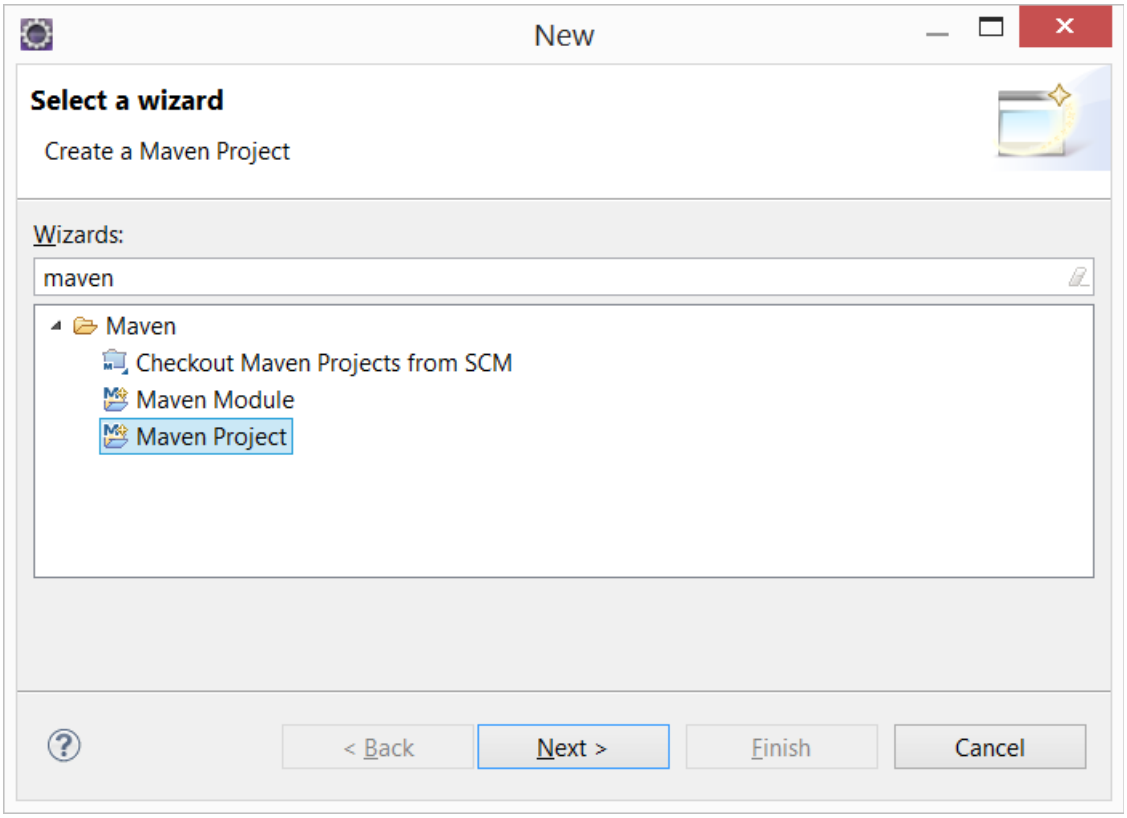
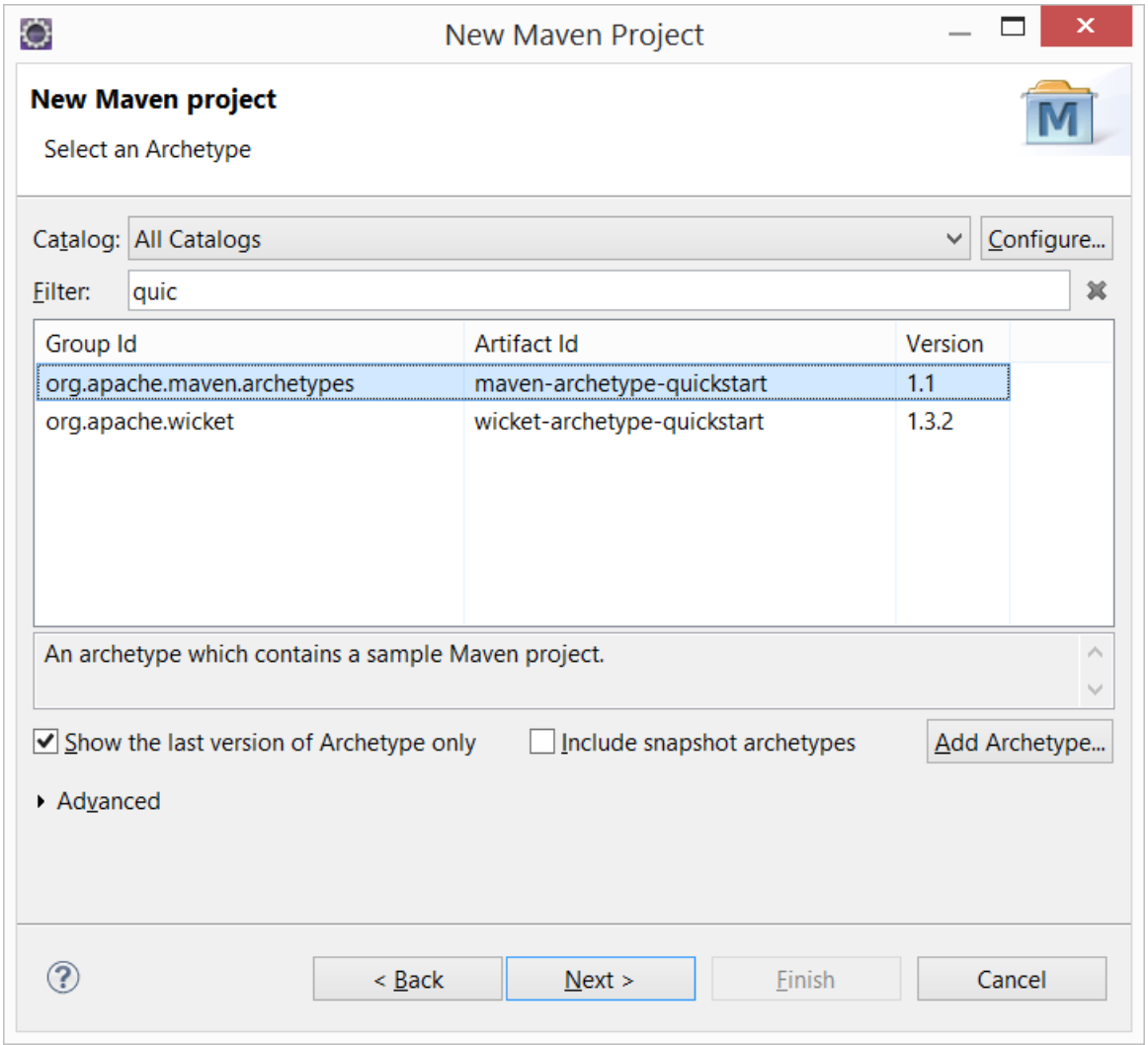
Criando um projeto Simples com Selenium

Crie um projeto Maven Project Simples:

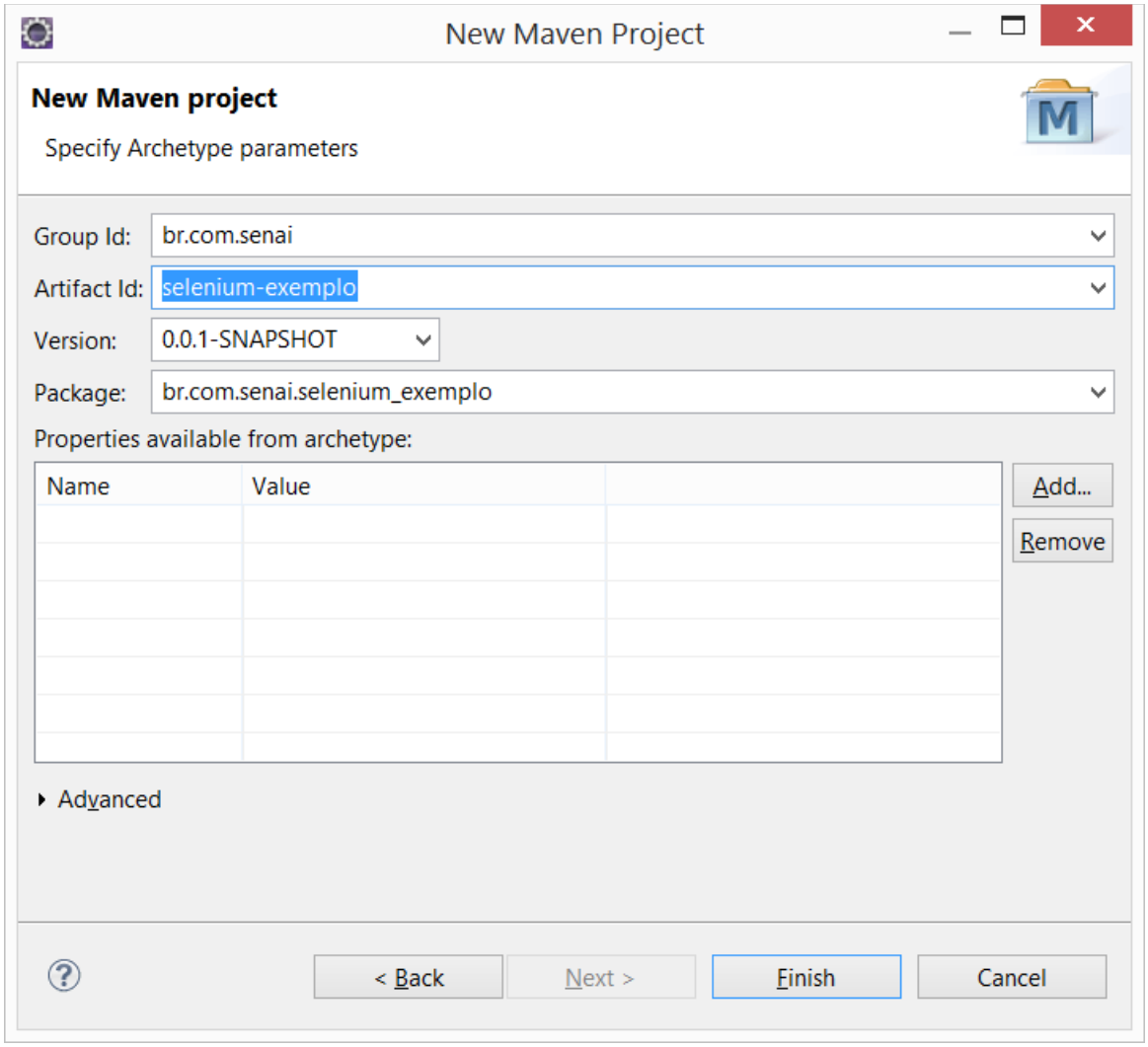


Escolha a opção Quick Start:

Na



próxima tela preencha os campos conforme figura abaixo:



Com o projeto criado, crie a classe HelloSeleniumTeste:

```
package br.com.senai.selenium_exemplo;

import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;

public class HelloSeleniumTest {

	@Before
	public void setUp() throws Exception {

	}

	@Test
	public void testSearchInGooglePage() {

		WebDriver driver  = new FirefoxDriver();

//		Digo qual url para acessar
```

```
                driver.get("http://www.google.com");

//              Agora vamos buscar o elemento na página

                WebElement inputTextGoogle = driver.findElement(By.name("q"));

                inputTextGoogle.sendKeys("Edjalma Queiroz");

/*              faz um submit na página

 *              poderia buscar o botão search e fazer o submit tb.

 */

                inputTextGoogle.submit();


assertTrue(driver.getPageSource().contains(driver.findElement(By.id("gbqfq")).g
etText()));

        }

}
```

Adicione a dependencia ao Selenium-Server-StandAlone 2.44.0 ao pom do seu projeto:

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-server</artifactId>

<version>2.44.0</version>

</dependency>

Altere a versão do Junit para a 4.11

```
    <dependency>

            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
    </dependency>
```

Após salvar o arquivo pom.xml, o eclipse realizará o download das depenências que essa biblioteca necessita.

Entendendo o código

Apesar de que em alguns trechos eu coloquei comentários somente para facilitar o entendimento, vou explicar alguns pontos que considero importantes.

WebDriver: é uma interface do Selenium que todo Web Browser Drivers implementa. O Firefox Browser tem sua implementação, assim como IE e Chrome, cada um com sua particularidade, e é preciso dar uma olhada na documentação sobre como implementar.

Depois que instanciamos o *driver,* dizemos a URL que queremos testar (nesse caso será do Google), mas em um projeto JEE, por exemplo, vamos colocar o caminho onde está nossa aplicação.

Em seguida pesquisamos pelos elementos na página, para isso no Chrome podemos usar o atalho F12, clica na lupa que fica no rodapé e clica sobre o *input text* e ver qual o nome daquele campo. Podemos usar o *id, nome* etc. Veja:

Depois que fizemos isso, criamos uma variável para representar esse campo :

```
WebElement inputTextGoogle = driver.findElement(By.name("q"));
```

E em seguida invocamos o método *sendKeys(…)* e passamos o valor que queremos que seja digitado no*input*. Para descobrir e conhecer melhor os métodos disponíveis tem que passar por alguns minutos vendo o que temos na documentação do framework.

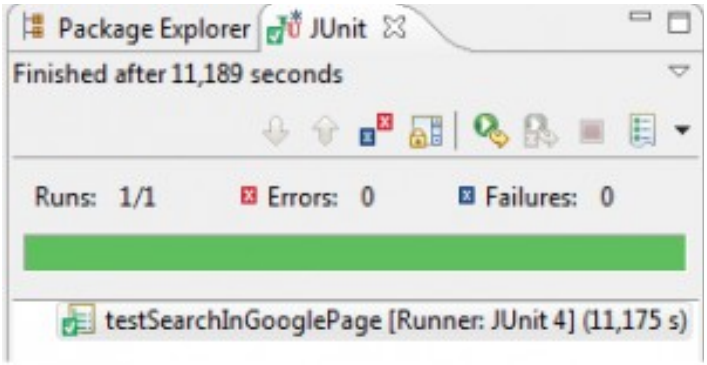
```
inputTextGoogle.sendKeys("Edjalma Queiroz");
```

Logo em seguida fizemos o *submit* página

```
inputTextGoogle.submit();
```

Criando o assert

Veja que no teste fizemos um assert simples, que apenas verifica se o conteúdo retornado pelo navegador é o mesmo pesquisado, Note que em seu sistema isso pode ser uma Regra de Negócio:



O teste passa. Na verdade esse teste não tem nada de inteligente. Se você reparar, ele verifica se o input que pesquisamos na primeira página do Google é o mesmo na página do resultado da busca.

Claro que em nossa aplicação íamos testar algo mais voltado para regras de negócio. E o método*getPageSource()* nos ajuda nisso, em busca de um elemento na página corrente.

Execute o teste, como um Junit Test.