02

Carregando o formulário de cadastro de leilão

Transcrição

Agora que já aprendemos a utilizar o padrão Page Object para organizarmos o código, podemos continuar a escrita dos testes Selenium seguindo esse novo padrão. Nossa próxima atividade será testar o cadastro de um leilão. Depois que fizermos o login e entrarmos na tela de listagem de leilões, teremos um link que navega para o endereço http://localhost:8080/leiloes/new (http://localhost:8080/leiloes/new), referente ao formulário de cadastro de um novo leilão.

Nessa página é necessário preencher três informações: o nome do leilão, o valor do lance inicial e a data de abertura. Para uma simulação, usaremos os valores "teste", "100" e "15/11/2020". Temos também um campo "Usuário", mas este puxa o usuário logado no sistema automaticamente. Ao clicarmos em "Salvar", enviamos o formuário e somos redirecionados para a tela de listagem, com nosso leilão cadastrado sendo exibido como o último registro. Essa é a funcionalidade que iremos testar.

Como ela possui diversos detalhes, quebraremos esse processo em dois vídeos de modo a torná-lo menos trabalhoso. O primeiro passo será criarmos uma nova classe de teste, `LeiloesTest`, em um pacote `br.com.alura.leilao.leiloes`, onde armazenaremos todos os testes relacionados com o leilão, como listagem, cadastro ou exclusão.

Para facilitar nosso trabalho, podemos mover a classe `LoginTest` para o novo pacote, renomea-la para `LeiloesTest`, remover todos os métodos (já que eles são relacionados à funcionalidade de login) e substituir alguns parâmetros de modo a adequá-los ao novo cenário:

- `LoginPage` se tornará `LeiloesPage`
- `paginaDeLogin` se tornará `paginaDeLeiloes`

```
public class LeiloesTest {

    private LeiloesPage paginaDeLeiloes;

    @BeforeEach
    public void beforeEach() {
        this.paginaDeLeiloes = new LeiloesPage();
    }

    @AfterEach
    public void afterEach(){
        this.paginaDeLeiloes.fechar();
    }

}
```

COPIAR CÓDIGO

Teremos um erro de compilação, já que a classe `LeiloesPage` não existe. Como esse será um Page Object parecido com `LoginPage`, vamos copiá-lo e adaptá-lo. Substituiremos `URL_LOGIN` por `URL_LEILOES`, adaptando o endereço para http://localhost:8080/leiloes (http://localhost:8080/leiloes), e removeremos todos os métodos relacionados à funcionalidade de login, restando apenas o construtor da classe e o `fechar()`, que fecha o navegador após o teste ser finalizado.

```
public class LeiloesPage {

    private static final String URL_LEILOES = "http://localhost:8
    private WebDriver browser;

    public LeiloesPage() {

        System.setProperty("webdriver.chrome.driver", "/drivers/c
        this.browser = new ChromeDriver();
        browser.navigate().to(URL_LEILOES);
    }

    public void fechar() {
        this.browser.quit();
    }
}
```

COPIAR CÓDIGO

Começaremos então a escrever os novos testes. Em `LeiloesTest`, criaremos o método `deveriaCadastrarLeilao()` com a anotação `@Test` do JUnit. Aqui temos um problema. Em teoria, nosso `beforeEach()` está navegando para a listagem de leilões. Desejamos acessar a página de formulário, o que pode ser feito clicando no botão "Novo Leilão".

Entretanto, se deslogarmos do sistema esse botão não aparece, e se tentarmos acessar a URL diretamente seremos redirecionados para a página de login. Sendo assim, para efetuarmos o teste, primeiro teremos que navegar para a tela de login e preencher o formulário. Ao fazermos login, seremos redirecionados para a página de listagem, e só então poderemos navegar ao formulário. Eventualmente você desenvolverá testes que precisam navegar por múltiplas páginas, o que é bem comum.

Em `LeiloesTest`, removeremos o método `beforeEach()`, pois não queremos acessar diretamente a página de leilões. No teste `deveriaCadastrarLeilao()` teremos todo o fluxo de execução do nosso cenário. Portanto, começaremos criando uma variável `paginaDeLogin` do tipo `LoginPage`.

Eventualmente você encontrará testes que instanciam múltiplos Page Objects, algo completamente normal. Depois de instanciarmos nossa `paginaDeLogin`, chamaremos o método `preencheFormularioDeLogin()` com os parâmetros `fulano` e `pass` representando usuário e senha, e completaremos o login com `efetuarLogin()`.

```
@Test
public void deveriaCadastrarLeilao() {
    LoginPage paginaDeLogin = new LoginPage();
    paginaDeLogin.preencheFormularioDeLogin("fulano", "pass");
    paginaDeLogin.efetuaLogin();
}
```

COPIAR CÓDIGO

Como o método `efetuarLogin()` navega para a listagem de leilões, ou seja, `LeiloesPage`, ele poderia nos devolver um objeto desse tipo. Sendo assim, atribuiremos o retorno do método a `this.paginaDeLeiloes`, que não tínhamos instanciado ainda. Teremos um erro de compilação, já que o retorno de `efetuarLogin()` no momento é um `void`. Substituiremos `void` por um objeto `LeiloesPage` e finalizaremos o método retornando esse objeto.

```
@Test
public void deveriaCadastrarLeilao() {
    LoginPage paginaDeLogin = new LoginPage();
    paginaDeLogin.preencheFormularioDeLogin("fulano", "pass");
```

```
        this.paginaDeLeiloes = paginaDeLogin.efetuaLogin();
    }
```

COPIAR CÓDIGO

```
    public LeiloesPage efetuaLogin() {
        browser.findElement(By.id("login-form")).submit();
        return new LeiloesPage();

    }
```

COPIAR CÓDIGO

Quando instanciamos um Page Object do tipo `LoginPage`, ele seta a variável de ambiente do driver e abre a janela do navegador. No caso de `LeiloesPage`, queremos continuar na mesma janela. Sendo assim, ao instanciarmos o `LeiloesPage`, passaremos o próprio `browser` que está sendo utilizado no momento. Nos testes com Selenium utilizando Page Objects é comum compartilhar o webdriver para continuar na mesma página.

```
    public LeiloesPage efetuaLogin() {
        browser.findElement(By.id("login-form")).submit();
        return new LeiloesPage(browser);

    }
```

COPIAR CÓDIGO

Teremos novamente um erro de compilação, pois o construtor da nossa classe atualmente é vazio. Corrigiremos isso recebendo o webdriver como parâmetro. Além disso, deixaremos de instanciar esse driver no construtor, pois ele já foi setado, e simplesmente atribuiremos o `browser` passado como parâmetro. Como a navegação já foi feita pelo `LoginPage` quando fizemos o envio do formulário, também não precisaremos navegar para a página de leilões.

```
public LeiloesPage(WebDriver browser) {

    System.setProperty("webdriver.chrome.driver", "/drivers/chror
    this.browser = browser;
}
```

COPIAR CÓDIGO

Feito isso, precisaremos criar um método que clica no botão "Novo Leilão" para acessarmos o formulário de cadastro. Esse botão na verdade é um link que navega para http://localhost:8080/leiloes/new (http://localhost:8080/leiloes/new). A partir de `paginaDeLeiloes`, criaremos o método `carregarFormulario()` que faz essa navegação a partir de `this.browser.navigate().to()`.

Temos em nosso Page Object uma variável `URL_LEILOES` que navega para a listagem de leilões, e que não mais utilizaremos. Vamos substituí-la por `URL_CADASTRO_LEILAO`, uma variável apontando para http://localhost:8080/leiloes/new (http://localhost:8080/leiloes/new), e a passaremos como parâmetro da nossa navegação.

```
public class LeiloesPage {

    private static final String URL_CADASTRO_LEILAO = "http://loc

//...código omitido

    public void carregarFormulario() {
        this.browser.navigate().to(URL_CADASTRO_LEILAO);
    }
}
```

COPIAR CÓDIGO

Nosso método `carregarFormulario()` retornará um objeto `CadastroLeilaoPage`, também recebendo o `browser` (já que queremos continuar na mesma janela do navegador). Esse objeto será um novo Page Object referente ao formulário de cadastro de um novo leilão.

```
public CadastroLeilaoPage carregarFormulario() {
        this.browser.navigate().to(URL_CADASTRO_LEILAO);
        return new CadastroLeilaoPage(browser);
}
```

COPIAR CÓDIGO

Criaremos essa nova classe no pacote `br.com.alura.leilao.leiloes`. Nela teremos, além do método `CadastroLeilaoPage()`, que atribui ao nosso atributo `Webdriver browser` o `browser` recebido por parâmetro, o método `fechar()` responsável por fechar o navegador.

```
public class CadastroLeilaoPage {

    private WebDriver browser;

    public CadastroLeilaoPage(WebDriver browser) {
        this.browser = browser;
    }

    private void fechar() {
        this.browser.quit();
    }
}
```

COPIAR CÓDIGO

Em `LeilaoTest`, o método `deveriaCadastrarLeilao()` navegará para o formulá de cadastro de um leilão, nos devolvendo um objeto `CadastroLeilaoPage` que

armazenaremos em uma variável `paginaDeCadastro`.

```
@Test
public void deveriaCadastrarLeilao() {
    LoginPage paginaDeLogin = new LoginPage();
    paginaDeLogin.preencheFormularioDeLogin("fulano", "pass");
    this.paginaDeLeiloes = paginaDeLogin.efetuaLogin();
    CadastroLeilaoPage paginaDeCadastro = paginaDeLeiloes.carrega
}
```

COPIAR CÓDIGO

Nessa aula o objetivo era fazermos o passo-a-passo para carregarmos o formulário. Sendo assim, executaremos nosso teste. O navegador será aberto e conseguiremos logar no sistema corretamente, sendo redirecionados para o cadastro de novo leilão. Ou seja, nosso teste passará corretamente, ainda que não tenha nenhuma assertiva.

Perceba que esse teste foi um pouco mais trabalhoso, já que para chegarmos na página de formulário precisamos passar por todo o processo de logar e navegar a partir da listagem de leilões. Esse tipo de código, com Page Objects que navegam e devolvem outros Page Objects, é bastante comum. O ato de um método criar um novo Page Object e compartilhar o webdriver, de modo a evitar a abertura de uma nova janela em um único cenário de teste, também é comum.

No próximo vídeo prosseguiremos preenchendo o formulário e verificando se tudo funcionou conforme o esperado.