

(48) 4052-9536 / 9540

[contato@qualister.com.br](mailto:contato@qualister.com.br)



qualister

- Terceirização de profissionais
- Consultoria de teste
- Avaliação de usabilidade
- Automação de testes
- Testes de performance
- Treinamentos

# Treinamento

## Selenium Avançado

## Importante

- É proibida a cópia e reprodução de qualquer parte do conteúdo desta apresentação incluindo, mas não limitado a, textos, imagens, gráficos e tabelas. Esta apresentação é protegida pelas leis de Copyright e são propriedade de Qualister Consultoria e Treinamento LTDA.
- Não é permitido modificar, copiar, guardar em banco de dados público, alugar, vender ou republicar qualquer parte desta apresentação, sem prévia permissão explícita do autor.
- Quando houver permissão de uso deste material, é obrigatória a referência bibliográfica conforme as normas vigentes.



## Elias Nogueira

**Email:** elias.nogueira@gmail.com

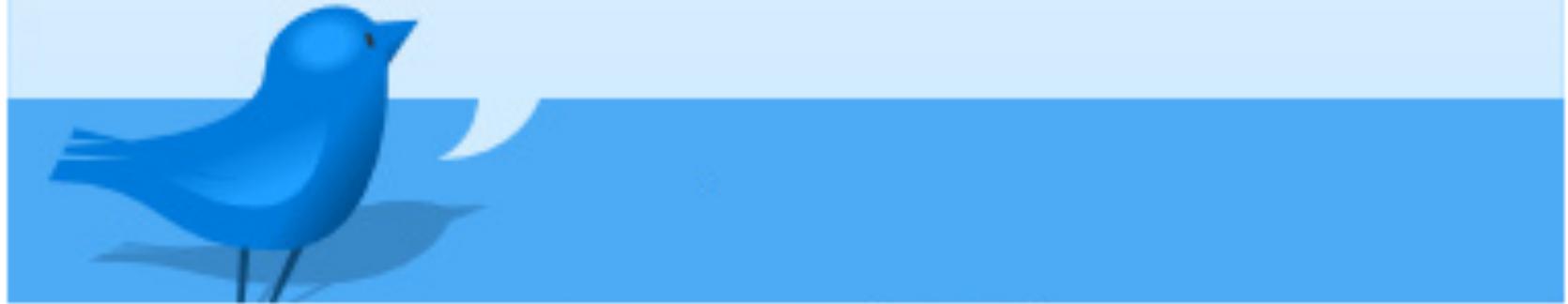
**Blog:** <http://sembugs.blogspot.com>

Arquiteto de Teste de Software com ênfase em automação de Teste.  
Atualmente trabalha como Consultor Técnico na linha de ALM da Hewlett Packard e Consultor em Automação e Teste e Instrutor na Qualister.

Possui a certificação CSTE – Certified Software Tester pela QAI e possui um Pos Graduação em Teste de Software

Já deu diversas palestras sobre Java e Teste de Software. Ativo na comunidade de Teste também coordenou alguns eventos e encontros de testes, como no Testadores e no The Developers Conference

[twitter.com/eliasnogueira](https://twitter.com/eliasnogueira)





[www.slideshare.net/elias.nogueira](http://www.slideshare.net/elias.nogueira)

# Sobre a Qualister

- **Fundação:** 2007.
- **Sobre a Qualister:** A Qualister é uma empresa nacional, constituída a partir da união de profissionais qualificados e certificados na área de testes e qualidade de software, com o objetivo de integrar, implementar e implantar soluções com base nas melhores práticas do mercado e normas internacionais.
- **Colaboradores:** A Qualister é composta por colaboradores pós-graduados e certificados na área de testes (CBTS, CSTE) com larga experiência na indústria de Tecnologia da Informação.
- **Área de atuação:** A Qualister é uma empresa especializada em serviços de qualidade e teste de software. Tem como linhas de atuação consultoria em teste/qualidade de software, outsourcing (terceirização dos serviços através da alocação de profissionais) e treinamentos.
- **Localização:** A Qualister está localizada em Biguaçu na Grande Florianópolis/SC e está instalada no CITEB – Centro de Inovação Tecnologia de Biguaçu no campus da universidade UNIVALI.

## Alguns clientes



IBOPE



# Alguns clientes



# Parcerias internacionais

	<p>Soluções para automação, profilling e gestão</p> <p><b>TestComplete™</b> automate your tests</p> <p><b>AQtime</b> eliminate leaks and bottlenecks</p> <p><b>Automated Build Studio</b> simplify your deployment</p>
 <p><b>Soft Logica</b> <a href="http://www.softlogica.com">www.softlogica.com</a></p>	<p>Soluções para testes de performance</p> <p><b>WAPT PRO</b> WEB APPLICATION TESTING</p> <p><b>WAPT</b></p> <p><b>SERVER SUPERVISOR</b></p>
	<p>Soluções de apoio a avaliação de usabilidade</p> <p><b>Morae</b> TechSmith</p>

# Exemplos

- Todos os exemplos podem ser visualizados e utilizados através do arquivo .zip que cada um recebeu por email
- **Selenium IDE:** na pasta Treinamento Selenium Avancado \Scripts\_Selenium\_IDE
- **Seleium Data Driven:** na pasta Treinamento Selenium Avancado \Scripts\_Selenium\_IDE\_DDT
- **Selenium Java:** na pasta Treinamento Selenium Avancado \Projetos\_Java

A photograph of six people (three men and three women) standing behind a large white rectangular sign. They are all smiling and looking towards the camera. The sign is blank, intended for text to be overlaid.

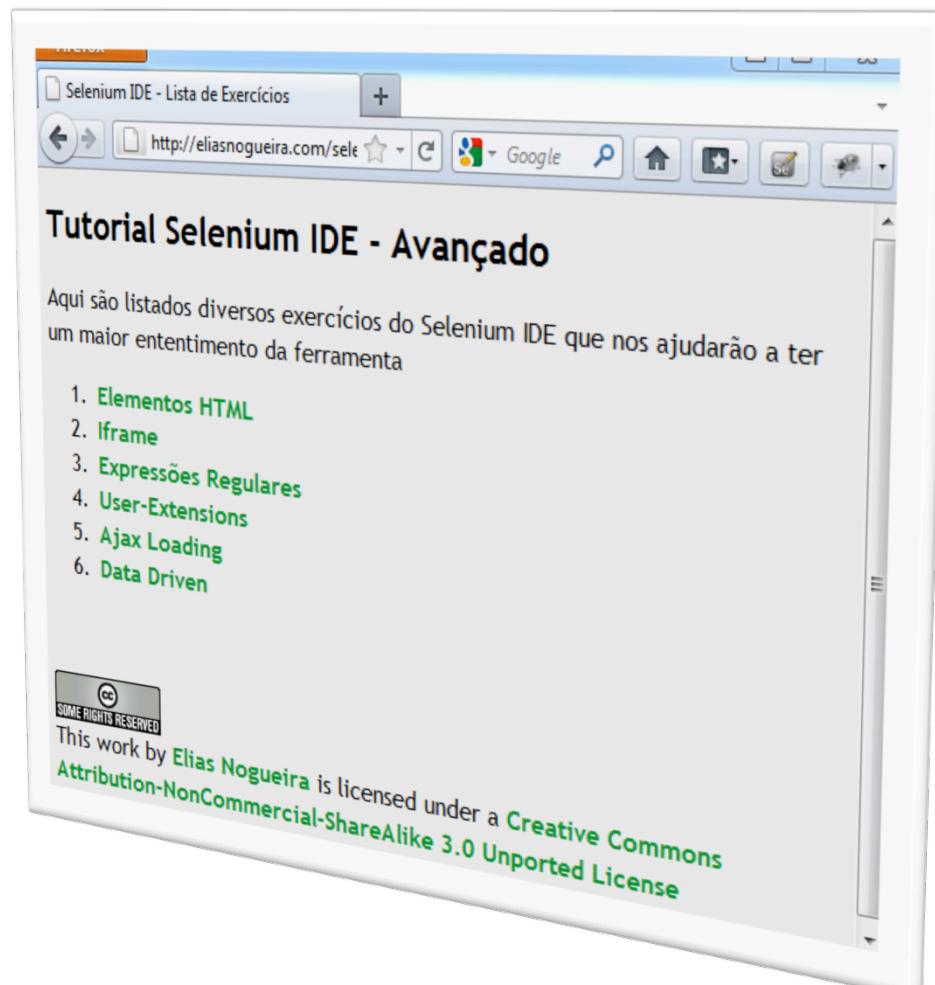
# **Exercícios do Selenium IDE**

# Exercícios com Selenium IDE

[http://eliasnogueira.com/selenium/exercicios/selenium\\_ide/ajybuyje/avancado/](http://eliasnogueira.com/selenium/exercicios/selenium_ide/ajybuyje/avancado/)

<http://tinyurl.com/seleniumavancado>

- Elementos HTML
- Iframe
- Expressões Regulares
- User-Extensions
- Ajax Loading
- Data Driven

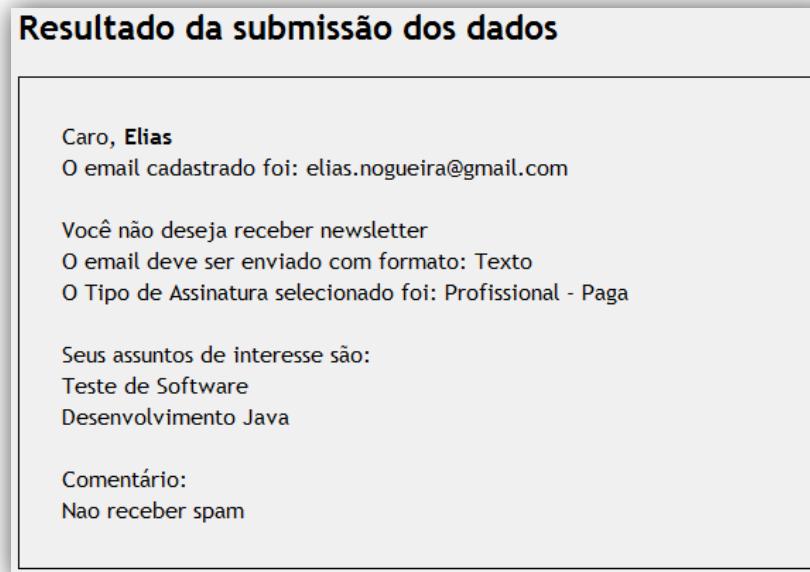


# Elementos HTML

- Consiste em apresentar uma forma fácil de interagir com todos os elementos de um form HTML disponíveis
- Será utilizado para a transição do script de Selenium 1 para Selenium 2
- Contém os seguintes elementos
  - Input
  - Password
  - Checkbox
  - Radio button
  - Combo
  - List
  - Memo

# Elementos HTML

- Este exemplo consiste em preencher todos os dados do formulário
- Após o preenchimento é necessário verificar se os dados inseridos estão na tela de resultado
- Exemplo funcional em **Treinamento Selenium Avançado**  
**\Scripts\_Selenium\_IDE\ElementosHTML.html**



## Abrir o Selenium IDE e clicar no link “Elementos HTML”

1) Em **Nome** digitar: *Fulano*

2) Em **Email** digitar: *fulano@terra.com*

3) Em **Senha** digitar: *123456*

4) Clicar para desmarcar **Receber a newsletter?**

5) Em **Formato de Email** selecionar: *Texto*

6) No **Tipo de Assinatura** selecionar: *Profissional - Paga*

7) Na lista **Selecionar o(s) assunto(s) de seu interesse** selecione os seguintes itens:

- *Teste de Software*
- *Analise de Negocios/Requisitos*

8) Em **Comentario** coloque: *Nao desejo receber spam*

9) Clicar no botão **Enviar**

10) O seguinte texto deve aparecer na tela, faça um **assertText**:

“Caro, Fulano

O email cadastrado foi: *fulano@terra.com.br*

Você selecionou o item *Receber newsletter*

O email deve ser enviado com formato: *Texto*

O Tipo de Assinatura selecionado foi: *Profissional - Paga*

Seus assuntos de interesse são:

*Teste de Software*

*Análise de Negócio/Requisitos*

Comentário:

*Nao receber spam”*

The screenshot shows the Selenium IDE interface with the following details:

- File URL:** http://eliasnogueira.com/
- Toolbar:** Includes buttons for Fast, Slow, and various execution controls.
- Table View:** Shows the recorded Selenium commands in a table format.

Command	Target	Value
open		/selenium/exercicios/sel...
type	name=nome	Fulano
type	name=email	fulano@terra.com.br
type	name=senha	123456
click	//input[@name='formato']	
select	name=tipo	label=Profissional - Paga
addSelection	name=interesse[]	label=Teste de Software
addSelection	name=interesse[]	label=Análise de Negóci...
type	name=comentario	Nao receber spam
clickAndWait	name=submit	
assertText	id=resposta	Caro, Fulano O email ca...

- Search Fields:** Command, Target, Value.
- Log Tab:** Displays the log output of the executed commands.

Log
[info] Executing:  type   name=nome   Fulano
[info] Executing:  type   name=email   fulano@terra.com.br
[info] Executing:  type   name=senha   123456
[info] Executing:  select   name=tipo   label=Profissional - Paga
[info] Executing:  addSelection   name=interesse[]   label=Teste de Software
[info] Executing:  addSelection   name=interesse[]   label=Análise de Negóci...
[info] Executing:  type   name=comentario   Nao receber spam
[info] Executing:  clickAndWait   name=submit
[info] Executing:  assertText   id=resposta   Caro, Fulano O email cadastrado foi: fulano@terra.com.br Você selecionou o item Receber newsletter O email deve ser enviado com formato:

# Iframe

- Consiste em como efetuar a automação de uma pagina html contendo um iframe
- Iframe é um código html que permite abrir uma pagina html em outra
- Hoje, com Ajax, iframes foram substituídos por divs e outros elementos, estando presente somente em paginas mais antigas

# iFrame

- Neste caso iremos automatizar o calculo de um salário, preenchendo todos os campos na tela
- Apos isso validaremos se os dados informados estarão corretos e o calculo efetuado

Selenium IDE - Exercicio 9

http://eliasnogueira.com/

Tutorial Selenium IDE - iFrame

Calcula Salario

Salário: R\$

**Calcular**

Caso queira efetuar descontos personalizados preencha os campos abaixo:

Desconto VT:	<input type="text"/>
Desconto Tk:	<input type="text"/>
Desconto Ir:	<input type="text"/>

Voltar

CC SOME RIGHTS RESERVED

This work by Elias Nogueira is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#)

## Abrir o Selenium IDE e clicar no link “IFrame”

1) Em **Salario** digitar: 5000

2) Em **Desconto Vt** digitar: 6

3) Em **Desconto Tk** digitar: 6

4) Em **Desconto Ir** digitar: 10

5) Clicar no botão **Calcular** abaixo do campo  
**Desconto Ir**

6) No **Tipo de Assinatura** selecionar: *Profissional - Paga*

7) Os seguintes resultados devem ser apresentados em tela, selecione os dados e utilize **assertText**:

### Resultados

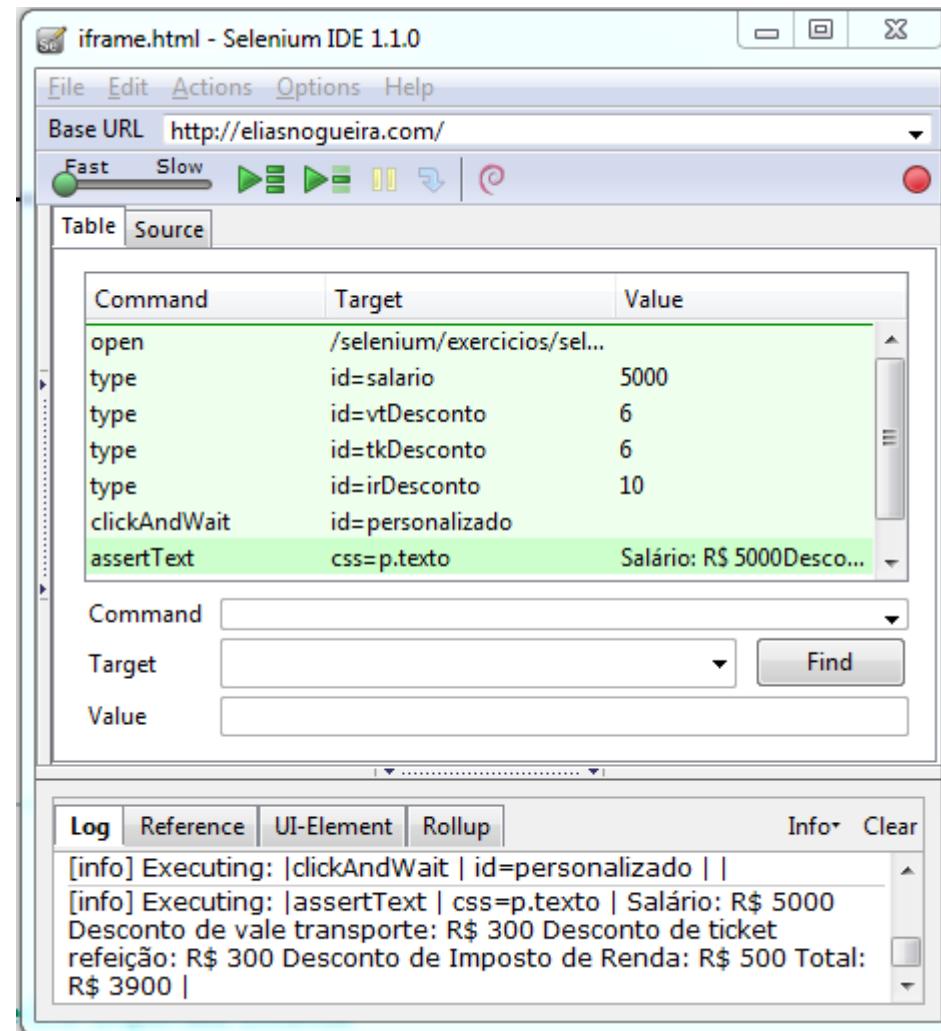
Salário: R\$ 5000

Desconto de vale transporte: R\$ 300

Desconto de ticket refeição: R\$ 300

Desconto de Imposto de Renda: R\$ 500

Total: R\$ 3900



The screenshot shows the Selenium IDE interface with the following details:

- Toolbar:** Includes File, Edit, Actions, Options, Help, and playback controls (Fast, Slow, play, stop, etc.).
- Base URL:** http://eliasnogueira.com/
- Table View:** Shows recorded Selenium commands in a table with columns: Command, Target, and Value.

Command	Target	Value
open	/selenium/exercicios/sel...	
type	id=salario	5000
type	id=vtDesconto	6
type	id=tkDesconto	6
type	id=irDesconto	10
clickAndWait	id=personalizado	
assertText	css=p.texto	Salário: R\$ 5000Desco...
- Search Fields:** Command, Target, and Value input fields.
- Log Tab:** Displays the executed commands and their results:

```
[info] Executing: |clickAndWait| id=personalizado ||  
[info] Executing: |assertText| css=p.texto | Salário: R$ 5000  
Desconto de vale transporte: R$ 300 Desconto de ticket  
refeição: R$ 300 Desconto de Imposto de Renda: R$ 500 Total:  
R$ 3900 |
```

# IFrame

- Na maioria dos casos a automação com iframes se dará da seguinte forma (pelo Selenium ou alterando o script)
  - Acessando diretamente o id do elemento dentro do iframe
    - Ex: id=salario
  - Navegando por xpath ate o elemento
    - Ex: //form[@id='form1']/table/tbody/tr/td[2]/input
- Exemplo funcional em **Treinamento Selenium Avancado\Scripts\_Selenium\_IDE\iframe.html**

# Expressões Regulares

- No Selenium temos um recurso chamado ***patterns***. Eles servem de apoio a todas a asserções/verificações de dados
- Os patterns podem ser:
  - Globais: habilita a utilização de caracteres coringa
  - Expressão Regular: habilita a utilização de *regexp*
  - Exatos: valida informações exatas mesmo com coringas
- Nosso maior foco e em Expressões Regulares
- Mais informações sobre Patters no Selenium

[http://seleniumhq.org/docs/02\\_selenium\\_ide.html#matching-text-patterns](http://seleniumhq.org/docs/02_selenium_ide.html#matching-text-patterns)

# Expressões Regulares

- Neste caso iremos gerar boletos de pagamento bancário e validar uma serie de informações contidas no boleto em tela

**Boleto Bancário**

Neste exercício nós iremos efetuar a automação e validar diversos dados através de expressões regulares. Não se prenda em validar os dados do formulário para o boleto, preencha todos os campos com valores válidos.

**Informações do Sacado**

Sacado:

Documento:

Endereço:

Bairro/Cidade:

Taxa:

Valor Total:  Ex: 300,00

## Abrir o Selenium IDE e clicar no link “Expressões Regulares”

1) Em **Sacado** digitar: *Alexandre Santos*

2) Em **Documento** digitar: *445.576.982-04*

3) Em **Endereço** digitar: *Av Paulista, 1345 apt 178*

4) Em **Bairro/Cidade** digitar: *Bela Cintra/SP*

5) Em **Taxa** digitar: *2,5*

6) Em **Valor Total** digitar: *300,00*

7) Em **Banco** selecionar: *Itau*

8) Clicar no botão **Gerar Boleto**

9) Selecionar a linha digitável colocar um *verifyText*

10) Selecionar o valor do boleto colocar um *verifyText*

## Executar o script (será executado com sucesso!)

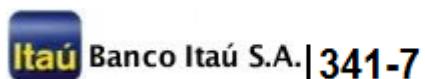
11) Alterar o valor do select no script do Selenium de “label=Itau” para “label=bradesco”

## Executar o script (ocorrerá uma falha na verificação)

The screenshot shows the Selenium IDE interface with the title "ExpressaoRegular - Selenium IDE 1.1.0". The "Table" tab is selected in the top navigation bar. The table lists the following actions:

Command	Target	Value
open		/selenium/exercicios/sel...
type	id=sacado	Alexandre Santos
type	id=sacadoDOC	445.576.982-04
type	id=endereco1	Av Paulista, 1345 apt ...
type	id=endereco2	Bela Cintra/SP
type	id=taxaBoleto	2,50
type	id=valor	300,00
select	id=banco	label=Bradesco
clickAndWait	id=submit	
verifyTextPresent		regexp:[0-9]{5}.[0-9]{5} [0...

Below the table are input fields for "Command", "Target", and "Value". At the bottom of the window are tabs for "Log", "Reference", "UI-Element", and "Rollup", along with "Info" and "Clear" buttons.



34191.75124 34567.861561 51387.710000 2 5073000030200

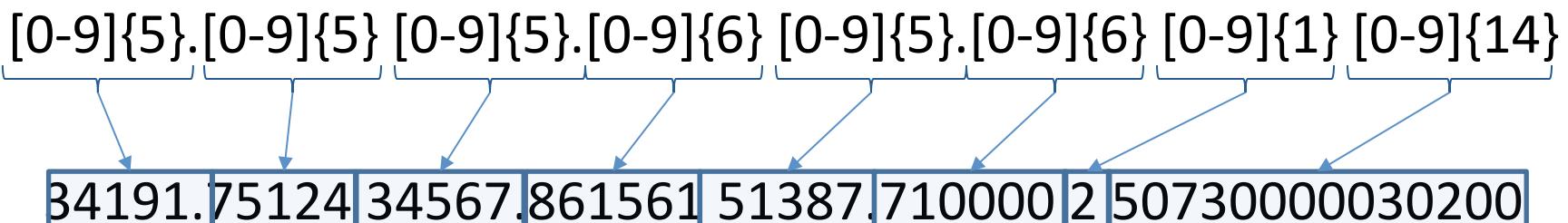
Linha digitável

# Expressões Regulares

- A execução fixa de valor para o primeiro boleto gerado no Itaú executa com sucesso, agora quando trocamos o banco o script falha
- Isso porque a linha digitável não é a mesma
- Para resolver esse problema na verificação da linha digitável devemos colocar uma expressão regular:
  - regexp:[0-9]{5}.[0-9]{5} [0-9]{5}.[0-9]{6} [0-9]{5}.[0-9]{6} [0-9]{1} [0-9]{14}

# Expressões Regulares

- Para criar esta expressão basta substituir os dados fixos pelos da expressão regular



- `[0-9]` indica que queremos números de zero (0) a nove (9)
  - `{valor}` indica a quantidade de caracteres neste conjunto
  - Essa expressão deve ser colocada no lugar do dado fixo
- Exemplo funcional em **Treinamento Selenium Avancado**  
`\Scripts_Selenium_IDE\ExpressaoRegular.html`

# User-Extensions

- No Selenium IDE podemos criar nossos próprios comandos
- Esse recurso é útil para as mais diversas finalidades
- O objetivo do exercício é gerarmos um CPF e uma senha automaticamente na execução do script
- Para isso é necessário conhecimento em programação Javascript
- Para saber mais acesse:  
[http://seleniumhq.org/docs/08\\_user\\_extensions.html](http://seleniumhq.org/docs/08_user_extensions.html)
- Exemplo funcional em **Treinamento Selenium Avançado**  
**\Scripts\_Selenium\_IDE\UserExtension.html**

## Abrir o Selenium IDE e clicar no link “User Extensions”

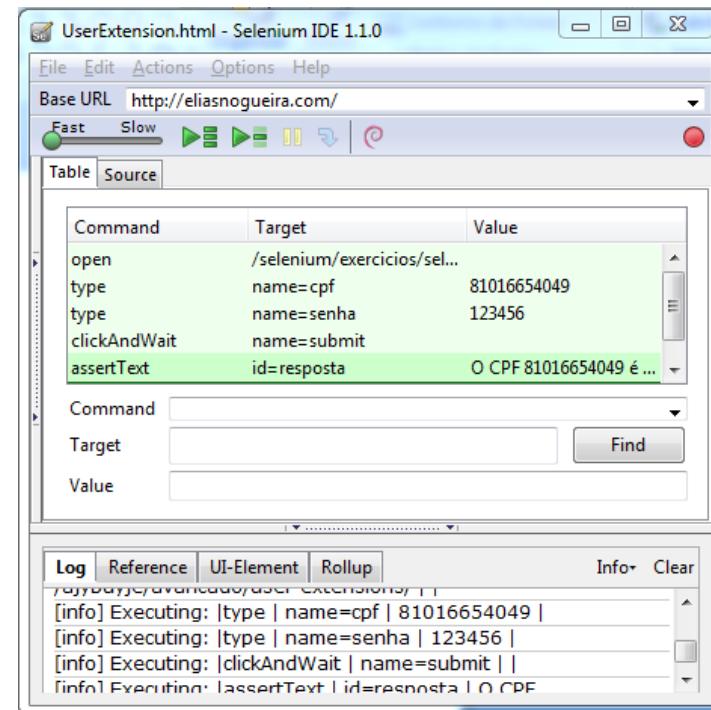
1) Em CPF digitar 81016654049

2) Em Senha digitar:123456

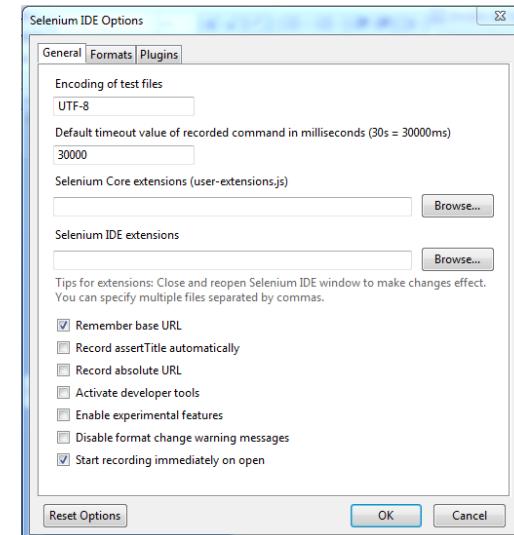
3) Clicar no botao Enviar

4) Selecionar o texto informando que o CPF e valido e colocar um *assertText*

Salve o script

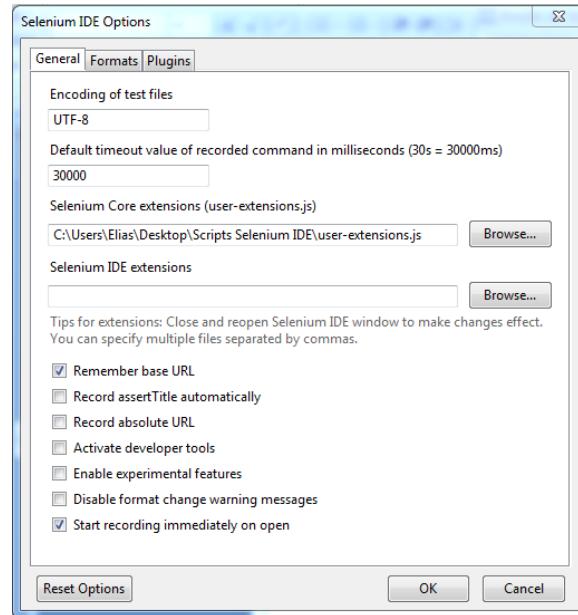


4) No Selenium clicar no menu Option/Options... para que a tela de opções seja apresentada



5) Clicar no botão **Browse...** para o item **Selenium Code extensions (user-extensions.js)** e selecionar o arquivo **user-extension.js** contido no diretório de scripts.

Apos isso clicar no botão **OK**



6) Fechar o Selenium e abri-lo novamente.  
Agora é possivel visualizar dois novos metodos

- generateCPF
- generatePassword

A utilização de ambos é a mesma: basta colocar como alvo o nome do elemento que receberá o valor de CPF ou senha

Command	Target
open	/selenium/exercic
generateCPF	name=cpf
generatePassword	name=senha
storeValue	name=cpf

7) Abra o script gravado anteriormente, clique com o botão direito sobre o item **type | name=cpf |**

**81016654049**

Selecione o item **Insert new command**

Clique sobre a linha em branco e:

- no campo **Command** selecione o comando **generateCPF**
- no campo **Target** insira **name=cpf**

Insira outro comando (**Insert new command**, mas agora:

- no campo **Command** selecione o comando **generatePassword**
- no campo **Target** insira **name=senha**

The screenshot shows the Selenium IDE interface with the title "UserExtension.html - Selenium IDE 1.1.0". The "Table" tab is selected in the top navigation bar. Below it, there is a list of recorded commands in a table:

Command	Target	Value
open	/selenium/exercicios/sel...	
generateCPF	name=cpf	
generatePassword	name=senha	

Below the table, there is a configuration panel with three fields:

- Command:** dropdown set to "generatePassword"
- Target:** input field containing "name=senha"
- Value:** empty input field

A "Find" button is located to the right of the Target field.

8) Abaixo do generatePassword insira um novo comando (**Insert new command**) e:

- no campo **Command** selecione o comando **storeValue**
- no campo **Target** insira **name=cpf**
- no campo **Value** digite: **cpf**

The screenshot shows the Selenium IDE interface with the title "UserExtension.html - Selenium IDE 1.1.0". The "Table" tab is selected in the top navigation bar. Below it, there is a list of recorded commands in a table:

Command	Target	Value
generateCPF	name=cpf	
generatePassword	name=senha	
storeValue	name=cpf	cpf
clickAndWait	name=submit	

9) Agora na linha do **assertText**, clique sobre ela e substitua o **CPF** por  **\${cpf}**

The screenshot shows the Selenium IDE interface with the title "UserExtension.html - Selenium IDE 1.1.0". The "Table" tab is selected in the top navigation bar. Below it, there is a configuration panel for the "assertText" command:

assertText	id=resposta	O CPF \${cpf} é Válido!
------------	-------------	-------------------------

Below the table, there is a configuration panel with three fields:

- Command:** dropdown set to "assertText"
- Target:** input field containing "id=resposta"
- Value:** input field containing "O CPF \${cpf} é Válido!"

A "Find" button is located to the right of the Target field.

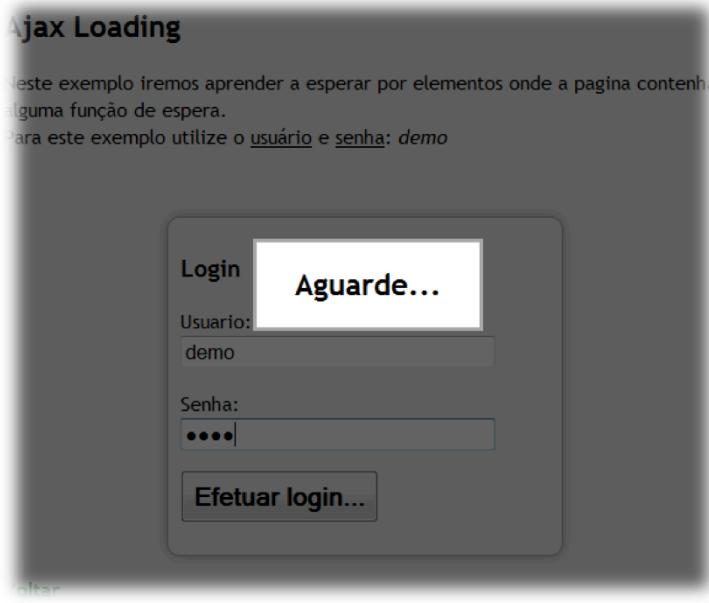
10) Execute o script

# Ajax Loading

- Hoje quando executamos ações em páginas web, muitas delas fazem isso assincronamente (sem dar refresh na tela), onde somos alertados de alguma ação por um feedback visual
- Geralmente este feedback visual se dá de duas maneiras: ou texto ou imagem
- Como grande parte das aplicações web hoje possuem requisições assíncronas (ajax) veremos como proceder com a espera destes feedbacks visuais
- Exemplos funcionais em \Treinamento Selenium Avançado \Scripts\_Selenium\_IDE\AjaxLoadingFalhaLogin.html e AjaxLoadingLoginSucesso.html

# Ajax Loading

- Neste exemplo iremos aprender a fazer a espera de duas maneiras, através da inserção de usuário e senha corretos e incorretos
- No exemplo um texto “Aguarde...” é apresentado enquanto o usuário e senha são validados



## Abrir o Selenium IDE e clicar no link “Ajax Loading”

1) Em **Usuario** digitar *demo*

2) Em **Senha** digitar:123456

3) Clicar no botao **Efetuar Login...**

4) Selecionar o texto abaixo e inserir um **assertText**:  
*Usuario ou senha invalidos.*

5) Execute o script. Ele ira dar um erro por não encontrar o elemento que contem o texto

**[error] Element css=p.error not found**

Para fazer com que ele valide o texto temos que esperar por algum feedback visual, no caso o texto **Aguarde...**

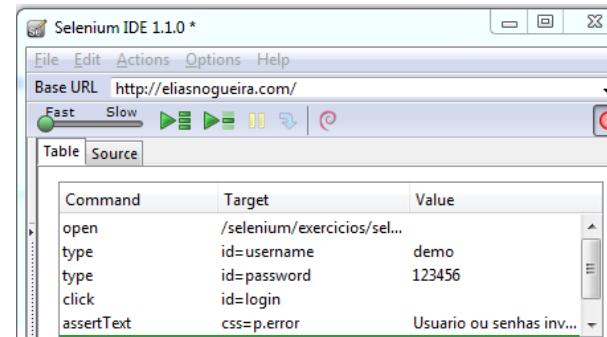
6) Insira um novo comando (**Insert new command**) acima do comando **assertText**. Coloque as seguintes informações:

- No campo **Command**: **waitForTextNotPresent**
- No campo **Target**: **Aguarde...**

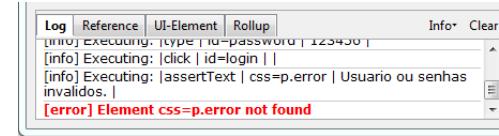
7) Execute o script. Ele deve executar com sucesso.

Isso vai ocorrer porque esperamos ate que a mensagem **“Aguarde...”** tenha sumido da tela, que e o feedback visual que podemos nos basear para validar a mensagem de erro no login.

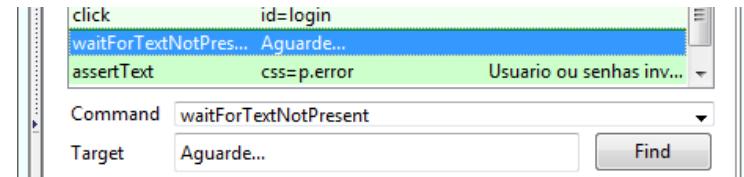
Alternativamente poderíamos esperar também pelo próprio elemento que esta apresentando o texto ou pelo próprio texto de erro no login.



Command	Target	Value
open	/selenium/exercicios/sel...	
type	id=username	demo
type	id=password	123456
click	id=login	
assertText	css=p.error	Usuario ou senhas invalidos.

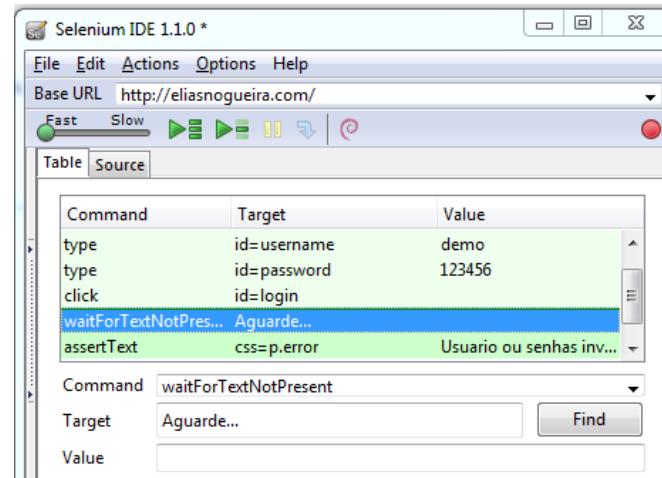


Log Reference UI-Element Rollup  
[info] Executing: type | id=password | 123456 |  
[info] Executing: click | id=login |  
[info] Executing: assertText | css=p.error | Usuario ou senhas invalidos.  
[error] Element css=p.error not found



click id=login  
waitForTextNotPres... Aguarde...  
assertText css=p.error Usuario ou senhas invalidos.

Command: waitForTextNotPresent  
Target: Aguarde... Find



Command	Target	Value
type	id=username	demo
type	id=password	123456
click	id=login	
waitForTextNotPres... Aguarde...		
assertText	css=p.error	Usuario ou senhas invalidos.

Command: waitForTextNotPresent  
Target: Aguarde... Find  
Value

## Abrir o Selenium IDE e clicar no link “Ajax Loading”

1) Em Usuário digitar *demo*

2) Em Senha digitar:*demo*

3) Clicar no botão Efetuar Login...

4) Selecionar o texto abaixo e inserir um **assertText**:  
*Voce efetuou o login com sucesso!*

5) Execute o script. Ele irá dar um erro por não encontrar o elemento que contém o texto

*[error] Element css=p.success not found*

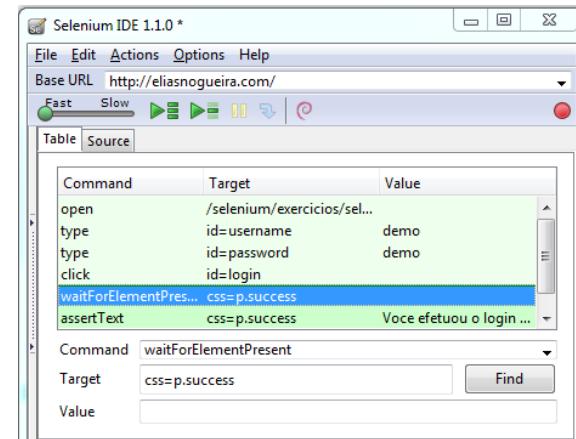
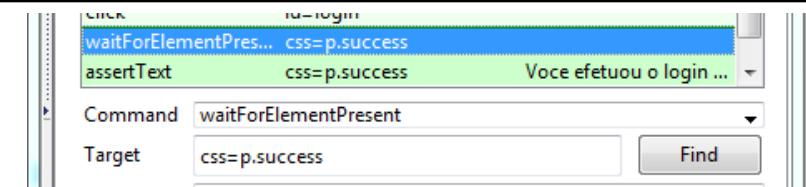
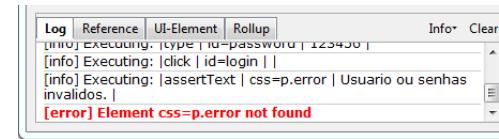
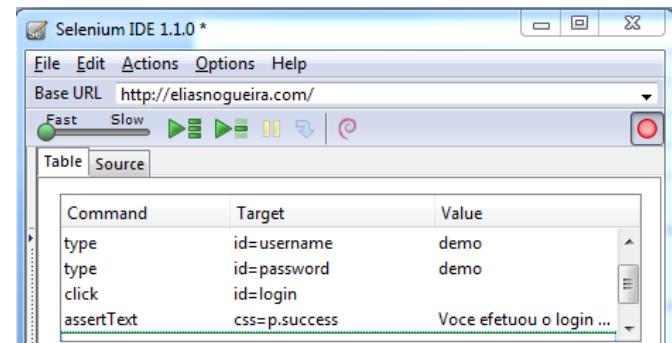
Neste caso o feedback visual do Aguarde... não funciona, pois depois que ele retoma a tela de sucesso ainda há uma ação (a box desliza) mostrando o texto segundos depois

6) Insira um novo comando (**Insert new command**) acima do comando **assertText**. Coloque as seguintes informações:

- No campo **Command**: **waitForElementPresent**
- No campo **Target**: **css=p.success**

7) Execute o script. Ele deve executar com sucesso. Isso vai ocorrer porque agora esperamos pelo elemento que contém o texto de sucesso.

Essa abordagem é melhor do que esperar pelo próprio texto, pois primeiro validamos o elemento e depois o texto., ai temos erros diferentes quando algum problema acontecer e saber mais rapidamente o que realmente ocorreu



A photograph of six people (three men and three women) standing behind a large, blank white rectangular sign. They are all smiling and looking towards the camera. The sign is positioned horizontally across the middle of the frame.

**Data Driven com  
Selenium IDE**

# Data Driven com Selenium IDE

- Data Driven é o nome dado a abordagem dirigida a dados. Isso quer dizer que um script Data Driven não terá dados fixos, e sim parâmetros para receber qualquer dado
- Uma fonte de dados se faz necessária para guardar os dados que serão utilizados
- Além disso o script deve executar o número de vezes igual a quantidade de dados na fonte de dados
- Para saber mais sobre Data Driven Testing  
[http://en.wikipedia.org/wiki/Data-driven\\_testing](http://en.wikipedia.org/wiki/Data-driven_testing)

# Data Driven com Selenium IDE

- Com o Selenium IDE o Data Driven usa como fonte de dados um arquivo XML por script, onde há um formato específico para armazenar estes dados
- Além disso três extensões se fazem necessárias para que o script do Selenium IDE saiba como trabalhar com os dados
- Essa é uma alternativa rápida quando queremos:
  - Gerar massa de dados na aplicação
  - Testar uma grande quantidade de vezes o mesmo script com dados diferentes

# Ferramentas necessárias

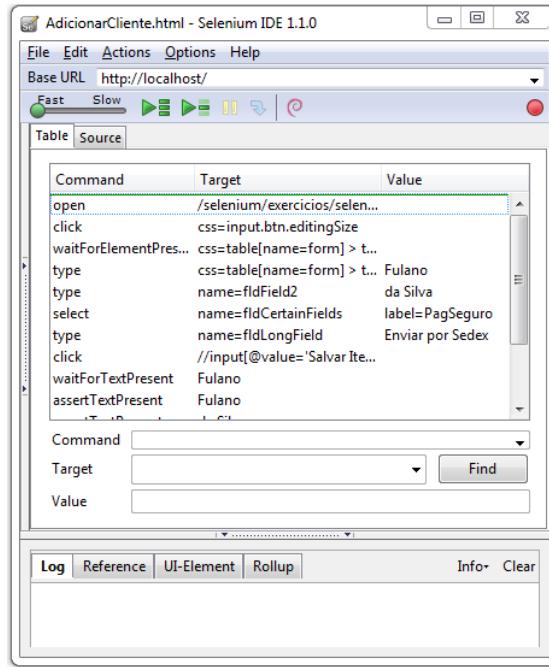
- Flow Control/Side Flow
  - <https://github.com/darrenderidder/sideflow>
- IncludeCommand4IDE
  - [http://wiki.openqa.org/download/attachments/283/includeCommand4IDE\\_1\\_1.zip](http://wiki.openqa.org/download/attachments/283/includeCommand4IDE_1_1.zip)
- DataDriven
  - [http://wiki.openqa.org/download/attachments/14975018/datadriven\\_v0.2.zip](http://wiki.openqa.org/download/attachments/14975018/datadriven_v0.2.zip)

# Configurações do DataDriven

## 1) Utilizaremos o exemplo do AjaxCrud

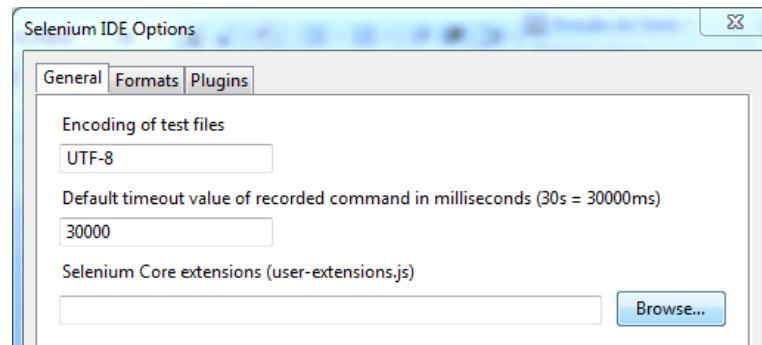
Efetue a automação de um script de inclusão de um cliente ou utilize o arquivo **AdicionarCliente.html**

O script básico é o ponto de partida, pois precisaremos atualizá-lo



## 2) Adicionando os arquivos .js no Selenium

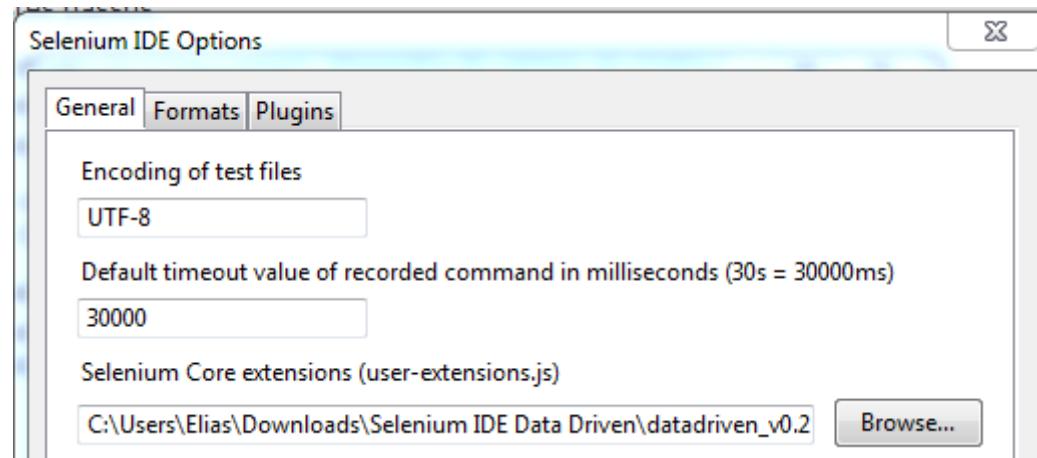
Vá até o menu **Options/Options** e clique sobre o botão **Browse...** para **Selenium Core extensions (user-extensions.js)**



### 3) Adicionando os arquivos .js no Selenium

Selecione os seguintes arquivos :

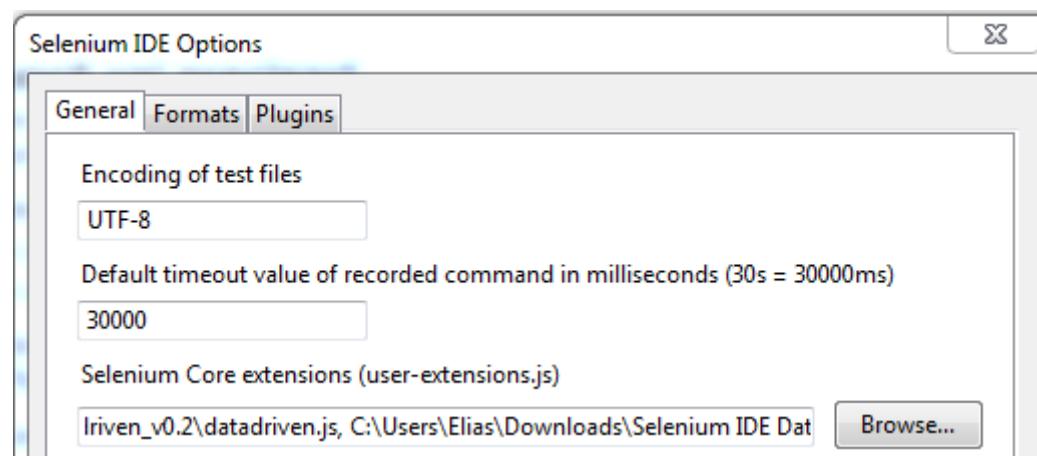
- **datadriven\_v0.2\datadriven**
- **includeCommand4IDE\_1\_1\includeCommand4IDE.js**
- **sideflow.js**



### 4) Validando a inserção dos arquivos

Após ter adicionado os três arquivos .js feche o Selenium e abra-o novamente.

Nenhum erro pode ocorrer e o caminho para todos os arquivos estão separados por vírgula



## 5) Criando o arquivo XML

Um arquivo XML deve ser criado para que seja possível ligar com o script em HTML do Selenium.

Um exemplo do arquivo para o script **AdicionarCliente.html** está ao lado

A tag **<testdata>** representa um conjunto de dados. Não pode ter seu nome alterado. No final do arquivo fechamos a tag **</testdata>**

Dentro de um **<testdata>** podem haver diversos **<test>**. Eles representam os registros da massa de dados. Dentro do **<test>** colocamos as propriedades necessárias, que viram automaticamente variáveis no Selenium, onde as utilizamos através de  **\${propriedade}**  
Todas as propriedades de cada teste devem ter o mesmo nome

Observações:

- Só pode haver um **<testdata>**
- Pode haver propriedades a mais em cada **<test>** mas não de menos
- É recomendado um arquivo .xml (massa de dados) para cada script HTML

```
<testdata>
```

```
 <test nome="Fulano" sobrenome="da Silva" pagamento="PagSeguro" observacao="Envio por Sedex" />
 <test nome="Ciclano" sobrenome="Santos" pagamento="PayPal" observacao="Envio PAC" />
 <test nome="Deltrano" sobrenome="Matos" pagamento="Cartao de Credito" observacao="Envio Sedex" />
```

```
</testdata>
```

## 6) Alterando o script HTML

O script de teste deve ser alterado para suportar o data driven.

Ao lado estão cada linha que deve ser colocada no Selenium IDE

1 linha: deve ser colocada a chamada do arquivo de massa de dados  
*loadTestData | file:///C:/arquivo.xml*

Apos o comando open: deve ser colocado o inicio do laço ate que não exista mais dados

*while | !testdata.EOF()*

Apos o comando while: deve ser colocado o comando que pegara o próximo registro da massa de dados

*nextTestData*

Substituir os dados fixos pelas propriedades do <test> no arquivo de massa de dados: para cada propriedade, substituir os dados fixos. Ex:

*Type | nomePessoa | \${nome}*

Final do arquivo: deve ser colocado a finalização do laço

*endWhile*

## 7) Exemplo do script

Ao lado podemos visualizar o script alterado para o suporte ao Data Driven

The screenshot shows the Selenium IDE interface with a recorded script titled "AdicionarCliente.html". The script uses a data-driven approach, as indicated by the parameterized values in the "Value" column.

Command	Target	Value
loadTestData	file:///C:/Qualister/Scripts_Selenium/...	
open	/selenium/exercicios/selenium_comp...	
while	!testdata.EOF()	
nextTestData		
click	css=input.btnAddSize	
waitForElementPresent	css=table[name=form] > tbody > tr ...	
type	css=table[name=form] > tbody > tr ...	
type	name=fldField2	
select	name=fldCertainFields	
type	name=fldLongField	
click	//input[@value='Salvar Item']	
waitForTextPresent		
assertTextPresent		
endWhile		

In the "Value" column, several entries are replaced by parameters: \${nome}, \${sobrenome}, \${pagamento}, and \${observacao}. These parameters are highlighted with blue boxes, indicating they are data-driven variables.

# Data Driven com Selenium IDE

- Exemplo funcional em **Treinamento Selenium Avancado**  
**\Scripts\_Selenium\_IDE\_DDT\**
- Você irá utilizar para executar o exemplo:
  - AdicionarCliente.html
  - AdicionarCliente.xml
  - user-extensions.js

# Data Driven com Selenium IDE

- Para a execução no DDT com Selenium IDE por linha de comando é necessário:
  - Criar um arquivo user-extensions.js
  - O arquivo deve conter os dados dos plugins, nesta seqüência
    - FlowControl
    - IncludeCommand4IDE
    - DataDriven
- A execução por linha de comando deve ter a seguinte sintaxe

```
|-----|  
| java -jar servidor_selenium.jar -userExtension user-  
| extensions.js htmlSuite *browser "http://pagina" "C:\pasta  
| \suite.html" "C:\pasta\resultado.html"  
|-----|
```

# Data Driven com Selenium IDE

- Em um exemplo onde:
  - **servidor\_selenium** = C:\selenium-server-standalone-2.3.0.jar
  - **Local da user-extensions.js** = C:\script
  - **Local da Suite** = C:\script
  - **Local onde o resultado sera salvo** = C:\script

```
-----  
java -jar selenium-server-standalone-2.3.0.jar -userExtensions C:\script\user-  
extensions.js -htmlSuite *firefox "http://eliasnogueira.com" "C:\script  
\Suite.html" "C:\script\resultado.html"  
-----
```

# Pesquisa em Paginação com Selenium IDE

- É totalmente possível pesquisar por um determinado registro dentro em uma tabela com paginação com o Selenium IDE
- O que será passado, além do exemplo funcional, é a lógica de como fazer isso para qualquer tabela com paginação
- Exemplo funcional em Qualister  
`\Treinamento_Selenium_Avancado\Scripts_Selenium_IDE_DDT  
\paginacao.html`

# Pesquisa em Paginação com Selenium IDE

- Lógica

**Gravação do Script**

- Deve conter o open da pagina apenas

**Criação das Variáveis**

- Criação de todas as variáveis necessárias como: valor pesquisado, contador, paginação e local do valor

**Verificações**

- Controle se o valor pesquisado esta presente, senão continua o laço navegando na paginação ou executando a ação necessária

**Navegação**

- Navegação entre a paginação

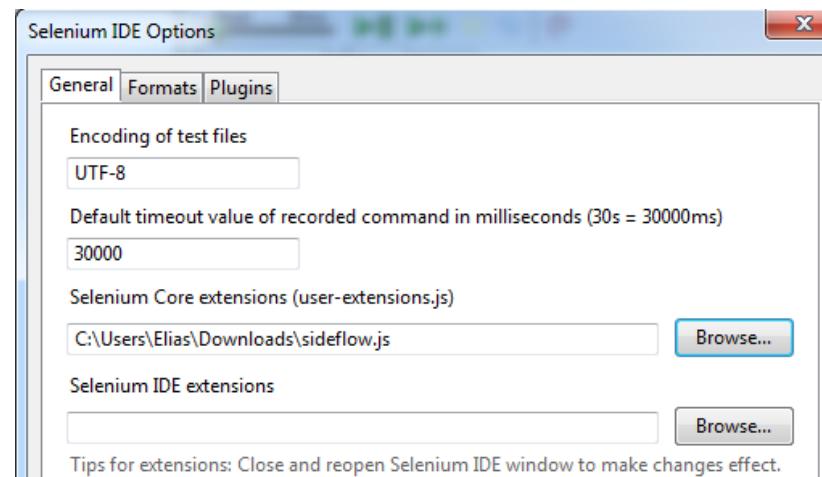
**Ação**

- Comando que deve ser executado caso o valor pesquisado esteja presente

## 1) Adicionar a extensão Flow Control

Adiciona o arquivo **sideflow.js** no **Selenium Core extension** através do menu **Option/Option** no Selenium.

Clique no botão OK, feche o Selenium e depois abra-o novamente



## 2) Gravar o script base

Vamos utilizar o exemplo **Data Driven** para remover um item da lista.

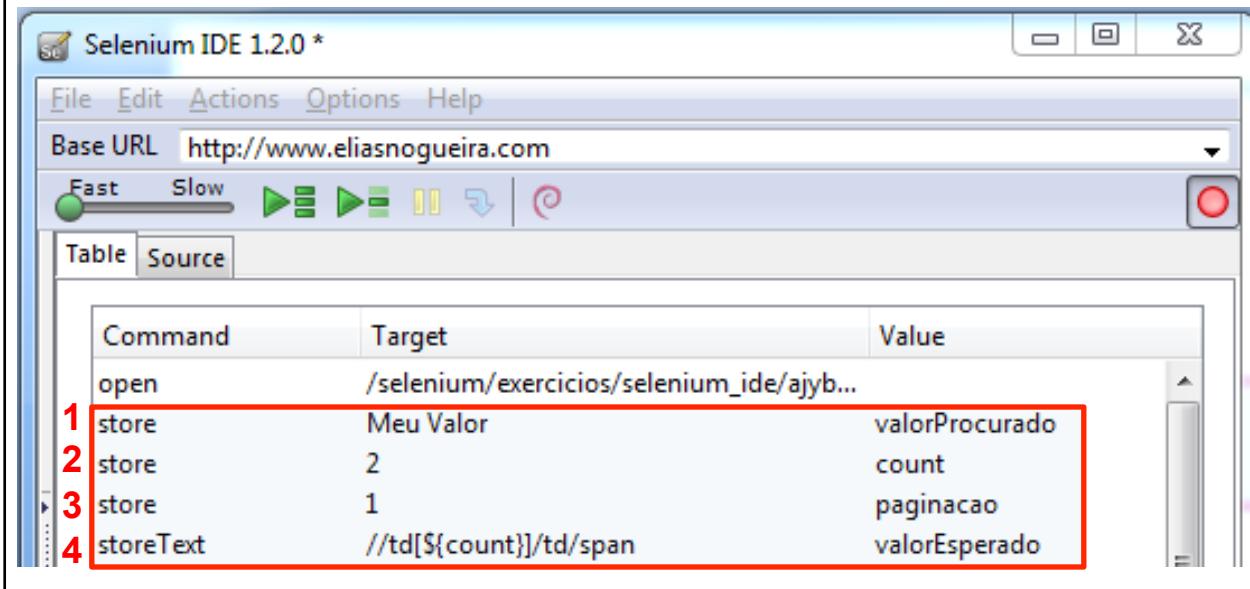
Execute pelo menos estes passos:

- Clicar na paginação
- Selecionar o nome e usar o comando **assertText** com o **Target** para o xPath
- Clicar no botão **delete** e com o **Target** para o xPath

Command	Target
click	link=2
assertText	//tr[4]/td/span
click	//tr[4]/td[5]/input
assertConfirmation	Tem certeza que você deseja re...

### 3) Adicionar as variáveis

Adicione as variáveis necessárias para o funcionamento da pesquisa do valor



Command	Target	Value
open	/selenium/exercicios/selenium_ide/ajybz...	
1 store	Meu Valor	valorProcurado
2 store	2	count
3 store	1	paginacao
4 storeText	//td[{\$count}]/td/span	valorEsperado

**Comando 1:** guardará o valor que pesquisaremos dentro da tabela

**Comando 2:** guardará o numero inicial da linha. O valor no exemplo está como 2 porque a primeira linha da tabela é o cabeçalho

**Comando 3:** guardará o número de cada item da paginação (ex: 1 - 2 - 3 - 4) para termos a referência do clique na próxima paginação da tabela

**Comando 4:** guardará o valor esperado que estará na tabela. Note que é o mesmo xPath do `assertText`, porém ao invés da posição fixa, colocaremos o contador referente a linha da tabela. Logo o script ira procurar na linha 2, 3, 4, e assim por diante

#### 4) Verificações

Faz todas as verificações necessária, sendo:

- enquanto não encontra
- se encontrou
- se leu as 3 linhas da tabela

Command	Target	Value
assertText	//tr[4]/td/span	Bart
1 label	laco	
2 while	storedVars["valorProcurado"] != storedVars["valorEsperado"]	
3 storeText	//tr[\${count}]/td/span	valorEsperado
4 gotof	storedVars["valorProcurado"] == storedVars["valorEsperado"]	clica
5 gotoif	storedVars["count"] > 3	pula
6 store	javascript{storedVars["count"]++}	
7 endwhile		

**Comando 1:** rotulo para identificar bloco de execução

**Comando 2:** loop de execução enquanto o **valorProcurado** for diferente do **valorEsperado**

**Comando 3:** pega o texto do local (xPath) e joga na variável **valorEsperado**

**Comando 4:** Se o **valorProcurado** for igual ao **valorEsperado** vai para o rotulo **clica** (se encontrou)

**Comando 5:** Se o contador de linhas da tabela for maior que três vai para o rotulo **pula** (próxima paginação)

**Comando 6:** Incrementa o contador de linhas da tabela

**Comando 7:** Finaliza o laço while

## 5) Navegação e Ação

Faz todas as verificações necessária, sendo:

- enquanto não encontra
- se encontrou
- se leu as 3 linhas da tabela

Command	Target
store	javascript{storedVars["count"]++}
endWhile	
<b>1</b> label	pula
<b>2</b> store	javascript{storedVars["paginacao"]++}
<b>3</b> click	link=\${paginacao}
<b>4</b> store	javascript{storedVars["count"] = 2}
<b>5</b> gotolabel	laco
<b>6</b> label	clica
<b>7</b> click	//tr[\${count}]/td[5]/input
<b>8</b> assertConfirmation	Tem certeza que você deseja remover este item?

**Comando 1:** bloco de execução chamado **pula**, quando um **gotolf** vai se a expressão for verdadeira

**Comando 2:** incrementa a variável de paginação

**Comando 3:** clica na próxima paginação, onde o link é o número incrementado

**Comando 4:** volta o contador com o valor original (2 porque 1 é a linha de cabeçalho da tabela)

**Comando 5:** bloco de execução chamado **laco**, quando um **gotolf** vai se a expressão for verdadeira

**Comando 6:** clica no botão **delete** referente a linha do **valorProcurado**

**Comando 7:** garante que a confirmação da remoção apareça e clique em OK

A photograph of six people (three men and three women) standing behind a large white rectangular sign. They are all smiling and looking towards the camera. The sign is blank, except for the text "Selenium 1 x Selenium 2 Transição" which is printed on it. The background is plain white.

# **Selenium 1 x Selenium 2**

## **Transição**

# Selenium 1 x Selenium 2

- Com a chegada do Selenium 2 (WebDriver) os scripts em linguagem de programação sofreram uma grande mudança de arquitetura
- Como principais melhorias temos
  - Remoção da dependência do Server
  - Browsers independentes
  - Arquitetura em PageObjects
  - Comandos específicos para certos elementos
  - Maior velocidade na execução

# Principais Mudanças (transição)

Selenium 1	Selenium 2
<b>Instanciando o Browser</b>	
<pre>Selenium selenium = new DefaultSelenium("localhost", 4444, "*firefox", "http://www.meusite.com"); selenium.start();</pre>	<pre>WebDriver driver = new FirefoxDriver(); driver.get(" http://www.meusite.com.br");</pre>
<b>Acessando um Elemento HTML</b>	
<pre>selenium.ACAO ("name=inserir");</pre>	<pre>driver.findElement(...)</pre>
<pre>selenium.click("id=inserir");</pre>	<pre>//id</pre> <pre>driver.findElement(By.id("inserir"))</pre>
<pre>selenium.click ("name=inserir");</pre>	<pre>//name</pre> <pre>driver.findElement(By.name("inserir"))</pre>
<pre>selenium.click ("//input[@name="inserir"]);</pre>	<pre>//xpath</pre> <pre>driver.findElement(By.xpath("//input[@name="inserir"]"))</pre>
<b>Interagindo com Combobox</b>	
<pre>selenium.select("name=tipo", "label=Profissional - Paga");</pre>	<pre>Select select = new Select(driver.findElement(By.name("tipo"))); select.selectByVisibleText("Profissional - Paga");</pre>
<b>Interagindo com ListBox</b>	
<pre>selenium.addSelection("name=interesse[]", "label=Teste de Software");</pre>	<pre>Select select = new Select(driver.findElement(By.name("interesse[]"))); select.selectByVisibleText("Teste de Software");</pre>

# Principais Mudanças (transição)

Selenium 1	Selenium 2
	<b>Preenchendo campos (inputs)</b>
selenium.type("name=nome", "Elias");	driver.findElement(By.name("nome")).sendKeys("Elias");
	<b>Submetendo dados de um formulario</b>
selenium.click("name=submit");	driver.findElement(By.name("submit")).click(); driver.findElement(By.name("submit")).submit();
	<b>Capturando texto</b>
selenium.getText("id=resposta")	driver.findElement(By.id("resposta")).getText()
	<b>Capturando screenshots</b>
selenium.captureEntirePageScreenshot("C:\arquivo.png");	File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE); FileUtils.copyFile(scrFile, new File("C:\arquivo.png"));
	<b>Interagindo com alertas</b>
selenium.getAlert();	Alert alert = driver.switchTo().alert(); String texto = alert.getText();
	<b>Interagindo com confirmações</b>
selenium.getConfirmation(); chooseCancelOnNextConfirmation();	Alert alert = driver.switchTo().alert(); alert.dismiss();

# Principais Mudanças (transição)

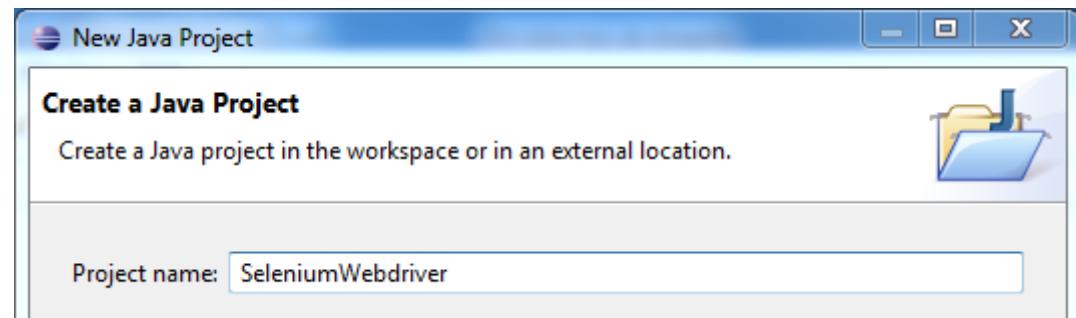
Selenium 1	Selenium 2
<b>Executando Javascript</b>	
selenium.getEval( <b>meuJS()</b> );	JavascriptExecutor js = (JavascriptExecutor) driver; js.executeScript( <b>"meuJS()"</b> );
<b>VerifyTextPresent</b>	
selenium.verifyTextPresent( <b>"texto"</b> );	String dado = driver.findElement(By.tagName( <b>"body"</b> )).getText(); if (dado.contains( <b>"texto"</b> )) { //executa algo }
<b>Esperas</b>	
for (int second = 0; second++ < 60) { if (second >= 60) fail( <b>"timeout"</b> ); try { if (selenium.getText( <b>"texto"</b> )) break; } catch (Exception e) {} Thread.sleep(1000); }	Topico Waits com Selenium 2

# Exercício Transição Selenium

- Iremos exportar os arquivos do exercício sobre Elementos HTML e depois transformar os scripts do Selenium 1 para Selenium 2
- Exemplo funcional em **Treinamento Selenium Avancado\Projetos\_Java\SeleniumWebDriver**

## 1) Nome do Projeto

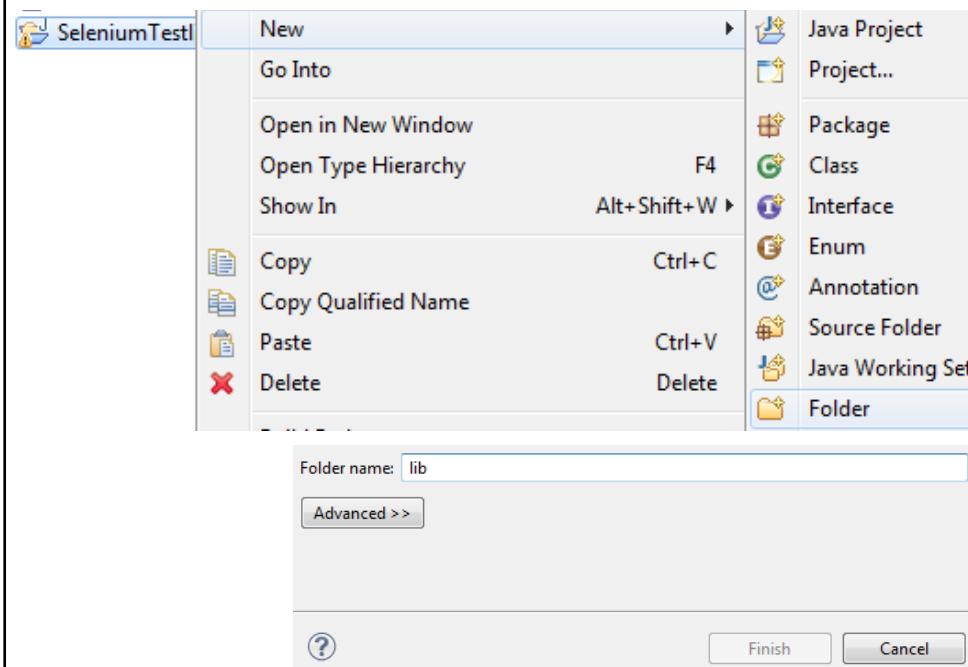
No campo *Project name* coloque o nome do projeto como “*SeleniumWebdriver*” e clique no botão **Finish**



## 2) Criar pasta lib e adicionar as bibliotecas

Agora clique com o botão direito sobre o nome do projeto e selecione o menu **New/Folder**

Na tela **New Folder** coloque no campo *Folder name* o nome *lib*



### 3) Copiar as bibliotecas para o projeto

Em nosso projeto de exemplo as bibliotecas já estão adicionadas, porém os links de cada biblioteca serão listadas ao lado.

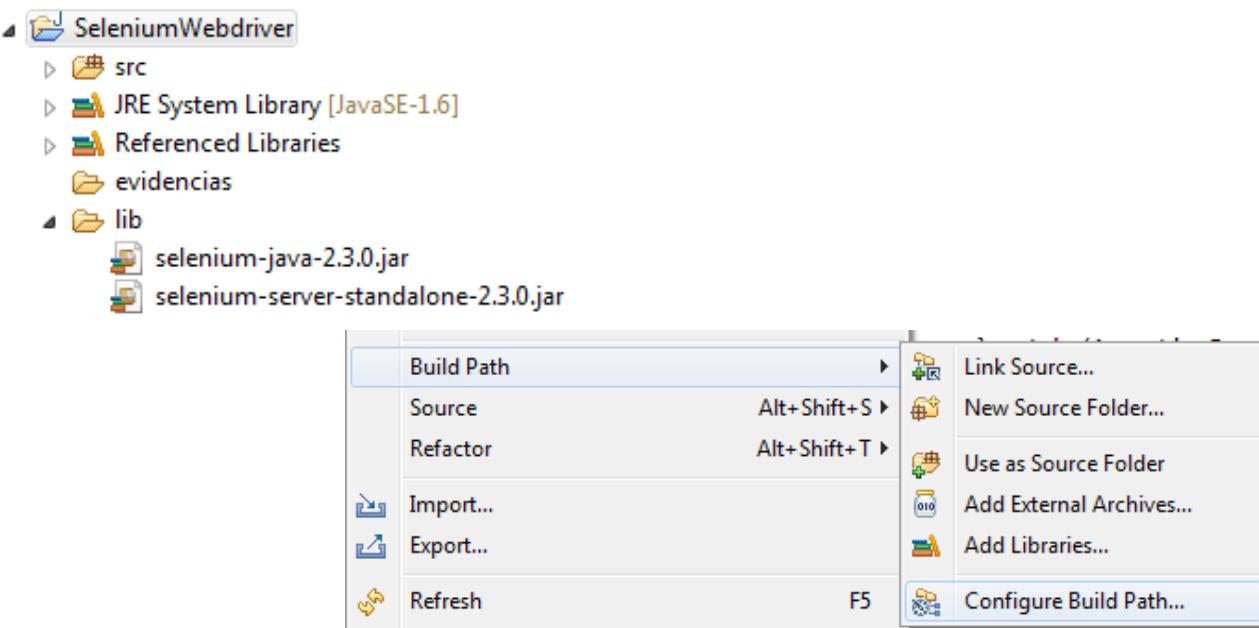
Os arquivos .jar devem ser copiados para **a pasta do usuário/workspace**

- Selenium Server
- Selenium Java Client Driver

### 4) Atualizando e adicionando as bibliotecas

Após o download e a copia para a pasta **lib** clique bom o botão direito sobre o projeto e selecione **Refresh**.

Após isso clique bom o botão direito novamente e selecione **Build Path/Configure Build Path**

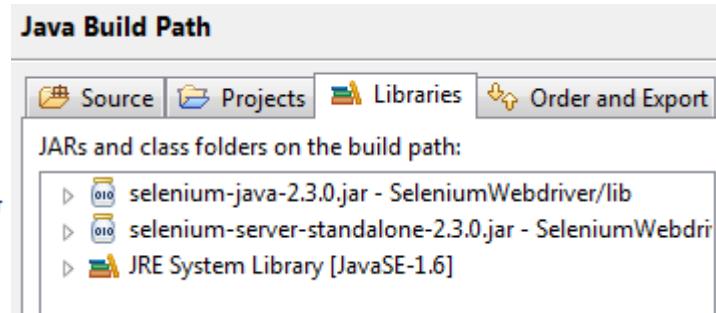
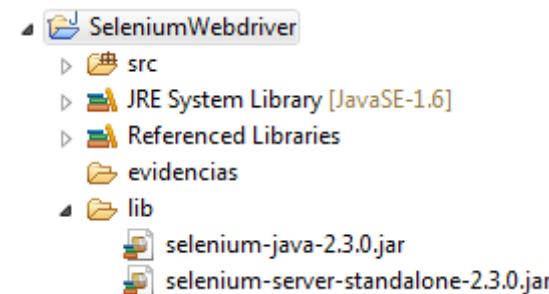
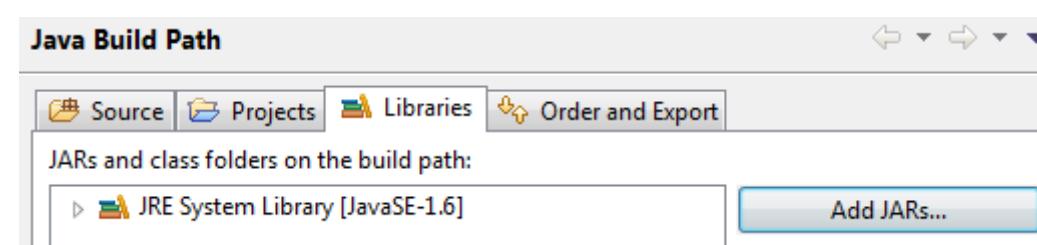


## 5) Adicionando as bibliotecas

Clique sobre a aba *Libraries* e depois sobre o botão *Add Jars...*

Na tela apresentada selecione o projeto e va até a pasta *lib*. Selecione todos os arquivos listados e clique em OK

Após isso todos os arquivos estarão listados na aba *Libraries*. Clique no botão OK

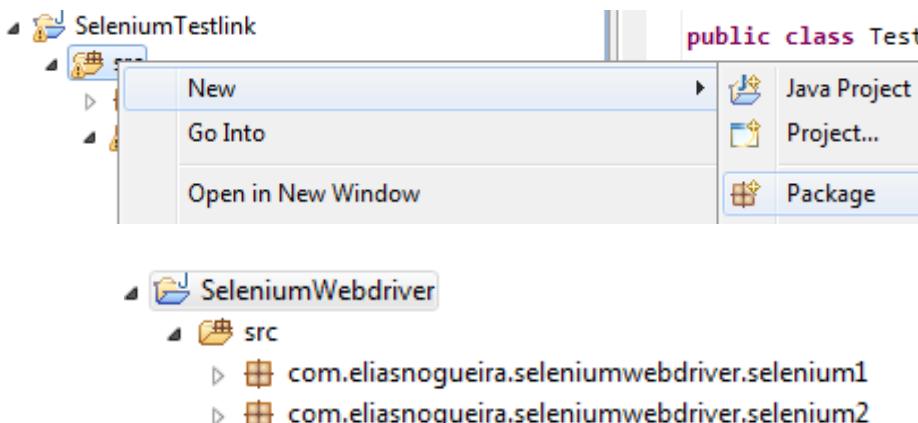


## 6) Criação do pacote

Vá até a pasta *src*, clique bom o botão direito e selecione *Package*.

Em *Name* coloque qualquer nome de sua escolha. No exemplo nome utilizado é *com.eliasnogueira.seleniumwebdriver.selenium1*

Clique no botão *Finish*

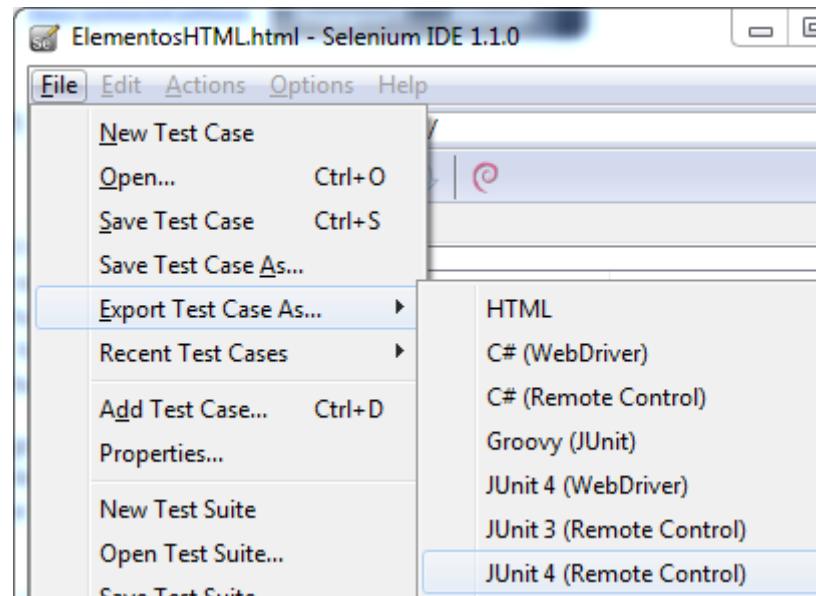


## 7) Exportando o script do Selenium IDE

Abra o script do exercicio Elementos HTML.

Selecione o menu **File/Export Test Case As.../Junit 4 (Remote Control)**

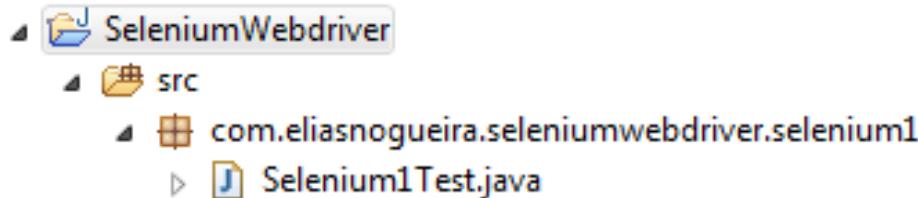
Salve o script com o nome:  
**Selenium1Teste.java** no Desktop



## 8) Adicionando o script

Va ate o Desktop, clique e arraste o arquivo

**Selenium1Test.java** para o Eclipse soltando no pacote  
**com.eliasnogueira.seleniumwebdriver.selenium1**



**Observação:** o script exportado necessita que iniciemos o Selenium RC por linha de comando. Faremos online a adição do código para não precisar iniciar o Selenium RC por linha de comando.

O script pronto pode ser visualizada na sua pasta de projetos (download de arquivos do curso)

## 9) Criação do pacote do classe do Selenium 2

Vá até a pasta *src*, clique bom o botão direito e selecione **Package**.

Em **Name** coloque qualquer nome de sua escolha. No exemplo nome utilizado é **com.eliasnogueira.seleniumwebdriver.selenium2**

Clique no botão **Finish**

Agora clique com o botão direito sobre o pacote criado e selecione **New/Class**

Coloque o nome **Selenium2Teste** e clique em **Finish**

```
@Test
public void testSelenium1() throws Exception {
    driver.get("http://eliasnogueira.com/selenium/exercicios/selenium_ide/ajybuyje/avanca");

    driver.findElement(By.name("nome")).sendKeys("Elias");
    driver.findElement(By.name("email")).sendKeys("elias.nogueira@gmail.com");
    driver.findElement(By.name("senha")).sendKeys("123456");
    driver.findElement(By.name("newsletter")).click();
    driver.findElement(By.xpath("//input[@name='formato' and @value='texto']")).click();

    Select select = new Select(driver.findElement(By.name("tipo")));
    select.selectByVisibleText("Profissional - Paga");

    select = new Select(driver.findElement(By.name("interesse[]")));
    select.selectByVisibleText("Teste de Software");
    select.selectByVisibleText("Gerencia PMI/Agil");

    driver.findElement(By.name("comentario")).sendKeys("Nao quero receber spam");

    File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
    FileUtils.copyFile(scrFile, new File("C:\\\\Users\\\\Elias\\\\workspace\\\\SeleniumWebdriver\\\\"));

    driver.findElement(By.name("submit")).click();

    FileUtils.copyFile(scrFile, new File("C:\\\\Users\\\\Elias\\\\workspace\\\\SeleniumWebdriver\\\\"));

    assertEquals("Caro, Elias\\nO email cadastrado foi: elias.nogueira@gmail.com\\n\\nVocê ni
        driver.findElement(By.id("resposta")).getText());
}
```

Iremos criar este código ao-vivo, porém o código já pronto pode ser visualizado no download do projeto **SeleniumWebdriver**, pasta *src*, pacote **com.eliasnogueira.seleniumwebdriver.selenium2**

Basicamente podemos usar a tabela de transição (slides anteriores)

A photograph of six business professionals—three men and three women—standing in a row against a white background. They are all smiling and holding a large, blank white rectangular sign between them. The sign is positioned in front of the middle two individuals.

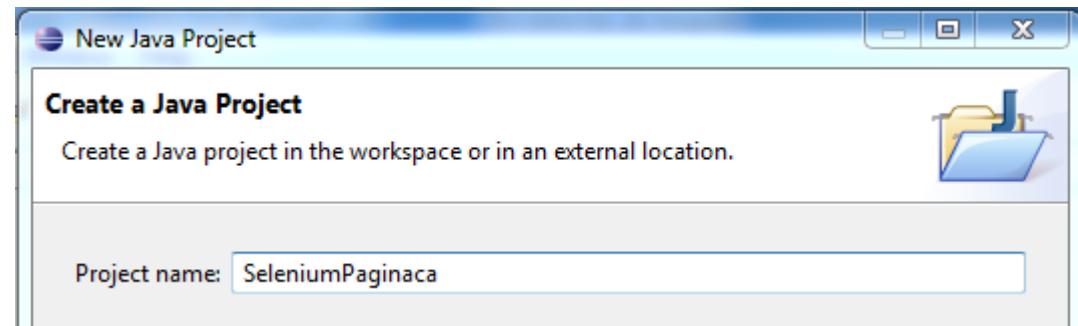
# Pesquisa em Paginação com WebDriver

# Pesquisa em paginação com WebDriver

- Funciona na mesma lógica da pesquisa em paginação com o Selenium IDE, porém com menos código
- Exemplo funcional em **Treinamento Selenium Avancado\Projetos\_Java\Selenium\_Paginacao**
- Sabemos que no exercício de paginação
  - Teremos que ter um contador iniciando em 2, porque o cabeçalho é a primeira linha
  - Teremos que ter um contador para o link da paginação
  - Teremos que fazer o loop para encontrar o valor desejado

## 1) Nome do Projeto

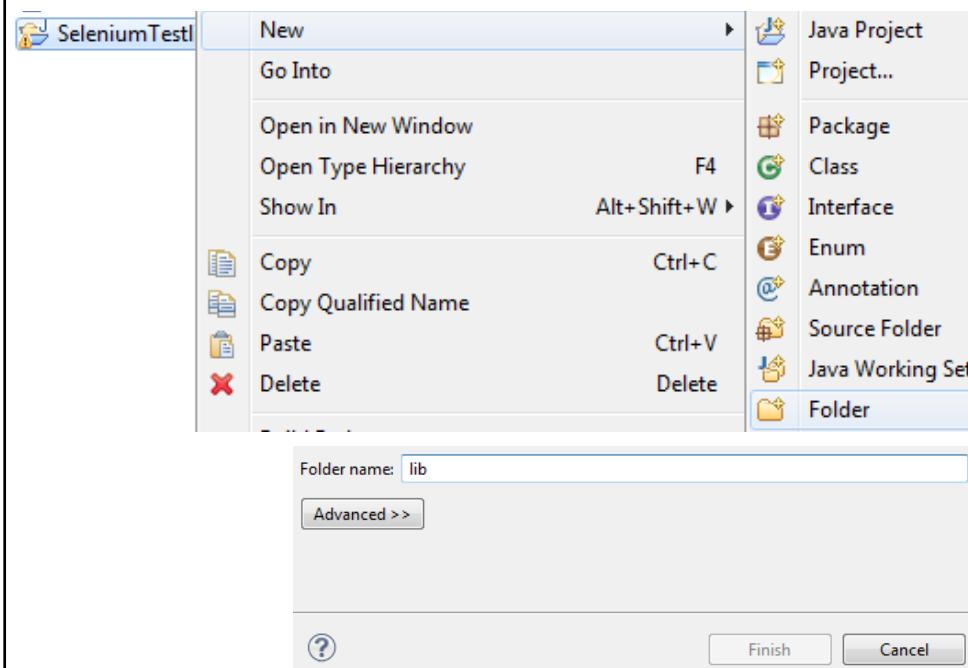
No campo *Project name* coloque o nome do projeto como “*SeleniumPaginacao*” e clique no botão **Finish**



## 2) Criar pasta lib e adicionar as bibliotecas

Agora clique com o botão direito sobre o nome do projeto e selecione o menu **New/Folder**

Na tela **New Folder** coloque no campo *Folder name* o nome *lib*



### 3) Copiar as bibliotecas para o projeto

Em nosso projeto de exemplo as bibliotecas já estão adicionadas, porém os links de cada biblioteca serão listadas ao lado.

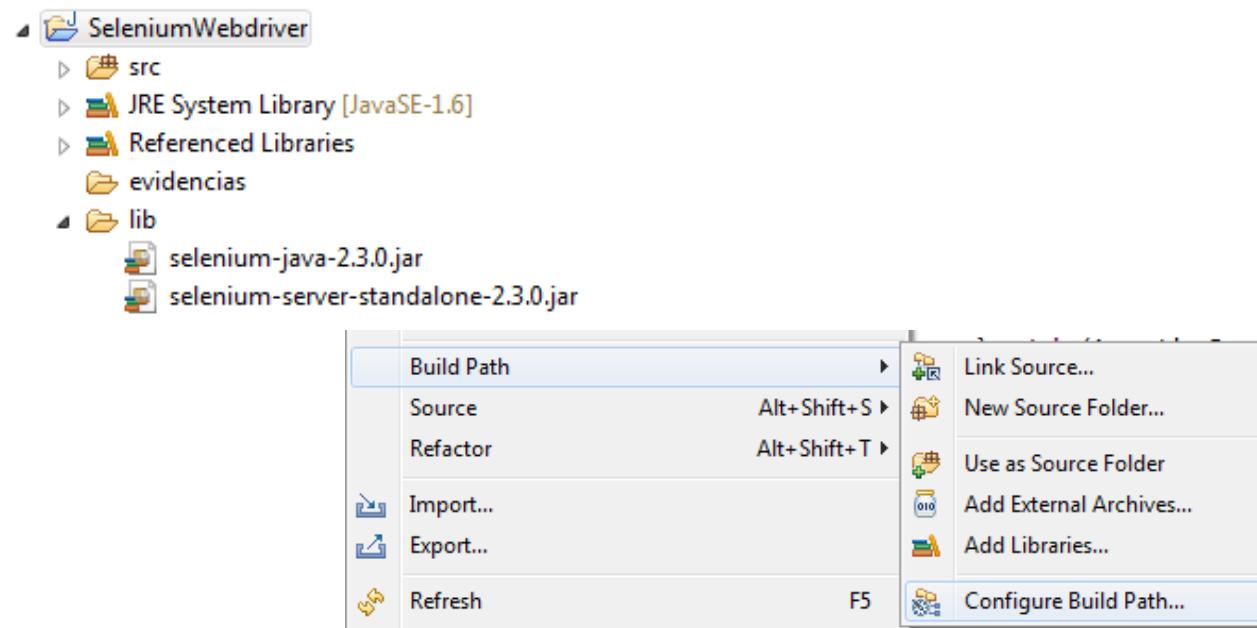
Os arquivos .jar devem ser copiados para **a pasta do usuário/workspace**

- Selenium Server
- Selenium Java Client Driver

### 4) Atualizando e adicionando as bibliotecas

Após o download e a copia para a pasta **lib** clique bom o botão direito sobre o projeto e selecione **Refresh**.

Após isso clique bom o botão direito novamente e selecione **Build Path/Configure Build Path**

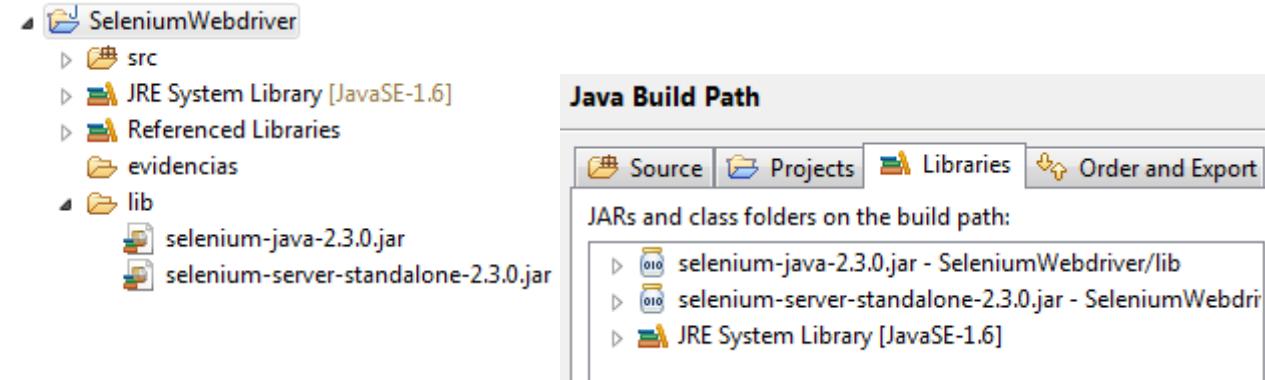
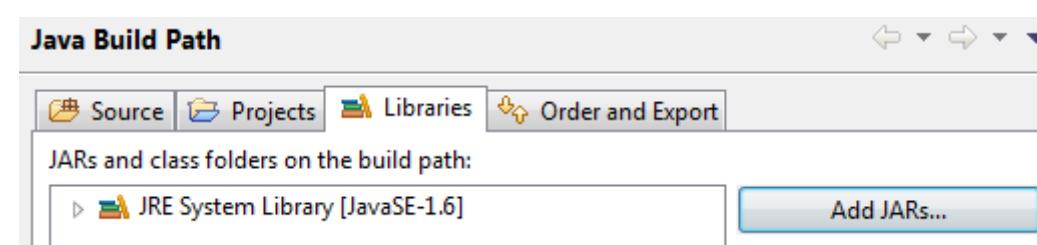


## 5) Adicionando as bibliotecas

Clique sobre a aba *Libraries* e depois sobre o botão *Add Jars...*

Na tela apresentada selecione o projeto e va até a pasta *lib*. Selecione todos os arquivos listados e clique em **OK**

Após isso todos os arquivos estarão listados na aba *Libraries*. Clique no botão **OK**

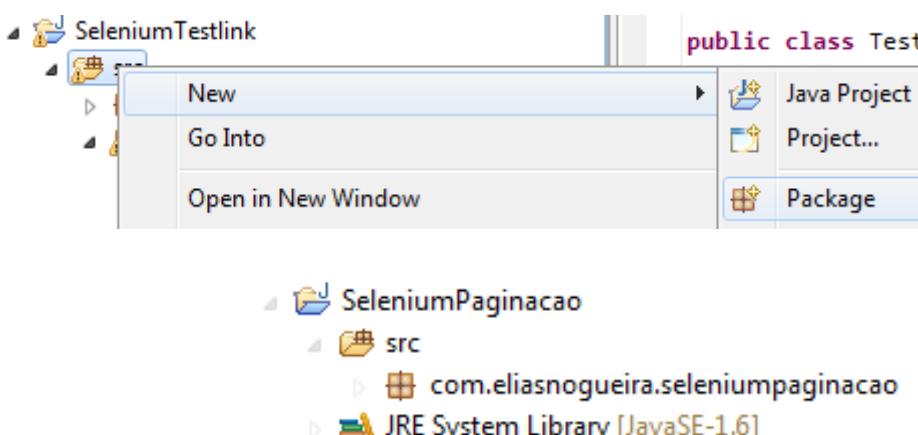


## 6) Criação do pacote

Vá até a pasta *src*, clique bom o botão direito e selecione *Package*.

Em *Name* coloque qualquer nome de sua escolha. No exemplo nome utilizado é *com.eliasnogueira.seleniumpaginacao*

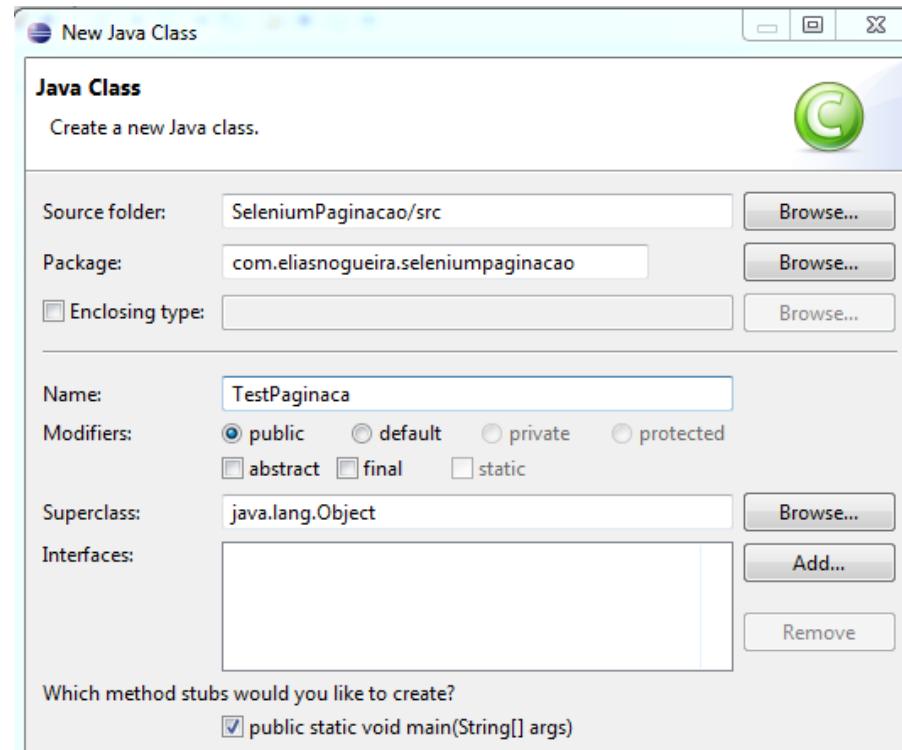
Clique no botão **Finish**



## 7) Criando a classe de teste

Clique com o botão direito sobre a package criada anteriormente e selecione **New/Class**

Coloque o nome de **TestePaginacao**, marque o item **public static void main (String[] args)** e clique em **Finish**



## 8) Criando as variáveis iniciais

No inicio do código cria a instancia do WebDriver e a abertura da pagina do exemplo

```
public static void main(String[] args) throws InterruptedException {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.eliasnogueira.com/selenium/exercicios/selenium_ide/");
```

## 9) Variáveis de controle

É necessário criar as variáveis de controle para o loop e paginação

**String nome** = nome que utilizaremos como base para encontrá-lo

**int count** = contador das linhas da tabela, que inicia em 2 porque a linha 1 é o cabeçalho

**int paginacao** = contador do link de paginação

**String localizacaoNome** = variável para guardar da linha onde o script estiver percorrendo

**boolean saida** = controle para sair do loop

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.eliasnogueira.com/selenium/
    
        // variaveis iniciais
    String nome = "Teste 6";
    int count = 2;
    int paginacao = 1;
    String localizacaoNome = null;
    boolean saida = false;
```

## 9) Loop inicial

Utilizaremos um laço **do-while** no script.

O inicio do script não terá nenhum controle, são a verificação se o nome encontrado é igual ao que esperamos.

Logo abaixo do comando **do** temos a localização de um elemento, que é a linha onde se encontra o nome. Dentro da expressão há o contador para poder ler o próximo nome enquanto não o encontra

Também há a flag que informará que poderemos sair do laço

```
do {  
    localizacaoNome = driver.findElement(By.xpath("//tr[" + count + "]/td/span")).getText();  
  
    // se encontrou o nome para remover clica no botao 'remover'  
    if (nome.equals(localizacaoNome)) {  
        driver.findElement(By.xpath("//tr[" + count + "]/td[5]/input")).click();  
        Alert alerta = driver.switchTo().alert();  
  
        if (alerta.getText().equals("Tem certeza que voce deseja remover este item?")) alerta.accept();  
  
        saida = true; // seta o final do while  
    }  
}
```

## 10) Lógica de paginação

Se o nome procurado não for igual ao que ele pegou do elemento na primeira iteração (count = 2) teremos que ter o bloco **else** para controlar a paginação.

Primeiro incrementamos o contador

Depois, através de um **if**, verificamos que ele é maior que 4, porque só existem 3 linhas em cada paginação

Caso o script tenha lido as 3 linhas (count > 4) teremos

Zera o contador passando ele para o valor 2 novamente

Incrementamos o contador da paginação

Clicamos sobre o link da paginação utilizando o contador como valor

Ele voltará ao loop até encontrarmos (ou não) o valor. Logo **while** será executado até que tenhamos encontrado o nome. Depois disso fechamos o browser e a pesquisa é encerrada com sucesso.

```
        } } else {  
            count++;  
  
            // se já passou pelos três elementos na tela clica na próxima paginação  
            if (count > 4) {  
                count = 2;  
                paginacao++;  
                driver.findElement(By.linkText(String.valueOf(paginacao))).click();  
            }  
        }  
    } while (!saida);  
  
    driver.close();  
}
```

## 11) Script completo

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.eliasnogueira.com/selenium/exercicios/selenium_ide/ajybuyje/avancado/datadriven/");

    // variaveis iniciais
    String nome = "Teste 6";
    int count = 2;
    int paginacao = 1;
    String localizacaoNome = null;
    boolean saida = false;

    do {
        localizacaoNome = driver.findElement(By.xpath("//tr[" + count + "]/td/span")).getText();

        // se encontrou o nome para remover clica no botao 'remover'
        if (nome.equals(localizacaoNome)) {
            driver.findElement(By.xpath("//tr[" + count + "]/td[5]/input")).click();
            Alert alerta = driver.switchTo().alert();

            if (alerta.getText().equals("Tem certeza que voce deseja remover este item?")) alerta.accept();

            saida = true; // seta o final do while
        }

        } else {
            count++;

            // se ja passou pelos tres elementos na tela clica na proxima paginacao
            if (count > 4) {
                count = 2;
                paginacao++;
                driver.findElement(By.LinkText(String.valueOf(paginacao))).click();
            }
        }
    } while (!saida);

    driver.close();
}
```

A photograph of six people (three men and three women) standing behind a large, blank white rectangular sign. They are all smiling and looking towards the camera. The background is plain white.

**Waits no WebDriver**

# WebDriver Waits

- Anteriormente, no Selenium 1, um *waitFor* transformado em Java e um for com uma thread
- No WebDriver existem duas formas de efetuar waits:
  - Implícita: faz espera por qualquer elemento por uma quantidade x de segundos
  - Explicita: quando necessitamos esperar por certa condição

# WebDriver Waits

- Exemplo de **Implicit Wait**

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

- Neste caso o WebDriver irá esperar até 10 segundos em todas as esperas que ocorrerem na página (como um elemento não disponível ainda), funcionando igual ao antigo *waitFor* do Selenium 1

# WebDriver Waits

- Exemplo de **Explicit Wait**

```
(new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {  
    @Override  
    public Boolean apply(WebDriver d) {  
        return driver.findElement(By.tagName("body")).getText().contains("Fulano");  
    }  
});
```

- Quando queremos esperar mais tempo por alguma condição sem alterar a *implicit wait* podemos usar esta estratégia para esperar que a condição retornada seja *true*

# WebDriver Waits - Exercício

- Para este exercício usaremos o exemplo de DataDriven programando todo o script, analisando o comportamento da pagina e utilizando o Firebug descobrir cada elemento

The screenshot shows a web application interface. At the top, there is a search bar labeled "Nome:" with an empty input field. Below the search bar is a table with the following data:

Nome	Sobrenome	Forma de Pagamento	Observacao	Acao
Elias	Nogueira	PagSeguro	Envio por Sedex	<input type="button" value="delete"/>
Fulano	da Silva	PayPal	Enviar por Sedex	<input type="button" value="delete"/>

At the bottom of the table, there is a button labeled "Adicionar Item".

## 1) Criação do pacote do classe do Selenium 2

No projeto **Selenium2Exemplo**, dentro da pasta src e pacote **com.eliasnogueira.selenium2 exemplo.test** contem a classe **AdicionarCliente.java**

```
@Test
public void adicionaCliente() {
    /**
     * Instancia para executar o Selenium no Firefox
     * Basta trocar o FirefoxDriver por outro "BrowserDriver" para executar em outro browser.
     * Ex: InternetExplorerDriver();
     */
    final WebDriver driver = new InternetExplorerDriver();

    // Espera implicita no Selenium para todos os elementos
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

    driver.get("http://eliasnogueira.com/selenium/exercicios/selenium_ide/ajybuyje/avancado/d");
    driver.findElement(By.cssSelector("input.btn.editingSize")).click();
    driver.findElement(By.cssSelector("table[name=form] > tbody > tr > td > input[name=fldFile"));
    driver.findElement(By.name("fldField2")).sendKeys("da Silva");
    new Select(driver.findElement(By.name("fldCertainFields"))).selectByValue("PayPal");
    driver.findElement(By.name("fldLongField")).sendKeys("Enviar por Sedex");
    driver.findElement(By.xpath("//input[@value='Salvar Item']")).submit();

    /**
     * Espera explicita do Selenium por um texto na pagina.
     * Esta espera substitui o comando 'isTextPresent()' no Selenium 1
     */
    (new WebDriverWait(driver, 1)).until(new ExpectedCondition<Boolean>() {
        @Override
        public Boolean apply(WebDriver d) {
            return driver.findElement(By.tagName("body")).getText().contains("Fulano");
        }
    });

    Assert.assertEquals(driver.findElement(By.xpath("//span")).getText(), "Fulano");
    driver.close(); // fecha o browser
}
```

O projeto **Selenium2Exemplo** contem o o codigo de exemplo das esperas implicitas e explicitas

A photograph of six people (three men and three women) standing behind a large, blank white rectangular sign. They are all smiling and looking towards the camera. The sign is held horizontally by the group. The background is plain white.

# Selenium + JUnit

# Selenium + JUnit

- Por padrão no Selenium 1 devemos ter o suporte de algum framework de teste unitário
- No Selenium 2 isso não é mandatório, mas podemos utilizar um de nosso gosto
- Os próximos slides apresentarão como utilizar o Junit com Selenium através do *DDT (Data Driven Testing)* com dados fixos em cada script ou com uma planilha Excel

# Selenium + JUnit

- Para fazer o JUnit executar nossos scripts através de massa de dados fixa teremos que utilizar duas anotações:
  - `@RunWith(Parameterized.class)`
  - `@Parameters`
- Devemos também ter uma método que retorna uma lista de objetos para o teste
- Exemplo funcional em **Treinamento Selenium Avançado\Projetos\_Java\Selenium2JUnit**

# Selenium + JUnit

- Utilizaremos o script Java (*Data Driven*) criado anteriormente para este exercício
- A seguir serão passadas quais as alterações necessárias, de forma genérica, em qualquer script para necessitar de massa de dados fixa em código
- O cenário será o mesmo, onde iremos parametrizar:
  - Nome
  - Sobrenome
  - Pagamento
  - Observação

## 1) Adicionando Anotação na classe

Na classe de teste iremos adicionar a seguinte anotação  
`@RunWith(Parameterized.class)`

```
@RunWith(Parameterized.class)
public class AdicionarCliente {

    private WebDriver driver;
```

## 2) Criar as variáveis para utilização automática do script

E necessário criarmos as variáveis que serão utilizadas automaticamente pelo script. Após a declaração da classe, coloque o seguinte bloco de variáveis:

```
private String nome;
private String sobrenome;
private String pagamento;
private String observacao;
```

```
import java.util.Arrays;

* Adiciona um Cliente
@RunWith(Parameterized.class)
public class AdicionarCliente {

    private WebDriver driver;

    * é necessário criar as variáveis para que os dados da massa de dados sejam inseridos
    private String nome;
    private String sobrenome;
    private String pagamento;
    private String observacao;
```

### 3) Criar um construtor com as variaveis

Também é necessário a criação de um construtor para receber os dados. A ordem que estiver no construtor e a que serão os parâmetros utilizados

Crie o contractor como mostra o código ao lado

```
/**
 * O construtor é necessário para passar todos os dados do método com os dados
 * para a utilização do teste
 */
public AdicionarCliente(String nome, String sobrenome, String pagamento, String observacao) {
    this.nome = nome;
    this.sobrenome = sobrenome;
    this.pagamento = pagamento;
    this.observacao = observacao;
}
```

### 4) Crie o método que preencherá o DDT

Agora é necessário a criação do método padrão com os dados de execução. Note que os dados estão em ordem (nome, sobrenome, pagamento e observação) e devem respeitar sempre esta ordem

Esse método faz com que os dados sejam passados para os parâmetros e construtor na execução do script

```
@Parameters
public static Collection<Object[]> data() {
    return Arrays.asList(new Object[][] {
        {"Elias", "Nogueira", "PayPal", "Envio por Sedex 10"},
        {"Fulano", "da Silva", "PagSeguro", "Envio por encom. normal"},
        {"Deltrano", "Santos", "Cartão de Crédito", "Envio por Sedex"}
    });
}
```

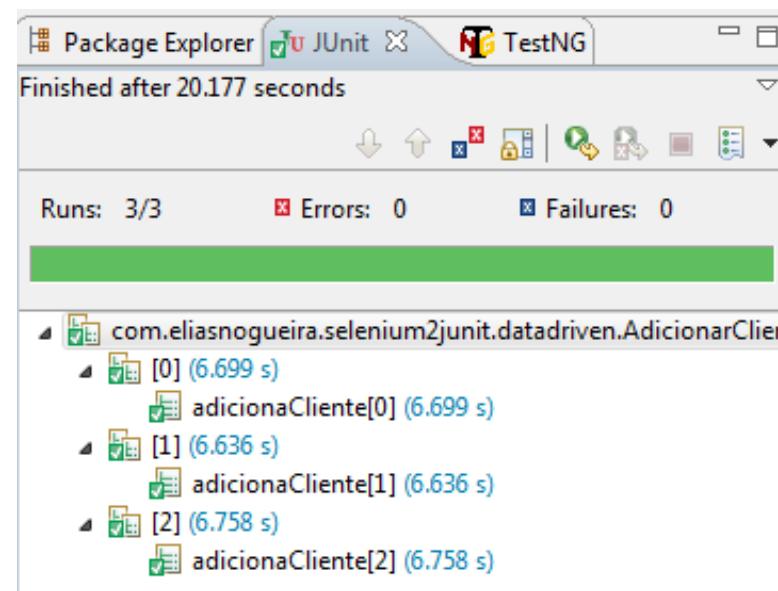
**Observação:** Em cada abertura e fechamento de chaves e que são determinados os dados. Eles sempre devem respeitar a quantidade e mesma ordem. Para passar para o próximo dado temos que colocar uma vírgula depois do fechamento da chave, abrir uma nova chave e inserir os dados.

## 5) Substituir os dados fixos pelas variáveis

Agora basta substituir os dados fixos pelas variáveis no script para que ela possa executar com os dados que inserimos no método anterior.

```
driver.get("http://localhost/selenium/exercicios/selenium_comp/scysnlkw/ajaxCRUD/exemplo/");
driver.findElement(By.cssSelector("input.btn.editingSize")).click();
driver.findElement(By.cssSelector("table[name=form] > tbody > tr > td > input[name=fldField1]")).sendKeys(nome);
driver.findElement(By.name("fldField2")).sendKeys(sobrenome);
new Select(driver.findElement(By.name("fldCertainField"))).selectByValue(pagamento);
driver.findElement(By.name("fldLongField")).sendKeys(observacao);
driver.findElement(By.xpath("//input[@value='Salvar Item']")).submit();
```

O script será executado o numero de vezes que tivermos dados no método. No exemplo abaixo tivemos três execuções do script com sucesso e com dados diferentes



# Selenium + JUnit

- Agora utilizaremos o mesmo modelo de DDT agora com planilha Excel (.xls)
- O modelo praticamente é o mesmo, mudando apenas a chamada para o arquivo Excel
- Neste modelo de DDT a primeira linha já deve conter dados (não ter o cabeçalho), mas pode ser criado a parte

# Selenium + JUnit

- Para que isso seja possível existe no projeto **Selenium2JUnit** uma classe chamada **SpreadsheetData** no pacote **com.eliasnogueira.seleniumjunitddt.util**
- Voce pode criar sua propria classe. A única restricao para utilizacao com DDT e ela retornar uma colecao de objetos

Collection<Object [] >

# Selenium + JUnit

- A classe **AdicionaClienteDDT.java** contém apenas a mudança no método que, ao invés dos dados fixos, chama o arquivo Excel em algum diretório.

```
@Parameters
public static Collection<Object[]> data() throws IOException {
    InputStream spreadsheet = new FileInputStream("massaDados/massa_de_dados.xls");
    return new SpreadsheetData(spreadsheet).getData();
}
```

A photograph of six people (three men and three women) standing behind a large, blank white rectangular sign. They are all smiling and looking towards the camera. The sign is held horizontally by the group. The background is plain white.

**Selenium + TestNG**

# Selenium + TestNG

- O TestNG é um framework mais robusto que o JUnit, mas em essência possui as mesmas funcionalidades
- Para fazer o TestNG executar nossos scripts através de massa de dados fixa teremos que utilizar duas anotações:
  - @DataProvider (name=“dados”)
  - @Test(dataProvider =“dados”)
- Exemplo funcional em **Treinamento Selenium Avançado\Projetos\_Java\Selenium2TestNG**

# Selenium + TestNG

- Existe mais uma funcionalidade interessante no TestNG, alem do @DataProvider que e a criação de grupos
- Um grupo e uma “tag” onde podemos depois executar os scripts por esta tag
- Muito útil quando queremos executar parte dos testes (sanidade) ou somente testes para determinada funcionalidade

# Selenium + TestNG

- O grupo é determinado na anotação do Teste, como no exemplo abaixo

```
@Test(description="Adiciona e remove cliente", dataProvider = "massaDados", groups = "sanidade")
public void adicionaRemoveCliente(String nome, String sobrenome, String pagamento, String observacao) {
    adicionaCliente(nome, sobrenome, pagamento, observacao);
    removeCliente();
}
```

- A execução do TestNG é baseada por arquivos XML e pode ser de diferentes formas. A execução por grupo fica como a imagem abaixo

```
<test name="Execucao">
    <groups>
        <run>
            <include name="sanidade"/>
        </run>
    </groups>
</test>
```

# Selenium + TestNG

- Para utilizar o TestNG devemos baixar sua biblioteca (arquivo .jar) e também o plugin no Eclipse

Download

<http://testng.org/doc/download.html>

Plugin Eclipse

<http://testng.org/doc/eclipse.html>

## 1) Baixando o arquivo jar

Efetue o download do TestNG e efetua a descompactação do arquivo baixado.

Utilize o arquivo **testng-6.2.jar** como lib do Eclipse



## 2) Instalando o plugin no Eclipse

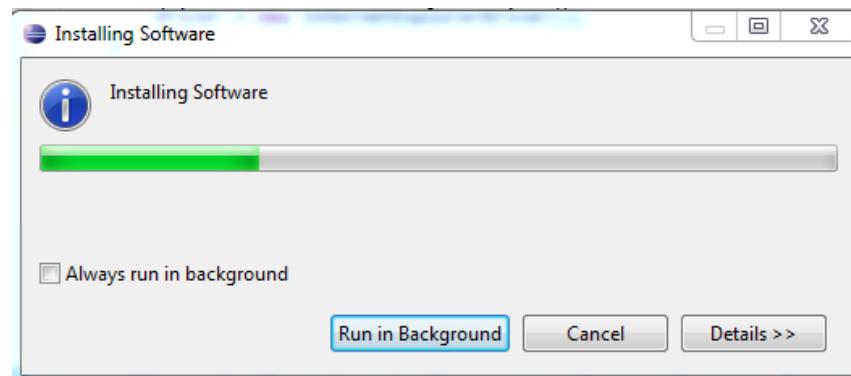
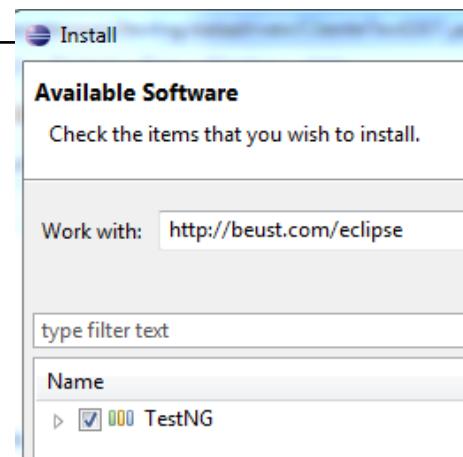
No Eclipse selecione o menu **Help/Instal New Software...**

Adicione a seguinte URL no campo **Work With:**  
<http://beust.com/eclipse>

Selecone o item **TestNG** e clique em **Next**

Na próxima tela clique em **Next** novamente e aceita a licença do **TestNG**

A instalação do plugin ira iniciar



### 3) Adicionando Anotação no método de execução

No método de teste colocaremos a anotação **@Test** juntamente com os parâmetros de descrição, massa de dados e grupo

```
@Test(description="Adiciona e remove cliente", dataProvider = "massaDados", groups = "sanidade")
public void adicionaRemoveCliente(String nome, String sobrenome, String pagamento, String observacao) {
    adicionaCliente(nome, sobrenome, pagamento, observacao);
    removeCliente();
}
```

### 4) Crie o método que preenchera o DDT

Agora é necessário a criação do método padrão com os dados de execução. Note que os dados estão em ordem (nome, sobrenome, pagamento e observação) e devem respeitar sempre esta ordem

```
@DataProvider(name = "massaDados")
public Object[][] createData1() {
    return new Object[][] {
        { "Elias", "Nogueira", "PagSeguro", "Envio por enc. normal" },
        { "Joao", "dos Santos", "PayPal", "Envio Sedex 10" },
    };
}
```

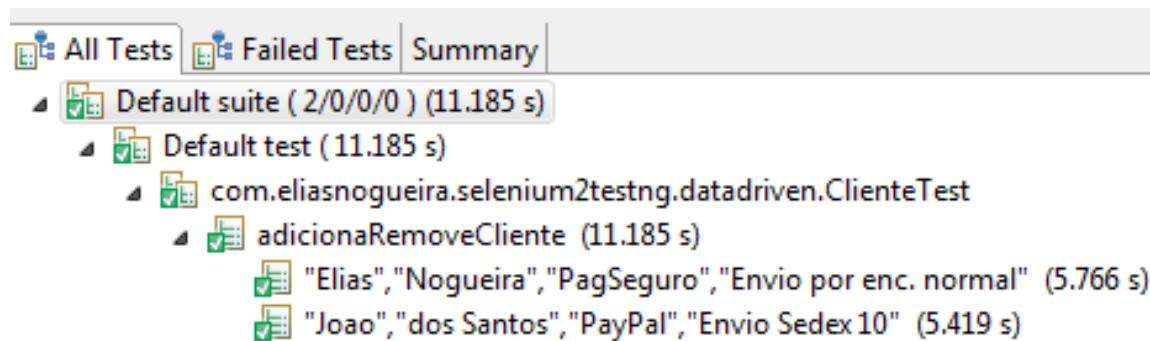
Esse método faz com que os dados sejam passados para os parâmetros e construtor na execução do script

## 5) Substituir os dados fixos por variáveis

No TestNG não precisamos criar previamente as variáveis. O importante é que as variáveis na assinatura do método devem estar na mesma ordem do DDT

```
public void adicionaCliente(String nome, String sobrenome, String pagamento, String observacao) {  
  
    // Espera implícita no Selenium para todos os elementos  
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
  
    driver.get("http://localhost/selenium/exercicios/selenium_comp/scysnlkw/ajaxCRUD/exemplo/");  
    driver.findElement(By.cssSelector("input.btn.editingSize")).click();  
    driver.findElement(By.cssSelector("table[name=form] > tbody > tr > td > input[name=fldField1]")).sendKeys(nome);  
    driver.findElement(By.name("fldField2")).sendKeys(sobrenome);  
    new Select(driver.findElement(By.name("fldCertainField"))).selectByValue(pagamento);  
    driver.findElement(By.name("fldLongField")).sendKeys(observacao);  
    driver.findElement(By.xpath("//input[@value='Salvar Item']")).submit();  
  
    Assert.assertEquals(driver.findElement(By.xpath("//span")).getText(), nome);  
}
```

Da mesma forma que no JUnit o script será executado a quantidade de vezes que existir dados no método de DDT



# Selenium + TestNG

- A forma do DDT com Excel também mudou para este exemplo. Agora podemos utilizar um arquivo Excel com diversas planilhas ou tabelas

Dados	nome	sobrenome	pagamento	observacao
Elias	Nogueira	Cartao de Credito	Envio por Sedex	
Bart	Simpson	PayPal	Envio Internacional	Dados

- Veremos como montar a base do nosso exemplo

# Selenium + TestNG

- Para que isso seja possível existe no projeto **Selenium2TestNG** uma classe chamada **SpreadsheetData** no pacote **com.eliasnogueira.selenium2testng.util**
- Você pode criar sua própria classe. A única restrição para utilização com DDT é ela retornar uma coleção de objetos

Collection<Object [] >

# Selenium + TestNG

- A classe **ClienteTest.DDT** contem apenas a mudança no método que, ao invés dos dados fixos, chama o arquivo Excel em algum diretório.

```
@DataProvider(name = "massaDados")
public Object[][] createData1() throws IOException {
    Object[][] testData = SpreadsheetData.readExcelData("AdicionarCliente", "massaDados/massa_de_dados.xls","Dados");
    return testData;
}
```

# Selenium + TestNG

- Nesta abordagem, diferente do JUnit, podemos ter uma tabela de dados e diversas planilhas.
- Passaremos o nome da planilha, local e nome do arquivo e nome da tabela
- Podemos ter diversas tabelas dentro de uma planilha, o único cuidado é que devemos colocar uma “tag” com o nome da tabela no inicio e no fim dela, como na imagem abaixo

B	C	D	E	F	G
Dados	nome	sobrenome	pagamento	observacao	
Elias	Nogueira	Cartao de Credito	Envio por Sedex		
Bart	Simpson	PayPal	Envio Internacional		
					Dados
AdicionarCliente					

A photograph of six business professionals—three men and three women—standing in a row against a white background. They are all smiling and holding a large, blank white rectangular sign between them. The sign is positioned in front of them, partially obscuring their torsos.

# Selenium Page Objects

# PageObjects

- É um modelo de modelagem e arquitetura dos testes a fim de oferecer as funcionalidades dentro de uma página como serviço
- Cada funcionalidade de uma página vira um serviço. No caso mais clássico de uma tela de cadastro temos as funções:
  - Adicionar
  - Remover
  - Procurar
  - Atualizar

# PageObjects

- Existem algumas regras e recomendações básicas para a utilização do PageObjects
  - Cada método tem como declaração no retorno sempre a própria classe
  - Cada método retorna uma nova instancia do driver
  - Todas as validações devem ser feitas fora do código-fonte
- Para as validações é ideal criar métodos que procurem por certo texto ou ponto de controle.
- Em alguns casos podemos ter validações dentro da chamada da ação, mas devemos procurar colocá-los somente se necessário
- Exemplo funcional em **TreinamentoSelenium Avancado\Projetos\_Java\Selenium2PageObject**

# Exemplo de Teste com PageObjects

```
public class CadastroCliente {  
    private final WebDriver driver;  
  
    public CadastroCliente(WebDriver driver) {  
        this.driver = driver;  
    }  
  
    public CadastroCliente adicionar(String nome) {  
        //codigo para adicionar  
        return new CadastroCliente(driver);  
    }  
    public CadastroCliente remover(String nome) {  
        // codigo para remover  
        return new CadastroCliente(driver);  
    }  
    public CadastroCliente pesquisar(String nome) {  
        // codigo para pesquisar  
        return new CadastroCliente(driver);  
    }  
    public CadastroCliente atualizar(String nomeAtual, String novoNome) {  
        // codigo para atualizar  
        return new CadastroCliente(driver);  
    }  
  
    public boolean isClienteCadastradoComSucesso(String nome) {  
        return driver.findElement(By.tagName("body")).getText().contains(nome);  
    }  
  
    public boolean isClienteRemovidoComSucesso(String nome) {  
        return !driver.findElement(By.tagName("body")).getText().contains(nome);  
    }  
  
    public boolean isClienteAtualizadoComSucesso(String novoNome) {  
        return !driver.findElement(By.tagName("body")).getText().contains(novoNome);  
    }  
}
```

# Execução do Teste com PageObjects

```
public class ExecutaCliente {  
  
    protected WebDriver driver;  
  
    @Test  
    public void testClienteCompleto() {  
        CadastroCliente cliente = new CadastroCliente(driver);  
  
        cliente.adicionar("Fulano");  
        Assert.assertEquals(cliente.isClienteAdicionadoComSucesso("Fulano"), true);  
  
        cliente.atualizar("Fulano", "Deltrano");  
        Assert.assertEquals(cliente.isClienteAdicionadoComSucesso("Deltrano"), true);  
        Assert.assertEquals(!cliente.isClienteAdicionadoComSucesso("Fulano"), true);  
  
        cliente.pesquisar("Dentrano");  
        Assert.assertEquals(cliente.isClienteAdicionadoComSucesso("Deltrano"), true);  
  
        cliente.remover("Dentrano");  
        Assert.assertEquals(cliente.isClienteAdicionadoComSucesso("Deltrano"), true);  
    }  
}
```

A photograph of six business professionals—three men and three women—standing in a row against a white background. They are all smiling and holding a large, blank white rectangular sign between them. The sign has a slight shadow, suggesting it is a physical object.

# **Execução em diversos browsers**

# Execução em diversos browsers

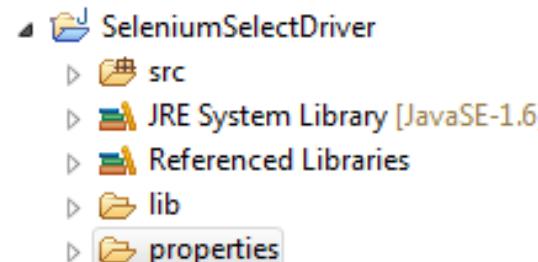
- Sabemos que um dos pontos fortes do Selenium é executar seus testes em diversos browsers web
- Podemos fazer isso pelo Selenium RC ou via linguagem de programação
- Veremos aqui como criar uma classe em Java para tratar da execução de cada browser
- Exemplo funcional em **Treinamento Selenium Avancado\Projetos\_Java\SeleniumSelectDriver**

# Execução em diversos browsers

- No Selenium 2 cada browser tem um classe ligado a ele. Isso existe para facilitar a interação entre os comandos do Selenium e os browsers
- No Selenium 2 os seguintes drivers estão disponíveis:
  - Chrome
  - Firefox
  - Internet Explorer
  - Opera
  - ~~Safari~~
  - iPhone
  - Android
  - HTMLUnitDriver

## 1) Itens necessários para o projeto

E necessário criar uma pasta de qualquer nome, no exemplo utilizaremos **properties**

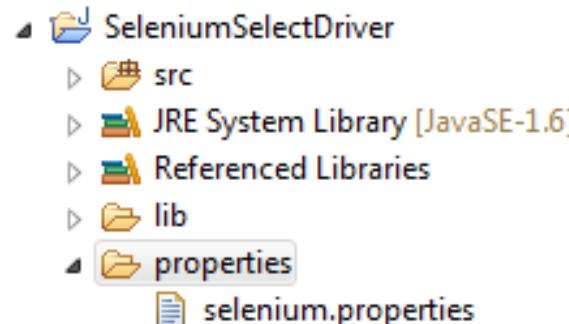


## 2) Criar um arquivo na pasta criada

E necessário também criarmos um arquivo com qualquer nome e colocá-lo na pasta criada (neste exemplo a pasta **properties**)

O nome usado agora como exemplo será **selenium.properties**

Após isso o arquivo precisa ter a propriedade necessária, neste exemplo colocaremos **selenium.browser = firefox**



A screenshot of a code editor window showing the 'selenium.properties' file. The file contains the following line:

```
selenium.browser = ie
```

## 1) Classe de controle

E necessário uma classe que controle qual browser iremos utilizar. No exemplo existe a classe **SeleniumUtils.java** que contem o método **getDriver()** que é o responsável por

```
public static WebDriver getDriver(String browser) {
    if (driver == null) {
        if (browser.equals(FIREFOX)) driver = new FirefoxDriver();

        if (browser.equals(GOOGLECHROME)) {
            DesiredCapabilities capabilities = DesiredCapabilities.chrome();

            capabilities.setJavascriptEnabled(true);
            capabilities.setCapability("chrome.binary", "C:\\\\Users\\\\Elias\\\\AppData\\\\Local\\\\Google\\\\Chrome\\\\Application\\\\chrome.exe");
            System.setProperty("webdriver.chrome.driver", "C:\\\\chromedriver.exe");

            driver = new ChromeDriver(capabilities);
        }

        if (browser.equals(IE)) {
            DesiredCapabilities capabilities = DesiredCapabilities.internetExplorer();

            capabilities.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNORING_SECURITY_DOMAINS, true);
            driver = new InternetExplorerDriver(capabilities);
        }

        if (browser.equals(OPERA)) driver = new OperaDriver();
    }
    return driver;
}
```

## Explicação

Para os drives do **Firefox** e **Opera** não é necessário efetuar qualquer configuração, basta instanciar o dever com a sua respectiva classe.

Para o **GoogleChrome** é necessário efetuar uma configuração. Para isso utilizamos a classe **DesiredCapabilities** para informar estas configurações.

Basicamente habilitamos a permissão de executar javascript (no código de exemplo) no browser, porém devemos informar o caminho de dois aplicativos:

1. executável do GoogleChrome (onde ele foi instalado)
2. executável que habilita a execução do Chrome

Para o item 1 habilitamos a capacidade de saber onde o browser está instalado

```
capabilities.setCapability("chrome.binary", "C:\\\\Users\\\\Elias\\\\AppData\\\\Local\\\\Google\\\\Chrome\\\\Application\\\\chrome.exe");
```

Para o item 2 temos que efetuar o download de um aplicativo (cada sistema operacional tem o seu) e informar onde este aplicativo está

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

O aplicativo deve ser baixado pelo seguinte link: <http://code.google.com/p/chromium/downloads/list>

Para o **Internet Explorer** adicionamos a capacidade de ignorar warnings por causa dos alertas de segurança no acesso as páginas. Se isso não for feito programaticamente teremos que configurar o IE para que ele não apresente mensagens de segurança

```
capabilities.setCapability(InternetExplorerDriver.INTRODUCE_FLAKINESS_BY_IGNORING_SECURITY_DOMAINS, true);
```

## 2) Classe de carregamento da propriedade do arquivo

Dentro da mesma classe `SeleniumUtils.java` que contem o método `getSeleniumProperties()` que e o responsável por carregar o arquivo de propriedade `selenium.properties` na pasta `properties`.

Você pode alterar depois para a pasta que quiser e mudar o nome do arquivo, porem deve mudar no código apontado para o novo local e nome do arquivo.

Para este método basta passar o nome da propriedade que esta no arquivo e ele retorna o seu valor

```
public static String getSeleniumProperties(String name) {
    Properties properties = new Properties();
    String value = null;

    try {
        properties.load(new FileInputStream("properties/selenium.properties"));
        value = properties.getProperty(name);

    } catch (IOException e) {
        e.printStackTrace();
    }

    return value;
}
```

### 3) Utilização

A utilização é muito simples. Ao invés de instanciarmos o browser que queremos executar os testes iremos chamar os dois métodos da classe **SeleniumUtils.java**. Chamamos o primeiro método para pegar o driver necessário e o outro para carregar o driver pelo nome do arquivo.

Assim toda vez que quisermos trocar de browser para a execução dos testes não precisaremos alterar o script de teste, somente o arquivo **selenium.properties**

```
public class TestBrowserExecution implements SeleniumInterface {  
  
    protected WebDriver driver;  
  
    @Before  
    public void setup() {  
        driver = SeleniumUtils.getDriver(SeleniumUtils.getSeleniumProperties("selenium.browser"));  
    }  
  
    @Test  
    public void testaAcesso() {  
        driver.get("http://eliasnogueira.com/selenium/exercicios/selenium_ide/ajybuyje/avancado/");  
        driver.findElement(By.tagName("body")).getText().contains("Elementos HTML");  
    }  
  
    @After  
    public void tearDown() {  
        driver.close();  
    }  
}
```

A photograph of six business professionals—three men and three women—standing behind a large, blank white rectangular sign. They are all smiling and looking towards the camera. The background is plain white.

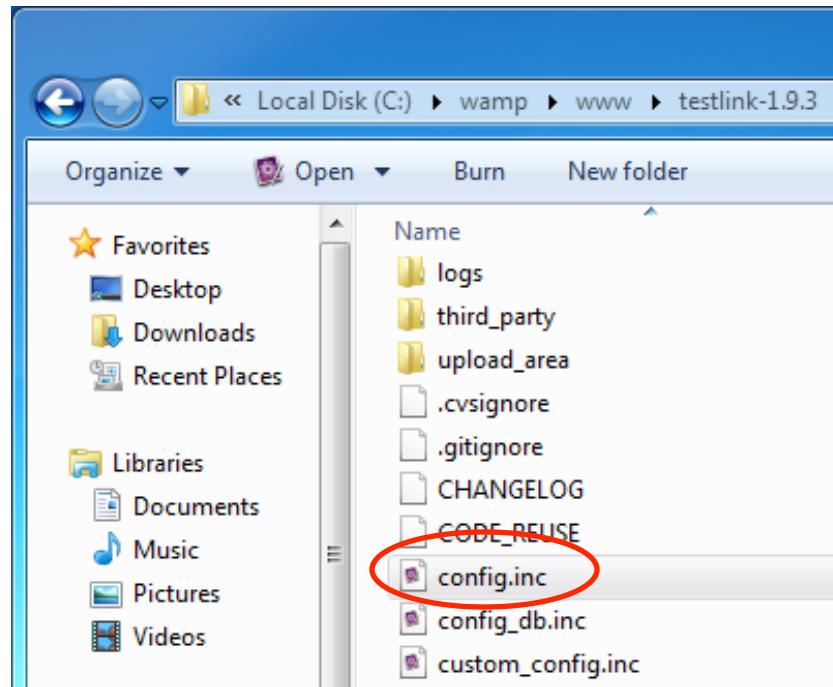
# **Integração Selenium + Testlink**

# Integração Selenium e Testlink

- A integração se dá por meio de uma API via XML-RPC
- Deve ser configurável e habilitado no Testlink
- Cada funcionalidade pode ser tratada como demanda (classes ou arquivos-fonte)
- Será utilizada uma API criada para a comunicação do Testlink com Java
- Exemplo funcional em **Treinamento Selenium Avançado**  
**\Projetos\_Java\SeleniumTestlink**

# Configurações no Testlink

**1) Adicionando as propriedades necessárias.**  
Va até a pasta de instalação do Testlink e abra o arquivo *config\_inc.php*



**2) Copiando as propriedades**

Procure pelo texto “[API]”

Depois copie a propriedade  
\$tlCfg->api->enabled =  
FALSE;

```
371 // -----
372 /* [API] */
373
374 /**
375  * XML-RPC API availability (disabled by default)
376 $tlCfg->api->enabled = FALSE;
```

### 3) Edite o arquivo de configuração de usuário

Na pasta de instalação do Testlink procure pelo arquivo custom\_config\_inc.php.example e renomeie para custom\_config\_inc.php removendo o “.example”

CHANGELOG	211,574
CODE_REUSE	558
config.inc.php	63,841
custom_config.inc.php.example	8,478
firstLogin.php	3,775
index.php	1,259
LICENSE	18,009
config.inc	7/2/2011 10:33 AM PHP File
config_db.inc	8/1/2011 11:13 PM PHP File
custom_config.inc	8/5/2011 12:14 PM PHP File
firstLogin	7/2/2011 10:33 AM PHP File
index	7/2/2011 10:33 AM PHP File
LICENSE	7/2/2011 10:33 AM File

### 4) Cole a propriedade

Edite o arquivo custom\_config.php e cole no final do arquivo a propriedade copiada anteriormente e coloque o valor para TRUE

```
207  /* [API] */
208
209  /** XML-RPC API availability (disabled by default) */
210 $tlCfg->api->enabled = TRUE;
211
```

## 5) Habilitar chamada da automação

Volte ao arquivo config.inc.php procure pela propriedade “\$tlCfg->exec\_cfg->enable\_test\_automation”

Após encontrá-la, copie toda a linha da propriedade

```
621 /* [Test Executions] */  
622  
623 // ENABLED -> enable XML-RPC calls to external test automation server  
624 // new buttons will be displayed on execution pages  
625 // DISABLED -> disable  
626 $tlCfg->exec_cfg->enable_test_automation = DISABLED;  
...
```

## 6) Cole a propriedade

Edite o arquivo custom\_config.php e cole no final do arquivo a propriedade copiada anteriormente e coloque o valor para ENABLED

```
215 /* [Test Executions] */  
216  
217 // ENABLED -> enable XML-RPC calls to external test automation server  
218 // new buttons will be displayed on execution pages  
219 // DISABLED -> disable  
220 $tlCfg->exec_cfg->enable_test_automation = ENABLED;  
...
```

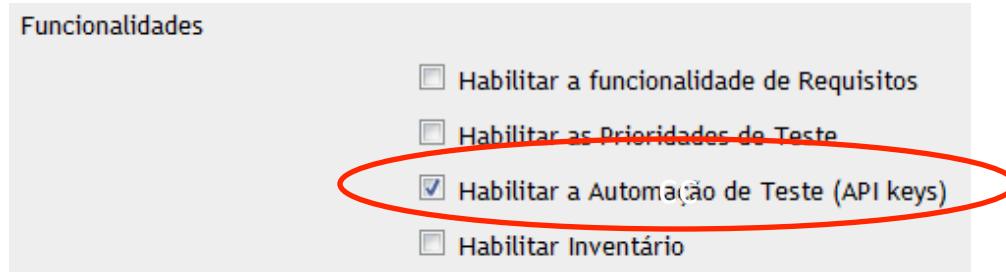
## 7) Habilitando a Automação de Teste no Testlink

Acesse o Testlink, edite ou crie um novo projeto.

Na tela de detalhes do projeto marque o item “**Habilitar a Automação de Teste (API keys)**”

Início -> Projeto de Teste -> Gerenciar Projeto de Teste

- Clicar no botão *Criar* ou
- Clicar sobre o nome do projeto para editá-lo



## 8) Gerar chave API

No Teslink, vá ao item “**Pessoal**” e encontre o item “**API interface**”.

Clique sobre o botão “**Gerar uma nova chave**” para a geração de uma chave.

Copie e guarde esta chave para utilização futura

### API interface

Chave de acesso pessoal API = Nenhum

**Gerar uma nova chave**

### API interface

Chave de acesso pessoal API = 6124518907fec74a176763d573fcf401

**Gerar uma nova chave**

## 9) Criar estrutura no Testlink

Não será necessário executar este passo no treinamento, basta acessar o Testlink do curso e visualizar o projeto “Curso Selenium Avançado”

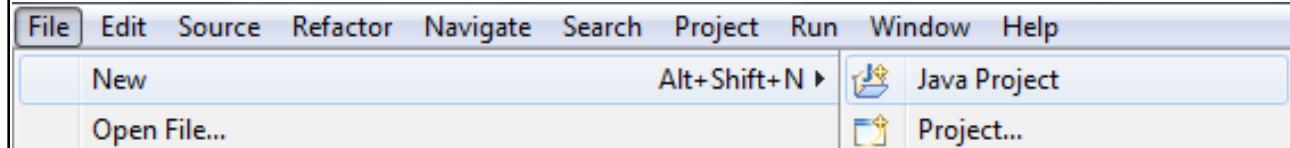
Necessário criar toda a estrutura no Testlink

- Projeto de Teste (se não existir)
- Plano de Teste
- Suíte de Teste
- Caso de Teste
- Build (Baseline)

## 10) Criar projeto no Eclipse

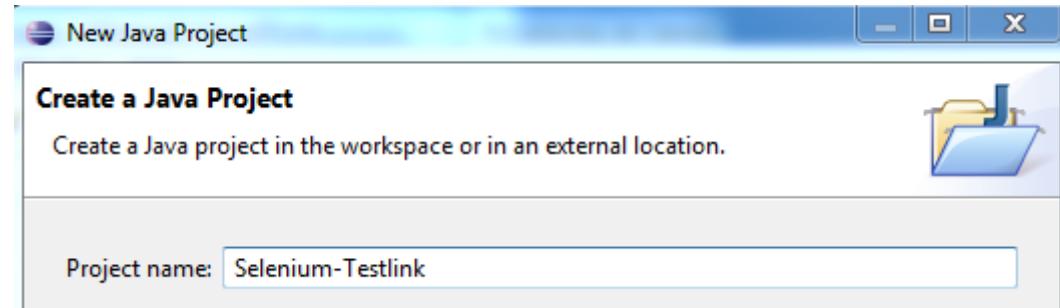
Agora será necessário escrever código dentro do Eclipse em Java para fazer a integração, que veremos nos próximos passos.

Agora, no Eclipse, selecione o menu **File/New/Java Project** para criar um novo projeto



## 11) Nome do Projeto

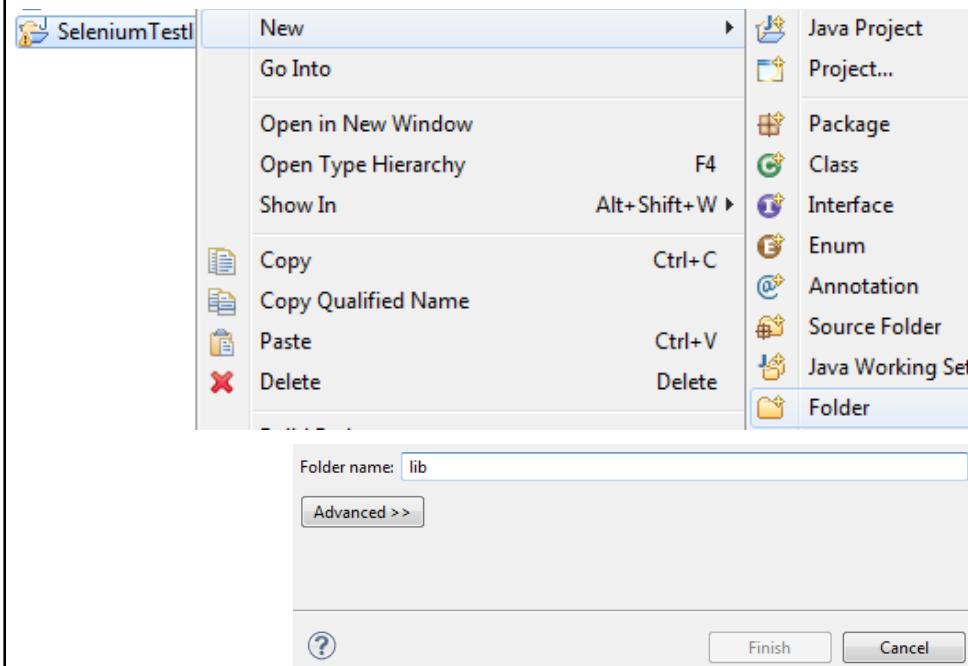
No campo *Project name* coloque o nome do projeto como “*SeleniumTestlink*” e clique no botão **Finish**



## 12) Criar pasta lib e adicionar as bibliotecas

Agora clique com o botão direito sobre o nome do projeto e selecione o menu **New/Folder**

Na tela **New Folder** coloque no campo *Folder name* o nome *lib*



### 13) Copiar as bibliotecas para o projeto

Em nosso projeto de exemplo as bibliotecas já estão adicionadas, porém os links de cada biblioteca serão listadas ao lado.

Os arquivos .jar devem ser copiados para **a pasta do usuário/workspace**

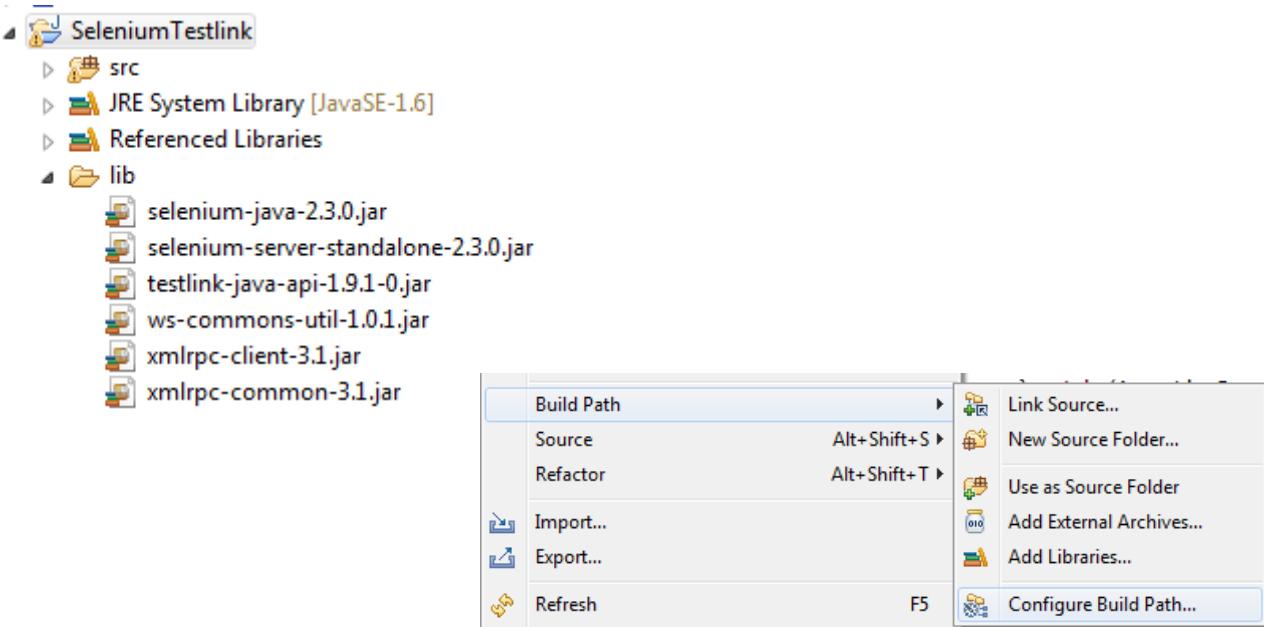
- [Selenium Server](#)
- [Selenium Java Client Driver](#)
- [Testlink Java API](#)
- [Apache WS Common Utils](#)
- [XML RPC Client](#)
- [XML RPC Common](#)

**OBS:** Os links para XML RPC Client e XML RPC Common são os mesmos, basta baixar em um dos dois links

### 14) Atualizando e adicionando as bibliotecas

Após o download e a copia para a pasta **lib** clique bom o botão direito sobre o projeto e selecione **Refresh**.

Após isso clique bom o botão direito novamente e selecione **Build Path/Configure Build Path**

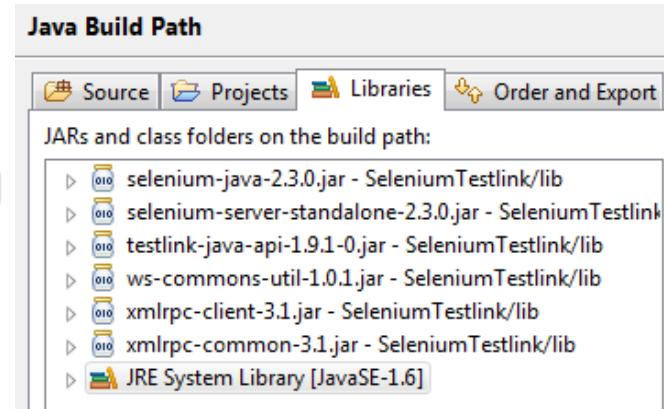
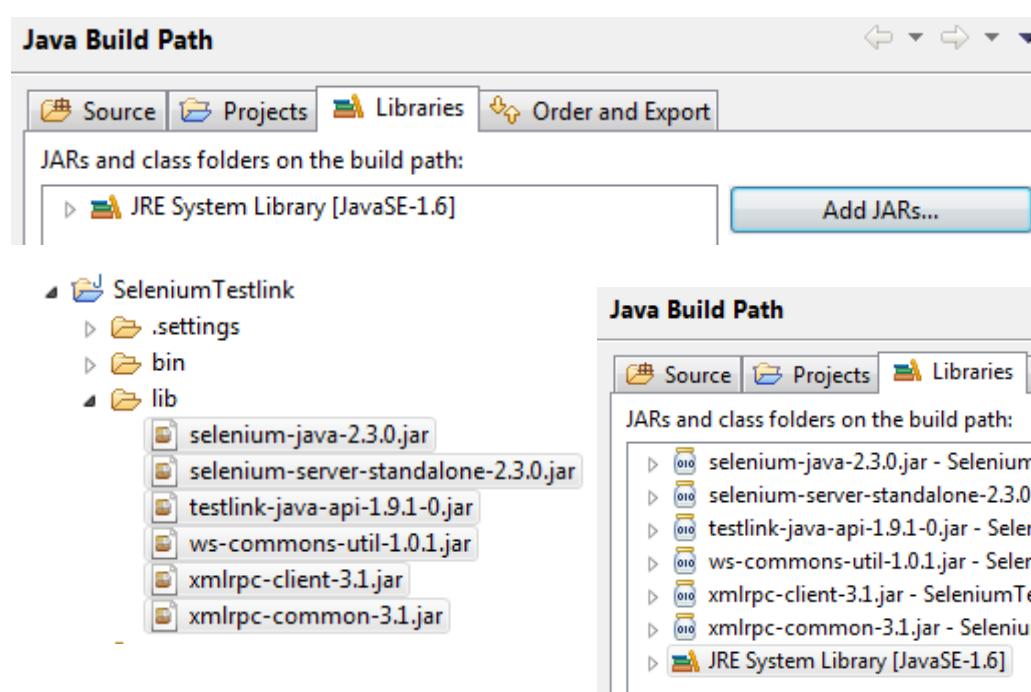


## 15) Adicionando as bibliotecas

Clique sobre a aba *Libraries* e depois sobre o botão *Add Jars...*

Na tela apresentada selecione o projeto e va até a pasta *lib*. Selecione todos os arquivos listados e clique em **OK**

Após isso todos os arquivos estarão listados na aba *Libraries*. Clique no botão **OK**

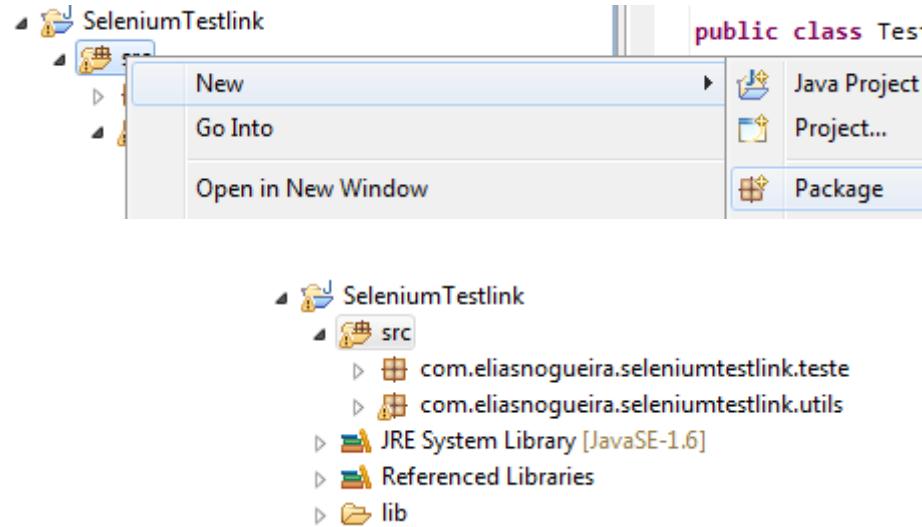


## 16) Criação do pacote utils

Vá até a pasta *src*, clique bom o botão direito e selecione *Package*.

Em *Name* coloque qualquer nome de sua escolha. No exemplo nome utilizado é *com.eliasnogueira.seleniumtestlink.utils*

Clique no botão *Finish*



## 17) Código para conexão com o Testlink

Trecho de código necessário para fazer a conexão com o Testlink.

Em nosso exemplo a classe criada tem o nome **TestlinkUtil** e no pacote **com.eliasnogueira.seleniumtestlink.util**

```
import java.net.MalformedURLException;
import java.net.URL;

import br.eti.kinoshita.testlinkjavaapi.TestLinkAPI;
import br.eti.kinoshita.testlinkjavaapi.TestLinkAPIException;

public class TestlinkUtil {

    public static TestLinkAPI getTestlinkAPI() throws MalformedURLException, TestLinkAPIException {
        String url = "http://localhost/testlink-1.9.3/lib/api/xmlrpc.php";
        String devKey = "6124518907fec74a176763d573fcf401";
        TestLinkAPI api = null;

        try {
            new URL(url);
            api = new TestLinkAPI(url, devKey);

        } catch ( Exception e ) {
            e.printStackTrace();
            System.exit(-1);
        }

        System.out.println(api.ping());
        return api;
    }
}
```

## 18) Código para reportar o resultado do Testlink

Trecho de código necessário reportar o resultado do teste no Testlink.

Em nosso exemplo a classe criada tem o nome **TestlinkMetodos** e no pacote **com.eliasnogueira.seleniumtestlink.util**

```
import java.net.MalformedURLException;
import java.util.Map;

import br.eti.kinoshita.testlinkjavaapi.TestLinkAPI;
import br.eti.kinoshita.testlinkjavaapi.TestLinkAPIException;
import br.eti.kinoshita.testlinkjavaapi.model.Build;
import br.eti.kinoshita.testlinkjavaapi.model.ExecutionStatus;
import br.eti.kinoshita.testlinkjavaapi.model.ReportTCResultResponse;
import br.eti.kinoshita.testlinkjavaapi.model.TestPlan;

public class TestlinkMetodos {

    public static ReportTCResultResponse executarCasoDeTeste(String planoTeste, String projeto,
                                                               String suite, String casoDeTeste, ExecutionStatus status, String nota)
            throws MalformedURLException, TestLinkAPIException {

        TestLinkAPI testlink = TestlinkUtil.getTestlinkAPI();
        TestPlan testlinkPlan = testlink.getTestPlanByName(planoTeste, projeto);
        Build build = testlink.getLatestBuildForTestPlan(testlinkPlan.getId());

        Integer testCaseId = testlink.getTestCaseIDByName(casoDeTeste, suite, projeto, null);
        Integer testCaseExternalId = 0;
        Integer testPlanId = testlinkPlan.getId();
        Integer buildId = build.getId();
        String buildName = build.getName();
        String notes = nota;
        Boolean guess = false;
        String bugId = null;
        Integer platformId = 0;
        String platformName = null;
        Map<String, String> customFields = null;
        Boolean overwrite = false;
        return testlink.setTestCaseExecutionResult(testCaseId, testCaseExternalId,
                                                   testPlanId, status, buildId, buildName, notes, guess, bugId,
                                                   platformId, platformName, customFields, overwrite);
    }
}
```

## 19) Utilizando o envio do resultado de teste dentro do teste no Selenium

```
try {
    // código do Selenium omitido
    // no final do teste colocar o seguinte trecho
    TestlinkMetodos.executarCasoDeTeste("Plano AjaxCRUD",
        "Projeto Teste", "Suite CRUD", "Adiconar um novo Cliente",
        ExecutionStatus.PASSED, "Passou com sucesso" );

} catch (Exception e) {
    TestlinkMetodos.executarCasoDeTeste("Plano AjaxCRUD",
        "Projeto Teste", "Suite CRUD", "Adiconar um novo Cliente",
        ExecutionStatus.FAILED, e.toString());
    TestCase.fail();
}
```

O teste no Selenium deve ter no início do código um `try` (controle de exceção) para que seja possível detectar se algum erro ocorre.

No código acima o envio de resultado é com sucesso, pois não foi detectado nenhum erro onde o status da execução (`ExecutionStatus`) passa OK (`PASSED`) para o Testlink

No bloco `catch` é onde o código irá detectar se um erro irá ocorrer., logo colocamos a mesma chamada de envio de resultado para o Testlink, porém notificando um erro pelo status da execução (`ExecutionStatus`) como erro (`FAILED`) e passando a mensagem de erro através do método `e.toString()`;

Ainda dentro do bloco `catch` é necessário colocar o código `TestCase.fail();` para informar que o teste não executou com sucesso.

A photograph of six people (three men and three women) standing behind a large white rectangular sign. They are all smiling and looking towards the camera. The sign is blank, except for the text "Integração Selenium + Mantis" centered on it.

# **Integração Selenium + Mantis**

# Integração Selenium e Mantis

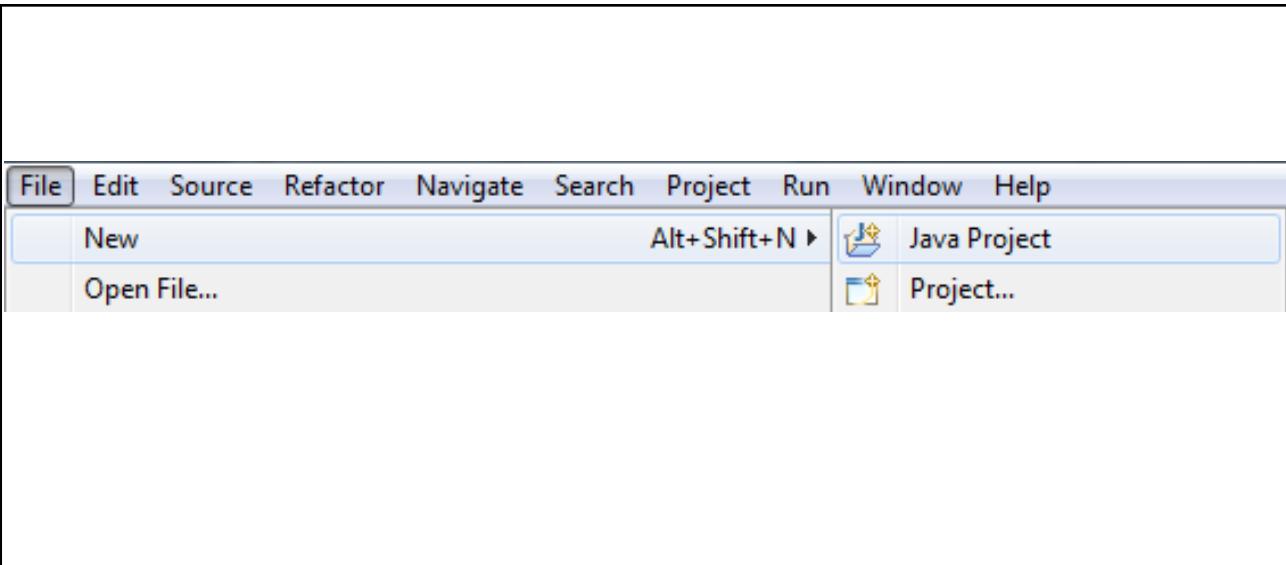
- A integração se dá por meio de uma API via XML-RPC
- Não é necessário configuração no Mantis
- Cada funcionalidade pode ser tratada como demanda (classes ou arquivos-fonte)
- Será utilizada uma API criada para a comunicação do Mantis com Java
- Exemplo funcional em **Treinamento Selenium Avançado**  
**\Projetos\_Java\SeleniumMatis**

# Criando projeto de Integração com Mantis

## 1) Criar projeto no Eclipse

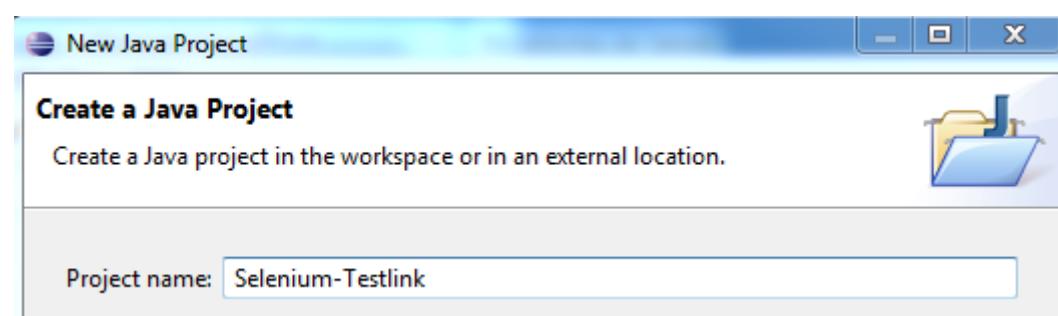
Agora será necessário escrever código dentro do Eclipse em Java para fazer a integração, que veremos nos próximos passos.

Agora, no Eclipse, selecione o menu **File/New/Java Project** para criar um novo projeto



## 2) Nome do Projeto

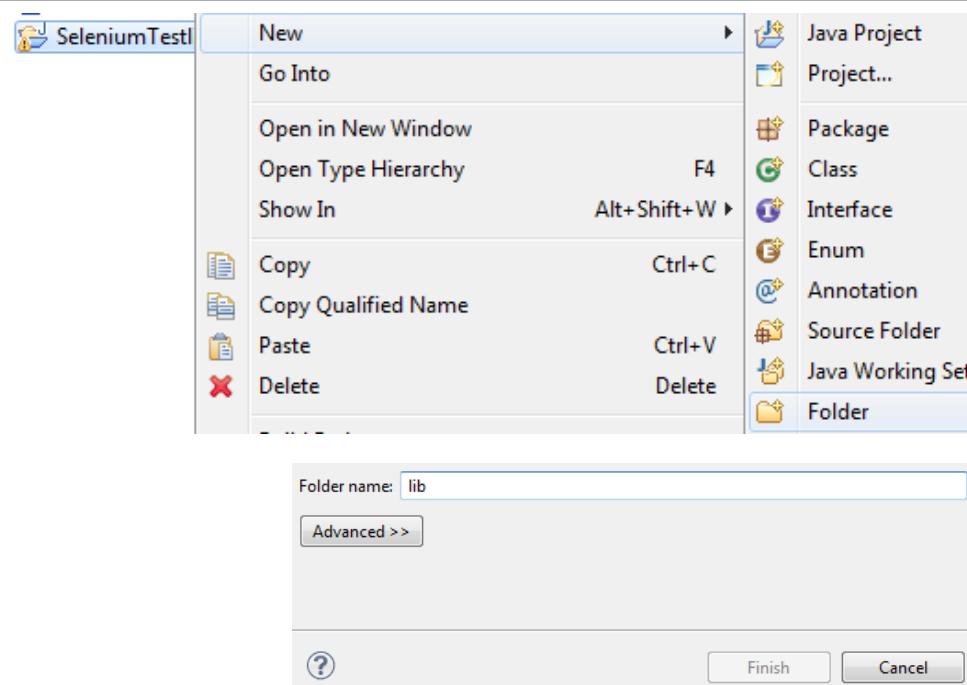
No campo **Project name** coloque o nome do projeto como "**SeleniumMantis**" e clique no botão **Finish**



### 3) Criar pasta lib e adicionar as bibliotecas

Agora clique com o botão direito sobre o nome do projeto e selecione o menu **New/Folder**

Na tela **New Folder** coloque no campo **Folder name** o nome **lib**



### 4) Copiar as bibliotecas para o projeto

Em nosso projeto de exemplo as bibliotecas já estão adicionadas, porém os links de cada biblioteca serão listadas ao lado.

Os arquivos .jar devem ser copiados para **a pasta do usuario/workspace**

Os arquivos necessários estão dentro do arquivo zip do *MantisConnect-Client-API*. Adicione os seguintes arquivos:

- axis-REV609819.jar
- commons-discovery-0.2.jar
- jaxrpc.jar
- wsdl4j-1.5.1.jar

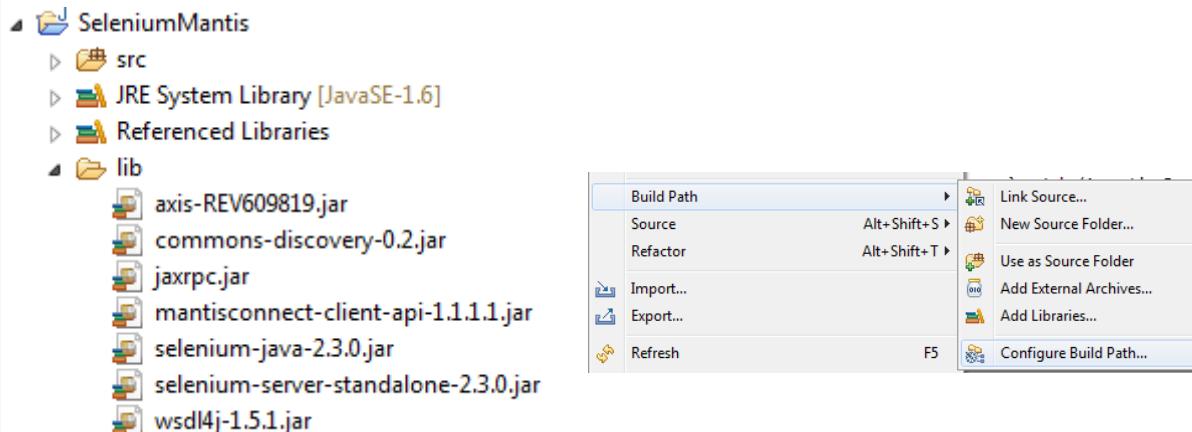
Copiar também o arquivo *mantisconnect-client-api-1.1.1.jar* que está no arquivo .zip

**OBS:** Os links para XML RPC Client e XML RPC Commom são os mesmos, basta baixar em um dos dois links

## 5) Atualizando e adicionando as bibliotecas

Após o download e a copia para a pasta **lib** clique bom o botão direito sobre o projeto e selecione **Refresh**.

Após isso clique bom o botão direito novamente e selecione **Build Path/Configure Build Path**

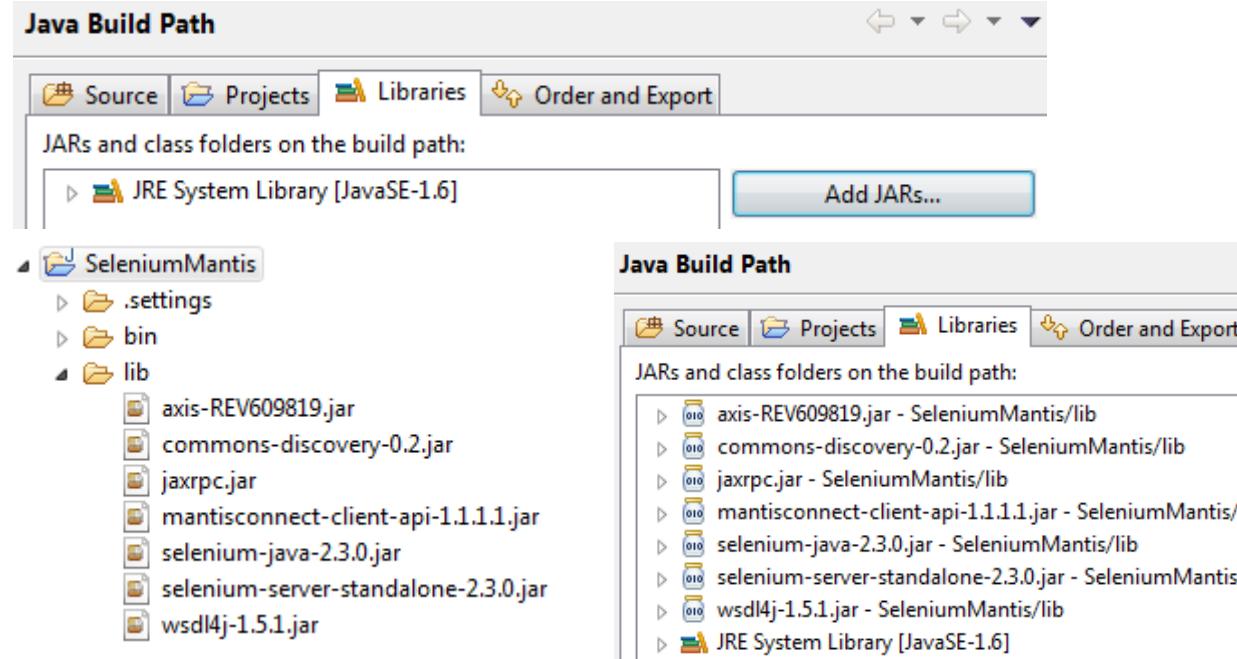


## 6) Adicionando as bibliotecas

Clique sobre a aba **Libraries** e depois sobre o botão **Add Jars...**

Na tela apresentada selecione o projeto e va até a pasta **lib**. Selecione todos os arquivos listados e clique em OK

Após isso todos os arquivos estarão listados na aba **Libraries**. Clique no botão OK

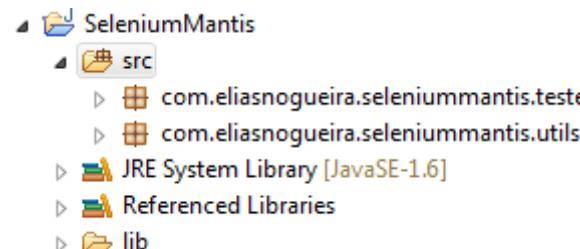
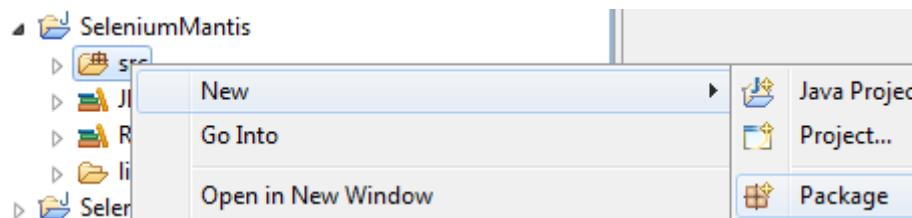


## 7) Criação do pacote utils

Vá até a pasta **src**, clique bom o botão direito e selecione **Package**.

Em **Name** coloque qualquer nome de sua escolha. No exemplo nome utilizado é **com.eliasnogueira.seleniummantis.utils**

Clique no botão **Finish**



## 8) Código para conexão com o Mantis

Trecho de código necessário para fazer a conexão com o Testlink.

Em nosso exemplo a classe criada tem o nome **MantisUtil** e no pacote **com.eliasnogueira.seleniummantis.util**

```
public class MantisUtil {

    private static MantisUtil instance = null;
    private static IMCSession sessao = null;
    private String urlMantis = "http://localhost/mantisbt-1.2.6/api/soap/mantisconnect.php";
    private String usuario = "Administrator";
    private String senha = "root";

    public MantisUtil() throws MalformedURLException, MCException {
        URL url = new URL(urlMantis);
        sessao = new MCSession(url, usuario, senha);
    }

    public static MantisUtil getInstance() {
        if (instance == null) {
            try {
                instance = new MantisUtil();
            } catch (MalformedURLException ex) {
                Logger.getLogger(MantisUtil.class.getName()).log(Level.SEVERE, null, ex);
            } catch (MCException ex) {
                Logger.getLogger(MantisUtil.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        return instance;
    }

    public static IMCSession getSessao() throws MalformedURLException, MCException {
        if (sessao == null) {
            getInstance();
        }
        return sessao;
    }
}
```

## 9) Código para reportar um bug do Mantis

Trecho de código necessário para reportar um bug no Mantis.

Em nosso exemplo a classe criada tem o nome

**MantisMetodo** e no pacote  
**com.eliasnogueira.seleniummantis.util**

```
public class MantisMetodo {  
  
    public static String reporIssue(String projeto, String sumario, String descricao,  
                                    String categoria, String informacaoAdicional, String evidencia, String nomeArquivo) {  
        IMCSession sessao = null;  
        String arquivo = nomeArquivo + ".png";  
        String bugID = null;  
  
        try {  
            sessao = MantisUtil.getSessao();  
            IProject projetoMantis = sessao.getProject(projeto);  
            Issue issue = new Issue();  
            issue.setProject(new MCAttribute(projetoMantis.getId(), projetoMantis.getName()));  
            issue.setAdditionalInformation(null);  
            issue.setOs(System.getProperty("os.name"));  
            issue.setOsBuild(System.getProperty("os.version"));  
            issue.setPlatform(System.getProperty("os.arch"));  
            issue.setSeverity(new MCAttribute(70, "crash"));  
            issue.setReproducibility(new MCAttribute(10, "always"));  
            issue.setSummary(sumario + new Date());  
            issue.setDescription(descricao);  
            issue.setCategory(categoria);  
            issue.setPriority(new MCAttribute(40, "high"));  
            issue.setAdditionalInformation(informacaoAdicional);  
            long id = sessao.addIssue(issue);  
            sessao.addIssueAttachment(id, arquivo, "image/png", Base64.decodeBase64(evidencia));  
            bugID = String.valueOf(id);  
  
        } catch (MalformedURLException e) {  
            System.err.println("Erro na URL de acesso ao Mantis");  
            e.printStackTrace();  
        } catch (MCEException e) {  
            System.err.println("Erro na comunicacao com o Mantis");  
            e.printStackTrace();  
        }  
  
        return bugID;  
    }  
}
```

## 10) Utilizando report de bug com o Mantis na execução do script do Selenium

```
try {
    // código do Selenium omitido
    // no final do teste colocar o seguinte trecho
} catch (Exception e) {
    String projeto = "Projeto Teste";
    String sumario = "Teste Report Automatico";
    String descricao = "Este erro foi gerado automaticamente";
    String categoria = "General";
    String informacaoAdicional = e.getMessage();
    String evidencia = selenium.captureScreenshotToString();
    String nomeArquivo = "AdicionarClienteErro";

    MantisMetodo.reporIssue(projeto, sumario, descricao,
                           categoria, informacaoAdicional, evidencia, nomeArquivo);
    TestCase.fail();
}
```

O teste no Selenium deve ter no início do código um `try` (controle de exceção) para que seja possível detectar se algum erro ocorre.

Quando um erro ocorre e cai no bloco `catch` são criados algumas variáveis temporárias apenas para ficar mais fácil a alteração dos dados.

Depois é utilizado a função de report de bugs para enviar o erro ao Mantis.

Note que a variável `informacaoAdicional` pega qual o erro gerado e a variável `evidencia` tira uma screenshot da tela para anexar no bug.

Desta maneira toda a vez que um erro ocorrer um bug com uma imagem da tela no momento do erro é cadastrado automaticamente no Mantis

# Integração Selenium, Mantis e Testlink

- Visualize o exemplo ambas ferramentas sendo utilizadas no mesmo teste
- Exemplo funcional em **Treinamento Selenium Avancado\Projetos\_Java\IntegracaoSelenium**

A photograph of six people (three men and three women) standing behind a large white rectangular sign. They are all smiling and looking towards the camera. The sign is blank, with no writing on it.

**Selenium IDE + Jenkins**

# Jenkins Seleniumhq Plugin

- Permite executar o scripts do Selenium IDE diretamente no Jenkins
- É ideal para testes de sanidade quando cada build é gerada
- Necessita inserir o caminho do servidor (Selenium RC) e Suite + Resultados

# Instalando Seleniumhq plugin

## 1) Acessando os plugins do Jenkins

Primeiro iremos ao painel de plugins do Jenkins. Dentro do Jenkins acesse: **Jenkins/Manage Jenkins/Manage Plugins/Avaliable**

The screenshot shows the Jenkins Plugin Manager interface at the URL <http://localhost:8080/pluginManager/available>. The title bar says "Jenkins". The main area has a blue header with the Jenkins logo. Below it, there's a breadcrumb navigation: "Jenkins > Plugin Manager". There are two main tabs: "Updates" and "Available", with "Available" being the active tab. Below the tabs are two buttons: "Install" and "Manage Jenkins". Under the "Available" tab, there's a section titled "Artifact Uploaders". It lists two plugins:

	ArtifactDeployer Plugin	Artifactory Plugin
<a href="#">View</a>	This plugin makes it possible to cop	This plugin allows deploying Maven

## 2) Selecionando o plugin

Na lista de plugins disponíveis procure pelos dois plugins abaixo e, em seguida, clique no botão **Install** no final da pagina:

- Seleniumhq Plugin
- seleniumhtmlreport Plugin
- SeleniumRC plugin

The screenshot shows the Jenkins Plugin Manager interface again, but this time the "Available" tab is not active. Instead, the "Updates" tab is active. In the "Updates" tab, there are two checkboxes selected for the "Seleniumhq Plugin" and "seleniumhtmlreport Plugin". Both descriptions mention that these plugins allow running and loading HTML Selenese suite results.

	Seleniumhq Plugin	seleniumhtmlreport Plugin
<input checked="" type="checkbox"/>	This plugin allows you to run and load HTML Selenese suite result generate by \$ trend report of test result. The Seleniumhq plug in can be <a href="#">downloaded here</a> .	
<input checked="" type="checkbox"/>		This plugin visualizes the results of selenium tests.

### 3) Instalação dos plugins e reinicio do Jenkis

A tela de instalação de plugins será apresentado.

Selecione o item **Restart Jenkins when installation is complete and no jobs are running** para que o Jenkins reinicie após a instalação

### Installing Plugins/Upgrades

#### Preparation

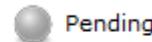
- Checking internet connectivity

seleniumhtmlreport Plugin



Pending

Seleniumhq Plugin



Pending

→  **Restart Jenkins when installation is complete and no jobs are running**

### 4) Visualizando o plugin instalado

Apos o reinicio do Jenkins clique no item **Manage Plugin** e depois na aba **Installed**

Os dois plugins devem aparecer

Updates	Available	Installed	Advanced
Enabled		Name ↓	
		<input checked="" type="checkbox"/>	<a href="#">CVS Plugin</a>
		<input checked="" type="checkbox"/>	<a href="#">Maven Integration plugin</a>
		<input checked="" type="checkbox"/>	<a href="#">Hudson Seleniumhq plugin</a> This plugin integrates <a href="#">Seleniumhq</a> to Hudson.
		<input checked="" type="checkbox"/>	<a href="#">Selenium HTML report</a>
		<input checked="" type="checkbox"/>	<a href="#">SSH Slaves plugin</a>
		<input checked="" type="checkbox"/>	<a href="#">Subversion Plugin</a>

## 5) Criando um job de exemplo

Agora clique no link Jenkins no canto superior esquedo.  
Depois clique no item New job para criarmos um novo projeto de exemplo



## 6) Selecionando o tipo de job

No campo **Job Name** insira o nome **Selenium IDE** e selecione o item **Build a free-style software project**

Após isso clique no botão **OK**

Job name

**Build a free-style software project**  
This is the central feature of Jenkins. J  
than software build.

**Build a maven2/3 project**  
Build a maven2 project. Jenkins takes

**Build multi-configuration project**  
Suitable for projects that need a large

**Monitor an external job**  
This type of job allows you to record t  
dashboard of your existing automatio

## 7) Adicione o caminho do SeleniumRC

Va em Jenkins/Manage Jenkins/  
**Configure System** e no item  
**Selenium Remote Control** e  
seleccione o caminho do  
Selenium RC

**Selenium Remote Control**

htmlSuite Runner	<input type="text" value="C:\selenium-server-standalone-2.3.0.jar"/>
	<input type="text" value="selenium-server.jar path"/>

## 8) Criando a build para o Selenium

No item Build clique sobre **Add build step** e selecione **SeleniumHQ htmlsuite Run**

Apos isso preencha os campos na tela:

- **browser**
- **startURL**
- **suiteFile**
- **resultFile**
- **other**

### Build

Add build step ▾

- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke top-level Maven targets
- SeleniumHQ htmlSuite Run**

### Build

#### SeleniumHQ htmlSuite Run

browser \*firefox

startURL http://eliasnogueira.com/selenium/exercicios/selenium\_

suiteFile C:\Users\Elias\Desktop\Scripts\_Selenium\_IDE\SuiteElér

resultFile tests\resultado.html

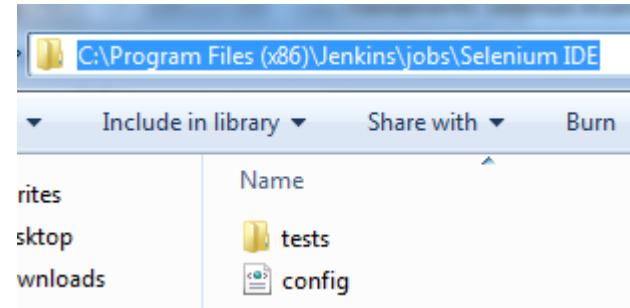
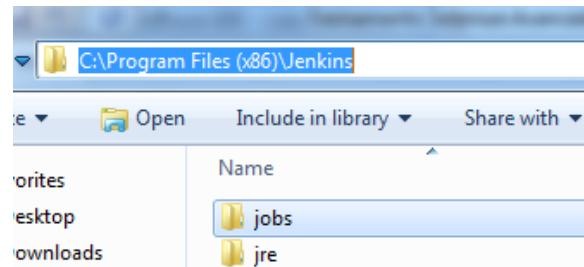
other

Delete

## 9) Criando a pasta de resultados

Como no exemplo estamos com uma pasta de resultados vamos criar a pasta **tests** no projeto. Para isso va ate o diretório de instalação do **Jenkins/jobs/Selenium IDE**

Crie uma pasta com o nome **tests**



## 10) Publicando o relatório para visualização

Na item **Post-build Actions** marque a checkbox **Publish Selenium Html Report** e no campo **Selenium tests results location** coloque o nome da pasta de teste, no caso **tests**

Após clique no botão **Save**

### Post-build Actions

- Aggregate downstream test results
- Archive the artifacts
- Build other projects
- Publish JUnit test result report
- Publish Javadoc
- Publish Selenium Html Report

Selenium tests results location

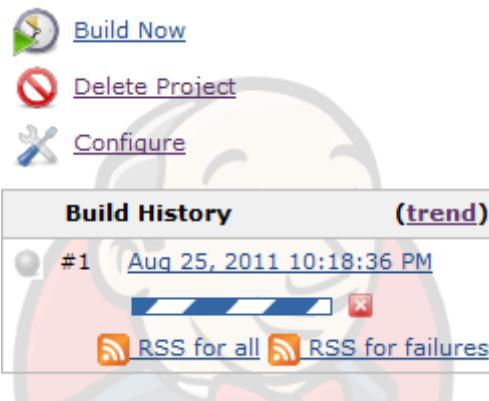
## 11) Executando a build

Após as configurações e hora de executar a build.

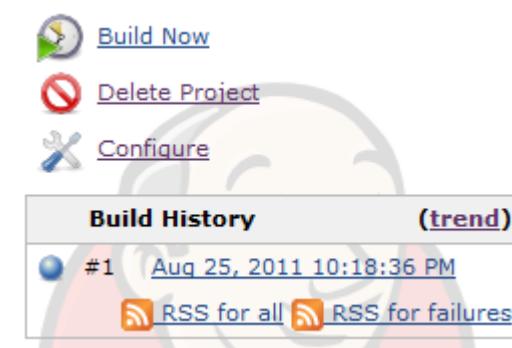
Clique no link **Build Now**

A box **Build History** começará a processar a build e, no final apresentara um “bolinha” azul, informando que o resto ocorreu com sucesso

### Executando



### Finalizado



## 12) Visualizando o relatório

Clique sobre o nome gerado da execução (uma data). Haverá um link chamado **Selenium Html Report**. Clique sobre ele. Ele irá apresentar todas as execuções

Jenkins

Jenkins \* Selenium IDE \* #1

- [Back to Project](#)
- [Status](#)
- [Changes](#)
- [Console Output](#)
- [Edit Build Information](#)
- [Selenium Html Report](#)

Jenkins

Jenkins \* Selenium IDE \* #1 \* Selenium Html Report

ENABLE AUTO REFRESH

The Selenium test reports.

Result	Name	Tests total	Tests passes	Tests failures	Commands passes	Commands failures	Commands errors	Duration
✓	<a href="#">resultado.html</a>	1	1	0	1	0	0	1
	Summen	1	1	0	1	0	0	1 s = 0.0 min and 1 s

A photograph of six people (three men and three women) standing behind a large white rectangular sign. They are all smiling and looking towards the camera. The sign is blank, except for the text "WebDriver + Jenkins" and "com JUnit" which is overlaid on the image. The background is plain white.

**WebDriver + Jenkins**  
**com JUnit**

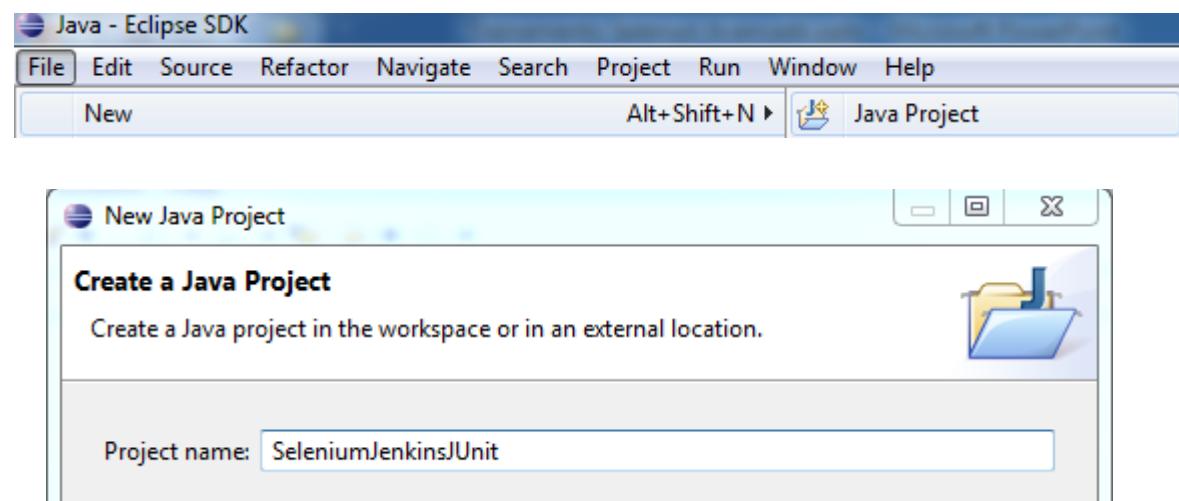
# WebDriver + Jenkins + JUnit

- Outra forma de executar nossos testes pelo Jenkins é com o projeto Java criado
- Para que isso seja possível é necessário utilizar alguma ferramenta de tarefas automatizadas
- Nos exemplos utilizaremos a ferramenta ANT para a geração das tarefas automatizadas
- Exemplo funcional em **Treinamento Selenium Avançado\Projetos\_Java\SeleniumJenkinsJUnit**

## 1) Novo projeto

Criaremos um novo projeto no Eclipse. Para isso abre o Eclipse e selecione o menu **File/New/Java Project**

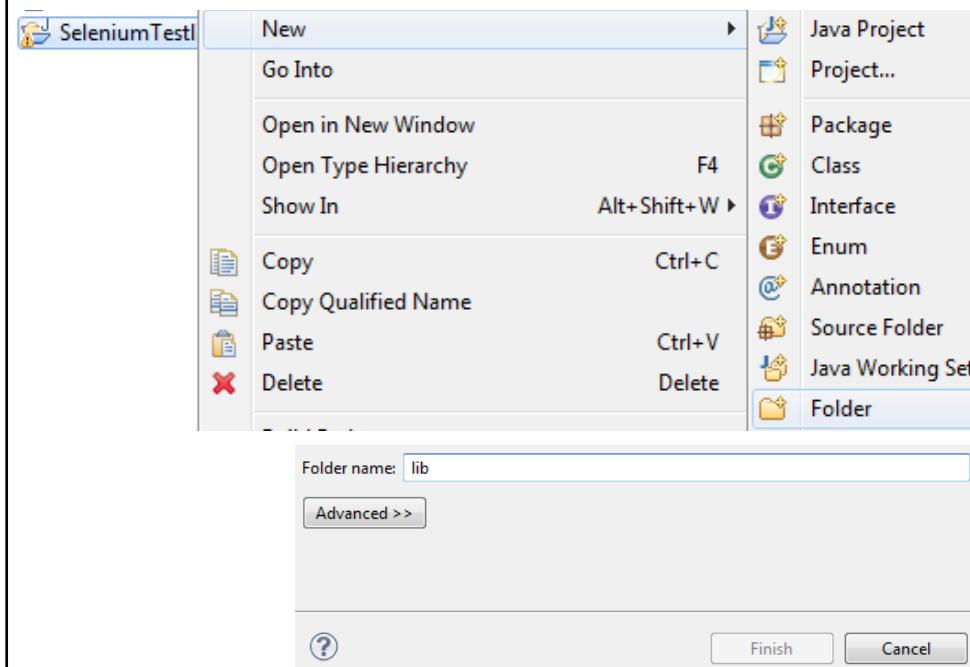
No campo **Project name** coloque o nome do projeto como “**SeleniumJenkinsJUnit**” e clique no botão **Finish**



## 2) Criar pasta lib e adicionar as bibliotecas

Agora clique com o botão direito sobre o nome do projeto e selecione o menu **New/Folder**

Na tela **New Folder** coloque no campo **Folder name** o nome **lib**



### 3) Copiar as bibliotecas para o projeto

Em nosso projeto de exemplo as bibliotecas já estão adicionadas, porém os links de cada biblioteca serão listadas ao lado.

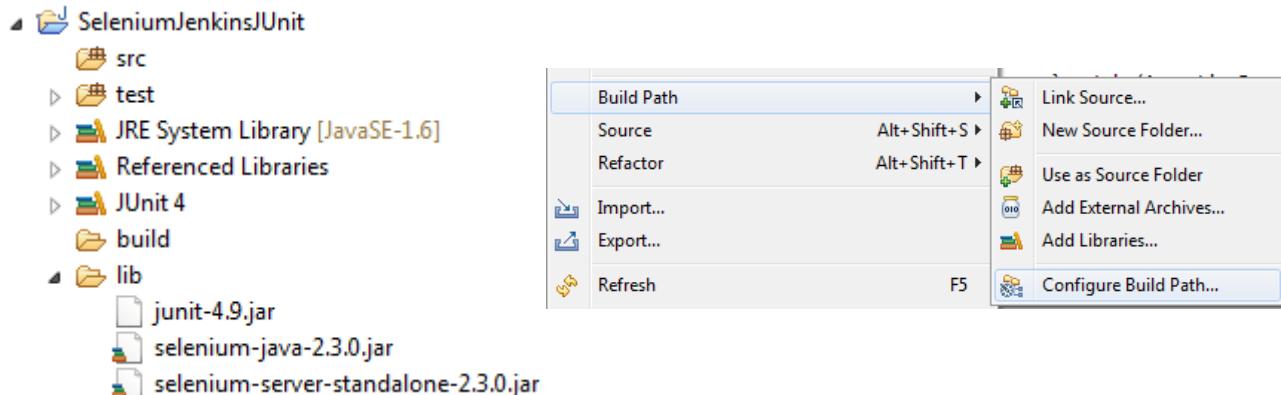
Os arquivos .jar devem ser copiados para **a pasta do usuário/workspace**

- Selenium Server
- Selenium Java Client Driver
- JUnit library (download do JUnit, necessário para a execução da build)

### 4) Atualizando e adicionando as bibliotecas

Após o download e a copia para a pasta **lib** clique bom o botão direito sobre o projeto e selecione **Refresh**.

Após isso clique bom o botão direito novamente e selecione **Build Path/Configure Build Path**

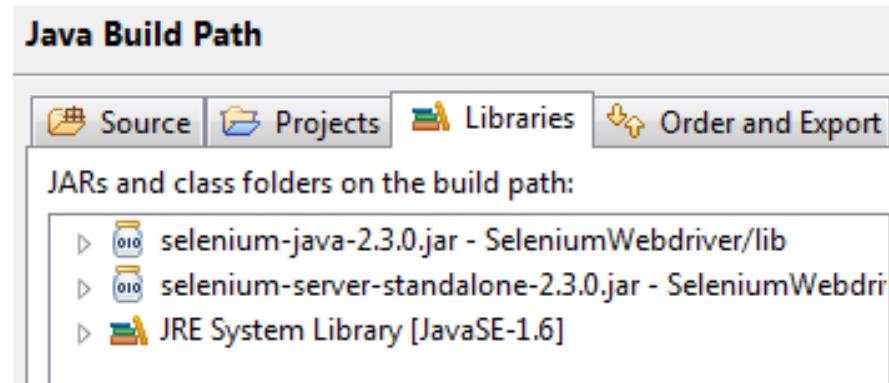
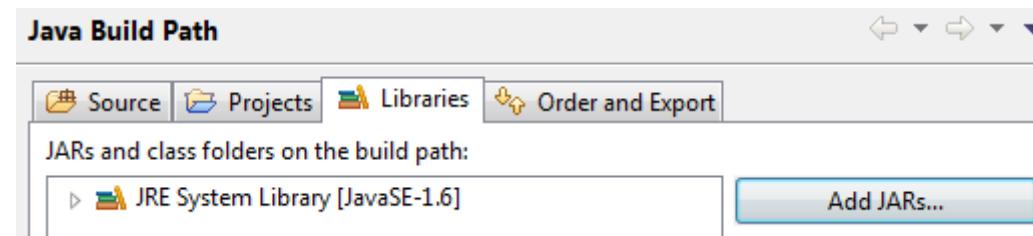


## 5) Adicionando as bibliotecas

Clique sobre a aba **Libraries** e depois sobre o botão **Add Jars...**.

Na tela apresentada selecione o projeto e va até a pasta **lib**. Selecione todos os arquivos menos o **junit-4.9.jar** e clique em OK

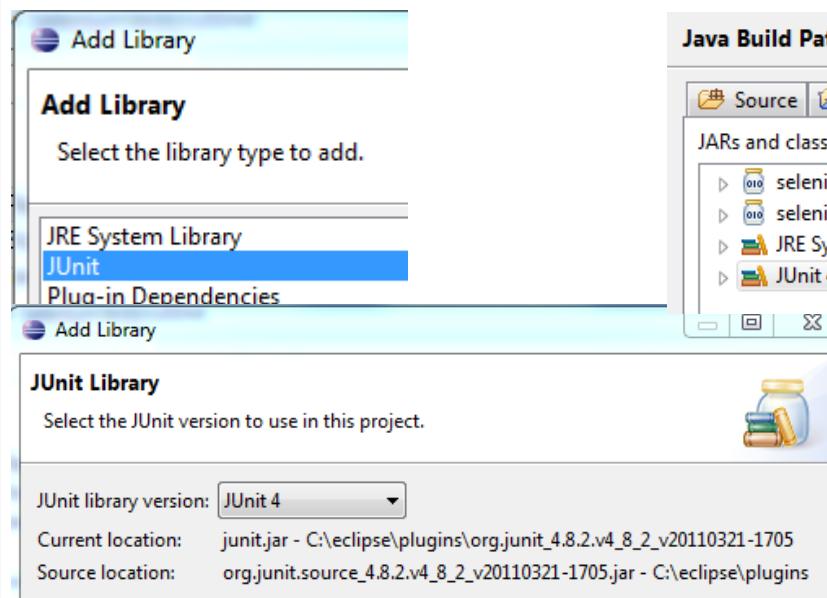
Após isso todos os arquivos estarão listados na aba **Libraries**. Clique no botão OK



## 6) Adicionando a Library do JUnit

Ainda no **Build Path** sobre a aba **Libraries** clique no botão **Add Library**.

Selecione **JUnit** e clique em **Next**. Após isso selecione **JUnit 4** na combo **JUnit library version** e clique em **OK**



## 7) Criação do pacote

Vá até a pasta `src`, clique bom o botão direito e selecione **Package**.

Em **Name** coloque qualquer nome de sua escolha. No exemplo nome utilizado é `com.eliasnogueira.seleniumjenkins.test`

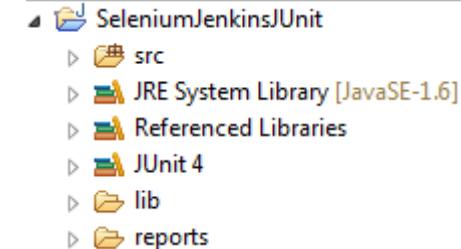
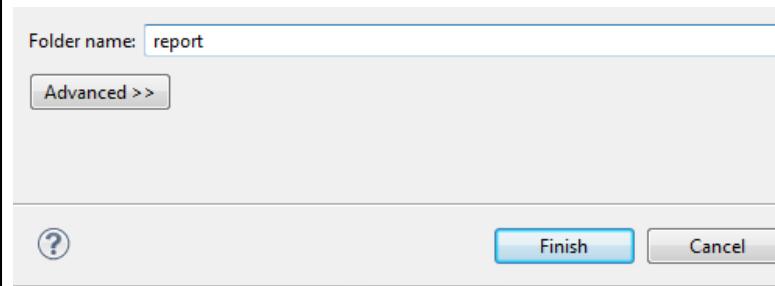
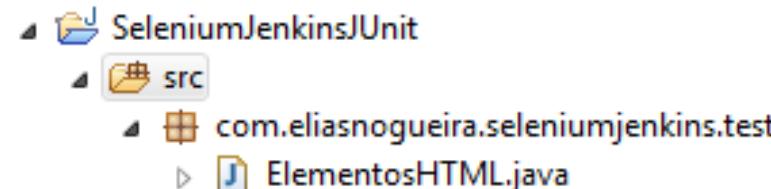
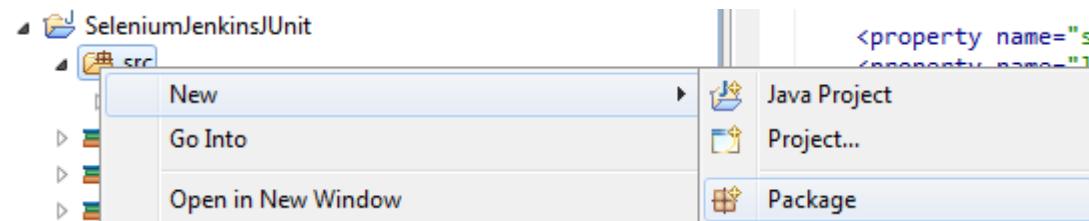
Clique no botão **Finish**

Utilizaremos aqui o exemplo do exercício **ElementosHTML**

## 8) Criação da pasta de relatórios

O JUnit irá gerar relatórios que utilizaremos mais tarde.

Para criar a pasta de relatórios clique com o botão direito sobre o nome do projeto e selecione **New** e selecione **Folder**. Em **Folder name** coloque “reports” e clique em **Finish**

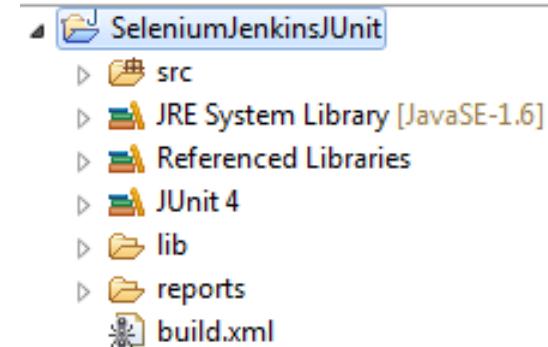
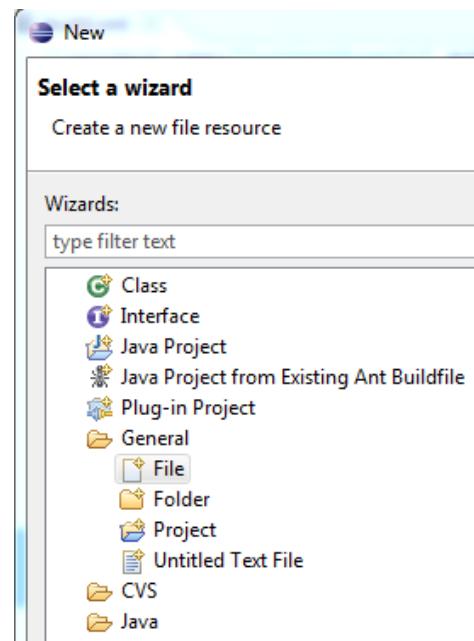


## 9) Criando a build ANT

Agora será necessário criar a build com o ANT.

Para isso clique com o botão direito no nome do projeto e selecione **New/Other**. Na janela apresentada selecione **General/File** e clique em **Next**

Apos isso coloque em **File name** “**build.xml**” e clique em **Finish**



## 10) Criando a base do arquivo

Abra o arquivo, que estará em branco e copie o mesmo texto ao lado.

As *targets* (tarefas que serão executadas) estão vazias neste primeiro momento

 <target name="compile-test"> </target> <target name="test"> </target> </project>" data-bbox="318 560 950 875"/>

```
<project name="JunitAntExample" default="test" basedir=".">
  <target name="compile-test">
  </target>
  <target name="test">
  </target>
</project>
```

## 11) Adicionando os caminhos e classpaths no build ANT

Antes de criar as atividades automáticas das *targets* precisamos criar os caminhos e o *classpath*. Os caminhos são definidos pela tag **property**.

Na no *classpath* nos criamos uma tag **path** para incluir todas as classes necessárias e as bibliotecas (.jar) para que não ocorram erros no momento de execução das *targets*.

Copie os mesmos comandos da tela abaixo entre o projeto (**project**) e as *targets*



```
<project name="JunitAntExample" default="test" basedir=".">
    <property name="src" value="src" />
    <property name="lib" value="lib" />
    <property name="bin" value="bin" />
    <property name="test.reports" value="reports" />

    <path id="test.classpath">
        <pathelement location="${bin}" />
        <pathelement location="lib/junit-4.9.jar" />
        <fileset dir="${lib}">
            <include name="**/*.jar"/>
        </fileset>
    </path>
```

## 12) Criando os comandos para target de compilação

A target de compilação é necessária para que a cada posterior execução automatizada do *Jenkins* todo o código de teste seja compilado para executar de maneira correta (sem que existam “sujeiras”)

Utilizamos a *tag javac* para compilar todas as classes de um diretório (*src*) e colocá-las (arquivos *.class*) em outro diretório (*bin*) utilizando o *classpath* (bibliotecas que adicionamos) para que nenhum erro ocorra

Copie o conteúdo da imagem abaixo e coloque dentro da *target* de compilação

```
<target name="compile-test">
    <javac srcdir="${src}" destdir="${bin}" verbose="true" includeantruntime="true">
        <classpath refid="test.classpath" />
    </javac>
</target>
```

## 12) Criando os comandos para target de execução – parte 1

Para garantir que teremos sempre o ultimo relatório de execução utilizamos uma *tag* de remoção de diretório (**delete**) e outra de criação (**mkdir**)

Após utilizados a *tag* do JUnit para executar os testes, onde ele executará todas as classes contidas na pasta **src** e criara um arquivo após a execução no formato XML com os resultados.

Copie o conteúdo abaixo e insira na **target test**

```
<target name="test" depends="compile-test">
    <delete dir="${test.reports}" />
    <mkdir dir="${test.reports}" />

    <junit fork="yes" printsummary="yes" haltonfailure="no" showoutput="yes">
        <batchtest fork="yes" todir="${test.reports}" >
            <fileset dir="${src}">
                <include name="**/*.java" />
            </fileset>
        </batchtest>
        <formatter type="xml" />
        <classpath refid="test.classpath" />
    </junit>
```

### 13) Criando os comandos para target de execução – parte 2

Agora será feita a geração automática do relatório de execução em HTML.

Para isso a *tag junitreport* é utilizada pegando o arquivo XML gerado na execução dos testes e jogando na pasta de relatório (*reports*)

Copie o conteúdo abaixo e adicione após a *tag </junit>*

```
<junitreport todir="${test.reports}">
    <fileset dir="${test.reports}">
        <include name="TEST-*.xml" />
    </fileset>
    <report todir="${test.reports}" />
</junitreport>

</target>
```

## 14) Garantindo a execução da build

Antes de passar para o *Jenkins* vamos executar a build. Para isso clique com o botão direito sobre o arquivo **build.xml** e selecione **Run As/Ant Build**

O Console do Eclipse começará a mostrar a execução de compilação e execução e o teste iniciará.

Após a finalização do teste podemos ver o relatório gerado na pasta **reports**

The screenshot shows the Eclipse IDE interface. On the left, there's a 'Run As' context menu open over a file named 'build.xml'. The menu items include '1 Ant Build' (selected), '2 Ant Build...', and 'External Tools Configurations...'. Below the menu is a 'Console' view showing the execution of an Ant build. The output in the console is as follows:

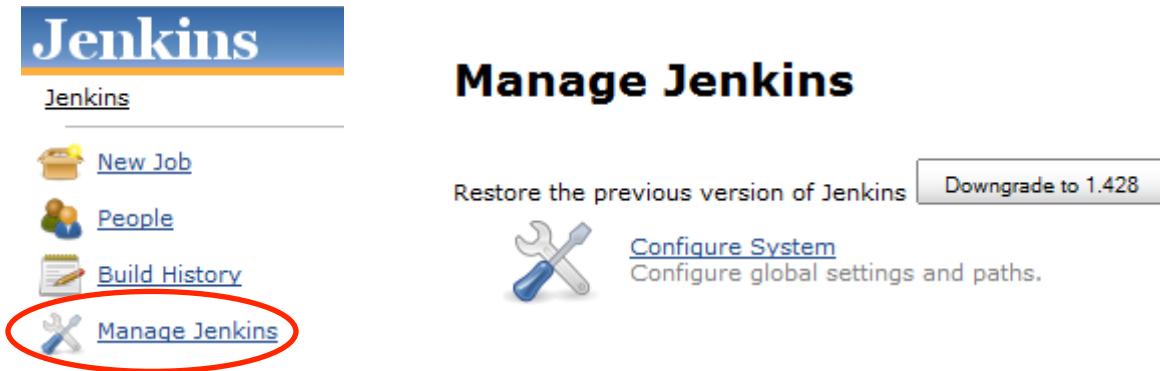
```
<terminated> SeleniumJenkinsJUnit build.xml [Ant Build] C:\Program Files\Java\jre6\bin\javaw.exe (Sep 10, 2011)
Buildfile: C:\Users\Elias\workspace\SeleniumJenkinsJUnit\build.xml
compile-test:
test:
    [delete] Deleting directory C:\Users\Elias\workspace\SeleniumJenkinsJUnit\reports
    [mkdir] Created dir: C:\Users\Elias\workspace\SeleniumJenkinsJUnit\reports
    [junit] Running com.eliasnogueira.seleniumjenkins.test.ElementosHTML
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 31.502 sec
[junitreport] Processing C:\Users\Elias\workspace\SeleniumJenkinsJUnit\reports\TESTS-
[junitreport] Loading stylesheet jar:file:/C:/eclipse/plugins/org.apache.ant_1.8.2.v2
[junitreport] Transform time: 943ms
[junitreport] Deleting: C:\Users\Elias\AppData\Local\Temp\null256997950
BUILD SUCCESSFUL
Total time: 34 seconds
```

To the right of the console is a 'Navigator' view showing the project structure. It includes a 'SeleniumJenkinsJUnit' folder containing 'src', 'Referenced Libraries', 'JUnit 4', 'lib', and a 'reports' folder. The 'reports' folder contains subfolders 'com' and 'all-tests.html', along with other files like 'allclasses-frame.html', 'alltests-errors.html', 'alltests-fails.html', 'index.html', 'overview-frame.html', 'overview-summary.html', 'stylesheet.css', 'TEST-com.eliasnogueira.seleniumjuni', and 'TESTS-TestSuites.xml'. A 'build.xml' file is also listed at the bottom of the 'reports' folder.

## 15) Configurando o Jenkins

Vá até o Jenkins e clique sobre **Manage Jenkins**

Após clique no link **Configure System**



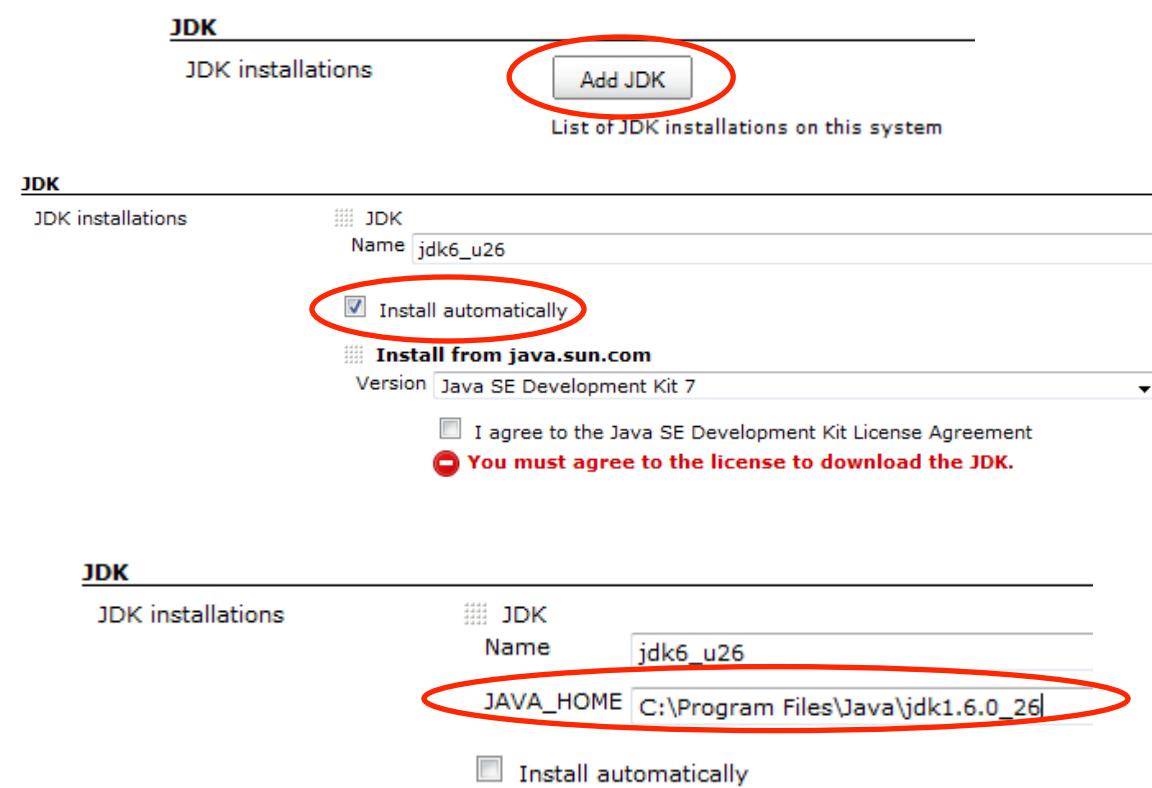
The screenshot shows the Jenkins interface with the title "Manage Jenkins". In the sidebar on the left, there are several links: "New Job", "People", "Build History", and "Manage Jenkins". The "Manage Jenkins" link is highlighted with a red circle. On the right side, there is a "Configure System" button with the sub-instruction "Configure global settings and paths".

## 16) Configurando o Jenkins - JDK

No item **JDK** clique no botão **Add JDK**.

Depois dê um nome para a JDK no campo **Name** e desmarque a opção **Install automatically**.

Agora no campo **JAVA\_HOME** insira o caminho da JDK do seu computador



The screenshots show the "JDK installations" section of the Jenkins configuration. The first screenshot shows the "Add JDK" button circled in red. The second screenshot shows a new entry for "jdk6\_u26" with the "Install automatically" checkbox checked. A note below says "You must agree to the license to download the JDK." The third screenshot shows the "JAVA\_HOME" field set to "C:\Program Files\Java\jdk1.6.0\_26", which is also circled in red.

## 17) Configurando o Jenkins - ANT

No item **Ant** clique no botão **Add Ant**.

Depois dê um nome para a Ant no campo **Name** e deixe marcada a opção **Install automatically**.

Após isso clique no botão **Save**

Ant
Ant installations

Name

Install automatically

Install from Apache  
Version

## 18) Criando o job no Jenkins

Vá até o Jenkins e clique sobre o link **New Job**

Após, em **Job name**, coloque “**Exemplo JUnit Jenkins**” e clique no botão **OK**

Jenkins

[Jenkins](#)

[!\[\]\(566f29a7c7ec393e917ff96204034ab8\_img.jpg\) New Job](#)

[!\[\]\(184750c149ee2565668ca0c173dfe883\_img.jpg\) People](#)

[!\[\]\(35ab5041e965be13d3917a4cef5f266f\_img.jpg\) Build History](#)

[!\[\]\(0814e3fcb719a991c24c503abf8057c5\_img.jpg\) Manage Jenkins](#)

Job name

**Build a free-style software project**

This is the central feature of Jenkins.  
than software build.

## 19) Configurando o projeto – parte 1

Como criamos o projeto na nossa maquina sem a utilização de algum controle de versão iremos apontar para o projeto em nossa maquina.

Clique no botão **Advanced** em **Advanced Project Options**

Após selecione o item **Use custom workspace** e coloque o caminho completo para o projeto

### Advanced Project Options

**Advanced...**

### Advanced Project Options

- Quiet period
- Retry Count
- Block build when upstream project is building
- Block build when downstream project is building
- Use custom workspace

Directory

C:\Users\Elias\workspace\SeleniumJenkins\JUnit

## 20) Configurando o projeto – parte 2

Agora iremos fazer o link entre o projeto e o Jenkins para a execução automática.

Em **Build** clique no botão **Add build step** e selecione **Invoke Ant**

Agora em **Ant Version**

selecione o nome criado na configuração anterior do ANT e em **Targets** escreva **test**

### Build

**Add build step ▾**

- Execute Windows batch command
- Execute shell
- Invoke Ant**
- Invoke top-level Maven targets
- SeleniumHQ htmlSuite Run

### Build

**Invoke Ant**

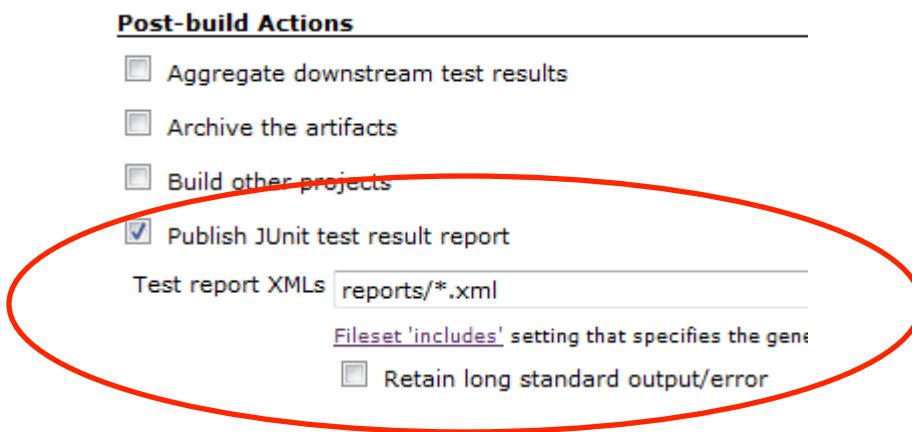
Ant Version **default**

Targets **test**

## 21) Configurando o projeto – parte 3

Em Post-build Actions selecione o item **Publish JUnit test result report** e no campo **Test report XMLs** coloque `reports/*.xml`

Isso fará com que o relatório seja exibido no dashboard do projeto



## 22) Executando o projeto

Depois de toda a configuração basta clicarmos no link **Build Now**

A box de **Build History** começará a executar apresentando uma barra branca e azul.

Aguarde até o fim da execução.

The screenshot shows the Jenkins dashboard for the project "Exemplo JUnit Jenkins". The top navigation bar has a blue header with the Jenkins logo and the URL "Jenkins \* Exemplo JUnit Jenkins". Below the header is a horizontal menu with icons and links: "Back to Dashboard" (green arrow), "Status" (magnifying glass), "Changes" (notepad), "Workspace" (blue folder), "Build Now" (hourglass icon circled in red), "Delete Project" (red circle with slash), and "Configure" (gear icon). A large red oval highlights the "Build History" section at the bottom left. This section contains a table with one row: "#1 Sep 10, 2011 7:51:18 PM". To the right of the table is a progress bar with a white bar and a red X button. Below the progress bar are two RSS feed icons: "RSS for all" and "RSS for failures". To the right of the dashboard, the project name "Project Exemplo JUnit Jenkins" is displayed in large bold letters. Below the project name are two links: "Workspace" (blue folder icon) and "Recent Changes" (notepad icon). At the bottom, there is a "Permalinks" section with a single item: "Last build (#1), 1.1 sec ago".

## 23) Visualizando a execução do projeto

Após a execução do projeto a box de **Build History** deve conter a execução com uma bola verde ou azul ao lado da data de execução.

Clique sobre a data para exibir os detalhes da execução

### Jenkins

[Jenkins](#) » [Exemplo JUnit Jenkins](#) » #1

 [Back to Project](#)

 [Status](#)

 [Changes](#)

 [Console Output](#)

 [Edit Build Information](#)

 [Test Result](#)



**Build #1 (Sep 10, 2011 7:51:18 PM)**



No changes.



Started by anonymous user



[Test Result](#) (no failures)

## 24) Visualizando a execução do projeto - Console

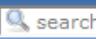
Para visualizar o console (saída das execuções da build) clique no link **Console Output**

Se algum erro ocorrer conseguiremos visualizar o detalhe nesta seção.

Todas as *targets* executadas pela build estão listadas na box **Executed Ant Targets**

**Jenkins**

Jenkins » Exemplo JUnit Jenkins » #1

 search

 [Back to Project](#)

 [Status](#)

 [Changes](#)

 [Console Output](#)

 [Edit Build Information](#)

 [Test Result](#)

**Executed Ant Targets**

- [compile-test](#)
- [test](#)

**Console Output**

```
Started by user anonymous
[SeleniumJenkinsJUnit] $ cmd.exe /C ""C:\Program Files (x86)\Jenkins\tools\default\bin\ant.bat" test && exit %ERR%
Buildfile: C:\Users\Elias\workspace\SeleniumJenkinsJUnit\build.xml

compile-test:

test:
[delete] Deleting directory C:\Users\Elias\workspace\SeleniumJenkinsJUnit\reports
[mkdir] Created dir: C:\Users\Elias\workspace\SeleniumJenkinsJUnit\reports
[junit] Running com.eliasnogueira.seleniumjenkins.test.ElementosHTML
[junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 51.789 sec
[junitreport] Processing C:\Users\Elias\workspace\SeleniumJenkinsJUnit\reports\TESTS-TestSuites.xml to C:\Windows\TEMP\TESTS-TestSuites.html
[junitreport] Loading stylesheet jar:file:/C:/Program%20Files%20(x86)/Jenkins/tools/default/lib/ant-junit.jar!/org/
optional/junit/xsl/junit-frames.xsl
[junitreport] Transform time: 1681ms
[junitreport] Deleting: C:\Windows\TEMP\null11208254363

BUILD SUCCESSFUL
Total time: 1 minute 2 seconds
Recording test results
Finished: SUCCESS
```

## 25) Visualizando a execução do projeto – Resultados de Teste

Para visualizar o resultado de teste com o relatório do JUnit agregado clicamos no link **Test Result**  
Ele mostrará a execução de todos os testes (classes) em todos os pacotes executados

### Jenkins

[Jenkins](#) » [Exemplo JUnit Jenkins](#) » [#1](#) » [Test Results](#)

 [Back to Project](#)

 [Status](#)

 [Changes](#)

 [Console Output](#)

 [Edit Build Information](#)

 [History](#)

 [Test Result](#)

### Test Result

0 failures

### All Tests

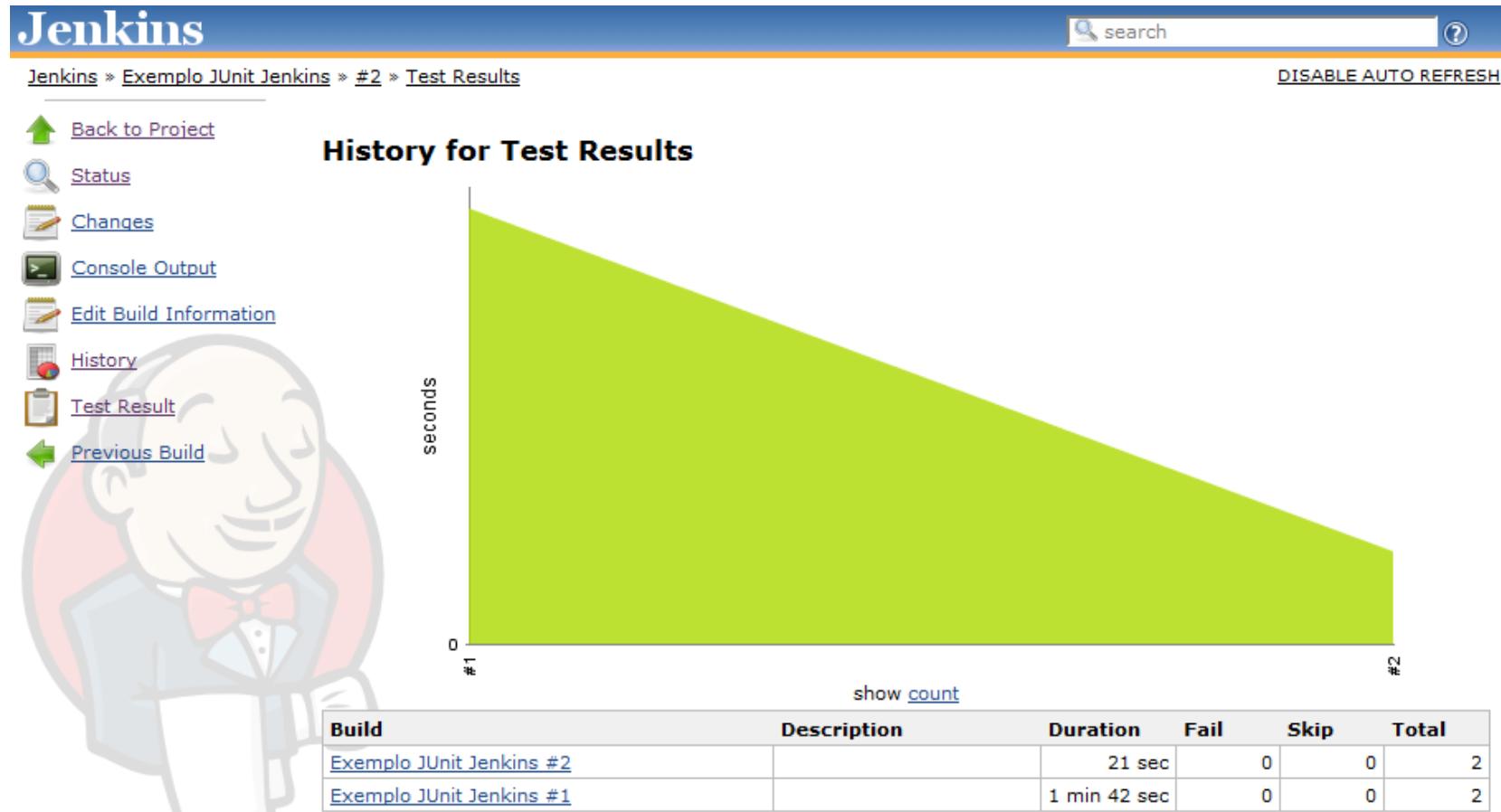
Package	Duration
<a href="#">com.eliasnoqueira.seleniumjenkins.test</a>	1 min 42 sec

## 26) Visualizando a execução do projeto – Histórico do Teste

Dentro do resultado da build é possível visualizar o histórico de execução.

Para acessar o histórico clicamos no link **History**.

São exibidos os dados de cada build e um gráfico de tempo de execução ou de contagem de execuções



A photograph of six people (three men and three women) standing behind a large white rectangular sign. They are all smiling and looking towards the camera. The sign is blank, intended for the text to be overlaid.

**WebDriver + Jenkins  
com TestNG**

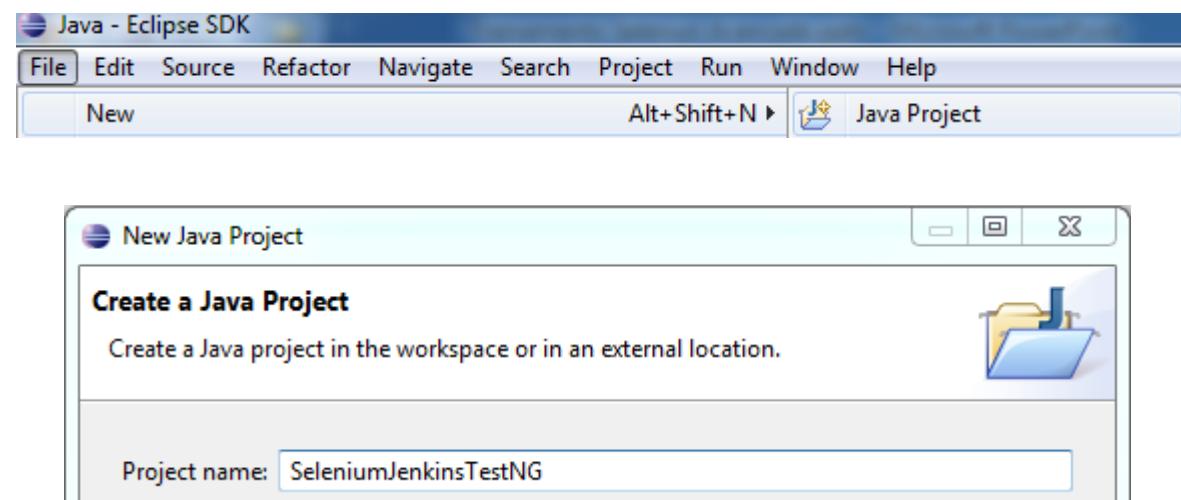
# WebDriver + Jenkins + TestNG

- Trabalha da mesma forma do JUnit, porém podemos utilizar o arquivo **testng.xml** para o controle da execução do teste
- Exemplo funcional em **Treinamento Selenium Avancado\Projetos\_Java\SeleniumJenkinsTestNG**

## 1) Novo projeto

Criaremos um novo projeto no Eclipse. Para isso abre o Eclipse e selecione o menu **File/New/Java Project**

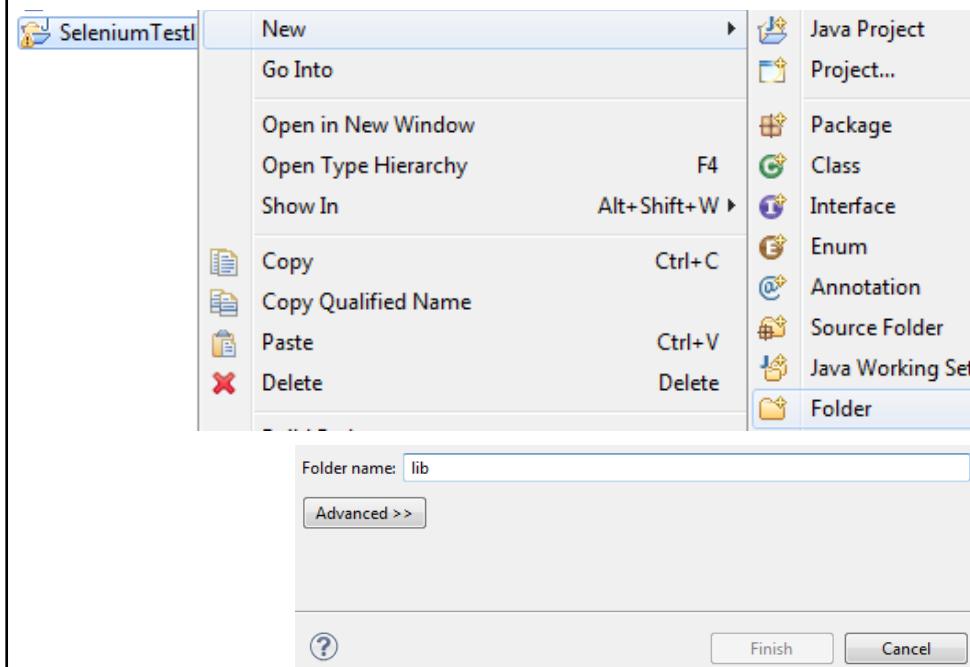
No campo **Project name** coloque o nome do projeto como “**SeleniumJenkinsTestNG**” e clique no botão **Finish**



## 2) Criar pasta lib e adicionar as bibliotecas

Agora clique com o botão direito sobre o nome do projeto e selecione o menu **New/Folder**

Na tela **New Folder** coloque no campo **Folder name** o nome **lib**



### 3) Copiar as bibliotecas para o projeto

Em nosso projeto de exemplo as bibliotecas já estão adicionadas, porém os links de cada biblioteca serão listadas ao lado.

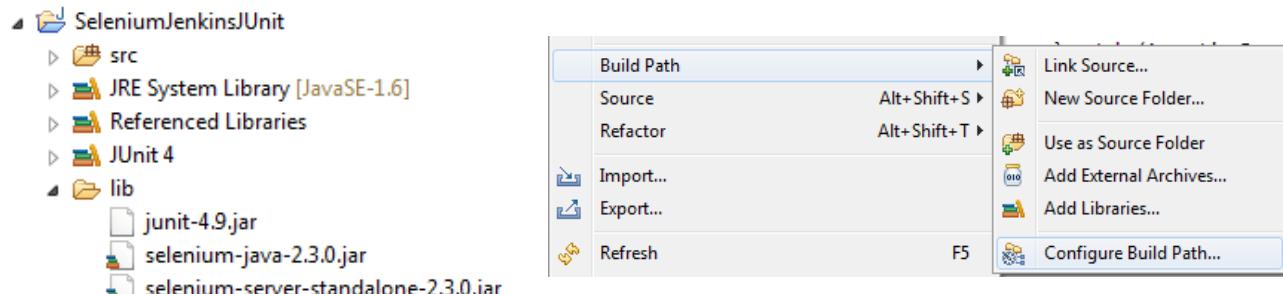
Os arquivos .jar devem ser copiados para **a pasta do usuário/workspace**

- Selenium Server
- Selenium Java Client Driver
- TestNG library (download do TesNG, necessário para a execução da build)

### 4) Atualizando e adicionando as bibliotecas

Após o download e a copia para a pasta **lib** clique bom o botão direito sobre o projeto e selecione **Refresh**.

Após isso clique bom o botão direito novamente e selecione **Build Path/Configure Build Path**

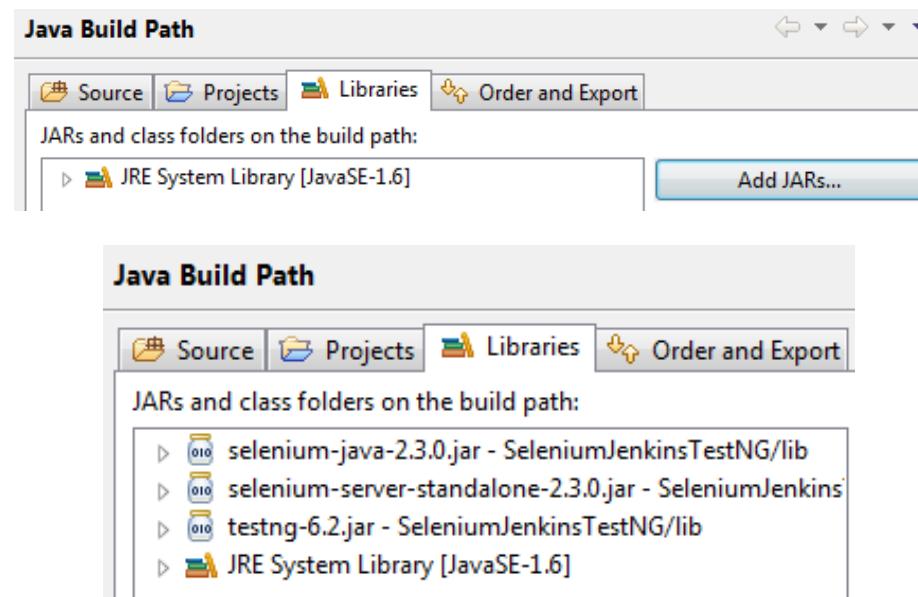


## 5) Adicionando as bibliotecas

Clique sobre a aba *Libraries* e depois sobre o botão *Add Jars...*

Na tela apresentada selecione o projeto e va até a pasta *lib*. Selecione todos os arquivos e clique em OK

Após isso todos os arquivos estarão listados na aba *Libraries*. Clique no botão OK



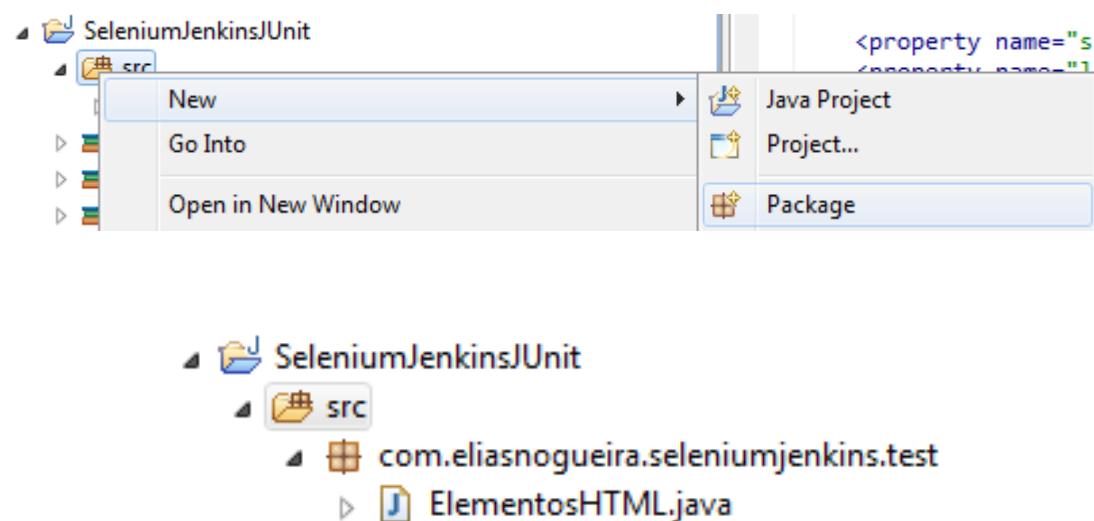
## 6) Criação do pacote

Vá até a pasta *src*, clique bom o botão direito e selecione *Package*.

Em *Name* coloque qualquer nome de sua escolha. No exemplo nome utilizado é *com.eliasnogueira.seleniumjenkins.test*

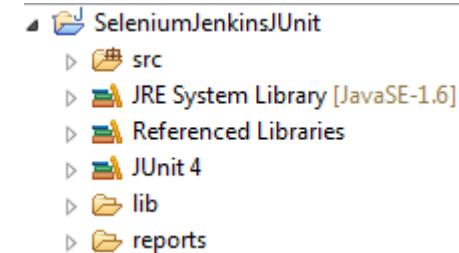
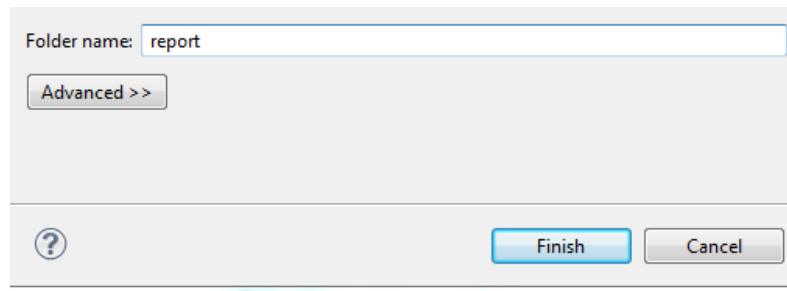
Clique no botão *Finish*

Utilizaremos aqui o exemplo do exercício *ElementosHTML*



## 7) Criação da pasta de relatórios

O JUnit irá gerar relatórios que utilizaremos mais tarde. Para criar a pasta de relatórios clique com o botão direito sobre o nome do projeto e selecione **New** e selecione **Folder**. Em **Folder name** coloque “reports” e clique em **Finish**

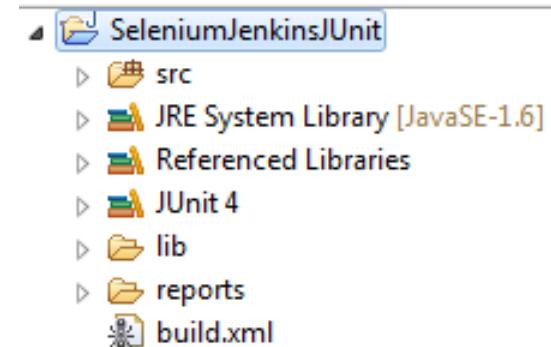
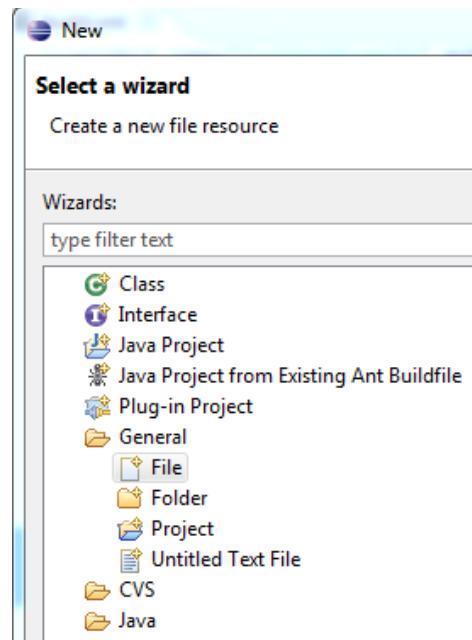


## 8) Criando a build ANT

Agora será necessário criar a build com o ANT.

Para isso clique com o botão direito no nome do projeto e selecione **New/Other**. Na janela apresentada selecione **General/File** e clique em **Next**

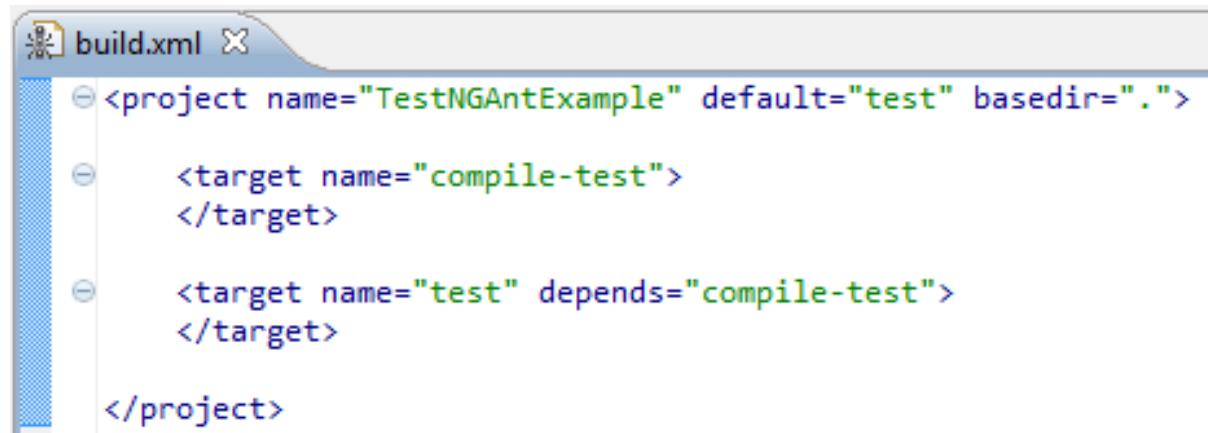
Após isso coloque em **File name** “build.xml” e clique em **Finish**



## 9) Criando a base do arquivo

Abra o arquivo, que estará em branco e copie o mesmo texto ao lado.

As *targets* (tarefas que serão executadas) estão vazias neste primeiro momento



```
<project name="TestNGAntExample" default="test" basedir=".">
    <target name="compile-test">
    </target>
    <target name="test" depends="compile-test">
    </target>
</project>
```

## 10) Adicionando os caminhos e classpaths no build ANT

Antes de criar as atividades automáticas das *targets* precisamos criar os caminhos e o *classpath*. Os caminhos são definidos pela tag **property**.

Na no *classpath* nos criamos uma tag **path** para incluir todas as classes necessárias e as bibliotecas (.jar) para que não ocorram erros no momento de execução das *targets*.

Copie os mesmos comandos da tela abaixo entre o projeto (**project**) e as *targets*

A screenshot of an IDE interface showing the build.xml file. The window title is "build.xml". The XML code is displayed with color-coded syntax highlighting: blue for tags, green for attributes, and purple for values. The code defines a project named "TestNGAntExample" with a default target "test" and a basedir ". ". It includes four properties: "src", "lib", "bin", and "test.reports". A "path" element is defined with an id "test.classpath", containing a "pathelement" pointing to "\${bin}" and a "fileset" with a "dir" of "\${lib}" and an "include" pattern of "\*\*/\*.jar".

```
<project name="TestNGAntExample" default="test" basedir=".">
    <property name="src" value="src" />
    <property name="lib" value="lib" />
    <property name="bin" value="bin" />
    <property name="test.reports" value="reports" />

    <path id="test.classpath">
        <pathelement location="${bin}" />
        <fileset dir="${lib}">
            <include name="**/*.jar"/>
        </fileset>
    </path>
```

## 11) Criando os comandos para target de compilação

A target de compilação é necessária para que a cada posterior execução automatizada do *Jenkins* todo o código de teste seja compilado para executar de maneira correta (sem que existam “sujeiras”)

A tag taskdef vai informar ao script para utilizar a classe e as tags do TestNG

Utilizamos a tag **javac** para compilar todas as classes de um diretório (*src*) e colocá-las (arquivos *.class*) em outro diretório (*bin*) utilizando o *classpath* (bibliotecas que adicionamos) para que nenhum erro ocorra

Copie o conteúdo da imagem abaixo e coloque dentro da *target* de compilação

```
<taskdef name="testng" classname="org.testng.TestNGAntTask" classpath="lib/testng-6.2.jar"/>

<target name="compile-test">
    <javac srcdir="${src}" destdir="${bin}" verbose="true" includeantruntime="true">
        <classpath refid="test.classpath" />
    </javac>
</target>
```

## 12) Criando os comandos para target de execução

Para garantir que teremos sempre o ultimo relatório de execução utilizamos uma *tag* de remoção de diretório (**delete**) e outra de criação (**mkdir**)

Após utilizados a *tag* do tesng para executar os testes, onde ela chamará o arquivo **testng.xml** que contém o que deve ser executado no teste.

Criaremos o arquivo **testng.xml**

Copie o conteúdo abaixo e insira na **target test**

```
<target name="test" depends="compile-test">
    <delete dir="${test.reports}"/>
    <mkdir dir="${test.reports}"/>

    <testng classpathref="test.classpath" outputDir="${test.reports}" haltOnfailure="true">
        <xmlfileset dir="${basedir}" includes="testng.xml"/>
    </testng>
</target>
```

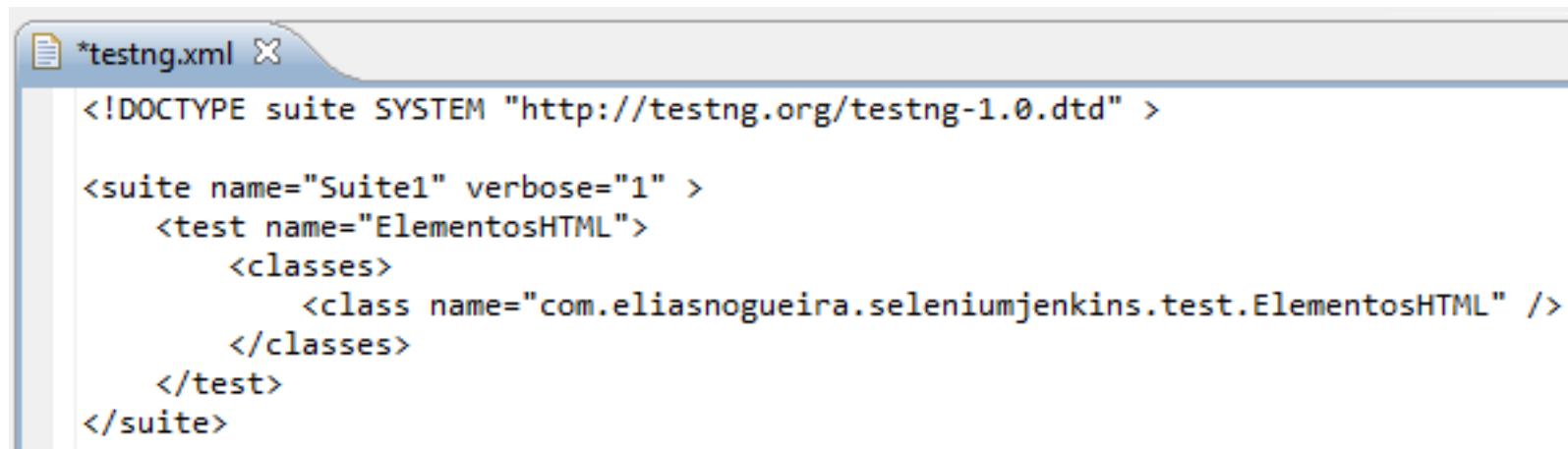
### 13) Criando o arquivo testng.xml

Quando executamos os testes através de uma build podemos (não necessariamente) criar um arquivo chamado **testng.xml** com toda a lógica de execução, que pode ser de diversas formas: única classe, pacote, grupos, etc...

Clique bom o botão direito sobre o nome do projeto e selecione **New/Other**. Na janela apresentada selecione **General/File** e clique em **Next**

Apos isso coloque em **File name “testng.xml”** e clique em **Finish**

Copie o conteúdo do arquivo abaixo e cole no arquivo **testng.xml**



```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

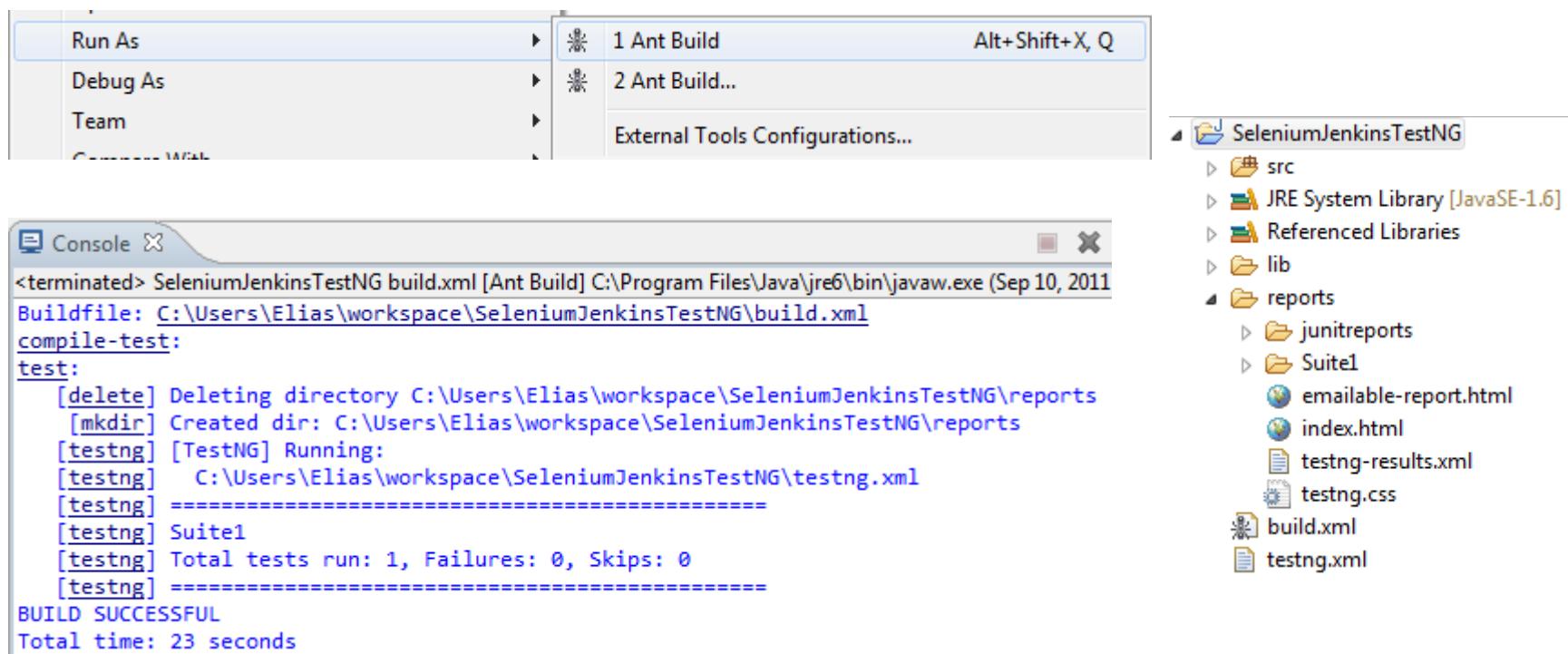
<suite name="Suite1" verbose="1" >
    <test name="ElementosHTML">
        <classes>
            <class name="com.eliasnogueira.seleniumjenkins.test.ElementosHTML" />
        </classes>
    </test>
</suite>
```

## 14) Garantindo a execução da build

Antes de passar para o *Jenkins* vamos executar a build. Para isso clique com o botão direito sobre o arquivo **build.xml** e selecione **Run As/Ant Build**

O Console do Eclipse começará a mostrar a execução de compilação e execução e o teste iniciará.

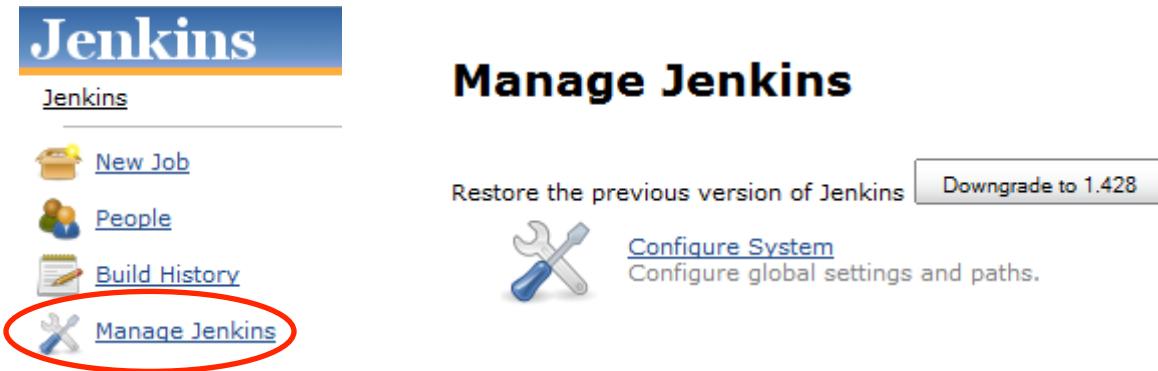
Após a finalização do teste podemos ver o relatório gerado na pasta **reports**



## 15) Configurando o Jenkins

Vá até o Jenkins e clique sobre **Manage Jenkins**

Após clique no link **Configure System**



The screenshot shows the Jenkins interface with the title "Manage Jenkins". In the sidebar on the left, there are several links: "New Job", "People", "Build History", and "Manage Jenkins". The "Manage Jenkins" link is highlighted with a red circle. On the right side, there is a "Configure System" button with the sub-instruction "Configure global settings and paths".

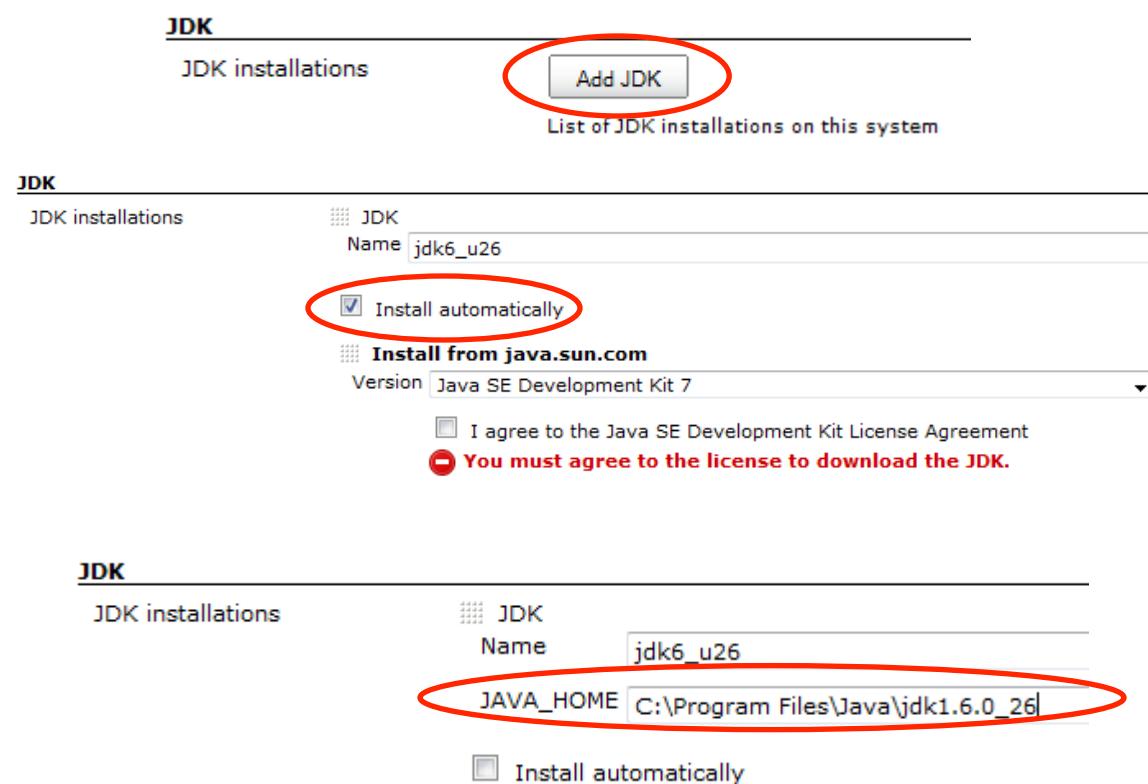
## 16) Configurando o Jenkins - JDK

No item **JDK** clique no botão **Add JDK**.

Depois dê um nome para a JDK no campo **Name** e desmarque a opção **Install automatically**.

Agora no campo **JAVA\_HOME** insira o caminho da JDK do seu computador

**OBS:** se você já fez esse passo antes, ignore-o



The screenshots illustrate the steps to add a new JDK configuration:

- The first screenshot shows the "JDK installations" section with the "Add JDK" button circled in red.
- The second screenshot shows the details for a new JDK named "jdk6\_u26". The "Install automatically" checkbox is checked and circled in red. A note at the bottom states "You must agree to the license to download the JDK".
- The third screenshot shows the same configuration after the "JAVA\_HOME" field has been filled with the path "C:\Program Files\Java\jdk1.6.0\_26".

## 17) Configurando o Jenkins - ANT

No item **Ant** clique no botão **Add Ant**.

Depois dê um nome para a Ant no campo **Name** e deixe marcada a opção **Install automatically**.

Após isso clique no botão **Save**

Ant
Ant installations

Name

Install automatically

Install from Apache  
Version

## 18) Criando o job no Jenkins

Vá até o Jenkins e clique sobre o link **New Job**

Após, em **Job name**, coloque "**Exemplo JUnit Jenkins**" e clique no botão **OK**

Jenkins

New Job

People

Build History

Manage Jenkins

Job name

Build a free-style software project  
This is the central feature of Jenkins. It can be used to build almost anything, than software build.

## 19) Configurando o projeto – parte 1

Como criamos o projeto na nossa maquina sem a utilização de algum controle de versão iremos apontar para o projeto em nossa maquina.

Clique no botão **Advanced** em **Advanced Project Options**

Após selecione o item **Use custom workspace** e coloque o caminho completo para o projeto

Advanced Project Options

**Advanced...**

Advanced Project Options

Quiet period  
 Retry Count  
 Block build when upstream project is building  
 Block build when downstream project is building  
 Use custom workspace

Directory

## 20) Configurando o projeto – parte 2

Agora iremos fazer o link entre o projeto e o Jenkins para a execução automática.

Em **Build** clique no botão **Add build step** e selecione **Invoke Ant**

Agora em **Ant Version**

selecione o nome criado na configuração anterior do ANT e em **Targets** escreva **test**

Build

**Add build step ▾**

- Execute Windows batch command
- Execute shell
- Invoke Ant**
- Invoke top-level Maven targets
- SeleniumHQ htmlSuite Run

Build

**Invoke Ant**

Ant Version   
Targets

## 21) Configurando o projeto – parte 3

Em Post-build Actions selecione o item Publish TestNG Results e no campo TestNG XML report pattern coloque reports/testng-result.xml

Isso fará com que o relatório seja exibido no dashboard do projeto

<input checked="" type="checkbox"/> Publish TestNG Results	
TestNG XML report pattern	<input type="text" value="reports/testng-results.xml"/>
Escape Test description string?	<input checked="" type="checkbox"/>
Escape exception messages?	<input checked="" type="checkbox"/>

## 22) Executando o projeto

Depois de toda a configuração basta clicarmos no link **Build Now**

A box de **Build History** começará a executar apresentando uma barra branca e azul.

Aguarde até o fim da execução.

The screenshot shows the Jenkins dashboard for the project "Exemplo TestNG Jenkins". The top navigation bar includes links for Back to Dashboard, Status, Changes, Workspace, Build Now (circled in red), Delete Project, Configure, and TestNG Results. Below the navigation is a "Build History" section with a single build entry (#1, Sep 11, 2011 12:18:37 AM) which is currently executing, indicated by a progress bar. A red oval also surrounds this entire "Build History" section. To the right of the history are links for TestNG Results, Workspace, and Recent Changes. At the bottom, there are "Permalinks" for RSS feeds for all and failures.

Jenkins » Exemplo TestNG Jenkins

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Project

Configure

TestNG Results

Build History (trend)

#1 Sep 11, 2011 12:18:37 AM

RSS for all RSS for failures

TestNG Results

Workspace

Recent Changes

Permalinks

## 23) Visualizando a execução do projeto

Após a execução do projeto a box de **Build History** deve conter a execução com uma bola verde ou azul ao lado da data de execução.

Clique sobre a data para exibir os detalhes da execução

**Jenkins**

Jenkins » Exemplo TestNG Jenkins » #3

[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[Edit Build Information](#)

[TestNG Results](#)

[Previous Build](#)

**Build #3 (Sep 11, 2011 3:42:01 PM)**

No changes.

Started by anonymous user

[TestNG Results](#)

- Total Tests: 1 (+0)
- Failed Tests: 0 (+0)
- Skipped Tests: 0 (+0)
- Failed Configurations: 0 (+0)
- Skipped Configurations: 0 (+0)

## 24) Visualizando a execução do projeto - Console

Para visualizar o console (saída das execuções da build) clique no link **Console Output**

Se algum erro ocorrer conseguiremos visualizar o detalhe nesta seção.

Todas as *targets* executadas pela build estão listadas na box **Executed Ant Targets**

# Jenkins

[Jenkins](#) » [Exemplo TestNG Jenkins](#) » #3

[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[Edit Build Information](#)

[TestNG Results](#)

[Previous Build](#)

### Executed Ant Targets

- [compile-test](#)
- [test](#)

## Console Output

```
Started by user anonymous
[SeleniumJenkinsTestNG] $ cmd.exe /C ""C:\Program Files (x86)\Jenkins\tools\default\bin\ant.bat -f C:\Users\Elias\workspace\SeleniumJenkinsTestNG\build.xml

compile-test:

test:
    [delete] Deleting directory C:\Users\Elias\workspace\SeleniumJenkinsTestNG\reports
    [mkdir] Created dir: C:\Users\Elias\workspace\SeleniumJenkinsTestNG\reports
    [testng] [TestNG] Running:
    [testng]   C:\Users\Elias\workspace\SeleniumJenkinsTestNG\testng.xml
    [testng]
    [testng]
    [testng]
    [testng] =====
    [testng] Suite1
    [testng] Total tests run: 1, Failures: 0, Skips: 0
    [testng] =====
    [testng]

BUILD SUCCESSFUL
Total time: 8 seconds
Looking for TestNG results report in workspace using pattern: reports/testng-results.xml
Finished: SUCCESS
```

## 25) Visualizando a execução do projeto – Resultados de Teste

Para visualizar o resultado de teste com o relatório do JUnit agregado clicamos no link **TestNG Results**  
Ele mostrará a execução de todos os testes (classes) em todos os pacotes executados

**Jenkins**

Jenkins » Exemplo TestNG Jenkins » #3 » [TestNG Results](#)

[Back to Project](#)  
 [Status](#)  
 [Changes](#)  
 [Console Output](#)  
 [Edit Build Information](#)  
 [\*\*TestNG Results\*\*](#)  
 [Previous Build](#)



**TestNG Results**

0 failures ( $\pm 0$ )

**Failed Tests**  
No Test method failed

**Failed Configuration Methods**  
No Configuration method failed

**Skipped Tests**  
No Test method was skipped

**Skipped Configuration Methods**  
No Configuration method was skipped

**All Tests (grouped by their packages)**

[hide/expand the table](#)

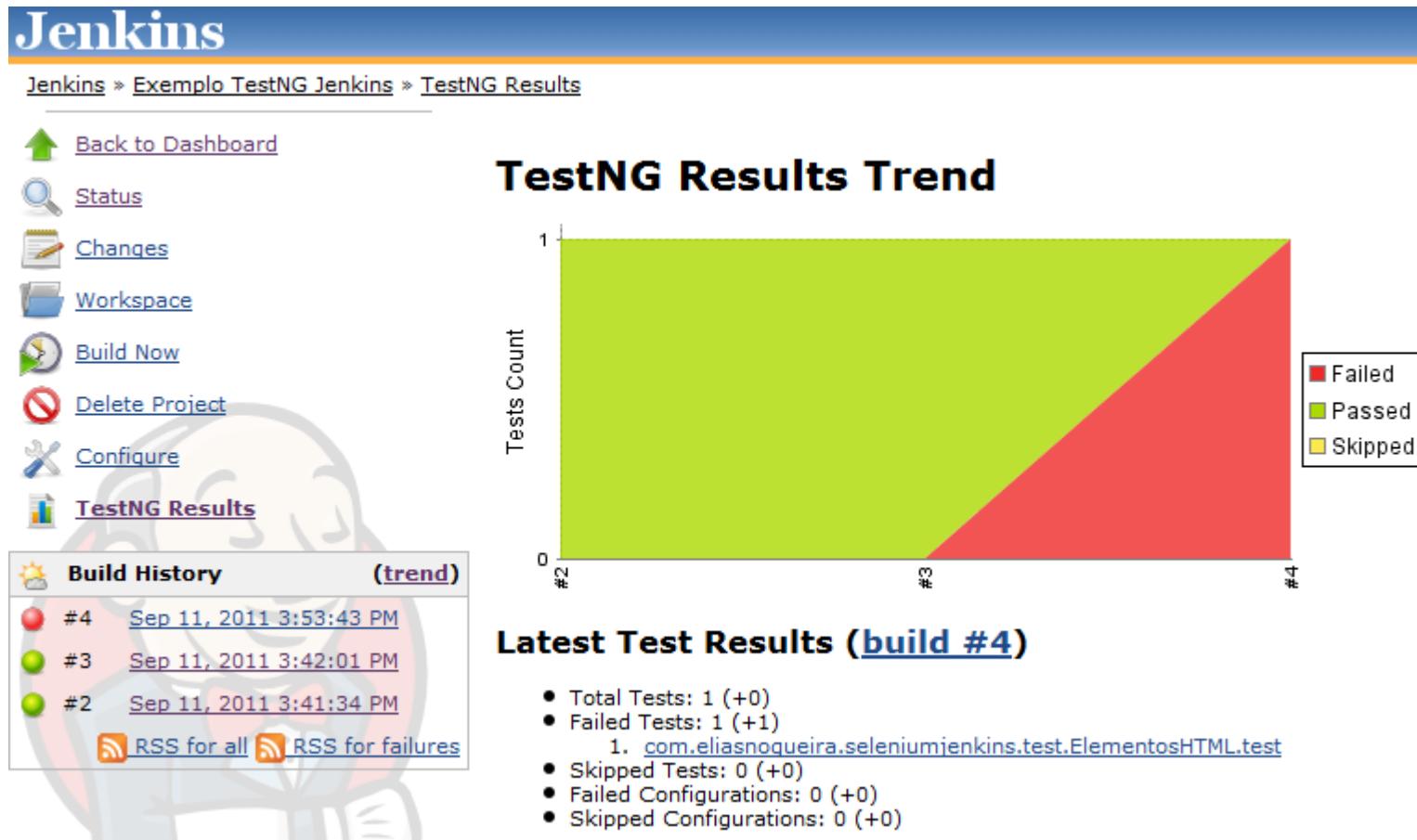
Package	Duration
com.eliasnoqueira.seleniumjenkins.test	2 sec 949 msec

## 26) Visualizando a execução do projeto – Histórico do Teste

Dentro do resultado da build é possível visualizar o histórico de execução.

Para acessar o histórico clicamos no link **TestNG Results** pelo dashboard (pagina inicial) do projeto

São exibidos os dados de cada build e um gráfico de tempo de execução ou de contagem de execuções



# Referências

- **Página do SeleniumHQ:** <http://seleniumhq.org/>
- **Download do Selenium (IDE, RC e Plugins):** <http://seleniumhq.org/download/>
- **Página do WebDriver:** <http://code.google.com/p/selenium/>
- **Documentação Selenium:** <http://seleniumhq.org/docs/>
- **Referência comandos Selenium IDE:**  
<http://release.seleniumhq.org/selenium-core/1.0/reference.html>
- **Referência comandos/API WebDriver (Java):**  
<http://selenium.googlecode.com/svn/trunk/docs/api/java/index.html>
- **Selenium Wiki:** <http://code.google.com/p/selenium/w/list>
- **Site JUnit:** <http://www.junit.org/>
- **ANT:** <http://ant.apache.org/>
- **Junit + ANT:** <http://ant.apache.org/manual/Tasks/junit.html>
- **Site TestNG:** <http://testng.org/doc/index.html>
- **TestNG Eclipse Plugin:** <http://testng.org/doc/eclipse.html>
- **TestNG + ANT:** <http://testng.org/doc/ant.html>
- **Site Jenkins CI:** <https://jenkins-ci.org/>