软件测试实验报告

单元测试

|  |  |
| --- | --- |
| 班 级： | 1622302 |
| 姓 名： | 陈梓鹏 |
| 学 号： | 162230217 |

# 目录

1. 常见缺陷类型
2. 被测工具类
3. 实验步骤
4. 添加依赖
5. 注入错误
6. 设计断言
7. 编写main方法驱动待测类
8. 使用Junit实现单元测试
9. 实验方法分析比较
10. 结论
11. 附录

本次实验报告主要改进了以下两点：

1. 本次实验通过设置断言机制来自动审核测试结果，可以输出简单的测试报告。之前的实验并没有对测试结果进行自动核对，需要人工检查测试结果是否正确。
2. 本次实验在使用junit测试时增加了被测类之前没有测试到的模块，增加了测试的完备性。
3. **常见缺陷类型**

软件缺陷（Defect），即为计算机软件或程序中存在的某种破坏正常运行能力的问题、错误，或者隐藏的功能缺陷。缺陷的存在会导致软件产品在某种程度上不能满足用户的需要。IEEE729-1983对缺陷有一个标准的定义：从产品内部看，缺陷是软件产品开发或维护过程中存在的错误、毛病等各种问题；从产品外部看，缺陷是系统所需要实现的某种功能的失效或违背。

主要有以下几种类型：

F- Function ：影响了重要的特性、用户界面、产品接口、硬件结构接口和全局数据结构。并且设计文档需要正式的变更。如逻辑，指针，循环，递归，功能等缺陷。

A- Assignment： 需要修改少量代码，如初始化或控制块。如声明、重复命名，范围、限定等缺陷。

I- Interface： 与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表相互影响的缺陷。

C- Checking： 提示的错误信息，不适当的数据验证等缺陷。

B Build/package/merge ：由于配置库、变更管理或版本控制引起的错误。

D- Documentation： 影响发布和维护，包括注释。

G- Algorithm ：算法错误。

U-User Interface：人机交互特性：屏幕格式，确认用户输入，功能有效性，页面排版等方面的缺陷。

P-Performance：不满足系统可测量的属性值，如：执行时间，事务处理速率等。

N-Norms：不符合各种标准的要求，如编码标准、设计符号等。

1. **被测工具类**

本次实验选择org.apache.commons.math3.stat.descriptive.summary.Product作为被测对象，使用intellij idea建立maven工程作为测试项目。

主要功能简述：返回给定数组内数字的乘积，或者返回给定数组和给定权重的乘积。

主要功能代码添加在附录。

1. **实验步骤**
2. **添加依赖**

本次实验用到org.apache.common和junit两样工具。

<dependencies>  
 <dependency>  
 <groupId>org.apache.commons</groupId>  
 <artifactId>commons-math3</artifactId>  
 <version>3.6.1</version>  
 </dependency>  
  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>compile</scope>  
 </dependency>  
</dependencies>

1. **注入错误**

@Override  
public double evaluate(final double[] values, final int begin, final int length)  
 throws MathIllegalArgumentException {  
 double product = Double.*NaN*;  
 if (test(values, begin, length, true)) {  
 product = 0.0;

//product = 1.0  
 for (int i = begin; i < begin + length; i++) {  
 product \*= values[i];  
 }  
 }  
 return product;  
}

高亮处替换注释掉的代码。该缺陷使得一切返回值都为0。

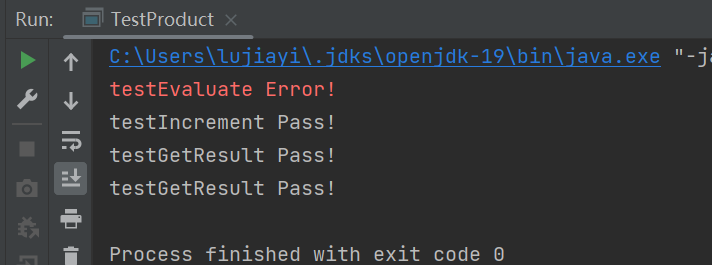
1. **设计断言机制**

public void assertEqual(double src,double des,String s)  
{  
 if(src!=des)  
 System.*err*.println(s+"Error!");  
 else System.*out*.println(s+"Pass!");  
}

针对被测对象的特点，只需要设计一种检查机制。并没有按照断言的定义，使程序直接中断运行，而是以输出错误提示信息的方式来告知程序出错。

1. **编写main方法驱动待测类**

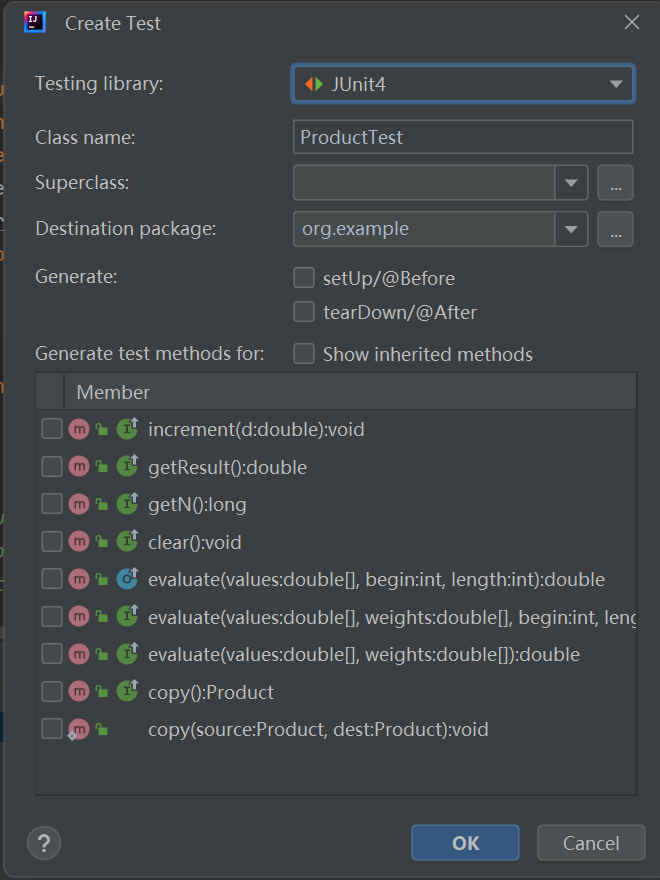
package org.example;  
  
public class TestProduct {  
 public static void main(String[] args)  
 {  
 TestProduct myTest= new TestProduct();  
 myTest.testEvaluate();  
 myTest.testIncrement();  
 myTest.testGetResult();  
 myTest.testClear();  
 }  
  
 public void assertEqual(double src,double des,String s)  
 {  
 if(src!=des)  
 System.*err*.println(s+"Error!");  
 else System.*out*.println(s+"Pass!");  
 }  
 public void testEvaluate()  
 {  
 Product p = new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 assertEqual(p.evaluate(values),120.0,"testEvaluate ");  
 }  
  
 public void testIncrement()  
 {  
 Product p = new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 int i;  
 for(i=0;i<values.length;i++)  
 {  
 p.increment(values[i]);  
 }  
 assertEqual(p.getN(),5,"testIncrement ");  
 }  
  
  
 public void testGetResult()  
 {  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 Product p = new Product();  
 int i;  
 for(i=0;i<values.length;i++)  
 {  
 p.increment(values[i]);  
 }  
 assertEqual(p.getResult(),120.0,"testGetResult ");  
 }  
  
  
 public void testClear()  
 {  
 Product p = new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 int i;  
 for(i=0;i<values.length;i++)  
 {  
 p.increment(values[i]);  
 }  
 p.clear();  
 assertEqual(p.getN(),0,"testGetResult ");  
 }  
}

测试结果如下：  
 

由于evaluate方法中被注入了错误，所以测试显示出错。

符合预期，测试正确。

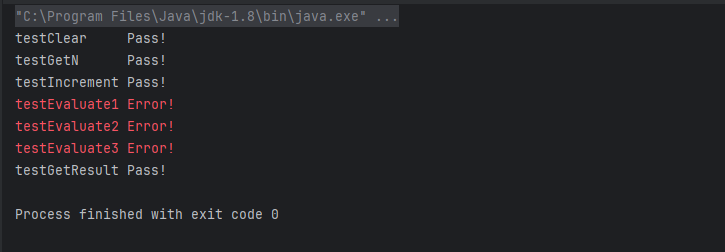
1. **使用Junit实现单元测试**



package test.org.example;   
  
import org.example.Product;  
import org.junit.Test;  
import org.junit.Before;   
import org.junit.After;   
  
*/\*\*   
\* Product Tester.   
\*   
\** ***@author*** *<Authors name>   
\** ***@since*** *<pre>4月 9, 2025</pre>   
\** ***@version*** *1.0   
\*/*public class ProductTest {   
  
@Before  
public void before() throws Exception {   
}   
  
@After  
public void after() throws Exception {   
}  
  
public void assertEqual(double src,double des,String s)  
{  
 if(src!=des)  
 System.*err*.println(s+"Error!");  
 else System.*out*.println(s+"Pass!");  
}  
  
  
*/\*\*   
\*   
\* Method: increment(final double d)   
\*   
\*/*@Test  
public void testIncrement() throws Exception {  
 Product p= new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 for(int i=0;i<values.length;i++)  
 {  
 p.increment(values[i]);  
 }  
 assertEqual(p.getN(),5, "testIncrement ");  
}   
  
*/\*\*   
\*   
\* Method: getResult()   
\*   
\*/*@Test  
public void testGetResult() throws Exception {   
  
 Product p= new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
  
 for(int i=0;i<values.length;i++)  
 {  
 p.increment(values[i]);  
 }  
  
 assertEqual(p.getResult(),120.0,"testGetResult ");  
}   
  
*/\*\*   
\*   
\* Method: getN()   
\*   
\*/*@Test  
public void testGetN() throws Exception {   
  
 Product p= new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
  
 for(int i=0;i<values.length;i++)  
 {  
 p.increment(values[i]);  
 }  
  
 assertEqual(p.getN(),5,"testGetN ");  
}   
  
*/\*\*   
\*   
\* Method: clear()   
\*   
\*/*@Test  
public void testClear() throws Exception {   
  
 Product p= new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
  
 for(int i=0;i<values.length;i++)  
 {  
 p.increment(values[i]);  
 }  
 p.clear();  
 assertEqual(p.getN(),0,"testClear ");  
}   
  
*/\*\*   
\*   
\* Method: evaluate(final double[] values, final int begin, final int length)   
\*   
\*/*@Test  
public void testEvaluateForValuesBeginLength() throws Exception {  
 Product p= new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 assertEqual(p.evaluate(values,0,5),120,"testEvaluate1 ");  
}   
  
*/\*\*   
\*   
\* Method: evaluate(final double[] values, final double[] weights, final int begin, final int length)   
\*   
\*/*@Test  
public void testEvaluateForValuesWeightsBeginLength() throws Exception {  
 Product p= new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 double[] weight = new double[] { 2.0,2.0,1.0,1.0,1.0 };  
 assertEqual(p.evaluate(values,weight,0,5),240,"testEvaluate2 ");  
}   
  
*/\*\*   
\*   
\* Method: evaluate(final double[] values, final double[] weights)   
\*   
\*/*@Test  
public void testEvaluateForValuesWeights() throws Exception {  
 Product p= new Product();  
 double[] values = new double[] { 1.0,2.0,3.0,4.0,5.0 };  
 double[] weight = new double[] { 2.0,2.0,1.0,1.0,1.0 };  
 assertEqual(p.evaluate(values,weight),240,"testEvaluate3 ");  
}   
}

为了增加测试的完备性，Junit比main方法多测试了三个类。

测试结果：



由测试结果可以看出，7个测试均可以运行，但是第4，5，6个测试由于Evaluate注入了错误，所以打印出错的结果。

符合预期，测试正确。

1. **实验方法分析比较**

综合实验结果，查阅资料后，可以总结出使用JUnit对比main方法测试的好处主要有五点：

1）可以书写一系列的测试方法，对项目所有接口或者方法进行单元测试。

2）启动后自动化测试，并判断执行结果, 不需要人为的干预。

3）只需要查看最后结果，就知道整个项目的方法接口是否通畅。。

4）每个单元测试用例相对独立，由Junit启动，自动调用。不需要添加额外的调用语句。

5）添加，删除，屏蔽测试方法，不影响其他的测试方法。开源框架都对JUnit有相应的支持。

而main方法对比JUnit则有以下劣势：

1）对多个方法调用，需要添加打印或者输出语句。

2）添加了新的测试方法，需要在main方法添加方法调用。

3）不能形成整体的测试结果。

4）需要对打印或者输出结果进行人为的判断。

但是JUnit对比main同样存在缺点：

1）JUnit不允许使用参数，main可以自定义添加参数。

2）JUnit只能进行单元测试，是一种单一工具。

3）JUnit 单元测试不支持多线程，如果想使用Junit 进行多线程测试，需要先睡眠主线程。

对于intellij idea来说，JUnit将所有的测试代码装封在一个类里，只要运行一个类就能进行批量的单元测试，使得main函数得到解放，这对工程来说也是至关重要的。

1. **结论**

JUnit是专一的单元测试工具，它提供了一个开源框架，可以更快地编写测试代码，提高工程质量。JUnit 测试可以自动运行，检查自己的结果，并提供即时反馈，没有必要通过测试结果报告来手动梳理。相对于main方法来说，JUnit自动化程度高，操作便捷，反馈结果及时且鲜明。但是同时也牺牲了一些自由度，使得开发人员不能自由提供参数。。

综上所述，在进行单元测试时推荐使用JUnit工具而不是main函数。

1. **附录**

/\*  
 \* Licensed to the Apache Software Foundation (ASF) under one or more  
 \* contributor license agreements. See the NOTICE file distributed with  
 \* this work for additional information regarding copyright ownership.  
 \* The ASF licenses this file to You under the Apache License, Version 2.0  
 \* (the "License"); you may not use this file except in compliance with  
 \* the License. You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/  
package org.apache.commons.math3.stat.descriptive.summary;  
  
import java.io.Serializable;  
  
import org.apache.commons.math3.stat.descriptive.AbstractStorelessUnivariateStatistic;  
import org.apache.commons.math3.exception.MathIllegalArgumentException;  
import org.apache.commons.math3.exception.NullArgumentException;  
import org.apache.commons.math3.stat.descriptive.WeightedEvaluation;  
import org.apache.commons.math3.util.FastMath;  
import org.apache.commons.math3.util.MathUtils;  
  
*/\*\*  
 \* Returns the product of the available values.  
 \* <p>  
 \* If there are no values in the dataset, then 1 is returned.  
 \* If any of the values are  
 \* <code>NaN</code>, then <code>NaN</code> is returned.</p>  
 \* <p>  
 \* <strong>Note that this implementation is not synchronized.</strong> If  
 \* multiple threads access an instance of this class concurrently, and at least  
 \* one of the threads invokes the <code>increment()</code> or  
 \* <code>clear()</code> method, it must be synchronized externally.</p>  
 \*  
 \*/*public class Product extends AbstractStorelessUnivariateStatistic implements Serializable, WeightedEvaluation {  
  
 */\*\* Serializable version identifier \*/* private static final long *serialVersionUID* = 2824226005990582538L;  
  
 */\*\*The number of values that have been added \*/* private long n;  
  
 */\*\*  
 \* The current Running Product.  
 \*/* private double value;  
  
 */\*\*  
 \* Create a Product instance  
 \*/* public Product() {  
 n = 0;  
 value = 1;  
 }  
  
 */\*\*  
 \* Copy constructor, creates a new {@code Product} identical  
 \* to the {@code original}  
 \*  
 \* @param original the {@code Product} instance to copy  
 \* @throws NullArgumentException if original is null  
 \*/* public Product(Product original) throws NullArgumentException {  
 *copy*(original, this);  
 }  
  
 */\*\*  
 \* {@inheritDoc}  
 \*/* @Override  
 public void increment(final double d) {  
 value \*= d;  
 n++;  
 }  
  
 */\*\*  
 \* {@inheritDoc}  
 \*/* @Override  
 public double getResult() {  
 return value;  
 }  
  
 */\*\*  
 \* {@inheritDoc}  
 \*/* public long getN() {  
 return n;  
 }  
  
 */\*\*  
 \* {@inheritDoc}  
 \*/* @Override  
 public void clear() {  
 value = 1;  
 n = 0;  
 }  
  
 */\*\*  
 \* Returns the product of the entries in the specified portion of  
 \* the input array, or <code>Double.NaN</code> if the designated subarray  
 \* is empty.  
 \* <p>  
 \* Throws <code>MathIllegalArgumentException</code> if the array is null.</p>  
 \*  
 \* @param values the input array  
 \* @param begin index of the first array element to include  
 \* @param length the number of elements to include  
 \* @return the product of the values or 1 if length = 0  
 \* @throws MathIllegalArgumentException if the array is null or the array index  
 \* parameters are not valid  
 \*/* @Override  
 public double evaluate(final double[] values, final int begin, final int length)  
 throws MathIllegalArgumentException {  
 double product = Double.*NaN*;  
 if (test(values, begin, length, true)) {  
 product=0.0;  
 //product = 1.0;  
 for (int i = begin; i < begin + length; i++) {  
 product \*= values[i];  
 }  
 }  
 return product;  
 }  
  
 */\*\*  
 \* <p>Returns the weighted product of the entries in the specified portion of  
 \* the input array, or <code>Double.NaN</code> if the designated subarray  
 \* is empty.</p>  
 \*  
 \* <p>Throws <code>MathIllegalArgumentException</code> if any of the following are true:  
 \* <ul><li>the values array is null</li>  
 \* <li>the weights array is null</li>  
 \* <li>the weights array does not have the same length as the values array</li>  
 \* <li>the weights array contains one or more infinite values</li>  
 \* <li>the weights array contains one or more NaN values</li>  
 \* <li>the weights array contains negative values</li>  
 \* <li>the start and length arguments do not determine a valid array</li>  
 \* </ul></p>  
 \*  
 \* <p>Uses the formula, <pre>  
 \* weighted product = &prod;values[i]<sup>weights[i]</sup>  
 \* </pre>  
 \* that is, the weights are applied as exponents when computing the weighted product.</p>  
 \*  
 \* @param values the input array  
 \* @param weights the weights array  
 \* @param begin index of the first array element to include  
 \* @param length the number of elements to include  
 \* @return the product of the values or 1 if length = 0  
 \* @throws MathIllegalArgumentException if the parameters are not valid  
 \* @since 2.1  
 \*/* public double evaluate(final double[] values, final double[] weights,  
 final int begin, final int length) throws MathIllegalArgumentException {  
 double product = Double.*NaN*;  
 if (test(values, weights, begin, length, true)) {  
 product=0.0;  
 //product = 1.0;  
 for (int i = begin; i < begin + length; i++) {  
 product \*= FastMath.*pow*(values[i], weights[i]);  
 }  
 }  
 return product;  
 }  
  
 */\*\*  
 \* <p>Returns the weighted product of the entries in the input array.</p>  
 \*  
 \* <p>Throws <code>MathIllegalArgumentException</code> if any of the following are true:  
 \* <ul><li>the values array is null</li>  
 \* <li>the weights array is null</li>  
 \* <li>the weights array does not have the same length as the values array</li>  
 \* <li>the weights array contains one or more infinite values</li>  
 \* <li>the weights array contains one or more NaN values</li>  
 \* <li>the weights array contains negative values</li>  
 \* </ul></p>  
 \*  
 \* <p>Uses the formula, <pre>  
 \* weighted product = &prod;values[i]<sup>weights[i]</sup>  
 \* </pre>  
 \* that is, the weights are applied as exponents when computing the weighted product.</p>  
 \*  
 \* @param values the input array  
 \* @param weights the weights array  
 \* @return the product of the values or Double.NaN if length = 0  
 \* @throws MathIllegalArgumentException if the parameters are not valid  
 \* @since 2.1  
 \*/* public double evaluate(final double[] values, final double[] weights)  
 throws MathIllegalArgumentException {  
 return evaluate(values, weights, 0, values.length);  
 }  
  
  
 */\*\*  
 \* {@inheritDoc}  
 \*/* @Override  
 public Product copy() {  
 Product result = new Product();  
 // No try-catch or advertised exception because args are valid  
 *copy*(this, result);  
 return result;  
 }  
  
 */\*\*  
 \* Copies source to dest.  
 \* <p>Neither source nor dest can be null.</p>  
 \*  
 \* @param source Product to copy  
 \* @param dest Product to copy to  
 \* @throws NullArgumentException if either source or dest is null  
 \*/* public static void copy(Product source, Product dest)  
 throws NullArgumentException {  
 MathUtils.*checkNotNull*(source);  
 MathUtils.*checkNotNull*(dest);  
 dest.setData(source.getDataRef());  
 dest.n = source.n;  
 dest.value = source.value;  
 }  
  
}