07

Organizando o código de teste

Transcrição

Agora que finalizamos os testes do cadastro de leilão, tanto quando ele é bem sucedido como quando ele falha devido à validação, conversaremos mais um pouco sobre organização de código. Além do Page Object, existe também outra boa prática no desenvolvimento de testes E2E, que é tentar organizar os códigos dos Page Objects e de quaisquer outras classes de teste.

A ideia é mantermos o código simples, facilitando a leitura e a manutenção. Nesse sentido, temos um problema em nosso código, algo que talvez você já tenha percebido: todos os nossos Page Objects possuem códigos em comum.

Atualmente, todas essas classes precisam do webdriver para fazer a navegação e encontrar os elementos, além de possuírem um método que fecha o browser e assim por diante. São códigos em comum que estão repetidos, e que portanto podem ser eliminados de alguma forma.

Em uma linguagem orientada a objetos, podemos utilizar o recurso da herança, criando uma classe base (também chamada de "classe mãe" ou "classe pai") que serviria como base para todos os Page Objects.

No pacote `br.com.alura.leilao`, criaremos uma classe `PageObject` que representará, como o próprio nome diz, os Page Objects do nosso projeto. Sendo assim, todas as classes desse tipo herdarão da nova classe, e tudo que for comum entre elas será herdado de `PageObject`.

```
package br.com.alura.leilao;

public class PageObject {

}
```

COPIAR CÓDIGO

Começaremos pelo atributo `webdriver`, que é comum a todos os nossos Page Objects. Sendo assim, vamos movê-lo para a nova classe. Para conseguirmos acessá-lo a partir das classes filhas, precisaremos alterar o atributo de `private` para `protected`.

```
import org.openqa.selenium.WebDriver;

public class PageObject {

    protected WebDriver browser;

}
```

COPIAR CÓDIGO

Removendo a criação desse atributo da classe `CadastroLeilaoPage`, teremos um erro de compilação. Vamos resolvê-lo usando o `extends`, fazendo com que a classe `CadastroLeilaoPage` herde de `PageObject`.

```
public class CadastroLeilaoPage extends PageObject {

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

    public CadastroLeilaoPage(WebDriver browser) {
        browser = browser;
```

```
    }
//...
```

COPIAR CÓDIGO

No `LoginPage`, estamos instanciando o `ChromeDriver`. Removeremos essas instruções dessa página, passando-as para um construtor da classe `PageObject`, que será responsável por setar a variável de ambiente do driver do Google Chrome e instanciá-lo.

```
public class PageObject {

    protected WebDriver browser;

    public PageObject() {
        System.setProperty("webdriver.chrome.driver", "/drivers/c
        this.browser = new ChromeDriver();
    }
}
```

COPIAR CÓDIGO

Entretanto, se fizermos isso, os Page Objects sempre abrirão uma nova janela do navegador, sendo que nem todo cenário precisa disso. Portanto, receberemos o `browser` no construtor e o armazenaremos. Se o parâmetro do construtor vier preenchido, atribuiremos o `browser`; do contrário, criaremos uma nova instância de `ChromeBrowser`. Conseguiremos atender a essas duas situações usando uma condicional `if/else`.

```
public class PageObject {

    protected WebDriver browser;
```

```
    public PageObject(WebDriver browser) {
        System.setProperty("webdriver.chrome.driver", "/drivers/c
        if (browser == null) {
            this.browser = new ChromeDriver();
        } else {
            this.browser = browser;
        }
    }
}
```

COPIAR CÓDIGO

Outro código comum entre os Page Objects é o método `fechar()`, que fecha a janela do navegador. Também vamos movê-lo para a classe `Page Object`.

```
public class PageObject {

    protected WebDriver browser;

    public PageObject(WebDriver browser) {
        System.setProperty("webdriver.chrome.driver", "/drivers/c
        if (browser == null) {
            this.browser = new ChromeDriver();
        } else {
            this.browser = browser;
        }
    }

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

COPIAR CÓDIGO

Agora precisamos herdar da classe `PageObject` em todas as nossas páginas, além de remover os códigos repetidos. Em `LoginPage`, incluiremos a instrução `extends Page Object` e, no construtor da própria classe, chamaremos um `super()` com o parâmetro `null` para chamarmos o construtor da classe mãe. Nesse caso, estamos passando um parâmetro nulo já que, como `LoginPage` é a primeira página, desejamos que o construtor crie uma nova instância do `ChromeDriver`, abrindo uma nova janela do navegador.

```
public class LoginPage extends PageObject {

    private static final String URL_LOGIN = "http://localhost:808

    public LoginPage() {
        super(null);
        browser.navigate().to(URL_LOGIN);
    }
```

COPIAR CÓDIGO

Também faremos esse processo em `LancesPage`, removendo as linhas responsáveis por instanciar o `ChromeDriver` e o método `fechar()`, e herdando da classe `PageObject.

```
public class LancesPage extends PageObject {
    private  static final String URL_LANCES = "http://localhost:8

    public LancesPage () {
        super(null);
        browser.navigate().to(URL_LANCES);
    }
```

COPIAR CÓDIGO

Em `LeiloesPage`, adicionaremos a instrução `extends PageObect`, excluiremos a variável `WebDriver`, excluiremos o métod `fechar()` e, no construtor da classe, chamaremos o `super()` recebendo o `browser` como parâmetro.

```
public class LeiloesPage extends PageObject {

    private static final String URL_CADASTRO_LEILAO = "http://loc
    private static final String URL_LEILOES = "http://localhost:8

    public LeiloesPage(WebDriver browser) {
        super(browser);
    }
```

COPIAR CÓDIGO

Em `CadastroLeilaoPage`, também usaremos o `super()` no construtor e excluiremos o método `fechar()`.

```
public class CadastroLeilaoPage extends PageObject {

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

    public CadastroLeilaoPage(WebDriver browser) {
        super(browser);
    }
```

COPIAR CÓDIGO

Feito isso, já temos o início de uma organização de código, começamos a ter um reaproveitamento: todos os Page Objects vão herdar da classe `PageObect`, e, no

futuro, tudo que for comum a esse tipo de objeto será criado nesta classe. Agora conseguimos refatorar nosso código de modo a facilitarmos a leitura e a manutenção.

Devemos sempre verificar se tudo continua funcionando como esperado na aplicação. A princípio não temos nenhum erro de compilação, mas como fizemos mudanças no projeto, é interessante rodarmos os testes para termos certeza. Após nossos testes passaram, teremos uma única janela aberta, pois no código do `HelloWorldSelenium` nós não fechamos o navegador.

Como essa classe não possui um Page Object, podemos simplesmente adicionar uma nova linha `browser.quit()` para fecharmos a janela.

```
public class HelloWorldSelenium {

    @Test
    public void hello() {
        System.setProperty("webdriver.chrome.driver", "/drivers/c
        WebDriver browser = new ChromeDriver();
        browser.navigate().to("http://localhost:8080/leiloes");
        browser.quit();
    }
}
```

COPIAR CÓDIGO

Você também poderia buscar o reaproveitamento de código nas classes de teste, nas quais utilizamos o JUnit, verificando se elas possuem instruções em comum e, em caso positivo, criando uma classe base para os testes que seria herdada nas demais.