Automação de testes com Selenium WebDriver Utilização

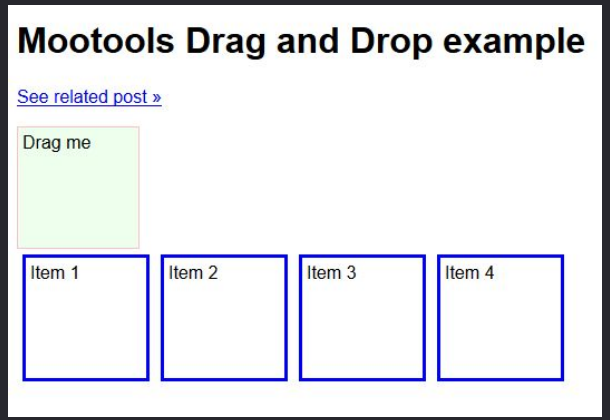
Antônio Moraes Trindade

https://about.me/amtrindade

Navegação por múltiplas janelas

Navegação por múltiplas janelas

Selenium WebDriver nos permite navegar por diferentes janelas abertas durante a execução do teste.
Para testarmos esta funcionalidade, vamos criar um teste que acesse a home da página de treinos e clique no link “Drag and Drop”



Navegação por múltiplas janelas

- Clique no link “Drag and Drop”
- Valide o título da página com getTitle() “Mototools Drag and Drop example”
- Volte o foco para página principal e valide o título da página “Treino Automação de Testes”

Dicas:
1. Capture as abas abertas.

```
ArrayList<String> tabs = new
ArrayList<String>(driver.getWindowHandl
es());
```

2. Dê o foco do driver para a aba que deseja interagir.

```
driver.switchTo().window(tabs.get(0))
```

Navegação por ações do browser

Navegação por ações do browser

Vamos agora utilizar os comandos de forward e backward simulando estas ações do browser. Equivalem as mesmas ações do browser.

- driver.navigate().back();
- driver.navigate().forward();

Navegação por ações do browser

Atividade: Elaborar um teste que realize as seguintes ações:

- Realizar a navegação pelas páginas dos links “Calculadora” e “Localizar Table”
- A cada página validar o título com getTitle()
- Voltar para home através do back() e validar o título
- Ir até a página “Localizar Table” e validar o título através do forward()

Categorização de testes com JUnit

Framework para testes unitários e funcionais

Categorizar testes com JUnit

As categorias no JUnit são notações onde se pode categorizar e organizar a execução dos testes. Com isso, podemos escolher quais testes devem ser executados de acordo com a categoria que queremos para o momento.

Notação	Descrição
@Category	Representa uma categoria de teste. Deve ser adicionada aos métodos com a anotação @Test
@IncludeCategory	Inclui uma ou mais categorias de testes a uma suíte de testes
@ExcludeCategory	Exclui uma ou mais categorias de testes de uma suíte de testes
@RunWith(Categorias.class)	Informa suíte de testes que está será executada utilizando as categorias de testes

Categorizar testes com JUnit

A categoria é uma interface em Java. O nome da interface será o identificador da categoria.

Vamos criar 2 interfaces em nosso projeto.

Utilize um package específico para as interfaces.

No Eclipse acesse: File > New > Interface

Categorizar testes com JUnit

Adicione conforme exemplo abaixo as anotações aos testes:

```java
public class LoginTest {

    @Test
    @Category({MainTests.class, FastTests.class})
    public void testLoginComSenhaCorreta() {
        System.out.println("Executou testLoginComSenhaCorrega");
    }

    @Test
    @Category(FastTests.class)
    public void testLoginComSenhaInvalida(){
        System.out.println("Executou testLoginComSenhaInvalida");
    }

    @Test
    @Category(SlowTests.class)
    public void testLoginComUsuarioInvalido(){
        System.out.println("Executou testLoginComUsuarioInvalido");
    }
}
```

```java
public class ProdutoTest {

    @Test
    @Category({MainTests.class, FastTests.class})
    public void testConsultaProduto() {
        System.out.println("Executou testConsultaProduto");
    }

    @Test
    @Category(SlowTests.class)
    public void testConsultaBrinde() {
        System.out.println("Executou testConsultaBrinde");
    }
}
```

Categorizar testes com JUnit

Para executar os testes com esta categorização, vamos precisar criar suítes de testes.
Nesta suíte de testes precisaremos adicionar 3 informações:

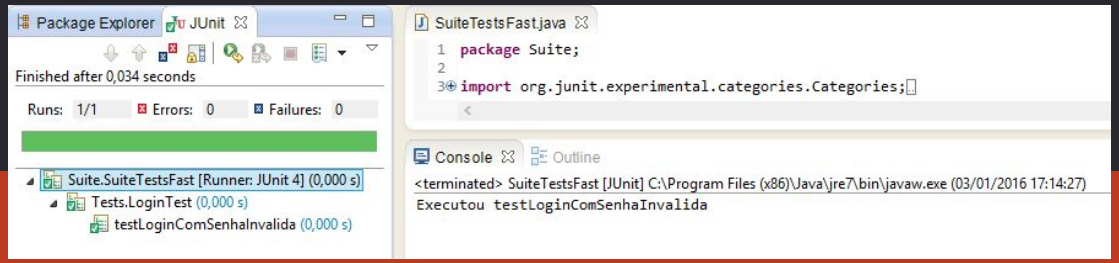
1. A notação @RunWith informa a execução apenas das Categorias
2. A notação @IncludeCategory inclui as Categorias ou @ExcludeCategory exclui as Categorias da execução
3. A notação @SuiteClasses são as classes que serão executadas

Categorizar testes com JUnit

Crie 2 suítes, conforme estrutura da imagem abaixo. Execute as suítes e analise o resultado conforme a categorização proposta.

SuiteTestsFast.java

```
1  package Suite;
2
3  import org.junit.experimental.categories.Categories;
13
14 //Informa para o JUnit que ele usará as categorias para a execução
15 @RunWith(Categories.class)
16
17 //Informa as classes de testes que estão contidas nesta suíte
18 @SuiteClasses({ProdutoTest.class, LoginTest.class})
19
20 @IncludeCategory(FastTests.class)
21
22 @ExcludeCategory(MainTests.class)
23
24 public class SuiteTestsFast {
25
26 }
27
```



Expressões Regulares

O que são Expressões regulares?

Uma expressão regular é uma String especialmente formatada que mostra um padrão de pesquisa e substituição de textos.

Para criar uma expressão regular é necessário informar caracteres especiais usados no padrão da expressão. Esses caracteres são conhecidos como metacaracteres, indicando a ocorrência de números, letras entre outros caracteres no texto.

Leia mais em: Entendendo de uma vez por todas Expressões Regulares

Caracteres utilizados para construir uma expressão regular

Caractere	Descrição	Metacaractere
.	Busca qualquer caractere	
\d	Busca qualquer número	[0-9]
\D	Busca qualquer caractere que não seja número	[^0-9]
\w	Busca qualquer caractere de letras e números	[a-zA-Z_0-9]
\W	Busca qualquer caractere que não sejam letras e números	[^\w]
\s	Busca qualquer caractere de espaço em branco, tabulações	[\t\n\x0B\f\r]
\S	Busca qualquer caractere sem espaço em branco	[^\s]

O método matches

Especifica uma expressão regular e localiza o conteúdo do objeto String em que está sendo aplicada essa expressão. Para saber se essa correspondência foi atendida ou não, é retornado um valor booleano.

```
public class TestaExpressoes {
        public static void main(String[] args) {
                boolean nome = "Maria".matches("Maria");
                System.out.println("Retorno = "+nome);
        }
}
```

Modificadores

Caractere adicionado depois de um o delimitador final, onde acaba mudando o jeito como a função irá tratar a expressão. Abaixo alguns modificadores que podem ser utilizados:

(?i) – Ignora maiúsculas e minúsculas;

(?s) – Faz com que o caractere encontre novas linhas;

(?x) – Permite inclusão de espaços e comentários.

Quantificadores

O quantificador é um caractere que consegue informar quantas vezes um metacaractere pode ser repetido.

Expressão	Descrição
X{n}	X procura a ocorrência de um caractere n vezes
X{n,}	X pelo menos n vezes
X{n,m}	X pelo menos n mas não mais que m
X?	0 ou 1 vez
X*	0 ou mais vezes
X+	1 ou mais vezes
X{n}	X procura a ocorrência de um caractere n vezes

Metacaracteres de fronteira

Esses metacaracteres definem se a String começa ou termina com um determinado padrão

Metacaractere	Objetivo
* ^	Inicia
* $	Finaliza
* |	Ou (condição)

Agrupadores

Tem como objetivo agrupar conjuntos de caracteres que tenham alguma das ações descritas na figura abaixo.

Metacaractere	Objetivo
* [...]	Agrupamento
* [a-z]	Alcance
* [a-e][i-u]	União
* [a-z&&[aeiou]]	Interseção
* [^abc]	Exceção
* [a-z&&[^m-p]]	Subtração
* \x	Fuga literal

Para criar expressões regulares

Para testar a expressão regular:

http://tools.lymas.com.br/regexp_br.php

Exercício: Expressões regulares

Elaborar 4 testes para geração de CPF e CNPJ randômico, COM e SEM pontuação.

Os testes devem passar sempre, validando a máscara gerada no campo textfield não importando qual o número gerado, desde que este respeite a máscara definida.

Acesse: https://www.geradordecpf.org
Acesse: https://www.4devs.com.br/gerador_de_cnpj

Drag and Drop

Drag and drop

WebDriver nos permite também fazer o drag and drop em páginas web, bastando utilizar o comando.

```
new Actions(driver).dragAndDrop(Origem, Destino).perform();
```

```
WebElement Origem = driver.findElement(By.id(“origem"));

WebElement Destino = driver.findElement(By.xpath("destino"));
```

Desafio drag and drop

http://jqueryui.com/resources/demos/droppable/default.html

1. Valide o texto dos dois componentes disponibilizados em tela.
2. Arraste o componente 1 para a posição 2.
3. Valide o texto alterado da posição 2.

Screenshots como evidência de teste

Screenshot como evidência

O WebDriver nos permite tirar screenshots de telas gravando evidências da execução de nossos testes.

Para isto nos fornece a interface TakesScreenshot para captura. Esta interface nos prove o método getScreenshotAs() para capturar a tela na instância do driver.

Screenshot como evidência

O código abaixo faz a captura e armazenamento da imagem.

```
File scrnShot =
 ((TakesScreenshot)driver).getScreenShotAs(OutputType.FILE);

FileUtils.copyFile(scrnShot , new File(“e:\\main_page.png”));
```

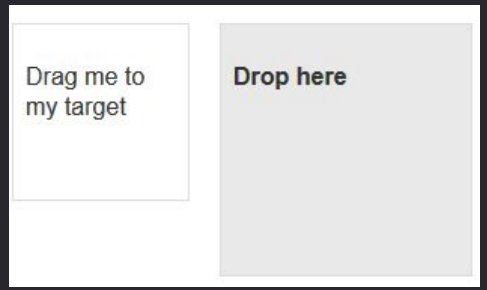
Para isso precisamos importar as bibliotecas:

```
import org.openqa.selenium.TakesScreenshot;
import org.apache.commons.io.FileUtils;
Import java.io.File;
```

Exercício: Screenshot como evidência

Vamos modificar o script do drag and drop, efetuando um screenshot antes de arrastar o componente, e após arrastar.

1º) Screenshot ao carregar a aplicação;



2º) Screenshot após o drag and drop;