Project2 白盒测试报告

1 前言

测试的模块是 MyCalendar，类名是 MyCalendar，总共 51 行代码，算空白行。

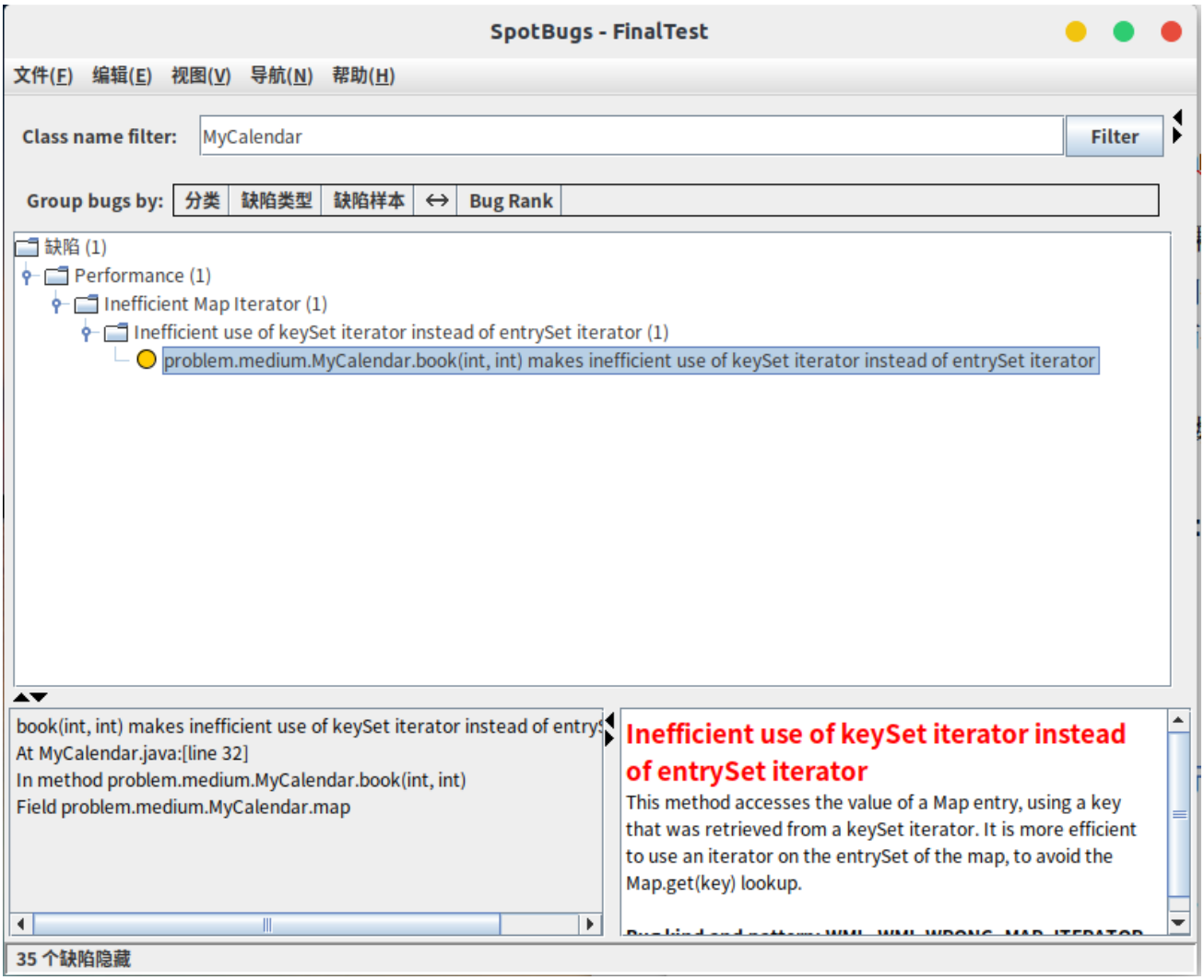
MyCalendar 的功能是以课程开始和结束的时间来添加课程。其主要保证的是添加的课程的开始和结束时间不能够重叠，如果重叠则会添加失败，返回 false，反之，添加成功，返回 true。其类里面有且只有两个函数，现在介绍一下这个两个函数的作用：

+MyCalendar()：构造函数，new 一个对象自动调用，其方法体里面初始了 map 变量。

+book(int start, int end): boolean：接受课程的开始和结束时间，往 map 里面添加课程信息，成功返回 true，失败返回 false。

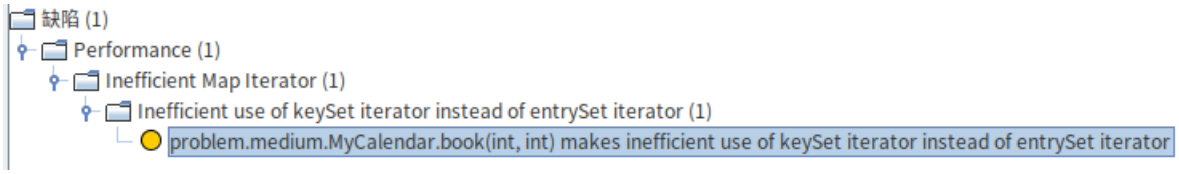
2 静态白盒测试

2.1 测试结果



2.2 缺陷分析

1) Bug Explorer 窗口



2) Bug Info 窗口

Inefficient use of keySet iterator instead of entrySet iterator

This method accesses the value of a Map entry, using a key that was retrieved from a keySet iterator. It is more efficient to use an iterator on the entrySet of the map, to avoid the Map.get(key) lookup.

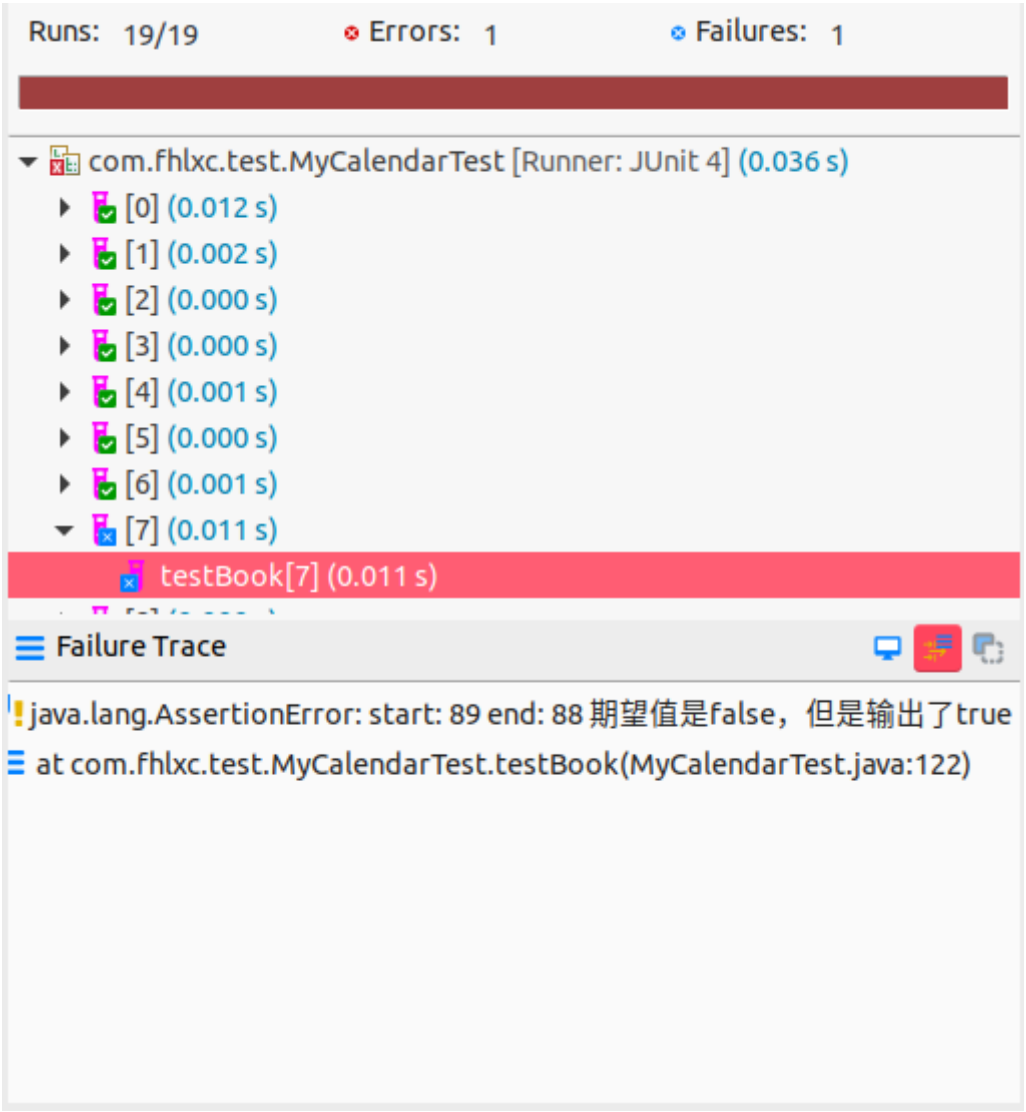
Bug kind and pattern: WMI - WMI_WRONG_MAP_ITERATOR

3) 缺陷分析

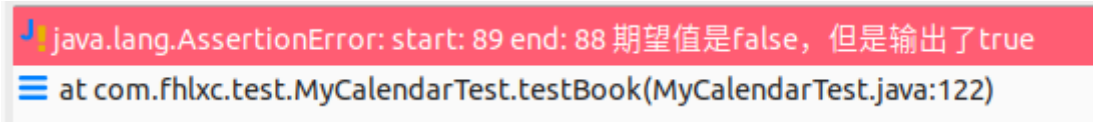
book 函数中是通过先获取 map 的 key，调用 Map.get(key)来获取值，这样的话是多此一举，直接使用 Map.EntrySet()，可一次性将 key 和 value 全部取出来的方式进行遍历。有人做过测试，这两种方式遍历 map 的效率大概是 1.5：1。

3 动态白盒测试

3.1 测试结果



3.2 缺陷分析



分析：start 与 end 之间应该满足的条件是 start>=end，如果 end 比 start 小，应该返回的是 false，但是程序运行结果是 true，进一步分析代码，发现其未对 start 和 end 之间的关系进行判断，导致了输出错误的结果。

分析：为了使得自己所写的测试代码的覆盖率尽可能的百分百，增加了一组测试用例，抛出不存在类型的异常。程序的整体设计思路是，利用 static 代码块，对 List 进行初始化，这个 List 存的是 map 的初始值，总共设有四种类型：Empty（初始 map 为空）、OneSize（初始 map 有一个元素）、NonEmpty（初始 map 非空且大小大于 1）、Error（为了覆盖 100%，补的类型），0、1、2、3 与上述四个类型依次对应，之后，调用 initMap 函数在测试前，对 map 初始，其中为了覆盖 switch 的 default 分支，而引入了 Error 类型，使得覆盖率百分百，default 分支抛出违法参数的异常。

3.3 测试覆盖率

分支覆盖：

Element	Coverage	Covered Branch	Missed Branche
ZigZagOrderLevelTraversalBST	0.0 %	0	22
BeautifulArrangement.java	0.0 %	0	24
CombinationSum3.java	0.0 %	0	24
MyCalendar.java	95.8 %	23	1
MyCalendar	95.8 %	23	1
MyCalendar()		0	0
book(int, int)	95.8 %	23	1
RemoveComments.java	0.0 %	0	24

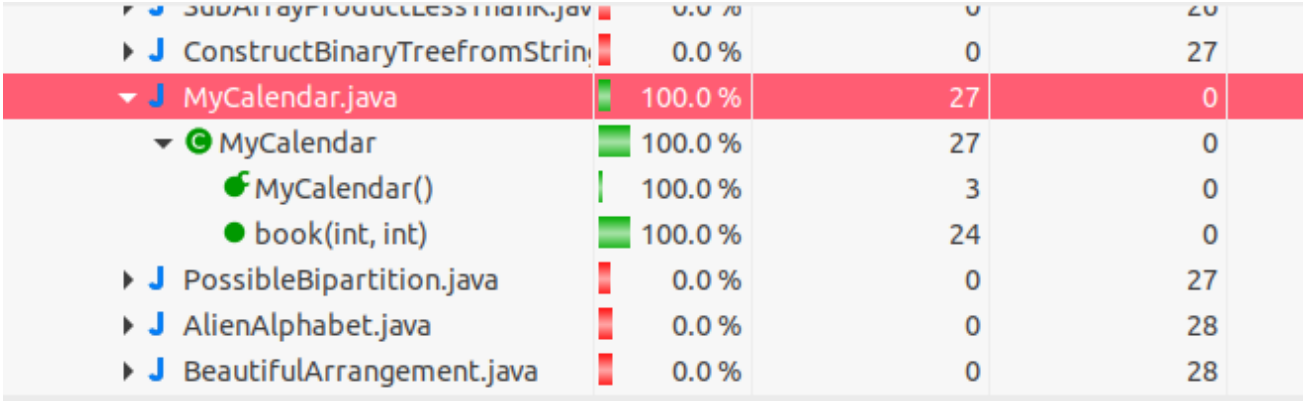
未达到百分百的原因如下：

首先看未达到百分百的代码：

```
31              if (prev == null) {
32                  prev = map.get(key);
33                  if (end <= key) {
34                      break;
35                  }
36              } else {
37                  int[] current = map.get(key);
38                  if (start >= current[1])
39                      continue;
40                  if (end < prev[1] || end > current[0])
41                      return false;
42                  if (start >= current[1] || start < prev[1])
43                      return false;
44                  break;
```

首先介绍一下 TreeMap 这个数据结构，它会将添加到里面的元素按 key 的值排序，默认是从小到大的顺序，再看一下代码，prev 始终是 map 的第一个元素，也就是最小的那个片段，也就是说 current[1]无论如何都是大于等于 prev[1]的，那么 start 大于等于了 current[1]，就不可能在比 prev[1]更小，所以这条分支是不可能被覆盖的。

行覆盖：


ConstructBinaryTreefromStrin 0.0 % 0 27
MyCalendar.java 100.0 % 27 0
MyCalendar 100.0 % 27 0
MyCalendar() 100.0 % 3 0
book(int, int) 100.0 % 24 0
PossibleBipartition.java 0.0 % 0 27
AlienAlphabet.java 0.0 % 0 28
BeautifulArrangement.java 0.0 % 0 28