Material 2 – Descrevendo melhor o Junit e Selenium WebDriver

1. Estrutura de uma classe de testes do JUnit

Listagem 1: Exemplo de uma classe de testes básica

```
public class TestesTelaCadastroUsuario {

	@BeforeClass
	public static void setUpTest(){

	}

	@AfterClass
	public static void tearDownTest(){

	}

	@Test
	public void testaOpcoesDoSelectUF() {

	}

	@Test
	public void testaBotaoEnviar(){

	}

	//	...
}
```

- setUpTest(): método que é executado antes do teste. Normalmente, é nele que criamos uma nova instância do navegador com o Selenium WebDriver, abrimos conexões com bancos de dados etc. Esse método deve vir acompanhado da anotação @BeforeClass.
- tearDownTest(): método que, ao final de todos os testes, é executado para encerrar uma instância do navegador, fechar uma conexão com um banco de dados etc. Esse método deve vir acompanhado da anotação @AfterClass.
- testaAlgumaCoisa(): dentro de uma classe de testes, podemos criar n métodos que testam as diversas partes de um software, e é dentro desses métodos que inserimos todos os comandos para testá-lo. É muito interessante isolar cada caso de teste dentro de seu próprio método. Outra dica importante é seguir sempre um padrão de nomenclatura que defina de forma clara O QUÊ o método testa. Alguns exemplos: testaFocoDosCampos(), testaValidacaoDeData(), testaInclusao(), testaExclusao() etc. Esses métodos devem vir sempre acompanhados da anotação @Test, identificando o método como um método de teste.

2. Os métodos assert

Através dos métodos assert, analisamos **os dados de entrada e de saída** de um sistema. Podemos comparar valores retornados pelo sistema com valores esperados pelo caso de teste, tendo como retorno resultados positivos ou negativos. Por exemplo, como resultado do cálculo da soma entre os números "2" e "2" (valores de entrada), esperamos o valor "4" (valor esperado pelo caso de teste).

Imaginando que somos desenvolvedores, construiremos então um simples método em Java chamado calculaSoma(), esse que receberá como valores de entrada dois números inteiros e retornará como valor de saída a soma desses dois números. Veja:

Listagem 2: Método calculaSoma

```
public int calculaSoma(int a, int b){
        int soma = a + b;
        return soma;
}
```

Agora, através do método assertEquals() da biblioteca do JUnit, checaremos se o retorno do método calculaSoma(), passando como parâmetro os números "2" e "2", é igual a "4". Veja:

Listagem 3: Método para testar a função calculaSoma

```
@Test
public void testaSomaEntreDoisNumerosInteiros(){
}
        assertEquals(4,calculaSoma(2,2));
}
```

Pronto. Acabamos de criar um exemplo simples de teste unitário! Note que o método assertEquals() recebe dois parâmetros. O primeiro é o número "4", que representa o valor esperado como resultado do cálculo (ou resultado esperado pelo caso de teste). E o segundo parâmetro é o retorno do método calculaSoma(). Resumindo, ele não fará nada mais que comparar o retorno do métodocalculaSoma() com o número "4". Se os valores corresponderem, o JUnit mostrará o teste como “Passed”. Caso contrário, mostrará o teste como “Failed”.

Frisando, os métodos assert serão úteis nos nossos testes para fazer comparações entre strings, booleans, números inteiros, números fracionados, objetos Java, etc. Como o nosso foco são os testes funcionais, seguiremos para o Selenium WebDriver, assim poderemos entender como é feita a integração entre as duas ferramentas.

Selenium WebDriver

A API Selenium WebDriver tem como principal objetivo **automatizar ações do navegador**, tais como submits de formulários, seleções em menus dropdown, digitação em campos texto, varredura de dados em elementos, HTML etc. Para entender melhor seus conceitos, a seguir, detalharemos algumas interfaces e outras classes principais que compõem o Selenium WebDriver.

1. A interface WebDriver

Podemos dizer que a interface WebDriver é a mais importante de um projeto Selenium WebDriver. Nela, temos os métodos que controlam o navegador, selecionam elementos de páginas HTML, etc. Veja abaixo um exemplo em que declaramos um objeto do tipo WebDriver:

Listagem 4: Declarando um objeto WebDriver

```
private static WebDriver driver;
```

Instaciado o objeto, podemos chamar seus métodos. Os dois métodos mais comuns e importantes da interface WebDriver são o get() e o findElement(). Veja esses e outros métodos comentados logo abaixo:

Listagem 5: Métodos do WebDriver

```
// Fecha a janela corrente.
driver.close();

// Encontra o primeiro elemento de uma tela HTML através de um dado argumento.
driver.findElement(By by);

// Encontra todos os elementos de uma tela HTML através de um dado argumento.
driver.findElements(By by);

// Abre uma nova URL no navegador.
driver.get();

// Retorna uma string que contém a URL aberta pelo navegador.
driver.getCurrentUrl();

// Retorna o código fonte da última página aberta pelo navegador.
driver.getPageSource();

// Retorna o título da página aberta pelo navegador.
driver.getTitle();

// Retorna um identificador da janela em questão.
driver.getWindowHandle();

// Retorna identificadores que podem ser utilizados para movimentação entre
janelas.
driver.getWindowHandles();

// Envia comandos futuros para uma janela (ou frame) diferente.
driver.switchTo();

// Permite gerenciar cookies do navegador, logs, timeouts etc.
driver.manage();

// Abstração que permite acessar o histórico e navegar para uma determinada URL.
driver.navigate();
```

```
// Fecha a instância do Selenium WebDriver e todas os navegadores associados.
driver.quit();
```

2. A interface WebElement

Como o próprio nome já sugere, o tipo WebElement serve para armazenar um elemento web, ou seja, um componente HTML de uma tela. Por exemplo, podemos criar um objeto do tipo WebElement e armazenar nele um input específico da tela, para que possamos posteriormente preenchê-lo com algum conteúdo textual. Veja o exemplo:

Listagem 6: Criando um WebElement a partir de um elemento da tela

```
WebElement elemento = driver.findElement(By.id("txt-nome"));
```

No exemplo acima, chamamos o método findElement() do objeto driver (proveniente da interface WebDriver), e passamos como parâmetro de busca um id específico (txt-nome). Dessa forma, o Selenium WebDriver é capaz de varrer toda a estrutura HTML da página em teste até encontrar o elemento cujo id seja igual a txt-nome, armazenando-o na variável elemento.

O método findElements() trabalha de forma semelhante. Sua única particularidade é o seu retorno: uma lista de objetos do tipo WebElement. Com esse método, é possível armazenar toda a estrutura de containers HTML que possuem sub-elementos aninhados. Alguns exemplos: <ul> (listas não ordenadas formadas por várias linhas <li>), <tables> (tabelas formadas por linhas <tr> e células <td>), <form> (formulários formados por vários <input> de diferentes tipos) etc. Vejamos abaixo o exemplo de uma tabela HTML:

Listagem 7: Exemplo de tabela HTML

```
<table>
	<tr>
		<th>Nome</th>
		<th>Idade</th>
	</tr>
	<tr>
		<td>Márcio</td>
		<td>42</td>
	</tr>
	<tr>
		<td>Fabiane</td>
		<td>40</td>
	</tr>
	<tr>
		<td>Aline</td>
		<td>12</td>
	</tr>
	<tr>
		<td>Felipe</td>
		<td>10</td>
	</tr>
	<tr>
		<td>Maria</td>
		<td>2</td>
	</tr>
</table>
```

O próximo passo é criar com o Selenium WebDriver uma lista que receberá todas as células da

tabela acima (elementos <td>). Com a lista criada, será possível fazer diversas iterações para manipular os dados dos nós. O exemplo abaixo ilustra a criação da lista e de um loop que imprime os valores contidos em cada célula da tabela:

Listagem 8: Acessando as células da tabela HTML

```
// Declarando um objeto do tipo WebDriver.
private static WebDriver driver;

// Cria lista de objetos do tipo WebElement que recebe as células da tabela
acima.
List<WebElement> celulas = driver.findElements(By.tagName("td"));

// Loop que imprime o valor de cada célula.
for(WebElement c : celulas){
        System.out.println(c.getText());
}
```

Existem vários outros métodos na interface WebElement que nos permitem "manusear" o objeto e seus atributos. Vejamos abaixo os mais utilizados:

Listagem 9: Principais métodos da interface WebElement

```
// Limpa o conteúdo do elemento.
elemento.clear();

// Clica e, consequentemente, muda o foco da tela para o elemento.
elemento.click();

// Retorna o valor do atributo passado como parâmetro de dado elemento.
elemento.getAttribute();

// Retorna o valor de uma propriedade CSS passada como parâmetro de dado
elemento.
elemento.getCssValue();

// Retorna o ponto da tela do canto superior esquerdo de dado elemento web.
elemento.getLocation();

// Retorna a dimensão do elemento (largura e altura).
elemento.getSize();

// Retorna o nome da tag HTML de dado elemento.
elemento.getTagName();

// Retorna o texto presente dentro do elemento.
elemento.getText();

// Retorna verdadeiro ou falso se dado elemento estiver visível ou não na tela.
elemento.isDisplayed();

// Retorna verdadeiro ou falso se dado elemento estiver ativo ou não na tela.
elemento.isEnabled();

// Retorna verdadeiro ou falso se dado elemento estiver selecionado ou não na
tela.
elemento.isSelected();

// Insere caracteres num determinado elemento da tela.
elemento.sendKeys();

// Envia dados para o servidor se o elemento em questão for um formulário.
```

```
elemento.submit();
```

3. A classe Select

A classe Select serve para dar suporte às tags <select> do código HTML. Contém métodos que facilitam o trabalho com esse tipo de elemento. Vejamos a seguir um exemplo de como utilizar essa classe:

Listagem 10: Utilizando a classe Select

```
// Declarando um objeto do tipo WebDriver.
private static WebDriver driver;

// Instancia um novo objeto do tipo Select, passando como parâmetro
// o primeiro elemento da tela cuja tag seja igual a "select".
Select dropdown = new Select(driver.findElement(By.tagName("select")));
```

Criamos um novo objeto chamado dropdown, proveniente da classe Select, e atribuímos a ele um elemento da tela de tag igual a <select>. Vejamos os principais métodos que podemos chamar em objetos do tipo Select:

Listagem 11: Principais métodos da classe Select

```
a// Remove todas as seleções.
dropdown.deselectAll();

// Remove seleção de uma opção específica através de um dado índice.
dropdown.deselectByIndex(int index);

// Remove seleção de todas as opções que tiverem seus atributos "value"
especificado.
dropdown.deselectByValue(String value)

// Remove seleção de todas as opções que forem iguais ao argumento especificado.
dropdown.deselectByVisibleText(String text);

// Retorna objeto do tipo WebElement contendo a primeira tag "<option>"
selecionada.
dropdown.getFirstSelectedOption();

// Retorna uma lista do tipo WebElement com todas as tags "<option>"
selecionadas.
dropdown.getAllSelectedOptions();

// Retorna uma lista do tipo WebElement com todas as tags "<option>".
dropdown.getOptions();

// Retorna verdeiro ou falso, dependendo do valor do atributo "multiple".
dropdown.isMultiple();

// Seleciona uma opção específica do menu através de um dado índice.
dropdown.selectByIndex(int index);

// Seleciona as opções que tiverem seus atributos "value" iguais ao
especificado.
dropdown.selectByValue(String value);

// Seleciona as opções que forem iguais aos argumentos especificados.
dropdown.selectByVisibleText(String text);
```

Agora que entendemos os princípios básicos das duas ferramentas, revisando, para criar a automação dos testes funcionais de um software com JUnit e Selenium WebDriver, devemos lembrar que teremos que executar as seguintes etapas:

- Definir a estrutura da classe de testes conforme formato do JUnit (criar a classe de testes, os métodos setUpTest(), teardDownTest() e os métodos de teste propriamente ditos).
- Codificar as ações do navegador utilizando a biblioteca do Selenium WebDriver (procurar por elementos na tela, clicar em elementos, inserir dados em elementos etc).
- Codificar os testes dos dados de entrada e saída através de comandos assert do JUnit.

Veremos agora um exemplo prático de implementação de uma classe de testes que:

- Abre a página inicial da DevMedia.
- Testa se o título da página inicial é igual a DevMedia - Cursos, Tutoriais e Vídeos para Desenvolvedores.
- Popula os campos do formulário de login da DevMedia.
- Submete o formulário de login.

Listagem 12: Exemplo prático de classe de teste

```
// Importa as bibliotecas do JUnit, Selenium WebDriver, List etc.
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

// Classe JUnit que contém os métodos de teste.
public class TestaDevmedia {

	// Declarando um objeto do tipo WebDriver, utilizado pelo Selenium
WebDriver.
	private static WebDriver driver;

	// Método que inicia tudo que for necessário para o teste
	// Cria uma instância do navegador e abre a página inicial da DevMedia.
	@BeforeClass
	public static void setUpTest(){
		driver = new FirefoxDriver();
		driver.get("http://www.devmedia.com.br");
	}

	// Método que finaliza o teste, fechando a instância do WebDriver.
	@AfterClass
	public static void tearDownTest(){
		driver.quit();
	}

	// Testa título "DevMedia - Cursos, Tutoriais e Vídeos para
Desenvolvedores".
	@Test
	public void testaTituloDaPagina(){
		assertEquals("DevMedia - Cursos, Tutoriais e Vídeos para
Desenvolvedores", driver.getTitle());
	}
```

```
        // Método que testa o login no site DevMedia.
        @Test
        public void testaLoginDevMedia() {

        // Instancia um novo objeto do tipo "WebElement", e passa como parâmetro
        // um elemento da tela cujo valor do atributo "name" seja igual a
"usuario".
        WebElement element = driver.findElement(By.name("usuario"));

        // Insere dados no elemento "usuario".
        element.sendKeys("user@devmedia.com.br");

        // Atribui ao objeto “element” o elemento de atributo "name" igual a
"senha".
        element = driver.findElement(By.name("senha"));

        // Insere dados no elemento "senha".
        element.sendKeys("123456");

        // Clica no botão "OK" e submete os dados para concluir o login.
                driver.findElement(By.id("imglogar")).click();
        }
}
```

Este tutorial contém os princípios básicos e essenciais para bom entendimento do funcionamento das ferramentas. Ambas são muito completas e possuem uma infinidade de características que podem ser úteis na implementação de seus testes. Cabe a cada testador saber identificar quais os pontos críticos que mereçam atenção para testes automatizados, bem como se aprofundar mais nas tecnologias.

Para informações mais avançadas, consulte as documentações oficiais.