

# 契约式软件设计与测试的闯关式案例

《面向对象设计与构造》课程

北京航空航天大学计算机学院

孙青 吴际

# 提纲

- 案例背景
- 教学目标
- 教学要点
- 教学组织方式
- 案例实施成效
- 问题探讨

# 案例背景

- 契约式设计基于断言来规约类的行为，是高质量软件开发的一种重要方法，由Bertrand Meyer最早提出（Eiffel语言）
  - ◆ MIT等知名高校纷纷将契约式设计思想融入到相关的课程教学活动中
    - 北航的《面向对象设计与构造》课程从2013年开始讲授和训练
  - ◆ 在工业界实践中，契约表示与高级编程语言相结合
    - 在IDE中表达契约设计，便于追踪契约与代码之间的关系
    - 帮助开发人员在开发的过程中及时找到设计错误
    - 促进沟通交流与理解

# 案例背景

- 规模适中的软件案例是成功开展案例教学的关键
  - ◆ 能够展现契约设计的特点和益处
  - ◆ 能够展现不同同学的设计思考差异
- 案例选自于北航面向对象课程的作业训练
  - ◆ 第3周的训练项目
  - ◆ 融合了第9和第10周的知识点训练要求（契约式设计）
  - ◆ 融合了第12周的单元测试知识点及训练要求（面向覆盖率目标的测试设计与优化）
- **单电梯调度系统**
  - ◆ 案例规模：4~6个类
  - ◆ 真实世界可感知，易于理解
  - ◆ 有足够的算法深度来发挥
- 先序知识点要求
  - ◆ 对象与对象化编程的基本知识：继承、多态、抽象.....
  - ◆ 已经会编写300行左右的面向对象程序
  - ◆ 掌握了方法规格和类规格的基本概念
  - ◆ 提前预习如何利用Junit开展单元测试

# 提纲

- 案例背景
- 教学目标
- 教学要点
- 教学组织方式
- 案例实施成效
- 问题探讨

# 教学目标

## □ 训练目标

- ◆ 契约与代码的**双向关联**，打通设计与实现的壁垒
  - 数据抽象
    - 基于需求，以类为单位规约数据必须满足的约束，并实现代码(validation code)
  - 方法抽象
    - 基于需求和类设计，设计方法规格并实现
- ◆ 契约、代码和测试的**三层次逻辑整合**
  - 以规格为依据，设计和实现测试用例
  - 基于规格对所实现代码进行自动化测试

## □ 学时有限，如何落地？

- ◆ 引入**JSF**，通过一定规范性的注释来描述数据抽象和方法抽象
- ◆ 借助**Junit**，规范和简化进行单元测试所要做的工作

# 提纲

- 案例背景
- 教学目标
- 教学要点
- 教学组织方式
- 案例实施成效
- 问题探讨

# 教学要点

## □ 类的数据抽象

### ◆ 不变式 (Invariant)

- 定义对象有效的判定条件，即类成员变量需要满足的约束
- 类的任何操作都不能导致不变式为假，否则就破坏了对象的有效性

### ◆ 类的不变式如何描述？

- 采用JSF语法
- 例如

– 对于请求队列QueryList中的所有请求，在任何时刻都满足按时间非降序排列

$\backslash \text{all int } i, j; 0 \leq i < j < \text{this.size};, \text{this}[i].\text{queryTime} \leq \text{this}[j].\text{queryTime}$



# 教学要点

## □ 类的数据抽象

### ◆ 不变式 (Invariant)

- 类的任何操作都不能导致不变式为假，否则就破坏了对对象的有效性

### ◆ 不变式如何实现？

- 为了在程序运行时针对对象有效性进行检查，把不变式中的约束条件实现为 `repOk` 的方法，该方法不带参数，返回布尔值
- 测试时该类的每一个方法都应该使用它来确认对象的有效性

```
public boolean repOk(){  
    if(!(targetFloor<=10&&targetFloor>=1)) return false;  
    if(!(queryTime>= 0 && queryTime < inftyTime)) return false;  
    if(queryDirection!=Direction.NONE || queryDirection!=Direction.UP || queryDirection!=Direction.DOWN)  
        return false;  
    return true;  
}
```

# 教学要点

## □ 方法抽象

- ◆ 调用者和被调用者之间的沟通机制
  - 前置条件（Pre-condition）
    - 规定方法能够正确处理的输入范围
    - 常见的前置条件设置
      - » 限制输入参数的合理范围
      - » 限制方法所属对象的状态
  - 后置条件（Post-condition）
    - 描述方法执行之后必须达到的效果
    - 常见的后置条件
      - » 方法执行显式输出内容（返回值）要满足的条件
      - » 方法执行隐式输出内容（写入文件等）要满足的条件
- ◆ 满足前置条件，但是方法执行后不满足后置条件→callee有bug
- ◆ 不满足前置条件→caller有bug

# 教学要点

## □ 方法抽象

- 使用JSF对方法抽象进行描述：@REQUIRES、@MODIFIES、@EFFECTS
- 例如，以下方法为请求队列QueueList中的一个方法，完成从队列中删除指定位置的请求，根据需求，该方法的后置条件为：从空的请求队列中删除请求，必须抛出空队列异常；从请求队列中删除非法位置（小于0或大于队列当前元素个数）的请求，必须抛出非法请求的异常；只有当队列中删除元素成功时返回布尔值真值。

```
public boolean remove(int index) throws EmptyQueueException, InvalidIndexException{
    /*@MODIFIES:this
    @EFFECTS:
        normal_behavior
        (\old(this).get(index) !=null) ==> (this.size == \old(this).size-1) && (this.contains(\old(this).get(index))==false) &&
        (\result==true);
        (\old(this).size ==0)==>exceptional_behavior(EmptyQueueException)
        (index >=\old(this).size) ==> exceptional_behavior (InvalidIndexException);
        (index < 0) ==> exceptional_behavior (InvalidIndexException);
    */
```

# 教学要点

## □ 基于Junit的测试

- ◆ 使用Junit框架来设计和编写可执行的单元测试用例
- ◆ 测试用例依据规格进行
- ◆ 例子：
  - 对电梯类Elevator的moveUP()方法进行单元测试

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;

public class TestQueryList {

    Elevator ele=new Elevator(10,1);

    @Test
    public void TestMoveup(){

        assertEquals(ele.moveUP(),true);
    }

}
```

# 提纲

- 案例背景
- 教学目标
- 教学要点
- 教学组织方式
- 案例实施成效
- 问题探讨

# 教学组织方式

- ❑ 契约式设计比较抽象，写文档的方式难以保证投入度和效果
- ❑ 基于平台开展在线实验
  - ◆ 给定时间内完成任务
  - ◆ 任务提交可自动在线检查
  - ◆ 本案例已全部在Trustie(educoder)上上线，并以开放尝鲜
- ❑ 实验组织
  - ◆ 完整的软件需求说明+不完整的实现代码
  - ◆ 闯关式设计



# 教学组织方式

## □ 训练内容设计

### ◆ 数据抽象的训练

- 给出参考程序中的部分类代码，要求给出不变式并将缺少的repOk方法补充完整
- 给出参考程序中类的不变式和不完整的代码，要求据此将代码补充完整，使得可以满足不变式要求

### ◆ 方法抽象的训练

- 给定类的部分方法代码，要求结合需求理解代码，并给出方法规格
- 给定方法规格说明，要求根据规格实现代码
- 给定方法规格说明和有缺陷的代码实现，要求理解规格和代码，然后找到并修改与规格说明不符的代码

### ◆ 测试用例的设计训练

- 根据给定的方法规格来设计和编写针对单个方法的测试用例
- 根据类的不变式，围绕对象状态设计综合性的测试用例

# 教学组织方式

## □ 示例1：给出类的不变式并实现约束条件代码

- ◆ 依据对需求的理解，对Elevator类的不变性的具体要求写出不变式

```
代码文件    已保存
30
31  /**
32   * 类的约束条件检查
33   * @return
34   *     true:类成员变量满足约束条件
35   *     false: 类成员变量不满足约束条件
36   *
37   * @return
38   */
39  /*****不变式开始*****/
40  AF(c)={Elevator, curStatus queryTime, queryDirection}
41  Where lowLevel=<curStatus.targetFloor<=highLevel , 0=<queryTime, curStatus.queryDirection={UP, DOWN, NONE}
42
43  ****不变式结束*****/
44  public boolean repOk(){
45      /*****Begin*****/
46
47      if(getCurTime()<0){
48          return false;
49      }
50      if ((curStatus.getDirection()!=Query.Direction.NONE) || (curStatus.getDirection()!=Query.Direction.UP) || (curStatus
51      {return false;}
52      if (curStatus.getTarget()>highLevel || curStatus.getTarget()<lowLevel)
53      {return false;}
54      return true;
55
56      /*****End*****/
57  }
```



# 教学组织方式

## □ 示例2: 基于类的不变式补全代码

```
58      /**
59       * 类的约束条件检查
60       * @return
61       *      true:类成员变量满足约束条件
62       *      false: 类成员变量不满足约束条件
63       */
64      public boolean repOk(){
65          /*****Begin*****/
66          if(!(targetFloor<=10&&targetFloor>=1)) return false;
67          if(!(queryTime>= 0 && queryTime < inftyTime)) return false;
68          if(queryDirection!=Direction.NONE||queryDirection!=Direction.UP||query
69          return true;
70      }
71
72      /****End*****/
73
74
75
76
77
78
79
80
81
82
83
84
85
```

### 测试结果

✓ 4/4 全部通过

▶ 测试集 1



▶ 测试集 2



▶ 测试集 3



▶ 测试集 4



# 教学组织方式

## □ 示例3：依据方法规格描述补全方法代码

```
37
38 public boolean append(Query req) {
39     /**@ REQUIRES: req != null ;
40         @ MODIFIES: this;
41         @ EFFECTS:
42         (this.lastTime>req.queryTime)==>\result=false;
43         (req.targetFloor=low && req.queryDirection==Direction.DOWN)==>\result=false;
44         (req.targetFloor=high && req.queryDirection==Direction.UP)==>\result=false;
45         (this.queue.size == \old(this.queue).size+1) && (this.queue.contains(req)==true)&&(this.queue.lastTime==req.queryTime)
46     */
47     /*****Begin*****/
48     if(req==null) return false;
49     if(lastTime > req.getTime()) return false;
50     if(req.getTarget()==lowLevel && req.getDirection()==Query.Direction.DOWN) return false;
51     if(req.getTarget()==highLevel && req.getDirection()==Query.Direction.UP) return false;
52     queue.add(req);
53     this.lastTime=req.getTime();
54     return true;
55     /*****End*****/
56 }
```

### 测试结果

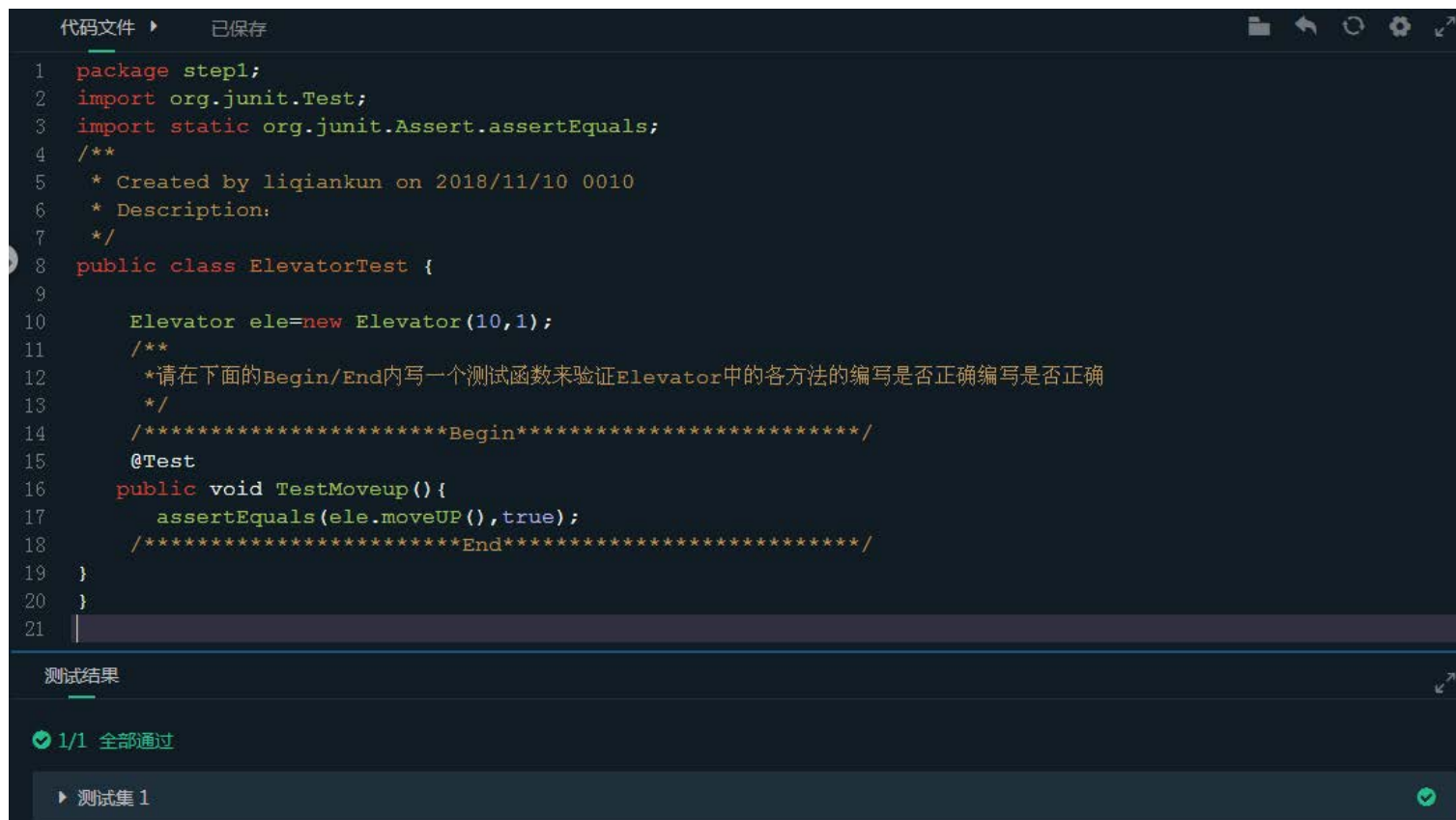
❗ 4/5 共有5组测试集，其中有1组测试结果不匹配。详情如下：

▶ 测试集 1



# 教学组织方式

## □ 示例4：基于方法规格设计编写单元测试用例



```
代码文件 ▶ 已保存
1 package step1;
2 import org.junit.Test;
3 import static org.junit.Assert.assertEquals;
4 /**
5  * Created by liqiankun on 2018/11/10 0010
6  * Description:
7  */
8 public class ElevatorTest {
9
10     Elevator ele=new Elevator(10,1);
11     /**
12      *请在下面的Begin/End内写一个测试函数来验证Elevator中的各方法的编写是否正确编写是否正确
13      */
14     /*****Begin*****/
15     @Test
16     public void TestMoveup(){
17         assertEquals(ele.moveUP(),true);
18     }
19     /*****End*****/
20 }
21
```

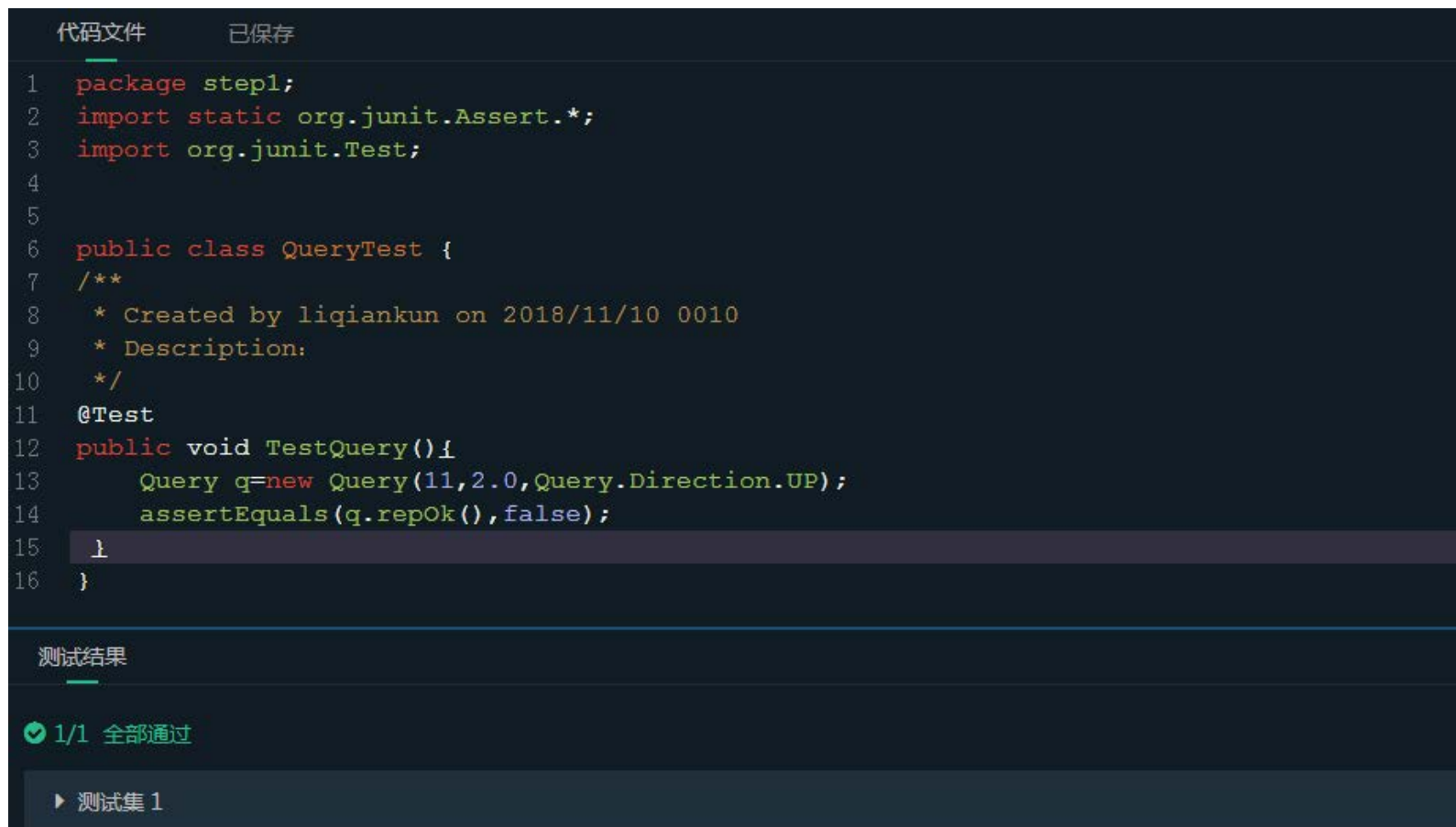
测试结果

✓ 1/1 全部通过

▶ 测试集 1 ✓

# 教学组织方式

## □ 基于类规格设计编写类的单元测试



```
代码文件    已保存
1  package step1;
2  import static org.junit.Assert.*;
3  import org.junit.Test;
4
5
6  public class QueryTest {
7      /**
8       * Created by liqiankun on 2018/11/10 0010
9       * Description:
10      */
11      @Test
12      public void TestQuery(){
13          Query q=new Query(11,2.0,Query.Direction.UP);
14          assertEquals(q.repOk(),false);
15      }
16  }
```

测试结果

✓ 1/1 全部通过

▶ 测试集 1

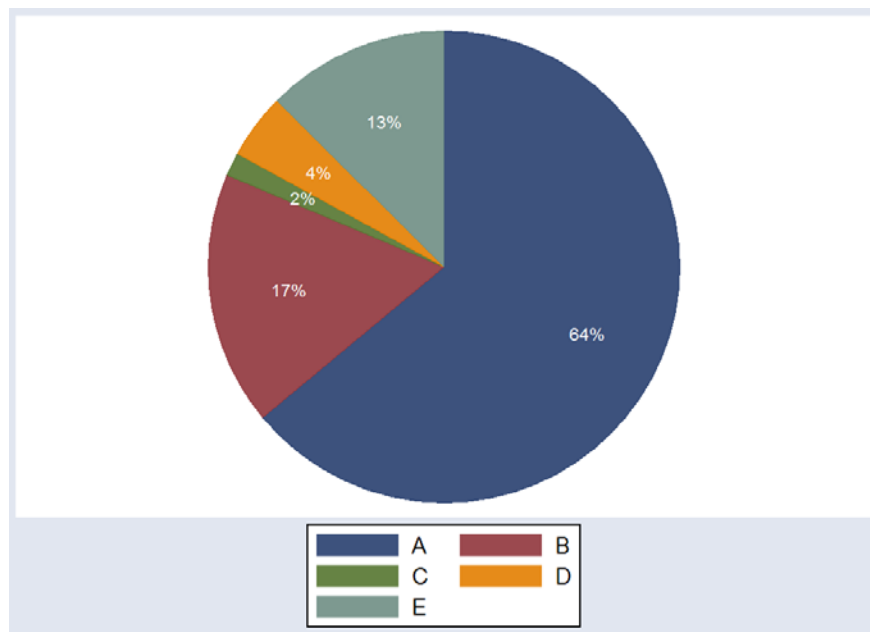
# 提纲

- 案例背景
- 教学目标
- 教学要点
- 教学组织方式
- 案例实施成效
- 问题探讨

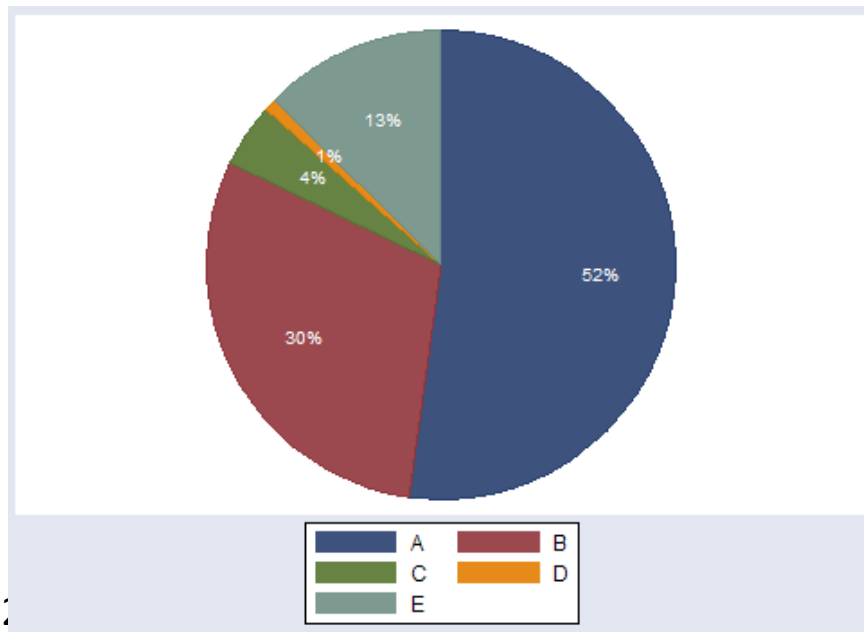
# 案例实施成效

- 案例分两次实施，2学时/次
- 以统一的标准对学生实验成果进行评判，分为A~E五档
  - ◆ 未提交或无效提交的比例均为13%，与理论课作业无效提交的学生名单基本吻合
  - ◆ 多数学生能迅速理解训练要点并按时完成
  - ◆ 半数以上的学生能在规定的时间较好地完成训练内容

方法抽象及测试用例设计训练



数据抽象及测试用例设计训练



# 提纲

- 案例背景
- 教学目标
- 教学要点
- 教学组织方式
- 案例实施成效
- 问题探讨


# 问题探讨

- 案例教学受欢迎，但借鉴和重用难度大
  - ◆ 教学目标
  - ◆ 学生群体
  - ◆ 落地实施平台
- 规模/难度适当、提供足够“扑腾”空间的教学案例是抓手
- 案例的剪裁和使用不可回避
  - ◆ 训练任务细粒度设计
  - ◆ 训练任务可组合
  - ◆ 平台支持



# 问题探讨

- 后置条件是归纳方法语义的重要手段，需要什么方法来规范化描述方法的后置条件？
  - ◆ JSF
  - ◆ OCL
  - ◆ JML
- 如何补充训练帮助学生发现方法规格和不变式的逻辑冲突？
  - ◆ 每个测试开始之前调用repOK，结束时调用repOK，确保对象的有效性
  - ◆ 逻辑推理
- 如何基于规格来设计得到高覆盖率效果的测试用例集？
  - ◆ 逻辑划分与组合



感谢各位专家！

敬请批评指正！

