Design Patterns (Padrões de Projetos)

Factory Method

Factory Method

O **Factory Method** é um **padrão de projeto criacional** que fornece uma interface para a criação de objetos, mas permite que as subclasses decidam **qual classe concreta será instanciada**.

Em vez de criar objetos diretamente usando new, o Factory Method delega a responsabilidade de criação para subclasses, promovendo o **princípio do aberto/fechado (OCP - Open/Closed Principle)** e reduzindo o **acoplamento** entre as classes.

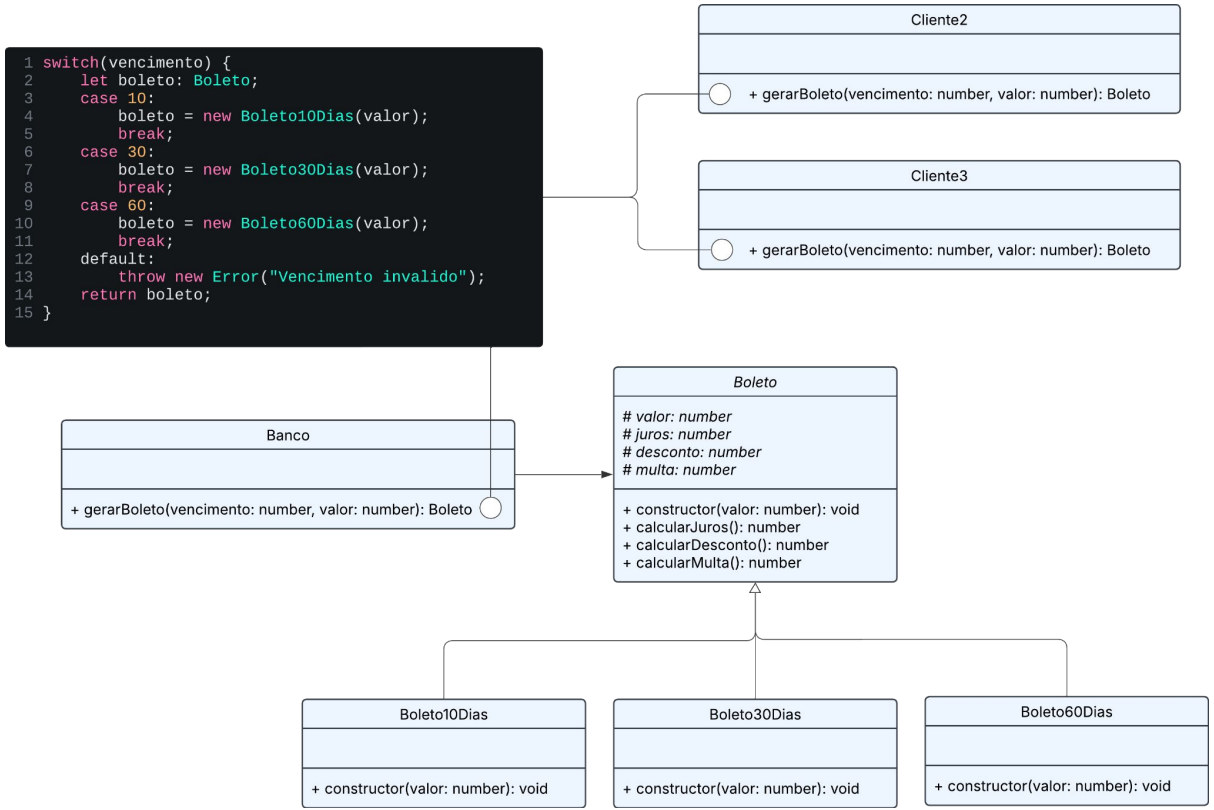
Problema

Temos um sistema de geração de boletos, onde os dias de vencimento influenciam as taxas cobradas:

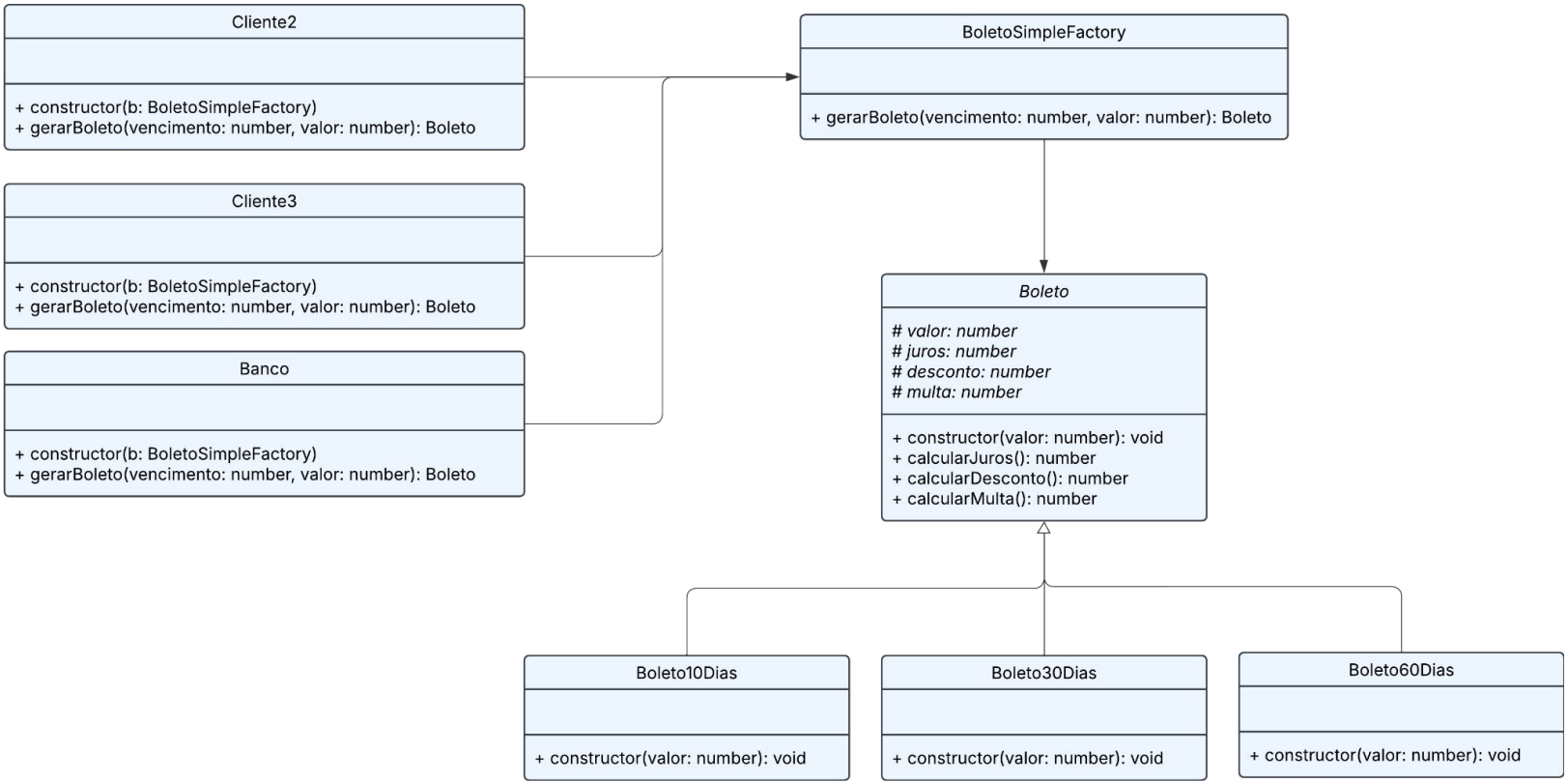
Dias p/ Vencimento	Juros	Desconto	Multa
10	2%	10%	5%
30	5%	5%	10%
60	10%	0%	20%

Problema

```
switch(vencimento) {
    let boleto: Boleto;
    case 10:
        boleto = new Boleto10Dias(valor);
        break;
    case 30:
        boleto = new Boleto30Dias(valor);
        break;
    case 60:
        boleto = new Boleto60Dias(valor);
        break;
    default:
        throw new Error("Vencimento invalido");
    return boleto;
}
```



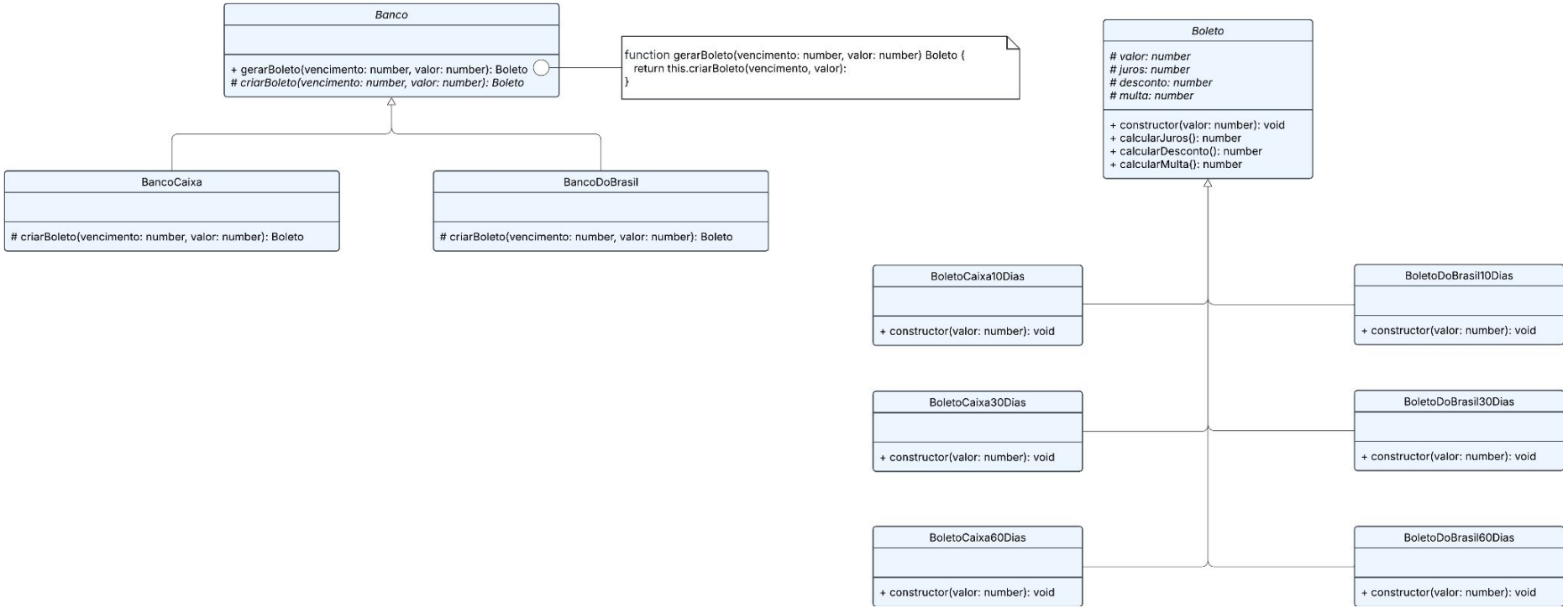
Problema


Cliente2
+ constructor(b: BoletoSimpleFactory)
+ gerarBoleto(vencimento: number, valor: number): Boleto
Cliente3
+ constructor(b: BoletoSimpleFactory)
+ gerarBoleto(vencimento: number, valor: number): Boleto
Banco
+ constructor(b: BoletoSimpleFactory)
+ gerarBoleto(vencimento: number, valor: number): Boleto
BoletoSimpleFactory
+ gerarBoleto(vencimento: number, valor: number): Boleto
Boleto
valor: number
juros: number
desconto: number
multa: number
+ constructor(valor: number): void
+ calcularJuros(): number
+ calcularDesconto(): number
+ calcularMulta(): number
Boleto10Dias
+ constructor(valor: number): void
Boleto30Dias
+ constructor(valor: number): void
Boleto60Dias
+ constructor(valor: number): void

Problema

Banco	Dias p/ Vencimento	Juros	Desconto	Multa
Caixa	10	2%	10%	5%
Caixa	30	5%	5%	10%
Caixa	60	10%	0%	20%
Banco do Brasil	10	3%	5%	2%
Banco do Brasil	30	5%	2%	5%
Banco do Brasil	60	10%	0%	15%

Problema


Banco
+ gerarBoleto(vencimento: number, valor: number): Boleto
# criarBoleto(vencimento: number, valor: number): Boleto
function gerarBoleto(vencimento: number, valor: number) Boleto {
return this.criarBoleto(vencimento, valor);
}
Boleto
valor: number
juros: number
desconto: number
multa: number
+ constructor(valor: number): void
+ calcularJuros(): number
+ calcularDesconto(): number
+ calcularMulta(): number
BancoCaixa
criarBoleto(vencimento: number, valor: number): Boleto
BancoDoBrasil
criarBoleto(vencimento: number, valor: number): Boleto
BoletoCaixa10Dias
+ constructor(valor: number): void
BoletoDoBrasil10Dias
+ constructor(valor: number): void
BoletoCaixa30Dias
+ constructor(valor: number): void
BoletoDoBrasil30Dias
+ constructor(valor: number): void
BoletoCaixa60Dias
+ constructor(valor: number): void
BoletoDoBrasil60Dias
+ constructor(valor: number): void

Quando usar?

- Quando uma classe não sabe antecipar qual tipo de objeto deve criar, ou seja, entre várias classes possíveis não é possível prever qual delas deve ser utilizada.
- Quando se precisa que uma classe delegue para suas subclasses especificações dos objetos que instanciam.
- Quando classes delegam responsabilidade a uma dentre várias subclasses auxiliares, se deseja manter o conhecimento nelas e ainda saber qual subclasse foi utilizada em determinado contexto.

Consequências

- O padrão Factory Method elimina o forte acoplamento entre classes concretas
- Criar objetos dentro de uma classe com um método factoryMethod() é sempre mais flexível do que criar um objeto diretamente.
- Os clientes podem achar os métodos de fabrica úteis, e os utilizá los de forma direta.

Exercícios

Sistema de Notificações

Como um administrador de um sistema de mensagens,
eu quero enviar notificações para os usuários,
para que eles possam receber alertas via e-mail ou SMS.

Requisitos:

- Deve haver uma classe Notification com um método send(message: string): void.
- Deve ser possível criar notificações do tipo EmailNotification e SMSNotification.
- Utilize **Factory Method** para instanciar o tipo correto de notificação.

Exercícios

Plataforma de Pagamentos

Como um cliente de uma plataforma de e-commerce,
eu quero poder pagar minhas compras com diferentes métodos de pagamento,
para que eu possa escolher entre cartão de crédito, PayPal ou boleto bancário.

Requisitos:

- Deve haver uma interface Payment com um método processPayment(amount: number): void.
- As implementações devem incluir CreditCardPayment, PayPalPayment e BoletoPayment.
- O sistema deve utilizar o **Factory Method** para criar a instância correta do método de pagamento.

Exercícios

Plataforma de Streaming

Como um desenvolvedor de uma plataforma de streaming,
eu quero que o sistema suporte diferentes tipos de mídia (áudio, vídeo e podcast),
para que os usuários possam consumir conteúdos de diferentes formatos de forma padronizada.

Requisitos:

- Deve haver uma interface Media com métodos play(): void e stop(): void.
- As implementações devem incluir AudioMedia, VideoMedia e PodcastMedia.
- O sistema deve utilizar **Factory Method** para criar dinamicamente a mídia correta com base no tipo recebido.
- Deve haver um tratamento de erro caso o tipo seja inválido.

Código fonte

https://github.com/paeeglee/patterns