CONGRESSO CATARINENSE DE CIÊNCIA DA COMPUTAÇÃO
2022. Ano 9, Vol. 9. ISSN: 2319-0655

CRIAÇÃO DE BIBLIOTECA PARA SOFTWARE NUTRICIONAL

Vinícius Marzall Lippel
Instituto Federal Catarinense - Câmpus Rio do Sul
viniciusmlippel@gmail.com

Abstract. *The aim of this project was to develop a library to facilitate the creation of softwares for nutritional purposes, allowing the storage of nutritional values data. It was developed in Java language and allowed the application of a wide range of advanced OOP concepts.*
Key-words: *Nutritional; Library; Java.*

Resumo. *Este projeto buscou desenvolver uma biblioteca para facilitar a elaboração de softwares de controle nutricional, permitindo o armazenamento de dados referentes a valores nutricionais. O desenvolvimento ocorreu na linguagem Java e permitiu a aplicação de diversos conceitos avançados de programação orientada a objetos.*
Palavras-chave: *Biblioteca; Nutricional; Java.*

1. Introdução

Este trabalho visa, com o objetivo de melhorar a compreensão dos conceitos de programação orientada a objetos, a implementação de uma biblioteca para implementação de softwares de controle nutricional. Ela foi desenvolvida na linguagem Java, utilizando alguns dos conceitos aprendidos ao decorrer da matéria de POOII, como princípios SOLID, Injeção de Dependência, Generics e Design Patterns e buscando empregar boas práticas de programação. Testes unitários foram implementados através da ferramenta JUnit, para garantir a consistência do projeto, e a documentação foi especificada utilizando Javadoc.

2. Desenvolvimento

Para o armazenamento dos valores nutricionais de ingredientes, alimentos e refeições foram criadas quatro classes principais de armazenamento: *NutriValue*, para valores nutricionais, *Ingredient*, para ingredientes, *Food*, para alimentos, e *Meal*, para refeições. Estas classes interagem entre si, onde um alimento possui uma lista de ingredientes, uma refeição possui uma lista de alimentos e uma lista de ingredientes, e todas estas possuem um atributo do tipo *NutriValue*, para armazenamento dos respectivos valores nutricionais. Existem ainda as classes Mineral e Vitamin, que armazenam as características de cada mineral e proteína, e a classe Fat, que armazena as quantidades de gorduras totais, saturadas e trans do valor nutricional.

2.1. Classe NutriValue

A classe NutriValue é a mais importante do sistema. Nela são armazenados os dados referentes aos valores de uma tabela nutricional de um determinado alimento, ingrediente ou refeição. O atributo *servingSize* é responsável por armazenar a que tamanho de porção (por exemplo, 100g) a que referem-se os valores seguintes. Os atributos *calories, carbs* e *protein* armazenam as quantidades de calorias, carboidratos e proteínas, respectivamente. O atributo *fat* armazena um objeto do tipo *Fat,* que contém as quantidades de gorduras totais, saturadas e trans. Os atributos *mineralList* e *vitaminList* armazenam seus respectivos objetos de lista, que contém um *ArrayList* do tipo *Amount* do tipo *Mineral* ou *Vitamin*, onde é armazenado um mineral ou vitamina e sua quantidade.

```
14 public class NutriValue {
15
16     private double servingSize;
17     private double calories;
18     private double carbs;
19     private Fat fat;
20     private double protein;
21     private MineralList mineralList;
22     private VitaminList vitaminList;
```

Figura 1 - Atributos da classe NutriValue

Os métodos empregados pela classe são: *addMineral()* e *addVitamin(),* que adicionam minerais e vitaminas à suas respectivas listas; *sum(),* que soma os valores de outro objeto NutriValue aos seus próprios; *proportional(),* que retorna os valores em porcentagem (escala de 0 a 1); e *multiply(),* que multiplica os valores por um determinado número.

```
169    public void addVitamin(Vitamin vitamin, double amount) {
170        Amount<Vitamin> newVitamin = new Amount<Vitamin>(vitamin, amount);
171        if(this.vitaminList == null)
172            this.vitaminList = new VitaminList();
173        this.vitaminList.add(newVitamin);
174    }
```

Figura 2 - Método *addVitamin()*

```
182    public void sum(NutriValue nutriValue) {
183        this.servingSize = this.servingSize + nutriValue.getServingSize();
184        this.calories = this.calories + nutriValue.getCalories();
185        this.carbs = this.carbs + nutriValue.getCarbs();
186
187        if(this.fat != null)
188            this.fat.sum(nutriValue.getFat());
189        else
190            this.fat = nutriValue.getFat();
191
192        this.protein = this.protein + nutriValue.getProtein();
193
194        if(nutriValue.getMineralList() != null) {
195            if(this.mineralList == null)
196                this.mineralList = new MineralList();
197            this.mineralList.sum(nutriValue.getMineralList());
198        }
199
200
201        if(this.vitaminList != null && nutriValue.getVitaminList() != null)
202            this.vitaminList.sum(nutriValue.getVitaminList());
203        else
204            this.vitaminList = nutriValue.getVitaminList();
205    }
```

Figura 3 - Método *sum()*

```
213    public NutriValue proportional() {
214        NutriValue prop = new NutriValue();
215        prop.setServingSize(1);
216        prop.setCalories(this.calories / this.servingSize);
217        prop.setCarbs(this.carbs / this.servingSize);
218        if(this.fat != null)
219            prop.setFat(this.fat.proportional(this.servingSize));
220        prop.setProtein(this.protein / this.servingSize);
221        if(this.mineralList != null)
222            prop.setMineralList(this.mineralList.proportionTo(this.servingSize));
223        if(this.vitaminList != null)
224            prop.setVitaminList(this.vitaminList.proportionTo(this.servingSize));
225        return prop;
226    }
```

Figura 4 - Método *proportional()*

Além disso, para a construção de um objeto utiliza-se o padrão de projeto Fluent Interface. Isto permite com que a classe seja instanciada com diferentes combinações de atributos, sem ser necessária a criação de construtores diferentes para cada ocasião.

```
48    public NutriValue carbs(double carbs) {
49        this.carbs = carbs;
50        return this;
51    }
```

Figura 5 - Exemplo de Fluent Interface

2.2. Classe Ingredient

A classe *Ingredient* armazena as informações de um ingrediente. O atributo *name* armazena o nome do ingrediente, *info,* suas informações, e *nutriValue,* seu valor nutricional.

```
11 public class Ingredient {
12
13     private String name;
14     private String info;
15     private NutriValue nutriValue;
```

Figura 6 - Atributos da classe Ingredient

2.3. Classe Food

A classe *Food* armazena as informações de um alimento ou prato. O atributo *name* armazena seu nome, *info,* suas informações, *ingredientList,* sua lista de ingredientes, e *nutriValue,* seu valor nutricional.

```
12 public class Food {
13
14     private String name;
15     private String info;
16     private IngredientList ingredientList;
17     private NutriValue nutriValue;
```

Figura 7 - Atributos da classe Food

Além disso, a classe possui também o método *addIngredient(),* que adiciona um novo ingrediente à lista de ingredientes, de forma equivalente ao método *addVitamin()* representado na figura 2.

2.4. Classe Meal

A classe *Food* armazena as informações de uma refeição. O atributo *name* armazena seu nome, *info,* suas informações, *foodList,* sua lista de alimentos, *ingredientList,* sua lista de ingredientes, e *nutriValue,* seu valor nutricional.

```
13 public class Meal {
14
15     private String name;
16     private String info;
17     private FoodList foodList;
18     private IngredientList ingredientList;
```

Figura 8 - Atributos da classe Meal

Os métodos contidos nesta classe são *addFood(), addIngredient()* e *nutriValue(),* o último possuindo a função de calcular o valor nutricional da refeição com base nos ingredientes e alimentos que a compõem.

```
124     public NutriValue nutriValue(double servingSize) {
125         NutriValue total = new NutriValue();
126         if(this.foodList != null) {
127             NutriValue foodNV = this.foodList.nutriValue(servingSize);
128             total.sum(foodNV);
129         }
130         if(this.ingredientList != null) {
131             NutriValue ingrNV = this.ingredientList.nutriValue(servingSize);
132             total.sum(ingrNV);
133         }
134 
135         NutriValue nutriValue = total.proportional();
136         nutriValue.multiply(servingSize);
137         return nutriValue;
138     }
```

Figura 9 - Método nutriValue()

2.5. Classes Mineral e Vitamin

As classes *Mineral* e *Vitamin* possuem a função de armazenar os dados referentes aos minerais e vitaminas presentes no valor nutricional. Possuem apenas os atributos *name* e *info,* para armazenar seu nome e informações.

```
 9 public class Mineral {
10 
11     private String name;
12     private String info;
```

Figura 10 - Atributos da classe Mineral

2.6. Classe Fat

A classe Fat armazena as quantidades de gorduras totais, saturadas e trans presentes no valor nutricional. Seus atributos são *total, saturated* e *trans.*

```
10 public class Fat {
11     private double total;
12     private double saturated;
13     private double trans;
```

Figura 11 - Atributos da classe Fat

Seus métodos são *sum(), proportional()* e *multiply(),* que possuem funções equivalentes aos métodos de mesmo nome citados nos itens anteriores.

2.7. Classes de Lista

As classes de lista servem para armazenar a lista de um determinado objeto e suas quantidades, utilizando um ArrayList do tipo *Amount*. Elas foram criadas para possibilitar a criação de métodos específicos referentes a aplicação desta biblioteca. O método *add(),* por exemplo, não funciona como um add() comum: caso o objeto que esteja sendo adicionado à lista já esteja contido nela, este não é adicionado de novo, apenas suas quantidades são somadas.

```
14 public class VitaminList {
15
16     private ArrayList<Amount<Vitamin>> vitaminList;
```

Figura 12 - Exemplo de atributo de classe de lista

```
53    public void add(Amount<Vitamin> vitamin) {
54        Amount<Vitamin> inList = searchByName(vitamin.getObject().getName());
55        if(inList == null)
56            this.vitaminList.add(vitamin);
57        else {
58            int i = this.vitaminList.indexOf(inList);
59            double sum = this.vitaminList.get(i).getAmount() + vitamin.getAmount();
60            this.vitaminList.get(i).setAmount(sum);
61        }
62    }
```

Figura 13 - Exemplo de método add()

2.7. Classe Amount

Por último, mas não menos importante, temos a classe *Amount,* que possui a função de armazenar um objeto genérico e sua quantidade. Sua importância na biblioteca consiste em permitir a criação das listas de objetos e suas quantidades.

```
11 public class Amount <T> {
12
13     private T object;
14     private double amount;
```

Figura 14 - Atributos da classe Amount

3. Conclusão

O desenvolvimento deste trabalho foi de suma importância para a aplicação dos dos conceitos aprendidos no decorrer da matéria de POOII. Além possibilitar a implementação de certos conceitos, incentivou a busca por uma melhor compreensão também daqueles que não foram aplicados, dado ter sido necessário verificar quais deles se encaixavam ou não no escopo do projeto. Apesar de ainda necessitar algumas alterações e expansões, o desenvolvimento desta biblioteca alcançou um estágio

razoável, podendo ser usado como base para o desenvolvimento de uma biblioteca semelhante porém mais completa.