**《软件测试》**

**实验报告三 ——决策表分析方法**

**姓 名： 汪成肸 学 号： 2020105002**   **院 系： 计算机与信息学院 专 业：计算机科学与技术**

**实 验 室： 实验日期： 2023/04/10**

**总评成绩： 审阅教师：**

### 一、实验目的：

1. 练习基本的编程能力；
2. 巩固 git 协作式管理工具的使用；
3. 巩固 gradle 项目构建 Java 应用的基本操作
4. 掌握决策表分析方法设计测试用例的方法

### 二、实验环境：

IntelliJ IDEA 2022.2.2

### 三、实验要求：

1. 实现电话账单收费功能
2. 采用决策表分析方法设计测试用例，并采用Junit5 编写电话账单收费程序测试用例，并提交到代码仓库
3. 使用checkstyle生成代码规范检查报告
4. 使用jacoco生成测试覆盖率报告
5. 使用spotbugs插件检查代码，生成代码安全检查报告，并修复检查出来的bug

四、实验步骤与内容

**1.需求：电话账单计费**

对于电话账单来说，当春季和秋季标准时间与夏令时时间进行转换时会产生一个很有意思的问题: 春季，这种转换发生在(3月某个)星期日凌晨2:00点，这时要将时钟设置为凌晨3:00点；秋季， 转换通常在11月的第一个星期日，时钟要从2:59:59调回2:00:00。

请为长途电话服务函数开发计费类，使用等价类分析方法构建测试用例并使用Junit5测试。

**采用如下计费规则计算通话费：**

通话时间小于等于20分钟时，每分钟收费0.05美元，通话时间不足1分钟按1分钟计算。

通话时间大于20分钟时，收费1.00美元，外加超过20分钟的部分每分钟0.10美元；

不到1分钟按1分钟计算

**假设：**

通话计费时间从被叫方应答开始计算，到呼叫方挂机时结束；

通话时间的秒数向上进位到分钟；

没有超过30个小时的通话。

**2.电话计费类TelephoneBilling类实现代码**

1. **import** java.time.LocalDateTime;
2. **import** java.time.Month;
3. **import** java.time.temporal.ChronoUnit;
4. **import** java.time.temporal.TemporalAdjusters;
6. **public** **class** TelephoneBilling {
8. **public** **static** **double** calculateCallCost(LocalDateTime startDateTime, LocalDateTime endDateTime) {
9. **double** callCost = 0.0;
11. // 计算通话时长（分钟），向上进位到最近的一分钟
12. **long** durationInMinutes = ChronoUnit.MINUTES.between(startDateTime, endDateTime) + 1;
14. // 计算起始日期和结束日期之间的小时数
15. **long** hoursBetween = ChronoUnit.HOURS.between(startDateTime, endDateTime);
17. // 检查通话是否跨越了春季或秋季的夏令时变更
18. **if** (isDuringDaylightSavingTimeChange(startDateTime) || isDuringDaylightSavingTimeChange(endDateTime)) {
19. // 对于春季变更，将通话时长减少1小时；对于秋季变更，将通话时长增加1小时
20. durationInMinutes += (isDuringDaylightSavingTimeChange(startDateTime) ? -60 : 60);
21. }
23. // 根据通话时长计算通话费用
24. **if** (durationInMinutes <= 20) {
25. callCost = durationInMinutes \* 0.05;
26. } **else** {
27. callCost = 1.0 + (durationInMinutes - 20) \* 0.1;
28. }
30. // 如果通话时长不足1分钟，按照1分钟计算
31. **if** (durationInMinutes < 1) {
32. callCost = 0.05;
33. }
35. **return** callCost;
36. }
38. // 辅助方法，用于检查给定的日期和时间是否处于夏令时变更期间
39. **private** **static** **boolean** isDuringDaylightSavingTimeChange(LocalDateTime dateTime) {
40. Month month = dateTime.getMonth();
41. **int** dayOfMonth = dateTime.getDayOfMonth();
42. **int** year = dateTime.getYear();
44. **if** ((month == Month.MARCH && dayOfMonth > 7) || (month == Month.NOVEMBER && dayOfMonth < 7)) {
45. // 超过3月的第二个星期日或者11月的第一个星期日之后
46. **return** **false**;
47. }
49. LocalDateTime nextSunday = LocalDateTime.of(year, month, 1, 2, 0).with(TemporalAdjusters.next(DayOfWeek.SUNDAY));
50. **if** ((month == Month.MARCH && dayOfMonth >= nextSunday.getDayOfMonth())
51. || (month == Month.NOVEMBER && dayOfMonth < nextSunday.getDayOfMonth())) {
52. // 在或之后3月的下一个星期日或者在11月的下一个星期日之前
53. **return** **true**;
54. }
56. **return** **false**;
57. }
58. }

**3.决策表分析方法：**

决策表分析方法是一种用于描述和分析复杂决策规则的技术。它通过将决策规则表示为决策表的形式，从而使得决策规则更加直观和易于理解。决策表通常由条件（条件部分）和动作（动作部分）组成，用于描述在不同条件下需要执行的不同动作。

决策表分析方法主要包括以下步骤：

1. 收集和整理决策规则：将需要分析的决策规则收集并整理成决策表的形式。决策表通常使用二维表格的形式，其中条件作为表格的列，动作作为表格的行。
2. 确定条件和动作：在决策表中标识出条件和动作。条件是触发决策规则执行的前提条件，动作是在满足条件时需要执行的操作。
3. 分析决策规则：通过对决策表的分析，可以识别出规则之间的重复、冲突、遗漏等情况。可以使用各种技术和工具，如逻辑分析、决策表验证等来进行分析。
4. 优化决策规则：根据分析结果，可以对决策规则进行优化，消除重复、解决冲突、填补遗漏等问题，从而提高决策表的效率和可靠性。
5. 实施决策规则：将经过分析和优化的决策规则应用到实际决策过程中，根据条件的满足情况执行相应的动作。

**4.决策表分析方法设计测试用例：**

**startDateTime endDateTime expectedCallCost**

**2023/1/1 10:00 2023/1/1 11:00 5.1**

**2023/3/10 8:30 2023/3/10 9:30 5.1**

**2023/11/5 1:30 2023/11/5 2:30 5.1**

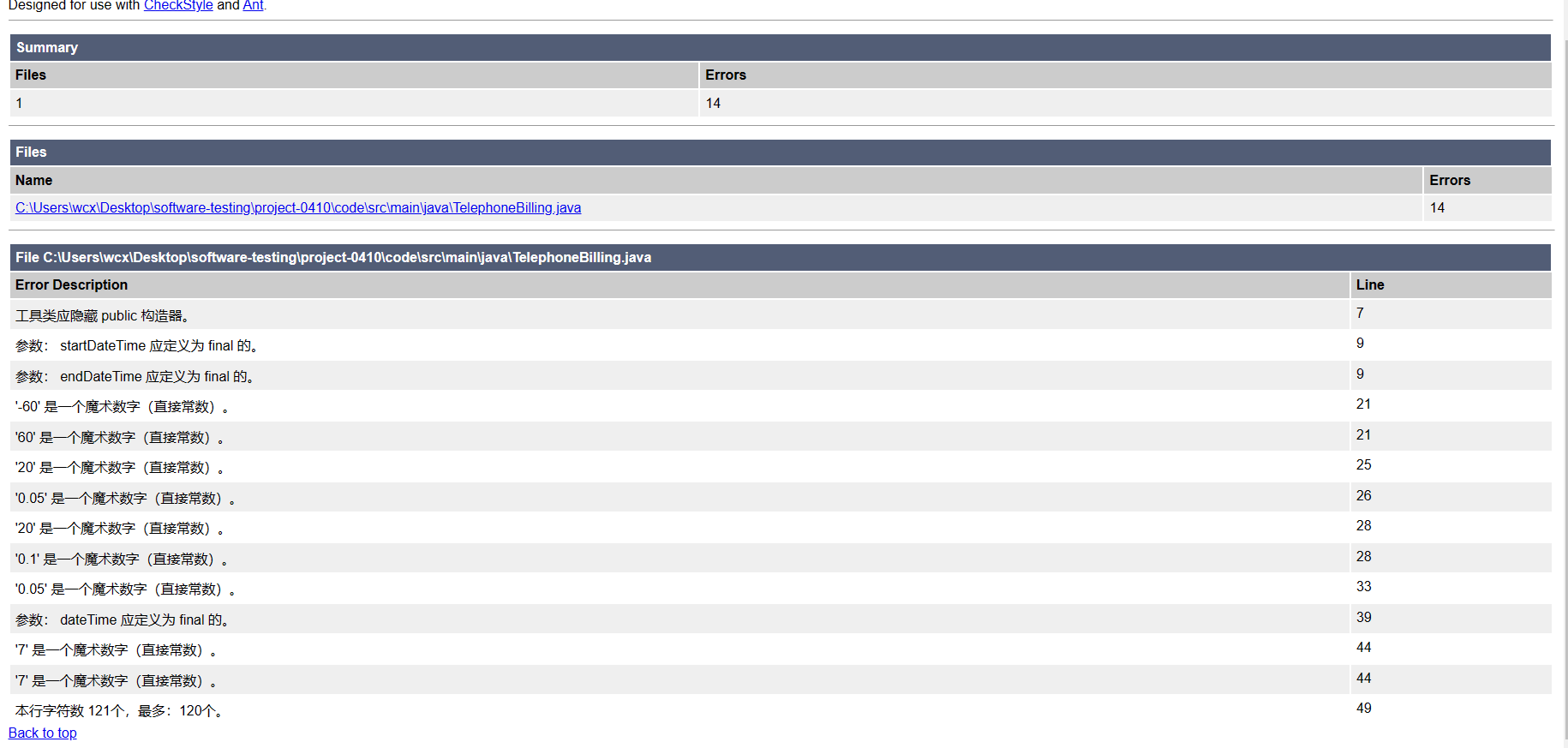
**2023/7/1 15:30 2023/7/1 17:00 8.1**

**2023/12/31 23:45 2024/1/1 0:15 2.1**

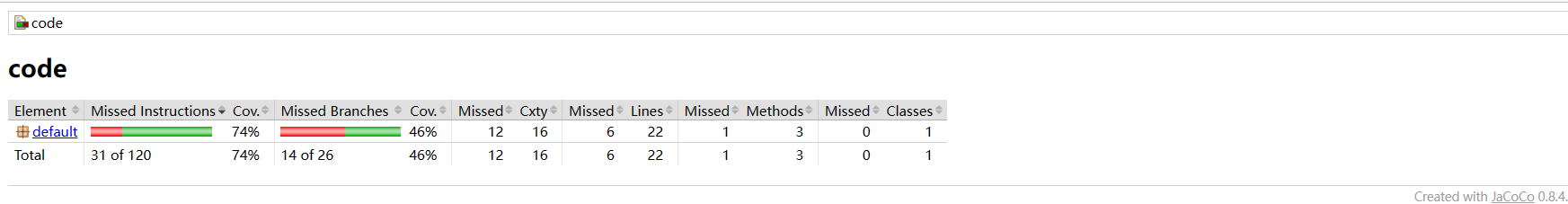
**5.Junit5编写测试代码：**

1. **import** org.junit.jupiter.params.ParameterizedTest;
2. **import** org.junit.jupiter.params.provider.CsvFileSource;
4. **import** java.time.LocalDateTime;
5. **import** java.time.format.DateTimeFormatter;
7. **import** **static** org.junit.jupiter.api.Assertions.assertEquals;
9. **public** **class** TelephoneBillingTest {
11. @ParameterizedTest
12. @CsvFileSource(resources = "/test\_data.csv", numLinesToSkip = 1)
13. **public** **void** testCallCost(String startDateTime, String endDateTime, **double** expectedCallCost) {
14. // 将字符串类型的日期时间转换为实际的日期时间对象，根据具体的实现方式进行转换
15. // 示例中使用了 LocalDateTime 类，可以根据实际情况选择合适的日期时间类
16. LocalDateTime start = LocalDateTime.parse(startDateTime, DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
17. LocalDateTime end = LocalDateTime.parse(endDateTime, DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
19. // 调用计算通话费用的方法，计算实际的通话费用
20. **double** actualCallCost = TelephoneBilling.calculateCallCost(start, end);
22. // 断言实际的通话费用与预期的通话费用是否相等
23. assertEquals(expectedCallCost, actualCallCost);
24. }
25. }

**6.checkstyle生成代码规范检查报告**

****

**7.jacoco生成测试覆盖率报告**

****

### 五、结论分析与体会

通过这次实验，通过学习决策表分析方法，我深刻体会到这一方法在软件测试和规则管理方面的重要性。决策表分析方法以结构化的方式进行规则描述，并以可视化的形式呈现，使得规则更加清晰和易于理解。通过深入挖掘潜在规则，决策表分析方法可以帮助发现隐藏的规则关系，从而更好地测试和验证系统的行为。此外，决策表分析方法还可以汇聚多方利益，将业务规则、系统需求、用户期望等不同方面的要求整合到一个决策表中，从而提高规则管理的效率和一致性。在实际应用中，决策表分析方法可以有效地减少测试用例数量，提高测试覆盖率，从而降低测试成本，提高软件质量。总的来说，决策表分析方法是一种强大的工具，可以帮助测试人员和规则管理人员更好地理解、验证和管理复杂的业务规则系统。

### 六、仓库地址

https://github.com/shuaishuaixi03/software-testing/tree/main/project-0410/code