

Padrões de projeto: **Memento, Builder e Prototype**

Luana Fraga, Davi Jatobá, Vitor Roma

Padrão de projeto Memento

- É um padrão de projeto comportamental que permite ao usuário salvar o estado de um objeto;
- O estado salvo pode ser utilizado para reverter o objeto a um estado anterior ou, para acessar os dados do objeto, quando o acesso direto aos getters e setters do mesmo quebra o encapsulamento.

Prós x Contras

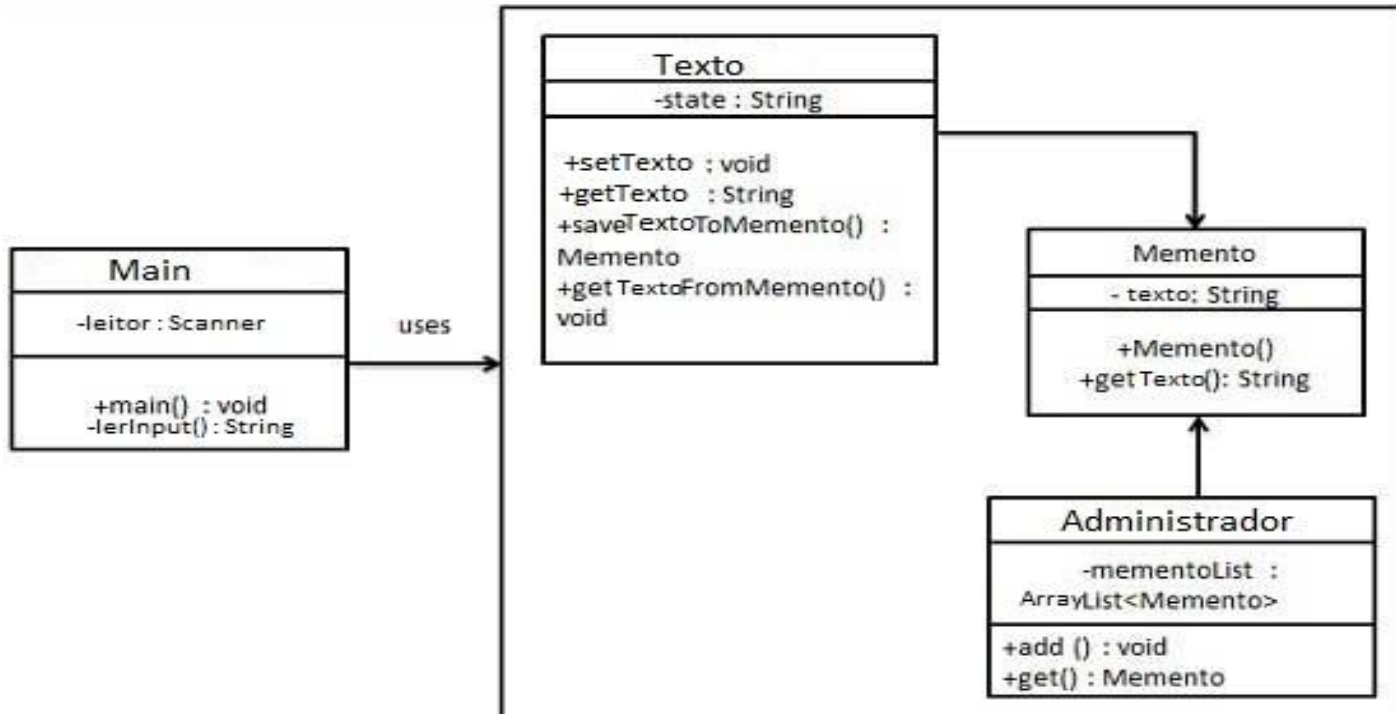
Prós:

- Produzir retratos sem violar o encapsulamento;
- Simplificar o código da classe modelo;

Contras:

- Pode consumir muita memória se os mementos forem criados com frequência e/ou se a classe classe modelo for muito grande.

Diagrama- Com uso do padrão



Código - Com uso do padrão

```
public class Texto {  
    private String texto = "";  
  
    public void set(String texto) {  
        this.texto = texto;  
    }  
  
    public String getTexto() {  
        return texto;  
    }  
  
    public Memento saveToMemento() {  
        return new Memento(texto);  
    }  
  
    public void restoreFromMemento(Memento memento) {  
        texto = memento.getSavedTexto();  
    }  
}
```

```
public class Memento {  
    private final String texto;  
  
    public Memento(String textoToSave)  
    {  
        texto = textoToSave;  
    }  
  
    public String getSavedTexto()  
    {  
        return texto;  
    }  
}
```

Código - Com uso do padrão

```
import java.util.ArrayList;

public class Administrador {
    private ArrayList<Memento> mementoList = new ArrayList<Memento>();

    public void add(Memento texto){
        mementoList.add(texto);
    }

    public Memento get(){
        return mementoList.remove(mementoList.size()-1);
    }
}
```

```

import java.util.Scanner;

public class Main {
    private static Scanner leitor = new Scanner(System.in);

    public static void main(String[] args) {
        String texto = new String("");
        Texto txt = new Texto();
        txt.saveToMemento();
        Administrador admin = new Administrador();

        while (true) {
            try {
                System.err.println("Digite: o novo texto ou, \"des\" para desfazer: ");
                texto = lerString();
                if (texto.equals("des")) {
                    txt.restoreFromMemento(admin.get());
                } else {
                    txt.set(texto);
                    admin.add(txt.saveToMemento());
                }
                System.out.println(txt.getTexto());
            } catch (IndexOutOfBoundsException e) {
                System.out.println("Não há mais texto.");
            }
        }
    }

    private static String lerString() {
        String input = leitor.nextLine();
        leitor.reset();
        return input;
    }
}

```

Resultado

```
Digite: o novo texto ou, "des" para desfazer:
Desmonstrando
Desmonstrando
Digite: o novo texto ou, "des" para desfazer:
Funcionamento
Funcionamento
Digite: o novo texto ou, "des" para desfazer:
do Memento
do Memento
Digite: o novo texto ou, "des" para desfazer:
des
do Memento
Digite: o novo texto ou, "des" para desfazer:
des
Funcionamento
Digite: o novo texto ou, "des" para desfazer:
des
Desmonstrando
Digite: o novo texto ou, "des" para desfazer:
des
Não há mais texto.
Digite: o novo texto ou, "des" para desfazer:
```


Padrão de projeto Builder

- O Builder é um padrão de projeto criacional, que permite a construção de objetos complexos passo a passo. É especialmente útil quando você precisa criar um objeto com muitas opções possíveis de configuração.

Onde se aplica? Qual a solução proposta pelo padrão?

Temos uma classe Venda, que recebe - entre outros - um atributo cliente, que é um objeto da classe “Cliente”, e uma lista de objetos da classe “Item”.

Parte da complexidade desse código é, na hora da criação de uma venda, a necessidade de se criar uma instância para cada objeto. Além é claro, de ter que criar uma lista para adicionar cada um dos itens vendidos. Sem contar que teríamos que sempre lidar com os números excessivos de argumentos nos construtores das classes, o que não é algo considerado bom de trabalhar.

A solução proposta pelo padrão é eliminar a complexidade na criação de objetos e também deixar mais intuitivo este processo

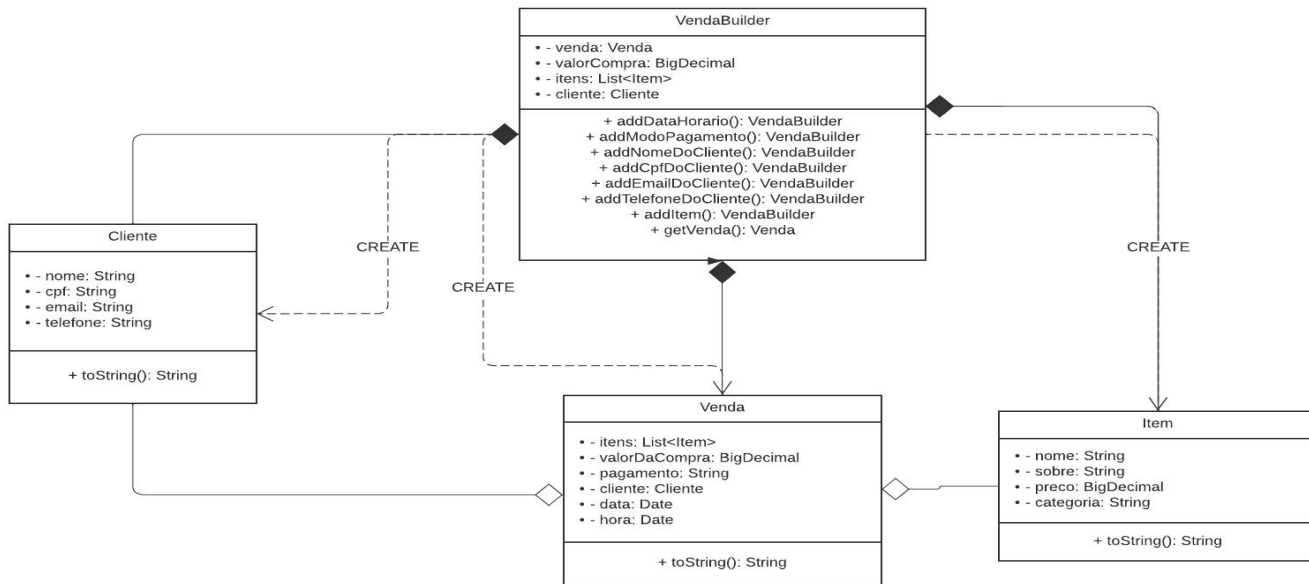
Diagrama - Sem uso do padrão:



Código - Sem uso do padrão

```
public class Main {  
    public static void main(String[] args) throws ParseException {  
  
        List<Item> itens = new ArrayList<Item>();  
  
        Cliente cliente = new Cliente("Julia", "12345678912", "julia@gmail.com", "7592934590");  
  
        BigDecimal precoComida = new BigDecimal("29.90");  
        BigDecimal precoBebida = new BigDecimal("4.00");  
        Item comida = new Item("Macarrao", "Macarrao ao molho branco", precoComida, "Massas");  
        Item bebida = new Item("Coca cola", "Coca cola em lata", precoBebida, "Bebidas");  
        itens.add(comida);  
        itens.add(bebida);  
  
        BigDecimal valorDaCompra = precoComida.add(precoBebida);  
        SimpleDateFormat formataData = new SimpleDateFormat("dd/MM/yyyy");  
        SimpleDateFormat formataDataHora = new SimpleDateFormat("HH:mm:ss");  
        Date data = formataData.parse("05/07/2022");  
        Date hora = formataDataHora.parse("12:23:53");  
  
        Venda venda = new Venda(itens, valorDaCompra, "Pix", cliente, data, hora);  
  
        System.out.println(venda);  
    }  
}
```

Diagrama- Com uso do padrão



Código - Com uso do padrão

```
0 public class VendaBuilder {
1
2     private Venda venda;
3     private Cliente cliente;
4     private List<Item> itens = new ArrayList<Item>();
5     private BigDecimal valorCompra = new BigDecimal("0");
6
7
8     public VendaBuilder() {
9         this.venda = new Venda();
10        this.cliente = new Cliente();
11    }
12
13    public static VendaBuilder builder() {
14        return new VendaBuilder();
15    }
16
17    public VendaBuilder addDataHorario(String data, String hora){
18        SimpleDateFormat formataData = new SimpleDateFormat("dd/MM/yyyy");
19        SimpleDateFormat formataDataHora = new SimpleDateFormat("HH:mm:ss");
20        Date dataDate = new Date();
21        Date horaDate = new Date();
22        try {
23            dataDate = formataData.parse(data);
24            horaDate = formataDataHora.parse(hora);
25        } catch (ParseException e) {
26            e.printStackTrace();
27        }
28
29        this.venda.setData(dataDate);
30        this.venda.setHorario(horaDate);
31        return this;
32    }
33    public VendaBuilder addModoPagamento(String pagamento){
34        this.venda.setPagamento(pagamento);
35        return this;
36    }
37 }
```

Código - Com uso do padrão

```
    }  
    public VendaBuilder addNomeDoCliente(String nome) {  
        this.cliente.setNome(nome);  
        return this;  
    }  
    public VendaBuilder addCpfDoCliente(String cpf) {  
        this.cliente.setCpf(cpf);  
        return this;  
    }  
    public VendaBuilder addEmailDoCliente(String email) {  
        this.cliente.setEmail(email);  
        return this;  
    }  
    public VendaBuilder addTelefoneDoCliente(String telefone) {  
        this.cliente.setTelefone(telefone);  
        return this;  
    }  
    public VendaBuilder addItem(String nome, String descricao, String valor, String categoria) {  
        BigDecimal valorBD = new BigDecimal(valor);  
        this.itens.add(new Item(nome, descricao, valorBD, categoria));  
        this.valorCompra.add(valorBD);  
        return this;  
    }  
    public Venda getVenda() {  
        this.venda.setCliente(cliente);  
        this.venda.setItens(itens);  
        this.venda.setPreco(valorCompra);  
        return this.venda;  
    }  
}
```

Código - Com uso do padrão

```
public class Main {  
    public static void main(String[] args) throws ParseException {  
  
        Venda venda = VendaBuilder.builder()  
            .addNomeDoCliente("Julia")  
            .addCpfDoCliente("12345678912")  
            .addModoPagamento("Pix")  
            .addDataHorario("05/07/2022", "12:23:53")  
            .addEmailDoCliente("julia@gmail.com")  
            .addTelefoneDoCliente("75992934560")  
            .addItem("Macarrão", "Macarrao ao molho branco", "29.90", "Massas")  
            .addItem("Coca cola", "Coca cola em lata", "4.00", "Bebidas")  
            .getVenda();  
  
        System.out.println(venda);  
    }  
}
```


Padrão de projeto Prototype

- Padrão de projeto Criacional que permite a clonagem de objetos, mesmo complexo, sem acoplamento à suas classes específicas

Visão Geral

- O tipo do objeto criado é determinado pelo protótipo
- Evita recriação de objetos complexos ou “caros”
- Evita explosão de subclasses

Onde se aplica? Qual a solução proposta pelo padrão?

- Utilizado geralmente quando temos a instância de classe (objeto) que será utilizado uma grande quantidade de vezes.

Diagrama - Geral:

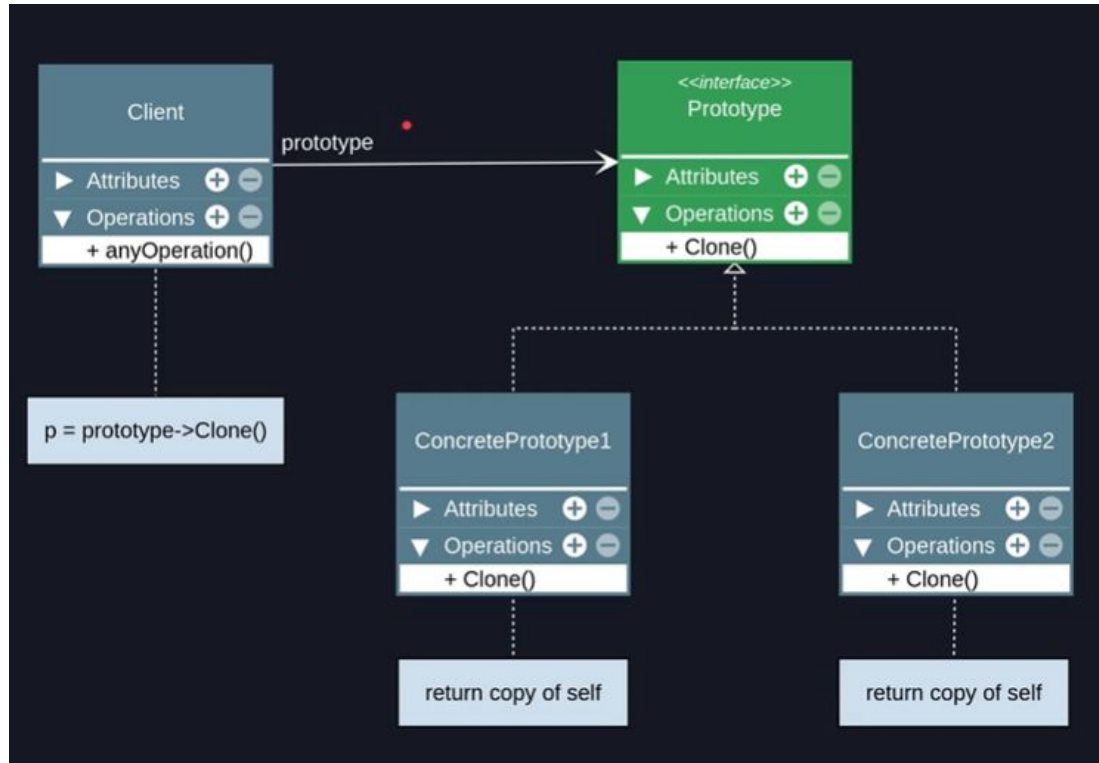
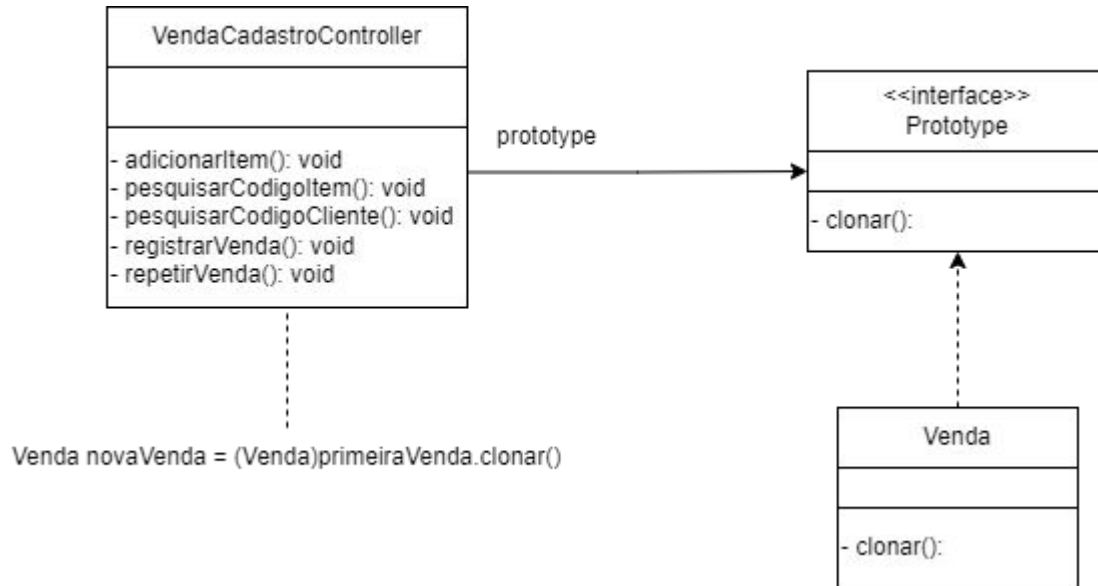


Diagrama do Projeto com Prototype



Código - Com uso do padrão (TypeScript)

```
function Person(firstName, lastName, age) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
  this.age = age;  
}  
  
const personPrototype = {  
  firstName: 'Luiz',  
  lastName: 'Miranda',  
  age: 30,  
  
  fullName() {  
    return `${this.firstName} ${this.lastName}`;  
  },  
};  
  
Person.prototype = Object.create(personPrototype);  
  
const person1 = new Person('Luiz', 'Miranda', 30);  
console.log(person1.fullName());
```

Código - Com uso do padrão (Java)

```
package app;  
  
public interface Prototype {  
    Prototype clonar();  
}
```

Código - Com uso do padrão (Java)

```
package app;

public class Carro implements Prototype{

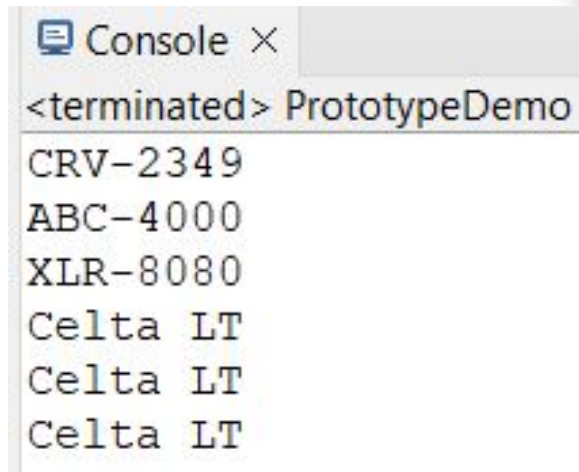
    private String modelo;
    private String marca;
    private String placa;
    private String cor;
    private String cambio;
    private boolean tetoSolar;
```

```
    public Carro(String modelo, String marca, String placa,
        String cor, String cambio, boolean tetoSolar) {
        this.modelo = modelo;
        this.marca = marca;
        this.placa = placa;
        this.cor = cor;
        this.cambio = cambio;
        this.tetoSolar = tetoSolar;
    }

    @Override
    public Prototype clonar() {
        return new Carro(modelo, placa, marca, cor, cambio, tetoSolar);
    }
}
```


Código - Com uso do padrão (Java)

```
public class PrototypeDemo {  
  
    public static void main(String[] args) {  
  
        Carro celta = new Carro("Celta LT", "JFX-9300",  
                                "Chevrolet", "Vermelho", "Manual", false  
  
        Carro celta1 = (Carro) celta.clonar();  
        Carro celta2 = (Carro) celta.clonar();  
        Carro celta3 = (Carro) celta.clonar();  
  
        celta1.setPlaca("CRV-2349");  
        celta2.setPlaca("ABC-4000");  
        celta3.setPlaca("XLR-8080");  
  
        System.out.println(celta1.getPlaca());  
        System.out.println(celta2.getPlaca());  
        System.out.println(celta3.getPlaca());  
        System.out.println(celta1.getModelo());  
        System.out.println(celta2.getModelo());  
        System.out.println(celta3.getModelo());  
    }  
}
```



Console X
<terminated> PrototypeDemo
CRV-2349
ABC-4000
XLR-8080
Celta LT
Celta LT
Celta LT

Prós x Contras

Prós:

- Clonar objetos sem acoplá-los a suas classes concretas.
- Se livrar de códigos de inicialização repetidos em troca de clonar protótipos pré-construídos.
- Produzir objetos complexos mais convenientemente.

Contras:

- Clonar objetos complexos que têm referências circulares pode ser bem complicado.

REFERÊNCIAS

- <https://www.geeksforgeeks.org/memento-design-pattern/>
- https://www.tutorialspoint.com/design_pattern/memento_pattern.htm
- <https://www.youtube.com/c/Ot%C3%A1vioMiranda>
- <https://refactoring.guru/pt-br/design-patterns/prototype/java/example>
- https://www.youtube.com/watch?v=t2HRMeBizsY&ab_channel=MikeM%C3%B8llerNielsen
- <https://www.mballem.com/post/simplificando-com-builder-pattern/>
- <https://www.devmedia.com.br/implementando-padroes-criacionais-em-java/34185>
- <https://refactoring.guru/pt-br/design-patterns/builder/java/example#:~:text=O%20Builder%20%C3%A9%20um%20padr%C3%A3o,o%20mesmo%20processo%20de%20constru%C3%A7%C3%A3o.>

**OBRIGADA PELA
SUA ATENÇÃO!**