Page Object Model

Selenium

Manual

O que é o “Page Object Model”?

É um padrão de projeto designado para realização de testes automatizados que ficou popular devido enfatizar mantenabilidade e redução de duplicação de código.

Benefícios de sua implantação:

- Separação de responsabilidades entre código a ser testado e especificidades da página em si - como seus locadores e layout.
- Organização do conteúdo a ser testado ou operado com escopo e localização bem delimitados.

Devido ao dinamismo da web, se fez necessário repensar a forma de como são construídas as soluções ao seu entorno. E no que tange testabilidade de navegação, surgiu o temido P.O.M...

Declaração de classe

```
1 reference
public class LoginPage
{
    4 references
    protected static IWebDriver _driver;
    1 reference
    private By _usernameBy = By.CssSelector("user_name");
    1 reference
    private By _passwordBy = By.Id("password");
    1 reference
    private By _signinBy = By.XPath("//sign_in");

    0 references
    public LoginPage(IWebDriver driver) ⇒ _driver = driver;
    0 references
    public LoginPage EnterCredentials(string userName, string password)
    {
        _driver.FindElement(_usernameBy).SendKeys(userName);
        _driver.FindElement(_passwordBy).SendKeys(password);
        _driver.FindElement(_signinBy).Click();

        return this;
    }
}
```

Asserções importantes:

- ***driver*** como membro estático e recebido via construtor da classe.
- ***seletores*** delimitados por propriedades privadas, evitando que seu estado seja mudado por agentes externos.
- criação de métodos para interagirem com os seletores, assim expondo apenas comportamentos que se espera da página.
- Tudo isso junto garante o encapsulamento e o princípio da responsabilidade única.
- ...e outros pontos serem destacados logo mais.

Além do mais, o padrão nos permite:

- Trabalhar e desenvolver ***bibliotecas de classes*** como provedores de serviços/funcionalidades independentes.
- Viabiliza a utilização/implementação em conjunto com outros padrões de desenvolvimento e técnicas de programação como a ***Injeção de Dependência*** – que será abordada em outra parte.
- Escrever código de forma consistente e contínua.
- Possui uma ótima estática e a leitura do código se torna agradável.

Na teoria pode parecer simples é fácil, mas como fica tudo isso na prática? ***Para isso vamos simular o cenário onde supostamente precisamos consultar números de CEP no site dos Correios para preencher um formulário do Detran.***

Roadmap de implementação

Descrição

1. ***Step***: criação de um projeto de biblioteca de classes
2. ***Step***: Definição das páginas a serem navegadas, criando assim uma hierarquia de pastas e arquivos de acordo o tipo de navegação.
3. ***Step***: Elaborar métodos que exponham de forma clara e objetiva a ação a ser executada na página.
4. ***Step***: Criação de uma classe para centralizar o fluxo da navegação - entry point da navegação em si -, onde serão chamados os métodos descritos nas classe “Page”.
5. ***Step***: Garantir que haja um controle de fluxo para que a transição entre as páginas sejam precisas e fluidas.

Roadmap de implementação

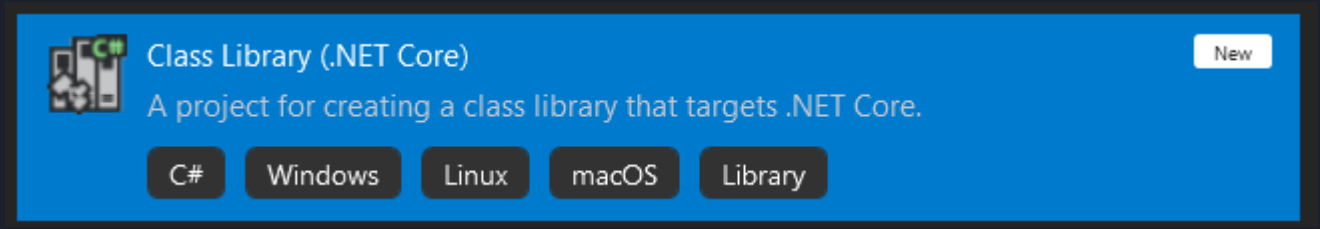
Ilustração #01

- ***Step***: criação de um projeto de biblioteca de classes

Comando via CLI:

```
xstrato in Examples on  master
⇒ dotnet new classlib -o POMDIP.Example.Navigation
```

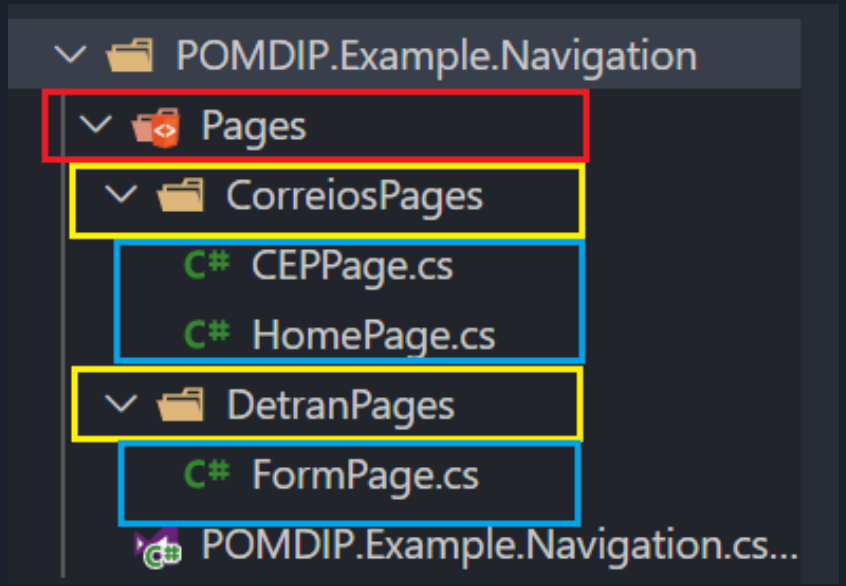
Através do Visual Studio:



Roadmap de implementação

Ilustração #02

➢ ***Step***: Definição das páginas a serem navegadas, criando assim uma hierarquia de pastas e arquivos de acordo o tipo de navegação.



CONVENÇÃO (Hierarquia):

Pages -> diretório mãe para todos os site e páginas contidos na navegação.

CorreiosPages / DetranPages -> diretórios onde ficaram para as páginas referente as navegações dos respectivos sites.

CEPPage / HomePage / FormPage -> as páginas propriamente dita contendo os seletores e métodos de manipulação da página.

Roadmap de implementação

Ilustração #03.1

➢ ***Step***: Elaborar métodos que exponham de forma clara e objetiva a ação a ser executada na página.

```
3 references
public class HomePage
{
    4 references
    protected static IWebDriver _driver;
    1 reference
    private By _searchTextBox = By.Id("acesso-busca");
    1 reference
    private By _searchButton = By.CssSelector("button.bt-link-ic i.ic-busca-out");
    0 references
    public HomePage(IWebDriver driver) => _driver = driver;
    1 reference
    public HomePage SearchByCEP(string CEP)
    {
        _driver.Url = "http://www.correios.com.br";
        _driver.FindElement(_searchTextBox).SendKeys(CEP);
        _driver.FindElements(_searchButton)[^1].Click();

        return this;
    }
}
```

Roadmap de implementação

Ilustração #03.2

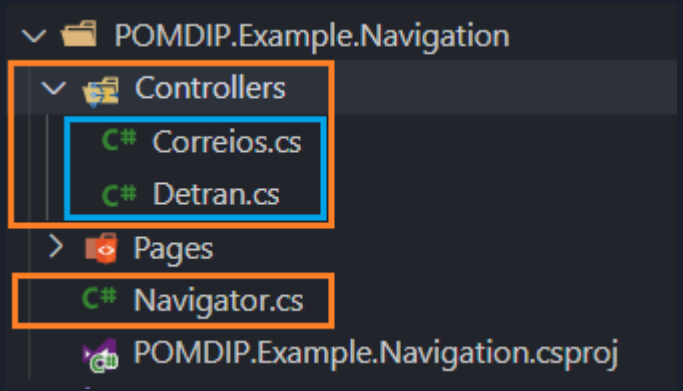
```
public class CEPPage
{
    6 references
    protected static IWebDriver _driver;
    1 reference
    private By _tableResults = By.CssSelector("table#resultado-DNEC td");
    1 reference
    private By _logradouro = By.CssSelector("td[data-th='Bairro/Distrito']");
    1 reference
    private By _bairro = By.CssSelector("td[data-th='Logradouro/Nome']");
    1 reference
    private By _localidade = By.CssSelector("td[data-th='Localidade/UF']");
    1 reference
    private By _cep = By.CssSelector("td[data-th='CEP']");
    0 references
    public CEPPage(IWebDriver driver) => _driver = driver;
    1 reference
    public bool HasResults() => _driver.FindElements(_tableResults).Any();
    1 reference
    public string[] GetCEPData()
    {
        var logradouro = _driver.FindElement(_logradouro).Text;
        var bairro = _driver.FindElement(_bairro).Text;
        var localidade = _driver.FindElement(_localidade).Text;
        var cep = _driver.FindElement(_cep).Text;

        return new[]{logradouro,bairro,localidade,cep};
    }
}
```

```
public class FormPage
{
    8 references
    protected static IWebDriver _driver;
    1 reference
    private By _logradouro = By.Id("proprietarioLogradouroTextBox");
    1 reference
    private By _bairro = By.Id("proprietarioBairroTextBox");
    1 reference
    private By _localidadeBox = By.Id("proprietarioVeiculoLabel");
    1 reference
    private By _localidadeField = By.Id("pesquisaTextBox");
    1 reference
    private By _localidadeButton = By.Id("selecionarButton");
    1 reference
    private By _cep = By.Id("proprietarioCepTextBox");
    0 references
    public FormPage(IWebDriver driver) => _driver = driver;
    1 reference
    public FormPage SetCorreiosData(string[] correiosData)
    {
        _driver.Url = "http://formulario.detran.sp.gov.br/Planilha.aspx";
        _driver.FindElement(_logradouro).SendKeys(correiosData[0]);
        _driver.FindElement(_bairro).SendKeys(correiosData[1]);
        _driver.FindElement(_cep).SendKeys(correiosData[3]);

        _driver.FindElement(_localidadeBox).Click();
        _driver.FindElement(_localidadeField).SendKeys(correiosData[2]);
        _driver.FindElement(_localidadeButton).Click();

        return this;
    }
}
```

Roadmap de implementação

Ilustração #04.1

- ***Step***: Criação de uma classe para centralizar o fluxo da navegação - entry point da navegação em si -, onde serão chamados os métodos descritos nas classe “Page”.




```
2 references
public class Correios
{
    2 references
    private HomePage _homePage { get; init; }
    3 references
    private CEPPage _cepPage { get; init; }
    0 references
    public Correios(HomePage homePage, CEPPage cepPage)
    {
        _homePage = homePage;
        _cepPage = cepPage;
    }
    1 reference
    public string[] Start(string cep)
    {
        _homePage.SearchByCEP(cep);

        if(_cepPage.HasResults())
            return _cepPage.GetCEPData();

        return default;
    }
}
```

Roadmap de implementação

Ilustração #05

- ***Step***: Garantir que haja um controle de fluxo para que a transição entre as páginas sejam precisas e fluidas.

```
2 references
public class Detran
{
    2 references
    private FormPage _formPage { get; init; }
    0 references
    public Detran(FormPage formPage) ⇒ _formPage = formPage;
    1 reference
    public void Start(string[] correiosData)
    {
        _formPage.SetCorreiosData(correiosData);
    }
}
```

```
public class Navigator
{
    2 references
    private Correios _correios { get; init; }
    2 references
    private Detran _detran { get; init; }
    0 references
    public Navigator(Correios correios, Detran detran)
        ⇒ (_correios, _detran) = (correios, detran);
    0 references
    public Task<bool> Start(string[] ceps)
    {
        try
        {
            foreach (var cep in ceps)
            {
                var data = _correios.Start(cep);

                if(data is not null)
                    _detran.Start(data);
            }
        }
        catch (System.Exception)
        {
            return Task.FromResult<bool>(false);
        }
        return Task.FromResult<bool>(true);
    }
}
```

Conclusão

O que podemos concluir até aqui?

- Desenvolvimento intuitivo e delimitado pelos princípios do padrão que em teoria o fluxo pode ser feito do início ao fim e tornando o resto do processo **incremental**.
- Temos clareza no que cada parte do código é responsável por executar.
- Flexibilidade e testabilidade.
- Escalabilidade.
- Isolar o que é otimização do que escrita do código fonte em si.

Código Fonte: https://github.com/xStrato/POMDIP.Example.git