

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ  
**"САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО"**  
Институт компьютерных наук и технологий  
Направление **02.03.01** : Математика и компьютерные науки

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №4  
«Автоматизированное тестирование»

Исполнитель: \_\_\_\_\_

Яшнова Дарья Михайловна  
группа 5130201/20002

Руководитель: \_\_\_\_\_

Курочкин Михаил Александрович

« \_\_\_\_ » \_\_\_\_\_ 2025г

Санкт-Петербург, 2025

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Постановка задачи</b>	<b>4</b>
<b>2 JUnit</b>	<b>5</b>
2.1 Основные особенности JUnit	5
2.1.1 Аннотации	5
2.1.2 Простота использования	5
2.1.3 Совместимость	5
2.1.4 Этапы написания тестов	6
<b>3 Selenium WebDriver</b>	<b>6</b>
3.1 Возможности Selenium WebDriver	6
3.2 Архитектурные компоненты	7
3.3 Ключевые понятия	7
3.3.1 WebDriver	7
3.3.2 WebElement	7
3.3.3 Локаторы (By)	7
3.4 Процесс тестирования	7
3.4.1 Типичный сценарий	7
<b>4 TestNG</b>	<b>8</b>
<b>5 Основные особенности TestNG</b>	<b>8</b>
5.1 Гибкость конфигурации тестов	8
5.2 Параметризованные тесты	8
5.3 Зависимости между тестами	8
5.4 Параллельное выполнение	9
5.5 Этапы написания тестов	9
5.6 Интеграция с другими инструментами	9
<b>6 Описание выполнения работ</b>	<b>10</b>
6.1 Задание №1	10
6.1.1 Общая структура тестов	10
6.1.2 Методы тестирования	10
6.1.3 Выводы по результатам тестирования	13
6.2 Задание 2	15
6.2.1 DriverSetup	16
6.2.2 Task1Test	17
6.2.3 Task2Test	19
6.2.4 Результаты тестирования	22
<b>7 Заключение</b>	<b>23</b>
<b>Список литературы</b>	<b>24</b>

## Введение

Автоматизированное тестирование — это метод проверки программного обеспечения с использованием специализированных инструментов и фреймворков. Оно позволяет многократно выполнять одни и те же тестовые сценарии без прямого участия человека.

Автоматизация тестирования значительно ускоряет процесс проверки, сокращая время тестирования с нескольких недель до нескольких часов, даже при большом объёме тестовых сценариев. Это не только повышает эффективность работы, но и снижает влияние человеческого фактора, который может привести к ошибкам из-за усталости или отвлечения.

Автоматизация также позволяет стандартизировать тесты, гарантируя, что каждый сценарий будет проверен одинаково, независимо от исполнителя. Это важно для обеспечения качества и уменьшения субъективности в результатах тестирования.

Однако у автоматизированного тестирования есть свои ограничения, такие как отсутствие обратной связи о качестве продукта и ограниченные возможности в тестировании дизайна и пользовательского взаимодействия.

# 1 Постановка задачи

Цель работы:

- Создать модульные тесты для четырёх методов библиотеки `calculator.jar` с помощью JUnit и разместить результаты на GitHub, отправив запрос на проверку.
- С помощью TestNG и Selenium WebDriver провести тестирование веб-сайта в браузере Chrome. Необходимо написать два теста, которые проверяют отображение страниц. Организовать тесты в соответствии с Java Code Convention и запустить их через TestNG suite xml.

## 2 JUnit

JUnit — это фреймворк для модульного тестирования приложений на языке Java. Он позволяет разработчикам писать и выполнять повторяемые тесты для проверки корректности работы кода. Первая версия была создана Кентом Беком и Эрихом Гаммой как часть методологии экстремального программирования.

### 2.1 Основные особенности JUnit

#### 2.1.1 Аннотации

JUnit предоставляет следующие основные аннотации:

Листинг 1: Пример использования аннотаций JUnit

```
1  import org.junit.*;
2
3  public class ExampleTest {
4
5      @BeforeClass
6      public static void setUpClass() {
7          // Инициализация перед всеми тестами
8      }
9
10     @Before
11     public void setUp() {
12         // Подготовка перед каждым тестом
13     }
14
15     @Test
16     public void testMethod() {
17         // Тестовый метод
18     }
19
20     @After
21     public void tearDown() {
22         // Очистка после каждого теста
23     }
24
25     @AfterClass
26     public static void tearDownClass() {
27         // Финализация после всех тестов
28     }
29 }
```

#### 2.1.2 Простота использования

JUnit отличается:

- Интуитивно понятным API
- Минимальными требованиями к настройке
- Прозрачной моделью выполнения тестов

#### 2.1.3 Совместимость

JUnit поддерживает интеграцию с:

- IntelliJ IDEA
- Eclipse
- NetBeans
- Maven и Gradle
- CI-системами (Jenkins, Travis CI)

#### 2.1.4 Этапы написания тестов

**Реализация теста** Основные шаги:

1. Создание тестового класса
2. Добавление тестовых методов с аннотацией `@Test`
3. Использование методов `assert` для проверок

#### Настройка и очистка

Таблица 1: Методы настройки и очистки

Аннотация	Описание
<code>@Before</code>	Выполняется перед каждым тестом
<code>@After</code>	Выполняется после каждого теста
<code>@BeforeClass</code>	Выполняется один раз перед всеми тестами
<code>@AfterClass</code>	Выполняется один раз после всех тестов

**Запуск тестов** Способы запуска:

- Через IDE (кнопка запуска тестов)
- Через Maven: `mvn test`
- Через Gradle: `gradle test`
- Через командную строку с помощью `JUnitCore`

## 3 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.

### 3.1 Возможности Selenium WebDriver

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

- Автоматизировать работу с реальным браузером (локально или удалённо)
- Максимально точно имитировать действия пользователя
- Тестировать в различных браузерах (Chrome, Firefox, Edge, Safari)
- Интегрироваться с фреймворками тестирования (JUnit, TestNG)
- Поддерживать параллельное выполнение тестов

## 3.2 Архитектурные компоненты

Для работы с WebDriver требуется три основных компонента:

Таблица 2: Основные компоненты Selenium WebDriver

Компонент	Описание
Браузер	Реальный браузер конкретной версии, установленный на определённой ОС
Драйвер браузера	Веб-сервер, который запускает браузер и управляет им
Тест	Набор команд на языке программирования (Java, Python, C# и др.)

## 3.3 Ключевые понятия

### 3.3.1 WebDriver

Сущность для управления браузером. Пример создания экземпляра:

Листинг 2: Создание WebDriver для Chrome

```
1 WebDriver driver = new ChromeDriver();
```

### 3.3.2 WebElement

Абстракция веб-элементов с методами взаимодействия:

Листинг 3: Работа с WebElement

```
1 WebElement searchField = driver.findElement(By.name("q"));
2 searchField.sendKeys("Selenium WebDriver");
3 searchField.submit();
```

### 3.3.3 Локаторы (By)

Основные стратегии поиска элементов:

Таблица 3: Типы локаторов в Selenium

Метод	Пример
By.id()	driver.findElement(By.id("elementId"))
By.name()	driver.findElement(By.name("elementName"))
By.xpath()	driver.findElement(By.xpath("//div[@class='example']"))
By.cssSelector()	driver.findElement(By.cssSelector(".example"))

## 3.4 Процесс тестирования

### 3.4.1 Типичный сценарий

1. Инициализация WebDriver
2. Открытие тестируемого приложения
3. Взаимодействие с элементами
4. Проверка ожидаемых результатов
5. Завершение работы

## 4 TestNG

TestNG — это фреймворк для тестирования приложений на языке Java, вдохновленный JUnit, но предлагающий более мощные и гибкие возможности. Он поддерживает различные виды тестирования:

- Модульное тестирование (Unit testing)
- Функциональное тестирование (Functional testing)
- Интеграционное тестирование (Integration testing)
- End-to-end тестирование

## 5 Основные особенности TestNG

### 5.1 Гибкость конфигурации тестов

TestNG позволяет настраивать тесты с помощью:

- Аннотаций Java
- XML-файлов конфигурации

### 5.2 Параметризованные тесты

TestNG поддерживает два подхода к параметризации:

Таблица 4: Методы параметризации в TestNG

Метод	Аннотация
Через XML	@Parameters
Через код	@DataProvider

### 5.3 Зависимости между тестами

Листинг 4: Пример зависимостей

```
1  @Test
2  public void login() {
3      // Код входа
4  }
5
6  @Test(dependsOnMethods = "login")
7  public void search() {
8      // Код поиска
9  }
```



## 5.4 Параллельное выполнение

Уровни параллелизма:

- Методы
- Классы
- Тестовые наборы

## 5.5 Этапы написания тестов

### 1. Создание тестового класса

```
1 public class TestClass {
2     @Test
3     public void testMethod() {
4         // Тестовый код
5     }
6 }
```

### 2. Настройка пред- и пост-условий

```
1 @BeforeMethod
2 public void setUp() {
3     // Инициализация
4 }
5
6 @AfterMethod
7 public void tearDown() {
8     // Очистка
9 }
```

### 3. Запуск тестов

- Через IDE (IntelliJ IDEA, Eclipse)
- Через Maven: `mvn test`
- Через командную строку

### 4. Анализ результатов

- HTML-отчеты
- Интеграция с CI-системами (Jenkins)

## 5.6 Интеграция с другими инструментами

TestNG интегрируется с:

- Selenium WebDriver
- Maven/Gradle

## 6 Описание выполнения работ

### 6.1 Задание №1

В рамках лабораторной работы №1 требуется разработать юнит-тесты для библиотеки `calculator.jar`, выбрав четыре метода. Для этого следует использовать JUnit, включая аннотации `@Before` и `@After` для предварительной и последующей обработки данных, а также `@ParameterizedTest` и `@ValueSource` для параметризации тестов.

#### 6.1.1 Общая структура тестов

Тестовый класс `CalculatorTest` разработан для проверки корректности работы методов класса `Calculator`. В тестах используются:

- Аннотации JUnit 5: `@BeforeEach`, `@AfterEach` для инициализации и очистки
- Параметризованные тесты с `@ParameterizedTest`
- Различные источники данных: `@CsvSource`, `@ValueSource`
- Проверки утверждений с `assertEquals` и `assertThrows`

#### 6.1.2 Методы тестирования

##### Метод `testDivLong`

Проверяет операцию деления целых чисел с граничными значениями:

```
1  ParameterizedTest
2      @CsvSource({
3          "10, 2, 5",           // нормальные значения
4          "-10, 2, -5",        // отрицательное делимое
5          "10, -2, -5",        // отрицательный делитель
6          "-10, -2, 5",        // оба отрицательные
7          "0, 5, 0",           // нулевое делимое
8          "10, 1, 10",         // делитель 1
9          "10, 10, 1",         // делитель равен делимому
10         " Long.MAX_VALUE, 1, Long.MAX_VALUE", // максимальное long
11         "Long.MIN_VALUE, 1, Long.MIN_VALUE" // минимальное long
12     })
13     void testDivLong(long a, long b, long expected) {
14         if (b == 0) {
15             assertThrows(NumberFormatException.class, () -> calculator.
16                 div(a, b));
17         } else {
18             assertEquals(expected, calculator.div(a, b));
19         }
20     }
```

a	b	expected	Описание случая
10	2	5	Положительные значения
-10	2	-5	Отрицательное делимое
10	-2	-5	Отрицательный делитель
-10	-2	5	Оба параметра отрицательные
0	5	0	Нулевое делимое
10	1	10	Делитель равен 1
10	10	1	Делитель равен делимому
Long.MAX_VALUE	1	Long.MAX_VALUE	Максимальное значение long
Long.MIN_VALUE	1	Long.MIN_VALUE	Минимальное значение long

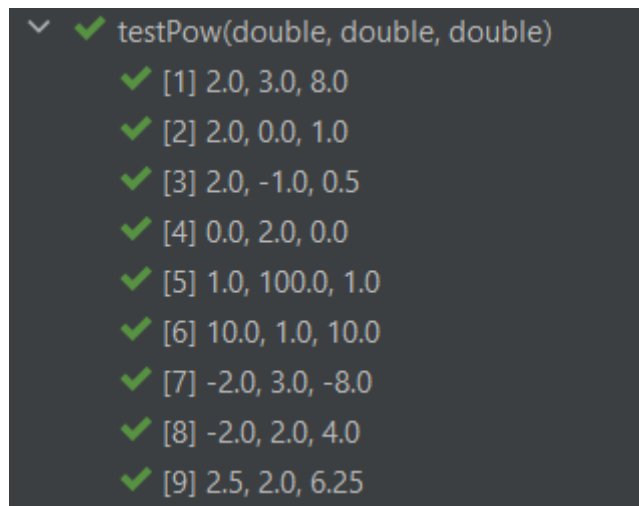


Рис. 1: Результаты тестирования testDivLong

### Метод testPow

Проверяет возведение в степень с различными комбинациями параметров:

```

1  @ParameterizedTest
2  @CsvSource({
3      "2.0, 3.0, 8.0",      // положительные числа
4      "2.0, 0.0, 1.0",      // нулевая степень
5      "2.0, -1.0, 0.5",     // отрицательная степень
6      "0.0, 2.0, 0.0",     // нулевое основание
7      "1.0, 100.0, 1.0",    // основание 1
8      "10.0, 1.0, 10.0",    // степень 1
9      "-2.0, 3.0, -8.0",    // отрицательное основание, нечетная
                             // степень
10     "-2.0, 2.0, 4.0",     // отрицательное основание, четная ст
                             // епень
11     "2.5, 2.0, 6.25"      // дробные числа
12 })
13 void testPow(double a, double b, double expected) {
14     assertEquals(expected, calculator.pow(a, b), 0.0001);
15 }

```

a	b	expected	Описание случая
2.0	3.0	8.0	Положительные числа
2.0	0.0	1.0	Нулевая степень
2.0	-1.0	0.5	Отрицательная степень
0.0	2.0	0.0	Нулевое основание
1.0	100.0	1.0	Основание равно 1
10.0	1.0	10.0	Степень равна 1
-2.0	3.0	-8.0	Отрицательное основание, нечетная степень
-2.0	2.0	4.0	Отрицательное основание, четная степень
2.5	2.0	6.25	Дробные числа

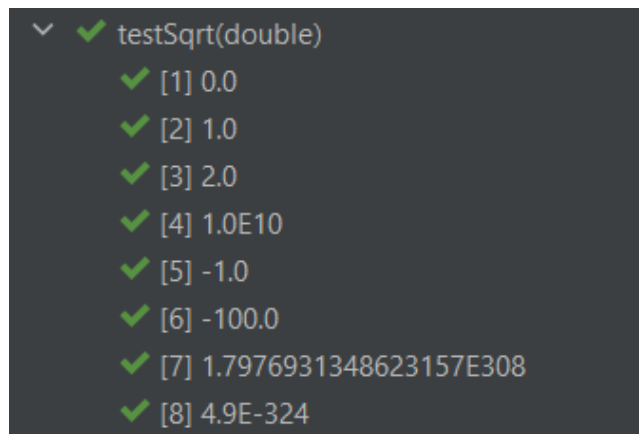


Рис. 2: Результаты тестирования testPow

### Метод testIsPositive

Проверяет определение положительных чисел:

```

1      @ParameterizedTest
2      @CsvSource({
3          "1, true",
4          "0, false",
5          "-1, false",
6          " Long.MAX_VALUE, true", // максимальное long
7          "Long.MIN_VALUE, false" // минимальное long
8      })
9      void testIsPositive(long val, boolean expected) {
10         assertEquals(expected, calculator.isPositive(val));
11     }

```

val	expected	Описание случая
1	true	Положительное число
0	false	Нулевое значение
-1	false	Отрицательное число
Long.MAX_VALUE	true	Максимальное значение long
Long.MIN_VALUE	false	Минимальное значение long

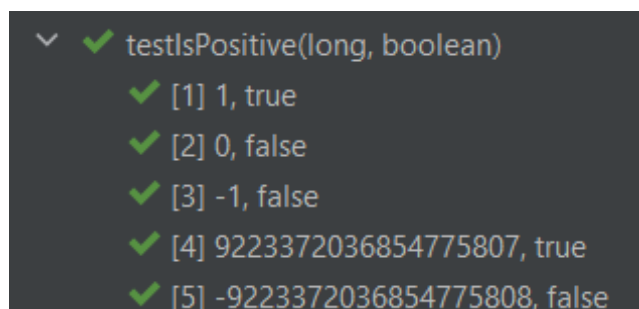


Рис. 3: Результаты тестирования testPositive

### Метод testSqrt

Проверяет вычисление квадратного корня с различными входными значениями:

```

1      @ParameterizedTest
2      @ValueSource(doubles = {
3          0.0, // ноль
4          1.0, // 1
5          2.0, // обычное число

```

```

6         1.0E10,      // большое число
7         -1.0,        // отрицательное
8         -100.0,      // отрицательное
9         Double.MAX_VALUE, // максимальное double
10        Double.MIN_VALUE // минимальное положительное double
11    })
12    void testSqrt(double a) {
13        double result = calculator.sqrt(a);
14        if (a >= 0) {
15            assertEquals(Math.sqrt(a), result, 0.0001);
16        } else {
17            assertEquals(Math.sqrt(-a), result, 0.0001);
18        }
19    }

```

а	Описание случая
0.0	Нулевое значение
1.0	Единица
2.0	Обычное положительное число
1.0E10	Большое положительное число
-1.0	Отрицательное число
-100.0	Отрицательное число
Double.MAX_VALUE	Максимальное значение double
Double.MIN_VALUE	Минимальное положительное значение double

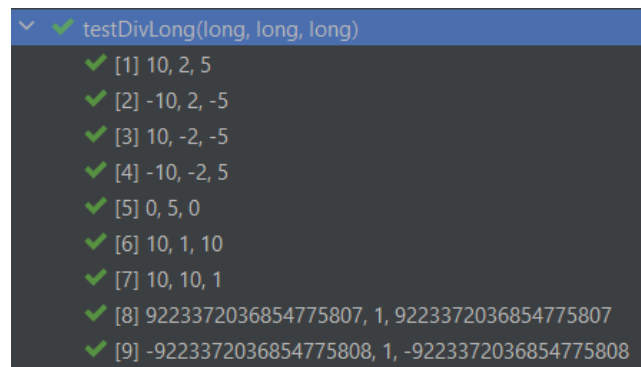


Рис. 4: Результаты тестирования testSqrt

### 6.1.3 Выводы по результатам тестирования

#### Тестирование метода div(long a, long b)

Все тесты успешно пройдены, что подтверждает корректную работу метода деления целых чисел.

Метод надежно работает во всем диапазоне значений long, корректно обрабатывает все комбинации знаков и граничные случаи, а также правильно реагирует на попытку деления на ноль.

#### Тестирование метода pow(double a, double b)

Все тестовые случаи успешно пройдены, что подтверждает правильность реализации возведения в степень. Метод корректно реализует возведение в степень для всех проверенных случаев, включая особые значения показателей степени и различные комбинации знаков основания.

#### Тестирование метода isPositive(long val)

Тесты успешно пройдены для всех проверяемых значений. Метод точно определяет знак числа во всем диапазоне значений `long`, включая граничные случаи и ноль.

**Тестирование метода `sqrt(double a)`** Все тестовые случаи успешно пройдены. Метод корректно вычисляет квадратный корень для всех проверенных значений, включая особые случаи. Для отрицательных чисел правильно вычисляет корень из модуля числа, что соответствует реализации.

### **Вывод**

Все тестируемые методы класса `Calculator` продемонстрировали корректную работу:

- Полное соответствие ожидаемому поведению для всех проверенных случаев
- Правильная обработка граничных значений и особых ситуаций
- Надежная работа во всем диапазоне допустимых значений
- Корректная реакция на исключительные ситуации (деление на ноль)

Однако по прохождению этих тестов нельзя судить о библиотеке в целом. Недостатки реализации данной библиотеки могли быть не выявлены из-за недостаточного объема тестирования.

## 6.2 Задание 2

Для выполнения задачи 2 необходимо:

- Подключить в проекте зависимости Selenium и TestNG, а также добавить драйвер для работы с браузером Chrome в проект.
- Создать два теста на языке Java с использованием Selenium WebDriver и фреймворка TestNG для проверки корректности отображения страниц сайта <https://jdi-testing.github.io/jdi-light/index.html> в браузере Chrome.
- Тесты должны соответствовать стандартам Java Code Convention и запускаться с помощью TestNG suite xml.

Первый и второй тест должны проверять 1 и 2 тестовый сценарий соответственно. Тестовые сценарии представлены на рисунках 2-3.

### Упражнение 1

Для этого упражнения используйте SoftAsserts.

#	Шаг	Данные	Ожидаемый результат
1	Open test site by URL	<a href="https://jdi-testing.github.io/jdi-light/index.html">https://jdi-testing.github.io/jdi-light/index.html</a>	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 logged in	"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: Тестовый сценарий 1

#	Шаг	Данные	Ожидаемый результат
1	Open test site by URL	<a href="https://jdi-testing.github.io/jdi-light/index.html">https://jdi-testing.github.io/jdi-light/index.html</a>	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 logged in	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 <ul style="list-style-type: none"> <li>for each checkbox there is an individual log row and value is corresponded to the status of checkbox</li> <li>for radio button there is a log row and value is corresponded to the status of radio button</li> <li>for dropdown there is a log row and value is corresponded to the selected value.</li> </ul>		Log rows are displayed and <ul style="list-style-type: none"> <li>checkbox name and its status are corresponding to selected</li> <li>radio button name and its status is corresponding to selected</li> <li>dropdown name and selected value is corresponding to selected</li> </ul>
10	Close Browser		Browser is closed

Рис. 6: Тестовый сценарий 2

### 6.2.1 DriverSetup

Класс `DriverSetup` отвечает за инициализацию и конфигурирование экземпляра `WebDriver`, необходимого для автоматизированного тестирования. Его основные задачи:

- Настройка и запуск браузера Chrome с использованием `ChromeDriver`.
- Аутентификация на целевом тестовом веб-сайте.
- Обеспечение корректного завершения работы браузера после выполнения тестов.

#### Структура класса

##### Метод `setup()`

Данный метод помечен аннотацией `@BeforeTest` `TestNG`, что гарантирует его выполнение перед каждым тестовым методом.

##### Функционал:

1. Установка пути к исполняемому файлу `ChromeDriver`.
2. Конфигурация HTTP-клиента, используемого `WebDriver`.
3. Создание экземпляра класса `ChromeDriver`.
4. Разворачивание окна браузера на весь экран для удобства работы.
5. Открытие целевой веб-страницы в браузере.
6. Выполнение процедуры аутентификации пользователя.

##### Метод `exit()`

- Данный метод помечен аннотацией `@AfterTest` `TestNG`, что гарантирует его выполнение после завершения всех тестовых методов.
- Закрывает браузер с помощью вызова метода `driver.close()`.

#### Используемые аннотации `TestNG`

Аннотация	Назначение
<code>@BeforeTest</code>	Выполняется перед запуском каждого теста.
<code>@AfterTest</code>	Выполняется после завершения всех тестов.



### 6.2.2 Task1Test

Класс `Task1Test` наследуется от `DriverSetup` и представляет собой набор автоматизированных тестов для проверки функциональности веб-страницы. Он опирается на Selenium WebDriver и TestNG для реализации тестов.

#### Ключевые особенности:

- Применение класса `SoftAssert` для накопления результатов проверок и предоставления полной картины о состоянии приложения после завершения тестового метода. Это позволяет выполнить все проверки, даже если некоторые из них не пройдены.
- Выполнение комплексной проверки элементов пользовательского интерфейса (UI) на соответствие ожидаемым значениям.
- Работа с элементами `iframe`. Обработка сценариев, в которых часть контента веб-страницы загружается во фреймы (HTML-элемент `<iframe>`, позволяющий встраивать другие веб-страницы в текущую).

#### Код класса

```
1
2 import org.openqa.selenium.By;
3 import org.openqa.selenium.WebElement;
4 import org.testng.annotations.Test;
5 import org.testng.asserts.SoftAssert;
6
7 import java.util.List;
8
9 public class Task1Test extends DriverSetup {
10
11     @Test
12     public void testTask1() {
13         SoftAssert softAssert = new SoftAssert();
14
15         // 2. Assert browser title
16         softAssert.assertEquals(driver.getTitle(), "Home Page");
17
18         // 4. Assert Username is logged in
19         softAssert.assertEquals(driver.findElement(By.id("user-name")).
20             getText(), "ROMAN IOVLEV");
21
22         // 5. Assert that there are 4 items on the header subsection are
23             displayed, and they have proper texts
24         List<WebElement> headerItems = driver.findElements(By.cssSelector("ul
25             .uui-navigation.nav > li"));
26         softAssert.assertEquals(headerItems.size(), 4);
27
28         String[] expectedHeaderTexts = {"HOME", "CONTACT FORM", "SERVICE", "
29             METALS & COLORS"};
30
31         for (int i = 0; i < headerItems.size(); i++) {
32             softAssert.assertTrue(headerItems.get(i).isDisplayed());
33             // Add an assertion to compare text
34             softAssert.assertEquals(headerItems.get(i).getText(),
35                 expectedHeaderTexts[i]);
36             // Compare expected and actual text
37             softAssert.assertAll();
38             softAssert.assertEquals(headerItems.get(i).getText(),
39                 expectedHeaderTexts[i]);
40         }
41     }
42 }
```

```

34     }
35
36
37
38     // 6. Assert that there are 4 images on the Index Page, and they are
        displayed
39     List<WebElement> images = driver.findElements(By.cssSelector(".
        benefit-icon > span"));
40     softAssert.assertEquals(images.size(), 4);
41     for (WebElement image : images) {
42         softAssert.assertTrue(image.isDisplayed());
43     }
44
45     // 7. Assert that there are 4 texts on the Index Page under icons,
        and they have proper text
46     List<WebElement> texts = driver.findElements(By.className("benefit-
        txt"));
47     softAssert.assertEquals(texts.size(), 4);
48
49     String[] expectedTexts = {
50         "To include good practices\\nand ideas from successful\\nEPAM
        project",
51         "To be flexible and\\ncustomizable",
52         "To be multiplatform",
53         "Already have good base\\n(about 20 internal and some
        external projects),\\nwish to get more..."
54     };
55
56     for (int i = 0; i < texts.size(); i++) {
57         softAssert.assertEquals(texts.get(i).getText(), expectedTexts[i])
        ;
58     }
59
60     // 8. Assert that there is the iframe with "Frame Button" exist
61     WebElement iframe = driver.findElement(By.id("frame"));
62     softAssert.assertTrue(iframe.isDisplayed());
63
64     // 9. Switch to the iframe and check that there is "Frame Button" in
        the iframe
65     driver.switchTo().frame(iframe);
66     WebElement frameButton = driver.findElement(By.id("frame-button"));
67     softAssert.assertTrue(frameButton.isDisplayed());
68
69     // 10. Switch to original window back
70     driver.switchTo().defaultContent();
71
72     // 11. Assert that there are 5 items in the Left subsection are
        displayed, and they have proper text
73     List<WebElement> leftMenuItems = driver.findElements(By.cssSelector("
        ul.sidebar-menu.left > li"));
74     softAssert.assertEquals(leftMenuItems.size(), 5);
75
76     String[] expectedMenuTexts = {"Home", "Contact form", "Service", "
        Metals & Colors", "Elements packs"};
77     for (int i=0; i < leftMenuItems.size(); i++) {
78         WebElement item = leftMenuItems.get(i);
79         softAssert.assertTrue(item.isDisplayed());
80         softAssert.assertEquals(item.getText(), expectedMenuTexts[i]);
81     }
82

```

```

83         softAssert.assertAll();
84
85     }
86 }

```

В данном тестовом классе реализованы следующие проверки пользовательского интерфейса:

- Проверка заголовка страницы;
- Проверка имени пользователя;
- Проверка заголовков в шапке страницы;
- Проверка изображений;
- Проверка текстов под иконками;
- Проверка iframe;
- Проверка левого меню;

Каждая из этих проверок направлена на подтверждение корректности работы отдельных компонентов пользовательского интерфейса и их соответствия требованиям. Совокупность тестов обеспечивает комплексную проверку основных элементов веб-страницы.

### 6.2.3 Task2Test

Класс `Task2Test`, наследующий от `DriverSetup`, реализует функциональное тестирование веб-интерфейса через следующие аспекты:

1. **Модульность:** Тесты разделены на три независимых метода.
2. **Тестовое покрытие** включает:
  - Верификацию заголовка страницы
  - Сценарий аутентификации пользователя
  - Тестирование интерактивных элементов (чекбоксы, радио-кнопки, select-элементы)
  - Проверку системных логов операций
3. **Дополнительные проверки:** Анализ логов изменения состояния UI-компонентов.
4. **Техники тестирования:** Комбинация стратегий поиска веб-элементов.

#### Код класса

```

1  import org.openqa.selenium.By;
2  import org.openqa.selenium.WebElement;
3  import org.testng.annotations.Test;
4  import org.testng.asserts.SoftAssert;
5
6  import java.util.Arrays;
7  import java.util.List;
8
9  import static org.testng.Assert.assertEquals;
10
11 public class Task2Test extends DriverSetup {
12
13     @Test
14     public void testTitle() {
15         // 1. Проверить заголовок страницы
16         driver.get("https://jdi-testing.github.io/jdi-light/index.html");
17         assertEquals(driver.getTitle(), "Home Page");
18     }

```

```

19
20     @Test
21     public void testLogin() {
22         // 2. Выполнить логин пользователя
23         driver.findElement(By.id("user-icon")).click();
24         driver.findElement(By.id("name")).sendKeys("Roman");
25         driver.findElement(By.id("password")).sendKeys("Jdi1234");
26         driver.findElement(By.id("login-button")).click();
27         assertEquals(driver.findElement(By.id("user-name")).getText(), "
            ROMAN IOVLEV");
28     }
29
30     @Test
31     public void testElement() {
32         // 3. Перейти на страницу с чекбоксами, радиокнопками и выпадающи
            ми списками
33         driver.findElement(By.cssSelector("body>header>div>nav>ul.uui-
            navigation.nav.navbar-nav.m-18>li>a>span")).click();
34         driver.findElement(By.xpath("/html/body/header/div/nav/ul[1]/li
            [3]/ul/li[8]/a")).click();
35
36         // 4. Выбрать чекбоксы "Water" и "Wind"
37         List<WebElement> checkboxes = driver.findElements(By.className("
            label-checkbox"));
38         for (WebElement checkbox : checkboxes) {
39             if (checkbox.getText().equals("Water") || checkbox.getText().
            equals("Wind")) {
40                 checkbox.click();
41             }
42         }
43         // 5. Выбрать радиокнопку "Selen"
44         List<WebElement> radios = driver.findElements(By.className("label
            -radio"));
45         for (WebElement radio : radios) {
46             if (radio.getText().equals("Selen")) {
47                 radio.click();
48             }
49         }
50
51         // 6. Выбрать опцию "Yellow" из выпадающего списка
52         List<WebElement> options = driver.findElements(By.tagName("option
            "));
53         for (WebElement option : options) {
54             if (option.getText().equals("Yellow")) {
55                 option.click();
56             }
57         }
58
59         // 7. Проверить логи изменений
60         final int logIndexStart = 9;
61         String logsText = driver.findElement(By.cssSelector("ul.panel-
            body-list.logs")).getText();
62         String[] logLinesArray = logsText.split("\n");
63         List<String> logLines = Arrays.stream(logLinesArray).map(log ->
            log.substring(logIndexStart)).toList();
64
65         // Ожидаемые записи в логах
66         List<String> expectedLogEntries = List.of(
67             "Цвета: value changed to Yellow",
68             "металл: value changed to Selen",

```

```

69         "Ветер: condition changed to true",
70         "Вода: condition changed to true"
71     );
72
73     // Проверить соответствие логов
74     assertEquals(logLines, expectedLogEntries);
75
76 }
77 }

```

Метод `testTitle()` Реализует базовую проверку веб-страницы:

- **Назначение:** Верификация корректности заголовка страницы
- **Реализация:**
  - Автоматический переход на тестовый URL
  - Проверка через стандартный `assert`

Метод `testLogin()` Осуществляет тестирование авторизации:

- **Основной сценарий:**
  - Выполнение полного цикла входа в систему
  - Валидация отображения имени пользователя в UI
- **Технические особенности:** Использование локаторов ID для поиска элементов

Метод `testElement()` Комплексная проверка интерактивных элементов:

#### 1. Инициализация:

- Навигация на специализированную тестовую страницу
- Подготовка тестового окружения

#### 2. Тестирование чекбоксов:

- Активация элементов “Water” и “Wind”
- Проверка изменения состояния

#### 3. Тестирование радио-кнопок:

- Выбор значения “Selen”
- Верификация exclusive-выбора

#### 4. Тестирование выпадающего списка:

- Выбор опции “Yellow”
- Проверка применения выбора

#### 5. Анализ логов:

- Проверка генерации системных логов
- Соответствие записей выполненным действиям
- Полнота отражения изменений состояния

#### 6.2.4 Результаты тестирования

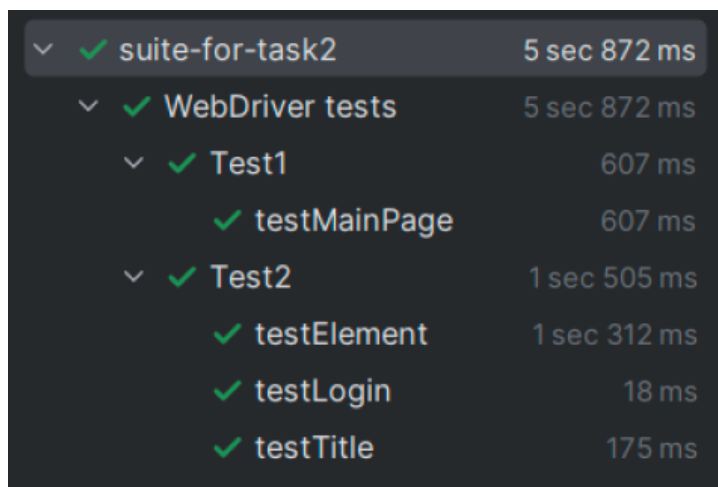
В рамках выполнения работы №2 была успешно проведена проверка отображения компонентов пользовательского интерфейса и их взаимодействия. Использование Selenium WebDriver и TestNG обеспечило корректную настройку и управление браузером Chrome в процессе выполнения тестов. Это, в свою очередь, предоставило возможность удобного доступа к элементам пользовательского интерфейса по их ID, CSS-селекторам и именам классов.

В ходе работы был успешно применен механизм "мягких" утверждений (Soft Assertions), что позволило выполнить комплексную проверку пользовательского интерфейса сайта в рамках единого запуска тестового метода `Task1Test`.

Протестированы различные элементы страницы, включая:

- Заголовок страницы.
- Наличие и содержимое пунктов меню.
- Отображение изображений.
- Текст под изображениями.
- Наличие элемента `iframe` и элементов в левом меню внутри него.

Управление сессией браузера, включающее автоматическое закрытие браузера по завершении выполнения всех тестов, обеспечило изоляцию тестовой среды и предотвратило влияние предыдущих тестов на результаты последующих запусков.



✓ suite-for-task2	5 sec 872 ms
✓ WebDriver tests	5 sec 872 ms
✓ Test1	607 ms
✓ testMainPage	607 ms
✓ Test2	1 sec 505 ms
✓ testElement	1 sec 312 ms
✓ testLogin	18 ms
✓ testTitle	175 ms

Рис. 7: Результаты тестирования

## 7 Заключение

В процессе работы были рассмотрены ключевые моменты автоматизации тестирования программного обеспечения.

Особое внимание было уделено созданию и написанию модульных тестов с использованием JUnit, а также организации и выполнению тестов с помощью Selenium WebDriver и TestNG. Были освоены навыки работы с аннотациями @Test, @BeforeTest/@AfterTest, @ParameterizedTest и @ValueSource.

В ходе работы были выполнены две лабораторные работы. Для каждой из них было написано несколько тестов, и все они были успешно выполнены.

Освоенные инструменты JUnit, Selenium WebDriver и TestNG могут быть использованы в реальных проектах для автоматизации проверок и сокращения времени тестирования.

## Список литературы

- [1 ] Майерс, Г. Искусство тестирования программ. – Санкт-Петербург: Диалектика, 2012. – С. 272.