CONGRESSO CATARINENSE DE CIÊNCIA DA COMPUTAÇÃO
2019. Ano 6, Vol. 6. ISSN: 2319-0655

COLETOR DE DADOS DOS PRODUTOS EM SITES DE LOJAS DE DEPARTAMENTOS

Luis Gustavo Block Nienkotter[1]
[1]Instituto Federal Catarinense – Rio do Sul – Campus Urbano
luis.block.nienk@gmail.com

Abstract. *This article has the objective of describe the structuring, source code, creation method, and the externs programs, that were used for the creation of a framework, in which it functions is to recover products data from the most various websites of department stores, arranging the data and making them available for the user.*
Key-words: *Framework; Data; Department stores.*

Resumo. *Este artigo tem como objetivo descrever a estruturação, o código-fonte, o método de criação, e os programas externos, que foram usados para a criação de um framework, que tem como função recuperar dados dos mais diversos produtos em vários sites de lojas de departamentos, organizando esses dados e disponibilizando eles para o usuário.*
Palavras-chave: *Framework; Dados; Lojas de departamento.*

1. Introdução

Para a criação desse framework foram utilizadas algumas técnicas, padrões de projeto, diagramas, etc., que serão descritos mais afrente nesse artigo, desse modo ao escrever o código, é possível faze-lo de forma muito mais organizada. Foi utilizada a linguagem de programação Java e o paradigma de Programação Orientado a Objeto (POO) que segundo Ashwin Urdhwareshe (2016) em seu artigo sobre POO trás a seguinte definição: “Object oriented programming is an approach which is mainly focused on the way object interacts to communicate and share the information”, numa tradução livre ele quis dizer que programação orientada a objeto é uma abordagem que tem como foco o jeito que os objetos interagem, se comunicam e compartilham informações.

A ideia desse framework é de fazer uma conexão com algum site (Essa conexão é feita com *Jsoup*, que é uma biblioteca feita para Java que será explicada mais a frente), e com essa conexão é possível pegar o HTML (linguagem de marcação utilizada para a criação de sites) da pagina, e com o HTML obtido pegar os dados disponíveis dos produtos. Assim que pegar os dados, categoriza-los e disponibiliza-los.

2. Metodologia

2.1. Testes Unitários

Testes são extremamente importantes no processo de criação de um software, pelo simples fato deles rastrearem os erros no código e evitarem problemas futuros, sendo assim validando o código. Segundo Michael Olan “Testing software from the beginning and throughout the entire development cycle is an essential software engineering practice”, que em uma tradução livre ele diz que testar o software no começo da aplicação e durante todo o processo de aplicação é uma pratica essencial de engenharia de software.

Para esse framework a criação dos testes foi feita antes do começo da criação da aplicação, assim a criação do código é voltada para que os testes previamente criados sejam aceitos.

2.2. Jsoup

Jsoup é uma biblioteca de código aberto, criado e distribuído pelo MIT. A definição do Jsoup segundo o seu próprio site é “Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and JQuery-like methods”, traduzindo livremente o que o texto quer dizer é: Jsoup é uma biblioteca em Java que trabalha com HTML do mundo real, e que provem uma API para extrair e manipular dados, usando o melhor de *DOM*, *CSS*, e métodos *JQuery*.

2.3. Diagrama de Classe

Diagramas de classe são muito importantes para qualquer projeto de software, pois com eles se tem uma noção geral do sistema, é uma representação gráfica da estrutura do sistema e suas relações. Na figura abaixo esta o diagrama de classe criado baseado na estruturação do framework criado.

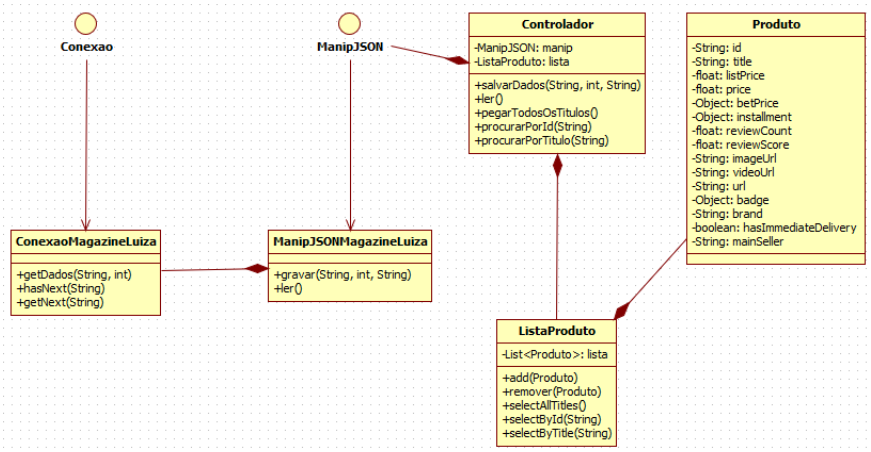

Figura 1 – Diagrama de Classe

2.4. Design Patterns

Design Patterns são mecanismos que expressam o design de estruturas de código. Designs Patterns identificam, nomeiam, e abstraem problemas que são comuns em estruturas que se utilizam da programação orientada a objeto (E. GAMMA, R. HELM, R. JOHNSON, J. VLISSIDES, 1992).

Existe diversos Design Patterns, porem nesse framework utilizaremos apenas dois deles, o de tipo criacional *Singleton* e o de tipo estrutural *Facade*.

2.4.1. Singleton

O design pattern Singleton é provavelmente o mais popular entre todos os design patterns, pelo simples fato de sua implementação ser fácil e rápida. E funciona da seguinte maneira: uma classe terá apenas uma instancia no sistema inteiro, colocando seu construtor como privado e tendo um método publico que retorna uma estancia do próprio objeto, assim é garantido que apenas um objeto seja instanciado e que não será possível fazer outras instancias em outras partes do código (K. STENCEL, P. WEGRZYNOWICZ, 2008).

A aplicação do Singleton na aplicação do framework foi feita na classe de conexão com o site, assim eu garante que na aplicação inteira eu possuo apenas uma conexão, assim não são feitas instancias desnecessárias.

2.4.2. Facade

Segundo Jan Bosch “The Facade design pattern is used to provide a single, integrated interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that simplifies the use of the subsystem.”, traduzindo esse texto livremente podemos entender que o design pattern *Facade* define uma única interface de integração para um conjunto de subsistemas, simplificando assim o seu uso.

O Facade na aplicação do framework fica por conta do Objeto que será chamado de Controlador, nele os métodos fazem conexão com toda a aplicação, fazendo com que a utilização do sistema fique muito mais fácil e pratica, pois será necessário apenas a chamada de um método.

2.5. Framework

Depois de criar os testes unitários, a tarefa passa a ser criar o código-fonte do framework, e para explicar esse processo vou começar da parte de conexão utilizando Jsoup e indo até o Controlador (que será o objeto Facade do sistema).

2.5.1. Conexão

Como existem diversos sites, e cada site repassa seus dados de forma diferente, é necessário criar uma classe para cada site, porem cada classe fará a mesma função apenas de modo diferente, por causa disso foi criado uma interface Conexão, e cada classe de conexão individual de cada site ira implementar essa interface, garantindo que tais métodos serão criados. Essas classes de conexão utilizarão do design pattern Singleton.

A classe de conexão foi pensada em pegar não só os produtos da pagina passada, mas em pegar de todas as paginas de produtos existentes no site, por conta disso foram criados três métodos, o método principal que faz a conexão com o site, pegando os dados o HTML e organizando os dados de uma maneira que fiquem em um formato Json.

```
public interface Conexao {
    public String getDados(String url, int nmrPages);

    public boolean hasNext(String url);

    public String getNext(String urlAtual);
}
```

Figura 2 – Interface Conexão

E dois métodos secundários, onde um verifica se a *url* passada possui uma próxima pagina e outro que pega a *url* da próxima pagina se existente.

2.5.2. Manipulação do Json

As classes de manipulação de Json tem a função de gravar e ler os dados em arquivos Json, essa classe recebe a *url* e com essa *url* pega os dados das funções das classes de conexão. Também utiliza de uma interface chamada ManipJSON, onde tem todos os métodos que devem ser implementados.

```
public interface ManipJSON {
    public boolean gravar(String url, int nmrPages);
    public List<Produto> ler();
}
```

Figura 3 – Interface ManipJSON

2.5.3. Classes produto

Todas as informações coletadas são salvas e abstraídas em uma classe chamada Produto que possui os seguintes parâmetros: title, listPrice, price, bestPrice, installment, reviewCount, reviewScore, imageUrl, videoUrl, url, badge, brand, hasImmediateDelivery, mainSeller. Dessa forma se torna mais fácil organizar os dados.

Além disso, possui uma classe com a lista de Produtos, essa recebe todos os produtos salvos, além de fazer buscas e verificações na própria lista.

2.5.4. Controlador

O Controlador é uma classe que será o Facade nesse framework, nesse possuem vários métodos que apenas chamam métodos do sistema, e retorna seus valores. Ele foi feito com a intenção de melhorar a experiência do usuário. A principal classe que o Controlador chama é o manipulador de Jason, pois o próprio manipulador chama os métodos da Conexão.

Para saber de qual site o Controlador deve acessar ele utiliza de injeção de dependência, onde o manipulador passado pode ser qualquer um que implemente a interface ManipJSON.

```
public class Controlador {

    private ManipJSON manip;
    private ListaProduto lista = new ListaProduto();

    /**
     *
     * @param manip
     */
    public Controlador(ManipJSON manip) {
        this.manip = manip;
    }

    /**
     *
     * @param url
     * @return true se os dados foram salvos, e false se não
     */
    public boolean salvarDados(String url, int nmrPages){...6 linhas }

    /**
     *
     * @return uma lista de produtos
     */
    public ListaProduto ler(){...4 linhas }

    /**
     *
     * @return os titulos da lista de produto
     */
    public List<String> pegarTodosOsTitulos(){...3 linhas }
```

Figura 4 – Classe do Controlador (Facade)

3. Resultados e Discussões

Apesar de todo o sistema estar funcional, a expansão dele pode ser grande ainda. O framework criado pode acessar apenas o site do Magazine Luiza, mas poderia acessar um numero muito maior de sites, podendo comparar produtos dos diversos sites, além disso, o sistema poderia fazer a comparação de preço com o decorrer do tempo,

podendo informar o usuário se a oferta do produto é boa ou não, mas para fazer tudo isso seria necessário mais tempo e esforço, pois cada site organiza os dados dos produtos de forma diferente, dificultando a aplicação.

4. Conclusão

Este framework apesar de ainda ser pequeno ele faz o papel que lhe foi empregado, que é acessar e disponibilizar os dados de um site, e tem um grande potencial de evolução, podendo se tornar um sistema complexo e muito mais completo que o atual.

Ele foi feito de forma organizada, coesa e desacoplada, para que caso quando houver alguma evolução ele possa ser feito de maneira mais fácil, além de facilitar o entendimento do código para outros programadores.

Referências

URDHWARESHE, ASHWIN. Object-Oriented Programming and its Concepts. **Innovative Space of Scientific Research Journals**, v. 26, n. 1, p. 1–6, 2016.

OLAN, MICHAEL. Unit Testing: Test Early, Test Often. **Journal of Computing Sciences in Colleges**, v. 19, n. 2, p. 319–328, 2003.

HEDLEY, JONATHAN. **Jsoup**. Disponível em: <https://jsoup.org/>. Acessado em: 3 de Julho de 2019.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Abstraction and Reuse of Object-Oriented Design**. 1992.

STENCEL, K.; WEGRZYNOWICZ, P. **Implementation Variants of the Singleton Design Pattern**. 2008.

BOSCH, J. **Design Patterns as Language Constructs**. 1998.