

SHANGHAI JIAO TONG UNIVERSITY



课程名称： 高级软件测试

学生姓名/学号: 唐天成515030910287

学生姓名/学号: 李林生515030910272

学生姓名/学号:

学生姓名/学号:

专 业: 软件工程

指导教师: 姚建国

学院(系): 电子信息与电气工程学院

目录

[一、 简介 3](#_Toc534578292)

[1.1、 编写目的 3](#_Toc534578293)

[1.2、测试范围 3](#_Toc534578294)

[二、 测试资源 3](#_Toc534578295)

[2.1、人力资源 3](#_Toc534578296)

[2.2、测试对象 3](#_Toc534578297)

[2.3、测试环境 3](#_Toc534578298)

[2.4、测试工具 4](#_Toc534578299)

[三、 进行回归测试 4](#_Toc534578300)

[3.1、修改部分测试用例 4](#_Toc534578301)

[3.2、再测试全部用例 6](#_Toc534578302)

[3.2.1、选择的策略介绍 6](#_Toc534578303)

[3.2.2、测试结果 6](#_Toc534578304)

[3.3、随机测试部分用例 6](#_Toc534578305)

[3.3.1、选择的策略介绍 6](#_Toc534578306)

[3.3.2、测试结果 7](#_Toc534578307)

[3.4、测试风险程度高的用例 7](#_Toc534578308)

[3.4.1、选择的策略介绍 7](#_Toc534578309)

[3.4.2、测试结果 8](#_Toc534578310)

[3.5、再测试修改的部分 8](#_Toc534578311)

[3.5.1、选择的策略介绍 8](#_Toc534578312)

[3.5.2、测试结果 9](#_Toc534578313)

[四、 对回归测试的分析 9](#_Toc534578314)

[4.1、分析回归测试在软件产品迭代更新中的优点 9](#_Toc534578315)

[4.2、分析所选回归测试方法对测试效率和安全性的影响 9](#_Toc534578316)

[4.2.1、再测试全部用例 10](#_Toc534578317)

[4.2.2、随机生成用例 10](#_Toc534578318)

[4.2.3、基于风险选择测试用例 10](#_Toc534578319)

[4.2.4、再测试修改的部分 10](#_Toc534578320)

# 简介

## 编写目的

本软件测试报告面向高级软件测试的课程作业，本次测试做的是回归测试。我们选取的程序是作业一和作业二都采用的单位转换程序，该程序分为温度和压强的单位转化，该程序一共有三个模块，分别为ConverterService , ConvertPressure, ConcertTemperature。我们在原来代码的基础上进行了修改和新功能的增加。我们增加了一个压强的单位，同时也修改了温度转换的逻辑实现。

之后，我们对该项目进行回归测试。由于代码的逻辑有了部分的修改，所以有的测试用例已经是不正确的了，因而我们首先在原来的代码里找出了那些已经没有用的测试用例，把它们删除掉了。之后我们采用不同的选择方法去筛选回归测试的测试用例集。

在回归测试结束之后我们编写测试报告，从而发现被测试的软件原有的功能是否还是符合设计的要求，发现软件开发中存在的各种问题，便于改正。

## 1.2、测试范围

本文档适用于“单位转化”程序的回归测试工作。

# 测试资源

## 2.1、人力资源

小组成员：

唐天成：讨论采用的回归测试策略，以及编写文档。

李林生：讨论采用的回归测试策略，修改代码和执行回归测试的不同策略。

## 2.2、测试对象

单位转化的小程序，分为温度和压强的单位转化，一共有三个模块，分别为ConverterService , ConvertPressure, ConcertTemperature，还有一个记录了常量的类Constants。

## 2.3、测试环境

系统：Windows10；java版本：1.8.0\_121

## 2.4、测试工具

测试工具：JUnit4；

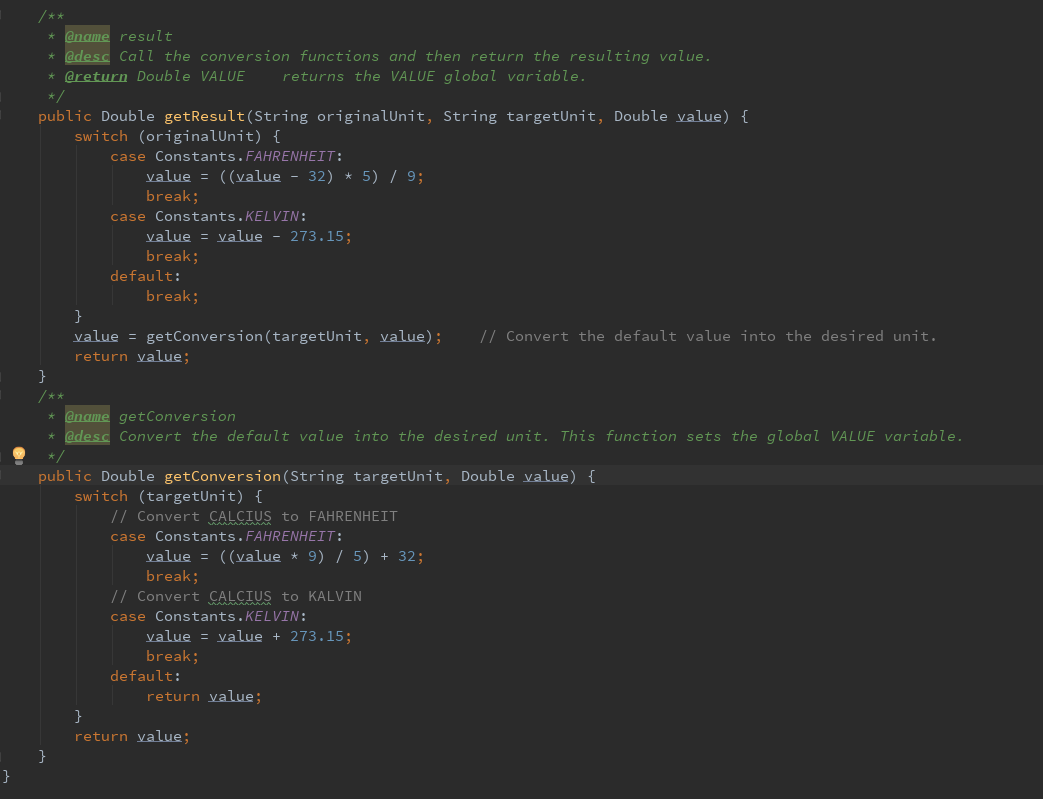
# 进行回归测试

## 3.1、修改部分测试用例

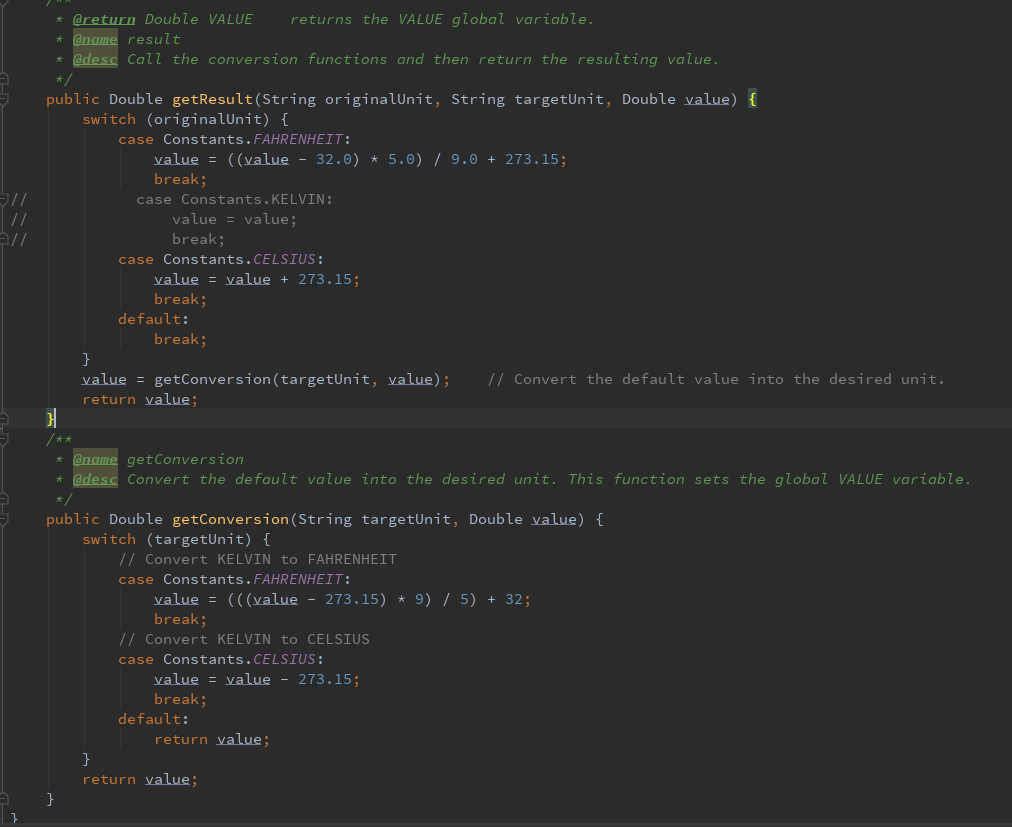
由于代码的逻辑发生了变化，我们首先找到那些原有的测试用例中不满足现在的要求的用例，去删除或者修改它们。

对于ConcertTemperature类，我们修改了温度转化的实现方式，但该类实现的功能并没有改变。

原始实现：

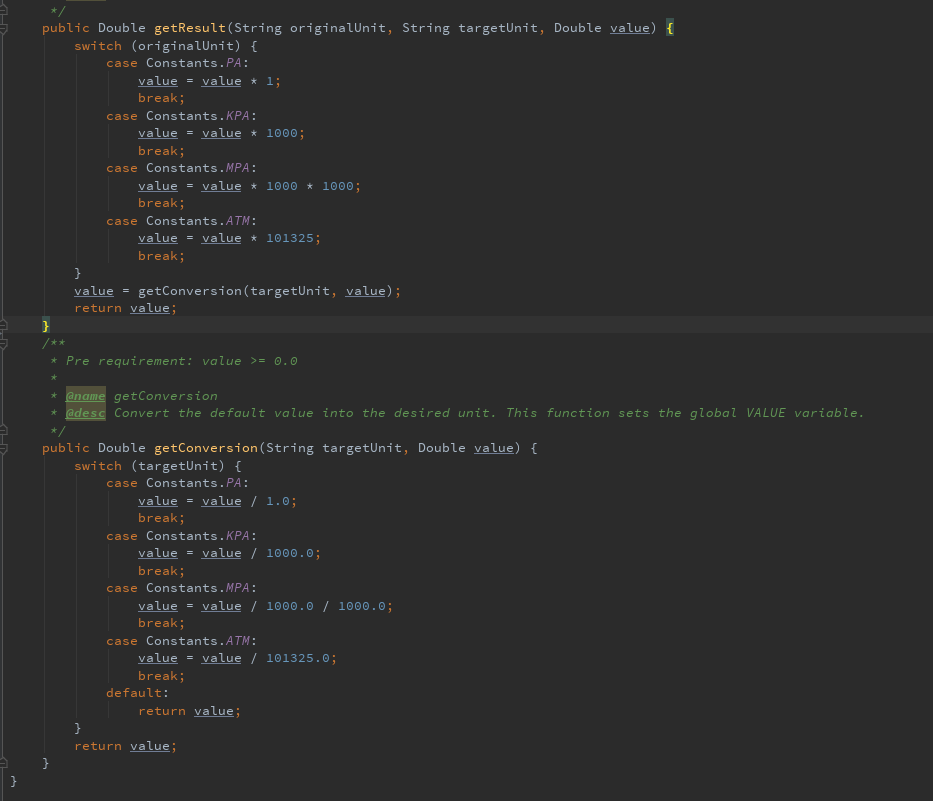


新的实现：



对于ConcertPressure类，我们增加了新的压强单位MPa。

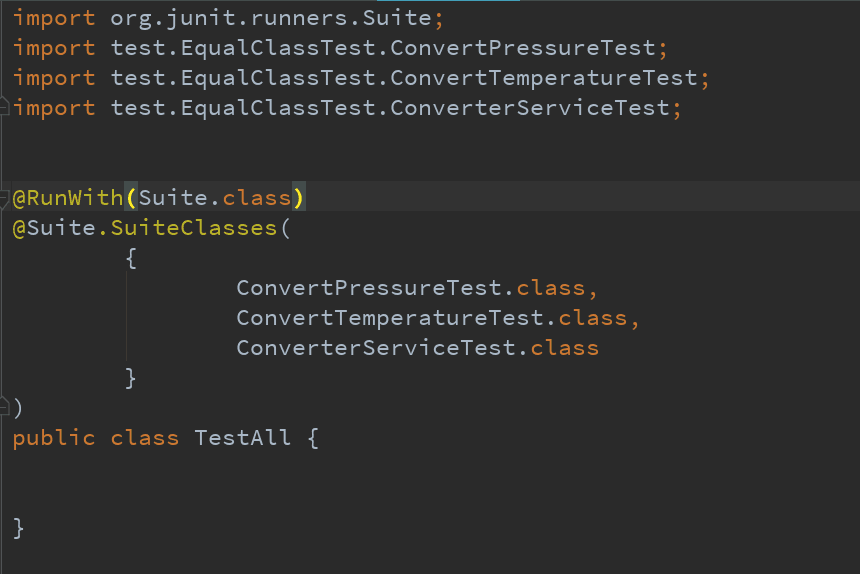
新的实现：



之后，我们尝试了不同的回归测试的用例选择策略，分别进行测试，从而便于分析和比较。

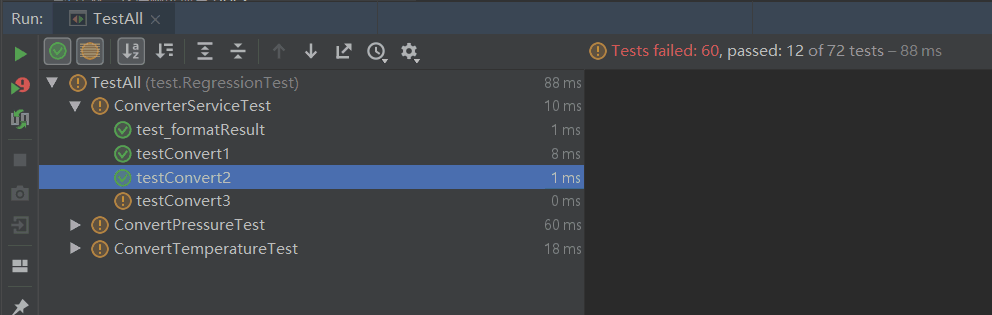
## 3.2、再测试全部用例

## 3.2.1、选择的策略介绍



TestAll方法采取的是重新测试所有的测试用例

## 3.2.2、测试结果



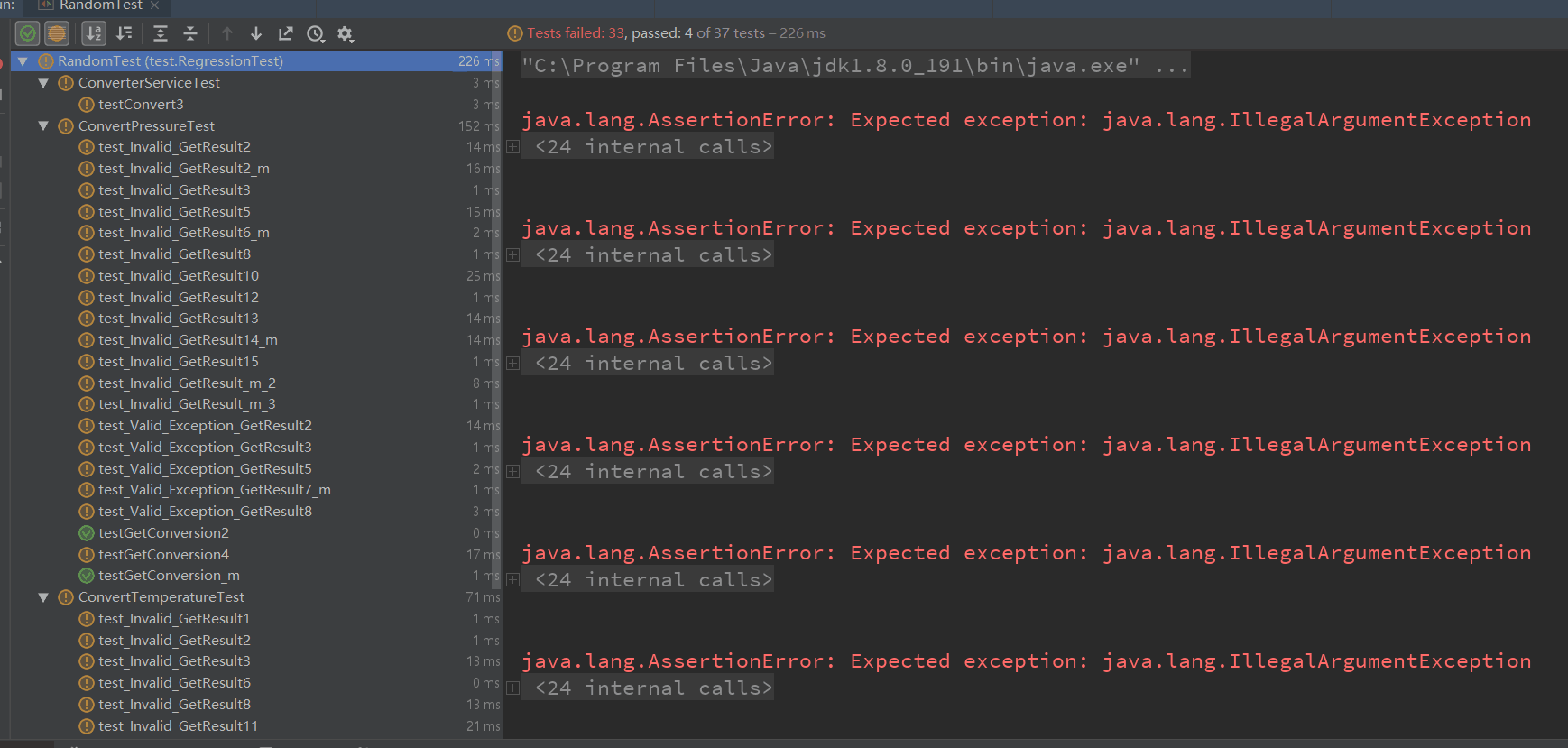
重新测试了全部的测试用例，72个用例中发现了60个测试不通过的情况。

## 3.3、随机测试部分用例

## 3.3.1、选择的策略介绍

从原有的测试用例中随机选择一部分用例（比例为0.5），来作为回归测试的测试用例。

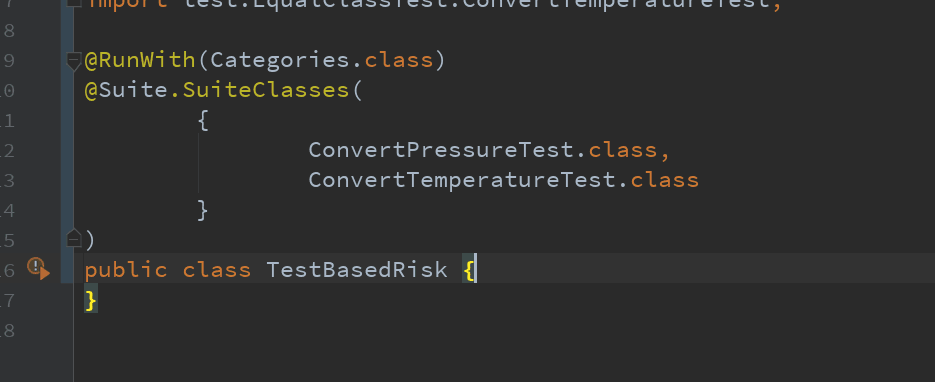
## 3.3.2、测试结果



测试用例的数量变少了，发现的问题也少了，37个用例中发现了33个测试不通过的情况。

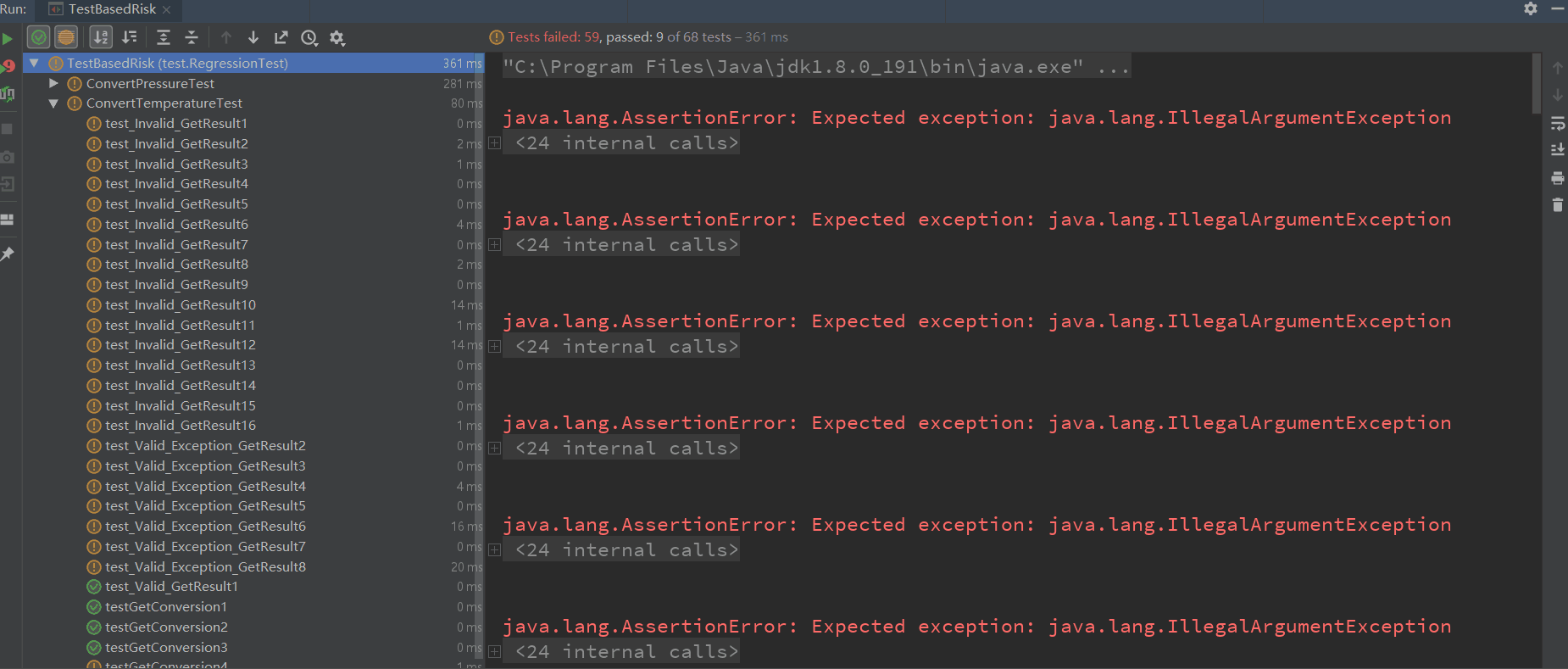
## 3.4、测试风险程度高的用例

## 3.4.1、选择的策略介绍



从原有的测试用例中选择出风险较高的用例，来作为回归测试的测试用例，针对我们的程序，我们分析需求和代码之后，认为风险程度最高的是 ConvertPressure和ConvertTemperature两个类，我们针对这两个类进行了基于风险的回归测试。

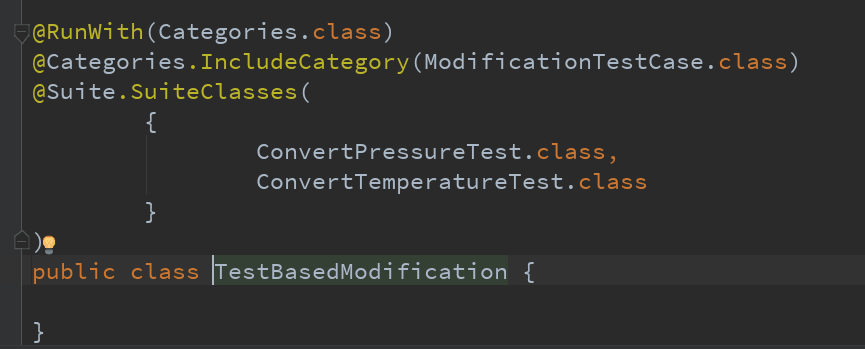
## 3.4.2、测试结果



虽然测试用例的数量变少了，但是测试仍然发现主要的问题，68个用例中发现了59个测试不通过的情况。

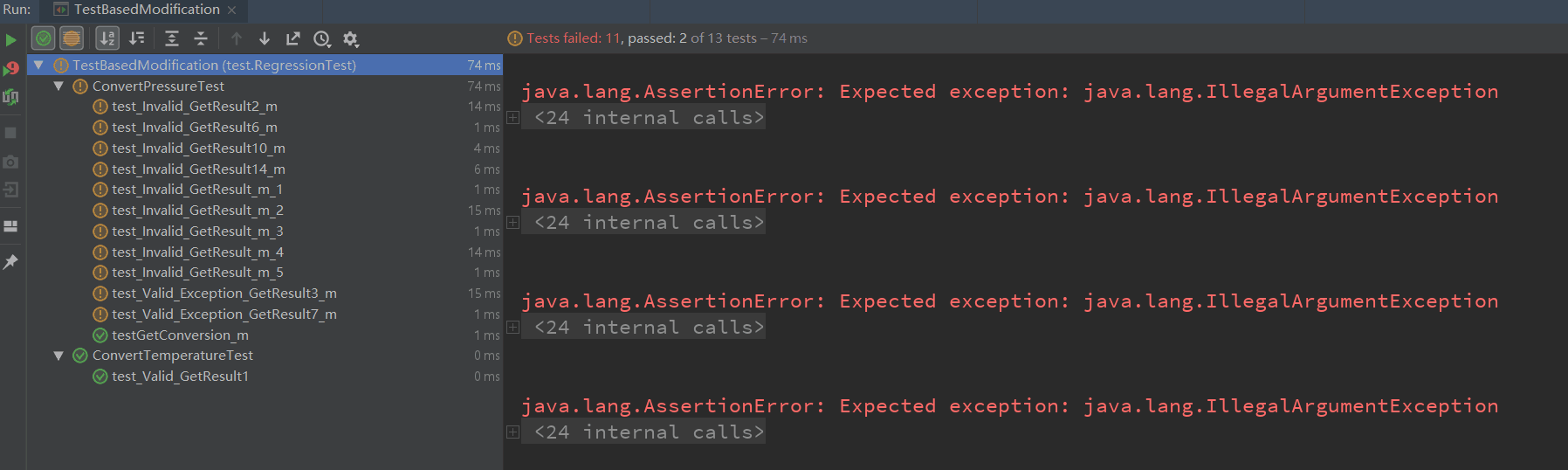
## 3.5、再测试修改的部分

## 3.5.1、选择的策略介绍



选择的测试用例能够覆盖代码修改的地方

## 3.5.2、测试结果



测试用例的数量变少了，也遗漏了一些缺陷，遗漏的缺陷是本来就存在于未修改的代码中的，13个用例中发现了11个测试不通过的情况，发现了修改部分代码。

# 对回归测试的分析

## 4.1、分析回归测试在软件产品迭代更新中的优点

在软件产品的迭代更新中，回归测试是必不可少的一部分。

我们经常会去修改之前的代码，但是修改旧代码很有可能会引入新的错误，可能会导致原来代码或者是其他代码的问题。同样，新加入程序的代码可能也会对旧的代码产生影响。所以说，我们需要进行回归测试，来确保代码的修改没有引入新的错误。否则的话，我们无法保证在软件产品迭代更新中原有的功能是否还保持完整。所以说，回归测试的优点之一就是它验证了修改的正确性及其影响。

此外，回归测试有效避免了缺陷的放大，可以说实在软件产品迭代更新中的第一道，也是最好的防线。随着软件开发的越来越大，开发时产生的问题的可能也会越来越大。如果不做回归测试的话，很有可能就将一个缺陷引入到一个系统中了，这样就把问题放大了，所以说回归测试可以降低系统测试、维护升级时候的成本，可以节约很多时间和金钱，它是测试过程中的重要组成部分，甚至在极端的编程方法中，每天多次进行回归测试也不过分。

另外，回归测试是需要时间的，不过我们可以通过选择不同的方法来缩短回归测试的时间。我们可以不用重新测试全部用例的方法，通过一个回归测试的选择方法来选取用于回归测试的测试用例，这样我们测试的用例就没有那么多，也不会占用特别多的时间。虽然这样可能也有一点风险，但是一般来说，这些测试用例如果覆盖了风险大的代码，或者覆盖了修改部分的代码，那么相对来说就是比较安全了。

## 4.2、分析所选回归测试方法对测试效率和安全性的影响

我们在这次的回归测试中尝试了不同的回归测试方法。这些不同的选择方法有的更加着重于回归测试的效率，有的更加注重于安全性。接下来我们比较一下这些不同的方法。

## 4.2.1、再测试全部用例

再测试全部用例指的是重新运行全部的测试用例。

这个方法虽然安全，但是时间的开销往往是无法接受的。随着不断的开发，测试用例会越来越多，这样的重复测试会带来相当大的工作量，往往会超出能承受的范围。

所以说这个方法的测试效率很低。

## 4.2.2、随机生成用例

这个方法指的是从原有的测试用例中随机选择一部分用例，来作为随机测试的测试用例。

这个方法非常节约时间，节约了人工去挑选测试用例的时间，同时因为只是随机选择一部分，进行测试的时间也变少了。这个方法的测试效率挺高的。

但是由于测试用例是随机选择出来的，它的安全性是难以得到保证的，可能会遗漏掉发现错误的测试用例。

## 4.2.3、基于风险选择测试用例

人工去分析哪些风险最高的代码部分，去优先选择那些关键的测试用例。

这个算是比较安全的方法了，毕竟就算非关键的代码出问题了，缺陷的严重性也不会太高。另外，测试的效率也大幅提升了。

这个回归测试方法不需要测试全部的用例，只需要选择其中一部分测试，时间就比再测试全部用例要短很多了。另外的时间开销也就只是去判断和维护各个测试用例或者模块的风险系数，有了制定好的风险标准，就可以方便地从中选择出测试用例了。

## 4.2.4、再测试修改的部分

人工去分析代码，选择的测试用例能够覆盖代码修改的地方。

这个方法需要去花时间分析哪些测试用例能够覆盖修改的代码。不过由于不是测试全部的测试用例，运行回归测试的时间少了很多。

不过从安全性的角度来说，还是存在一点点的缺陷。这样测试的话，只能验证本模块是否还存在缺陷，但不能验证其他和他有联系的模块会不会因为这次改动而引发错误。