面向对象课程第三次总结分析

tags：面向对象课程文档 Java

规格化设计

- 规格化设计是从人们的实践中逐渐发展起来的，从简单到复杂，从单一到复杂多样，从思想到实践。如今的设计语言都会对程序设计有一定的规格化要求，在世界范围内也有相应的标准化，例如：POSIX。
- 规格提供了抽象的描述；编写规格使定义的抽象清除显示出来，使注意力迅速集中在不一致不完全和不明确的问题上；规格也定义了用户和实现者之间的契约；规格可以适用于许多不同的目的，并可以成为一个有用的维护工具；因此规格化设计能获得人们的重视。

好的规格

过程规格

1. repOK 方法的规格：这类型的规格 EFFECTS 部分只需要根据不变式对各种情况进行列举，逻辑清晰

```
    @MODIFIES:None;
    @EFFECTS:(cityMap==null||taxis==null||!((Object)startTime instanceof Long)||
(RouteGuider==null)||req==null) ==> \result == false;
              (cityMap.length!=80||taxis.length!=100) ==> \result == false;
               (\exist int i;0<=i<80;cityMap[i].length!=80) ==> \result == false;
               (\exist int i,j;0<=i<80,0<=j<80,;cityMap[i][j]==null) ==> \result == false;
               (\exist int i;0<=i<100;taxis[i]==null) ==> \result == false;
               (Otherwise) ==> \result == true;
```

2. void addReq(request req)：虽然返回值是 void，但是 EFFECTS 部分需要指出 MODIFITES 部分的影响

```
    @REQUIRES: req != null;
    @MODIFIES: this;
    @EFFECTS: (!(req.start_point.equals(req.end_point)) && !
(\old(this.reqList).contains(req))) ==> (this.reqList.contains(req));
    @THREAD_EFFECTS: \locked();
```

3. Object previous()：迭代器的方法

```
    @MODIFIES: iterator
    @EFFECTS: normal behavior:
            (this.hasPrevious()) ==> \result == (Object)reqList.get(iterator-1) &&
iterator==\old(iterator)-1;
            (!hasPrevious()) ==> \result == null;
    @THREAD_EFFECTS: \locked()java
```

数据规格

1. 对类作用的概括

```
/* Overview: 这个类能够刷新地图上边的流量 提供测试者打开和关闭地图上边的方法
 * Objects: cityMap, taxis, startTime, gui;
 * cityMap类的setOpen和setClose方法能够打开和关闭地图上最初存在的道路
 * cityMap类依靠updateflow方法在run方法中周期性刷新流量
 * Abstract Function: AF(c) = (cityMap, taxis, startTime, gui), where
 *      cityMap = c.cityMap, taxis = c.taxis, startTime = c.startTime, gui = c.gui
 * Invariant: cityMap!=null && taxis!=null && gui!=null && cityMap.length==80 &&
taxis.length==100 &&
 *              (\all int i;0<=i<80;cityMap[i].length==80)&&(\all int
i,j;0<=i<80,0<=j<80,;cityMap[i][j]!=null)&&
 *              (\all int i;0<=i<100;taxis[i]!=null)
 */
```

2. 对类管理数据的概括

```
/* Overview: 这个类保存了请求的发出地与目的地以及请求发出时间等信息 提供了判断请求是否相同的方法
 * Objects: req_str, start_point, end_point, req_t;
 * request类提供了equals方法比较两个请求是否相等
 * request类提供了reqInfor方法得到需要的请求格式字符串
 * Abstract Function: AF(c) = (req_str, start_point, end_point, req_t), where
 *      req_str = c.req_str, start_point = c.start_point, end_point = c.end_point, req_t =
c.req_t
 * Invariant: !((req_str==null||start_point==null||end_point==null||!((Object)req_t
instanceof Double)))
 */
```

3. e

```
/* Overview: 这个类开启一个3s的抢单窗口 在请求发出位置周围4×4区域中记录抢单的出租车信息 并在抢单窗口结束后选取符合要求的出租车响应请求
 * Objects: cityMap, taxis, startTime, RouteGuider, req;
 * grabWindow通过getRange方法得到请求发出位置周围4×4区域
 * grabWidow的grab方法能够记录请求发出3s内的出租车信息 并在3s结束后选择符合要求的出租车
 * grabWindow在run方法中调用grab并分配出租车
 * Abstract Function: AF(c) = (cityMap, taxis, startTime, RouteGuider, req), where
 *      cityMap = c.cityMap, taxis = c.taxis, startTime = c.startTime, RouteGuider = c.RouteGuider, req = c.req
 * invariant: !((cityMap==null||taxis==null||!((Object)startTime instanceof Long)||(RouteGuider==null)||req==null)||
 *               (cityMap.length!=80||taxis.length!=100)||(\exist int i;0<=i<80;cityMap[i].length!=80)||
 *               (\exist int i;0<=i<100;taxis[i]==null))
 */
```

不好的规格

过程规格

1. void carStop() 第九次作业

- MODIFIES 部分修改的变量是用户不可见的，不需要写出具体变量，应该以 this 代替

```
    @MODIFIES: gui, status, waitTime;java
```

- 改进：

```
    @MODIFIES: this;
```

2. boolean equals(request r) 第九次作业

- 虽然限定了类型，但是 REQUIRES 没有对 null 进行限制

```
      @REQUIRES: None;
      @MODIFIES: None;
      @EFFECTS:
(start_point.equals(r.start_point)&&end_point.equals(r.end_point)&&req_t==r.req_t) ==>
\result == true;
                (!
(start_point.equals(r.start_point)&&end_point.equals(r.end_point)&&req_t==r.req_t)) ==>
\result == false;
```

- 改进：

```
      @REQUIRES: r!=null;
      @MODIFIES: None;
       @EFFECTS:
(start_point.equals(r.start_point)&&end_point.equals(r.end_point)&&req_t==r.req_t) ==>
\result == true;
                (!
(start_point.equals(r.start_point)&&end_point.equals(r.end_point)&&req_t==r.req_t)) ==>
\result == false;
```

3. void randomMove() 第十次作业

- 出租车选择流量最小的路径行走，在 EFFECTS 中可以对修改说明更加明确

```
    @MODIFIES: this;
    @EFFECTS: update taxi's location randomly;
```

- 改进：

```
    @MODIFIES: this;
    @EFFECTS: (this.nextPoint == point(which has the least flow)) && (update this.gui) &&
(update this.position);
```

4. void addReq(request req) 第九次作业

- 自然语言的描述可以替换成更规范的 JSF 形式

```
    @MODIFIES: reqList;
    @EFFECTS: (req出发地与目的地不同 && req不是相同请求) ==> reqList.add(req);
    @THREAD_EFFECTS: locked();
```

- 改进：

```
    @REQUIRES: req != null;
    @MODIFIES: this;
    @EFFECTS: (!(req.start_point.equals(req.end_point)) && !
(\old(this.reqList).contains(req))) ==> (this.reqList.contains(req));
    @THREAD_EFFECTS: \locked();
```

5. point getMinFlow(int i,int f) 第十次作业

- 先写完代码再补的 JSF，EFFECTS 部分代码痕迹很严重

```
    @MODIFIES: None;
    @EFFECTS:   (i==0) ==> \result == f==1?upPoint:origin_upPoint;
                (i==1) ==> \result == f==1?downPoint:origin_downPoint;
                (i==2) ==> \result == f==1?leftPoint:origin_leftPoint;
                (i==3) ==> \result == f==1?rightPoint:origin_rightPoint;
                (Otherwise) ==> \result == null;
    @THREAD_EFFECTS: \locked();
```

- 改进：

```
    @MODIFIES:None;
    @EFFECTS: (i==0&&f==1)==>\result==upPoint;   (i==0&&f==0)==>\result==origin_upPoint;
              (i==1&&f==1)==>\result==downPoint; (i==1&&f==0)==>\result==origin_downPoint;
              (i==2&&f==1)==>\result==leftPoint; (i==2&&f==0)==>\result==origin_leftPoint;
              (i==3&&f==1)==>\result==rightPoint;(i==3&&f==0)==>\result==origin_rightPoint;
              (Otherwise) ==> \result == null;
    @THREAD_EFFECTS: \locked();
```

数据规格

1. 类的功能概括

- 不应该逐一列举类的方法并解释，应该从整体上概述类的功能

```
/* Overview: 这个类提供使使用者能够通过控制台输入出租车请求并将请求提交到请求队列的方法
 * Objects: startTime, running, reqList, s_in, reqCheck;
 * input_handle通过input方法获取输入并生成合适的请求
 * input_handle通过在run方法中调用input获取乘客请求
 * Abstract Function: AF(c) = (startTime, running, reqList, s_in, reqCheck), where
 *      startTime = c.startTime, running = c.running, reqList = c.reqList, s_in = c.s_in,
reqCheck = c.reqCheck
 * Invariant: !((reqList==null)||reqCheck==null||s_in==null)
 */
```

- 改进：

```
/* Overview: 这个类使使用者能够通过控制台输入出租车请求并将请求提交到请求队列
 * Objects: startTime, running, reqList, s_in, reqCheck;
 * Abstract Function: AF(c) = (startTime, running, reqList, s_in, reqCheck), where
 *      startTime = c.startTime, running = c.running, reqList = c.reqList, s_in = c.s_in,
reqCheck = c.reqCheck
 * Invariant: !((reqList==null)||reqCheck==null||s_in==null)
 */
```

2. 类的数据管理

- 只是列举了类的属性，没有解释类的属性但是也没有告知用户管理的数据

```
/* Overview: 这个类提供使使用者能够通过控制台输入出租车请求并将请求提交到请求队列的方法
 * Objects: startTime, running, reqList, s_in, reqCheck;
 * input_handle通过input方法获取输入并生成合适的请求
 * input_handle通过在run方法中调用input获取乘客请求
 * Abstract Function: AF(c) = (startTime, running, reqList, s_in, reqCheck), where
 *      startTime = c.startTime, running = c.running, reqList = c.reqList, s_in = c.s_in,
reqCheck = c.reqCheck
 * Invariant: !((reqList==null)||reqCheck==null||s_in==null)
 */
```

- 改进：

```
/* Overview: 这个类提供使使用者能够通过控制台输入出租车请求并将请求提交到请求队列的方法，类调整程序中的请求队列，并管理与构造请求和检测请求相关的对象
 * Objects: startTime, running, reqList, s_in, reqCheck;
 * Abstract Function: AF(c) = (startTime, running, reqList, s_in, reqCheck), where
 *      startTime = c.startTime, running = c.running, reqList = c.reqList, s_in = c.s_in, reqCheck = c.reqCheck
 * Invariant: !((reqList==null)||reqCheck==null||s_in==null)
 */
```

3. 类的不变式

- 不变式直接来自 reqOK，应该使用或与式更好

```
/* Overview: 这个类提供使使用者能够通过控制台输入出租车请求并将请求提交到请求队列的方法
 * Objects: startTime, running, reqList, s_in, reqCheck;
 * input_handle通过input方法获取输入并生成合适的请求
 * input_handle通过在run方法中调用input获取乘客请求
 * Abstract Function: AF(c) = (startTime, running, reqList, s_in, reqCheck), where
 *      startTime = c.startTime, running = c.running, reqList = c.reqList, s_in = c.s_in, reqCheck = c.reqCheck
 * Invariant: !((reqList==null)||reqCheck==null||s_in==null)
 */
```

- 改进：

```
/* Overview: 这个类提供使使用者能够通过控制台输入出租车请求并将请求提交到请求队列的方法
 * Objects: startTime, running, reqList, s_in, reqCheck;
 * input_handle通过input方法获取输入并生成合适的请求
 * input_handle通过在run方法中调用input获取乘客请求
 * Abstract Function: AF(c) = (startTime, running, reqList, s_in, reqCheck), where
 *      startTime = c.startTime, running = c.running, reqList = c.reqList, s_in = c.s_in, reqCheck = c.reqCheck
 * Invariant: (reqList!=null)&&(reqCheck!=null)&&(s_in!=null)
 */
```

4 5

程序的bug与过程规格质量的关系

第九次作业

- 公测和互测没有bug

第十次作业

1. 存在一个险些导致无效的 bug：在等待红绿灯的 waitLight 方法中，出租车只会识别有红绿灯的路口的红绿灯变化来调整状态，对于没有红绿灯的路口则永远感知不到变化而一直等待，由于自己测试时使用的是全红绿灯文件并没有发现此 bug，导致作业最开始被无效，之后申诉成功
2. 与过程规格质量的分析：

- 在 waitLight 方法规格的 EFFECTS部分没有考虑到不存在红绿灯时需要返回的情况

第十一次作业

- 公测和互测没有bug