Padrões de projeto

Bruna Diirr

brunadiirr@ic.uff.br

Introdução

Escritores (livros, HQs, roteiros) raramente inventam novas histórias

- Ideias frequentemente reusadas!

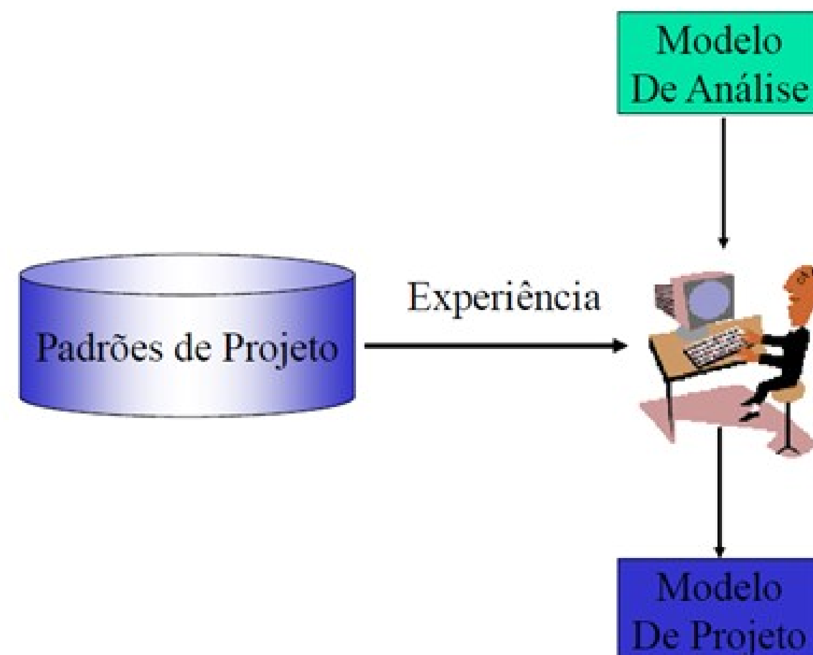
Projetistas também reutilizam soluções

- De preferência as boas!
- Problemas são tratados de modo a não reinventar soluções
- Experiência é o que torna uma pessoa um expert
  - Prever o que pode dar errado
  - Prever potenciais mudanças
  - Prever dependências indesejadas
- Porém, muitas vezes transmitidas de forma tácita, sem apoio de documentação

Padrões de projeto

"... describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

(Alexander, 1977)



Padrões de projeto

Soluções padronizadas para um problema recorrente no projeto de sistemas, que seja comprovadamente útil em um determinado contexto

- Soluções independente de tecnologia ou linguagem de programação
- Discussão se mantem no nível do projeto sem se ater a detalhes de implementação
- Ajudam a promover boa prática de projeto

Codificam o conhecimento existente, de forma que possa ser reaplicado em contextos diferentes, além de aprimorar a comunicação entre desenvolvedores

- Documentam experiência!

Tornam projetos OO mais flexíveis, elegantes e reusáveis

Padrões de projeto

Iniciaram a “febre” de padrões, mas existem outros:

- Padrões de gerenciamento
  - Ex. Métodos ágeis como SCRUM e XP
- Padrões organizacionais
- Padrões de análise
- Padrões arquiteturais
- Padrões de implementação

Veja aqui uma pequena lista de padrões (coletadas por Booch)

Por que aprender padrões?

- Aprender com experiência dos outros
  - Identificar problemas comuns e utilizar soluções testadas e bem documentadas
- Ajuda na documentação e na aprendizagem
  - Mais fácil compreensão de sistemas existentes e documentação da arquitetura do software
  - Novato pode agir mais como um especialista
- Aprender a programar bem com orientação a objetos
  - Padrões utilizam melhores práticas em OO para atingir resultados desejados
- Desenvolver software de melhor qualidade
  - Os padrões utilizam eficientemente polimorfismo, herança, modularidade, composição, abstração para construir código reutilizável, eficiente, de alta coesão e baixo acoplamento
- Vocabulário comum
  - Sistema menos complexo ao permitir que se fale em um nível mais alto de abstração

Mas não ache que padrões...

... fornecem uma solução exata

... resolvem todos os problemas de projeto

... aplicam-se apenas ao projeto OO
Só surgiram nesse contexto

Elementos do padrão de projeto

Nome
- Apelido para descrever o padrão

Problema
- Descreve quando o padrão pode ser aplicado

Contexto
- Características da situação em que o padrão se aplica

Solução
- Determina o que fazer para resolver o problema

Consequências
- Descreve resultados positivos (vantagens) e negativos (desvantagens) de aplicação do padrão

Usos conhecidos
- Ocorrência do padrão e de suas aplicações em sistemas conhecidos

Conjuntos de padrões de projetos

Muitas vezes o termo “padrão de projeto” é usado como sinônimo para os “padrões de projeto GoF”

Porém, existem diversos outros conjuntos de padrões de projeto além do GoF, apesar desses serem os mais conhecidos

Outros conjuntos de padrões de projeto:

- Padrões GRASP
- Padrões Arquiteturais (Martin Fowler)
- Padrões J2EE
- Etc.

Na disciplina, focaremos em GRASP e GoF!

Padrões GRASP (*General Responsibility and Assignment Software Patterns*)

Introduzidos por Craig Larman em seu livro "Applying UML and Patterns"

Descrevem princípios fundamentais da atribuição de responsabilidades a objetos, expressas na forma de padrões

- Responsabilidade = Fazer algo OU Conhecer lembrar algo
- Responsabilidade != Método
  - Métodos implementam responsabilidades
  - Objetos colaboram para cumprir responsabilidades

Exploram os princípios fundamentais de sistemas OO através de 9 padrões

Se você domina esses padrões, pode dizer que compreende o paradigma orientado a objetos

Padrões GRASP

Design Patterns GRASP	
Padrões Fundamentais	Controller
	Creator
	Information Expert
	Low Coupling
	High Cohesion
Padrões Avançados	Indirection
	Protected Variations
	Polymorphism
	Pure Fabrication

Padrões GRASP: Especialista (*Expert)*

Problema: Qual o princípio mais fundamental para atribuir responsabilidades?

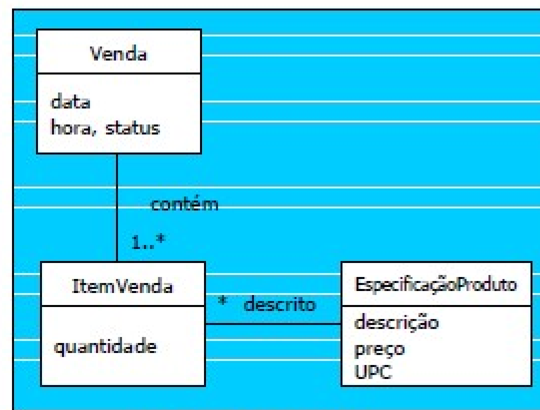
Solução: Atribua responsabilidade à uma classe que possui a informação necessária para cumpri-la.

"Quem tem a informação realiza o trabalho."

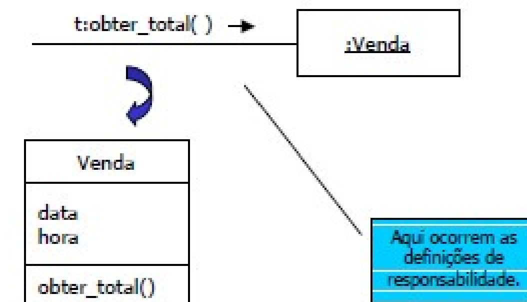
Exemplo: Quem seria responsável por calcular o *total-geral* de uma *venda*?

A classe *Venda* possui a informação para isso

Classe	Responsabilidade
Venda	sabe total geral da venda
Item de Venda	sabe sub-total de cada item
Especificação do Produto	sabe preço do produto



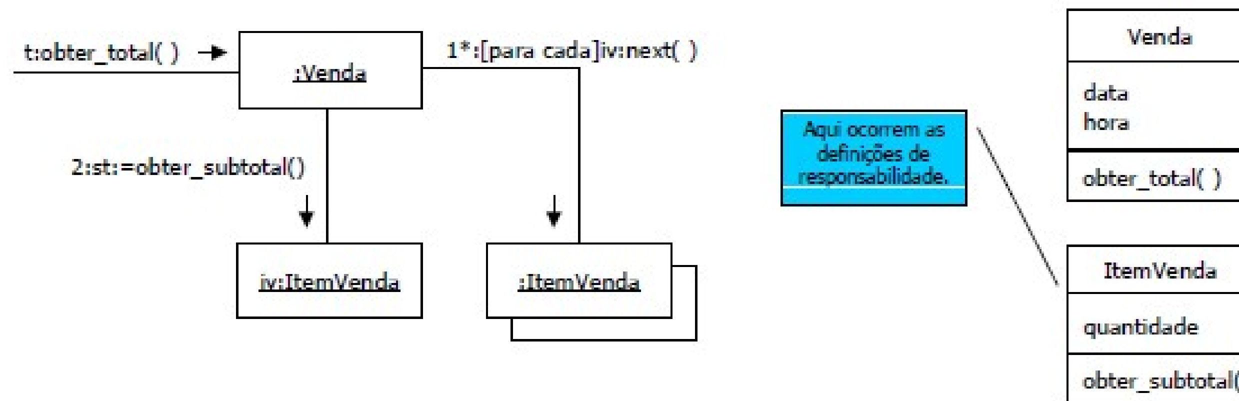
Modelo Conceitual Parcial



Padrões GRASP: Especialista (*Expert)*

Mas quem seria responsável pelo *subtotal* de um *ItemVenda*?

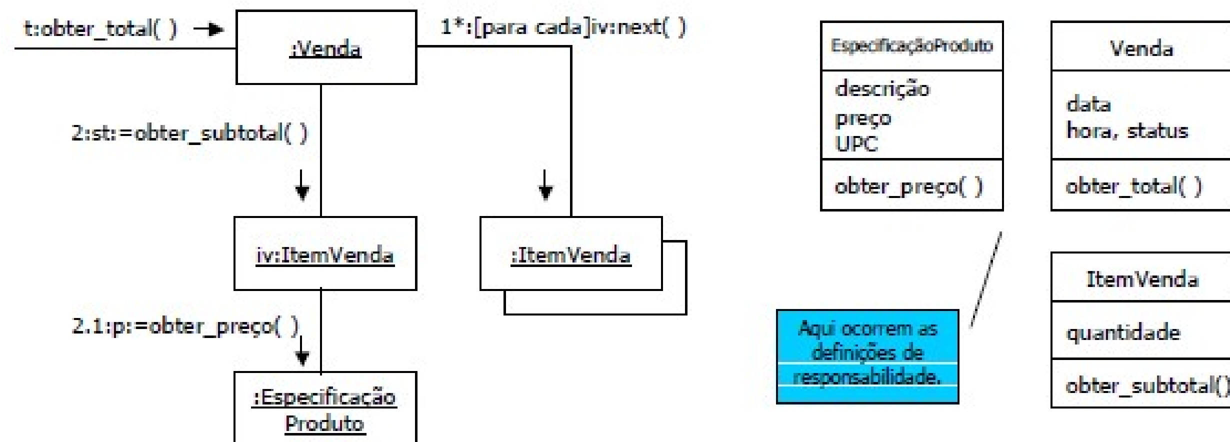
- Informação necessária: *ItemVenda.quantidade* e *EspecificaçãoProduto.preço*
- Pelo padrão, a classe *ItemVenda* deve ser a responsável



Padrões GRASP: Especialista (*Expert)*

Porém, para cumprir essa responsabilidade de conhecer ou informar seu subtotal um *ItemVenda* precisa conhecer o preço do Item.

– Portanto, o *ItemVenda* deve mandar uma mensagem para a *EspecificaçãoProduto* para saber o preço do item



Padrões GRASP: Especialista (*Expert*)

Observações

- É o padrão mais usado de todos para atribuir responsabilidades
- A informação necessária frequentemente está espalhada em vários objetos e mensagens são usadas para estabelecer colaborações
  - Muitos experts parciais
  - No exemplo, determinar o total de uma venda requer a colaboração de 3 objetos, em 3 classes diferentes
- O resultado final é diferente do mundo real
  - No mundo real, uma venda não calcula seu próprio total; Isso seria feito por pessoas
- Encapsulamento mantido (objetos usam sua própria informação para cumprir responsabilidades)
- Fraco acoplamento entre objetos e sistemas mais robustos/fáceis de manter
- Alta coesão (objetos fazem tudo que é relacionado à sua própria informação)

Padrões GRASP: Criador (*Creator*)

Problema: Quem seria responsável por criar novas instâncias de alguma classe?

Solução: Atribua à classe B a responsabilidade de criar uma instância da classe A se:

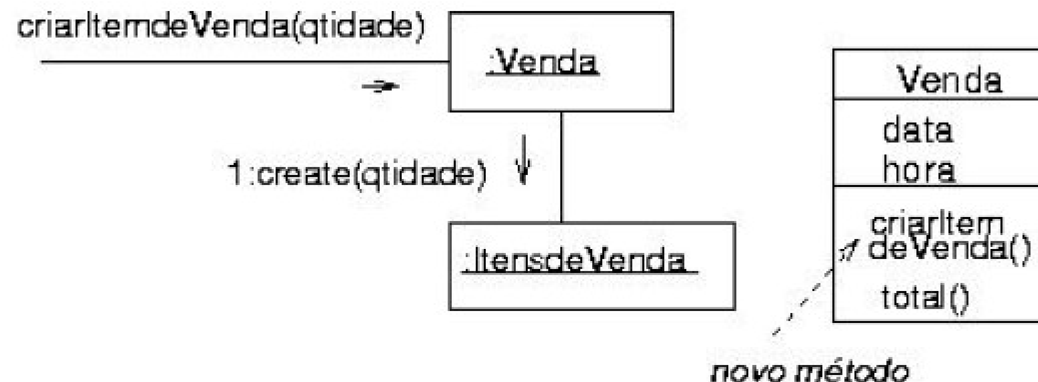
1. B agrega objetos de A
2. B contém A
3. B armazena instâncias de A
4. B usa objetos de A
5. B possui informação necessária a criação de A (B é um especialista para criar A)

Se mais de uma opção se aplica, escolha o B que agregue ou contenha objetos da classe A

Padrões GRASP: Criador (*Creator)*

Exemplo: Quem seria responsável por criar a instância da classe *Item de Venda*?

classe *Venda* agrega muitos objetos da classe *ItensdeVenda*



Para Criar uma Linha de Item de Venda

Padrões GRASP: Criador (*Creator*)

Observações:

- Fraco acoplamento (objeto precisa ser visível ao criador depois da criação de qualquer maneira)

Padrões GRASP: Alta coesão (*High coesion)*

Problema: Como manter a complexidade sob controle?

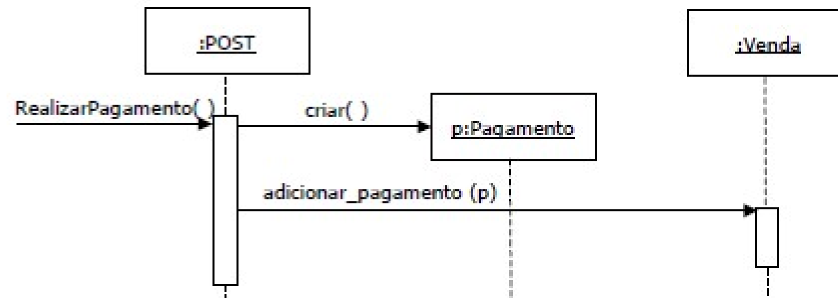
Classes que fazem muitas tarefas não relacionadas são mais difíceis de entender, de manter e de reusar, além de serem mais vulneráveis à mudança (baixa coesão)

Solução: Atribuir uma responsabilidade para que a coesão se mantenha alta

Padrões GRASP: Alta coesão (*High coesion)*

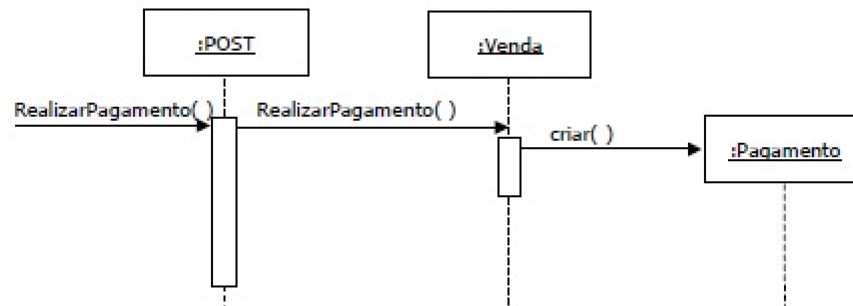
Exemplo: Quem seria responsável por criar um *Pagamento* e associá-lo à *Venda*?

Pelo padrão Criador, seria *POST*. Mas se *POST* for responsável pela maioria das operações do sistema, ele vai ficar cada vez mais sobrecarregado e sem coesão



Outra seria delegar a responsabilidade a *Venda*, aumentando a coesão de *POST*

A criação do processo de pagamento faz mais sentido, pois o *Pagamento* é parte de *Venda*



Padrões GRASP: Baixo acoplamento (*Low coupling*)

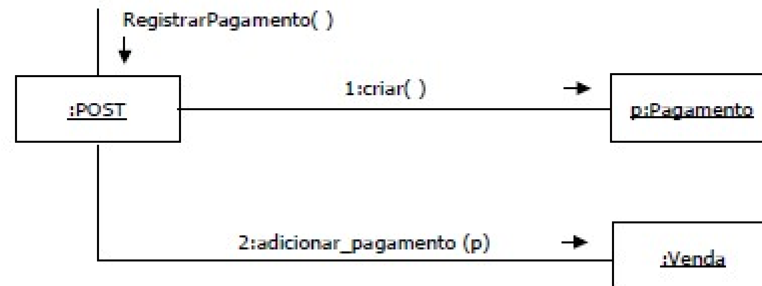
Problema: Como minimizar dependências e maximizar reuso?

Solução: Atribuir a responsabilidade de modo que o acoplamento (dependência entre classes) seja baixo

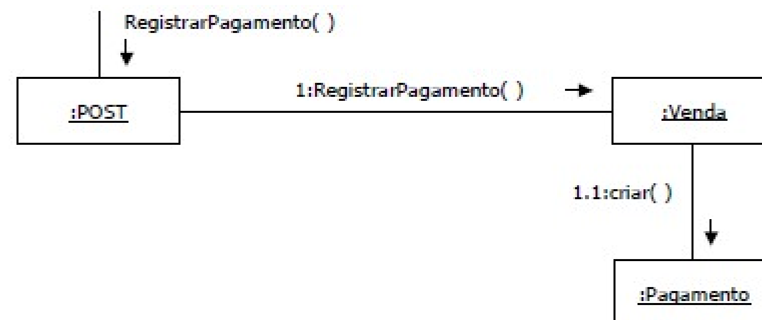
Padrões GRASP: Baixo acoplamento (*Low coupling)*

Exemplo: Quem deve ser responsável por criar um *Pagamento* e associá-lo à *Venda*?

Pelo padrão Criador, poderia ser *POST* (ele "registra" pagamentos no mundo real)



Uma solução melhor, para manter baixo acoplamento, seria a própria *Venda* criar *Pagamento*, pois *Venda* tem que ter conhecimento de *Pagamento*



Padrões GRASP: Controlador (*Controller)*

Problema: Quem seria responsável por lidar com um evento do sistema?

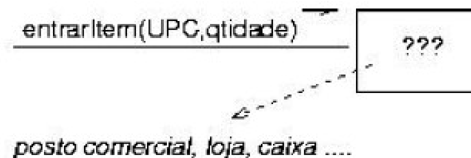
um controlador é um objeto de interface responsável por gerenciar um evento do sistema, definindo métodos para operações do sistema

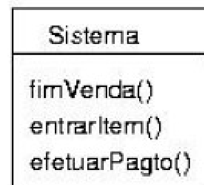
Solução: Atribua responsabilidades para lidar com mensagens de eventos do sistema a uma classe que:

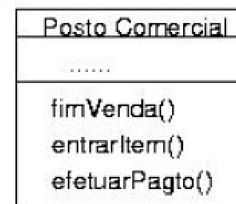
1. represente o sistema como um todo
2. represente a organização
3. represente algo ativo no mundo real envolvido na tarefa (por exemplo o papel de uma pessoa)
4. represente um controlador artificial dos eventos de sistema de um caso de uso

Padrões GRASP: Controlador (*Controller)*

Exemplo: Quem deveria ser o controlador para eventos do sistema tais como *entrarItem* e *fimVenda*?







Padrões GRASP: Controlador (*Controller)*

Observações:

- Use o mesmo controlador para todos os eventos do sistema no mesmo caso de uso
- Classes do tipo janela, applet, aplicações, documento não deveriam realizar tarefas associadas a eventos do sistema. Elas apenas recebem e delegam os eventos ao controlador
  - Um evento do sistema é gerado por um ator
- Um controlador não deve ter muitos atributos e nem manter informação sobre o domínio do problema
- Um controlador não deve realizar muitas tarefas, apenas delegá-las
- Um bom projeto deve dar vida aos objetos, atribuindo-lhes responsabilidades, até mesmo se eles forem seres inanimados
- A camada de apresentação (interface com o usuário) não deve tratar eventos do sistema

Padrões clássicos ou Padrões GoF (*Gang of Four*)

Introduzidos por Erich Gamma, John Vlissides, Ralph Jonhson e Richard Helm no livro "Design Patterns"

Descrevem soluções genéricas para problemas mais comuns do desenvolvimento de software OO

- Soluções obtidas através da coleta de experiências de sucesso na indústria de software, principalmente de projetos em C++ e SmallTalk

Descrevem 23 padrões

Padrões GoF

Classificação

Finalidade

- Criacionais
  - Ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados
- Estruturais
  - Preocupam-se com a forma como classes e objetos são compostos para formar estruturas maiores
- Comportamentais
  - Preocupam-se com algoritmos e a atribuição de responsabilidades entre os objetos, dando um grande foco na comunicação entre eles

Escopo

- Classe
  - Descrevem relacionamento entre classes e subclasses
  - Relacionamentos são fixos e definidos em tempo de compilação (Estáticos)
- Objeto
  - Descrevem relacionamentos entre objetos
  - Relacionamentos podem ser alterados em tempo de execução (Dinâmicos)

Padrões GoF

Classificação

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

GRASP x GoF

GoF

- Exploram soluções mais específicas

GRASP

- Práticas mais pontuais de aplicação de técnicas OO
- Ocorrem na implementação de vários padrões GoF

Padrões de projeto

Bruna Diirr

brunadiirr@ic.uff.br