



Workshop Selenium WebDriver



contato@qualister.com.br



(48) 3285-5615



twitter.com/qualister



facebook.com/qualister



linkedin.com/company/qualister



Instrutor

Elias Nogueira

Consultor de Teste e Qualidade de software na Qualister

Professor de Pós Graduação na Unisinos/RS e Uniasselvi/SC



[eliasnogueira](#)



[qualister.com.br](#)



[br.linkedin.com/in/eliasnogueira](#)



[youtube.com/user/qualistervideos](#)



- Fundada em 2007
- Mais de 1.000 clientes em todo o Brasil
- Mais de 50 cursos sobre teste de software
- Mais de 3.000 alunos formados
- Áreas de atuação:
 - Consultoria na área de teste qualidade de software
 - Cursos
 - Revenda de ferramentas

Mais de 1.000 clientes





Parcerias internacionais





Selenium WebDriver

1. O que é o Selenium
2. Passos iniciais
3. Mão na massa
 1. Identificando Elementos
 2. Interagindo com elementos
 3. Trabalhando com Ajax (esperas)
 4. Simulando ações de usuário



O que é o Selenium

Neste tópico descobriremos o que é o Selenium e quais ferramentas pertencem a este framework

- É um framework que possui um conjunto de ferramentas:

Selenium IDE



Selenium IDE é um plugin para o Firefox onde podemos gravar e executar as ações no browser. Também exporta o código gravado para o Selenium Remote Control

Selenium WebDriver



Executa ações de usuário nativamente nos browsers web de forma local ou remota

Selenium Remote Control



É um cliente/servidor que nos possibilita executar testes em browsers de forma local ou em outro computador usando uma linguagem de programação

Selenium Grid



Habilita o Selenium Remote Control a executar estes em diferentes máquinas e browsers ao mesmo tempo



Selenium WebDriver

- É uma API (Application Programming Interface)
- Executa ações em browsers web simulando um usuário
- Como API nós temos que programar/desenvolver scripts de teste
- Pode ser desenvolvido na seguintes linguagens nativamente





Pontos fortes

- Utilização de diversas linguagens para execução de testes
- Execução nativa no Firefox
- Execução através de driver no:
 - Internet Explorer
 - Google Chrome
 - Opera
 - Headless Browser (sem interface gráfica)
- Simulação de ações do usuário



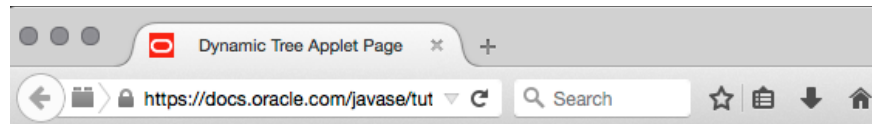
Selenium WebDriver

- No Workshop...
 - Usaremos a API em Java
 - Usaremos o Eclipse IDE para desenvolver os testes e Java
 - Usaremos o Junit para suporte aos testes

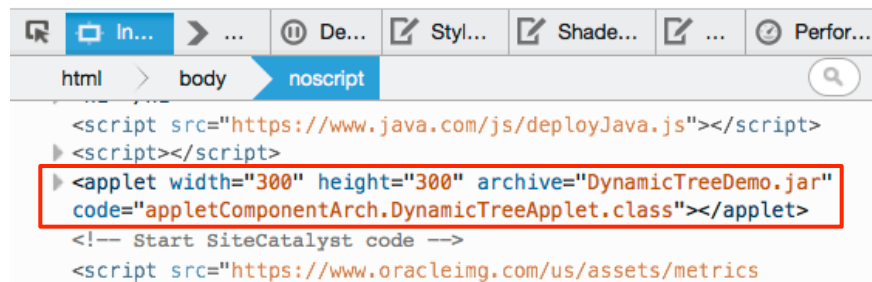
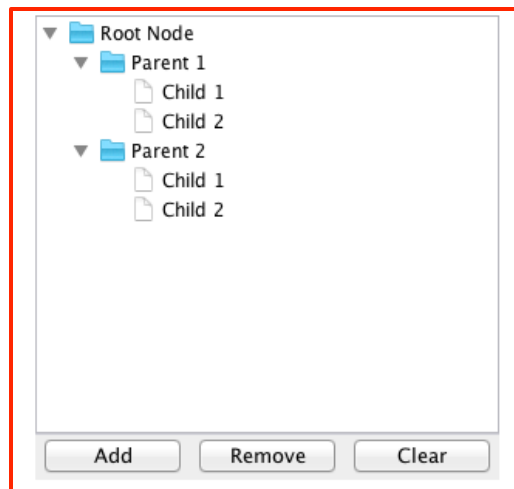


O que não é possível automatizar

- Applets



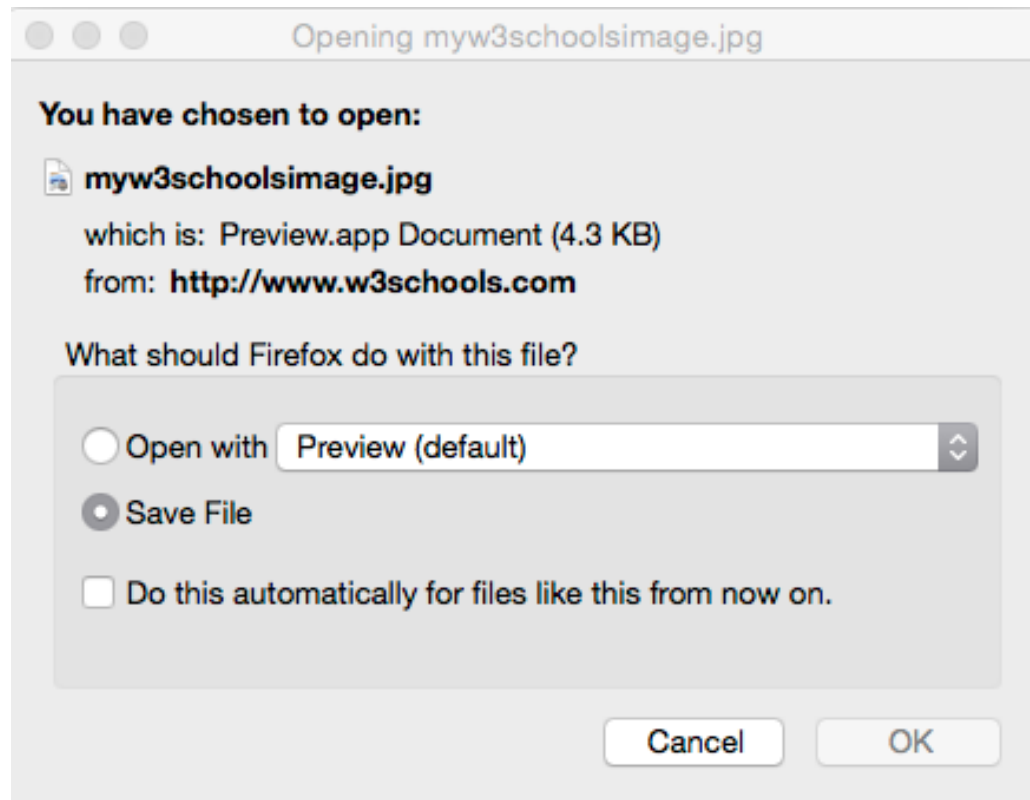
Dynamic Tree Applet Demo





O que não é possível automatizar

- Janela de Downloads



O que não é possível automatizar

- Flash/Flex

The screenshot displays the Apache Flex Tour De Flex Component Explorer 1.2 interface. The browser address bar shows `flex.apache.org/tourdeflex/`. The page title is "Apache Flex® Tour De Flex Component Explorer 1.2" with "308 examples" listed. The left sidebar shows a tree view of components, with "Text Controls" selected. The main content area shows a "TextInput Control Example" with a text input field containing "Hello World!" and a "Copy Text" button. A context menu is open over the text input, showing options: "Print...", "Settings...", "Global Settings...", and "About Adobe Flash Player 17.0.0.169...". Below the example, the "TextInputExample.mxml" code is shown, starting with `<?xml version="1.0"?>` and `<!--`. At the bottom, the HTML code for the page is displayed, with the following line highlighted in a red box:

```
<object id="TourDeFlex" width="100%" align="middle" height="100%" type="application/x-shockwave-flash" name="TourDeFlex" data="explorer.swf">
```

The bottom of the screenshot shows a copyright notice: "Copyright © 2014 The Apache Software Foundation, Licensed under the Apache License, Version 2.0. Apache Flex is trademark of The Apache Software Foundation." and a navigation bar with tabs for "Cons...", "HTML", "CSS", "Script", "DOM", and "Net".



Conhecimentos Básicos para Automação Web

Neste tópico aprenderemos quais são os pontos essenciais para a automação web



Conhecimentos Básicos

Elementos HTML

São tags que utilizamos em arquivos HTML que se transformam em elementos DOM. São características de elementos HTML:

- Nome do elementos são escritos com marcadores

```
<title>Workshop Selenium</title>
```

- Cada elemento possui atributos e valores

```
<button id="salvar" class="light">Salvar registro</button>
```

```
<input type="checkbox" checked />
```

- Cada elemento pode possuir outros elementos

```
<select id="estados">
```

```
  <option value="PR">Paraná</option>
```

```
  <option value="SC">Santa Catarina</option>
```

```
  <option value="RS">Rio Grande do Sul</option>
```

```
</select>
```




Conhecimentos Básicos

Ações Executadas no Navegador

- Abrir um browser web
- Navegar em uma página
- Ler o título de uma página
- Ler uma URL
- Pegar um texto da página
- Clicar em links
- Preencher campos
- Preencher formulários
- Clicar em botões



Conhecimentos Básicos

Ações Executadas no Navegador

- Abrir um browser web
- Navegar em uma página

—————→ **Navegação**

- Ler o título de uma página
- Ler uma URL
- Pegar um texto da página

—————→ **Interrogação**

- Clicar em links
- Preencher campos
- Preencher formulários
- Clicar em botões

—————→ **Manipulação**



Conhecimentos Básicos

Existe mais uma ação que utilizamos implicitamente (sem se dar conta) quando testamos uma página web

Sincronização

A sincronização é a capacidade de espera por alguma ação, como por exemplo o caso clássico de “Ajax Loading” ou um elemento ser apresentado na tela

Carregar dados...

Selecione o tipo de pessoa:

☐ Pessoa Física

☐ Pessoa Jurídica



Conhecimentos Básicos

Navegação

Ações sobre a página

Interrogação

Obtenção de dados e informações

Manipulação

Ações em elementos, cliques e preenchimentos

Sincronização

Esperas



Criando a estrutura inicial

Neste tópico aprenderemos como criar a estrutura inicial do projeto



Estrutura Inicial

Projeto Java

- Criação do projeto
- Associar as bibliotecas do Selenium ao classpath

Desenvolvimento dos scripts

- Criação de pacotes
- Criação de classes como JUnit Test Case



Navegação

Neste tópico aprenderemos como abrir o Firefox e como seguir com os próximos comandos do Selenium



Abrindo o Firefox

Sempre que usamos o WebDriver devemos informar qual browser web iremos utilizar

O Firefox é suportado nativamente no WebDriver

Outros browser não são suportados nativamente (IE, Google Chrome, Opera), sendo necessário configurações adicionais



Abrindo o Firefox

A classe WebDriver é o ponto de partida inicial para:

- Abrir o browser web
- Navegar entre páginas do browser
- Interrogar elementos
- Manipular elementos

```
@Test  
public void test() {  
    WebDriver driver = new FirefoxDriver();  
}
```



Exercício

Objetivo: abrir o browser Firefox sem acessar uma página

Comandos

```
WebDriver driver = new FirefoxDriver();
```

Resultado Esperado

Firefox aberto sem acessar uma página

Observações

Nenhuma



Abrindo o Firefox

Existe uma série de métodos para navegar em uma página

- driver

- `get("url")` → Acessa uma página
- `navigate()`
 - `to("URL")` → Acessa uma página
 - `to(java.net.URL)` → Acessa uma página
 - `back()` → Volta no histórico da página
 - `forward()` → Avança no histórico da página
 - `refresh()` → Atualiza a pagina



Abrindo o Firefox

```
@Test
public void test() {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://qualister.com.br");
    driver.navigate().to("http://www.qualister.com.br/blog");
    driver.navigate().back();
    driver.navigate().forward();
    driver.navigate().refresh();
}
```



Exercício

Objetivo: abrir o browser na página inicial da aplicação
<http://quickloja.qualister.info>

Comandos

```
WebDriver driver = new FirefoxDriver();  
driver.get("<url>");
```

Resultado Esperado

Firefox aberto na página do QuickLoja

Observações

Nenhuma



Fechando o Firefox

A classe WebDriver (o nosso driver) possui dois métodos bem semelhantes: *quit()* e *close()*

- **driver.close()**
Fecha a janela atual. Se for a última janela, fecha também o browser
- **driver.quit()**
Fecha o browser, mesmo que existam diversas janelas abertas



Fechando o Firefox

```
@Test
public void test() {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://qualister.com.br");
    driver.navigate().to("http://www.qualister.com.br/blog");
    driver.navigate().back();
    driver.navigate().forward();
    driver.navigate().refresh();

    driver.quit();
}
```

Fecha o browser (Firefox)



Inspeccionando Elementos

Neste tópico aprenderemos inspecionar os elementos e descobrir os seus atributos de uma forma mais rápida

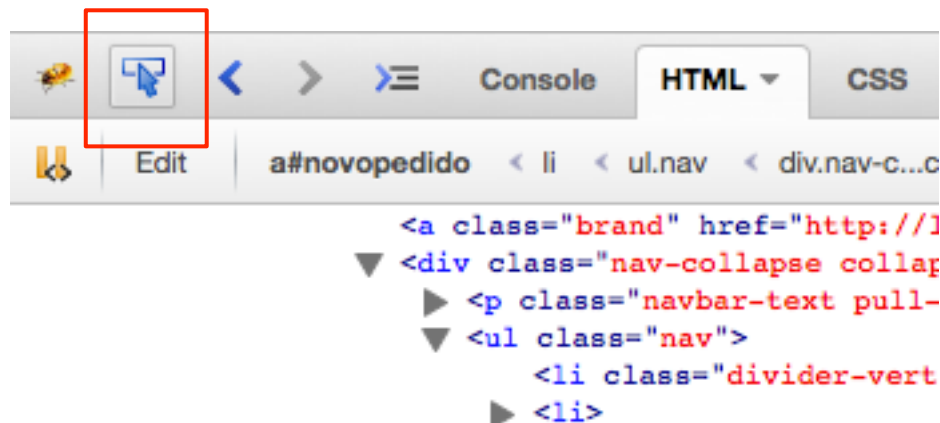


Inspecionando Elementos

Os browsers mais modernos possuem a funcionalidade de inspecionar elementos HTML, que é a prática de descobrir o código-fonte especificamente de um elemento.

Nós usaremos o Firebug, que pode ser extensível através de plugins para nos ajudar a localizar alguns elementos

No Firebug clicamos no botão de inspeção e movemos o mouse até o elemento





Inspecionando Elementos

Elemento selecionado



Produto

Valor

Quantidade

+ Adicionar

The screenshot shows the browser's developer tools with the 'HTML' tab selected. The breadcrumb trail indicates the path: `input#produtonome` < `div.span7` < `div.span9...ll-small` < `div.row-fluid` < `fieldset` < `div#tab3....e.active` < `div.tab-content` < `div.tabbable` < `for`. The expanded HTML tree shows the following structure:

```
<div class="span9 well well-small">
  <div class="span7">
    <label>Produto</label>
    <input id="produtonome" type="text" autocomplete="off" placeholder="" style="width: 90%">
    <input id="produtoid" type="hidden" value="">
    <input id="produtovalue" type="hidden" value="">
  </div>
</div>
```

The `<input id="produtonome" type="text" autocomplete="off" placeholder="" style="width: 90%">` line is highlighted in blue. An arrow points from the text 'Código-fonte HTML do elemento selecionado' to this line.

Código-fonte HTML do elemento selecionado



Interrogação

Neste tópico aprenderemos como localizar elementos de diversas formas



Localização de Elementos

Para encontra elementos utilizamos o objeto do WebDriver:

- driver
 - `findElement(By.<estrategia>)`

By.id Localiza o elemento pelo atributo *id*

By.name Localiza o elemento pelo atributo *name*

By.tagName Localiza o elemento por uma tag

By.linkText Localiza o elemento pelo nome do link

By.partialLinkText Localiza o elemento pelo nome parcial do link

By.cssSelector Localiza o elemento por CSS Selector

By.className Localiza o elemento pelo atributo *class*

By.xpath Localiza o elemento por xpath



Localização de Elementos

Localizando por ID e Name

São os primeiros que devemos tentar.

Geralmente um elemento possui um ID único ou um name, referente seu atributo de mesmo nome

- ID `<input type="text" id="usuariologin">`
- Name `<input type="password" name="usuariosenha">`

- Exemplo

```
driver.findElement(By.id("valor do atributo ID"));
```

```
driver.findElement(By.name("valor do atributo name"));
```



Exercício

Objetivo: Localizar os elementos de **login** e **senha** da página inicial do QuickLoja

Comandos

```
driver.findElement(By.id("<id>"));  
driver.findElement(By.id("<name>"));
```

Resultado Esperado

Localização dos elementos (ainda não há interação)

Observações

Nenhuma



Localização de Elementos

Localizando por CSS Selector

É uma forma de localização baseado nos principais seletores CSS

É a terceira forma que iremos adotar (se não conseguirmos encontrar um elemento por ID ou Name)

Seletor	Descrição
elemento	Localiza o elemento. Ex: div
#id	Localiza o elemento através do seu id. Ex: #dataInicio
.classe	Localiza o elemento através da(s) classe na tag class. Ex: .divNeto
elemento[atributo='valor']	Localiza um elemento pelo valor de um atributo. Ex: p[lang="us"]
elemento > elemento	Localiza o próximo elemento baseado no anterior. Ex: div > a



Localização de Elementos

```
// elemento
driver.findElement(By.cssSelector("h1"));

// ID
driver.findElement(By.cssSelector("#email"));

// classe
driver.findElement(By.cssSelector(".error"));

// valor de um atributo
driver.findElement(By.cssSelector("input[value='Queijo']"));

// baseado no elemento anterior
driver.findElement(By.cssSelector("#divConteudo > input"));
```




Localização de Elementos

Localizando por CSS Selector

Quando um elemento possui somente o atributo **class**, ou só por ele que podemos encontrar um elemento único pegamos o valor deste atributo, onde:

- Todo o valor deve ser colocado no *cssSelector*
- Cada espaço em branco deve ser substituído por um “.”

Exemplo:

- Elemento

```
<button class="btn btn-medium btn-primary" type="submit">Entrar</button>
```

- Código

```
driver.findElement(By.cssSelector(".btn.btn-medium.btn-primary"));
```



Exercício

Objetivo: Localizar o elemento do botão **Entrar** na tela inicial do QuickLoja

Comandos

```
driver.findElement(By.cssSelector("seletor"));
```

Resultado Esperado

Localização dos elementos (ainda não há interação)

Observações

Nenhuma



Manipulação

Neste tópico aprenderemos como interagir com os elementos web



Manipulação de Elementos

Manipulando o elemento

A manipulação ocorre pelo objeto WebElement ou como sequência da localização do elemento

Inicialmente iremos aprender a interagir como sequência de localização.

- `driver.findElement(By.<estrategia>)`
 - `click()` → Clica em um elemento
 - `clear()` → Limpa o texto de um elemento
 - `sendKeys("texto")` → Preenche um elemento com texto
 - `getText()` → Pega o texto de um elemento



Manipulação de Elementos

```
// Clicar
driver.findElement(By.id("ID")).click();

// Limpar campo
driver.findElement(By.name("name")).clear();

// Digitar texto
driver.findElement(By.cssSelector("css")).sendKeys("texto");

// Pegar texto
driver.findElement(By.name("name")).getText();
```



Exercício

Objetivo: Preencher o campo **Login** com “*admin*” e clicar no botão **Entrar**

Comandos

```
driver.findElement(By.<estrategia>).sendKeys("<texto>");  
driver.findElement(By.<estrategia>).click();
```

Resultado Esperado

Apresentar a mensagem “*Usuário ou senha incorretos*”

Observações

Nenhuma



Validação de resultados

Neste tópico aprenderemos como validar os resultados esperados de um script de teste



Manipulação de Elementos

Manipulando o elemento

Com a ajuda do JUnit poderemos validar o resultado esperado de um script de teste através dos seus métodos de **assert**

Eles são métodos de validação de diversos tipos onde, inicialmente, usaremos o método **assertEquals**

assertEquals(resultado esperado, resultado obtido);

- Resultado Esperado: geralmente um texto fixo sendo o resultado esperado da ação
- Resultado Obtido: geralmente o retorno (texto) do browser, sendo um texto de um elemento



Manipulação de Elementos

Como Resultado Obtido localizamos o elemento que contém o texto, e utilizamos o *getText()* para retornar o texto deste elemento

resultado esperado, sendo um texto fixo



```
assertEquals("Usuário ou senha incorretos",  
            driver.findElement(By.id("id")).getText());
```

resultado obtido, trazendo o texto do elemento





Exercício

Objetivo: Preencher o campo **Login** com “*admin*” e clicar no botão **Entrar**, na sequência validar o resultado

Comandos

```
driver.findElement(By.<estrategia>).getText();  
assertEquals(resultado esperado, resultado obtido);
```

Resultado Esperado

Validar que a mensagem “*Usuário ou senha incorretos*” está correta

Observações

Nenhuma



Exercício

Objetivo: Criar um novo script chamado “Login”. Preencher o campo **Login** com “selenium” e campo *Senha* com “teste” e clicar no botão **Entrar**

Comandos

```
driver.findElement(By.<estrategia>).sendKeys();  
driver.findElement(By.<estrategia>).click();
```

Resultado Esperado

Acessar a página inicial do QuickLoja após o login com sucesso

Observações

Para que a janela inicie maximizada, inserir o seguinte comando logo após a abertura do Firefox

```
driver.manage().window().maximize();
```



Trabalhando com Combos

Neste tópico aprenderemos como selecionar valores em elementos tipo Combo Box (select)



Trabalhando com Combos

Utilizamos a classe **Select** (org.openqa.selenium.support.ui.Select) para materializar um elemento em uma combo

Declarando um WebElement

```
WebElement elementoCombo = driver.findElement(By.id("combo"));
Select combo1 = new Select(elementoCombo);
```

Utilizando a sequência após a localização

```
Select combo2 = new Select(driver.findElement(By.id("combo")));
```



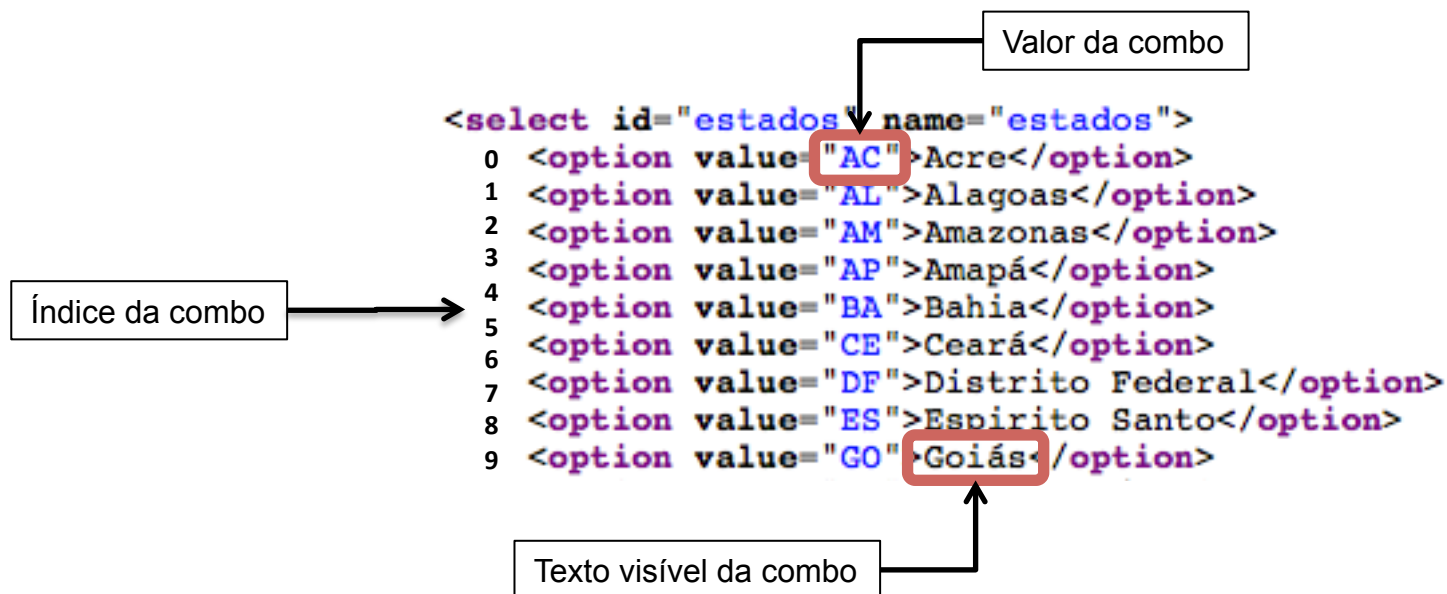
Trabalhando com Combos

Após materializar a combo habilitamos uma série de métodos para selecionar o valor de uma combo

selectByIndex(index) Seleciona um item pelo índice

selectByValue(value) Seleciona um item pelo valor

selectByVisibleText(text) Seleciona um item pelo texto visível





Trabalhando com Combos

```
Select combo2 = new Select(driver.findElement(By.id("combo")));  
combo2.selectByIndex(9);  
combo2.selectByValue("GO");  
combo2.deselectByVisibleText("Goiás");
```



Localização de Elementos

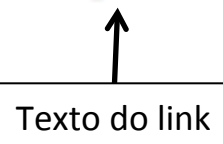
Localizando por linkText

É a forma de localizar um elemento que é um link.

Utilizamos o texto do link como parâmetro para localização

Exemplo do elemento

```
<a id="novopedido" href="/pedido/novopedido">Novo pedido</a>
```



Texto do link

Exemplo do código

```
driver.findElement(By.linkText("Novo pedido"));
```




Exercício

Objetivo: Ainda no script de Login, executar os seguintes passos e automatizá-los:

- Clicar em Movimentações
- Selecionar o item “Saída” na combo Tipo
- Inserir o valor “200,00” no campo Valor
- Inserir o texto “Saída de materiais” no campo “Itens/Observações”
- Clicar no botão Gravar
- Validar o texto “Sucesso ao inserir a movimentação”

Comandos

```
driver.findElement(By.<estrategia>).sendKeys();  
driver.findElement(By.<estrategia>).click();  
Select nome = new Select(<elemento>);  
nome.selectByVisibleText("texto da combo");
```

Resultado Esperado

Mostrar mensagem de sucesso na tela de movimentações, bem como movimentação listada.



Sincronização (Esperas)

Neste tópico aprenderemos como esperar por elementos antes de interagir e seguir o script de automação



Sincronização (Esperas)

Há duas formas de esperas: Implícitas e Explícitas

Aprenderemos apenas a forma Explícita, por:

- Não trazer resultados falso-positivo
- Melhorar a legibilidade do script
- Melhorar a manutenção do script

O ponto inicial da espera Explícita é a classe **WebDriverWait**, que traz uma espera “global” para o script de teste

```
WebDriverWait wait = new WebDriverWait(driver, 30);
```

Necessita do driver/browser atual

Tempo de espera



Sincronização (Esperas)

O objeto do *WebDriverWait* nos traz métodos para trabalharmos com a espera:

- **WebDriverWait wait....**

- `until(true)` → Aguada até determinada condição
- `pollingEvery(duracao, tempo)` → Determina o intervalo da contagem de tempo até a próxima tentativa
- `withMessage(mensagem)` → Adiciona uma mensagem quando ocorrer o timeout
- `Ignoring(exception)` → Ignora alguma *Exception* que possa ocorrer durante a espera
- `withTimeout(tempo)` → Sobrescreve o timeout



Sincronização (Esperas)

O método mais comum de utilização é o **until**

Ligamos o until a uma **ExpectedConditions** que é uma classe estática que possui uma série de métodos prontos de espera

Cada método pode ter um parâmetro diferente de um **WebElement**

```
wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id("pwd"))));
```

Um dos métodos de espera

Elemento que será aguardado





Sincronização (Esperas)

Utilizamos, na grande maioria das vezes, uma das duas **ExpectedConditions** abaixo:

- **presenceOfElementLocated**

Quando o elemento não existe no HTML, e logo uma ação é executada ele é inserido

- **visibilityOfElementLocated**

Quando o elemento existe na página, mas está invisível

```
<td>Camisa Regata</td>  
<td>R$ 50,00</td>  
<td>1</td>  
<td>R$ 50</td>
```

```
<input type="hidden" value="502" name="itemprodutoid1">  
<input type="hidden" value="1" name="itemprodutoquantidade1">  
<input type="hidden" value="50" rev="subtotal">
```



Sincronização (Esperas)

```
// Criação do objeto Wait e espera global
WebDriverWait wait = new WebDriverWait(driver, 10);

// Espera até a condição esperada
wait.until(
    ExpectedConditions.presenceOfElementLocated(
        By.cssSelector("css")));

// Segue executando o código caso a espera tenha retornado verdadeiro (terminou)
driver.findElement(By.name("name"));
```



Exercício

Objetivo: Crie um novo script chamado Pedido, executar os seguintes passos e automatizá-los (lembrado que o login deve estar presente):

- Clicar em Novo pedido
- Clicar na aba “Itens do pedido”
- Digitar “Camisa” no campo Produto
- Selecionar o item “Camisa T-Shirt”
- Preencher o campo Quantidade com “1”
- Clicar no botão Adicionar

Comandos

```
WebDriverWait wait = new WebDriverWait(driver, TEMPO);  
Wait.until(ExpectedConditions.<espera>);
```

Resultado Esperado

Selecionar com sucesso o item “Camisa T-Shirt” e adicioná-la no pedido



Localização por posição e métodos

Neste tópico aprenderemos como localizar elementos usando XPATH através de posicionamento ou funções



Localização de Elementos

Localizando por XPath

Xpath é uma linguagem de consulta a arquivos XML (utilizando tags atributos e valores). Como a estrutura de um arquivo HTML é semelhante ao XML, podemos utilizar esta forma de localização.

Seletor	Descrição
<code>//elemento</code>	Localiza o elemento. Ex: div
<code>//elemento[@atributo]</code>	Localiza o elemento que tenha o atributo descrito
<code>//elemento[@atributo="valor"]</code>	Localiza o elemento que tenha o seguinte valor do atributo descrito
<code>//elemento1/elemento2</code>	Localiza o elemento 2 através do elemento 1 (é filho)
<code>//elemento[numero]</code>	Localiza o elemento pelo seu numero/posição



Localização de Elementos

Exemplo de aplicação de posicionamento

Sempre que tivermos dados em uma tabela podemos usar a posição de linhas e colunas para localizar um elemento

Tabela de dados

Ações	Produto	Valor	Quantidade	Sub-total
<input type="button" value="Excluir"/>	Nike	R\$ 100,00	1	R\$ 100
<input type="button" value="Excluir"/>	Camisa Polo	R\$ 150,00	2	R\$ 300

```
HTML
Edit
input#produtonome < div.span7 < div.span9...ll-small < div.row-fluid < fieldset < div#tab3....e.active < div.tab-content < div
<td>
<td> Nike </td>
<td> R$ 100,00 </td>
<td> 1 </td>
<td> R$ 100 </td>
<input type="hidden" value="395" name="itemprodutoid1">
<input type="hidden" value="1" name="itemprodutoquantidade1">
<input type="hidden" value="100" rev="subtotal">
</tr>
```

Estrutura da tabela – td (linhas) e tr (colunas)



Localização de Elementos

Exemplo de aplicação de posicionamento

Neste exemplo possuímos dois produtos, logo para validar o primeiro e o segundo produto (pelo nome) será necessário colocar a posição da linha

coluna td				
Ações	Produto	Valor	Quantidade	Sub-total
Excluir	Nike	R\$ 100,00	1	R\$ 100
Excluir	Camisa Polo	R\$ 150,00	2	R\$ 300

linha | tr

//tr[1]/td[2] = primeira linha | segunda coluna

//tr[2]/td[2] = segunda linha | segunda coluna



Exercício

Objetivo: No script do Pedido, adicionar mais um item

- Digitar “Nike” no campo Produto
- Preencher o campo Quantidade com “1”
- Clicar no botão Adicionar
- Validar os campos Produto, Valor, Quantidade e Subtotal da tabela para os dois produtos

Comandos

Todos os que já conhecemos :-)

Resultado Esperado

Novo produto adicionado e dados dos produtos validados pelo script



Informações finais