Ministério da Educação
Secretaria de Educação Profissional e Tecnológico
Instituto Federal Catarinense
Campus Rio do Sul - Unidade Urbana

Vinicius Goulart Almeida

Design Patterns Criacionais

(Fluente Interface, Abstract Factory e Prototype)

Rio do Sul

Setembro de 2020

1. Prototype

Prototype é um Padrão de Projeto, classificado como um padrão de criação, utilizado para instanciar uma classe que pode ser muito trabalhosa ou complicada de instanciar, permite a criação de objetos a partir de um modelo original. Ou seja, o padrão Prototype permite que você instancie uma classe a partir de uma instância já existente, copiando as informações da instância original (mesmos atributos e comportamentos), a partir do método Clone(). Esse padrão de criação aproveita o estado existente de um objeto, permitindo a diminuição do tempo no processo de instanciação, facilitando a complexidade da criação de um novo objeto e gerando objetos na qual o tipo é desconhecido na instanciação. Como exemplo vou utilizar os animais:

No código mostramos como ficou nossa classe animal com todas as variáveis e com os métodos que iremos utilizar no exemplo.

```
1  public class Animal implements Cloneable {
2
3          private String nome;
4          private int idade;
5          private String cor;
6          private int qtdPatas;
7
8          public String getNome() {
9                  return nome;
10         }
11         public void setNome(String nome) {
12                 this.nome = nome;
13         }
14         public int getIdade() {
15                 return idade;
16         }
17         public void setIdade(int idade) {
18                 this.idade = idade;
19         }
20         public String getCor() {
21                 return cor;
22         }
23         public void setCor(String cor) {
24                 this.cor = cor;
25         }
26         public int getQtdPatas() {
27                 return qtdPatas;
28         }
29         public void setQtdPatas(int qtdPatas) {
30                 this.qtdPatas = qtdPatas;
31         }
32
33         public Object clone() {
34                 Object obj = null;
35
36                 try {
37                         obj = super.clone();
```

```
38                } catch (Exception e) {
39                      e.printStackTrace();
40                }
41                return obj;
42          }
43
44          @Override
45          public String toString() {
46                return "Animal [nome=" + nome + ", idade=" + idade
47    + ", cor=" + cor + ", qtdPatas=" + qtdPatas + "]";
48          }
49    }
```

Classe protótipo que irá fazer a clonagem do Objeto Animal.

```
1   public class PrototypeAnimal {
2
3         Animal prototypeAnimal;
4
5         public PrototypeAnimal(Animal animal) {
6               this.prototypeAnimal = animal;
7         }
8
9         public Animal geraClone() {
10              return (Animal) prototypeAnimal.clone();
11        }
12
13  }
```

.

Aqui temos nossa classe específica que neste caso utilizei o exemplo de um Gato.

```
1   public class Gato extends Animal {
2
3         public Gato(String nome, int idade, String cor, int
4   qtdPatas){
5               setNome(nome);
6               setIdade(idade);
7               setCor(cor);
8               setQtdPatas(qtdPatas);
9         }
10
11  }
```

E pra finalizar irei mostrar o Main, logo abaixo:

```
1   public class Main {
2         public static void main(String[] args) {
3
4               Animal animalParaClonar = new Gato("Garfield", 5,
5   "Amarelo", 4);
6
7               System.out.println("Animal para clonar:
8   "+animalParaClonar.toString());
9
10
```

```
11              PrototypeAnimal prototypeClonar = new
12  PrototypeAnimal(animalParaClonar);
13
14
15              Animal animalClonado =
16  prototypeClonar.geraClone();
17              animalClonado.setNome("Hulk");
18              System.out.println("1º Animal clonado:
19  "+animalClonado.toString());
20
21
22              Animal animalClonado2 =
23  prototypeClonar.geraClone();
24              animalClonado2.setIdade(9);
25              System.out.println("2º Animal clonado:
26  "+animalClonado2.toString());
27
28
29              Animal animalClonado3 =
30  prototypeClonar.geraClone();
31              animalClonado3.setCor("Branco");
32              System.out.println("1º Animal clonado:
33  "+animalClonado3.toString());
34
35        }
36  }
```

2. Abstract Factory

Abstract Factory é um padrão de projeto utilizados para criar grupos de objetos relacionados ou dependentes sem especificar suas classes concretas. Possui como objetivo remover a criação de objetos da classe de negócio. Permite que o cliente use uma interface abstrata para criar um conjunto de produtos sem saber se os produtos concretos serão ou não produzidos. Abstract Factory produz uma interface, na qual cada método criará um produto concreto e é implementado uma subclasse da Abstract Factory para fornecer as implementações. Por exemplo, criar uma interface que seria uma fábrica de jogos, essa fábrica cria outras fábricas, na qual cada fábrica cria um objeto de cada tipo, como uma fábrica de jogos de mundo aberto e outra fábrica para jogos de tiro.

Essa é minha interface para criar as fábricas de jogos com suas especialidades.

```
1   public interface IFabricaDeJogo {
2
3         IJogoMundoAberto criarJogoMundoAberto();
4         IJogoTiro criarJogoTiro();
5
6   }
```

Para criar os jogos de mundo aberto crio mais uma interface com a especialização em criar os jogos de mundo aberto, repetindo o processo com os jogos de tiro como demonstrado abaixo.

```
1  public interface IJogoMundoAberto {
2
3        String nomeJogoMundoAberto();
4        String categoriaJogoMundoAberto();
5
6  }
```

Após criamos a fábrica específica de cada construtora nesse exemplo vou utilizar a Rockstar games que cria jogos.

```
1  public class FabricaDeJogoRockstarGames implements
2  IFabricaDeJogo {
3
4        public IJogoMundoAberto criarJogoMundoAberto() {
5              return new GrandTheftAutoV();
6        }
7
8        public IJogoTiro criarJogoTiro() {
9              return new MaxPayne3();
10       }
11 }
```

Agora criamos os jogos da Rockstar com a Classe GrandTheAutoV e MaxPayne3, irei demonstrar somente um.

```
1  public class GrandTheftAutoV implements IJogoMundoAberto {
2
3        public String nomeJogoMundoAberto() {
4              return "Grand Theft Auto V (Rockstar Games)";
5        }
6
7        public String categoriaJogoMundoAberto() {
8              return "Jogo de Mundo Aberto";
9        }
10
11 }
```

Para finalizar irei mostrar o Cliente(Main):

```
1  public class ExCliente {
2        public static void main(String[] args) {
3
4              IFabricaDeJogo fabrica = null;
5              IJogoMundoAberto jogoMundoAberto = null;
6              IJogoTiro jogoTiro = null;
7  //ROCKSTAR GAMES
8              fabrica = new FabricaDeJogoRockstarGames();
9
10             jogoMundoAberto = fabrica.criarJogoMundoAberto();
11
```

```
12                System.out.println("\nNome do jogo:
13  "+jogoMundoAberto.nomeJogoMundoAberto());
14                System.out.println("Categoria do jogo:
15  "+jogoMundoAberto.categoriaJogoMundoAberto());
16
17                jogoTiro = fabrica.criarJogoTiro();
18                System.out.println("\nNome do jogo:
19  "+jogoTiro.nomeJogoTiro());
20                System.out.println("Categoria do jogo:
21  "+jogoTiro.categoriaJogoTiro());
22
23          }
24  }
```

3. Fluente Interface

Interfaces fluentes é uma maneira de simplificar a execução sequencial de funcionalidades disponibilizadas por um objeto. Trata-se de uma técnica que permite, basicamente, o agrupamento de um conjunto de chamadas a uma instância em uma única instrução. Classes concebidas segundo este padrão, dispensam os desenvolvedores de referenciar a todo o momento uma mesma estrutura, além de tornar mais conciso e legível o código-fonte que define um determinado comportamento. Como exemplo irei utilizar as notas fiscais, segue código abaixo:

Aqui criamos nossa interface com os métodos que desejamos ter no nosso programa.

```
1   public interface INotaFiscal {
2
3          INotaFiscal CalcularVlTotalProdutos();
4       INotaFiscal CalcularVlBaseIPI();
5       INotaFiscal CalcularVlIPI();
6       INotaFiscal AplicarDesconto(double percentual);
7       INotaFiscal CalcularVlBaseICMS();
8       INotaFiscal CalcularVlICMS();
9       INotaFiscal CalcularTotalNotaFiscal();
10  }
```

Após a criação da interface criamos nossa classe com a implementação dos métodos.

```
1   public class NotaFiscal implements INotaFiscal {
2
3          public NotaFiscal CalcularVlTotalProdutos()
4       {
5                // Instruções para cálculo do valor
6           // total dos produtos
7
8           return this;
9       }
10
```

```
11      public NotaFiscal CalcularVlBaseIPI()
12      {
13         // Instruções para cálculo do valor
14          // da base do IPI
15
16          return this;
17      }
18
19      public NotaFiscal CalcularVlIPI()
20      {
21         // Instruções para cálculo do valor do IPI
22
23          return this;
24      }
25
26      public NotaFiscal AplicarDesconto(double percentual)
27      {
28         // Aplicar percentual de desconto sobre
29          // o total dos produtos
30
31          return this;
32      }
33
34      public NotaFiscal CalcularVlBaseICMS()
35      {
36          // Instruções para cálculo do valor
37          // da base do ICMS
38
39          return this;
40      }
41
42      public NotaFiscal CalcularVlICMS()
43      {
44
45          // Instruções para cálculo do valor do ICMS
46
47          return this;
48      }
49
50      public NotaFiscal CalcularTotalNotaFiscal()
51      {
52         // Calcular total da nota considerando
53          // descontos e tributos que deverão ser
54          // somados ao mesmo
55
56          return this;
57      }
58  }
```

Para finalizar nosso Main a onde notamos que o código fica muito mais prático e legível.

```
1  public class Main {
2
3        public static void main(String[] args) {
4              NotaFiscal nota = new NotaFiscal()
5                           .CalcularTotalNotaFiscal()
6                           .CalcularVlBaseICMS();
7        }
8  }
```

4. Referências

BRIZENO, Marcos. Mão na massa: Abstract Factory. Disponível em: . Acesso em: 20 set. 2020.

FREEMAN, Elisabeth; FREEMAN, Eric. Use a cabeça! Padrões de Projeto. 2 ed revisada. Editora AltaBooks, 2008 MARIOTTI, Flávio S. Prototype – Padrão de Projeto com Microsoft .NET C#. Disponível em: . Acesso em: 20 set. 2020.

ROCHA, Helder da. Padrões de Desing com aplicações em Java. Disponível em: . Acesso em: 21 set. 2020.

THIENGO, Vinícius. Padrão de Projeto: Abstract Factory. Disponível em: . Acesso em: 21 set. 2020.