Министерство образования и науки Российской Федерации

ФГБОУ ВО Рыбинский государственный авиационный технический университет имени П.А. Соловьева

Факультет радиоэлектроники и информатики

Кафедра математического и программного обеспечения электронных вычислительных средств

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине

Тестирование и отладка программного обеспечения

по теме

Интеграционное тестирование

Студенты группы ИПБ-13 — Болотин Д. И., Ивашин А.В.

Преподаватель к.т.н., ст. преп. — Воробьев К. А.

Рыбинск 2016

Содержание

1. Общее описание тестируемой системы

Прект предназначен для шифрования/дешифрования текстов методами Моноалфавитной замены, Побитовой перестановки. На Рисунке 1 представлена диаграмма классов проекта.

После запуска программы пользователю требуется пройти авторицазию или регистрацию, после чего ему доступны 2 метода шифрования. Во время работы можно сохранить/загрузить текстовый файл.

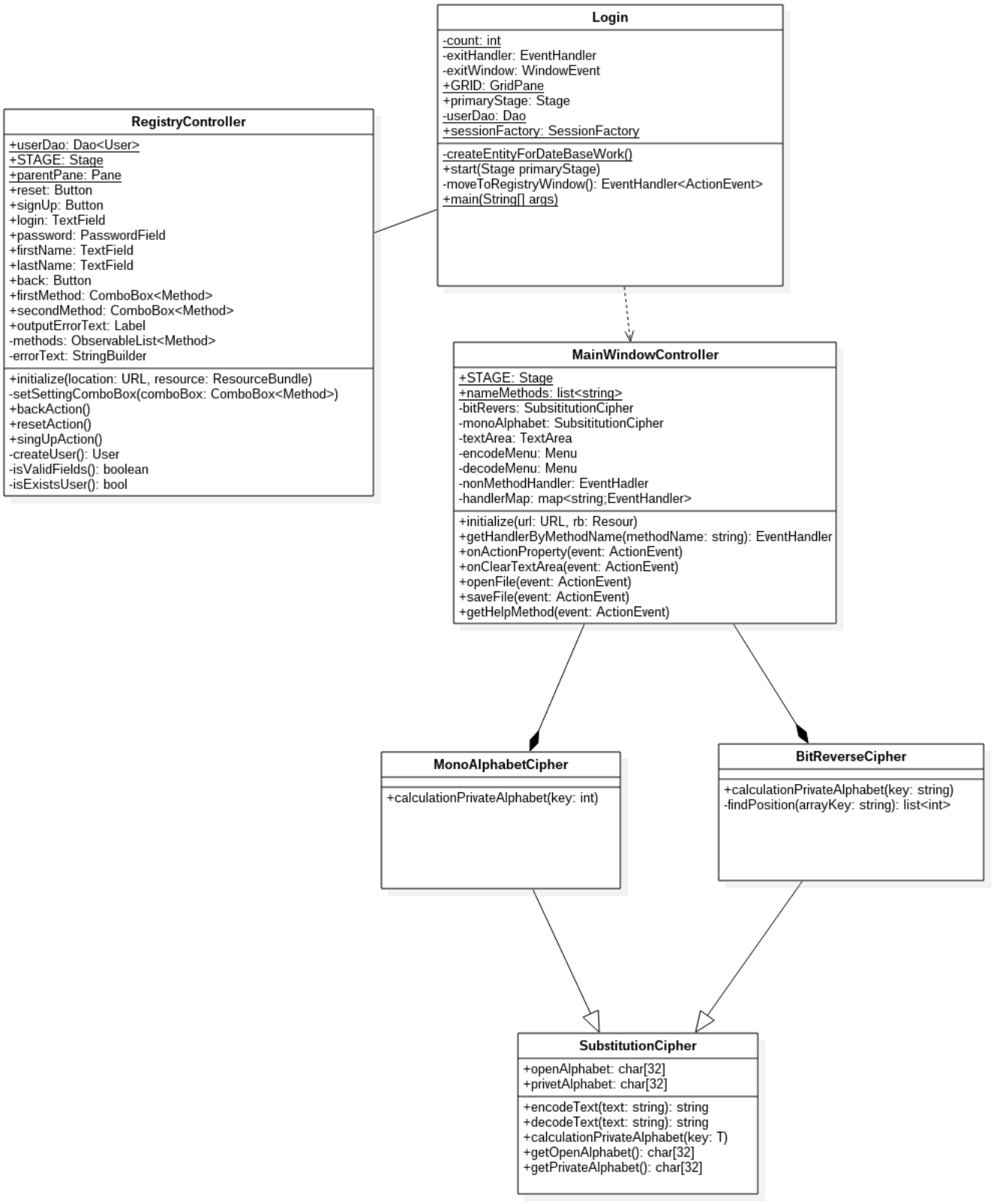

Рис. 1 – Диаграмма классов проекта

2. Общее описание тестируемых классов

В ходе данного тестирования проверяется только взаимодействие классов программы между собой, взаимодействие с другими системами не проверяется.

Тестированию подвержены классы, взаимодействующие друг с другом непосредственно, а также взаимодействие MonoAlphabetCipher и BitReversCipher посредством комбинации различных способов шифрования/дешифрования. Тестирование добавление пользователя в БД в данной работе не производится.

Для проведения тестирования использована библиотека JUnit, а также TestFX, которая нужна для моделирования взаимодействия пользователя с интерфейсом.

2.1. Ограничения для шифруемого/расшифруемого текста

Алфавит текста состоит из строчных букв русского алфавита и пробела кроме букв 'ё' и 'й'.

3. Общее описание тестирующих классов

Для проведения интеграционного тестирования была разработана система классов для тестирования пользовательского интерфейса(т.к. почти все классы программы взаимодействуют только через него), а также класс для тестирования взаимодействия методов шифрования (под названием IntegrationCipherTest). Диаграмма классов для тестирования пользовательского Gui представлена на рисунке 2, на ней упущены имена методов и атрибутов.

Рис. 2 – Тут будет диаграмма тестирующих классов, честно:)

4. Тестирование взаимодействия класса Login

Это главный класс программы. Он отвечает за окно авторизации. Пользователь должен пройти авторизацию или зарегистрироваться. Если после пятой попытки не удалось авторизиваться, то программа закрывается.

4.1. Закрытие программы при неверной авторизации

С помощью TestFx моделируется попытка неудачной авторизации (Листинг 1).

Листинг 1 – Тестирование неверной авторизации

```
public class LoginUiCloseTest extends LoginUiCustomTest {
    @Test
    public void threeWrongAuthorizationTest() {
        for (int i = 0; i < 3; i++) {
            clickOn(userName).write("Aleksey");
            clickOn(password).write("tttt");
            clickOn(authorization).sleep(100);
            verifyThat(resultAuthorization, hasText("Не верный пароль"));
            assertEquals(resultAuthorization.getFill(), Color.FIREBRICK);
            userName.clear();
            password.clear();
        }
        assertEquals(stage.isShowing(), false);
        assertEquals(sessionFactory.isClosed(), true);
    }
}
```

4.2. Реакция программы при неверном пароле

В окне должна появиться надпись "Неверный пароль!". Этот и последующие тесты класса Login описаны в классе LoginUiTest (Листинг 2).

Листинг 2 – Тестирование ввода неверного пароля

```
    @Test
    public void wrongAuthorizationTest() {
        assertEquals(sessionFactory.isClosed(), false);
        clickOn(userName).write("Aleksey");
        clickOn(password).write("retdfgdfh");
        clickOn(authorization);
        verifyThat(resultAuthorization, hasText("Не верный пароль"));
        assertEquals(resultAuthorization.getFill(), Color.FIREBRICK);
    }
```

4.3. Реакция программы при не существующем логине

Должно появиться сообщение "Такого пользователя не существует"(Листинг **??**).

Листинг 3 – Тестирование ввода неверного пароля

```
    @Test
```

```
    public void wrongAuthorizationTest() {
        assertEquals(sessionFactory.isClosed(), false);
        clickOn(userName).write("Aleksey");
        clickOn(password).write("retdfgdfh");
        clickOn(authorization);
        verifyThat(resultAuthorization, hasText("Не верный пароль"));
        assertEquals(resultAuthorization.getFill(), Color.FIREBRICK);
    }
```

4.4. Реакция программы при верной авторизации

При верной авторизации должно исчезнуть начальное окно и появиться основное окно программы (Листинг 4).

Листинг 4 – Тестирование ввода неверного пароля

```
    @Test
    public void openMainWindowTest() {
        assertEquals(sessionFactory.isClosed(), false);
        clickOn(userName).write("Aleksey");
        clickOn(password).write("yui");
        clickOn(authorization);
        verifyThat(resultAuthorization, hasText("Пароль верный"));
        assertEquals(resultAuthorization.getFill(), Color.GREEN);
        waitUntil("#AnchorPane", visible());
    }
```

4.5. Тестирование перехода к окну регистрации

При нажатии на кнопку “Регистация” , должно появится соответствующее окно, а окно для авторизации исчезнуть (Листинг 5).

Листинг 5 – Тестирование перехода к окну регистрации

```
    @Test
    public void openRegistryWindowTest() {
        openRegistryWindowAndTest();
        closeRegistryWindowAndTest();
    }
```

5. Тестирование взаимодействия классов MonoAlphabetCipher и BitReverseCipher

Тестирование осуществляется посредством многократного шифрования и расшифрования текста различными методами. После последовательного шифрования и расшифрования разными методами с неизменными ключами текст должен совпасть с исходным. Если провести такой же тест, но при расшифровании изменить ключь одного из методов, то результат должен не совпасть с исходным. Далее проведём тест, при котором зашуфруем несколько раз обоими методами с разнами ключами и расшуфруем в обратном порядке с соответствующими ключами, результат должен совпасть с исходным. Все эти три теста описаны в классе IntegrationCipherTest, который представлен в листинге 6.

Листинг 6 – класс IntegrationCipherTest

```
1  public class IntegrationCipherTest {
2      private static SubstitutionCipher<Integer> monoAlphabet;
3      private static SubstitutionCipher<String> bitRevers;
4      private String actualText;
5
6      @BeforeClass
7      public static void test() {
8          monoAlphabet = new MonoAlphabetCipher();
9          bitRevers = new BitReversCipher();
10     }
11
12     @Before
13     public void createCipherClass() {
14         actualText = "специальныи текст для тестов";
15     }
16
17     @Test
18     public void integrationCipherEqualsTest() {
19         monoAlphabet.calculationPrivateAlphabet(4);
20         bitRevers.calculationPrivateAlphabet("32451");
21         String expectedText = monoAlphabet.encodeText(actualText);
22         expectedText = bitRevers.decodeText(expectedText);
23         expectedText = bitRevers.encodeText(expectedText);
24         expectedText = monoAlphabet.decodeText(expectedText);
25         Assert.assertEquals(expectedText, actualText);
26     }
27
28     @Test
29     public void  integrationCipherNotEqualsTest() {
30         monoAlphabet.calculationPrivateAlphabet(4);
31         bitRevers.calculationPrivateAlphabet("32451");
32         String testText = monoAlphabet.encodeText(actualText);
33         testText = bitRevers.decodeText(testText);
34         bitRevers.calculationPrivateAlphabet("52341");
35         String expectedText = bitRevers.encodeText(testText);
36         Assert.assertNotEquals(expectedText, testText);
37     }
38
39     @Test
```

```
40      public void  integrationDeepCipherEqualsTest() {
41          monoAlphabet.calculationPrivateAlphabet(2);
42          String expectedText = monoAlphabet.encodeText(actualText);
43
44          bitRevers.calculationPrivateAlphabet("53241");
45          expectedText = bitRevers.encodeText(expectedText);
46          Assert.assertNotEquals(expectedText, actualText);
47
48          monoAlphabet.calculationPrivateAlphabet(3);
49          expectedText = monoAlphabet.decodeText(expectedText);
50
51          bitRevers.calculationPrivateAlphabet("24531");
52          expectedText = bitRevers.decodeText(expectedText);
53          Assert.assertNotEquals(expectedText, actualText);
54
55          expectedText = bitRevers.encodeText(expectedText);
56          expectedText = monoAlphabet.encodeText(expectedText);
57          Assert.assertNotEquals(expectedText, actualText);
58
59          bitRevers.calculationPrivateAlphabet("53241");
60          expectedText = bitRevers.decodeText(expectedText);
61
62          monoAlphabet.calculationPrivateAlphabet(2);
63          expectedText = monoAlphabet.decodeText(expectedText);
64
65          Assert.assertEquals(expectedText, actualText);
66      }
67  }
```

6. Тестирование взаимодействия класса MainWindow

Этот класс отвечает за главное окно программы и взаимодействует с классами, реализующими методы шифрования. Проведём тестирование, аналогичное тестированию классов MonoAlphabetCipher и BitReverseCipher, но теперь классы для шифрования будут взаимодействовать не напрямую, а через MainWindow(как и происходит на сомом деле), посредством моделирвания работы пользователя с gui. Оно представлено в листинге 7. Предедушее тестирование направлено на проверку корректности классов шифрования и их взаимодействия, а такущее преимущественно на корректность работы MainWindow с данными классами.

Листинг 7 – класс MainWindowUiIntegrationTest

```
public class MainWindowUiIntegrationTest extends CustomMainWindowUiTest {
    private TextArea textArea;
    String actualText;

    @BeforeClass
    public static void mainWindowSettings() {
        MainWindowController.nameMethods = Arrays.asList(
                methodNames.get(0), methodNames.get(5)
        );
    }

    @Before
    public void findTextArea() {
        textArea = find("#textArea");
        actualText = "специальныи текст для тестов";
        robot.clickOn(textArea).write(actualText);
    }

    @After
    public void clearTextArea() {
        textArea.clear();
    }

    @Test
    public void integrationCipherEqualsTest() {
        robot.clickOn("#encodeMenu").clickOn("#encodeMonoAlphabet")
                .clickOn("#txtFieldDialog").write('4').clickOn("#okDialog");
        robot.clickOn("#decodeMenu").clickOn("#decodeBitRevers")
                .clickOn("#txtFieldDialog").write("32451").clickOn("#okDialog");
        robot.clickOn("#encodeMenu").clickOn("#encodeBitRevers")
                .clickOn("#txtFieldDialog").write("32451").clickOn("#okDialog");
        robot.clickOn("#decodeMenu").clickOn("#decodeMonoAlphabet")
                .clickOn("#txtFieldDialog").write('4').clickOn("#okDialog");
        Assert.assertEquals(textArea.getText(), actualText);
    }

    @Test
    public void integrationCipherNotEqualsTest() {
        robot.clickOn("#encodeMenu").clickOn("#encodeMonoAlphabet")
                .clickOn("#txtFieldDialog").write('4').clickOn("#okDialog");
        robot.clickOn("#decodeMenu").clickOn("#decodeBitRevers")
                .clickOn("#txtFieldDialog").write("32451").clickOn("#okDialog");
```

```
43          String testText = textArea.getText();
44          robot.clickOn("#decodeMenu").clickOn("#decodeBitRevers")
45                  .clickOn("#txtFieldDialog").write("52341").clickOn("#okDialog");
46          Assert.assertNotEquals(textArea.getText(), testText);
47      }
48
49      @Test
50      public void  integrationDeepCipherEqualsTest() {
51          robot.clickOn("#encodeMenu").clickOn("#encodeMonoAlphabet")
52                  .clickOn("#txtFieldDialog").write("2").clickOn("#okDialog");
53          robot.clickOn("#encodeMenu").clickOn("#encodeBitRevers")
54                  .clickOn("#txtFieldDialog").write("53241").clickOn("#okDialog");
55          Assert.assertNotEquals(textArea.getText(), actualText);
56          robot.clickOn("#decodeMenu").clickOn("#decodeMonoAlphabet")
57                  .clickOn("#txtFieldDialog").write('3').clickOn("#okDialog");
58          robot.clickOn("#decodeMenu").clickOn("#decodeBitRevers")
59                  .clickOn("#txtFieldDialog").write("24531").clickOn("#okDialog");
60          Assert.assertNotEquals(textArea.getText(), actualText);
61          robot.clickOn("#encodeMenu").clickOn("#encodeBitRevers")
62                  .clickOn("#txtFieldDialog").write("24531").clickOn("#okDialog");
63          robot.clickOn("#encodeMenu").clickOn("#encodeMonoAlphabet")
64                  .clickOn("#txtFieldDialog").write('3').clickOn("#okDialog");
65          Assert.assertNotEquals(textArea.getText(), actualText);
66          robot.clickOn("#decodeMenu").clickOn("#decodeBitRevers")
67                  .clickOn("#txtFieldDialog").write("53241").clickOn("#okDialog");
68          robot.clickOn("#decodeMenu").clickOn("#decodeMonoAlphabet")
69                  .clickOn("#txtFieldDialog").write('2').clickOn("#okDialog");
70          Assert.assertEquals(textArea.getText(), actualText);
71      }
72  }
```

7. Выводы

В ходе работы с помошью библиотек JUnit и TestFX было разработано интеграционное тестироваие проекта. Тесты составлялись из рассчёта на то, что всякое непосредственное взаимодеёствие классов программы должно быть покрыто. В результате тестирования была обнаружена ошибка, не замеченная при модульном тестровании: в базовом классе SubstitutionCipher поле privetAlphabet (закрытый алфавит - правила сопоставления символов при шифровании) было объявлено как static, т.е. все наследники этого класса имели один закрытый алфавит на всех, актуальный только для класса, который последний обновил это поле. Такое поведение классов шифрования недопустимо, но не было замечено на предыдущем этапе. Других ошибок обнаружено не было.

8. Приложения

Это старые приложения, я их обновлю.

Листинг 8 – код модуля SubstitutionCipher.java

```
package encryptionMethods.base;

/**
 * Created by Алексей on 16.12.2016.
 */
public abstract class SubstitutionCipher<T> {
    // Откртый алфавит
    protected static final char[] openAlphabet = new char[32];
    // Закрытый алфавит
    protected static final char[] privateAlphabet = new char[32];

    public SubstitutionCipher() {
        createOpenAlphabet();
    }

    private void createOpenAlphabet() {
        openAlphabet[0] = '\u0020';
        for (int i = 0; i < 9; i++) {
            openAlphabet[i + 1] = (char) ('a' + i);
        }
        for (int i = 10; i < 32; ++i) {
            openAlphabet[i] = (char) ('a' + i);
        }
    }

    public abstract void calculationPrivateAlphabet(T key);

    /**
     * Закодировать текст
     *
     * @param originalText текст
     * @return закодированный текст
     */
    public String encodeText(String originalText) {
        originalText = originalText.replaceAll("\n", " ").toLowerCase();
        char[] text = originalText.toCharArray();
        char[] result = new char[text.length];
        int index;
        for (int i = 0; i < text.length; i++) {
            index = contains(openAlphabet, text[i]);
            if (index < 0) return "Недопустимый символ. Шифрование не возможно";
            result[i] = privateAlphabet[index];
        }
        return String.valueOf(result);
    }

    /**
     * Раскодировать текст
     *
     * @param originalText текст
```

```
51      * @return раскодированный текст
52      */
53     public String decodeText(String originalText) {
54         char[] text = originalText.toCharArray();
55         char[] result = new char[text.length];
56         int index;
57         for (int i = 0; i < text.length; i++) {
58             index = contains(privateAlphabet, text[i]);
59             if (index < 0) return "Недопустимый символ. Расшифровка не возможно";
60             result[i] = openAlphabet[index];
61         }
62         return String.valueOf(result);
63     }
64 
65     private int contains(char[] chars, char symbol) {
66         for (int i = 0; i < chars.length; i++) {
67             if (chars[i] == symbol) {
68                 return i;
69             }
70         }
71         return -1;
72     }
73 
74     public static char[] getOpenAlphabet() {
75         return openAlphabet;
76     }
77 
78     public static char[] getPrivateAlphabet() {
79         return privateAlphabet;
80     }
81 }
```

Листинг 9 – код модуля MonoAlphabetCipher.java

```java
package encryptionMethods.monoAlphabet;

import encryptionMethods.base.SubstitutionCipher;

/**
 * Класс для кодирования текста моноалфавитным методом Created by Алексей on
 * 25.03.2016.
 */
public class MonoAlphabetCipher extends SubstitutionCipher<Integer> {

    public MonoAlphabetCipher() {
        super();
    }

    /**
     * Создать закрытый алфавит
     *
     * @param key ключ смещения
     */
    @Override
    public void calculationPrivateAlphabet(Integer key) {
        for (int i = 0; i < 32; i++) {
            privateAlphabet[i] = openAlphabet[Math.floorMod(i + key, 32)];
        }
    }
}
```

Листинг 10 – код модуля BitReversCipher.java

```
package encryptionMethods.bitrevers;

import encryptionMethods.base.SubstitutionCipher;

/**
 * Класс для кодирования текста методом побитовой перестановки Created by
 * Алексей on 25.03.2016.
 */
public class BitReversCipher extends SubstitutionCipher<String> {
    public BitReversCipher() {
        super();
    }

    /**
     * Вычислить закрытый алфавит по заданому ключу
     *
     * @param key ключ
     */
    @Override
    public void calculationPrivateAlphabet (String key) {
        int[] arrayKey = findPosition(key);
        for (int i = 0; i < 32; i++) {
            StringBuilder stI = new StringBuilder(Integer.toBinaryString(i));
            for (int j = stI.length(); j < 5; j++) {
                stI.insert(0, "0");
            }
            StringBuilder charAlph = new StringBuilder();
            char[] mass = stI.toString().toCharArray();
            for (int anArrayKey : arrayKey) {
                charAlph.append(mass[anArrayKey]);
            }
            int exitI = Integer.parseInt(charAlph.toString(), 2);
            privateAlphabet[i] = openAlphabet[exitI];
        }
    }

    /**
     * Найти в каких позициях произошла перестановка
     *
     * @param arrayKey
     * @return
     */
    private int[] findPosition(String arrayKey) {
        int[] result = new int[arrayKey.length()];
        result[0] = arrayKey.indexOf("1");
        result[1] = arrayKey.indexOf("2");
        result[2] = arrayKey.indexOf("3");
        result[3] = arrayKey.indexOf("4");
        result[4] = arrayKey.indexOf("5");
        return result;
    }
}
```

Листинг 11 – код модуля MonoAlphabetCipherTest.java

```
import encryptionMethods.base.SubstitutionCipher;
import encryptionMethods.monoAlphabet.MonoAlphabetCipher;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertArrayEquals;
import static org.junit.Assert.assertEquals;

/**
 * Created by Алексей on 19.12.2016.
 */
public class MonoAlphabetCipherTest {
    SubstitutionCipher monoAlphabetCipher;

    @Before
    public void beforeTest() {
        monoAlphabetCipher = new MonoAlphabetCipher();
    }

    @After
    public void afterTest() {
        monoAlphabetCipher = null;
    }

    @Test(expected = NullPointerException.class)
    public void testNullCalculationPrivateAlphabet() {
        monoAlphabetCipher.calculationPrivateAlphabet(null);
    }

    @Test(expected = ClassCastException.class)
    public void testWrongArgumentCalculationPrivateAlphabet() {
        monoAlphabetCipher.calculationPrivateAlphabet("test");
    }

    @Test
    public void testCalculationPrivateAlphabet() {
        char[] actuals = {' ', 'а', 'б', 'в', 'г', 'д', 'е', 'ж', 'з', 'и', 'к', 'л', 'м
        monoAlphabetCipher.calculationPrivateAlphabet(0);
        assertArrayEquals(monoAlphabetCipher.getPrivateAlphabet(), actuals);
        assertArrayEquals(monoAlphabetCipher.getOpenAlphabet(), actuals);
        assertArrayEquals(monoAlphabetCipher.getOpenAlphabet(), monoAlphabetCipher.getOp
    }

    @Test
    public void testCalculationPrivateAlphabetCaesarCipher() {
        char[] actuals = {'в', 'г', 'д', 'е', 'ж', 'з', 'и', 'к', 'л', 'м', 'н', 'о', 'п
        monoAlphabetCipher.calculationPrivateAlphabet(3);
        assertArrayEquals(monoAlphabetCipher.getPrivateAlphabet(), actuals);
    }

    @Test
    public void testEncodeText() {
```

```
55          monoAlphabetCipher.calculationPrivateAlphabet(3);
56          assertEquals(monoAlphabetCipher.encodeText("тест"), "хифх");
57      }
58
59      @Test
60      public void testEncodeAndDecode() {
61          monoAlphabetCipher.calculationPrivateAlphabet(30);
62          String encode = monoAlphabetCipher.encodeText("тест");
63          assertEquals("тест", monoAlphabetCipher.decodeText(encode));
64      }
65
66      @Test
67      public void testWrongCharInEncodeAndDecodeText() {
68          monoAlphabetCipher.calculationPrivateAlphabet(1);
69          Assert.assertEquals(monoAlphabetCipher.encodeText("tttt"), "Недопустимый символ.
70          Assert.assertEquals(monoAlphabetCipher.encodeText("й"), "Недопустимый символ. Ш
71          Assert.assertEquals(monoAlphabetCipher.encodeText("ё"), "Недопустимый символ. Ш
72          Assert.assertEquals(monoAlphabetCipher.encodeText("-"), "Недопустимый символ. Ш
73          Assert.assertEquals(monoAlphabetCipher.encodeText("+"), "Недопустимый символ. Ш
74          Assert.assertEquals(monoAlphabetCipher.decodeText("tttt"), "Недопустимый символ.
75          Assert.assertEquals(monoAlphabetCipher.decodeText("й"), "Недопустимый символ. Ра
76          Assert.assertEquals(monoAlphabetCipher.decodeText("ё"), "Недопустимый символ. Ра
77          Assert.assertEquals(monoAlphabetCipher.decodeText("-"), "Недопустимый символ. Ра
78          Assert.assertEquals(monoAlphabetCipher.decodeText("+"), "Недопустимый символ. Ра
79      }
80  }
```

Листинг 12 – код модуля BitReversCipherTest.java

```
import encryptionMethods.base.SubstitutionCipher;
import encryptionMethods.bitrevers.BitReversCipher;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertArrayEquals;
import static org.junit.Assert.assertEquals;

/**
 * Created by Алексей on 19.12.2016.
 */
public class BitReversCipherTest {
    SubstitutionCipher bitReversCipher;

    @Before
    public void beforeTest() {
        bitReversCipher = new BitReversCipher();
    }

    @After
    public void afterTest() {
        bitReversCipher = null;
    }

    @Test
    public void testCalculationPrivateAlphabetEqualsOpenAlphabet() {
        char[] actuals = {' ', 'а', 'б', 'в', 'г', 'д', 'е', 'ж', 'з', 'и', 'к', 'л', 'м
        bitReversCipher.calculationPrivateAlphabet("12345");
        assertArrayEquals(bitReversCipher.getPrivateAlphabet(), actuals);
        assertArrayEquals(bitReversCipher.getOpenAlphabet(), actuals);
        assertArrayEquals(bitReversCipher.getOpenAlphabet(), bitReversCipher.getOpenAlph
    }

    @Test
    public void testCalculationPrivateAlphabet() {
        char[] actuals = {' ', 'р', 'з', 'ш', 'г', 'ф', 'м', 'ь', 'б', 'т', 'к', 'ъ', 'е
        bitReversCipher.calculationPrivateAlphabet("54321");
        assertArrayEquals(bitReversCipher.getPrivateAlphabet(), actuals);
    }

    @Test
    public void testFirstElementPrivateAlphabetAtAnyKey() {
        bitReversCipher.calculationPrivateAlphabet("54321");
        assertEquals(bitReversCipher.getPrivateAlphabet()[0], ' ');
        bitReversCipher.calculationPrivateAlphabet("53241");
        assertEquals(bitReversCipher.getPrivateAlphabet()[0], ' ');
        bitReversCipher.calculationPrivateAlphabet("13425");
        assertEquals(bitReversCipher.getPrivateAlphabet()[0], ' ');
        bitReversCipher.calculationPrivateAlphabet("31245");
        assertEquals(bitReversCipher.getPrivateAlphabet()[0], ' ');
    }

```

```
55      @Test
56      public void testEncode() {
57          bitReversCipher.calculationPrivateAlphabet("54321");
58          assertEquals(bitReversCipher.encodeText("тест"), "имси");
59      }
60
61      @Test
62      public void testEncodeDecode() {
63          bitReversCipher.calculationPrivateAlphabet("54321");
64          String encode = bitReversCipher.encodeText("тест");
65          assertEquals(bitReversCipher.decodeText(encode), "тест");
66      }
67
68      @Test(expected = IndexOutOfBoundsException.class)
69      public void testWrongValueForCalculationPrivateAlphabet() {
70          bitReversCipher.calculationPrivateAlphabet("00000");
71      }
72
73      @Test(expected = ClassCastException.class)
74      public void testWrongArgumentForCalculationPrivateAlphabet() {
75          bitReversCipher.calculationPrivateAlphabet(1213234);
76      }
77
78      @Test
79      public void testWrongCharInEncodeAndDecodeText() {
80          bitReversCipher.calculationPrivateAlphabet("54321");
81          Assert.assertEquals(bitReversCipher.encodeText("tttt"), "Недопустимый символ. Ш
82          Assert.assertEquals(bitReversCipher.encodeText("й"), "Недопустимый символ. Шифро
83          Assert.assertEquals(bitReversCipher.encodeText("ё"), "Недопустимый символ. Шифро
84          Assert.assertEquals(bitReversCipher.encodeText("-"), "Недопустимый символ. Шифро
85          Assert.assertEquals(bitReversCipher.encodeText("+"), "Недопустимый символ. Шифро
86          Assert.assertEquals(bitReversCipher.decodeText("tttt"), "Недопустимый символ. Ра
87          Assert.assertEquals(bitReversCipher.decodeText("й"), "Недопустимый символ. Расши
88          Assert.assertEquals(bitReversCipher.decodeText("ё"), "Недопустимый символ. Расши
89          Assert.assertEquals(bitReversCipher.decodeText("-"), "Недопустимый символ. Расши
90          Assert.assertEquals(bitReversCipher.decodeText("+"), "Недопустимый символ. Расши
91      }
92  }
```