Projetos de padrões: Chain of responsability(GOF) e Information Expert(GRASP)

Alunos: Alysson e Wellington
Professor: Rogério Xavier
Disciplina: Engenharia de Software 2

INSTITUTO FEDERAL
Rio Grande do Sul

O que é GOF?

- **GOF** (Gang of Four);
- Origem;
- Três grupos principais:
  - Padrões de Criação;
  - Padrões Estruturais;
  - Padrões Comportamentais;

Resumindo…

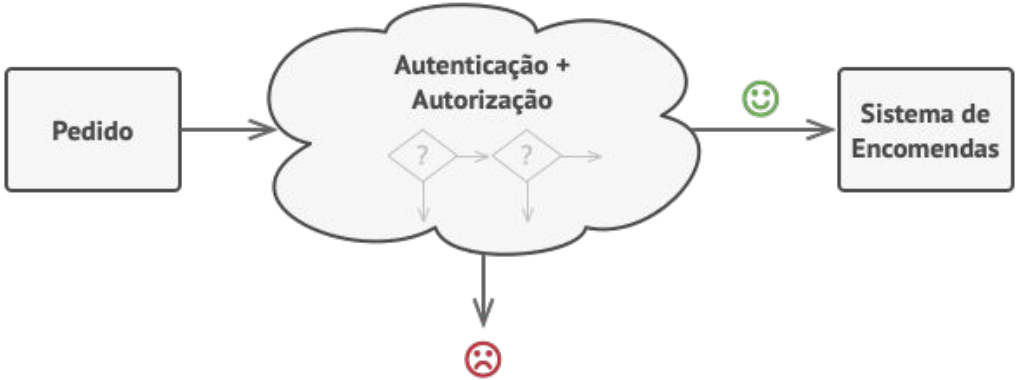
O objetivo principal do GOF é criar um conjunto de soluções reutilizáveis para problemas comuns de design em software, promovendo melhores práticas e abordagens para o desenvolvimento de sistemas mais flexíveis e manuteníveis. Esses padrões são usados para resolver problemas recorrentes de maneira eficiente e robusta.

Chain of responsibility

- **Propósito:** Permitir que uma requisição seja processada por uma cadeia de objetos, sem que o remetente da requisição precise saber qual objeto na cadeia irá tratá-la.

Problema x Solução

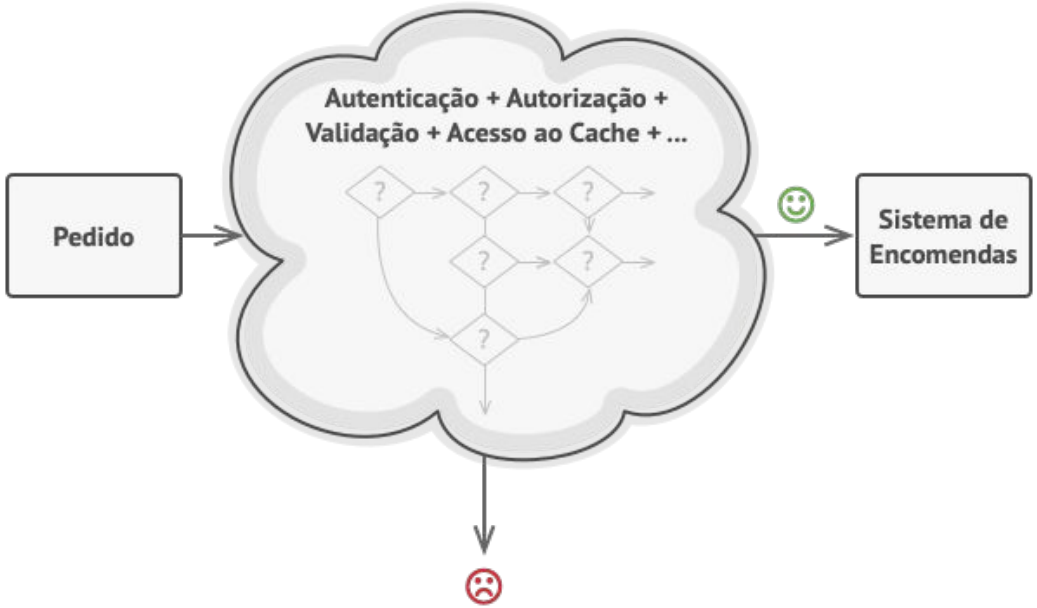
Problema



Imagine que você está criando um sistema de atendimento ao cliente com várias etapas de processamento, como verificação de dados do cliente, validação de crédito, e processamento de pagamento.

Fonte: Refactoring Guru. Padrão de Projeto Chain of Responsibility. Acessado em 2 de novembro de 2024.

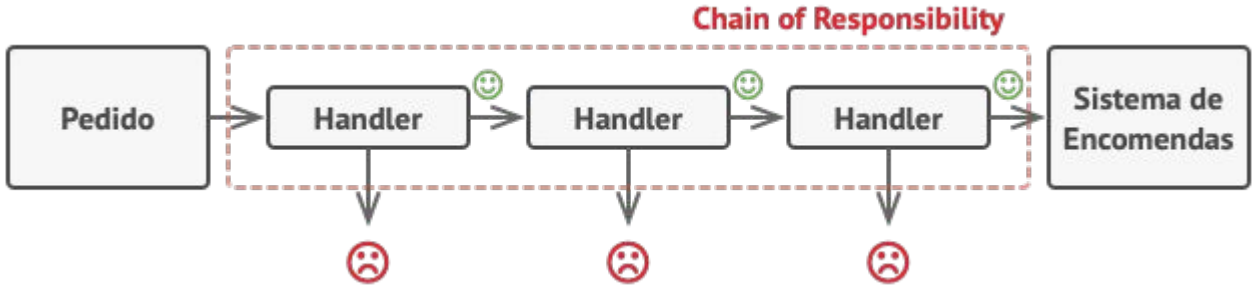
Problema



Fonte: Refactoring Guru. Padrão de Projeto Chain of Responsibility. Acessado em 2 de novembro de 2024.

Solução

Criar uma cadeia de objetos (ou manipuladores), onde cada objeto pode processar ou delegar a requisição para o próximo objeto na cadeia. O remetente da requisição apenas envia a solicitação à cadeia e os objetos, um após o outro, tentam processá-la. Se um objeto não pode processá-la, ele passa a requisição adiante.



Fonte: Refactoring Guru. Padrão de Projeto Chain of Responsibility. Acessado em 2 de novembro de 2024.

Prós

- Desacoplamento;
- Facilidade de Extensão;
- Flexibilidade;
- Responsabilidade de Processamento Distribuída;

Contras

- Cadeias longas podem ser difíceis de depurar;
- Desempenho;
- Complexidade;
- Cadeias vazias ou com manipuladores ineficazes;

Exemplo :

Processamento de Requisições em uma API de Pagamentos

```
5 references
public class PaymentRequest
{
    1 reference
    public decimal Amount { get; set; }
    2 references
    public bool HasSufficientFunds { get; set; }
    2 references
    public bool PaymentMethodValid { get; set; }
}
```

```
1 reference
public class BalanceCheckHandler : PaymentHandler
{
    4 references
    public override bool Handle(PaymentRequest request)
    {
        if (!request.HasSufficientFunds)
        {
            Console.WriteLine("Saldo insuficiente.");
            return false;
        }
        Console.WriteLine("Saldo verificado.");
        return _nextHandler?.Handle(request) ?? true;
    }
}
```

```
6 references
public abstract class PaymentHandler
{
    protected PaymentHandler _nextHandler;

    2 references
    public PaymentHandler SetNext(PaymentHandler handler)
    {
        _nextHandler = handler;
        return handler;
    }

    6 references
    public abstract bool Handle(PaymentRequest request);
}
```

```
1 reference
public class PaymentProcessingHandler : PaymentHandler
{
    3 references
    public override bool Handle(PaymentRequest request)
    {
        Console.WriteLine("Processando pagamento...");
        return true;
    }
}
```

```
1 reference
public class PaymentMethodValidationHandler : PaymentHandler
{
    3 references
    public override bool Handle(PaymentRequest request)
    {
        if (!request.PaymentMethodValid)
        {
            Console.WriteLine("Método de pagamento inválido.");
            return false;
        }
        Console.WriteLine("Método de pagamento válido.");
        return _nextHandler?.Handle(request) ?? true;
    }
}
```

```
public static void Main(string[] args)
{
    var paymentRequest = new PaymentRequest { Amount = 100, HasSufficientFunds = true, PaymentMethodValid = true };

    var balanceHandler = new BalanceCheckHandler();
    var paymentMethodHandler = new PaymentMethodValidationHandler();
    var processingHandler = new PaymentProcessingHandler();

    balanceHandler.SetNext(paymentMethodHandler).SetNext(processingHandler);

    if (balanceHandler.Handle(paymentRequest))
    {
        Console.WriteLine("Pagamento concluído com sucesso.");
    }
    else
    {
        Console.WriteLine("Falha no processamento do pagamento.");
    }
}
```

O que é GRASP?

- **GRASP** (General Responsibility Assignment Software Patterns);

- Coleção de padrões:

  Focada em orientar como distribuir e definir responsabilidades para objetos.

- Origem:

  descrita como uma coleção por Craig Larman em seu livro Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, na primeira edição de 1997.

Visão Geral do GRASP

- Objetivo principal;
- Importância do design;
- Principais padrões do GRASP:

  Information Expert, Controller, Creator, Low Coupling e High Cohesion.

Information Expert

- Exceções e Considerações;
- Uso do Modelo de Domínio;
- Vantagens e Desvantagens;
- Propósito;

Problema

Imagine que você está desenvolvendo um sistema de processamento de pedidos de empréstimos em um banco. Esse sistema precisa gerenciar diferentes etapas, como:

Verificação de Documentação: Confirma se o cliente enviou todos os documentos necessários.

Análise de Crédito: Calcula o risco com base no histórico do cliente.

Aprovação de Limite: Define o valor do empréstimo conforme a análise.

Processamento de Pagamento: Libera o valor ao cliente após a aprovação.

Solução

No sistema de processamento de pedidos de empréstimos, o Information Expert atribui cada etapa a quem possui as informações necessárias para realizá-la corretamente. A verificação de documentos é feita pela parte do sistema que conhece os documentos exigidos; à análise de crédito é responsabilidade do módulo que tem acesso ao histórico financeiro do cliente; a aprovação do limite de crédito é decidida pela entidade que possui as regras do banco para isso; e o pagamento é processado pela unidade que tem as informações de aprovação e da conta bancária do cliente. Essa distribuição das tarefas de forma lógica, com base nos dados controlados por cada "expert", torna o sistema mais eficiente e organizado.

Código

```java
class AprovacaoDeLimite {
    1 usage
    public static double definirLimiteDeCredito(Cliente cliente, double risco) {
        // Aqui o limite de crédito é aprovado com base no risco calculado
        if (risco > 0.5) {
            return cliente.getLimiteDeCredito() * 0.8; // Limite menor para alto risco
        }
        return cliente.getLimiteDeCredito(); // Limite original para baixo risco
    }
}
```

```java
class Documentos {
    1 usage
    private static final String DOCUMENTO_EXIGIDO = "RG, CPF, Comprovante de Renda";

    1 usage
    public static boolean verificarDocumentacao(Cliente cliente) {
        // Aqui verifica se o cliente tem todos os documentos necessários
        return cliente.getDocumento().contains(DOCUMENTO_EXIGIDO);
    }
}
```

Código

```java
class HistoricoDeCredito {
    1 usage
    public static double calcularRisco(Cliente cliente) {
        // Aqui o risco é calculado com base no histórico financeiro
        // Exemplo simplificado: se o limite de crédito é muito alto, o risco aumenta
        if (cliente.getLimiteDeCredito() > 50000) {
            return 0.8; // Alto risco
        }
        return 0.2; // Baixo risco
    }
}

class AprovacaoDeLimite {
    1 usage
    public static double definirLimiteDeCredito(Cliente cliente, double risco) {
        // Aqui o limite de crédito é aprovado com base no risco calculado
        if (risco > 0.5) {
            return cliente.getLimiteDeCredito() * 0.8; // Limite menor para alto risco
        }
        return cliente.getLimiteDeCredito(); // Limite original para baixo risco
    }
}
```

Código

```
class ProcessamentoDePagamento {
    1 usage
    public static void liberarPagamento(Cliente cliente, double limiteAprovado) {
        // Aqui o pagamento é liberado após aprovação do limite
        if (limiteAprovado > 0) {
            System.out.println("Pagamento de " + limiteAprovado + " liberado para o cliente " + cliente.getDocumento());
        } else {
            System.out.println("Limite de crédito não aprovado para o cliente " + cliente.getDocumento());
        }
    }
}
```

Código

```java
public class SistemaDeEmprestimo {
    public static void main(String[] args) {
        Cliente cliente = new Cliente( nome: "João",  documento: "RG, CPF, Comprovante de Renda",  limiteDeCredito: 60000);

        // Verificação de documentação
        if (Documentos.verificarDocumentacao(cliente)) {
            // Análise de crédito
            double risco = HistoricoDeCredito.calcularRisco(cliente);

            // Aprovação de limite
            double limiteAprovado = AprovacaoDeLimite.definirLimiteDeCredito(cliente, risco);

            // Processamento de pagamento
            ProcessamentoDePagamento.liberarPagamento(cliente, limiteAprovado);
        } else {
            System.out.println("Documentação incompleta. O processo de empréstimo não pode continuar.");
        }
    }
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\Je
Pagamento de 48000.0 liberado para o cliente RG, CPF, Comprovante de Renda
```

Quando usar?

Devemos usar o padrão Information Expert quando precisamos definir qual parte do sistema deve ser responsável por uma determinada tarefa, com base nas informações que cada classe ou objeto possui. Esse padrão é útil em situações onde podemos delegar responsabilidades de forma que cada classe manipule os dados que já controla diretamente, reduzindo o acoplamento entre as partes do sistema e aumentando a coesão.

Referências

- ALEX K. **Guiding Object-Oriented Design in Java**. Disponível em: https://alxkm.github.io/posts/grasp. Acesso em: 3 nov. 2024.
- CU BOULDER COMPUTER SCIENCE. **GRASP - General Responsibility Assignment Software Patterns**. Universidade do Colorado. Disponível em: https://home.cs.colorado.edu. Acesso em: 3 nov. 2024.
- WIKIPEDIA. **Object-Oriented Software Engineering Course - University of Illinois**. Universidade de Illinois. Disponível em: https://cs125.cs.illinois.edu/. Acesso em: 3 nov. 2024.

Referências

- BRUEGGE, Bernd; DUTOIT, Allen H. **Object-Oriented Software Engineering Using UML, Patterns, and Java™**. 3. ed. Upper Saddle River: Prentice Hall, 2009.
- LARMAN, Craig. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development**. 3. ed. Upper Saddle River: Prentice Hall, 2004.
- ARAÚJO, Rogério. **Padrões de Projetos GoF: Padrões de Criação.** *In:* Gran Tecnologia e Educação S/A. **Gran.** Brasilia, 31 ago. 2023. Disponível em: https://blog.grancursosonline.com.br/padroes-de-projetos-gof-padroes-de-criacao/. Acesso em: 2 nov. 2024.

Referências

- BRUEGGE, Bernd; DUTOIT, Allen H. **Object-Oriented Software Engineering Using UML, Patterns, and Java™**. 3. ed. Upper Saddle River: Prentice Hall, 2009.
- GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. ***Design patterns: elements of reusable object-oriented software****. São Paulo:* Addison-Wesley, 1995.
- REFRACTORY GURU. ***Chain of responsibility.*** Disponível em: https://refactoring.guru/pt-br/design-patterns/chain-of-responsibility.