Template Method

Padrões de Projeto
Comportamentais I

Prof. Me Jefferson Passerini

Template Method

Também conhecido como: Método Padrão

O Template Method é um padrão de projeto comportamental que define o esqueleto de um algoritmo na superclasse mas deixa as subclasses sobrescreverem etapas específicas do algoritmo sem modificar sua estrutura.

Motivação

- O padrão *Template Method* auxilia na definição de um algoritmo que contém algumas de suas partes definidas por métodos abstratos;
- Subclasses são responsáveis por implementar as partes abstratas deste algoritmo;
- Tais partes poderão ser implementadas de formas distintas, ou seja, cada subclasse irá implementar conforme sua necessidade;
- Deste modo a superclasse posterga algumas implementações para que sejam feitas por suas subclasses;
- Este padrão ajuda na ***reutilização de código*** e no ***controle de como o código deve ser executado***.

Situação Exemplo

- Para exemplificar considere o módulo de pagamentos de software de uma loja de confecções, este módulo foi desenvolvido a alguns anos atrás e possui as seguintes classes

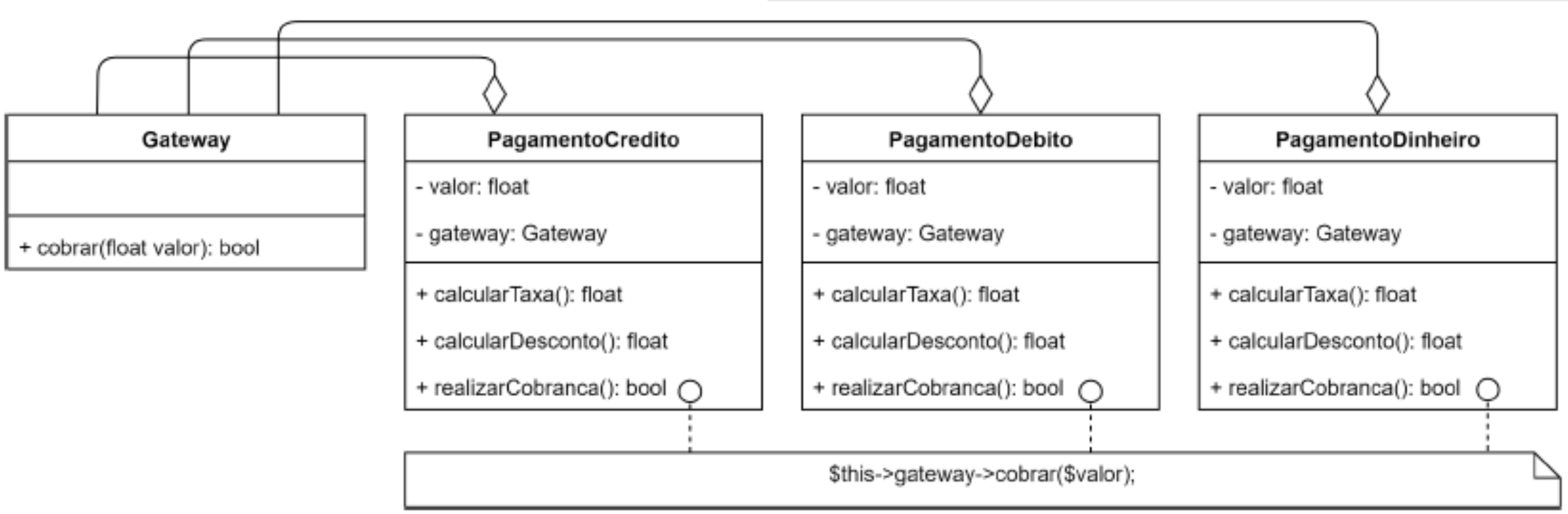


Diagrama de classes do exemplo sem *Template Method*

Regras da loja:

- **Taxa:**
  - ***Crédito*** – 5% do valor;
  - ***Débito*** – acrescentar R$4 fixo.
  - ***Dinheiro*** – sem taxa
- ***Desconto:***
  - ***Crédito*** – 2% somente sob os valores maiores de R$300
  - ***Débito*** – 5% sob o valor
  - ***Dinheiro*** – 10% sob o valor

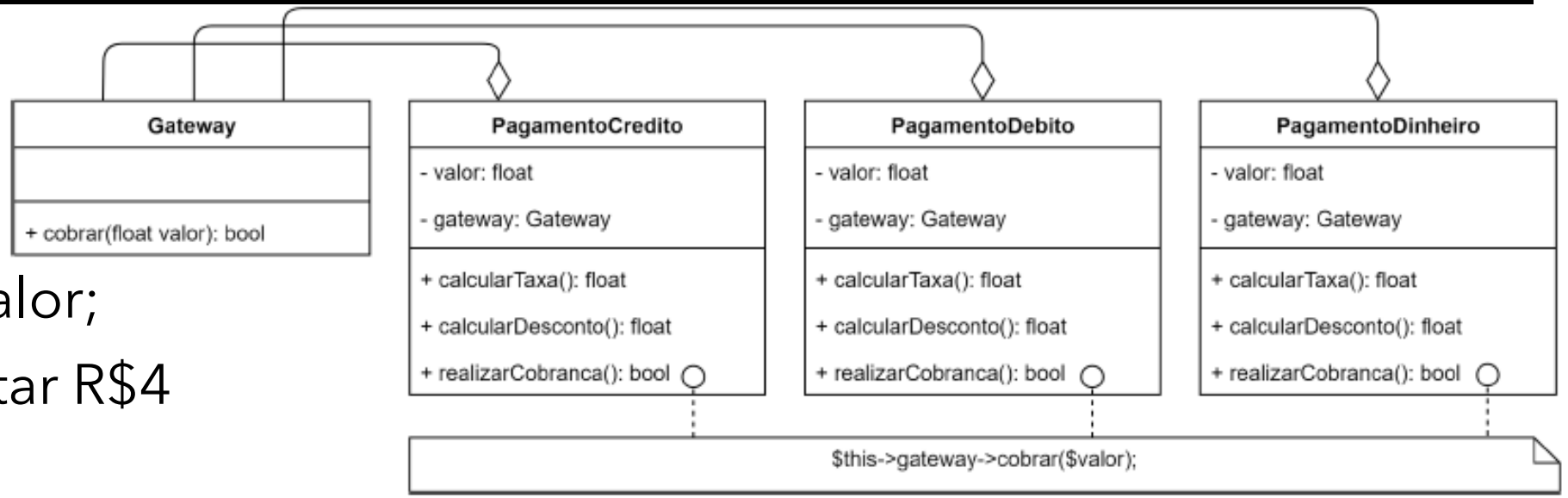


Diagrama de classes do exemplo sem *Template Method*

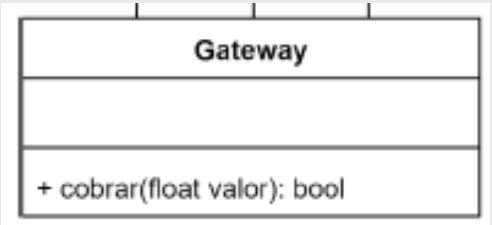
- A taxa é referente à cobrança feita pelo gateway de pagamentos utilizado pelo software da loja para realizar os pagamentos.
- O método ***realizarCobranca()*** presente nas 3 classes é o responsável por delegar a cobrança ao serviço de gateway.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Channels;
using System.Threading.Tasks;

namespace TemplateMethodProblema.GatewayCobranca
{
    8 referências
    public class Gateway
    {
        3 referências
        public Boolean cobrar(double valor)
        {
            bool[] respostas = {true, false};
            //imprime o valor cobrado
            Console.WriteLine("Valor Cobrado: R$"+valor);
            //retorna true ou false para confirmação da
            //cobrança por sorteio aleatorio.
            return respostas[new Random().Next(0,1)];
        }
    }
}
```

Java

```
5   package gatewayPagamento;
6
7   import java.util.Random;
8   /**
9    *
10   * @author jeffe
11   */
12  public class Gateway {
13
14      public Boolean cobrar(double valor){
15          Boolean[] respostas = {true, false};
16          //imprime valor cobrado
17          System.out.println("Valor Cobrado: R$"+valor);
18          return respostas[new Random().nextInt(2)];
19      }
20
21  }
```



Essa classe é responsável para simular o Gateway de pagamentos o método cobrar() retorna *true* ou *false* de forma aleatória

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TemplateMethodProblema.GatewayCobranca;

namespace TemplateMethodProblema.Pagamento
{
    1 referência
    public class PagamentoCredito
    {
        5 referências
        private double valor { get; set; }
        2 referências
        private Gateway gateway { get; set; }
        0 referências
        public PagamentoCredito(double valor, Gateway gateway)
        {
            this.valor = valor;
            this.gateway = gateway;
        }

        1 referência
        public double calcularTaxa()
        {
            return this.valor * 0.05;
        }
        1 referência
        public double calcularDesconto()
        {
            if (this.valor > 300)
                return this.valor * 0.02;
            else return 0;
        }
        0 referências
        public Boolean realizaCobranca()
        {
            double valorPago =
               this.valor+this.calcularTaxa()
               -this.calcularDesconto();
            return this.gateway.cobrar(valorPago);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TemplateMethodProblema.GatewayCobranca;

namespace TemplateMethodProblema.Pagamento
{
    1 referência
    public class PagamentoDebito
    {
        3 referências
        private double valor { get; set; }
        2 referências
        private Gateway gateway { get; set; }
        0 referências
        public PagamentoDebito(double valor, Gateway gateway)
        {
            this.valor = valor;
            this.gateway = gateway;
        }

        1 referência
        public double calcularTaxa()
        {
            return 4;
        }
        1 referência
        public double calcularDesconto()
        {
            return this.valor * 0.05;
        }
        0 referências
        public Boolean realizaCobranca()
        {
            double valorPago =
               this.valor + this.calcularTaxa()
               - this.calcularDesconto();
            return this.gateway.cobrar(valorPago);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TemplateMethodProblema.GatewayCobranca;

namespace TemplateMethodProblema.Pagamento
{
    1 referência
    public class PagamentoDinheiro
    {
        3 referências
        private double valor { get; set; }
        2 referências
        private Gateway gateway { get; set; }
        0 referências
        public PagamentoDinheiro(double valor, Gateway gateway)
        {
            this.valor = valor;
            this.gateway = gateway;
        }

        1 referência
        public double calcularTaxa()
        {
            return 0;
        }
        1 referência
        public double calcularDesconto()
        {
            return this.valor * 0.1;
        }
        0 referências
        public Boolean realizaCobranca()
        {
            double valorPago =
               this.valor + this.calcularTaxa()
               - this.calcularDesconto();
            return this.gateway.cobrar(valorPago);
        }
    }
}
```

Métodos Iguais – Repetição de código

```java
package pagamento;

import gatewayPagamento.Gateway;

/**...4 lines */
public class PagamentoCredito {
    private double valor;
    private Gateway gateway;

    public PagamentoCredito(double valor, Gateway gateway) {
        this.valor = valor;
        this.gateway = gateway;
    }

    public double calcularTaxa(){
        return this.valor*0.05;
    }

    public double calcularDesconto(){
        if(this.valor>300)
            return this.valor*0.05;
        else return 0;
    }

    public Boolean realizarCobranca(){
        double valorPago = this.valor+
                this.calcularTaxa()-
                this.calcularDesconto();
        return this.gateway.cobrar(valorPago);
    }
}
```

```java
package pagamento;

import gatewayPagamento.Gateway;

/**...4 lines */
public class PagamentoDebito {
    private double valor;
    private Gateway gateway;

    public PagamentoDebito(double valor, Gateway gateway) {
        this.valor = valor;
        this.gateway = gateway;
    }

    public double calcularTaxa(){
        return 4;
    }

    public double calcularDesconto(){
            return this.valor*0.05;
    }

    public Boolean realizarCobranca(){
        double valorPago = this.valor+
                this.calcularTaxa()-
                this.calcularDesconto();
        return this.gateway.cobrar(valorPago);
    }
}
```

```java
package pagamento;

import gatewayPagamento.Gateway;

/**...4 lines */
public class PagamentoDinheiro {
    private double valor;
    private Gateway gateway;

    public PagamentoDinheiro(double valor, Gateway gateway) {
        this.valor = valor;
        this.gateway = gateway;
    }

    public double calcularTaxa(){
        return 0;
    }

    public double calcularDesconto(){
            return this.valor*0.1;
    }

    public Boolean realizarCobranca(){
        double valorPago = this.valor+
                this.calcularTaxa()-
                this.calcularDesconto();
        return this.gateway.cobrar(valorPago);
    }
}
```

Métodos Iguais - Repetição de código

Padrões de Projetos Comportamentais – *Template Method*

C#

```csharp
using TemplateMethodProblema.GatewayCobranca;
using TemplateMethodProblema.Pagamento;

double valor = 100;
Gateway gateway = new Gateway();

Console.WriteLine("Crédito");
PagamentoCredito pgCred = new PagamentoCredito(valor, gateway);
pgCred.realizaCobranca();
Console.WriteLine("Débito");
PagamentoDebito pgDeb = new PagamentoDebito(valor, gateway);
pgDeb.realizaCobranca();
Console.WriteLine("Dinheiro");
PagamentoDinheiro pgDin = new PagamentoDinheiro(valor, gateway);
pgDin.realizaCobranca();
```

```
Console de Depuração do Mic
Crédito
Valor Cobrado: R$105
Débito
Valor Cobrado: R$99
Dinheiro
Valor Cobrado: R$90
```

Java

```java
package templatemethodproblema;

import gatewayPagamento.Gateway;
import pagamento.PagamentoCredito;
import pagamento.PagamentoDebito;
import pagamento.PagamentoDinheiro;
/**...4 lines */
public class TemplateMethodProblema {

    /**...3 lines */
    public static void main(String[] args) {
        double valor = 100;
        Gateway gateway = new Gateway();
        System.out.println("Crédito");
        PagamentoCredito pgCred = new PagamentoCredito(valor, gateway);
        pgCred.realizarCobranca();
        System.out.println("Débito");
        PagamentoDebito pgDeb = new PagamentoDebito(valor, gateway);
        pgDeb.realizarCobranca();
        System.out.println("Dinheiro");
        PagamentoDinheiro pgDin = new PagamentoDinheiro(valor, gateway);
        pgDin.realizarCobranca();
    }
}
```

```
Crédito
Valor Cobrado: R$105.0
Débito
Valor Cobrado: R$99.0
Dinheiro
Valor Cobrado: R$90.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Situação Exemplo

- O dono da loja de confecções está modernizando a loja e deseja aceitar novas formas de pagamento no futuro;
- Nossa tarefa é refatorar o módulo de pagamentos de modo que ele seja apto a aceitar novas formas de pagamento de maneira segura, sem afetar as formas de pagamentos existentes, e minimizando as chances de bugs.

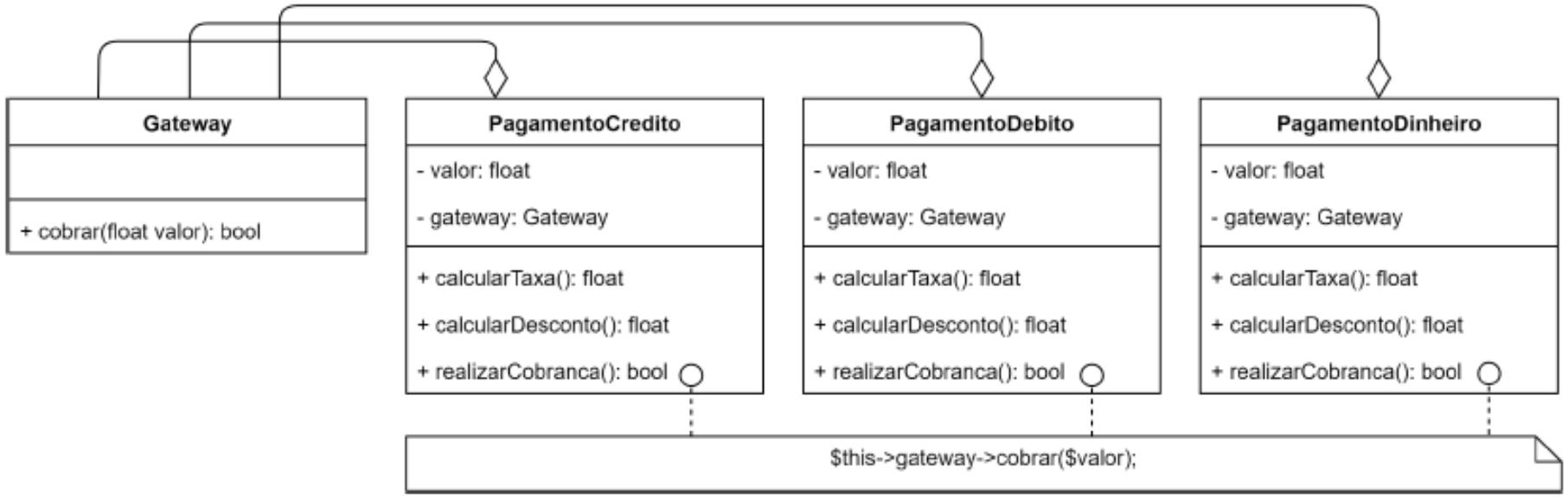


Diagrama de classes do exemplo sem *Template Method*

Situação Exemplo

- É possível encontrar:
  - Todas elas têm o método realizarCobranca() idêntico.
  - Todas possuem método calcularTaxa(), mas diferentes entre si.
  - Todos possuem método calcular desconto(), mas diferentes entre si
  - Todas mantêm atributos de valor e gateway.

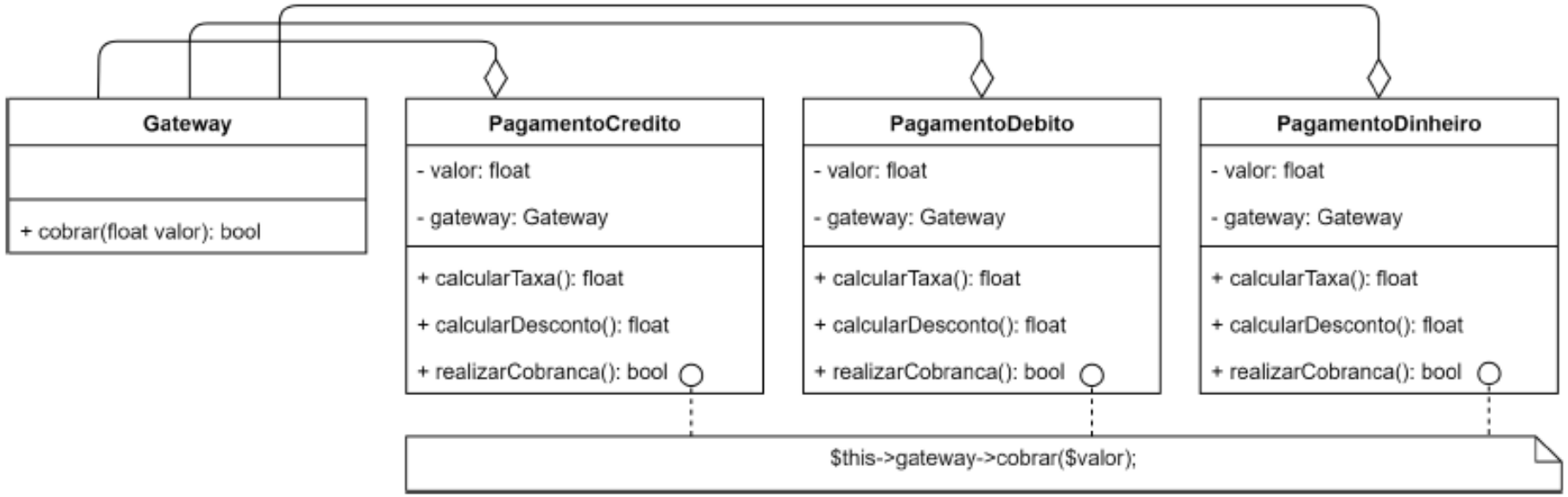


Diagrama de classes do exemplo sem *Template Method*

Situação Exemplo

- O método realizarCobranca() é quem dita como a cobrança será feita, é nele onde o algoritmo está implementado.
- Ele utiliza os métodos calcularTaxa() e calcular desconto() que são a única variação entre uma classe e outra.

Refatorando

- O método realizarCobranca() será transformada em uma classe Pagamento que será o nosso Template Method.
- E migrar tudo que é comum entre as classes para uma superclasse Pagamento.

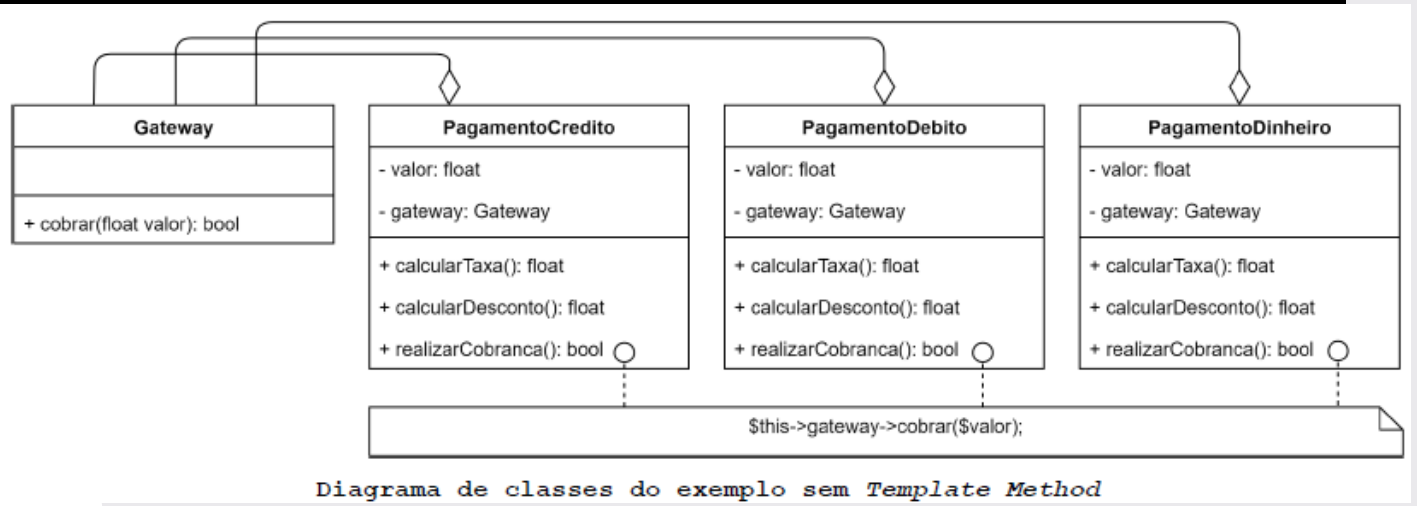


Diagrama de classes do exemplo sem *Template Method*

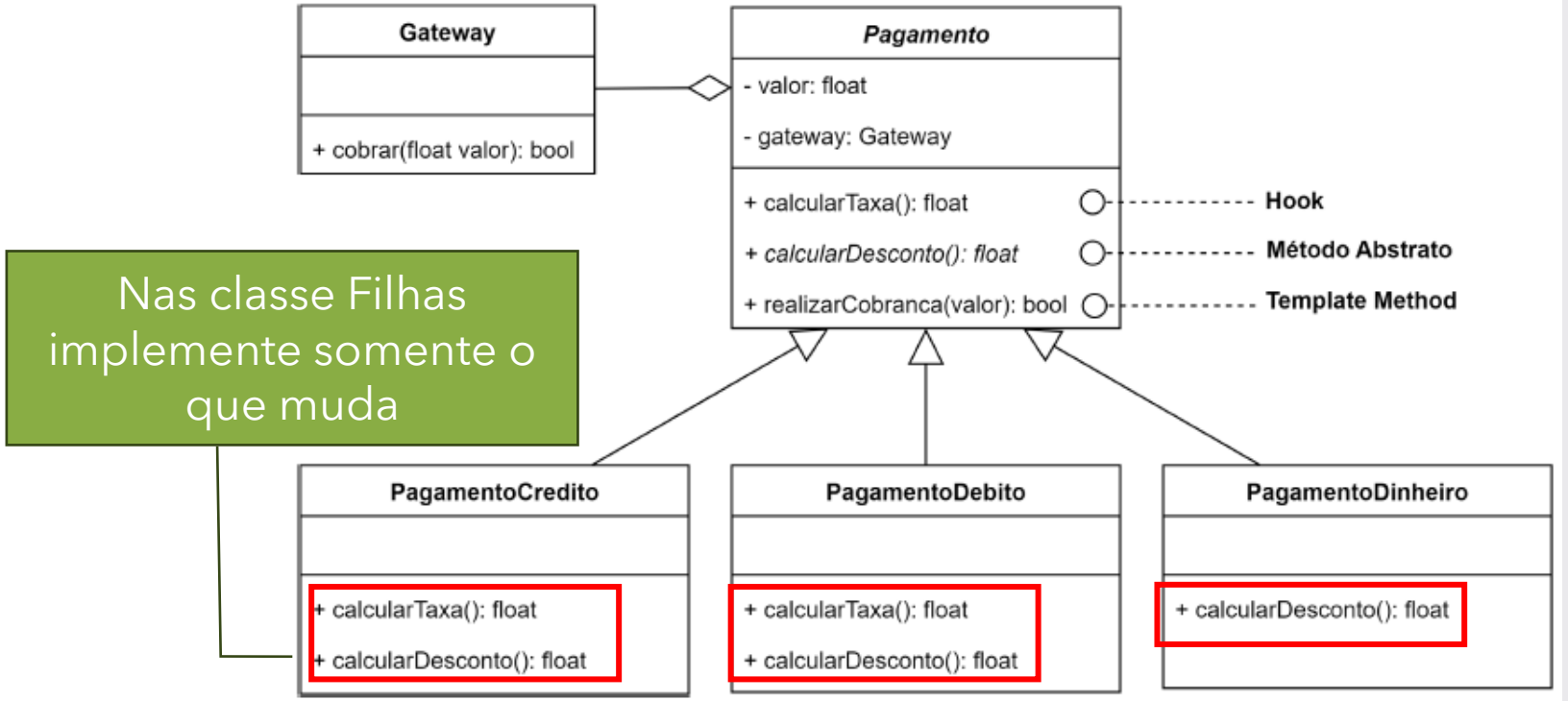


Diagrama de classes do exemplo com *Template Method*

Refatorando

- Os atributos **valor** e **gateway** ficam agora na superclasse Pagamento, assim as subclasses já terão esses recursos disponíveis

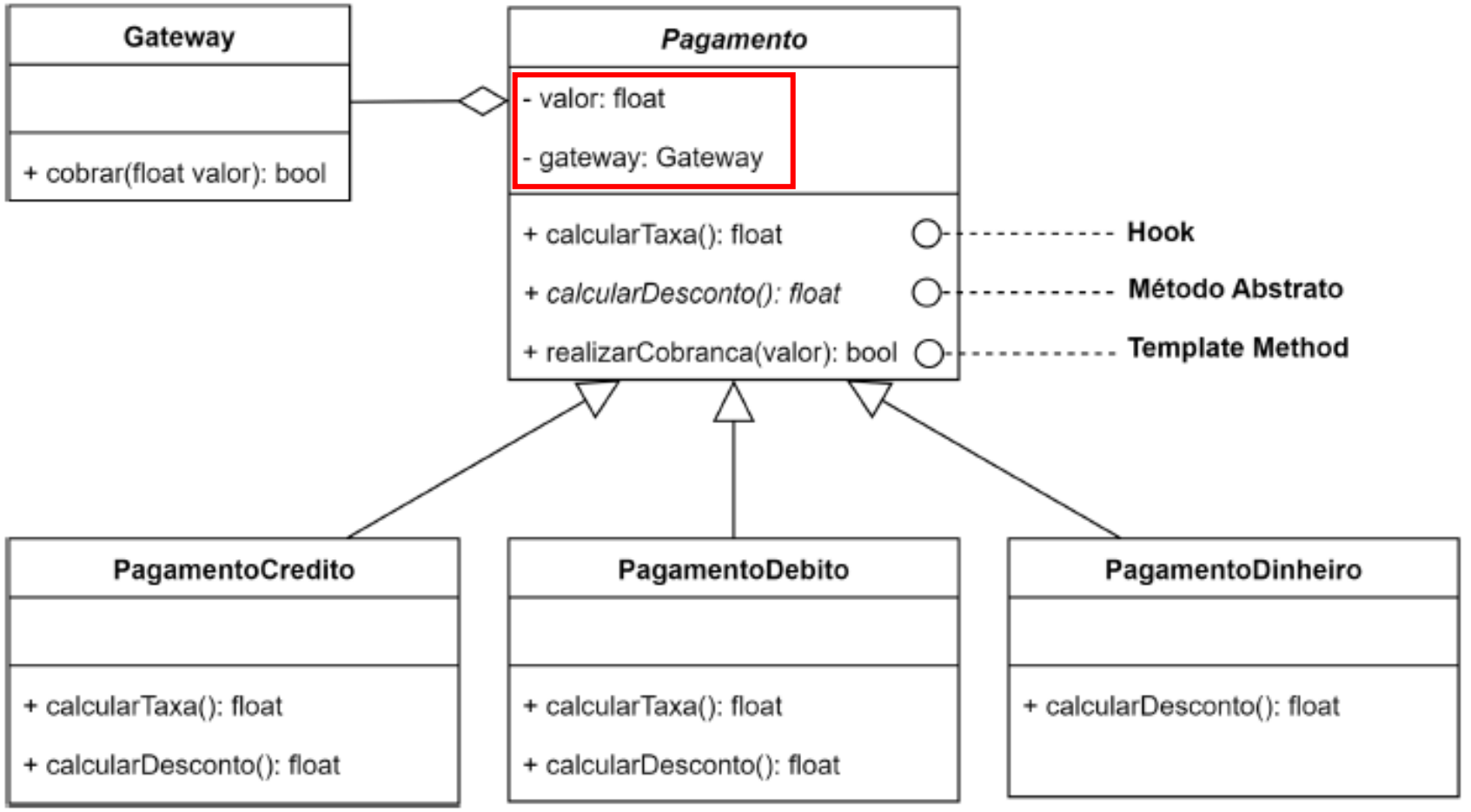


Diagrama de classes do exemplo com *Template Method*

Refatorando

- método realizarCobranca() deve ser declarado com final, isso garante que as subclasses não conseguirão subreescreve-la.
- Assim a cobrança terá suas regras definidas na superclasse.
- Será o novo **Template Method.**

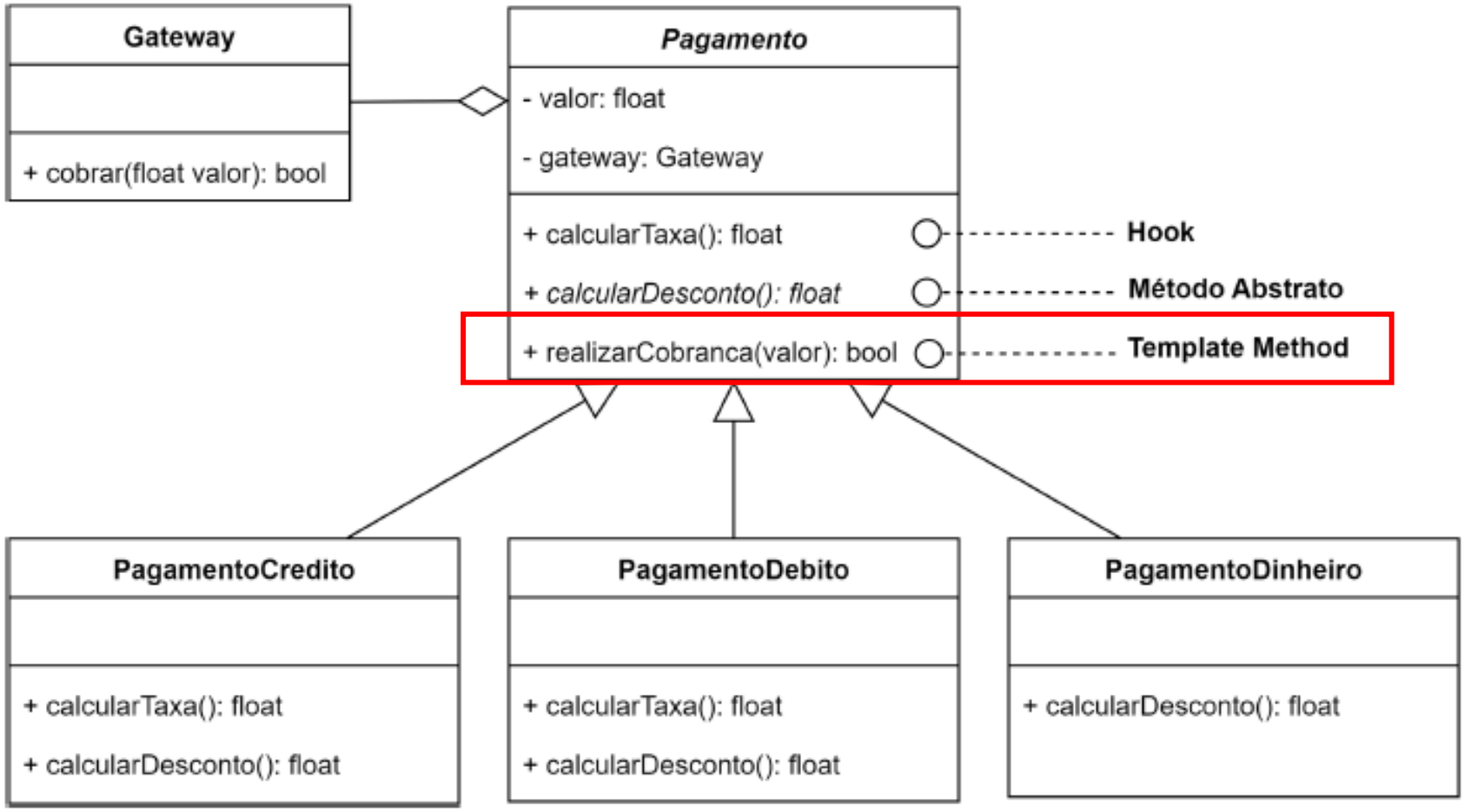


Diagrama de classes do exemplo com *Template Method*

Refatorando

- método calcularDesconto() foi declarado como abstrato, deste modo é responsabilidade das subclasses implementa-lo com suas regras especificas.

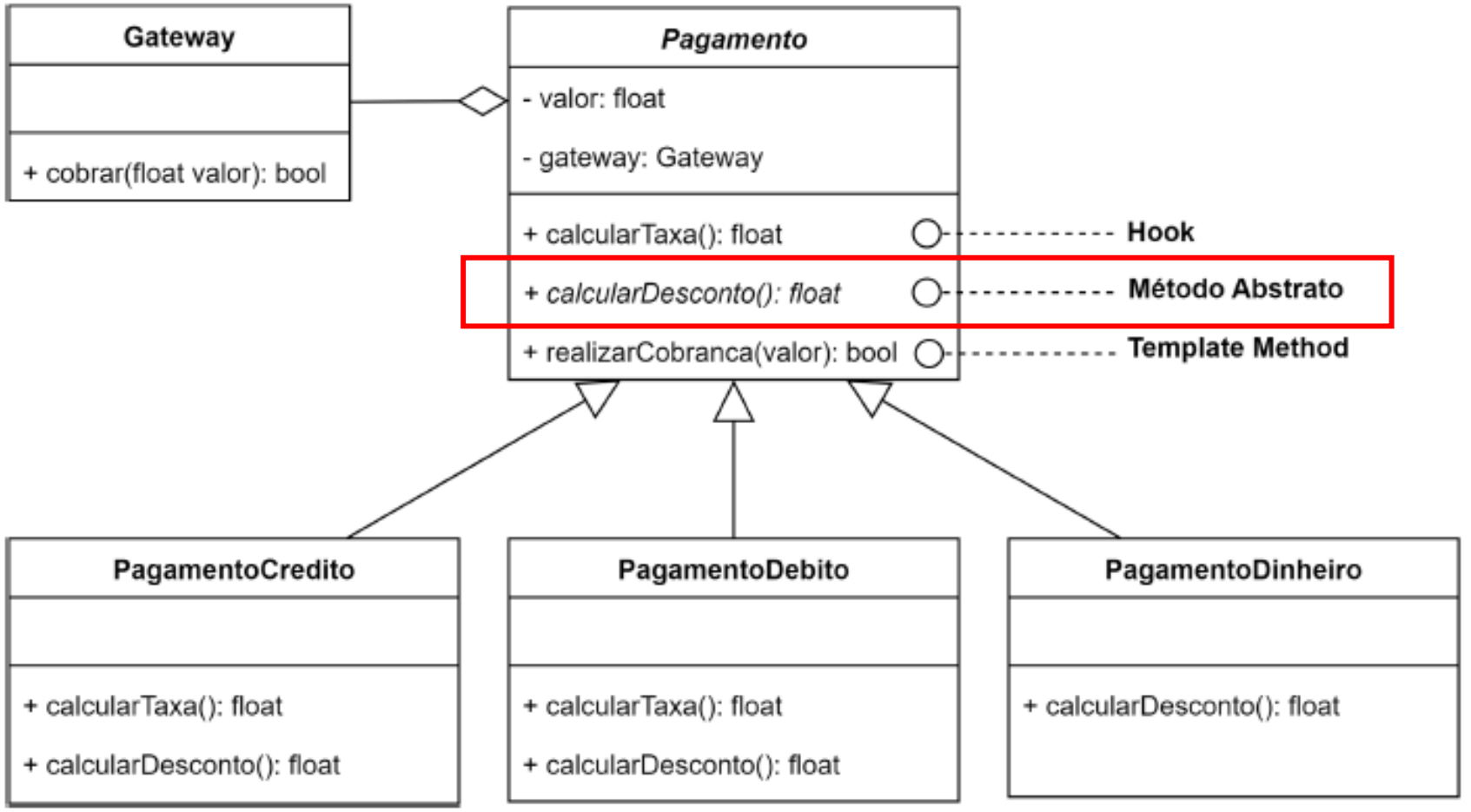


Diagrama de classes do exemplo com *Template Method*

Refatorando

- método calcularTaxa() é um Hook (gancho).
- Trata-se de um método implementado na classe abstrata mas recebe apenas uma implementação vazia ou mínima como padrão

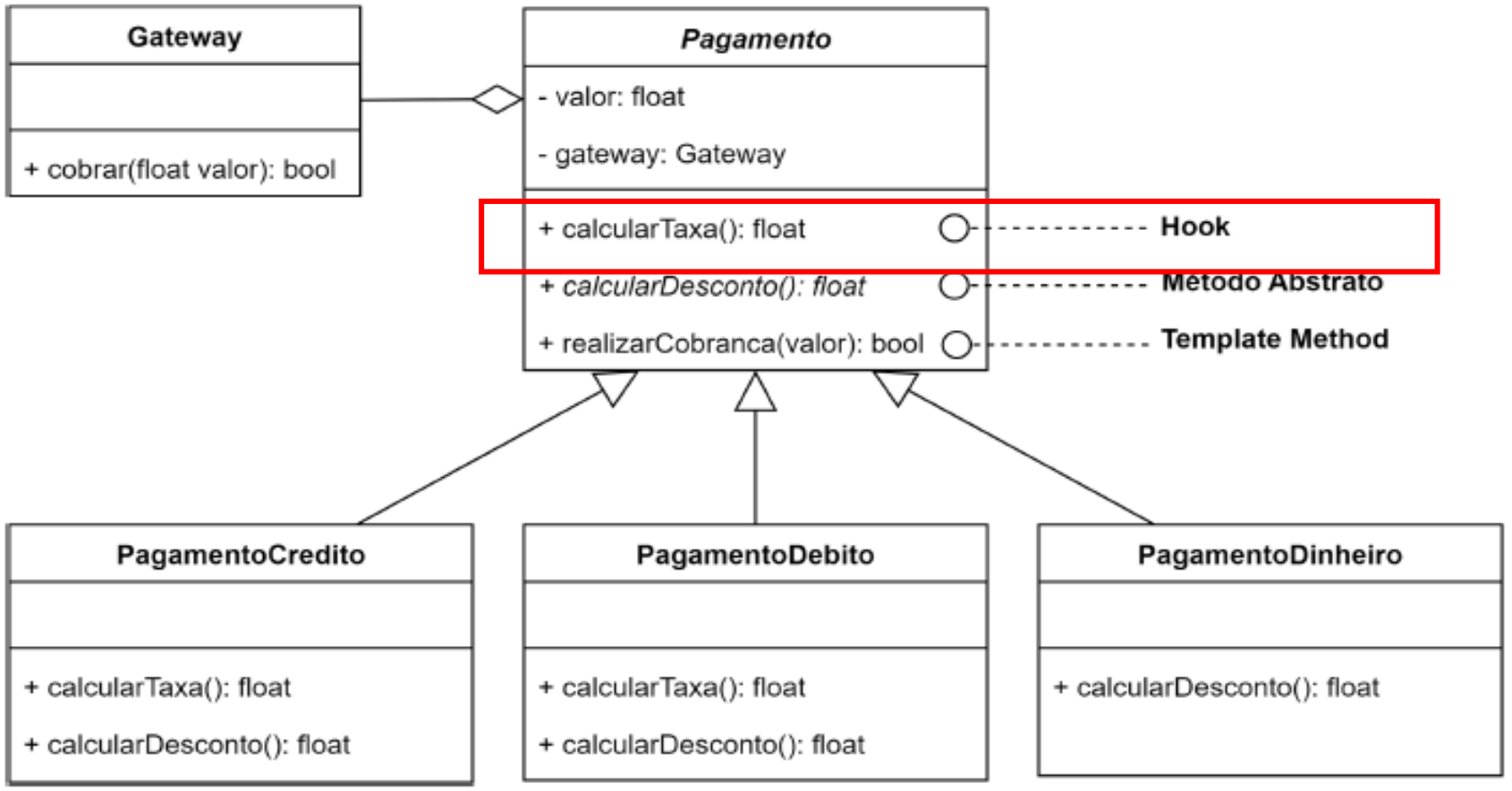


Diagrama de classes do exemplo com *Template Method*

Aqui no exemplo é útil devido ao fato que a classe PagamentoDinheiro não ter a incidência de cobrança de taxa, assim ela pode utilizar a implementação padrão do método que retorna 0. Já as classes PagamentoCredito e PagamentoDebito deverão sobrescreve-lo para suas regras.

Refatorando

- método calcularTaxa() é um Hook (gancho).
- Trata-se de um método implementado na classe abstrata mas recebe apenas uma implementação vazia ou mínima como padrão

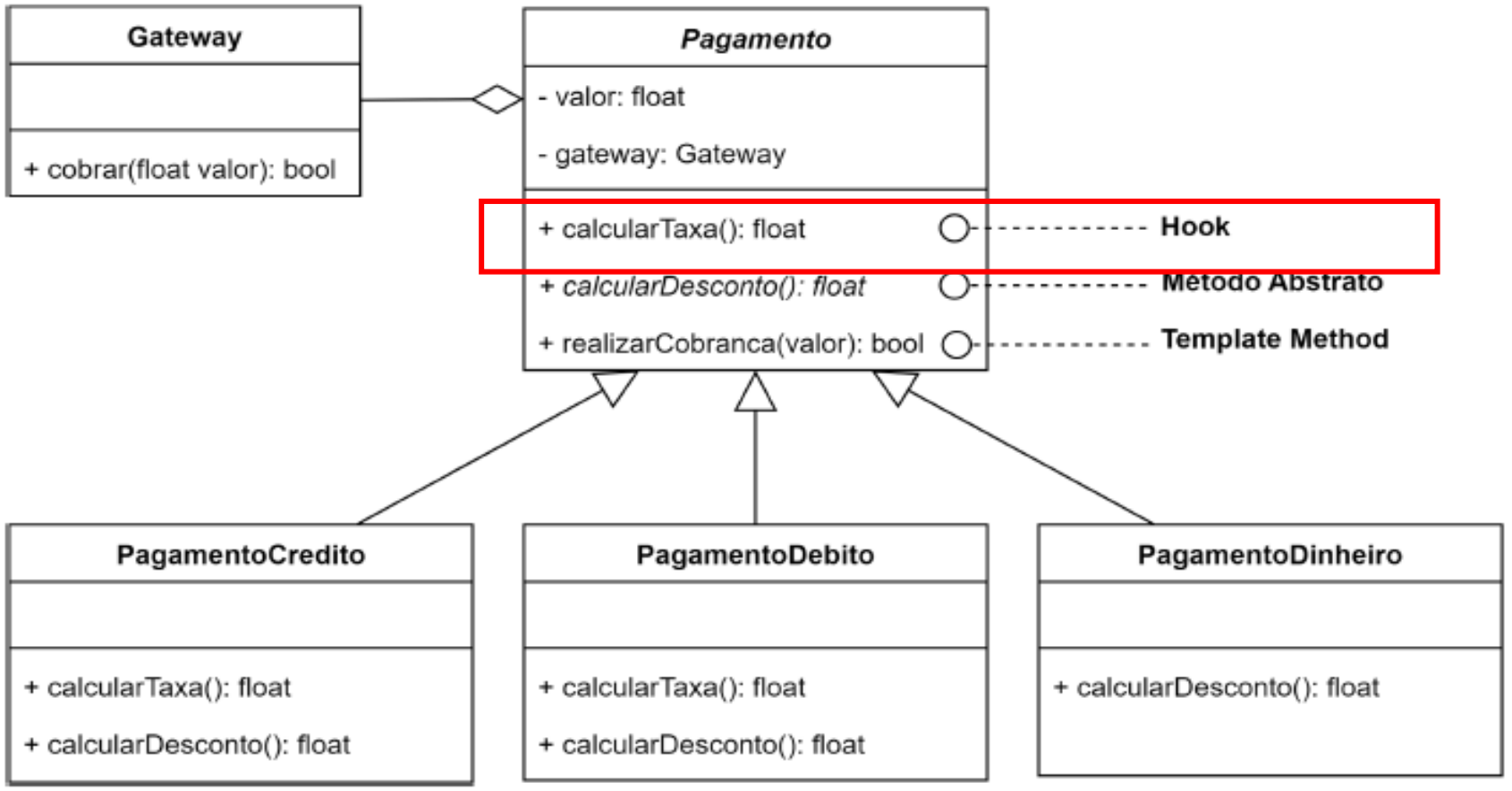


Diagrama de classes do exemplo com *Template Method*

Aqui no exemplo é útil devido ao fato que a classe PagamentoDinheiro não ter a incidência de cobrança de taxa, assim ela pode utilizar a implementação padrão do método que retorna 0. Já as classes PagamentoCredito e PagamentoDebito deverão sobrescreve-lo para suas regras.

Refatorando

- Vamos claramente uma **inversão de dependência** onde a superclasse depende das subclasses.
- Onde no natural seria o contrário.
- Quem dita as regras a respeito de como a cobrança será feita é o método realizaCobranca() da superclasse.
- Cabe as subclasses apenas completar as peças do quebra-cabeça.

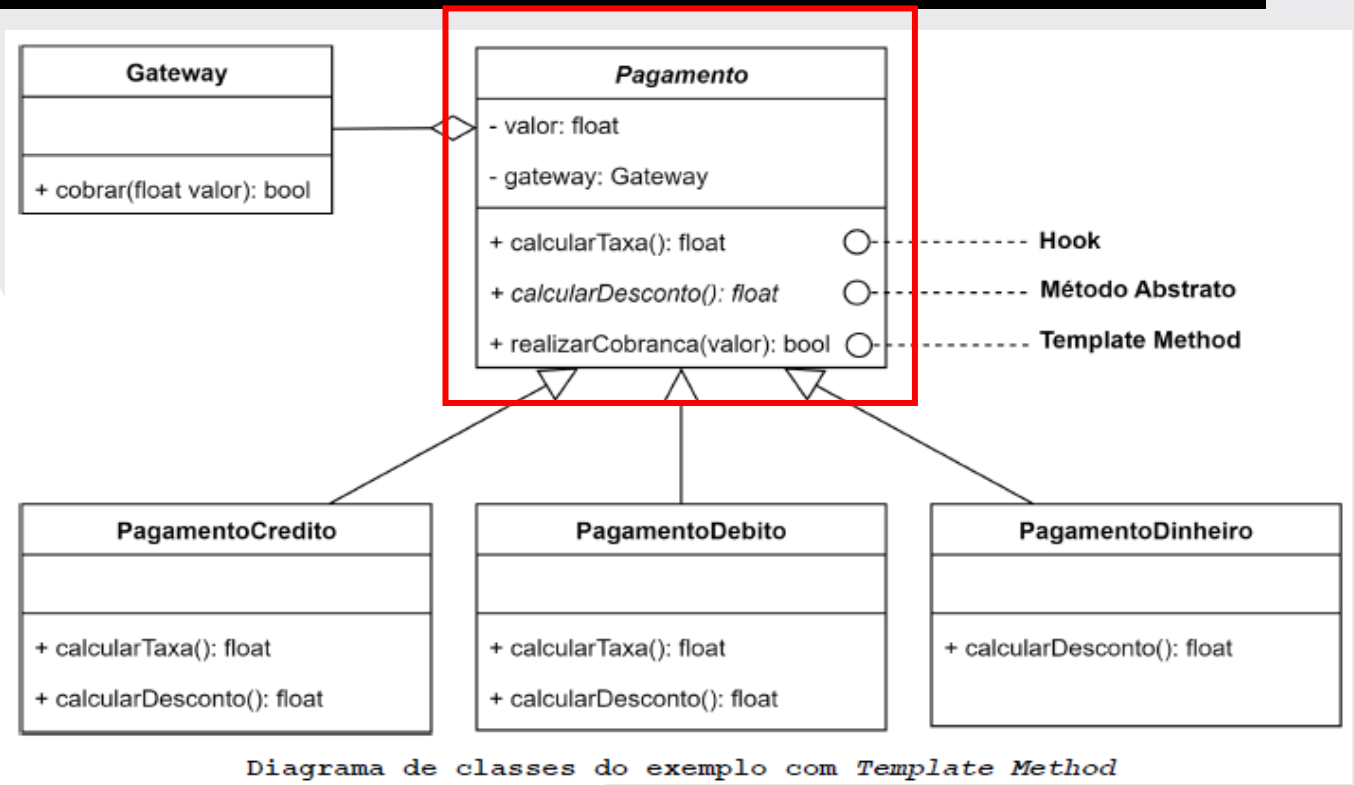


Diagrama de classes do exemplo com *Template Method*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TemplateMethodSolucao.GatewayCobranca;

namespace TemplateMethodSolucao.MetodoPagamento
{
    3 referências
    public abstract class Pagamento
    {
        9 referências
        protected double valor { get; set; }
        4 referências
        protected Gateway gateway { get; set; }

        //hook method
        3 referências
        public virtual double calcularTaxa()
        {
            return 0;
        }
        //implementado na subclasse
        4 referências
        public abstract double calcularDesconto();
        3 referências
        public Boolean realizaCobranca()
        {
            double valorPago =
               this.valor + this.calcularTaxa()
               - this.calcularDesconto();
            return this.gateway.cobrar(valorPago);
        }
    }
}
```

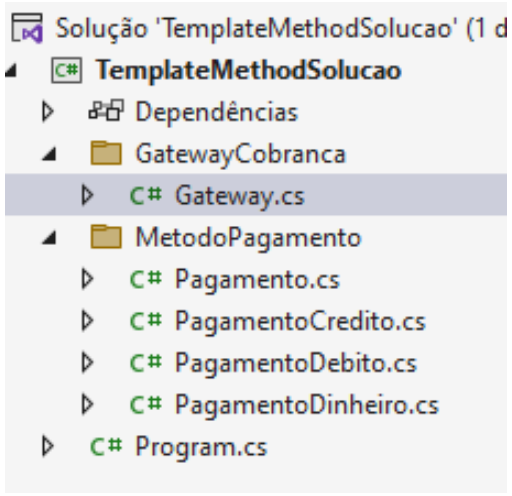
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TemplateMethodSolucao.GatewayCobranca;

namespace TemplateMethodSolucao.MetodoPagamento
{
    3 referências
    public class PagamentoCredito : Pagamento
    {
        1 referência
        public PagamentoCredito(double valor, Gateway gateway)
        {
            base.valor = valor;
            base.gateway = gateway;
        }

        2 referências
        public override double calcularTaxa()
        {
            return this.valor * 0.05;
        }

        2 referências
        public override double calcularDesconto()
        {
            if (this.valor > 300)
                return this.valor * 0.02;
            else return 0;
        }
    }
}
```

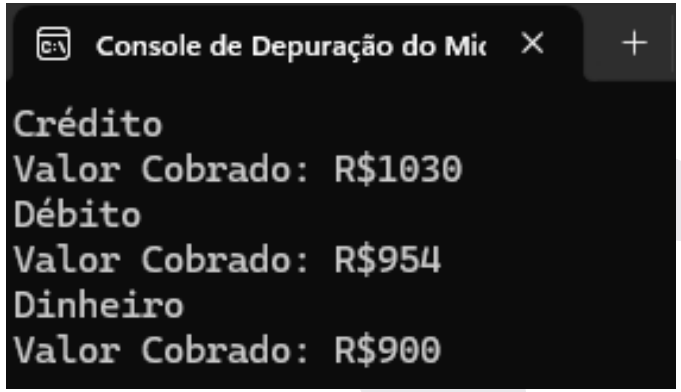
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TemplateMethodSolucao.GatewayCobranca;

namespace TemplateMethodSolucao.MetodoPagamento
{
    3 referências
    public class PagamentoCredito : Pagamento
    {
        1 referência
        public PagamentoCredito(double valor, Gateway gateway)
        {
            base.valor = valor;
            base.gateway = gateway;
        }

        2 referências
        public override double calcularTaxa()
        {
            return this.valor * 0.05;
        }

        2 referências
        public override double calcularDesconto()
        {
            if (this.valor > 300)
                return this.valor * 0.02;
            else return 0;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TemplateMethodSolucao.GatewayCobranca;

namespace TemplateMethodSolucao.MetodoPagamento
{
    3 referências
    public class PagamentoDinheiro : Pagamento
    {
        1 referência
        public PagamentoDinheiro(double valor, Gateway gateway)
        {
            base.valor = valor;
            base.gateway = gateway;
        }

        2 referências
        public override double calcularDesconto()
        {
            return this.valor * 0.1;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Channels;
using System.Threading.Tasks;

namespace TemplateMethodSolucao.GatewayCobranca
{
    6 referências
    public class Gateway
    {
        1 referência
        public Boolean cobrar(double valor)
        {
            bool[] respostas = {true, false};
            //imprime o valor cobrado
            Console.WriteLine("Valor Cobrado: R$"+valor);
            //retorna true ou false para confirmação da
            //cobrança por sorteio aleatorio.
            return respostas[new Random().Next(0,1)];
        }
    }
}
```

```
using TemplateMethodSolucao.GatewayCobranca;
using TemplateMethodSolucao.MetodoPagamento;

double valor = 1000;
Gateway gateway = new Gateway();

Console.WriteLine("Crédito");
PagamentoCredito pgCred =
    new PagamentoCredito(valor,gateway);
pgCred.realizaCobranca();

Console.WriteLine("Débito");
PagamentoDebito pgDeb =
    new PagamentoDebito(valor, gateway);
pgDeb.realizaCobranca();

Console.WriteLine("Dinheiro");
PagamentoDinheiro pgDin =
    new PagamentoDinheiro(valor, gateway);
pgDin.realizaCobranca();
```





```java
package metodoPagamento;

import gatewayPagamento.Gateway;

/**...4 lines */
public abstract class Pagamento {
    protected double valor;
    protected Gateway gateway;
    //metodo hook
    public double calcularTaxa(){
        return 0;
    }
    //metodo abstrato
    public abstract double calcularDesconto();

    public Boolean realizarCobranca(){
        double valorPago = this.valor+
                this.calcularTaxa()-
                this.calcularDesconto();
        return this.gateway.cobrar(valorPago);
    }

}
```

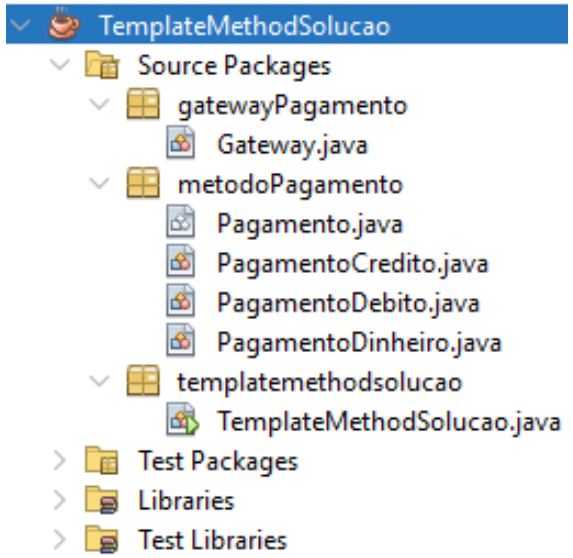
```java
package metodoPagamento;

import gatewayPagamento.Gateway;

/**...4 lines */
public class PagamentoCredito extends Pagamento {

    public PagamentoCredito(double valor, Gateway gateway) {
        super.valor = valor;
        super.gateway = gateway;
    }

    @Override
    public double calcularTaxa(){
        return super.valor*0.05;
    }

    @Override
    public double calcularDesconto(){
        if(super.valor>300)
            return super.valor*0.02;
        else return 0;
    }
}
```

```java
package metodoPagamento;

import gatewayPagamento.Gateway;

/**...4 lines */
public class PagamentoDebito extends Pagamento {

    public PagamentoDebito(double valor, Gateway gateway) {
        super.valor = valor;
        super.gateway = gateway;
    }

    @Override
    public double calcularTaxa(){
        return 4;
    }

    @Override
    public double calcularDesconto(){
        return super.valor*0.05;
    }

}
```

```java
package metodoPagamento;

import gatewayPagamento.Gateway;

/**...4 lines */
public class PagamentoDinheiro extends Pagamento {

    public PagamentoDinheiro(double valor, Gateway gateway) {
        super.valor = valor;
        super.gateway = gateway;
    }

    @Override
    public double calcularDesconto(){
            return super.valor*0.1;
    }
}
```

```java
package gatewayPagamento;

import java.util.Random;
/**...14 lines */
public class Gateway {

    public Boolean cobrar(double valor){
        Boolean[] respostas = {true, false};
        //imprime valor cobrado
        System.out.println("Valor Cobrado: R$"+valor);
        return respostas[new Random().nextInt(2)];
    }

}
```

```java
import gatewayPagamento.Gateway;
import metodoPagamento.PagamentoCredito;
import metodoPagamento.PagamentoDebito;
import metodoPagamento.PagamentoDinheiro;

/**
 *
 * @author jeffe
 */
public class TemplateMethodSolucao {

    /**...3 lines */
    public static void main(String[] args) {
        double valor = 1000;
        Gateway gateway = new Gateway();

        System.out.println("Crédito");
        PagamentoCredito pgCred =
            new PagamentoCredito(valor, gateway);
        pgCred.realizarCobranca();

        System.out.println("Débito");
        PagamentoDebito pgDeb =
            new PagamentoDebito(valor, gateway);
        pgDeb.realizarCobranca();

        System.out.println("Dinheiro");
        PagamentoDinheiro pgDin =
            new PagamentoDinheiro(valor, gateway);
        pgDin.realizarCobranca();
    }

}
```




```
run:
Crédito
Valor Cobrado: R$1030.0
Débito
Valor Cobrado: R$954.0
Dinheiro
Valor Cobrado: R$900.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Refatorando

- Na nova implementação não existe código repetido, as subclasses complementam a superclasse com os comportamentos que variam;
- Isso causa uma inversão de dependência que diz: "***Abstrações não devem ser baseadas em detalhes. Detalhes devem ser baseados em abstrações***";
- É exatamente o que estamos fazendo, pois os detalhes de cada tipo de pagamento dependem da classe abstrata Pagamento e não o contrário.

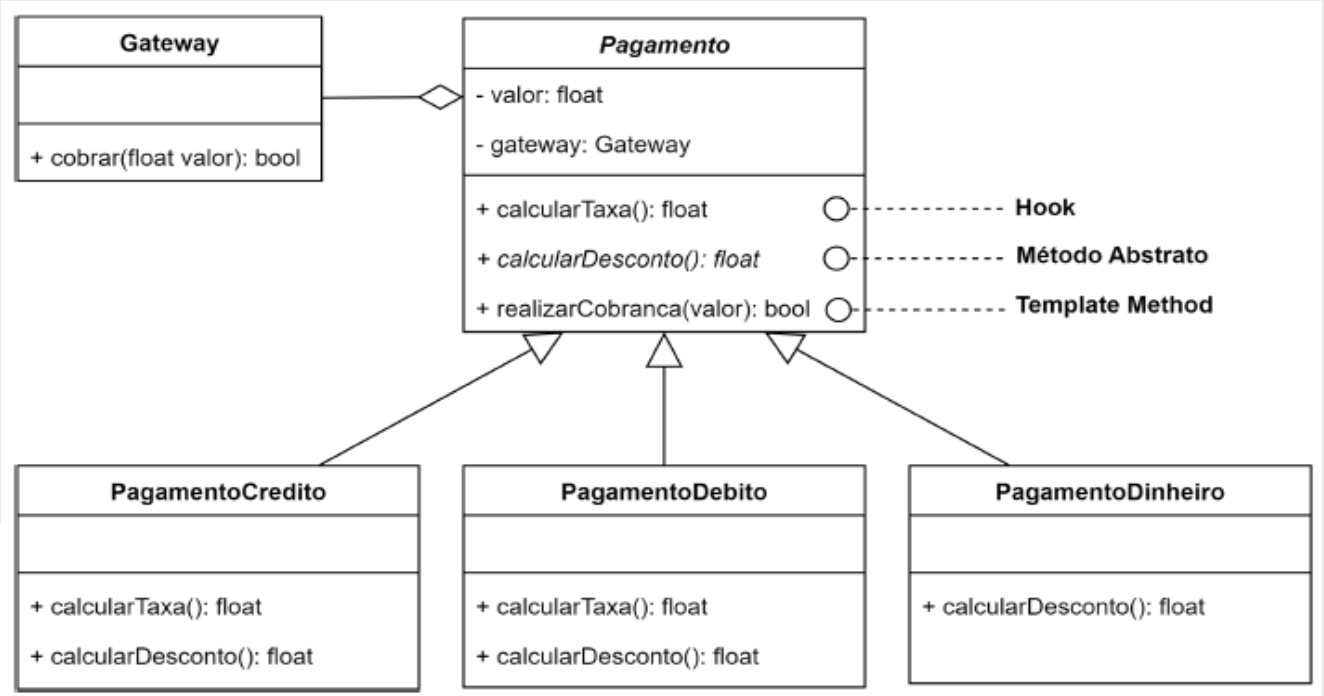


Diagrama de classes do exemplo com *Template Method*

Refatorando

- Se no futuro a loja decidir adotar um novo meio de pagamento, como via smartphone por exemplo;
- Basta criar uma nova classe PagamentoSmartphone e estender a classe Pagamento.
- Deste modo não será necessário modificar a classe Pagamento, seguindo o princípio: **"aberta para extensão e fechada para mudanças"**

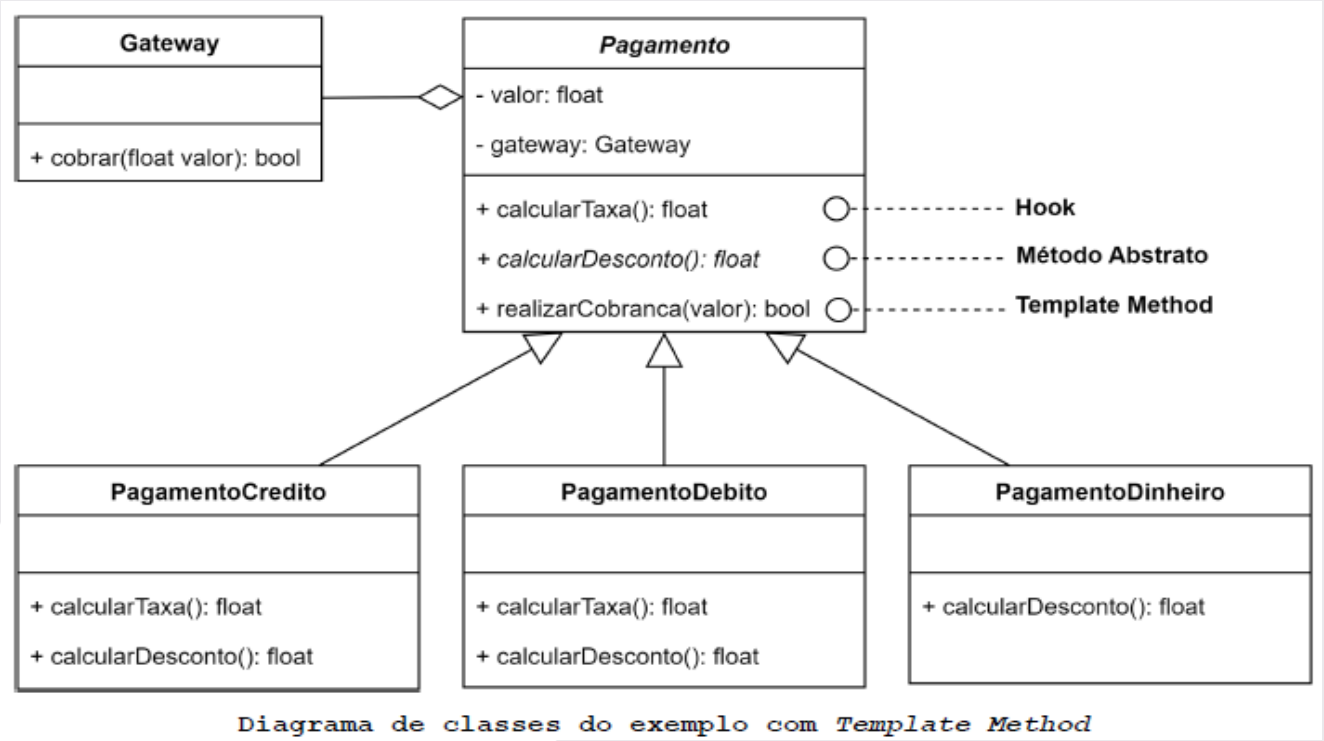


Diagrama de classes do exemplo com *Template Method*

Aplicabilidade

- Para implementar partes invariantes de um algoritmo apenas uma vez, deixando as subclasses apenas a implementação daquilo que pode variar;

- Controlar extensões de subclasses, sabendo o que as subclasses devem implementar e até onde devem implementar;

- Evitar duplicação de código entre classes comuns.

Componentes

- **ClasseAbstrata:** Superclasse abstrata que contém os métodos concretos e abstratos que serão comuns a todas suas subclasses. Implementa o templateMethod() que define o esqueleto de um algoritmo.

- **ClasseConcreta:** Classes que herdam os métodos concretos de ClasseAbstrata e implementam os métodos abstratos conforme suas especificidades.

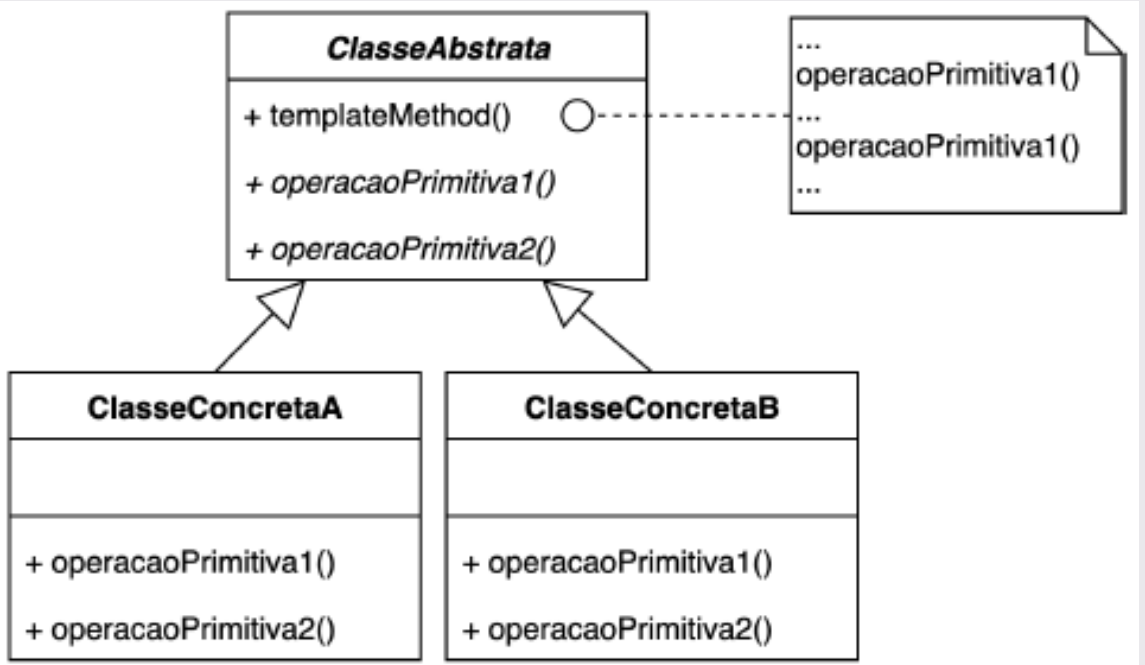


Diagrama de Classes

Consequências

- Os Template Methods são uma técnica fundamental para reutilização de código; São particularmente importantes em bibliotecas de classes, pois são meios para definir o comportamento comum nas classes das bibliotecas;
- Proporcionam a inversão de dependência. Isso se refere a como uma classe pai chama as operações de uma subclasse e não o contrário.
- Permitem controlar a sequência da execução de métodos das subclasses.
- Possibilitam ter pontos que chamam código ainda não implementado.

Consequências

- Podem chamar os seguintes tipos de operações:

  - Métodos concretos: implementados na própria classe abstrata onde o Template Method se encontra;
  - Métodos abstratos: implementados nas subclasses;
  - Operações primitivas e funções da linguagem;
  - Outros Template Methods
  - Hooks

Relações com outros padrões

- O **Factory Method** é uma especialização do **Template Method**. Ao mesmo tempo, o Factory Method pode servir como uma etapa em um Template Method grande.

- O **Template Method** é baseado em herança: ele permite que você altere partes de um algoritmo ao estender essas partes em subclasses. O **Strategy** é baseado em composição: você pode alterar partes do comportamento de um objeto ao suprir ele como diferentes estratégias que correspondem a aquele comportamento. O Template Method funciona a nível de classe, então é estático. O Strategy trabalha a nível de objeto, permitindo que você troque os comportamentos durante a execução.