// Testes

Selenium

Selenium

O Selenium é um conjunto de ferramentas de software diferentes. Cada ferramenta é utilizada em abordagens diferentes para apoiar a automação de testes.

- Selenium
  - Selenium RC / WebDriver.
- Selenium IDE
  - Plugin para navegadores utilizado para registrar ações.
- Selenium Grid
  - Executar os testes em paralelo.

Selenium WebDriver

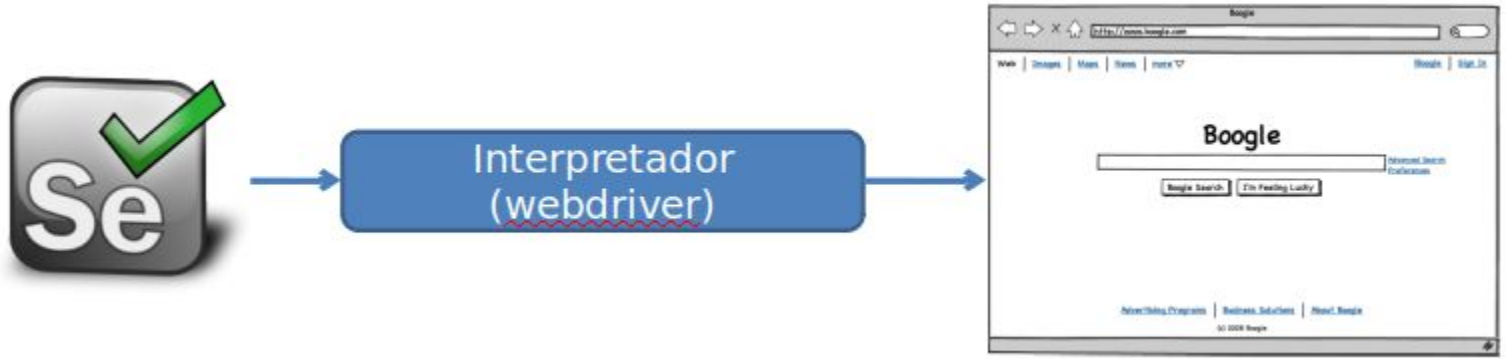
- WebDriver é uma ferramenta para automatizar testes de aplicativos web, e, em particular, para verificar se eles funcionam conforme o esperado.

- Selenium-WebDriver faz chamadas diretas para o navegador, ou seja, executa ações no navegador simulando o usuário.

- Suporte a várias linguagens de programação(java, .net, python, ruby).

Selenium WebDriver

- Geralmente o intermediador é disponibilizado em forma de executável.



// Testes

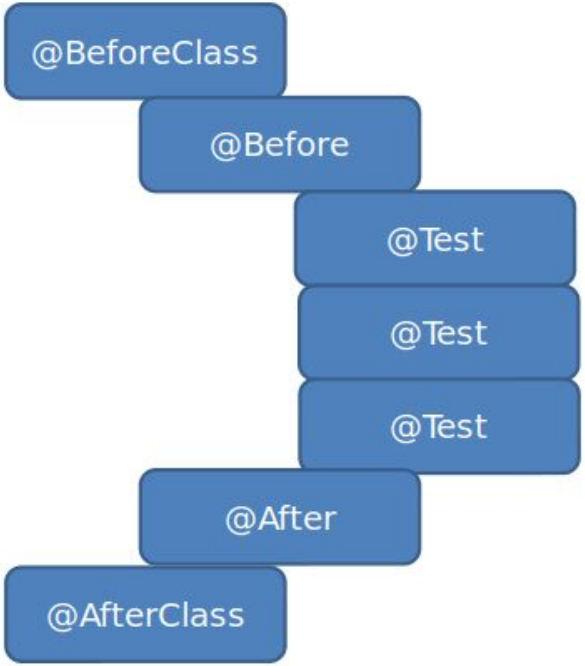
JUnit

- É um framework de testes para a criação de códigos para automação de testes em código java.
- Executa os testes automatizados.
- Checa os resultados dos testes e fornece uma resposta imediata.
- Oferece asserções para garantir o resultado esperado.

JUnit Annotations

Anotação	Descrição
@Test	Identifica um método como um método de teste;
@Test(expected = Exception.class)	Falha se o método não lançar a exceção nomeado;
@Before	Executa antes de cada teste;
@After	Executa depois de cada teste;
@BeforeClass	Executa apenas uma vez antes de todos os testes;
@AfterClass	Executa apenas uma vez depois de todos os testes;
@Ignore	Ignora um método de teste;

JUnit Annotations



JUnit Asserts

Asserção	Descrição
fail(String)	Força a falha;
assertTrue	Verifica se a condição boolean é verdade;
assertFalse	Verifica se a condição boolean é falso;
assertEquals	Verifica se os valores são iguais;
assertNull	Verifica se o objeto é nulo;
assertNotNull	Verifica se o objeto não é nulo;
assertSame	Verifica se as duas variáveis se referem ao mesmo objeto;
assertNotSame	Verifica se as duas variáveis se referem a diferentes objetos;

// Testes

Selenium WebDriver

WebDriver

- Para utilizar o WebDriver é necessário informar qual navegador iremos utilizar.

- Os navegadores não são suportados nativamente, sendo necessário o uso de Drivers específicos e configurações adicionais.

Configurando Ambiente

- Criar novo projeto Maven
  - com.matera.bootcamp.testes
  - selenium-webdriver-bootcamp

- Adicionar dependências do Maven
  - Selenium
  - JUnit

- Baixar ChromeDriver

Exercício

- Testar as anotações
  - BeforeClass
  - Before
  - Test
  - After
  - AfterClass

// Testes

WebDriver - Navegação

Navegação

- Métodos para navegação
  - WebDriver
    - get(String url);
    - navigate()
      - to(String url);
      - to(java.net.URL);
      - back();
      - forward();
      - refresh;

Navegação

- A Classe WebDriver é o ponto de partida.
- Criar um pacote chamado ***navegacao***

```
@Test
public void test(){

    System.setProperty("webdriver.chrome.driver","c:\\chromedriver.exe");

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

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 1 – Utilizando as funções de navegação.

Fechando Navegador

- WebDriver
  - close(); - Fecha a janela atual, se for a última fecha o navegador.
  - quit(); - Fecha o navegador mesmo que existam diversas janelas abertas.

- Atenção: Quando o WebDriver abre um navegador ele controla apenas este, sendo assim, não irá fechar navegadores que por ventura foram abertos manualmente.

// Testes

WebDriver - Interrogação

Interrogação

- WebElement é uma classe que representa um elemento HTML.
  - Botões
  - Caixas de Texto
  - Links
  - E todas as demais Tags suportadas pelo padrão HTML do W3C

- Um objeto do tipo WebElement irá guardar todos os atributos de um elemento.
- Nos dá o poder de interrogá-los

Interrogação

Método	Descrição
getText()	Retorna o texto de um elemento. Este texto está entre a tag do próprio elemento.Ex: *<elemento>Texto</elemento>*
getAttribute()	Retorna o valor de um atributo de um elemento. Ex: *<elemento atributo="valorAtributo"></elemento>;*
getTagName	Retorna o nome da tag html do elemento;
isEnabled	Retorna *true* se o elemento é habilitado;
isSelected	Retorna *true* se o elemento está selecionado. Usado em checkbox e radioButtons;
isDisplayed	Retorna *true* se o elemento está visível na tela;
getLocation	Retorna a posição x e y do elemento na tela;
getCSSValue	Retorna o valor de alguma propriedade css da Página;

Interrogação

- Para localizar elementos é utilizado o método ***findElement*** passando a estratégia necessária.

Estratégia	Descrição
Id	Localiza o elemento pelo atributo *Id.*
Name	Localiza o elemento pelo atributo *name.*
tagName	Localiza o elemento por uma *tag.* Ex: input.
linkText	Localiza o elemento pelo nome do link.
partialLinkText	Localiza o elemento pelo nome parcial do link.
cssSelector	Localiza o elemento pelo seletor CSS.
className	Localiza o elemento pelo Atributo *class*
Xpath	Localiza o elemento por xpath

Interrogação

- Exemplos:
- Capturando o elemento utilizando WebElement

  WebElement botao = driver.findElement(By.id(“btn”));

- Capturando o elemento sem WebElement

  driver.findElement(By.id(“btn”));

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 2 – Localizando Elementos.
  - Localizar os elementos da tela criando um WebElement para cada elemento.
  - Localizar utilizando By.id todos os elementos que possuem id
  - Localizar o título através de By.tagName

// Testes

WebDriver - CSS Selector

CSS Selector

- CSS Selector é a forma de localizar um elemento através de seu estilo

Seletor	Descrição
.class	Localiza o elemento através da tag class. Ex: driver.findElement(By.cssSelector(“.intro.duo”));
#id	Localiza o elemento através do seu id. Ex: driver.findElement(By.cssSelector(“#firstname”));
Elemento	Localiza o elemento. Ex: driver.findElement(By.cssSelector(“p”));
elemento[atributo='valor']	Localiza um elemento pelo valor de um atributo. driver.findElement(By.cssSelector(“input[value=‘button’]”));
elemento > elemento	Localiza o próximo elemento baseado no anterior. driver.findElement(By.cssSelector(“div > p”));

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 2 – Localizando Elementos CSS e XPath.
  - Localizar os elementos da tela criando um WebElement para cada elemento.
  - Localizar:
    - Div Pai
    - Div Filho
    - Div Neto 1
    - Data Inicio Relatório

CSS Selector Avançado

Seletor	Descrição
elemento[atributo^='valor']	Localiza o elemento que inicia com o valor de um atributo. Ex: driver.findElement(By.cssSelector("input[class^='cls']"));
elemento[atributo$='valor']	Localiza o elemento que termina com o valor de um atributo Ex: driver.findElement(By.cssSelector("input[id$='tst']"));
elemento[atributo*='valor']	Localiza o elemento que contém o valor de um atributo. Ex: driver.findElement(By.cssSelector("input[name*='nm']"));

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 2 – Localizando Elementos CSS e Xpath (Avançado).
  - Localizar os elementos da tela criando um WebElement para cada elemento.
  - Localizar:
    - Email
    - Senha
    - Repetir Senha
    - Botão Cadastrar

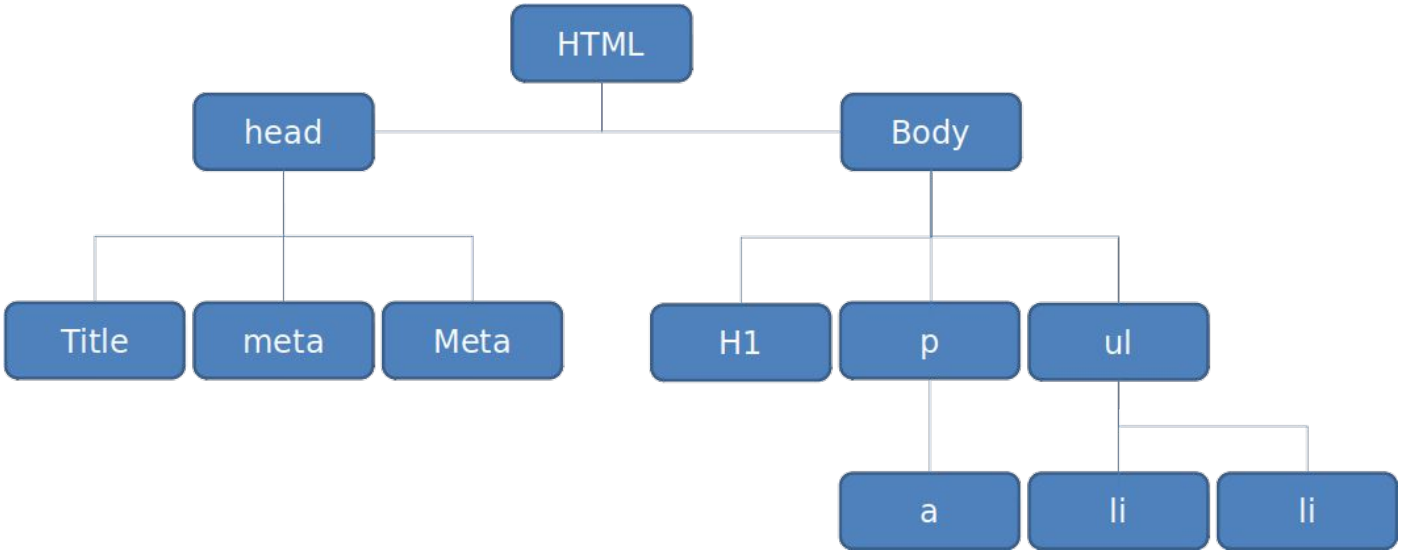
// Testes

XPath

XPath

- A linguagem XPath é baseada em uma representação de árvore do documento XML, e fornece a capacidade de navegar ao longo da árvore, selecionando nós por uma variedade de critérios.
- Possui mais métodos de pesquisa que o CssSelector.
- Pode ser mais lento na localização dos elementos.
- Pode quebrar facilmente se utilizarmos ele buscando a posição do elemento.

DOM (Document Object Model)



XPath Seletor Básico

Seletor	Descrição
//elemento	Localiza o elemento. *driver.findElement(By.xpath("//input"));*
//elemento[@atributo]	Localiza o elemento que tenha o atributo descrito. *driver.findElement(By.xpath("//input[@placeholder]"));*
//elemento[@atributo="valor"]	Localiza o elemento que tenha o valor do atributo descrito. *driver.findElement(By.xpath("//input[@class='odd']"));*
//elemento1/elemento2	Localiza o elemento 2 através do elemento 1 (é filho). *driver.findElement(By.xpath("//div/ul/li/a"));*
//elemento[numero]	Localiza o elemento pelo seu numero/posição. *driver.findElement(By.xpath("//input[2]"));*

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 2 – Localizando Elementos CSS e Xpath.
  - Localizar os elementos da tela criando um WebElement para cada elemento.
  - Localizar:
    - Div pai
    - Div filho
    - Div neto 2
    - Data Inicio Relatório

XPath Seletor Avançado

Seletor	Descrição
//elemento[starts-with(@atributo,'valor')]	Localiza o elemento pelo valor que o atributo inicia. *driver.findElement(By.xpath("//input[starts-with(@id,'email')]"));*
//elemento[contains(@atributo,'valor')]	Localiza o elemento pelo valor que o atributo possui. *driver.findElement(By.xpath("//input[contains(@id,'senha-')]"));*
//elemento[ends-with(@atributo, 'valor')]	Localiza o elemento pelo valor que o atributo termina. *driver.findElement(By.xpath("//ends-with(@id,'-senha')]"));*
//elemento[contains(text(), 'valor')]	Localiza o elemento pelo texto que um elemento possui. *driver.findElement(By.xpath("//input[contains(text(),'Senha')]"));*

XPath Seletor Avançado

- Xpath 2.0 permite a criação de cláusulas AND e OR para realizar a pesquisa de um elemento

  driver.findElement(By.xpath(“
  //input[contains(@id,’senha-’)] and //input[contains(@name, ’senha’)]”));

  driver.findElement(By.xpath(“
  //input[contains(@id,’senha-’)] or //input[contains(@name, ’senha’)]”));

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 2 – Localizando Elementos CSS e Xpath(avançado).
  - Localizar os elementos da tela criando um WebElement para cada elemento.
  - Localizar:
    - Email
    - Repetir Senha

Selenium WebDriver

- Navegação
  - Ações sobre uma página
- Interrogação
  - Obter informações de uma página
- Manipulação
  - Ações em elementos de uma página
- Sincronização
  - Esperas

// Testes

WebDriver - Manipulação

WebDriver - Manipulação

- Após interrogar um elemento, podemos manipulá-lo
- Um WebElement possui métodos para manipulação.
- WebElement
  - Click() - Simula ação do clique.
  - Clear() - Limpa o conteúdo do elemento, geralmente campos de texto.
  - sendKeys(String) – Envia eventos do teclado, geralmente preenche campos de texto.
  - Submit() – Simula o submit de um formulário.

ComboBox

- Existe uma classe específica para trabalhar com Combobox, chamada Select

  Select combo = new Select(WebElement);

- O parâmetro WebElement é um elemento descrito por WebElement:

  WebElement estadoSelect = new Select(driver.findElement(By.id("estado")));
  Select combo = new Select(estadoSelect);

- Ou direto:

  Select combo = new Select(driver.findElement(By.id("estado")))

ComboBox

- Métodos de manipulação de um select

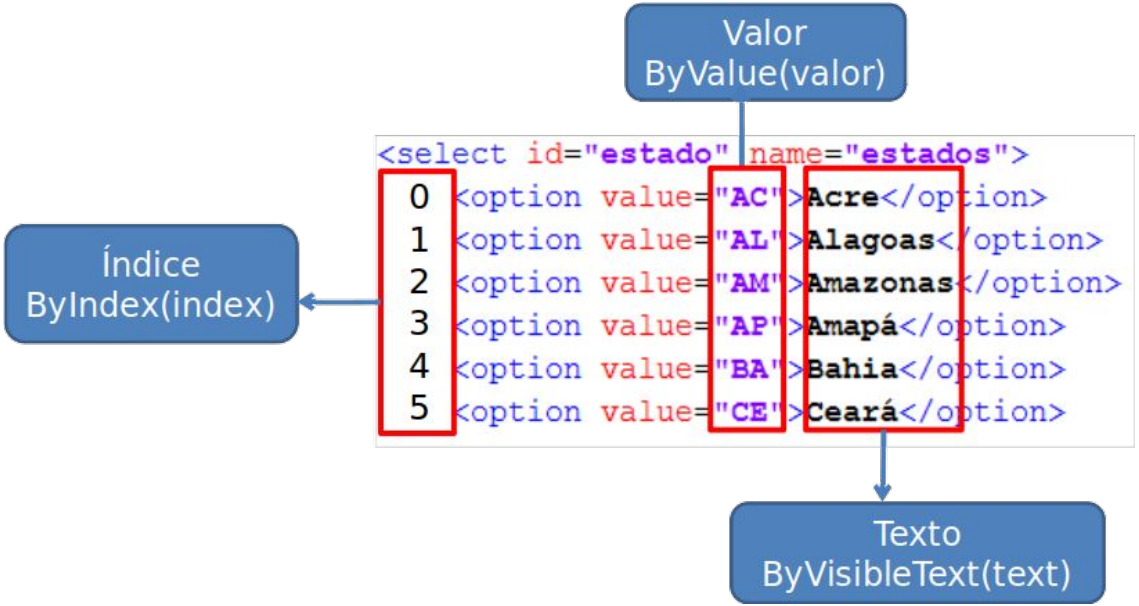
Seletor	Descrição
deselectAll()	Remove todas as seleções de um combo.
deselectByIndex(value)	Remove uma seleção pelo índice do item.
deselectByValue(value)	Remove uma seleção pelo valor do item.
deselectByVisibleText(text)	Remove uma seleção pelo texto visível do item.
getAllSelectedOptions()	Retorna todas os itens selecionados.
getFirstSelectedOption()	Retorna o primeiro item selecionado.

ComboBox

- Métodos de manipulação de um select

Seletor	Descrição
getOptions()	Retorna todos os itens da combo/list.
isMultiple()	Verifica se é uma combo ou list.
selectByIndex(index)	Seleciona um item pelo índice.
selectByValue(value)	Seleciona um item pelo valor.
selectByVisibleText(text)	Seleciona um item pelo texto visível.

ComboBox


Valor
ByValue(valor)
Índice
ByIndex(index)
Texto
ByVisibleText(text)
<select id="estado" name="estados">
0 <option value="AC">Acre</option>
1 <option value="AL">Alagoas</option>
2 <option value="AM">Amazonas</option>
3 <option value="AP">Amapá</option>
4 <option value="BA">Bahia</option>
5 <option value="CE">Ceará</option>

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 3 – Manipulando Elementos.
  - Preencher os dados conforme BDD.

// Testes

WebDriver - Sincronização

Sincronização

- Existem duas formas de sincronização
  - Explícita - Espera até a condição ser verdadeira, ou até o tempo limite.
  - Implícita - “Diz” ao WebDriver para sondar o DOM por um determinado período de tempo ao tentar encontrar qualquer elemento

- Utilizado em casos onde há espera por elementos

Espera Explícita

- Para a espera explícita temos que fazer uso de uma classe de suporte chamada WebDriverWait.

  WebDriverWait wait = new WebDriverWait(driver, 15);

Espera Explícita

Método	Descrição
Until(true)	Aguada até determinada condição
pollingEvery(duration, unit)	Determina o intervalo da contagem de tempo até a próxima tentativa
withMessage(message)	Adiciona uma mensagem quando ocorrer o timeout
ignoring(exceptionType)	Ignora alguma *Exception* que possa ocorrer durante a espera
withTimeout(timeout)	Sobrescreve o timeout

Espera Explícita

- O método mais utilizado é o until.
- O método until espera até que uma condição seja verdadeira, para isso utilizamos uma classe estática que possui diversos métodos prontos, que retornam essa condição. Essa classe chama-se ExpectedConditions.
- Cada método pode ter um parâmetro diferente de um WebElement.

wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.id(“teste”))));

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 4 – Esperando Elementos 1.
- Preencher os dados conforme BDD.
- Utilizar a Espera Explícita.

Espera Implícita

- A forma de espera implícita é a mais simples, mas que pode trazer uma demora na execução dos testes.

- Este tipo de espera serve para todo o script, não sendo necessário utilizar esperas explícitas.

- Exemplo:
  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

Exercício

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 4 – Esperando Elementos 2.
- Preencher os dados conforme BDD.
- Utilizar a Espera Implícita.

Selenium WebDriver

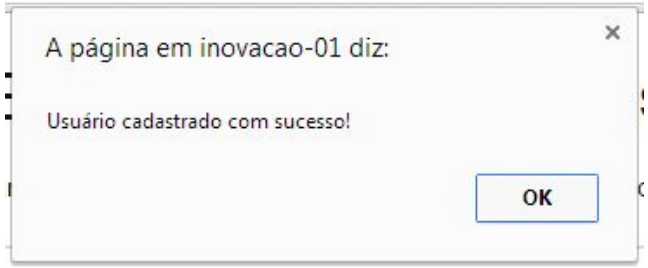
- Navegação
  - Ações sobre uma página
- Interrogação
  - Obter informações de uma página
- Manipulação
  - Ações em elementos de uma página
- Sincronização
  - Esperas

// Testes

Alertas e Confirmações

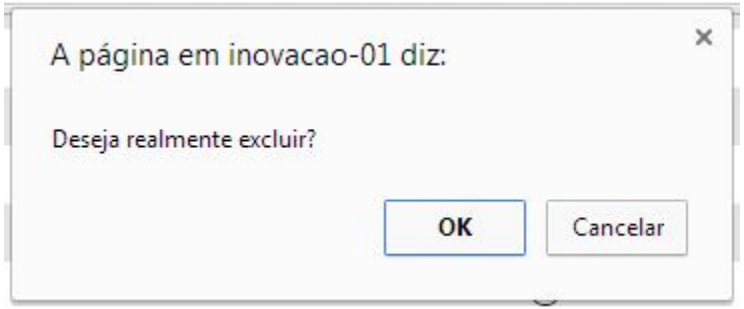
Alertas

- São mensagens em Javascript onde o browser consegue renderizar em um formato fora do browser (parecendo uma janela).
- Um alerta possui uma mensagem informativa e um botão de OK



Confirmações

- Uma confirmação possui uma mensagem, um botão de OK (que pode significar ‘sim’) e outro de Cancelar (que pode significar ‘não’).



Alertas e Confirmações

- Para interrogar um alerta é utilizado o comando:

  Alert alerta = driver.switchTo.alert();

- Após isso podemos manipulá-lo:

Método	Descrição
getText()	Retorna a Mensagem de texto de um alerta ou confimação
dismiss()	Clica no botão Cancelar de uma confirmação
accept()	Clica no botão Ok de um alerta ou confirmação
sendKeys(String)	Envia um texto se for do tipo *prompt*

Exercícios

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 5 - Alertas, Confirmações e Prompts 1.
  - Tentar logar sem preencher 1 dos campos, e interrogar a mensagem.

Exercícios

- Acessar a página de exercício
  - http://bootcamp-selenium.matera.com

- Acessar exercício 5 - Alertas, Confirmações e Prompts 2.
- Realizar 1 teste para cada botão, capturando a mensagem e manipulando todos os elementos.
- Utilizar Annotatios do JUnit

// Testes

ScreenShots

ScreenShots

- Um ponto importante para poder evidenciar o teste quando este falha é tirar um printscreen/screenshot da tela.
- Isso serve para identificarmos exatamente o ponto que a aplicação falhou.
- Uma screenshot no WebDriver é sempre extraída da forma abaixo:

```
File arquivo =
((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(arquivo, new File("caminho"));
```

ScreenShots

- Blocos try-catch é a forma mais eficaz de obter uma ScreenShot. Basta colocar o comando para obter a ScreenShot dentro do bloco catch e sempre que algo der errado, será feita a captura da tela.

- Exemplo:

```
try{
        //código de teste
}catch(Exception e){
        //código para capturar a tela
}finally{
        //código final, geralmente browser.quit();
}
```

Exercício

- Copie o código de testes da classe EsperaImplicitaTest
- Adicione blocos Try-Catch.
- Altere o script para ficar com a estrutura de captura automática de evidência quando um erro ocorrer.
- Modifique o script para ter um erro na comparação de resultados.

// Testes

Page Objects

Page Objects

- É um padrão de projeto que ajuda no desenvolvimento dos testes.
- Cada página que interagimos vira uma classe (objeto) que será utilizado como um serviço.
- Um PageObject não precisa representar uma página inteira. Pode representar uma seção que aparece muitas vezes dentro de um site ou página, tais como a navegação do site.
- O princípio fundamental é que não exista dentro do código de testes o conhecimento da estrutura HTML de uma página.

Page Objects

- Um Page Object sempre é uma página web, e apenas uma.
- Pode conter um construtor recebendo o browser como parâmetro.
- Não deve conter asserções dentro do Page Objects.

Page Objects

PageObject

```
Public void preencherCampoNome( String
nome) {
    driver.findElement(
By.id(“name”)).sendKeys(nome);
}

Public void preencherCampoSenha( String
senha) {
    driver.findElement(
By.id(“password”)).sendKeys(senha);
}

Public void clicarBotaoLogar(){
   driver.findElement(By.id(button)).click();
}
```

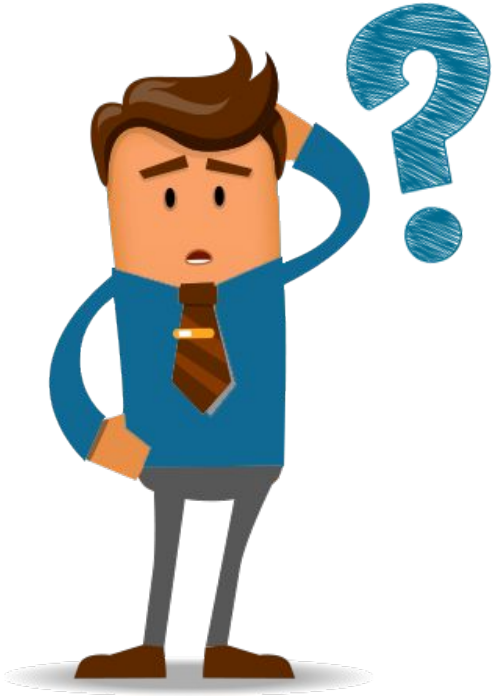
Test

```
PageObject po = new PageObject();
po.preencherCampoNome(“nome”);
po.PreencherCampoSenha(“123”);
po.clicarBotaoLogar();
```

Exercício

- Copiar o teste EsperaExplicitaTest
- Criar uma PageObject para a tela de login.

Dúvidas?



matera

www.matera.com