设计文档

# 需求分析图

## 

## 需求分析说明

1.乘客交互的数据特征：

乘客所在位置、乘客的目的地

2.乘客交互的时间特征：

开始发送订单的时间点、发送订单的时长

3.出租车交互的数据特征：

出租车所在位置、出租车所处状态、出租车信用

4.出租车交互的时间特征：

出租车休息的频率、出租车移动速度、出租车休息时长、上下客时长

5.具体交互关系：

乘客 -> 服务器端：告知服务器何时开始叫车、自己的位置在哪儿、目的地在哪儿。

服务器端 -> 乘客：反馈给乘客是否能有车来接他。

出租车 -> 服务器端：告知服务器端自己的位置、信用如何、现在是否能接客。

服务器端 -> 出租车：指挥出租车去接客并将乘客送到目的地、根据出租车行为增加出租车信用。

## LSP原则验证

新出租车类TraceableTaxi继承自Taxi，

TraceableTaxi中所有重写的方法的前置条件与后置条件都能保持父类的方法规格仍然成立。

举例：（achieveOrder方法在父类与子类实现的对比）

父类中的achieveOrder:

public synchronized boolean achieveOrder(Request request){  
 /\* @ REQUIRES: request != null;  
 @ MODIFIES: this;  
 @ EFFECTS: 存储请求所在位置与目的地到this.clientLocation和this.destination;  
 \*/  
  
 if (this.state == 0) {  
 this.state = 2;  
 this.clientLocation = request.getLocation();  
 this.destination = request.getDestination();  
 return true;  
 }  
 return false;  
}

子类中的achieveOrder:

public synchronized boolean achieveOrder(Request request){  
 /\* @ REQUIRES: request != null;  
 @ MODIFIES: this;  
 @ EFFECTS: 存储请求所在位置与目的地到this.clientLocation和 this.destination，并在this.serviceRecords中添加新记录;  
 \*/  
  
 if (this.state == 0) {  
 this.state = 2;  
 this.clientLocation = request.getLocation();  
 this.destination = request.getDestination();  
  
 //将新请求记录下来  
 this.recordIndex++;  
 int locationX = this.location[0];  
 int locationY = this.location[1];  
 ServiceRecord record = new ServiceRecord(request, locationX, locationY);  
 this.serviceRecords.add(record);  
  
 return true;  
 }  
 return false;  
}

可以看到，子类中的achieveOrder方法仅是在父类的achieveOrder方法中增加了服务记录的操作，其他操作继承自父类，并未违背父类的方法规格。

再来看这一段在扫描器线程（实现对可接单出租车的扫描）中的代码：

Taxi priorTaxi = null;  
  
for (int i = 0; i < probableTaxis.size(); i++) {  
 if (i == 0) {  
 priorTaxi = probableTaxis.get(i);  
 } else {  
 priorTaxi = checkPriority(priorTaxi, probableTaxis.get(i));  
 }  
}  
  
if (!priorTaxi.achieveOrder(request)) {  
 continue; //该出租车在寻找最优者时更改了状态  
}

这段代码首先申明了Taxi类的priorTaxi，之后又让priorTaxi执行了achieveOrder方法。

这里的priorTaxi可以是新出租车，也可以是普通出租车，而先前我也提到achieveOrder被重写过，而此段代码对于新旧出租车都能正常运行。

综上所述，新增的TraceableTaxi类满足LSP原则——任何父类出现的地方都可以使用子类来代替，并不会导致使用相应类的程序出现错误。