Aula 10 - Design Patterns

Disciplina: Arquitetura de Software

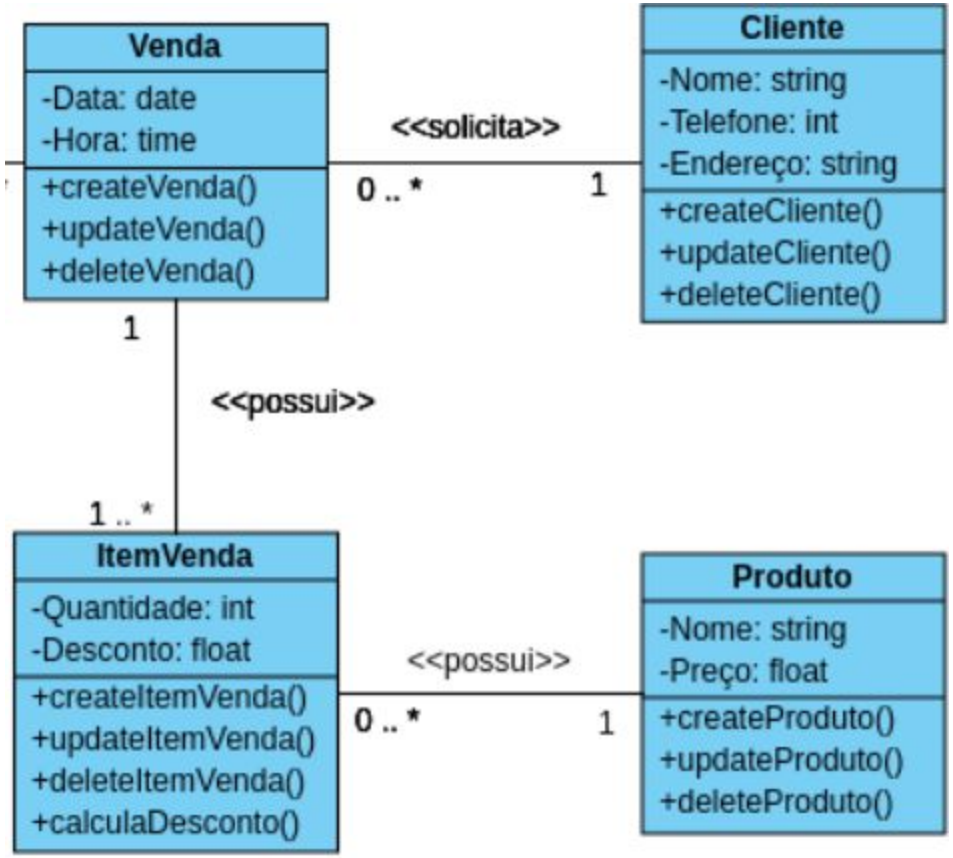
Prof. Me. João Paulo Biazotto

UniCesumar

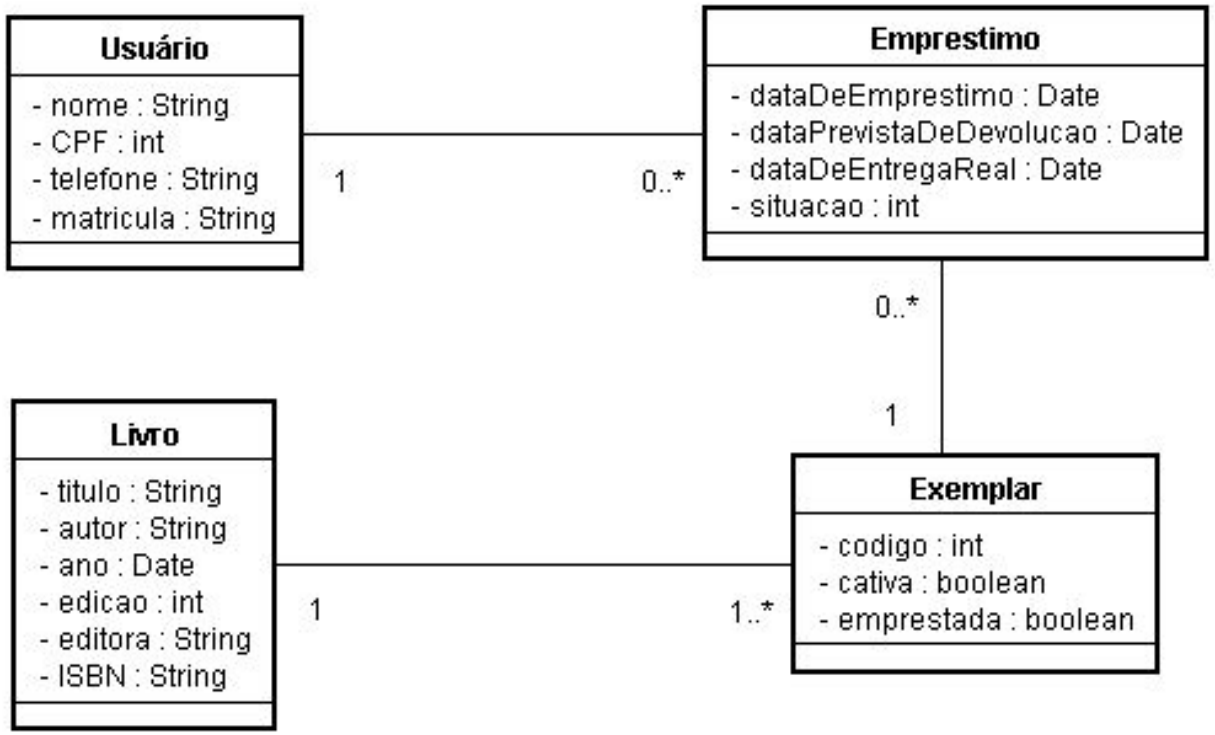
Problemática e Motivação

- Vários problemas de design de software se repetem em diferentes projetos;
- Milhares (ou até milhões) de programadores planejam e executam soluções para esses problemas;

Sistema de Vendas



Sistema de Bibliotecas



Problemática e Motivação

- Tais semelhanças também ocorrem na implementação dos sistemas;
- Uma venda pode ter muitas formas de pagamento, cada uma com uma especificidade:
  - Boleto;
  - Cartão de Crédito;
  - Cartão de Débito;

Problemática e Motivação

- Um sistema de notificação pode enviar mensagens por diferentes canais:
  - Instagram;
  - Whatsapp;
  - SMS;
- O mesmo sistema usando diferentes ***estratégias.***
- Qual seria a forma mais eficiente de implementar isso?

Problemática e Motivação

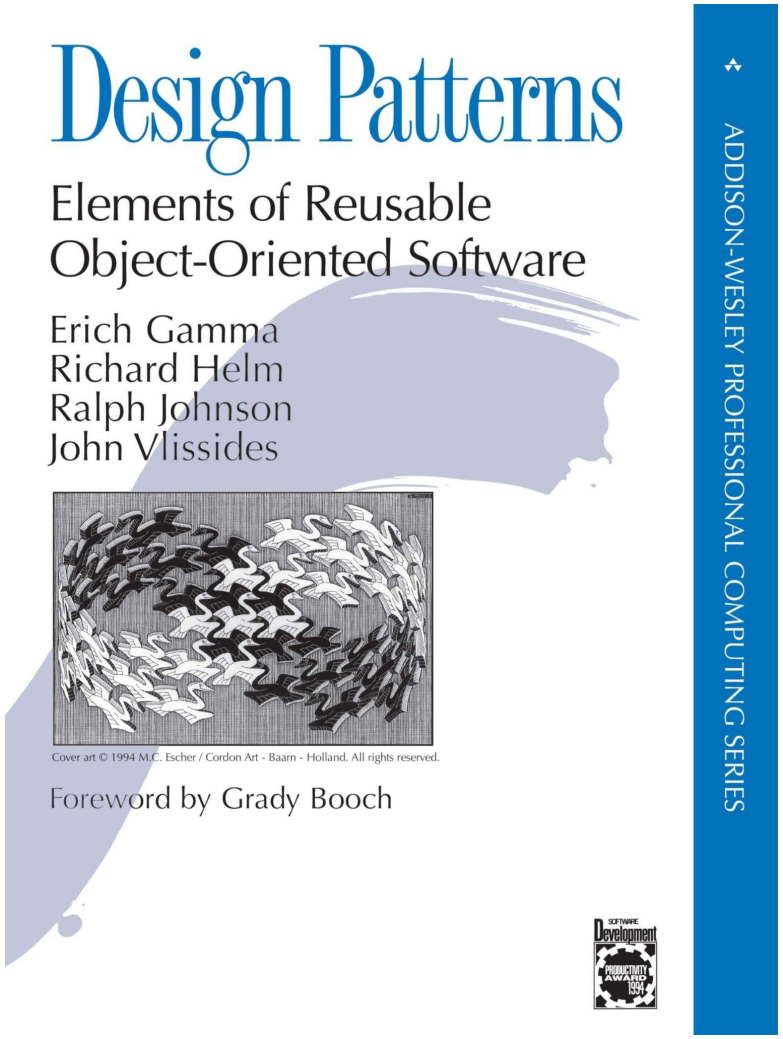
- Precisamos sempre implementar uma nova solução para os mesmos problemas?

Design Patterns

- Em 1994, quatro autores (que ficaram conhecidos como *Gang of Four - GoF)* perceberam que alguns desafios do design de software eram comuns;
- Vários desenvolvedores já haviam proposto soluções para tais problemas, e tais soluções poderiam ser re-usadas.
- Eles então catalogaram tais soluções e escreveram um livro;

Design Patterns

- O catálogo de padrões GoF é uma referência até hoje para o desenvolvimento eficiente de sistemas orientados a objetos;
- Eles descrevem 23 padrões de projetos.



O que são Design Patterns?

Design Patterns (ou padrões de projeto) são soluções reutilizáveis para problemas recorrentes no design de software. Eles fornecem um modelo comprovado para estruturar código de forma eficiente e escalável.

Por que usar Design Patterns?

- Melhor organização do código
- Maior reutilização
- Facilidade de manutenção
- Padrões amplamente conhecidos e documentados

Benefícios

- Reduzem a complexidade do código
- Facilitam a comunicação entre desenvolvedores
- Melhoram a testabilidade e extensibilidade

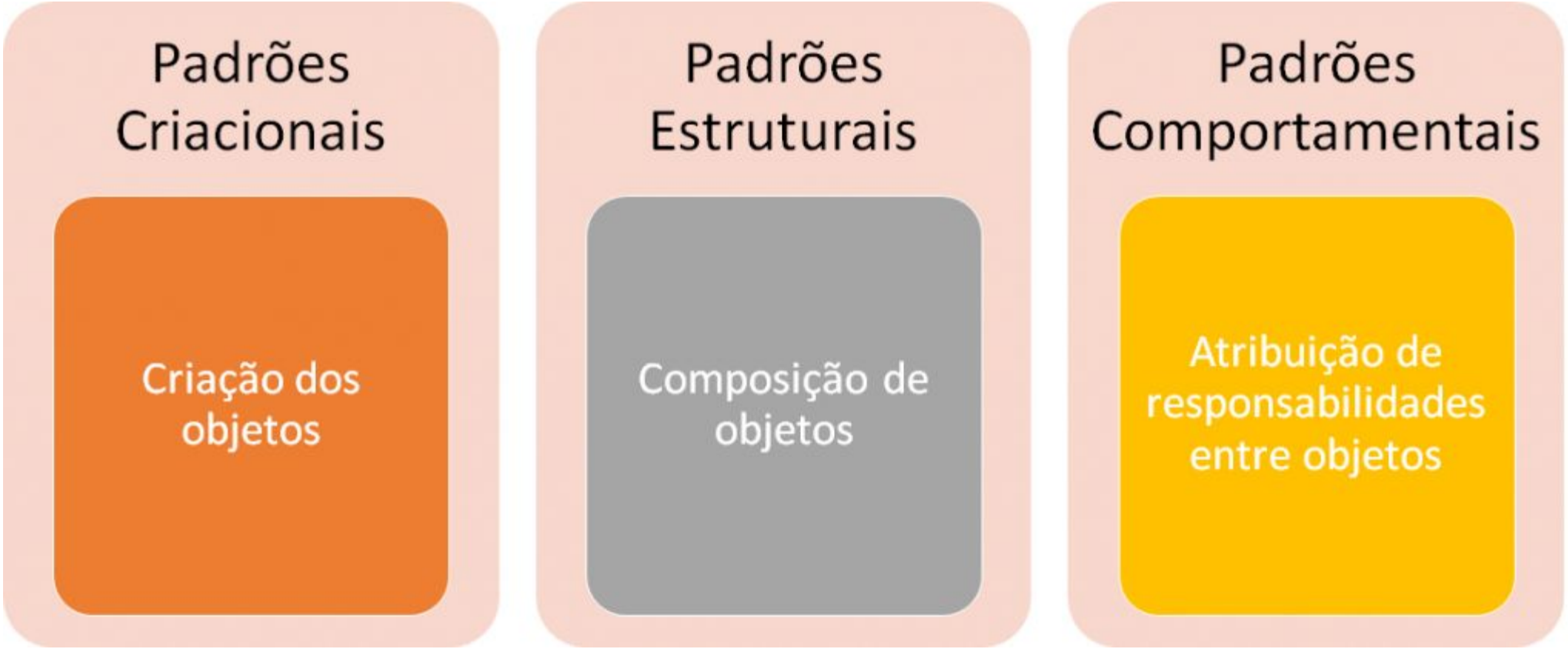
Limitações

- Requerem conhecimento prévio para aplicação correta
- Podem ser desnecessários para problemas simples
- Má implementação pode gerar código mais complexo

Como os Design Patterns se encaixam na Arquitetura de Software?

- Auxiliam na criação de sistemas modulares
- Reduzem o acoplamento entre componentes
- Melhoram a manutenção dos sistemas.

Categorias de Design Patterns



Categorias de Design Patterns

- Padrões Criacionais: *Focados na criação de objetos de forma eficiente e flexível.*

- Padrões Estruturais: *Auxiliam na organização de classes e objetos para formar estruturas maiores.*

- Padrões Comportamentais: *Definem como os objetos interagem e comunicam entre si.*

Padrões comportamentais

Padrões **comportamentais** se concentram em como os objetos **interagem** e como **responsabilidades** são distribuídas entre eles.

Padrões comportamentais

- Promover uma comunicação **flexível** e eficiente entre objetos.

- Evitar dependências **rígidas** e lógica de controle espalhada.

Padrões comportamentais

- Favorecem o baixo **acoplamento** e a alta coesão.

- Facilitam a manutenção, extensão e reutilização de código.

Padrões Comportamentais

- ***Template Method***: Define o esqueleto de um algoritmo, deixando passos específicos para as subclasses.
- ***Mediator*** : Centraliza a comunicação entre objetos para reduzir dependências diretas.
- ***State:*** Permite que um objeto altere seu comportamento quando seu estado muda.

Padrões Comportamentais

- ***Observer***: Notifica múltiplos objetos quando o estado de um objeto muda.
- ***Command***: Encapsula uma solicitação como um objeto.
- ***Iterator:*** Fornece uma forma de acessar os elementos de uma coleção sequencialmente.
- ***Strategy***: Permite alterar o algoritmo de uma tarefa em tempo de execução.

Padrões Comportamentais

- ***Observer:*** Notifica múltiplos objetos quando o estado de um objeto muda.
- ***Command:*** Encapsula uma solicitação como um objeto.
- ***Iterator:*** Fornece uma forma de acessar os elementos de uma coleção sequencialmente.
- ***Strategy:*** Permite alterar o algoritmo de uma tarefa em tempo de execução.

Problemática

- Você tem um sistema de vendas que oferece, inicialmente, apenas um tipo de pagamento.
- Assim, você implementa a lógica de pagamento na classe de venda.
- No entanto, a medida que o software cresce, você quer prover mais métodos de pagamento.

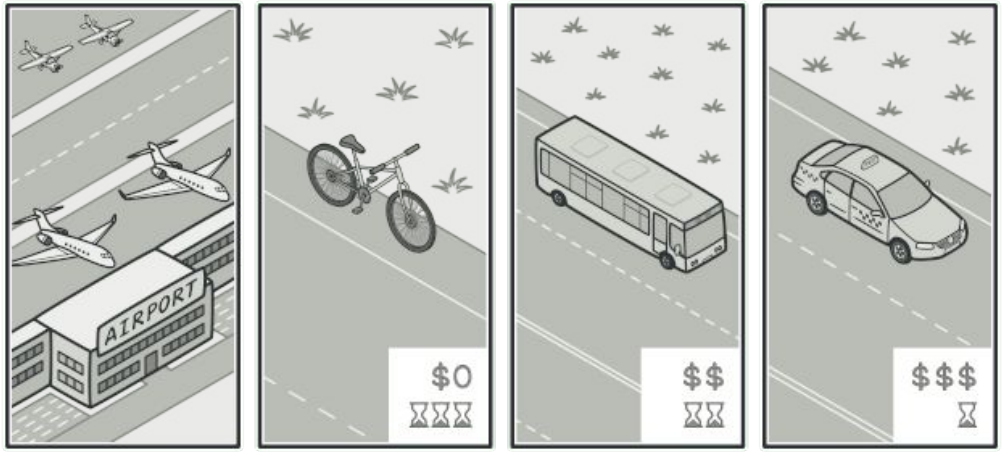
Problemática

- Qualquer **alteração** na lógica de pagamento — seja uma simples correção de bug ou um pequeno ajuste na pontuação de ruas — **afeta toda a classe**.

- O que fazer?

Padrão Strategy

- O Strategy Pattern propõe **mover algoritmos que variam para classes separadas**, chamadas de estratégias, permitindo que sejam intercambiáveis em tempo de execução.



Padrão Strategy

- Context: Classe principal que utiliza uma estratégia.

- Strategy (Interface): Define um contrato comum para todas as estratégias.

- Concrete Strategies: Implementações específicas do algoritmo.

Padrão Strategy

- Cálculos de desconto em carrinho de compras.
- Algoritmos de ordenação configuráveis.
- Estratégias de autenticação (senha, biometria, OAuth...).

Bora programar: Implementação do Padrão Strategy

DÚVIDAS?

UniCesumar