[i].getState();
         */
    }
}
```

BUG与规格的关系

第10次作业扣分主要来自于JSF，有多个方法在类中使用了synchronized(this)但是却未在JSF中表示出来。此外，由于在设计规格的时候过于复杂，导致一个类或者一个方法往往要完成多个任务，这样反过来让JSF非常难写，又增加了程序的耦合性，导致新增功能较为麻烦。

现在反思整个出租车调度系统程序的编写， 感到存在例如代码冗余、效率不高等问题。虽然随着时间的推移，大家逐渐不再执着于扣他人在程序功能上的BUG而是扣他人JSF上的BUG，所以这几次没有遇到功能性BUG的扣分，但是我还是在写第11次作业的时候发现了自己代码中存在的历史遗留问题。例如在出租车完成任务的时候，我只考虑到在将乘客送到目的地过程中道路状况的改变，没有考虑到出租车去乘客起点的过程中道路状况的改变，因此如果是在这个过程中道路被关闭了，出租车就可能出现“飞车”的情况。这也从另一个角度说明，在开始写代码之前明确方法的规格是很有必要的。