МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Направление 02.03.01 Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

«Автоматизированное тестирование»

Обучающийся: _____ Шклярова Ксения Алексеевна

Руководитель: _____ Курочкин Михаил Александрович

«_____»_____20___г.

Санкт-Петербург, 2025

Содержание

Введение **3**

1 Постановка задачи **4**

2 Описание средств автоматизации тестирования **5**

2.1 JUnit 5

2.2 Selenium WebDriver 6

2.3 Page Object 7

2.4 TestNG 7

3 Задание №1: JUnit **10**

3.1 Класс CalculatoTestBase 10

3.2 Тестирование метода cos(double a) 10

3.3 Тестирование метода sub(long a, long b) 12

3.4 Тестирование метода pow(double a, double b) 14

3.5 Тестирование метода mult(long a, long b) 16

3.6 Результаты задания №1 18

4 Задание №2: Selenium WebDriver **19**

4.1 Класс DriverSetup 20

4.2 Класс Test1 21

4.2.1 Класс Task1Test 21

4.3 Класс Test2 23

4.4 Результаты задания №2 25

Заключение **27**

Введение

Автоматизированное тестирование — это метод проверки программного обеспечения, в котором для многократного выполнения набора тестовых случаев применяются инструменты и фреймворки автоматизации. В отличие от ручного тестирования, которое полностью полагается на действия человека за компьютером, автоматизированные тесты разрабатываются однократно и могут быть запущены многократно с минимальным участием человека.

Этот подход позволяет существенно ускорить процесс проверки. Он не только повышает эффективность, но и минимизирует человеческий фактор, снижая риск пропуска дефектов.

Автоматизация способствует стандартизации процесса тестирования, гарантируя последовательное выполнение каждого сценария независимо от того, кто проводит тестирование. Это особенно важно для поддержания стабильного уровня качества и уменьшения субъективности в оценке результатов.

Ключевые преимущества автоматизации включают возможность имитации реальной нагрузки, что сложно осуществить в ручном тестировании, и высокую повторяемость тестов. Однако, автоматизированное тестирование имеет и ограничения. Оно не предоставляет прямой обратной связи о субъективных аспектах качества, таких как удобство использования и эстетичность дизайна, и ограничено в тестировании пользовательского взаимодействия.

Таким образом, автоматизированное тестирование не является полной заменой ручному тестированию, а представляет собой мощный инструмент для комбинированного подхода. Автоматизация оптимальна для рутинных, повторяющихся и ресурсоемких задач, в то время как ручное тестирование необходимо для сложных сценариев, исследований и оценки пользовательского опыта. Комплексное использование этих подходов обеспечивает высокое качество программного обеспечения, особенно в крупных и долгосрочных проектах, требующих тщательной и всесторонней проверки.

1 Постановка задачи

Требуется: провести автоматизированное тестирование программного кода на Java - предоставленного класса «Калькулятор», и web-интерфейса предоставленного сайта.

В рамках поставленной цели необходимо выполнить следующие задачи:

1. Разработать unit-тесты для методов класса «Калькулятор», используя фреймворк JUnit. Протестировать четыре метода класса.

2. Реализовать два теста для предоставленного сайта с использованием фреймворка TestNG и методов библиотеки Selenium WebDriver в соответствии с заданием.

2 Описание средств автоматизации тестирования

2.1 JUnit

JUnit — это мощный фреймворк для модульного тестирования Java-приложений. Он предоставляет разработчикам удобные инструменты для создания, организации и выполнения автоматизированных тестов, что способствует повышению качества кода и ускорению процесса разработки.

Основные особенности JUnit:

• Аннотации: JUnit предоставляет аннотации, такие как @Test, @Before, @After, @BeforeClass, @AfterClass, которые помогают в организации тестов.

• Простота в использовании: Легкий в освоении и использовании, что делает его популярным среди разработчиков.

• Поддержка параметризованных тестов: С помощью аннотаций @ParameterizedTest и @ValueSource можно передавать параметры в тесты.

• Совместимость с различными средами разработки: JUnit интегрируется с большинством IDE, таких как IntelliJ IDEA, Eclipse, NetBeans.

Основные аннотации

JUnit использует систему аннотаций для управления жизненным циклом тестов:

`@Test` Обозначает метод как тестовый случай

`@BeforeEach` Метод выполняется перед каждым тестом

`@AfterEach` Метод выполняется после каждого теста

`@BeforeAll` Статический метод, выполняемый один раз перед всеми тестами

`@AfterAll` Статический метод, выполняемый один раз после всех тестов

Методы проверок

JUnit предоставляет следующие основные методы для валидации результатов:

- `assertEquals(expected, actual)` — проверка равенства значений
- `assertTrue(condition)` — проверка истинности условия
- `assertFalse(condition)` — проверка ложности условия
- `assertNull(object)` — проверка на null
- `assertNotNull(object)` — проверка, что объект не null
- `assertThrows(exception, executable)` — проверка генерации исключения

2.2 Selenium WebDriver

Selenium — это бесплатная и открытая библиотека для автоматизированного тестирования веб-приложений. В рамках проекта Selenium разрабатывается серия программных продуктов с открытым исходным кодом, один из которых — Selenium WebDriver.

Selenium WebDriver — инструмент для автоматизации тестирования пользовательского интерфейса. Он позволяет:

- Имитировать действия пользователя в браузере
- Выполнять кросс-браузерное тестирование
- Интегрироваться с различными языками программирования
- Работать с современными веб-фреймворками

Для работы с WebDriver требуется три основных компонента:

• Браузер - реальный браузер конкретной версии, установленный на определённой ОС с индивидуальными настройками.

• Драйвер браузера — веб-сервер, который запускает браузер, отправляет ему команды и закрывает браузер по завершении.

• Тест — набор команд на языке программирования, использующий библиотеки Selenium WebDriver bindings.

При работе с Selenium WebDriver можно выделить несколько основных понятий:

• WebDriver — сущность для управления браузером (центральный элемент теста)

• WebElement — абстракция веб-элементов (поля ввода, кнопки, ссылки) с методами взаимодействия и получения их текущего статуса.

• By — абстракция над локатором веб-элемента. Этот класс инкапсулирует информацию о селекторе, а также сам локатор элемента, то есть всю информацию, необходимую для нахождения нужного элемента на странице.

Процесс взаиможействия с браузером через WebDriver API можно описать следующими шагами:

1. Создание экземпляра WebDriver.

2. Открытие тестируемого приложения по URL.

3. Проведение серии действий по нахождению элементов на странице и взаимодействию с ними.

4. Проведение проверки, которая и опрделеит в конечном счете результат выполнения теста.

5. Закрытие браузера.

Компоненты системы

Language Bindings Поддержка Java, C#, Python, Ruby, JavaScript

JSON Wire Protocol Протокол взаимодействия между клиентом и драйвером

Browser Drivers Специфичные драйверы для каждого браузера

Real Browsers Фактические браузеры (Chrome, Firefox, Edge и др.)

2.3 Page Object

Page Object — шаблон проектирования для автоматизации тестирования веб-приложений. Его цель - инкапсулировать взаимодействие с элементами страницы в отдельные классы, что обеспечивает:

• Разделение кода: Отделение кода тестов от кода, описывающего взаимодействие с веб-страницей.

• Повторное использование: Возможность повторного использования кода для различных тестов.

• Легкость в поддержке: При изменении дизайна страницы необходимо обновить только соответствующий Page Object класс.

Основные принципы Page Object:

1. Одна страница - один класс.

2. Инкапсуляция: Локаторы хранятся внутри класса, методы описывают действия пользователя.

3. Разделение тестов и реализации: Тесты работают только через методы Page Object.

2.4 TestNG

TestNG — это современный фреймворк для тестирования, созданный как альтернатива JUnit с более мощными функциями. Он поддерживает различные виды тестирования: модульное, функциональное, интеграционное.

Основные аннотации

`@Test` Основная аннотация для тестовых методов

`@BeforeSuite/@AfterSuite` Выполняются до/после всего набора тестов

`@BeforeTest/@AfterTest` Выполняются до/после тестового блока в XML

`@BeforeClass/@AfterClass` Выполняются до/после методов класса

`@BeforeMethod/@AfterMethod` Выполняются до/после каждого тестового метода

`@BeforeGroups/@AfterGroups` Выполняются до/после группы тестов

`@Parameters` Для параметризации тестов

`@DataProvider` Для подачи тестовых данных

Особенности TestNG

1. Группировка тестов

   Тесты можно объединять в группы с помощью аннотации @Test (groups = "groupName"), что позволяет запускать только определенные группы тестов.

2. Зависимости между тестами

   Аннотация @DependsOnMethods позволяет указывать зависимости между тестами, гарантируя выполнение тестов в определенном порядке.

3. Параллельное выполнение

   TestNG поддерживает параллельное выполнение тестов на уровне методов, классов и тестовых наборов, что ускоряет процесс тестирования.

4. Интеграция с другими инструментами

   TestNG легко интегрируется с Selenium, Maven, Jenkins и другими инструментами, что делает его удобным для CI/CD.

5. Гибкость конфигурации тестов

   Test.NG позволяет настраивать тесты с помощью аннотаций и XML-файлов, что дает больше контроля над выполнением тестовых сценариев.

6. Поддержка параметризованных тестов

   Аннотации @Parameters и @DataProvider позволяют передавать параметры в тесты, что упрощает тестирование с раз-личными входными данными.

Этапы написания тестов в TestNG:

1. Создание тестового класса: Написание тестовых методов с аннотацией @Test.

2. Настройка пред- и пост-условий: Использование аннотаций BeforeSuite, @AfterSuite, BeforeTest, @AfterTest, BeforeMethod, @AfterMethod и других для настройки и очистки.

3. Запуск тестов: Запуск тестов через IDE (например, Intelli] IDE.A или Eclipse) или с помощью XML-конфигурации для управления тестовыми наборами.

4. Анализ результатов: Просмотр отчетов TestNG, которые включают информацию о пройденных и неудачных тестах, времени выполнения и других метриках.

TestNG часто используется вместе с Selenium WebDriver и Page Object для автоматизации тестирования веб-приложений, обеспечивая структурированный и масштабируемый подход к написанию тестов.

3 Задание №1: JUnit

В задании №1 необходимо создать юнит-тесты для библиотеки calculator.jar, выбрав четыре метода. Необходимо использовать JUnit, включая аннотации @Before и @After для пред и постобработки, а также @Parameterized Test и @ValueSource для параметризации тестов.

Для выполнения задания мною были выбраны следующие методы: cos(double a), sub(long a, long b), pow(double a, double b), mult(long a, long b).

3.1 Класс CalculatoTestBase

Класс CalculatoTestBase представляет собой абстрактный базовый класс для тестирования функциональности калькулятора. Его основное назначение:

- Содержит общие настройки и поля для всех тестовых классов калькулятора
- Инициализирует экземпляр калькулятора перед выполнением всех тестов
- Определяет константу DELTA для сравнения чисел с плавающей точкой

```
1     public class CalculatorTestBase {
2     protected Calculator calculator;
3     protected static final double DELTA = 0.1;
4
5     @BeforeEach
6     void setUp() {
7         calculator = new Calculator();
8     }
9
10    @AfterEach
11    void tearDown() {
12        calculator = null;
13    }
14 }
```

3.2 Тестирование метода cos(double a)

Класс TestCos выполняет unit-тесты для метода cos(double a) в классе Calculator. Он проверяет корректность вычисления косинуса для для различных входных значений, свойства функции косинуса (четность и периодичность), правильность вычислений для граничных значений.

```
1 class TestCos extends CalculatorTestBase {
2     private static final double TOLERANCE = 0.01; // Допустимая погрешность
3
4     // Тест корректности вычислений косинуса
5     @ParameterizedTest
6     @CsvSource({
7             "1.0, 0",
8             "0.87, 0.5",
9             "0.71, 0.785",
```

```
10              "0.5, 1.047",
11              "0.0, 1.571",
12              "-1.0, 3.142",
13              "0.0, 4.712"
14      })
15      void testCorrectnessOfCosinusCalculation(double expectedOutcome, double angleInRadians) {
16          double result = calculator.cos(angleInRadians);
17          assertEquals(expectedOutcome, result, TOLERANCE,
18                  "Вычисленное значение косинуса не соответствует ожидаемому.");
19      }
20
21      // Тест четности функции косинуса (cos(-x) = cos(x))
22      @ParameterizedTest
23      @CsvSource({
24              "0.5, -0.5",
25              "1.047, -1.047",
26              "3.142, -3.142"
27      })
28      void testCosinusFunctionEvenness(double positiveAngle, double negativeAngle) {
29          double cosPositive = calculator.cos(positiveAngle);
30          double cosNegative = calculator.cos(negativeAngle);
31          assertEquals(cosPositive, cosNegative, TOLERANCE,
32                  "Косинус отрицательного угла не равен косинусу соответствующего положительного
                        угла.");
33      }
34
35      // Тест периодичности функции косинуса (cos(x + 2pi) = cos(x))
36      @ParameterizedTest
37      @CsvSource({
38              "0",        // 0
39              "1.047",    // pi/3
40              "2.094",    // 2pi/3
41              "3.142",    // pi
42              "4.189",    // 4pi/3
43              "5.236"     // 5pi/3
44      })
45      void testCosinusFunctionPeriodicity(double angle) {
46          double expected = calculator.cos(angle);
47          double actual = calculator.cos(angle + 2 * Math.PI);
48          assertEquals(expected, actual, TOLERANCE,
49                  "Косинус угла не остается неизменным после добавления 2  .");
50      }
51
52      // Дополнительный тест для граничных значений
53      @Test
54      void testCosinusSpecialValues() {
55          assertAll(
56                  () -> assertEquals(1.0, calculator.cos(0.0), TOLERANCE),
57                  () -> assertEquals(0.0, calculator.cos(Math.PI/2), TOLERANCE),
58                  () -> assertEquals(-1.0, calculator.cos(Math.PI), TOLERANCE),
59                  () -> assertEquals(0.0, calculator.cos(3*Math.PI/2), TOLERANCE),
```

```
60                 () -> assertEquals(1.0, calculator.cos(2*Math.PI), TOLERANCE)
61         );
62     }
63 }
```

Результат: были пройдены только тесты на проверку периодичности.

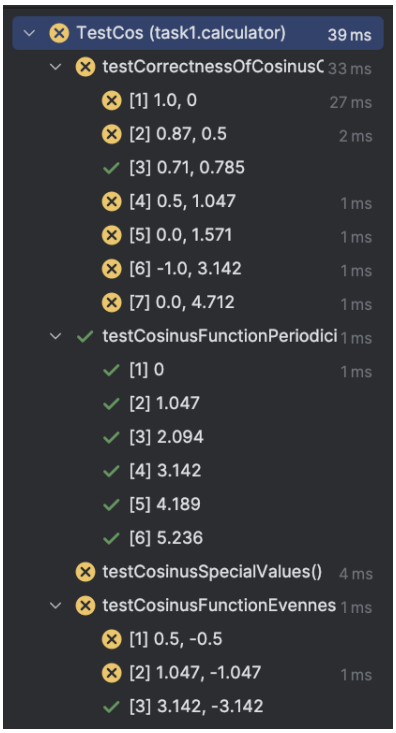

Рис. 1. Итоги тестов класса TestCos

3.3 Тестирование метода sub(long a, long b)

Класс SubLongTest выполняет unit-тесты для метода sub(long a, long b) в классе Calculator. Он проверяет корректность вычитания в стандартных случаях, поведение при переполнении и исчерпании диапазона значений, математические свойства операции вычитания, работу с граничными значениями типа long.

```
1  class SubLongTest extends CalculatorTestBase {
2
3      // Стандартные случаи вычитания
4      @ParameterizedTest
5      @CsvSource({
6              "10, 3, 7",                          // Положительные числа
7              "0, 0, 0",                           // Нули
8              "-5, 3, -8",                         // Отрицательное уменьшаемое
9              "5, -3, 8",                          // Отрицательное вычитаемое
10             "9223372036854775807, 1, 9223372036854775806",   // MAX_VALUE - 1
11             "-9223372036854775808, -1, -9223372036854775807" // MIN_VALUE - (-1)
12     })
13     void testSubLongNormalCases(long a, long b, long expected) {
14         assertEquals(expected, calculator.sub(a, b));
```

```
15      }
16
17      // Тесты на переполнение
18      @Test
19      void testSubLongOverflow() {
20          assertAll(
21                  () -> assertEquals(Long.MAX_VALUE, calculator.sub(Long.MIN_VALUE, 1)),
22                  () -> assertEquals(Long.MIN_VALUE + 1, calculator.sub(Long.MIN_VALUE, -1))
23          );
24      }
25
26      @Test
27      void testSubLongUnderflow() {
28          assertAll(
29                  () -> assertEquals(Long.MIN_VALUE, calculator.sub(Long.MAX_VALUE, -1)),
30                  () -> assertEquals(Long.MAX_VALUE - 1, calculator.sub(Long.MAX_VALUE, 1))
31          );
32      }
33
34      // Тесты коммутативности (a - b = -(b - a))
35      @ParameterizedTest
36      @CsvSource({
37              "5, 3",
38              "-10, 7",
39              "0, 0",
40              "9223372036854775807, 9223372036854775806"
41      })
42      void testSubtractionCommutativity(long a, long b) {
43          long result1 = calculator.sub(a, b);
44          long result2 = calculator.sub(b, a);
45          assertEquals(result1, -result2);
46      }
47
48      // Тесты вычитания самого себя (a - a = 0)
49      @ParameterizedTest
50      @ValueSource(longs = {0L, 1L, -1L, Long.MAX_VALUE, Long.MIN_VALUE})
51      void testSubtractionOfItself(long value) {
52          assertEquals(0L, calculator.sub(value, value));
53      }
54
55      // Тесты с большими числами
56      @ParameterizedTest
57      @CsvSource({
58              "9223372036854775807, 9223372036854775806, 1",  // MAX_VALUE - (MAX_VALUE-1)
59              "-9223372036854775808, -9223372036854775807, -1" // MIN_VALUE - (MIN_VALUE+1)
60      })
61      void testSubLongWithLargeNumbers(long a, long b, long expected) {
62          assertEquals(expected, calculator.sub(a, b));
63      }
64
65      // Тест вычитания с нулем
```

```
66      @ParameterizedTest
67      @ValueSource(longs = {0L, 1L, -1L, Long.MAX_VALUE, Long.MIN_VALUE})
68      void testSubtractionWithZero(long value) {
69          assertEquals(value, calculator.sub(value, 0));
70          assertEquals(-value, calculator.sub(0, value));
71      }
72  }
```

Результат: все тесты были пройдены.

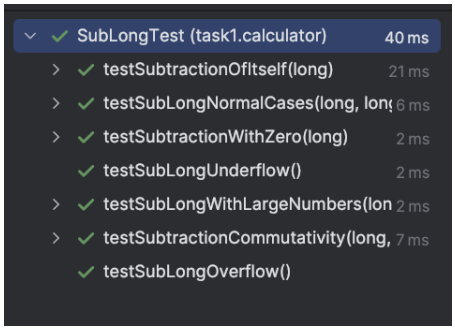

Рис. 2. Итоги тестов класса SubLongTest

3.4 Тестирование метода pow(double a, double b)

Класс PowTest выполняет unit-тесты для метода pow(double a, double b) в классе Calculator. Он проверяет корректность вычисления степени в стандартных случаях, поведение функции при граничных значениях, работу с бесконечностями и NaN (Not a Number), обработку отрицательных оснований с дробными степенями.

```
1   class PowTest extends CalculatorTestBase {
2       private static final double PRECISION = 0.000001;
3
4       // Стандартные случаи
5       @ParameterizedTest
6       @CsvSource({
7               "2.0, 3.0, 8.0",        // Положительные числа
8               "10.0, 0.0, 1.0",       // Любое число в степени 0
9               "2.0, -1.0, 0.5",       // Отрицательная степень
10              "1.0, 1000.0, 1.0",     // 1 в любой степени
11              "4.0, 0.5, 2.0",        // Дробная степень (квадратный корень)
12              "2.5, 2.0, 6.25"        // Дробное основание
13      })
14      void testPowNormalCases(double a, double b, double expected) {
15          assertEquals(expected, calculator.pow(a, b), PRECISION);
16      }
17
18      // Граничные значения и особые случаи
19      @ParameterizedTest
20      @CsvSource({
21              "0.0, 0.0, NaN",        // 0^0 — неопределенность
```

```
22              "0.0, 1.0, 0.0",        // 0 в положительной степени
23              "0.0, -1.0, Infinity", // 0 в отрицательной степени
24              "-2.0, 2.0, 4.0",       // Отрицательное основание, четная степень
25              "-2.0, 3.0, -8.0",      // Отрицательное основание, нечетная степень
26              "1.0, Infinity, NaN",  // 1 в степени Infinity
27              "1.0, -Infinity, NaN"  // 1 в степени -Infinity
28      })
29
30      void testPowEdgeCases(double a, double b, double expected) {
31          double actual = calculator.pow(a, b);
32          //System.out.println(actual);
33          if (Double.isNaN(expected)) {
34              assertTrue(Double.isNaN(actual));
35          } else {
36              assertEquals(expected, actual, PRECISION);
37          }
38      }
39
40      // Тесты с бесконечностью
41      @ParameterizedTest
42      @CsvSource({
43              "2.0, Infinity, Infinity",
44              "0.5, Infinity, 0.0",
45              "2.0, -Infinity, 0.0",
46              "0.5, -Infinity, Infinity"
47      })
48      void testPowWithInfinity(double a, double b, double expected) {
49          assertEquals(expected, calculator.pow(a, b), PRECISION);
50      }
51
52      // Тесты с NaN
53      @Test
54      void testPowWithNaN() {
55          assertAll(
56                  () -> assertTrue(Double.isNaN(calculator.pow(Double.NaN, 2.0))),
57                  () -> assertTrue(Double.isNaN(calculator.pow(2.0, Double.NaN))),
58                  () -> assertTrue(Double.isNaN(calculator.pow(Double.NaN, Double.NaN)))
59          );
60      }
61
62      // Тесты для отрицательного основания с дробной степенью
63      @Test
64      void testPowNegativeBaseWithFractionalExponent() {
65          assertAll(
66                  () -> assertTrue(Double.isNaN(calculator.pow(-4.0, 0.5))),  // sqrt(-4)
67                  () -> assertTrue(Double.isNaN(calculator.pow(-1.0, 1.5)))   // (-1)^(3/2)
68          );
69      }
70  }
```

Результат: были пройдены тесты на проверку стандартных случаев возведения в степень, а

также тесты с NaN. Остальные тесты не были пройдены.

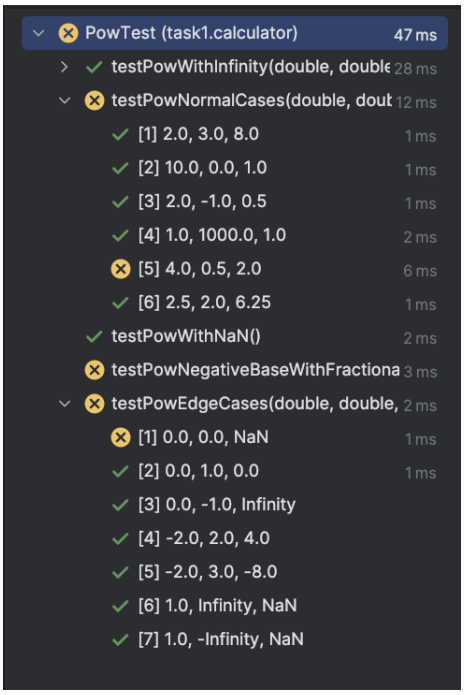

Рис. 3. Итоги тестов класса PowTest

3.5 Тестирование метода mult(long a, long b)

Класс MultLongTests выполняет unit-тесты для метода mult(long a, long b) в классе Calculator. Он проверяет проверяет общую корректность вычислений умножения, поведение при граничных значениях типа long, обработку переполнения при умножении, математические свойства операции умножения, особые случаи умножения.

```
1  class MultLongTests extends CalculatorTestBase {
2
3      // Тест общей корректности вычислений
4      @ParameterizedTest
5      @CsvSource({
6              "6, 2, 3",
7              "-15, 3, -5",
8              "0, 0, 5",
9              "0, 10, 0",
10             "9223372036854775807, 9223372036854775807, 1" // MAX_VALUE * 1
11     })
12     void testGeneralCorrectness(long expected, long a, long b) {
13         assertEquals(expected, calculator.mult(a, b));
14     }
15
16     // Тест граничных значений
17     @ParameterizedTest
18     @CsvSource({
19             "9223372036854775807, 9223372036854775807, 1", // MAX_VALUE
20             "-9223372036854775808, -9223372036854775808, 1" // MIN_VALUE
```

```
21      })
22      void testBoundaryValues(long expected, long a, long b) {
23          assertEquals(expected, calculator.mult(a, b));
24      }
25
26      // Тест переполнения (ожидаемое поведение)
27      @Test
28      void testOverflow() {
29          // MAX_VALUE * 2 = -2 (в результате переполнения)
30          assertEquals(-2L, calculator.mult(Long.MAX_VALUE, 2L));
31      }
32
33      // Тест умножения на 1 и -1
34      @ParameterizedTest
35      @CsvSource({
36              "5, 5, 1",
37              "-5, 5, -1",
38              "0, 0, 1"
39      })
40      void testMultiplicationByIdentity(long expected, long a, long b) {
41          assertEquals(expected, calculator.mult(a, b));
42      }
43
44      // Тест коммутативности (a*b = b*a)
45      @ParameterizedTest
46      @CsvSource({
47              "2, 3",
48              "-5, 4",
49              "0, 10"
50      })
51      void testCommutativeProperty(long a, long b) {
52          assertEquals(calculator.mult(a, b), calculator.mult(b, a));
53      }
54
55      // Тест особого случая MIN_VALUE * -1
56      @Test
57      void testMinValueMultiplication() {
58          // MIN_VALUE * -1 = MIN_VALUE (из-за переполнения)
59          assertEquals(Long.MIN_VALUE, calculator.mult(Long.MIN_VALUE, -1L));
60      }
61  }
```

Результат: все тесты были пройдены.

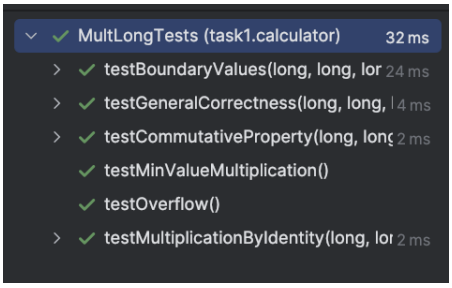

Рис. 4. Итоги тестов класса MultiLongTests

3.6 Результаты задания №1

1. Тестирование функции косинуса (TestCos)

Тестирование данной функции показало, что основная формула косинуса дает некорректные результаты. Также функция не сохраняет свойство cos(-x) = cos(x). Были получены неверные результаты для углов 0, /2, и т.д. Но прохождение теста testCosinusFunctionPeriodicity при неудаче остальных тестов в классе TestCos указывает на частичную корректность реализации функции косинуса.

2. Тестирование функции разности (SubLongTest)

Прохождение всех тестов в классе SubLongTest свидетельствует о корректной реализации операции вычитания для типа long в Calculator. Реализация соблюдает свойства вычитания (антикоммутативность, работа с нулем, тождественность). Калькулятор правильно обрабатывает Long.MAX_VALUE и Long.MIN_VALUE, что критично для целочисленных операций.

3. Тестирование фуннкции возведения в степень (PowTest)

Прохождение одних тестов и непрохождение других в классе PowTest указывает на частично корректную, но неполную реализацию функции возведения в степень (pow). Калькулятор правильно обрабатывает NaN и бесконечности, что критично для надежных математических вычислений. Но при этом были получены неверные результаты для простых случаев (дробные/отрицательные степени). Не обрабатываются особые случаи ($0^0, 0^{-1}$). Нет корректной проверки для дробных степеней.

4. Тестирование функции умножения (MultiLongTests)

Прохождение всех тестов в классе MultLongTests свидетельствует о корректной и надежной реализации операции умножения для типа long. Калькулятор корректно обрабатывает крайние случаи (например, MAX_VALUE * 2), следуя стандартному поведению Java. Соблюдение свойств умножения: коммутативность, умножение на 1 и -1, работа с нулем. Кроме того тесты покрывают все критичные сценарии, включая MIN_VALUE * -1, что подтверждает устойчивость реализации.

4 Задание №2: Selenium WebDriver

Необходимо разработать автоматизированные тесты для проверки корректной работы библиотеки Selenium WebDriver на примере веб-сайта JDI Testing (https://jdi-testing.github.io/jdi-light/index.html), используя браузер Chrome.

В качестве фреймворка для тестирования следует использовать TestNG. Это позволит структурировать тестовый код, организовать последовательный запуск тестов и формировать подробные отчеты о результатах. Selenium WebDriver будет использоваться для автоматизации взаимодействия с веб-сайтом через браузер.

Требуется реализовать два независимых теста, каждый из которых должен проверять корректное отображение отдельной страницы на сайте JDI Testing. Цель тестов — убедиться, что все веб-элементы и функциональные компоненты каждой страницы загружаются и отображаются правильно в браузере Chrome.

#	**Шаг**	**Данные**	**Ожидаемый результат**
1	Open test site by URL	https://jdi-testing.github.io/jdi-light/index.html	Test site is opened
2	Assert Browser title	"Home Page"	Browser title equals "Home Page"
3	Perform login	username: Roman pass: Jdi1234	User is logged
4	Assert Username is loggined	"ROMAN IOVLEV"	Name is displayed and equals to expected result
5	Assert that there are 4 items on the header section are displayed and they have proper texts	"HOME", "CONTACT FORM", "SERVICE", "METALS & COLORS"	Menu buttons are displayed and have proper texts
6	Assert that there are 4 images on the Index Page and they are displayed	4 images	Images are displayed
7	Assert that there are 4 texts on the Index Page under icons and they have proper text	4 texts below of each image	Texts are displayed and equal to expected
8	Assert that there is the iframe with “Frame Button” exist		The iframe exists
9	Switch to the iframe and check that there is “Frame Button” in the iframe		The “Frame Button” exists
10	Switch to original window back		Driver has focus on the original window
11	Assert that there are 5 items in the Left Section are displayed and they have proper text	“Home”, “Contact form”, “Service”, “Metals & Colors”, “Elements packs”	Left section menu items are displayed and have proper text
12	Close Browser		Browser is closed

Рис. 5. Задание 1

#	Шаг	Данные	Ожидаемый результат
1	Open test site by URL	https://jdi-testing.github.io/jdi-light/index.html	Test site is opened
2	Assert Browser title	"Home Page"	Browser title equals "Home Page"
3	Perform login	username: Roman pass: Jdi1234	User is logged
4	Assert User name in the left-top side of screen that user is loggined	ROMAN IOVLEV	Name is displayed and equals to expected result
5	Open through the header menu Service -> Different Elements Page		Page is opened
6	Select checkboxes	Water, Wind	Elements are checked
7	Select radio	Selen	Element is checked
8	Select in dropdown	Yellow	Element is selected
9	Assert that • for each checkbox there is an individual log row and value is corresponded to the status of checkbox • for radio button there is a log row and value is corresponded to the status of radio button • for dropdown there is a log row and value is corresponded to the selected value.		Log rows are displayed and • checkbox name and its status are corresponding to selected • radio button name and it status is corresponding to selected • dropdown name and selected value is corresponding to selected
10	Close Browser		Browser is closed

Рис. 6. Задание 2

4.1 Класс DriverSetup

Класс DriverSetup в Java предназначен для настройки тестовой среды с использованием Selenium WebDriver и TestNG. Он выполняет следующие функции: инициализирует драйвер Chrome с максимальным размером окна, открывает тестовый сайт и авторизуется на нем, а также закрывает браузер после завершения выполнения всех тестов.

Аннотации TestNG @BeforeSuite и @AfterSuite управляют выполнением этих действий в начале и конце тестового набора, обеспечивая корректную подготовку и завершение тестовой среды.

Листинг 1. DriverSetup

```
1  package edu.hsai.task2;
2
3  import org.openqa.selenium.By;
4  import org.openqa.selenium.WebDriver;
5  import org.openqa.selenium.chrome.ChromeDriver;
6  import org.testng.annotations.AfterSuite;
7  import org.testng.annotations.BeforeSuite;
8
9  /**
10  * Базовый класс для настройки и завершения работы тестов
11  */
12 public class DriverSetup {
13     protected static WebDriver driver;
14
15     @BeforeSuite
16     public void setup() {
17         // Настройка драйвера Chrome
18         System.setProperty("webdriver.chrome.driver", "src/test/resources/chromedriver.exe");
19         System.setProperty("webdriver.http.factory", "jdk-http-client");
```

```
20
21          driver = new ChromeDriver();
22          driver.manage().window().maximize(); // Развернуть окно на весь экран
23
24          // 1. Открыть тестовый сайт по URL
25          driver.navigate().to("https://jdi-testing.github.io/jdi-light/index.html");
26
27          // 3. Выполнить вход в систему
28          driver.findElement(By.cssSelector(".uui-profile-menu .dropdown-toggle")).click();
29          driver.findElement(By.id("name")).sendKeys("Roman");
30          driver.findElement(By.id("password")).sendKeys("Jdi1234");
31          driver.findElement(By.id("login-button")).click();
32      }
33
34      @AfterSuite
35      public void tearDown() {
36          // 10. Закрыть браузер (для всех тестов)
37          if (driver != null) {
38              driver.quit();
39          }
40      }
41  }
```

4.2 Класс Test1

4.2.1 Класс Task1Test

Класс Test1 предназначен для комплексной проверки главной страницы тестового сайта. После открытия сайта и успешной авторизации, класс выполняет верификацию различных элементов интерфейса, используя мягкие утверждения (SoftAsserts) для накопления результатов. В частности, проверяется заголовок страницы, наличие и корректность текста пунктов меню, видимость изображений и текста под иконками, а также наличие iframe и элементов левого меню.

Листинг 2. Task1

```
1  package edu.hsai.task2;
2
3  import org.openqa.selenium.By;
4  import org.openqa.selenium.WebElement;
5  import org.testng.annotations.Test;
6  import org.testng.asserts.SoftAssert;
7  import java.util.List;
8
9  /**
10  * Тестовый класс для Задания 1 (с использованием SoftAssert)
11  */
12 public class Task1 extends DriverSetup {
13     @Test
14     public void testHomePageElements() {
```

```
15          SoftAssert softAssert = new SoftAssert();
16
17          // 2. Проверить заголовок браузера
18          softAssert.assertEquals(driver.getTitle(), "Home Page", "Заголовок браузера должен быт
                ь 'Home Page'");
19
20          // 4. Проверить, что имя пользователя отображается
21          WebElement username = driver.findElement(By.id("user-name"));
22          softAssert.assertTrue(username.isDisplayed(), "Имя пользователя должно отображаться");
23          softAssert.assertEquals(username.getText(), "ROMAN IOVLEV", "Имя пользователя должно б
                ыть 'ROMAN IOVLEV'");
24
25          // 5. Проверить пункты меню в шапке сайта
26          List<WebElement> headerItems = driver.findElements(By.cssSelector("ul.uui-navigation.
                nav > li > a"));
27          String[] expectedHeaderTexts = {"HOME", "CONTACT FORM", "SERVICE", "METALS & COLORS"};
28
29          softAssert.assertEquals(headerItems.size(), 4, "Должно быть 4 пункта меню в шапке");
30          for (int i = 0; i < headerItems.size(); i++) {
31              softAssert.assertTrue(headerItems.get(i).isDisplayed(),
32                      "Пункт меню " + i + " должен отображаться");
33              softAssert.assertEquals(headerItems.get(i).getText(), expectedHeaderTexts[i],
34                      "Текст пункта меню " + i + " должен соответствовать ожидаемому");
35          }
36
37          // 6. Проверить иконки преимуществ
38          List<WebElement> images = driver.findElements(By.className("benefit-icon"));
39          softAssert.assertEquals(images.size(), 4, "Должно быть 4 иконки преимуществ");
40          images.forEach(image -> softAssert.assertTrue(image.isDisplayed(), "Иконка преимуществ
                a должна отображаться"));
41
42          // 7. Проверить тексты под иконками
43          List<WebElement> texts = driver.findElements(By.className("benefit-txt"));
44          String[] expectedTexts = {
45                  "To include good practices\nand ideas from successful\nEPAM project",
46                  "To be flexible and\ncustomizable",
47                  "To be multiplatform",
48                  "Already have good base\n(about 20 internal and\nsome external projects),\
                        nwish to get more "
49          };
50
51          softAssert.assertEquals(texts.size(), 4, "Должно быть 4 текста преимуществ");
52          for (int i = 0; i < texts.size(); i++) {
53              softAssert.assertTrue(texts.get(i).isDisplayed(), "Текст преимущества " + i + " до
                    лжен отображаться");
54              softAssert.assertEquals(texts.get(i).getText(), expectedTexts[i],
55                      "Текст преимущества " + i + " должен соответствовать ожидаемому");
56          }
57
58          // 8. Проверить наличие iframe
59          WebElement iframe = driver.findElement(By.id("frame"));
```

```
60          softAssert.assertTrue(iframe.isDisplayed(), "Фрейм должен отображаться");
61
62          // 9. Проверить кнопку во фрейме
63          driver.switchTo().frame(iframe);
64          WebElement frameButton = driver.findElement(By.id("frame-button"));
65          softAssert.assertTrue(frameButton.isDisplayed(), "Кнопка во фрейме должна отображаться
                ");
66          driver.switchTo().defaultContent();
67
68          // 11. Проверить пункты меню в левой секции
69          List<WebElement> leftMenuItems = driver.findElements(By.cssSelector("ul.sidebar-menu.
                left > li"));
70          String[] expectedLeftMenuTexts = {"Home", "Contact form", "Service", "Metals & Colors"
                , "Elements packs"};
71
72          softAssert.assertEquals(leftMenuItems.size(), 5, "Должно быть 5 пунктов меню слева");
73          for (int i = 0; i < leftMenuItems.size(); i++) {
74              softAssert.assertTrue(leftMenuItems.get(i).isDisplayed(),
75                      "Пункт меню " + i + " должен отображаться");
76              softAssert.assertEquals(leftMenuItems.get(i).getText(), expectedLeftMenuTexts[i],
77                      "Текст пункта меню " + i + " должен соответствовать ожидаемому");
78          }
79
80          softAssert.assertAll();
81      }
82  }
```

4.3 Класс Test2

Класс Test2 представляет собой структурированный набор тестов для проверки страницы "Different Elements"сайта. Тесты организованы в отдельные логические методы, что позволяет проверить каждый аспект функциональности независимо. Проверки включают в себя заголовок страницы и авторизацию, а также комплексный тест, имитирующий взаимодействие пользователя с элементами формы: выбор чекбоксов, радиокнопки и значения из выпадающего списка. Корректность работы также подтверждается анализом текста в логах операций.

Листинг 3. Task2

```
1  package edu.hsai.task2;
2
3  import org.openqa.selenium.By;
4  import org.openqa.selenium.WebElement;
5  import org.openqa.selenium.support.ui.Select;
6  import org.testng.annotations.Test;
7  import org.testng.asserts.SoftAssert;
8  import java.util.List;
9
10 /**
11  * Тестовый класс для Задания 2 (страница Different Elements)
```

```
12  */
13 public class Task2 extends DriverSetup {
14     @Test
15     public void testBrowserTitle() {
16         // 2. Проверить заголовок браузера
17         SoftAssert softAssert = new SoftAssert();
18         softAssert.assertEquals(driver.getTitle(), "Home Page", "Заголовок браузера должен быт
               ь 'Home Page'");
19         softAssert.assertAll();
20     }
21
22     @Test
23     public void testLogin() {
24         // 4. Проверить имя пользователя
25         SoftAssert softAssert = new SoftAssert();
26         WebElement username = driver.findElement(By.id("user-name"));
27         softAssert.assertTrue(username.isDisplayed(), "Имя пользователя должно отображаться");
28         softAssert.assertEquals(username.getText(), "ROMAN IOVLEV", "Имя пользователя должно б
               ыть 'ROMAN IOVLEV'");
29         softAssert.assertAll();
30     }
31
32     @Test
33     public void testDifferentElementsPage() {
34         SoftAssert softAssert = new SoftAssert();
35
36         // 5. Открыть страницу Different Elements через меню
37         driver.findElement(By.cssSelector(".dropdown-toggle")).click();
38         driver.findElement(By.xpath("//a[text()='Different elements']")).click();
39         softAssert.assertEquals(driver.getTitle(), "Different Elements",
40                 "Заголовок страницы должен быть 'Different Elements'");
41
42         // 6. Выбрать чекбоксы
43         WebElement waterCheckbox = driver.findElement(By.xpath("//label[contains(.,'Water')]/
               input"));
44         WebElement windCheckbox = driver.findElement(By.xpath("//label[contains(.,'Wind')]/
               input"));
45
46         waterCheckbox.click();
47         windCheckbox.click();
48
49         softAssert.assertTrue(waterCheckbox.isSelected(), "Чекбокс Water должен быть выбран");
50         softAssert.assertTrue(windCheckbox.isSelected(), "Чекбокс Wind должен быть выбран");
51
52         // 7. Выбрать радио-кнопку
53         WebElement selenRadio = driver.findElement(By.xpath("//label[contains(.,'Selen')]/
               input"));
54         selenRadio.click();
55         softAssert.assertTrue(selenRadio.isSelected(), "Радио-кнопка Selen должна быть выбрана
               ");
56
```

```
57          // 8. Выбрать значение в выпадающем списке
58          Select dropdown = new Select(driver.findElement(By.cssSelector("select.uui-form-
                element")));
59          dropdown.selectByVisibleText("Yellow");
60          softAssert.assertEquals(dropdown.getFirstSelectedOption().getText(), "Yellow",
61                  "В выпадающем списке должно быть выбрано 'Yellow'");
62
63          // 9. Проверить логи
64          List<WebElement> logs = driver.findElements(By.cssSelector(".logs li"));
65
66          softAssert.assertTrue(logs.get(0).getText().contains("Yellow") &&
67                          logs.get(0).getText().contains("Colors"),
68                  "Лог для изменения цвета должен быть корректным");
69
70          softAssert.assertTrue(logs.get(1).getText().contains("Selen") &&
71                          logs.get(1).getText().contains("metal"),
72                  "Лог для выбора металла должен быть корректным");
73
74          softAssert.assertTrue(logs.get(2).getText().contains("Wind") &&
75                          logs.get(2).getText().contains("true"),
76                  "Лог для чекбокса Wind должен быть корректным");
77
78          softAssert.assertTrue(logs.get(3).getText().contains("Water") &&
79                          logs.get(3).getText().contains("true"),
80                  "Лог для чекбокса Water должен быть корректным");
81
82          softAssert.assertAll();
83      }
84  }
```

4.4 Результаты задания №2

На рис. 7 представлены результаты тестов Test1 и Test2.

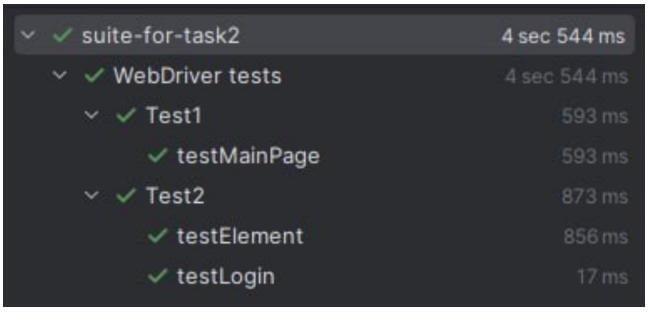

Рис. 7. Результаты тестов Test1 и Test2

Результатом задания №2 стало успешное прохождение всех тестов Selenium WebDriver и TestNG, что свидетельствует о корректной работе веб-приложения JDI. Тесты продемонстрировали ожидаемое поведение во всех проверяемых сценариях, включая корректное отображение элементов

интерфейса, функциональность элементов формы (чекбоксы, радиокнопки, выпадающие списки), систему логирования, авторизацию пользователя и навигацию по меню. Этот успех стал возможен благодаря тщательному подходу к тестированию, включающему: корректную настройку WebDriver для стабильного взаимодействия с Chrome, использование точных CSS и XPath селекторов для эффективного поиска элементов, применение мягких утверждений (SoftAsserts) для детального анализа результатов без прерывания тестов, широкий охват элементов страницы и управление сессией браузера для обеспечения чистоты тестовой среды.

Заключение

В ходе лабораторной работы были успешно выполнены все поставленные задачи по автоматизированному тестированию программного обеспечения.

Во-первых, для класса Calculator разработаны unit-тесты с использованием JUnit 5. Были протестированы методы cos(double a), sub(long a, long b), pow(double a, double b) и mult(long a, long b). В результате тестирования было обнаружено, что методы cos(double a) и pow(double a, double b) работают некорректно, в то время как методы sub(long a, long b) и mult(long a, long b) продемонстрировали стабильную и корректную работу.

Во-вторых, для веб-приложения JDI реализованы автоматизированные тесты с использованием Selenium WebDriver и TestNG. Созданы два тестовых класса: Task1 и Task2. Проверена корректность отображения главной страницы и страницы Different Elements, а также протестирована работа с элементами форм (чекбоксы, радиокнопки, выпадающие списки). Все тесты выполнены успешно, что подтверждает корректность работы тестируемого веб-приложения.

В результате выполнения работы были изучены ключевые аспекты автоматизации тестирования программного обеспечения, с акцентом на создании и написании unit-тестов с использованием JUnit 5, а также организации и выполнению тестов с помощью Selenium WebDriver и TestNG. Получены практические навыки работы с аннотациями @Test, @BeforeTest / @AfterTest, @ParameterizedTest, @ValueSource.

Написание тестов позволило получить уверенные навыки, необходимые для повышения эффективности и покрытия тестирования программного обеспечения. Освоенные инструменты (JUnit 5, Selenium WebDriver, TestNG) могут быть эффективно применены в реальных проектах для автоматизации рутинных проверок и существенного сокращения времени тестирования.