CONGRESSO CATARINENSE DE CIÊNCIA DA COMPUTAÇÃO
2023. Ano 10 Vol. 10 ISSN: 2319-0655

RELATÓRIO DE DESENVOLVIMENTO DE UMA BIBLIOTECA DE MANIPULAÇÃO DE IMAGENS NA LINGUAGEM JAVA

Júlio Werner Zanatta Koepsel[1], Rodrigo Curvello[2]
[1]Estudante do curso de Bacharelado em Ciências da Computação do Instituto Federal Catarinense, IFC, Rio do Sul – SC
[2]Professor do curso de Bacharelado em Ciências da Computação do Instituto Federal Catarinense, IFC, Rio do Sul – SC
julio.koepsel@gmail.com, rodrigo.curvello@ifc.edu.br

Abstract. *This report addresses the development of an image manipulation library in the Java programming language, using the Builder and Prototype creational design patterns. The project was developed as the final activity of the Object-Oriented Programming II discipline.*
Key-words: *Library; Image Manipulation; Java; Design Patterns; Builder; Prototype.*

Resumo. *Este relatório aborda o desenvolvimento de uma biblioteca de manipulação de imagens na linguagem de programação Java, utilizando os padrões de design criacional Builder e Prototype. O projeto foi desenvolvido como atividade final da disciplina de Programação Orientada a Objetos II.*
Palavras-chave: *Biblioteca; Manipulação de Imagens; Java; Padrões de Design; Builder; Prototype.*

1. Introdução

Padrões de projeto, ou padrões de design, são soluções típicas para problemas comuns no projeto de software. Eles são como projetos pré-fabricados que o desenvolvedor pode personalizar para resolver um problema recorrente de design em seu código. Não é possível simplesmente encontrar um padrão e copiá-lo em um programa, da mesma forma que é possível com funções ou bibliotecas prontas para uso. O padrão não é um trecho específico de código, mas um conceito geral para resolver um problema específico. O desenvolvedor pode seguir os detalhes do padrão e implementar uma solução que se adapte à realidade do seu próprio programa (REFACTORINGGURU, 2023).

O padrão Builder é um padrão de design criacional, ou seja, ele está envolvido com a criação de objetos. Ele foi introduzido para resolver alguns dos problemas com os padrões de design Factory e Abstract Factory quando o objeto contém muitos atributos. Esses problemas com grandes números de parâmetros podem ser resolvidos fornecendo um construtor com os parâmetros necessários e, em seguida, diferentes métodos setter para definir os parâmetros opcionais. O problema com esta abordagem é que o estado do objeto será inconsistente até que todos os atributos sejam definidos explicitamente. O padrão Builder resolve o problema com grande número de parâmetros opcionais e

estado inconsistente, fornecendo uma maneira de construir o objeto passo a passo e fornecendo um método que realmente retornará o objeto final (PANKAJ, 2022a).

O padrão Prototype, assim como o Builder, também é um padrão de design criacional. Ele é usado quando a criação de um objeto é cara e requer muito tempo e recursos e o desenvolvedor já possui um objeto semelhante. O padrão de Prototype fornece um mecanismo para copiar o objeto original para um novo objeto e então modificá-lo de acordo com as necessidades (PANKAJ, 2022b).

Este trabalho tem como objetivo o desenvolvimento de uma biblioteca de manipulação de imagens em Java, utilizando os padrões Builder e Prototype. O padrão Builder será utilizado para construir uma ordem de métodos de manipulação à uma imagem, como aplicar filtros, redimensionar ou girar, enquanto o padrão Prototype será utilizado para clonar uma imagem parcialmente manipulada para servir de base a outras manipulações.

2. Desenvolvimento

A biblioteca foi desenvolvida com a linguagem de programação Java e a IDE Visual Studio Code. O código final está disponível em um repositório GitHub através do link: https://github.com/juliokoepsel/manipuladorDeImagens.

A figura 1 mostra o diagrama de classe, que representa a estrutura do projeto, contendo 3 objetos no total. A interface ImagemPrototype permite a criação de clones para o padrão Prototype, enquanto a classe ManipuladorImagem implementa essa interface e é responsável pela manipulação da imagem desejada. A classe aninhada Builder corresponde ao padrão de mesmo nome e é responsável pela construção da imagem selecionada, através de métodos de manipulação.

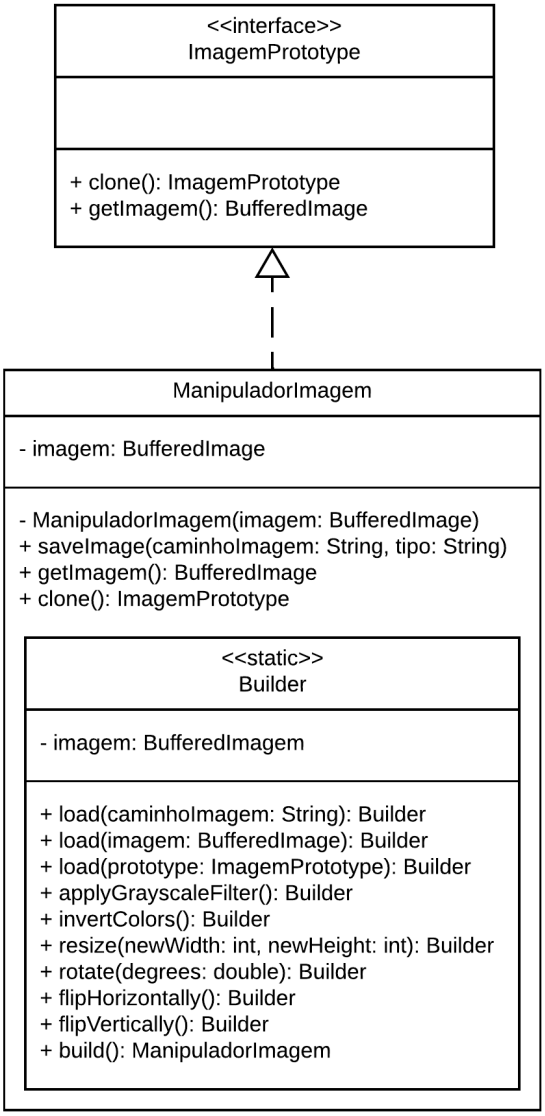

Figura 1 – Diagrama de Classe

Fonte: Dados Primários, 2023.

A figura 2 mostra parte da classe ManipuladorImagem, com construtor privado e métodos de gravação da imagem no sistema do usuário e de clonagem do objeto.

```
14  public class ManipuladorImagem implements ImagemPrototype {
15
16      private BufferedImage imagem;
17
18  >   /**...
23      private ManipuladorImagem(BufferedImage imagem) {
24          this.imagem = imagem;
25      }
26
27  >   /**...
34      public void saveImage(String caminhoImagem, String tipo) throws IOException {
35          File file = new File(caminhoImagem);
36          ImageIO.write(imagem, tipo, file);
37      }
38
39  >   /**...
42      @Override
43      public BufferedImage getImagem() {
44          return imagem;
45      }
46
47  >   /**...
50      @Override
51      public ImagemPrototype clone() {
52          return new ManipuladorImagem(imagem);
53      }
```

Figura 2 – Parte da classe ManipuladorImagem

Fonte: Dados Primários, 2023.

A figura 3 mostra a classe aninhada Builder minimizada, onde ficam os métodos de carregar imagem, de manipulação da imagem e de construção do objeto.

```
61      public static class Builder {
62          private BufferedImage imagem;
63
64  >       /**...
71  >       public Builder load(String caminhoImagem) throws IOException {...
76  >       /**...
83  >       public Builder load(BufferedImage imagem) throws IOException {...
87  >       /**...
94  >       public Builder load(ImagemPrototype prototype) throws IOException {...
98
99  >       /**...
104 >       public Builder applyGrayscaleFilter() {...
127 >       /**...
132 >       public Builder invertColors() {...
154 >       /**...
161 >       public Builder resize(int newWidth, int newHeight) {...
167 >       /**...
173 >       public Builder rotate(double degrees) {...
192 >       /**...
197 >       public Builder flipHorizontally() {...
213 >       /**...
218 >       public Builder flipVertically() {...
233
234 >       /**...
237         public ManipuladorImagem build() {
238             return new ManipuladorImagem(imagem);
239         }
240     }
```

Figura 3 – Classe aninhada Builder

Fonte: Dados Primários, 2023.

Após o desenvolvimento das classes principais da biblioteca, foi desenvolvido um teste para verificar o funcionamento dos métodos mais importantes do manipulador de imagens. A figura 4 mostra os dois testes de manipulação e armazenamento de uma imagem e de clonagem e utilização do Prototype como base para mais manipulações de uma imagem.

```java
public class ManipuladorImagemTest extends TestCase {

    /**...
    @Test
    public void testSaveImage() throws IOException {
        File file = new File("imagem_cinza_teste.jpg");
        if (file.exists()) {
            file.delete();
        }
        ManipuladorImagem imagemTeste = new ManipuladorImagem.Builder()
            .load(caminhoImagem:"imagem.jpg")
            .applyGrayscaleFilter()
            .build();
        imagemTeste.saveImage(caminhoImagem:"imagem_cinza_teste.jpg", tipo:"jpg");
        file = new File("imagem_cinza_teste.jpg");
        assertEquals(file.exists(), true);
    }

    /**...
    @Test
    public void testClone() throws IOException {
        File file = new File("imagem_invertida_teste.jpg");
        if (file.exists()) {
            file.delete();
        }
        ManipuladorImagem imagemTeste = new ManipuladorImagem.Builder()
            .load(caminhoImagem:"imagem.jpg")
            .build();
        ImagemPrototype imagemPrototypeTeste = imagemTeste.clone();
        imagemTeste = new ManipuladorImagem.Builder()
            .load(imagemPrototypeTeste)
            .invertColors()
            .build();
        imagemTeste.saveImage(caminhoImagem:"imagem_invertida_teste.jpg", tipo:"jpg");
        file = new File("imagem_invertida_teste.jpg");
        assertEquals(file.exists(), true);
    }
}
```

Figura 4 – Testes dos Métodos

Fonte: Dados Primários, 2023.

A figura 5 mostra o antes e depois da manipulação de uma imagem pelo teste testSaveImage, onde foi aplicado um filtro cinza sobre a imagem.

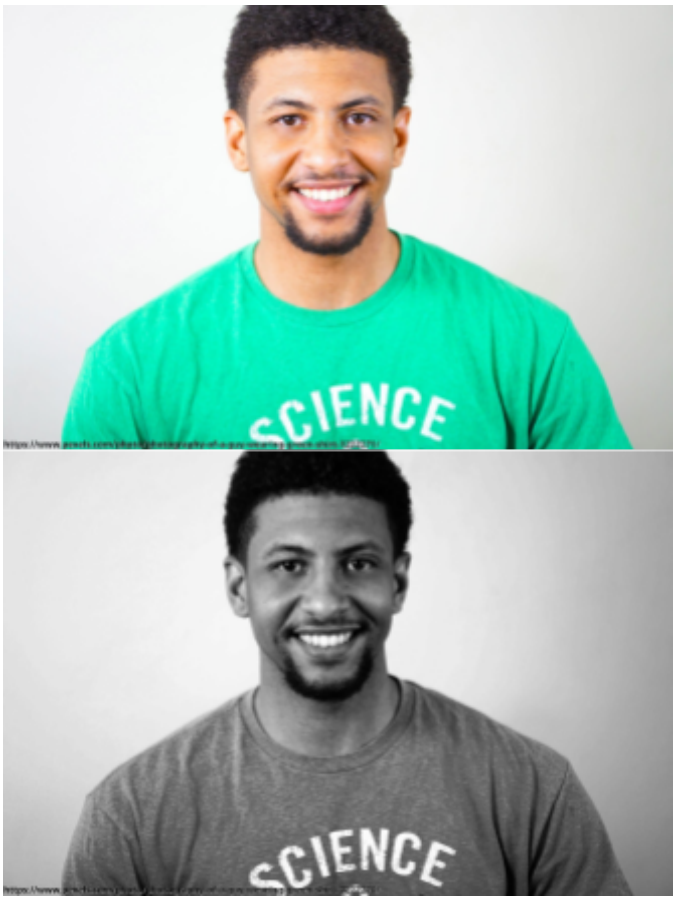

Figura 5 – Resultado do Teste

Fonte: Dados Primários, 2023.

A documentação completa do código está disponível em: https://juliokoepsel.github.io/juliokoepsel.github.io.

3. Considerações Finais

Existem diversos padrões de design para serem aplicados no desenvolvimento de um software, cada um com sua especialidade. Os padrões Builder e Prototype tem como foco a parte da criação do objeto.

Através da utilização desses dois padrões, foi possível desenvolver uma biblioteca de manipulação de imagens simples e de fácil uso. Entretanto, o desempenho dos métodos de manipulação não foi algo considerado durante o desenvolvimento do projeto, podendo ser algo abordado em futuros projetos.

Referências

REFACTORINGGURU. **What's a design pattern?**. 2023. Disponível em:

<https://refactoring.guru/design-patterns/what-is-pattern>. Acesso em: 05 dez. 2023.

PANKAJ. **Builder Design Pattern in Java**. 2022a. Disponível em: <https://www.digitalocean.com/community/tutorials/builder-design-pattern-in-java>. Acesso em: 05 dez. 2023.

PANKAJ. **Prototype Design Pattern in Java**. 2022a. Disponível em: <https://www.digitalocean.com/community/tutorials/prototype-design-pattern-in-java>. Acesso em: 05 dez. 2023.

https://medium.com/@julio.koepsel/relat%C3%B3rio-de-desenvolvimento-de-uma-biblioteca-de-manipula%C3%A7%C3%A3o-de-imagens-na-linguagem-java-cf14053e1008