

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение*  *высшего образования*  ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных

технологий (МОСИТ)

ПРАКТИЧЕСКАЯ РАБОТА №2

по дисциплине

**«Тестирование и верификация программного обеспечения»**

Тема: **«Модульное тестирование»**

**Студенты группы** ИКБО-20-21

Фомичев Р.А.

Сидоров С.Д.

Опришко В.Д.

(подпись студента)

**Принял руководитель работы** Овчинникова М.А.

(подпись руководителя)

Практические работы выполнены « » 2023 г.

Зачтено « » 2023 г.

Содержание

[1. Цель работы 3](#_Toc146705517)

[1.1 Задание для выполнения 3](#_Toc146705518)

[1.2 Материальная часть 3](#_Toc146705519)

[1.2.1 Модульное тестирование 3](#_Toc146705520)

[1.2.3 Спецификация AAA 3](#_Toc146705521)

[2. Инструменты 3](#_Toc146705522)

[2.1. Используемое ПО 3](#_Toc146705523)

[3. Выполнение задания 4](#_Toc146705524)

[3.1. Сидоров С.Д. 4](#_Toc146705525)

[3.1.1. Документация к программному продукту. 4](#_Toc146705526)

[3.1.2. Тестирование программного продукта другим участником группы 5](#_Toc146705532)

[3.1.3. Проведение тестирования 6](#_Toc146705533)

[3.1.4. Оценка документации 6](#_Toc146705534)

[3.2. Опришко В.Д. 6](#_Toc146705535)

[3.2.1. Документация к программному продукту 6](#_Toc146705536)

[3.2.2. Тестирование программного продукта другим участником команды 7](#_Toc146705543)

[3.2.3. Проведение тестирования 8](#_Toc146705544)

[3.2.4. Оценка документации 9](#_Toc146705545)

[3.3. Фомичев Р.А. 9](#_Toc146705546)

[3.3.1. Документация к программному продукту 9](#_Toc146705547)

[3.3.2. Тестирование программного продукта другим участником группы 10](#_Toc146705555)

[3.3.3. Проведение тестирования 12](#_Toc146705556)

[3.3.4. Оценка документации 13](#_Toc146705557)

[4. Выводы 14](#_Toc146705558)

[5. Список использованных источников 15](#_Toc146705559)

[6. Дополнения и приложения 15](#_Toc146705560)

# Цель работы

## 1.1 Задание для выполнения

Составление документации к программному продукту, анализ документации другого участница группы, написание модульных тестов в соответствии с спецификацией AAA ( Arrange, Act, Assert).

## 1.2 Материальная часть

## 1.2.1 Модульное тестирование

Модульное тестирование -метод тестирования, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

## 1.2.3 Спецификация AAA

Спецификация AAA ( Arrange, Act, Assert) - (входные данные, действие, ожидаемый результат.

# Инструменты

## Используемое ПО

Для проведения тестирования программного продукта написанного на языке Java использовалась библиотека JUnit.

# Выполнение задания

## Сидоров С.Д.

## Документация к программному продукту.

Программный продукт состоит из класса Main, содержащего три метода.

**Метод 1** - main, сгенерированный метод для запуска программы и отладки в ручном режиме.

**Метод 2 - solveQuadraticEquation**.

Функционал: вычислений корней квадратного уравнения.

Входные данные:

Int a - коэффициент перед неизвестным во второй степени;

Int b - коэффициент перед неизвестным в первой степени;

Int c - свободный член уравнения;

Ограничения на входные данные: тип integer

Выходные данные:

Строка типа String;

Возможные варианты выходных данных:

“none” - уравнение не имеет корней;

“корень” - уравнение с одним корнем;

“корень/корень” - уравнение с двумя корнями;

**Метод 3 - solveCubicEquation.**

Функционал: вычисление корней кубического уравнения.

Входные данные:

Int a - коэффициент перед неизвестным в третьей степени;

Int b - коэффициент перед неизвестным во второй степени;

Int c - коэффициент перед неизвестным в первой степени;

Int d - свободный член уравнения;

Ограничения на входные данные: тип float.

Выходные данные:

Строка типа String;

Возможные варианты выходных данных

“none” - уравнение не имеет корней;

“корень” - уравнение с одним корнем;

“корень/корень” - уравнение с двумя корнями;

“корень/корень/корень” - уравнение с тремя корнями;



## Тестирование программного продукта другим участником группы

Модульные тесты для программного продукта Сидорова С.Д. представлены на листинге 1.

Листинг 1 – Модульные тесты для программного продукта

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class MainTest {

@Test

public void testSolveQuadraticEquation\_noRealSolutions() {

String result = Main.solveQuadraticEquation(1, 2, 5);

assertEquals("none", result);

}

@Test

public void testSolveQuadraticEquation\_oneRealSolution() {

String result = Main.solveQuadraticEquation(1, -2, 1);

assertEquals("1.0", result);

}

@Test

public void testSolveQuadraticEquation\_twoRealSolutions() {

String result = Main.solveQuadraticEquation(1, -3, 2);

assertEquals("2.0/1.0", result);

}

@Test

public void testSolveQuadraticEquation\_anotherTwoRealSolutions() {

String result = Main.solveQuadraticEquation(1, 5, 6);

assertEquals("-2.0/-3.0", result);

}

@Test

public void testSolveCubicEquation\_noRealSolutionsForQuadratic() {

String result = Main.solveCubicEquation(0, 1, 2, 5);

assertEquals("none", result);

}

@Test

public void testSolveCubicEquation\_allRealAndEqual() {

String result = Main.solveCubicEquation(1, 3, 3, 1);

assertEquals("-1.0", result);

}

@Test

public void testSolveCubicEquation\_twoEqualOneDifferent() {

String result = Main.solveCubicEquation(1, 0, 0, 0);

assertEquals("0.0", result);

}

@Test

public void testSolveCubicEquation\_allRealAndDifferent() {

String result = Main.solveCubicEquation(1, -6, 11, -6);

assertEquals("3.0/2.0/1.0", result);

}

}

## Проведение тестирования

Таблица 1 – результаты проведения тестирования метода solveQuadraticEquation

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Полученный результат** |
| 1 2 5 | none | none |
| 1 -2 1 | 1.0 | 1.0 |
| 1 -3 2 | 2.0/1.0 | 2.0/1.0 |
| 1 5 6 | -2.0/-3.0 | -2.0/-3.0 |

Таблица 2 – результаты проведения тестирования метода solveCubicEquation

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Полученный результат** |
| 0 1 2 5 | none | none |
| 1 3 3 1 | -1.0 | -1.0 |
| 1 0 0 0 | 0.0 | -0.0 |
| 1 -6 11 -6 | 3.0/2.0/1.0 | 2.0/-1.0 |

Результаты тестирования представлены на рисунке 1.

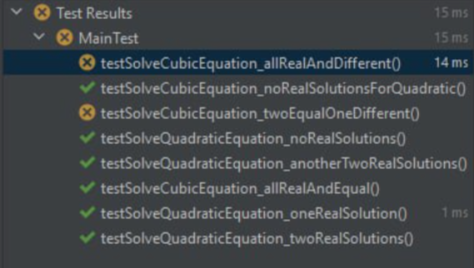


Рисунок 1 – Результаты тестирования

## Оценка документации

В ходе анализа документации ошибок не было выявлено. В ходе проведённого тестирования не было выявлено ошибок, код отрабатывает во всех ожидаемых случаях.

## Опришко В.Д.

## Документация к программному продукту

Программный продукт состоит из класса Main, содержащего три метода.

**Метод 1 - Main**, сгенерированный метод для запуска программы и отладки в ручном режиме.

**Метод 2 - isPrime.**

Функционал: Проверка числа на простоту.

Входные данные:

int number - число для проверки на простоту.

Ограничения на входные данные: переменная типа integer.

Выходные данные:

Булево значение true или false.

Возможные варианты выходных данных:

true - число является простым.

false - число не является простым.

**Метод 3 - getNthFibonacci.**

Функционал: Вычисление n-го числа Фибоначчи.

Входные данные:

int n - порядковый номер числа в последовательности Фибоначчи.

Ограничения на входные данные: n должно быть неотрицательным.

Выходные данные:

Целое число, представляющее n-е число Фибоначчи.

Возможные варианты выходных данных:

Любое неотрицательное целое число, соответствующее порядковому номеру в последовательности Фибоначчи.



## Тестирование программного продукта другим участником команды

Модульные тесты для программного продукта Опришко В.Д. представлены на листинге 2.

Листинг 2 – Модульные тесты для программного продукта

import org.junit.\*;

public class MainTest {

@Test

public void basicTestFibonacci() {

int actualFibonacci = ClassForTest.getNthFibonacci(0);

Assert.assertEquals(actualFibonacci, 0);

}

@Test

public void equalTestFibonacci() {

int actualFibonacci = ClassForTest.getNthFibonacci(1);

Assert.assertEquals(actualFibonacci, 1);

}

@Test

public void negativeTestFibonacci() {

int actualFibonacci = ClassForTest.getNthFibonacci(2);

Assert.assertEquals(actualFibonacci, 1);

}

Продолжение листинга 2

@Test

public void oneTestFibonacci() {

int actualFibonacci = ClassForTest.getNthFibonacci(3);

Assert.assertEquals(actualFibonacci, 2);

}

@Test

public void basicTestPrime() {

boolean actualPrime = ClassForTest.isPrime(33);

Assert.assertEquals(actualPrime, false);

}

@Test

public void equalTestPrime() {

boolean actualPrime = ClassForTest.isPrime(11);

Assert.assertEquals(actualPrime, true);

}

@Test

public void negativeTestPrime() {

boolean actualPrime = ClassForTest.isPrime(-2);

Assert.assertEquals(actualPrime, false);

}

@Test

public void oneTestPrime() {

boolean actualPrime = ClassForTest.isPrime(1);

Assert.assertEquals(actualPrime, false);

}

}

## Проведение тестирования

Таблица 3 – результаты проведения тестирования метода isFibonacci

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Полученный результат** |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 1 | 1 |
| 3 | 2 | 2 |

Таблица 4 – результаты проведения тестирования метода isPrime

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Полученный результат** |
| 33 | false | false |
| 11 | true | true |
| -2 | false | false |
| 1 | false | false |

Результаты тестирования представлены на рисунке 2.

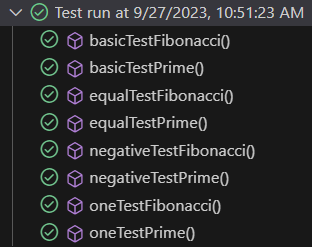


Рисунок 2 – Результаты тестирования

## Оценка документации

В ходе анализа документации ошибок не было выявлено. В ходе проведённого тестирования не было выявлено ошибок, код отрабатывает во всех ожидаемых случаях.

## Фомичев Р.А.

## Документация к программному продукту

Программный продукт состоит из класса Main, содержащего три метода.

**Метод 1** - main, сгенерированный метод для запуска программы и отладки в ручном режиме.

**Метод 2 - NOD**.

Функционал: вычислений наибольшего общего делителя двух чисел.

Входные данные:

Int a – первое число;

Int b – второе число;

Ограничения на входные данные: тип integer

Выходные данные:

Число типа integer;

Возможные варианты выходных данных:

“число” – наибольший общий делитель двух чисел;

**Метод 3 - NOK.**

Функционал: вычислений наименьшего общего кратного двух чисел.

Входные данные:

Int a – первое число;

Int b – второе число;

Ограничения на входные данные: тип integer

Выходные данные:

Число типа integer;

Возможные варианты выходных данных:

“число” – наименьший общее­ кратное двух чисел;



## Тестирование программного продукта другим участником группы

Модульные тесты для программного продукта Фомичева Р.А. представлены на листинге 3.

Листинг 3 – Модульные тесты программного продукта

import main.ClassForTest;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.testng.Assert;

class NodTest {

@Test

@DisplayName("Check nod basic correctness")

void basicTestNod() {

int actualNod = ClassForTest.NOD(33, 44);

Assert.assertEquals(actualNod, 11);

}

@Test

@DisplayName("Check nod equal correctness")

void equalTestNod() {

int actualNod = ClassForTest.NOD(11, 11);

Assert.assertEquals(actualNod, 11);

}

@Test

@DisplayName("Check nod b > a correctness")

void orderTestNod() {

int actualNod = ClassForTest.NOD(12, 4);

Assert.assertEquals(actualNod, 4);

}

@Test

@DisplayName("Check nod is one correctness")

void oneTestNod() {

int actualNod = ClassForTest.NOD(121, 131);

Assert.assertEquals(actualNod, 1);

Продолжение листинга 3

}

}

class NokTest {

@Test

@DisplayName("Basic nok test")

void basicTestNok() {

int actualNok = ClassForTest.NOK(4, 3);

Assert.assertEquals(actualNok, 12);

}

@Test

@DisplayName("Basic dif order nok test")

void basicDifOrderTestNok() {

int actualNok = ClassForTest.NOK(3, 4);

Assert.assertEquals(actualNok, 12);

}

@Test

@DisplayName("Equal nok test")

void equalTestNok() {

int actualNok = ClassForTest.NOK(4, 4);

Assert.assertEquals(actualNok, 4);

}

@Test

@DisplayName("Below zero A nok test")

void belowZeroATestNok() {

int actualNok = ClassForTest.NOK(-4, 3);

Assert.assertEquals(actualNok, 12);

}

@Test

@DisplayName("Below zero B nok test")

void belowZeroBTestNok() {

int actualNok = ClassForTest.NOK(4, -3);

Assert.assertEquals(actualNok, 12);

}

@Test

@DisplayName("Below zero A and B nok test")

void belowZeroABTestNok() {

int actualNok = ClassForTest.NOK(-4, -3);

Assert.assertEquals(actualNok, 12);

Продолжение листинга 3

}

@Test

@DisplayName("Zero A nok test")

void zeroATestNok() {

int actualNok = ClassForTest.NOK(0, 3);

Assert.assertEquals(actualNok, 0);

}

@Test

@DisplayName("Zero B nok test")

void zeroBTestNok() {

int actualNok = ClassForTest.NOK(3, 0);

Assert.assertEquals(actualNok, 0);

}

@Test

@DisplayName("Zero A and B nok test")

void zeroABTestNok() {

int actualNok = ClassForTest.NOK(0, 0);

Assert.assertEquals(actualNok, 0);

}

}

## Проведение тестирования

Таблица 5 – результаты проведения тестирования метода NOD

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Полученный результат** |
| 33 44 | 11 | 11 |
| 11 11 | 11 | 11 |
| 12 4 | 4 | 4 |
| 121 131 | 1 | 1 |

Таблица 6 - Результаты тестирования метода NOK

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Полученный результат** |
| 4 3 | 12 | 12 |
| 3 4 | 12 | 12 |
| 4 4 | 4 | 4 |
| -4 3 | 12 | 12 |
| 4 -3 | 12 | 12 |
| -4 -3 | 12 | 12 |
| 0 3 | 0 | 0 |
| 3 0 | 0 | 0 |
| 0 0 | 0 | 0 |

Результаты тестирования представлены на рисунках 3-4.

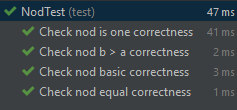


Рисунок 3 – Результаты тестирования метода NOD.

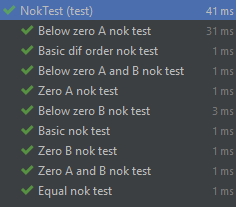


Рисунок 4 – Результаты тестирования метода NOK.

## Оценка документации

В ходе анализа документации ошибок не было выявлено. В ходе проведённого тестирования не было выявлено ошибок, код отрабатывает во всех ожидаемых случаях.

# Выводы

В ходе данной практической работы были продемонстрированы возможности тестирования программного продукта с помощью модульного тестирования согласно паттерну AAA.

# Список использованных источников

1. Статический анализ кода [Электронный ресурс] – <https://www.jetbrains.com/ru-ru/resharper/features/code_analysis.html>
2. Динамический анализ кода [Электронный ресурс] – <https://habr.com/ru/companies/pvs-studio/articles/580196/>
3. WebStrom – анализ кода [Электронный ресурс] – <https://www.jetbrains.com/ru-ru/resharper/features/code_analysis.html>
4. Использование статического и динамического анализа для повышения качества продукции и эффективности разработки [Электронный ресурс] – <https://www.swd.ru/print.php3?pid=828>

# Дополнения и приложения