МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Лабораторная работа №4
«Автоматизированное тестирование»
по дисциплине
«Методы тестирования программного обеспечения»

Студент,
группы 5130201/20102 _____ Тищенко А. А.

Преподаватель _____ Курочкин М. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

Содержание

1 Постановка задачи

Данная лабораторная работа делится на две части. В первой части необходимо разработать набор юнит-тестов к предоставленной библиотеке calculator.jar, которая содержит методы для проведения разнообразных операций над числами. Во второй части необходимо протестировать сайт в Chrome путем написания двух тестов, проверяющих корректность отображения страниц. Использовать для тестирования Selenuim WebDriver.

2 Средства автоматизации тестирования

2.1 JUinit

JUnit — это фреймворк, созданный для тестирования программного обеспечения на языке Java. Он предназначен для разработки и выполнения автоматизированных тестов, что дает разработчикам возможность проверять правильность функционирования своего кода и обнаруживать ошибки на ранних стадиях разработки.

2.1.1 Основные особенности JUnit

- **Модульное тестирование:** JUnit поддерживает тестирование отдельных модулей, позволяя разработчикам проверять работу отдельных компонентов (методов и классов) независимо друг от друга.

- **Аннотации:** JUnit использует аннотации, такие как `@Test`, `@Before`, `@After`, которые упрощают написание тестов и делают код более читабельным. Например, аннотация `@Before` позволяет выполнять определенный код перед каждым тестом, а @After — после его завершения.

- **Ассерты:** JUnit предоставляет различные методы ассертов, такие как assertEquals, assertTrue, assertNotNull, которые помогают проверять ожидаемые результаты тестов.

- **Группировка тестов:** Тесты могут быть сгруппированы с помощью аннотации @Suite, что позволяет запускать их вместе.

- **Интеграция:** JUnit просто интегрируется с различными инструментами и фреймворками, такими как Maven, Gradle и широко используемые IDE, включая Eclipse и IntelliJ IDEA, что упрощает процесс тестирования.

2.1.2 Преимущества использования JUnit

- **Упрощение процесса тестирования:** JUnit делает написание тестов более удобным и структурированным.

- **Автоматизация:** Позволяет автоматизировать тесты, что снижает вероятность ошибок и повышает качество кода.

- **Поддержка непрерывной интеграции:** JUnit хорошо совместим с системами непрерывной интеграции, позволяя запускать тесты автоматически при каждом изменении кода.

- **Снижение затрат на отладку:** Регулярное тестирование помогает выявлять проблемы на ранних стадиях, что уменьшает расходы на их исправление.

2.1.3 Функционал библиотеки

JUnit имеет широкий функционал для проверки совпадения ожидаемого результата, и результата полученного тестируемым методом.

Класс `junit.framework.Assert` предоставляет набор статических методов для проверки различных условий в тестах.

- `assertEquals(expected, actual)` - проверяет равенство двух значений. Если значения не равны, тест завершается с ошибкой. Имеет перегрузки для различных типов данных.
- `assertFalse(condition)` - проверяет, что переданное булево значение является `false`. Если значение `true`, тест завершается с ошибкой.
- `assertNotNull(object)` - проверяет, что объект не является `null`. Если объект `null`, тест завершается с ошибкой.
- `assertNull(object)` - проверяет, что объект является `null`. Если объект не `null`, тест завершается с ошибкой.
- `assertNotSame(unexpected, actual)` - проверяет, что два объекта не ссылаются на один и тот же экземпляр. Если ссылки идентичны, тест завершается с ошибкой.
- `assertSame(expected, actual)` - проверяет, что два объекта ссылаются на один и тот же экземпляр. Если ссылки разные, тест завершается с ошибкой.
- `assertTrue(condition)` - проверяет, что переданное булево значение является `true`. Если значение `false`, тест завершается с ошибкой.

Класс `junit.framework.TestCase` наследуется от `junit.framework.Assert` и предоставляет базовую функциональность для создания тестовых случаев.

- `run()` - основной метод, выполняющий тест. Содержит логику запуска и выполнения тестового случая.
- `setUp()` - метод, выполняемый перед каждым тестом. Используется для инициализации тестового окружения и подготовки данных. В аннотационной версии JUnit 4+ заменён на `@Before`.
- `tearDown()` - метод, выполняемый после каждого теста. Используется для очистки ресурсов после выполнения теста. В аннотационной версии JUnit 4+ заменён на `@After`.

2.1.4 Этапы написания тестов

1. **Реализация теста**: Написание тестового метода и аннотирование его с помощью. `@Test`

2. **Настройка и очистка** Использование аннотаций `@Before` и `@After` для выполнения операйи перед и после теста.

3. **Запуск тестов:** Использование встроенных средств IDE или командной строки для выполнения тестов.

2.2 Selenium WebDriver

Selenium WebDriver — это фреймворк с открытым исходным кодом для автоматизации тестирования веб-приложений. Он предоставляет программный интерфейс для взаимодействия с браузерами, позволяя эмулировать действия пользователя на веб-страницах. **Основные особенности Selenium WebDriver**:

- **Кроссбраузерность:** Поддержка всех популярных браузеров, включая Chrome, Firefox, Safari, Edge и Opera.
- **Многоязычность:** Возможность написания тестовых скриптов на различных языках программирования — Java, Python, C#, Ruby, JavaScript, PHP и Perl.
- **Прямое взаимодействие:** WebDriver напрямую отправляет команды браузеру и получает результаты, что обеспечивает более точное воспроизведение пользовательских действий.
- **Кроссплатформенность:** Возможность запуска тестов на различных операционных системах (Windows, MacOS, Linux).
- **Параллельное выполнение:** Поддержка одновременного запуска тестов в разных браузерах для ускорения тестирования.

Архитектура Selenium WebDriver состоит из четырех основных компонентов:

- **Selenium Client Libraries:** Набор библиотек для различных языков программирования, позволяющих писать и запускать тесты на предпочитаемом языке. JSON Wire Protocol: REST API на основе JSON, обеспечивающий передачу информации между клиентом и сервером через HTTP.
- **Browser Drivers:** Специфичные для каждого браузера драйверы (ChromeDriver, GeckoDriver для Firefox и др.), которые получают команды и выполняют их в соответствующем браузере.
- **Browsers:** Сами браузеры, в которых выполняются тестовые сценарии.

Этапы работы с WebDriver:

1. **Инициализация:** Создание экземпляра WebDriver и открытие браузера.
2. **Навигация:** Переход к нужной веб-странице с помощью метода get().
3. **Поиск элементов:** Обнаружение элементов на странице с использованием различных локаторов (ID, XPath, CSS-селекторы и др.).

4. **Взаимодействие:** Выполнение действий над элементами (клик, ввод текста, выбор из выпадающих списков и т.д.).
5. **Ожидание:** Использование явных и неявных ожиданий для синхронизации с динамическими элементами страницы.
6. **Проверка:** Получение информации о состоянии элементов и проверка результатов.
7. **Завершение:** Закрытие браузера и освобождение ресурсов.

Принцип выполнения команд:

1. Команда из тестового скрипта преобразуется в HTTP-запрос через JSON Wire Protocol.
2. Запрос передается соответствующему драйверу браузера.
3. Драйвер интерпретирует запрос и выполняет необходимые действия в браузере.
4. Результат действия возвращается обратно в виде HTTP-ответа.
5. Ответ преобразуется в формат, понятный тестовому скрипту.

3 Описание выполненных работ

3.1 Работа №1

В ходе работы необходимо прописать юнит тесты для методов библиотеки calcualtor.jar, реализующий функционал калькулятора, производящего вычисления суммы, разности, умножения, деления, возведения в степень, ивзлечение корня, а также значений базовых тригонометрических функций. Дря реализации тестов, необходимо использовать JUnit.

3.1.1 Класс CalculatorTest

Все тесты содержатся в классе CalculatorTest. В методе setUp() инициализируется объект `Calculator`. Константой DELTA задается допустимая погрешность. Код определения полей класса и метода setUp() представлен в листинге 1.

Листинг 1. Класс CalculatorTest

```
1  class CalculatorTest {
2      private static final double DELTA = 0.0001;
3      private Calculator calculator;
4
5      @BeforeEach
6      void setUp() {
7          calculator = new Calculator();
8      }
9  }
```

3.1.2 Тесты для метода Sum

В классе CalculatorTest реализованы следующие тесты для метода sum:

- `testLongSum` - тест для проверки суммы двух чисел типа long.
- `testDoubleSum` - тест для проверки суммы двух чисел типа double.

Код тестов представлен в листинге 2. Тесты параметризованы с помощью аннотации `@ParameterizedTest` и `@CsvSource`.

Листинг 2. Тесты для метода Sum

```
1  @ParameterizedTest
2  @CsvSource({
3          "1, 2, 3",
4          "5, 0, 5",
5          "-7, 7, 0",
6          "9223372036854775806, 1, 9223372036854775807"
7  })
8  void testLongSum(long a, long b, long expected) {
9      assertEquals(expected, calculator.sum(a, b));
```

```
10  }
11
12  @ParameterizedTest
13  @CsvSource({
14          "1.5,␣2.5,␣4.0",
15          "5.5,␣0.0,␣5.5",
16          "−3.5,␣3.5,␣0.0",
17          "0.1,␣0.2,␣0.3"
18  })
19  void testDoubleSum(double a, double b, double expected) {
20      assertEquals(expected, calculator.sum(a, b), DELTA);
21  }
```

Результаты запуска тестов:

Результаты запуска тестов представлены на рисунке 1.

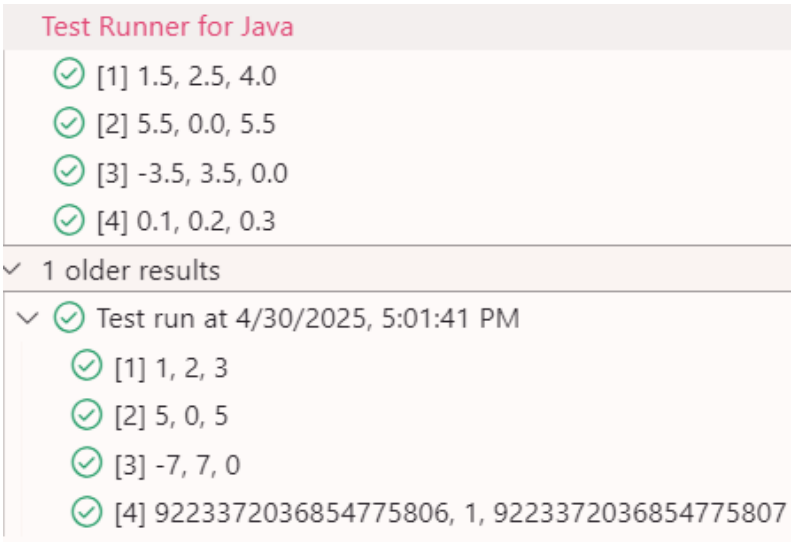

Рис. 1. Результаты запуска тестов для метода sum

По итогам запуска, метод sum прошел все тесты.

3.1.3 Тесты для метода Mul

В классе CalculatorTest реализованы следующие тесты для метода mul:

- `testLongMul` - тест для проверки произведения двух чисел типа long.
- `testDoubleMul` - тест для проверки произведения двух чисел типа double.

Код тестов представлен в листинге 3.

Листинг 3. Тесты для метода Mul

```
1  @ParameterizedTest
2  @CsvSource({
3          "2.5,␣3.0,␣7.5",
```

```
        "0.0, 5.5, 0.0",
        "-2.5, 3.0, -7.5",
        "-2.5, -3.0, 7.5"
})
void testDoubleMult(double a, double b, double expected) {
    assertEquals(expected, calculator.mult(a, b), DELTA);
}

@ParameterizedTest
@CsvSource({
        "2, 3, 6",
        "0, 5, 0",
        "-2, 3, -6",
        "-2, -3, 6",
        "1000, 1000, 1000000"
})
void testLongMult(long a, long b, long expected) {
    assertEquals(expected, calculator.mult(a, b));
}
```

Результаты запуска тестов:

Результаты запуска тестов представлены на рисунке 2.

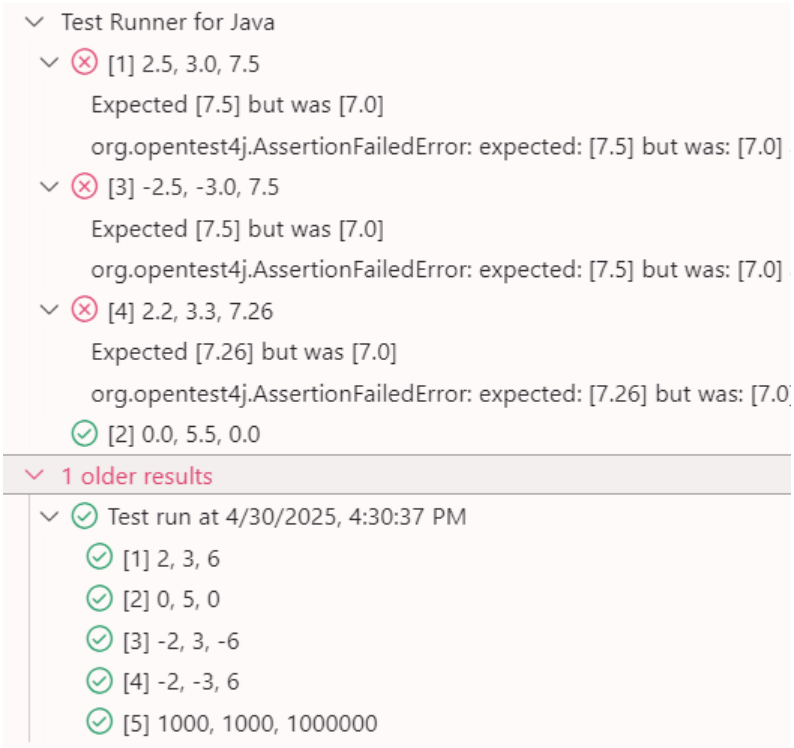

Рис. 2. Результаты запуска тестов для метода mul

По итогам запуска, метод mul для типа long прошел все тесты, а для типа double прошёл лишь 1 из 4 тестов.

3.1.4 Тесты для метода Sqrt

В классе CalculatorTest реализованы следующие тесты для метода sqrt:

- `testSqrt` - тест для проверки квадратного корня числа типа double. Проверяет положительные, отрицательные значения, а также 0.

Код тестов представлен в листинге 4.

Листинг 4. Тесты для метода Sqrt

```
1  @ParameterizedTest
2  @ValueSource(doubles = { 4.0, 0.0, −4.0, 1000000.0 })
3  void testSqrt(double value) {
4      double expected = Math.sqrt(value);
5      assertEquals(expected, calculator.sqrt(value), DELTA);
6  }
```

Результаты запуска тестов:

Результаты запуска тестов представлены на рисунке 3.

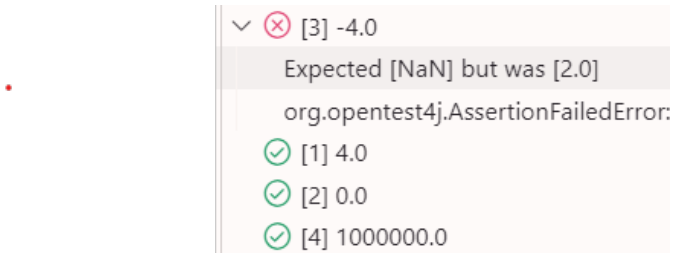

Рис. 3. Результаты запуска тестов для метода sqrt

По итогам запуска, метод sqrt не прошёл один из четырёх тестов. Метод неккоректно работает с отрицательными числами.

3.1.5 Тесты для метода Tg

В классе CalculatorTest реализованы следующие тесты для метода tg:

- `testTg` - тест для проверки тангенса числа типа double. Проверяет положительные, отрицательные значения, а также 0.

Код тестов представлен в листинге 5.

Листинг 5. Тесты для метода Tg

```
1  @ParameterizedTest
2  @ValueSource(doubles = { 0, Math.PI / 6, Math.PI / 4, −Math.PI / 3, 10 })
3  void testTg(double angle) {
4      double expected = Math.tan(angle);
5      double actual = calculator.tg(angle);
6      assertEquals(expected, actual, 0.0001);
7  }
```

Результаты запуска тестов:

Результаты запуска тестов представлены на рисунке 4.

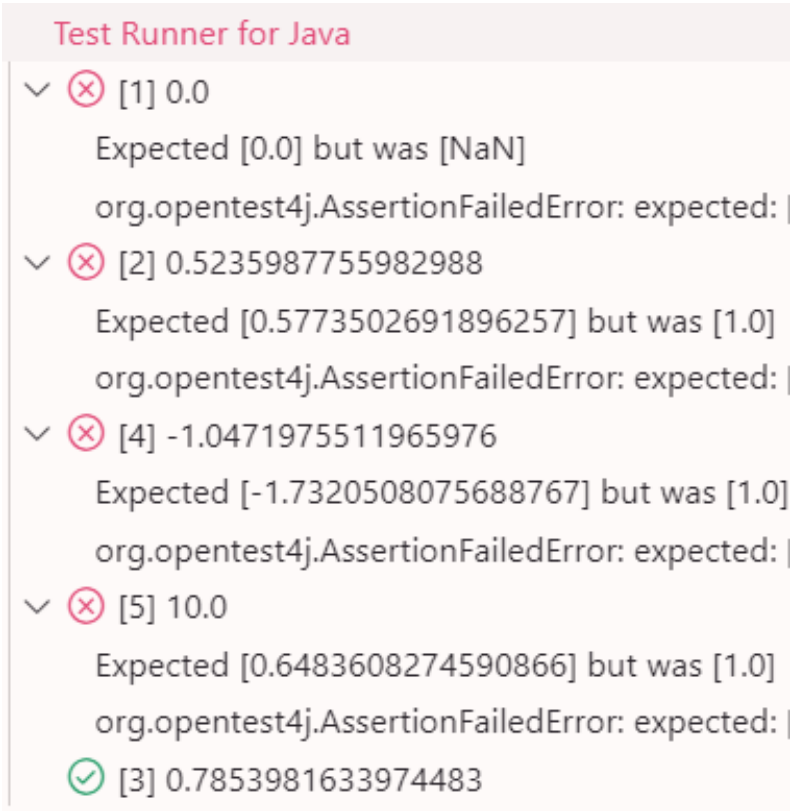

Рис. 4. Результаты запуска тестов для метода tg

По итогам запуска, метод tg прошел лишь 1 из 5 тестов.

3.1.6 Результаты работы №1

В результате комплексного тестирования библиотеки `calculator.jar` были получены следующие результаты:

1. **Тестирование метода сложения (SumTests)**:
    - Все тесты корректности сложения для целых чисел и чисел с плавающей точкой прошли успешно
    - Операции с граничными значениями (максимальные/минимальные значения `long` и `double`) выполняются без переполнения
    - Особые случаи (сложение с нулем) соответствуют ожидаемым результатам.
2. **Тестирование метода умножения (MulTests)**:
    - Метод для умножения целых чисел прошёл все тесты.

- Метод для умножения чисел с плавающей точкой прошёл 1 из 4 тестов.

3. **Тестирование метода квадратного корня (SqrtTests)**:

   - Метод прошёл 3 из 4 тестов.
   - Метод корректно обрабатывает положительные значения и 0.
   - Однако отрицательные числа обрабатываются некорректно. Вместо того, чтобы вернуть ошибку или комплексное число, метод вычисляет значения корня по модулю числа.

4. **Тестирование метода тангенса (tgTests)**:

   - Метод прошёл 1 из 5 тестов.
   - Функция тангенса фактически всегда возвращает 1 (кроме случаев, когда `sin(a) = 0`)
   - Свойство нечетности (`tg(-x) = -tg(x)`) не соблюдается, значение `tg(x)` равно 1.
   - Функция возвращает неккоректное значение при попытке вычислить `tg(0)`. При вычислении возваращется `NaN`, а не 0, как ожидается.

5. **Общие выводы:**

   Два из четырёх тестируемых методов не прошли все тесты. В реализации методов умножения и вычисления тангенса присутствуют ошибки.

   Метод умножения неккоректно работает с дробными числами. А метод для вычисления тангенса возвращает 1 для любого входного значения, кроме 0. При нулевом входном значении метод возвращает `NaN`.

3.2 Работа №2

В ходе выполнения работы №2 необходимо было реализовать два теста для тестирования web-сайта с помощью библиотеки Selenium WebDriver. Тесты должны проверять, что элементы сайта `https://jdi-testing.github.io/jdi-light/index.html` отображаются корректно и позволяют взаимодействовать с собой правильным образом. Реализация тестов должна быть выполнена согласно Java Code Convention и запускаться с помощью TestNG suite xml.

Тесты разделяются на 2 класса, в которых необходимо реализовать тесты, связанные с взаимодействием сайта.

В первом наборе тестов необходимо проверить корректность отображения страницы. Все сценарии, которые необходимо проверить, представлены в таблице 1

Таблица 1. Тест-кейсы для веб-приложения (с использованием SoftAsserts)

№	Шаг тестирования	Данные	Ожидаемый результат
1	Открыть тестовый сайт по URL	https://jdi-testing.github.io/jdi-light/index.html	Тестовый сайт открыт
2	Проверить заголовок браузера	"Home Page"	Заголовок соответствует "Home Page"
3	Выполнить вход в систему	Логин: Roman, Пароль: Jdi1234	Пользователь авторизован
4	Проверить отображение имени пользователя	"ROMAN IOVLEV"	Имя отображается корректно
5	Проверить пункты меню в шапке	"HOME "CONTACT FORM "SERVICE "METALS & COLORS"	4 пункта меню с правильным текстом
6	Проверить изображения на странице	4 изображения	Все изображения отображаются
7	Проверить тексты под иконками	4 текстовых блока	Тексты соответствуют ожидаемым
8	Проверить наличие iframe	Кнопка "Frame Button"	iframe существует
9	Проверить кнопку во фрейме	-	Кнопка "Frame Button"доступна
10	Вернуться в основное окно	-	Фокус на основном окне
11	Проверить левое меню	"Home "Contact form "Service "Metals & Colors "Elements packs"	5 пунктов меню с правильным текстом
12	Закрыть браузер	-	Браузер закрыт

Во втором наборе тестов необходимо проверить корректность взаимодействия пользователя с сайтом (в частности, правильность выбора чекбоксов, радиобаттонов и элементов из выпадающего списка). Таблица тестов, необходимых к реализации во втором упражнении, представлена в таблице 2.

Таблица 2. Тест-кейсы для веб-приложения (с использованием SoftAsserts)

№	Шаг тестирования (Testing Step)	Данные (Data)	Ожидаемый результат (Expected Result)
1	Открыть тестовый сайт по URL	https://jdi-testing.github.io/jdi-light/index.html	Тестовый сайт открыт (Test site is opened)
2	Проверить заголовок браузера (Check browser title)	"Home Page"	Заголовок соответствует "Home Page"(Title matches "Home Page")
3	Выполнить вход в систему (Perform login)	username: Roman, password: Jdi1234	Пользователь авторизован (User is logged in)
4	Проверить отображение имени пользователя (Verify username display)	"ROMAN IOVLEV"	Имя отображается корректно (Name is displayed correctly)
5	Открыть через хедер меню Service -> Different Elements Page (Navigate using header menu: Service -> Different Elements Page)		Страница открыта (Page is opened)
6	Выбрать чекбоксы (Select checkboxes)	Water, Wind	Элементы отмечены (Elements are checked)
7	Выбрать переключатель (Select radio)	Selen	Элемент отмечен (Element is selected)
8	Выбрать в один из выпадающего списка (Select in dropdown)	Yellow	Элемент выбран (Element is chosen)
9	Проверить, что для каждого чекбокса, radio и dropdown есть отдельная строчка лога (Verify that for each checkbox, radio, and dropdown there is a separate log row)		Логи отображаются и соответствуют выбранным значениям (Logs are displayed and correspond to selected values)
10	Закрыть браузер (Close browser)		Браузер закрыт (Browser is closed)

3.2.1 Класс DriverSetup

Класс `DriverSetup` выполняет первоначальную настройку `WebDriver` перед запуском тестов. Он устанавливает системные свойства для Chrome Driver, настраивает HTTP клиент и создает экземпляр Chrome Driver, открывая тестовый сайт и выполняет авторизацию пользователя.

Листинг 6.

```
1  public class DriverSetup {
2      protected static WebDriver driver;
3
4      @BeforeTest
5      public static void setup() {
```

```
6         System.setProperty("webdriver.chrome.driver", "src/test/resources/chromedriver.exe");
7         System.setProperty("webdriver.http.factory", "jdk-http-client");
8
9         driver = new ChromeDriver();
10
11        driver.navigate().to("https://jdi-testing.github.io/jdi-light/index.html");
12
13        driver.findElement(By.cssSelector("html > body > header > div > nav > ul.uui-navigation.
      navbar-nav.navbar-right > li > a > span")).click();
14        driver.findElement(By.id("name")).sendKeys("Roman");
15        driver.findElement(By.id("password")).sendKeys("Jdi1234");
16        driver.findElement(By.id("login-button")).click();
17    }
18
19    @AfterTest
20    public static void exit() {
21        driver.close();
22    }
23 }
```

Класс содержит следующие элементы:

- **driver**: Защищенная статическая переменная типа WebDriver, представляющая экземпляр браузера Chrome, используемый для выполнения тестов.
- **@BeforeTest setup()**: Статический метод, помеченный аннотацией @BeforeTest, выполняемый перед всеми тестовыми методами. Он выполняет следующие действия:
  - Устанавливает системные свойства ‘webdriver.chrome.driver‘ и ‘webdriver.http.factory‘.
  - Создает экземпляр ‘ChromeDriver‘.
  - Открывает тестовый сайт по URL: https://jdi-testing.github.io/jdi-light/index.html.
  - Выполняет вход в систему, находя и заполняя поля логина и пароля, а также нажимая кнопку входа.
- **@AfterTest exit()**: Статический метод, помеченный аннотацией ‘@AfterTest‘, выполняемый после всех тестовых методов. Он закрывает браузер с помощью ‘driver.close()‘.

3.2.2 Класс Task1Test

Класс `Task1Test` является тестовым классом, который выполняет проверки различных элементов на главной странице веб-сайта. Для проверки ожидаемых результатов используются "мягкие"утверждения (SoftAsserts), что позволяет продолжить выполнение теста даже в случае неудачи одного из утверждений.

Листинг 7. Класс Task1Test

```
1 package edu.hsai.homework2;
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
8  import java.util.List;
9
10 public class Task1Test extends DriverSetup {
11     @Test
12     public void testTask1() {
13         SoftAssert softAssert = new SoftAssert();
14
15         softAssert.assertEquals(driver.getTitle(), "Home␣Page");
16
17         softAssert.assertEquals(driver.findElement(By.id("user-name")).getText(), "
       ROMAN␣IOVLEV");
18
19         List<WebElement> headerItems = driver.findElements(By.cssSelector("ul.uui-
       navigation.nav␣>␣li"));
20         softAssert.assertEquals(headerItems.size(), 4);
21
22         String[] expectedHeaderTexts = {"HOME", "CONTACT␣FORM", "SERVICE", "METALS␣&␣
       COLORS"};
23         for (int i = 0; i < headerItems.size(); i++) {
24             softAssert.assertTrue(headerItems.get(i).isDisplayed());
25             softAssert.assertEquals(headerItems.get(i).getText(), expectedHeaderTexts[
       i]);
26         }
27
28         List<WebElement> images = driver.findElements(By.cssSelector(".benefit-icon␣>␣
       span"));
29         softAssert.assertEquals(images.size(), 4);
30         for (WebElement image : images) {
31             softAssert.assertTrue(image.isDisplayed());
32         }
33
34         List<WebElement> texts = driver.findElements(By.className("benefit-txt"));
35         softAssert.assertEquals(texts.size(), 4);
36
37         String[] expectedTexts = {
38             "To␣include␣good␣practices\nand␣ideas␣from␣successful\nEPAM␣project",
39             "To␣be␣flexible␣and\ncustomizable",
40             "To␣be␣multiplatform",
41             "Already␣have␣good␣base\n␣(about␣20␣internal␣and␣\n␣some␣external␣projects
       ),␣\n␣wish␣to␣get␣more..."
42         };
43
44         for (int i = 0; i < texts.size(); i++) {
45             softAssert.assertEquals(texts.get(i).getText(), expectedTexts[i]);
46         }
47
48         WebElement iframe = driver.findElement(By.id("frame"));
49         softAssert.assertTrue(iframe.isDisplayed());
50
51         driver.switchTo().frame(iframe);
52         WebElement frameButton = driver.findElement(By.id("frame-button"));
53         softAssert.assertTrue(frameButton.isDisplayed());
```

```
54
55          driver.switchTo().defaultContent();
56
57          List<WebElement> leftMenuItems = driver.findElements(By.cssSelector("ul.
        sidebar-menu.left␣>␣li"));
58          softAssert.assertEquals(leftMenuItems.size(), 5);
59
60          String[] expectedMenuTexts = {"Home", "Contact␣form", "Service", "Metals␣&␣
        Colors", "Elements␣packs"};
61          for (int i = 0; i < leftMenuItems.size(); i++) {
62              WebElement item = leftMenuItems.get(i);
63              softAssert.assertTrue(item.isDisplayed());
64              softAssert.assertEquals(item.getText(), expectedMenuTexts[i]);
65          }
66
67          softAssert.assertAll();
68      }
69  }
```

Класс содержит следующие основные компоненты:

- **Наследование от DriverSetup**: Класс `Task1Test` наследуется от класса `DriverSetup`, который выполняет предварительную настройку WebDriver и открывает веб-сайт.
- **@Test testTask1()**: Это тестовый метод, помеченный аннотацией `@Test`, который указывает, что это метод TestNG для выполнения тестов.
- **SoftAssert softAssert = new SoftAssert()**: Создание экземпляра `SoftAssert`, который позволяет собирать ошибки и не останавливать выполнение теста при первой неудаче.
- **Проверки (Assertions)**: Метод содержит ряд проверок с использованием `softAssert.assertEquals()` и `softAssert.assertTrue()` для проверки различных элементов веб-страницы:
  - Заголовок страницы (`softAssert.assertEquals(driver.getTitle(), "Home Page")`)).
  - Имя пользователя (`softAssert.assertEquals(driver.findElement(By.id("user-na "ROMAN IOVLEV")`)).
  - Элементы в секции заголовка (количество и текст элементов меню).
  - Изображения на главной странице (количество и отображение).
  - Тексты под иконками (количество и соответствие ожидаемым текстам).
  - Наличие и отображение iframe.
  - Наличие и отображение кнопки во iframe.
  - Элементы в левом меню (количество и текст элементов).
- **Переключение на iframe и обратно**: В коде происходит переключение на iframe для проверки содержимого внутри него, а затем возврат обратно к основному содержанию страницы.

- **softAssert.assertAll()**: Вызов этого метода в конце тестового метода позволяет убедиться, что все собранные ошибки будут выведены, и тест завершится с соответствующим статусом.

3.2.3 Класс Task2Test

Класс `Task2Test` является тестовым классом, который проверяет различные элементы и функциональности веб-сайта. Он использует библиотеку Selenium WebDriver для взаимодействия с веб-страницей и библиотеку TestNG для организации и выполнения тестов. Класс выполняет проверку заголовка страницы, имени пользователя, а также взаимодействует с элементами на странице "Different Elements"(чекбоксы, радиокнопки, выпадающий список) и проверяет логи.

Листинг 8. Класс Task2Test

```
1  public class Task2Test extends DriverSetup {
2      private static final By USER_NAME = By.id("user-name");
3
4      @Test
5      public void testBrowserTitle() {
6          assertEquals(driver.getTitle(), "Home␣Page", "Browser␣title␣should␣be␣'Home␣
       Page'");
7      }
8
9      @Test
10     public void testLogin() {
11         WebElement userNameElement = new WebDriverWait(driver, Duration.ofSeconds(10))
12         .until(ExpectedConditions.visibilityOfElementLocated(USER_NAME));
13
14         assertTrue(userNameElement.isDisplayed(), "Username␣should␣be␣displayed");
15         assertEquals(userNameElement.getText(), "ROMAN␣IOVLEV", "Username␣should␣be␣'
       ROMAN␣IOVLEV'");
16     }
17
18     @Test
19     public void testElements() {
20         WebElement serviceDropdown = driver.findElement(By.cssSelector("header␣.nav␣>␣
       li.dropdown"));
21         serviceDropdown.click();
22
23         WebElement differentElementsLink = driver.findElement(By.xpath("//a[text()='
       Different␣elements']"));
24         differentElementsLink.click();
25         assertEquals(driver.getTitle(), "Different␣Elements", "Заголовок␣страницы␣'
       Different␣Elements'␣неверный.");
26
27         List<String> checkboxesToSelect = Arrays.asList("Water", "Wind");
28         List<WebElement> checkboxes = new WebDriverWait(driver, Duration.ofSeconds(10)
       )
29         .until(ExpectedConditions.visibilityOfAllElementsLocatedBy(
30         By.cssSelector(".label-checkbox")
31         ));
32
33         for (WebElement checkbox : checkboxes) {
```

```
34              if (checkboxesToSelect.contains(checkbox.getText())) {
35                  if (!checkbox.findElement(By.tagName("input")).isSelected()) {
36                      checkbox.click();
37                  }
38                  assertTrue(checkbox.findElement(By.tagName("input")).isSelected());
39              }
40          }
41
42          String radioToSelect = "Selen";
43          List<WebElement> radios = new WebDriverWait(driver, Duration.ofSeconds(10))
44          .until(ExpectedConditions.visibilityOfAllElementsLocatedBy(
45          By.cssSelector(".label-radio")
46          ));
47
48          for (WebElement radio : radios) {
49              if (radio.getText().equals(radioToSelect)) {
50                  radio.click();
51                  assertTrue(radio.findElement(By.tagName("input")).isSelected());
52                  break;
53              }
54          }
55
56          String dropdownValueToSelect = "Yellow";
57          WebElement dropdown = new WebDriverWait(driver, Duration.ofSeconds(10))
58          .until(ExpectedConditions.elementToBeClickable(
59          By.cssSelector(".colors␣select")
60          ));
61
62          Select select = new Select(dropdown);
63          select.selectByVisibleText(dropdownValueToSelect);
64          assertEquals(select.getFirstSelectedOption().getText(), dropdownValueToSelect)
        ;
65
66          List<WebElement> logs = new WebDriverWait(driver, Duration.ofSeconds(10))
67          .until(ExpectedConditions.visibilityOfAllElementsLocatedBy(
68          By.cssSelector(".logs␣li")
69          ));
70          for (var elem : logs)
71          System.out.println(elem.getText());
72
73          List<String> expectedLogs = Arrays.asList(
74          "Water:␣condition␣changed␣to␣true",
75          "Wind:␣condition␣changed␣to␣true",
76          "metal:␣value␣changed␣to␣Selen",
77          "Colors:␣value␣changed␣to␣Yellow"
78          );
79
80          for (int i = 0; i < expectedLogs.size(); i++) {
81              String actualLog = logs.get((logs.size()-1) - i).getText().replaceAll("\\d
        {2}:\\d{2}:\\d{2}", "").trim();
82              assertTrue(actualLog.endsWith(expectedLogs.get(i)));
83          }
84      }
85  }
```

Класс содержит следующие основные компоненты:

- **Наследование от DriverSetup**: Класс `Task2Test` наследуется от класса `DriverSetup`, который выполняет предварительную настройку WebDriver и открывает веб-сайт.
- **Поле USER_NAME**: Приватное статическое поле `USER_NAME` типа `By`, содержащее локатор для элемента с именем пользователя.
- **@Test testBrowserTitle()**: Тестовый метод, который проверяет заголовок браузера на соответствие значению "Home Page". Использует `assertEquals` для проверки.
- **@Test testLogin()**: Тестовый метод, который проверяет, что имя пользователя отображается и соответствует ожидаемому значению "ROMAN IOVLEV". Использует явное ожидание (`WebDriverWait`) для проверки видимости элемента.
- **@Test testElements()**: Тестовый метод, который выполняет следующие шаги:
  - Открывает страницу "Different Elements"через меню "Service".
  - Выбирает чекбоксы "Water"и "Wind".
  - Выбирает радиокнопку "Selen".
  - Выбирает значение "Yellow"в выпадающем списке.
  - Проверяет логи на соответствие ожидаемым значениям.
- **Явные ожидания (WebDriverWait)**: Используются для ожидания появления и кликабельности элементов, что делает тесты более стабильными.
- **Проверка логов**: Код проверяет логи на соответствие ожидаемым значениям, учитывая порядок и формат записей.

3.2.4 Результаты работы №2

Результаты запуска тестов представлены на Рис. 5

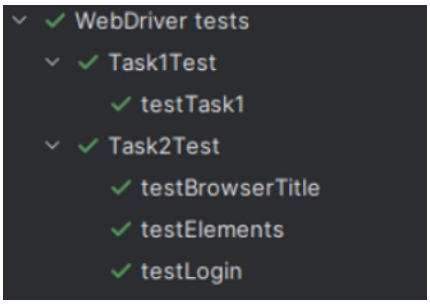

Рис. 5. Результаты выполнения тестов Selenium WebDriver

Все разработанные тесты для проверки веб-сайта успешно пройдены. Тесты охватывают широкий спектр элементов страницы, включая заголовки, элементы навигации, iframe, чекбоксы и выпадающие списки.

Заключение

В ходе выполнения лабораторной работы были изучены ключевые аспекты автоматизированного тестирования программного обеспечения. Работа охватила создание, написание и выполнение юнит-тестов для Java-библиотеки calculator.jar с использованием JUnit, а также организацию тестирования веб-страниц с помощью Selenium WebDriver и TestNG.

В результате тестирования библиотеки calculator.jar были обнаружены неточности в реализации некоторых методов, в частности, неверное вычисление тангенса и некорректная работа с числами с плавающей точкой при умножении. Эти ошибки могли бы остаться незамеченными при поверхностном ручном тестировании, что демонстрирует ценность автоматизированного подхода.

При тестировании веб-сайта с помощью Selenium WebDriver была подтверждена корректность отображения элементов и их функциональность. Успешное прохождение всех тестов показало, что сайт работает в соответствии с ожиданиями в тестовых сценариях.

Использованные инструменты автоматизированного тестирования продемонстрировали себя как эффективное средство для систематической проверки программного обеспечения на соответствие требованиям. Они позволяют выполнять тесты на больших наборах данных и многократно повторять одни и те же сценарии, что затруднительно при ручном тестировании.

Тем не менее, автоматизированное тестирование имеет свои ограничения. Оно фокусируется преимущественно на проверке функциональности и не способно в полной мере оценить удобство использования или обнаружить непредвиденные сценарии использования. Кроме того, сами автоматизированные тесты могут содержать ошибки.

Таким образом, можно заключить, что наиболее эффективный подход к тестированию сочетает в себе автоматизированные и ручные методы. Автоматизированное тестирование обеспечивает стабильность и повторяемость проверок, а ручное позволяет выявлять проблемы, связанные с удобством использования и нестандартными сценариями. Комплексное применение обоих подходов существенно повышает качество программного обеспечения.

Список литературы

[1] Что такое Selenium WebDriver? — Habr. [Электронный ресурс].
URL: https://habr.com/ru/articles/152971/ (дата обращения: 30.04.2025).

[2] Selenium IDE — Habr. [Электронный ресурс].
URL: https://habr.com/ru/articles/590607/ (дата обращения: 30.04.2025).