Padrões GRASP

GoF e GRASP

- Padrões GoF
  - exploram soluções mais específicas
- Padrões GRASP
  - refletem práticas mais pontuais da aplicação de técnicas OO
  - ocorrem na implementação de vários padrões GoF

Responsabilidades

"Contrato ou obrigação de um tipo ou classe"

[Booch e Rumbaugh]

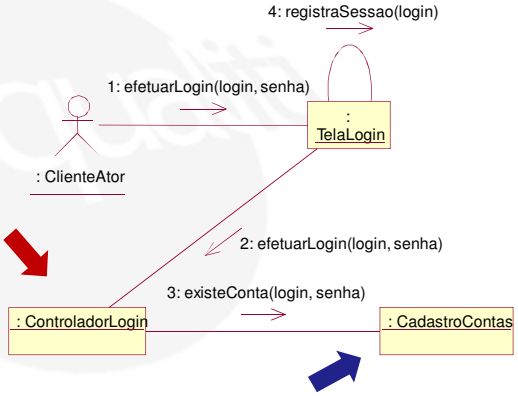
- Dois tipos de responsabilidades dos objetos:
  - De conhecimento (**knowing**):
    - sobre dados privativos e encapsulados; sobre objetos relacionados; sobre coisas que pode calcular ou derivar.
  - •De realização (***doing***):
    - fazer alguma coisa em si mesmo; iniciar uma ação em outro objeto; controlar e coordenar atividades em outros objetos.

Atribuição de Reponsabilidades

- Atribuição a objetos durante o design (projeto)
  - Responsabilidades → Classes e Métodos
  - tradução depende da granularidade da responsabilidade
- Responsabilidades do tipo *doing*
  - Realizadas por um único método ou uma coleção de métodos trabalhando em conjunto
- Responsabilidades do tipo *knowing*
  - inferidas a partir do modelo conceitual (são os atributos e relacionamentos)

Responsabilidades e Diagramas de Interação

- Diagramas de interação mostram escolhas ao atribuir responsabilidades a objetos
- No diagrama ao lado o **ControladorLogin** têm a responsabilidade de efetuar o login
  - método etuarLogin()
- O cumprimento dessa responsabilidade requer colaboração com **CadastroContas**



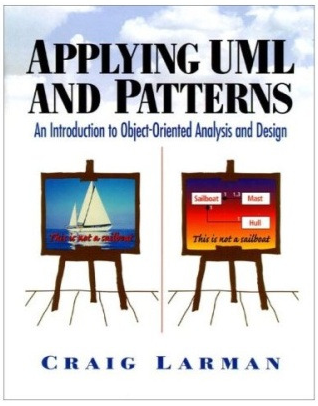
Exemplo do Internet Banking

Padrões GRASP

GRASP: General Responsibility and Assignment Software Patterns

- descrevem os princípios fundamentais da atribuição de responsabilidades a objetos, expressas na forma de padrões;
- Ajudam à compreender melhor a utilização de vários do paradigma O-O em projetos mais complexos;
- Exploram os princípios fundamentais de sistemas OO
  - 5 padrões fundamentais
  - 4 padrões avançados

Introduzidos no livro:



Padrões GRASP

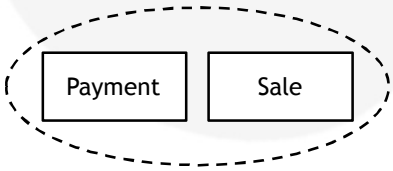
- Padrões básicos
  - Information Expert
  - Creator
  - High Cohesion
  - Low Coupling
  - Controller
- Padrões avançados
  - Polymorphism
  - Pure Fabrication
  - Indirection
  - Protected Variations

Expert (especialista de informação)

- **Problema**
  - No design, quando são definidas interações entre objetos
  - Precisamos de um princípio para atribuir responsabilidades a classes
- **Solução**
  - Atribuir uma responsabilidade ao especialista de informação: classe que possui a informação necessária para cumpri-la
  - Comece a atribuição de responsabilidades ao declarar claramente a responsabilidade

POS System

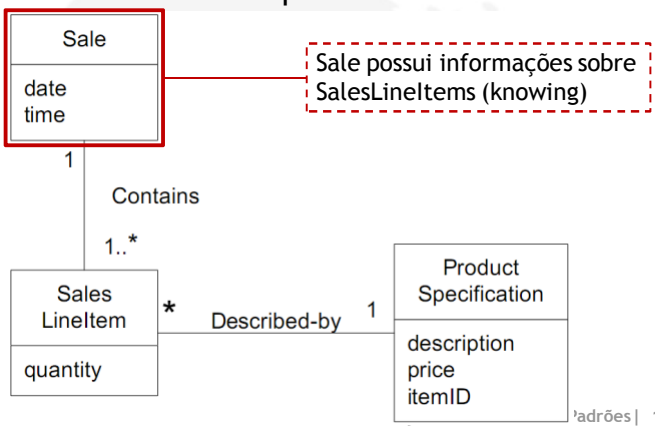
- Para a ilustração dos padrões GRASP analisaremos o exemplo do POS System.
  - Padrões GRASP são utilizados quase sempre durante a análise.
- POS System
  - Utilizado tipicamente em lojas (comércio)
  - Grava vendas (Sale) e gera pagamentos (Payment)



Foco dos casos de uso analisados aqui.

Expert

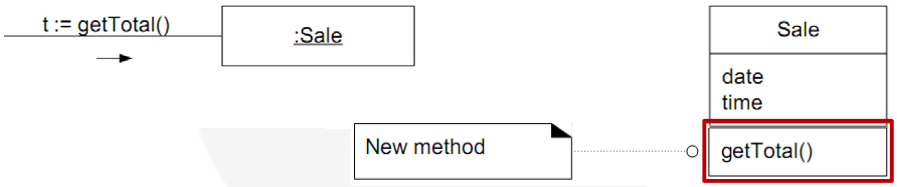
- No sistema abaixo, uma classe precisa saber o total geral de uma venda (Sale).
- Que classe deve ser a responsável?



slide com animação

Expert [cont.]

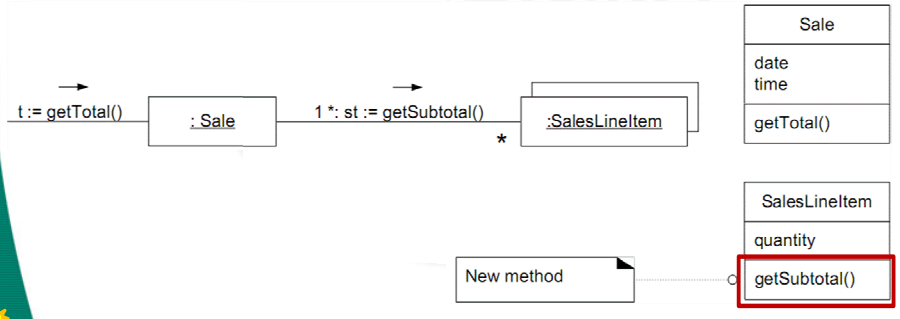
- A nova responsabilidade é conduzida por uma operação no diagrama de interação
  - Um novo método é criado



slide com animação

Expert [cont.]

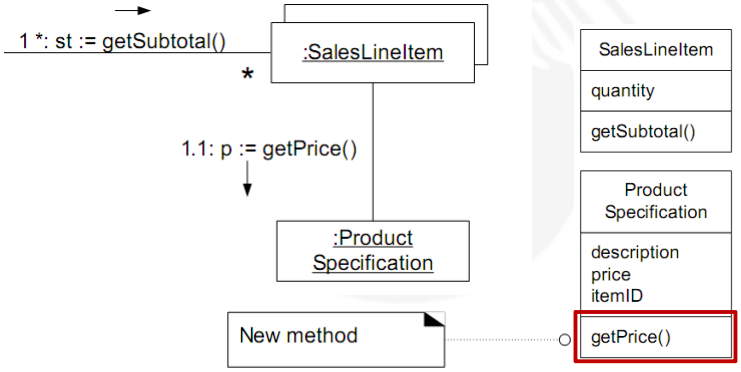
- Mas como a classe Sale vai calcular o valor total?
  - Somando os subtotais. Mas quem faz isto?
- A responsabilidade para cada subtotal é atribuída ao objeto SalesLineItem (item de linha do pedido)



slide com animação

Expert [cont.]

- O subtotal depende do preço.
- ProductSpecification é o especialista que conhece o preço, portanto ...
  - a responsabilidade é dele



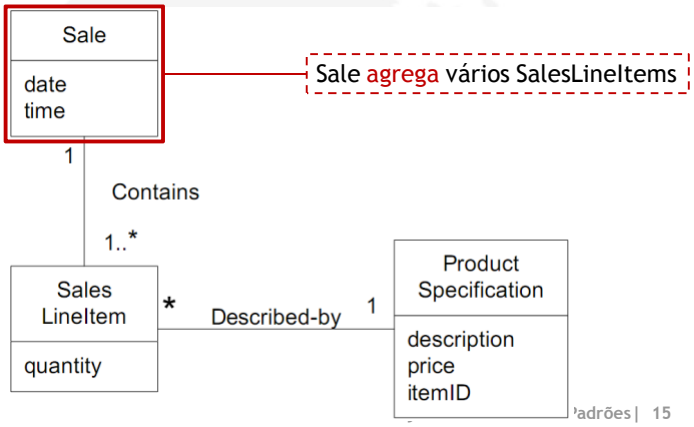
slide com animação

Creator

- **Problema**:
  - Que classe deve ser responsável pela criação de uma nova instância de uma classe?
- Solução:
  - Atribua a B a responsabilidade de criar A se:
    - B agrega A objetos;
    - B contém A objetos;
    - B guarda instâncias de A objetos;
    - B faz uso de A objetos;
    - B possui dados para inicialização de A.

Creator

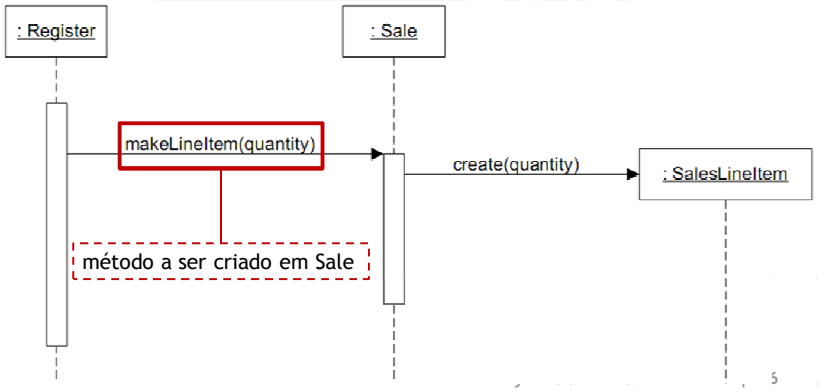
- Que classe deve ser responsável por criar uma instância do objeto SalesLineItem abaixo?



slide com animação

Creator

- A nova responsabilidade é conduzida por uma operação em um diagrama de interações
  - Um novo método é criado na classe de design para expressar isto.



slide com animação

Low Coupling

- Problema
  - Como suportar baixa dependência, baixo impacto devido a mudanças e reuso constante?
- Solução
  - Atribuir uma responsabilidade para que o acoplamento mantenha-se fraco.

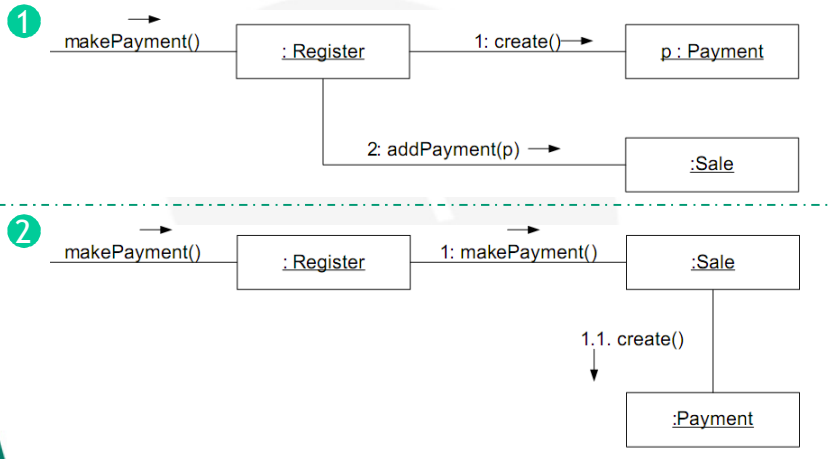
Acoplamento

- É uma medida de quanto um elemento está conectado a, ou depende de outros elementos.
- Uma classe com acoplamento forte depende de muitas outras classes.
  - Por isto, acoplamento forte é indesejável
- O acoplamento está associado à coesão:
  - Quanto maior a coesão, menor o acoplamento e vice-versa.

Low Coupling

- Como devemos atribuir uma responsabilidade para criar Payment e associá-lo com Sale?

- Qual das opções abaixo possui menor acoplamento?



High Cohesion

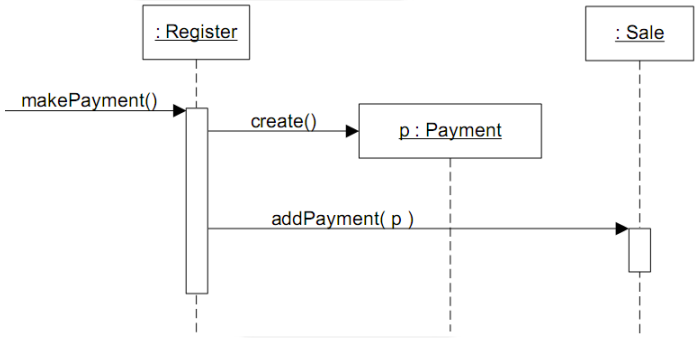
- **Problema**
  - Como manter a complexidade sob controle?
  - Classes que fazem muitas tarefas não relacionadas são:
    - mais difíceis de entender,
    - de manter e de reusar,
    - além de serem mais vulneráveis à mudança.
- **Solução**
  - Atribuir uma responsabilidade para que a coesão se mantenha alta.

Coesão

- Coesão [Funcional]
  - Uma medida de quão relacionadas ou focadasestão as responsabilidades de um elemento.
  - Exemplo: uma classe Cão é coesa se:
    - tem operações relacionadas ao Cão (morder, correr, comer, latir)
    - e apenas ao Cão (não terá por exemplo, validar, listarCaes, etc)
- Alta coesão promove design modular

High Cohesion

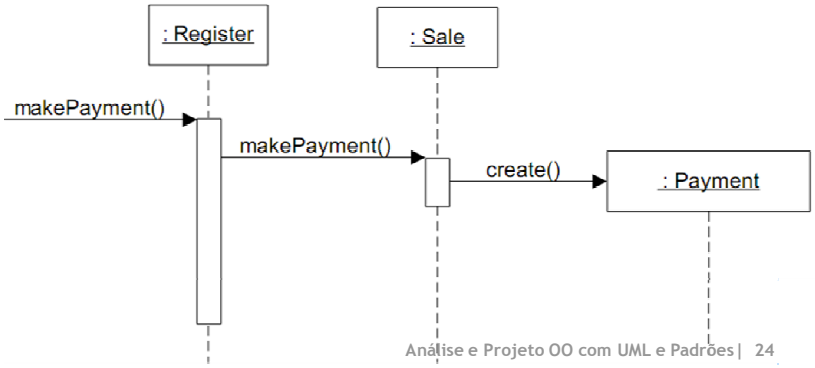
- Que classe é responsável por criar um pagamento (Payment) e associá-lo a uma venda (Sale)?



Register assumiu responsabilidade por uma coisa que é parte de Sale (fazer um pagamento não é responsabilidade de registrar)

High Cohesion

- Faz mais sentido que o pagamento seja parte de Sale
  - E não do registro, como aparecia na solução anterior
- Logo, Register delega a responsabilidade a Sale, diminuindo aumentando a coesão de Register

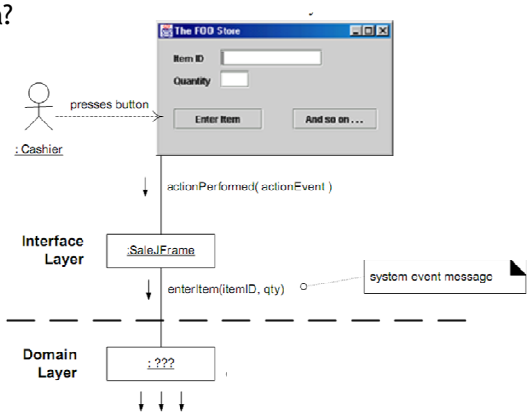


Controller

- **Problema**
  - Quem deve ser o responsável por lidar com um evento de uma interface de entrada?
- **Solução**
  - Atribuir responsabilidades para receber ou lidar com um evento do sistema para:
    - uma classe que representa todo o sistema ou subsistema (façade controller);
    - uma classe que representa cenário de caso de uso (controlador de caso de uso ou de sessão).

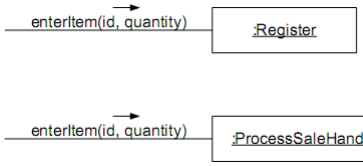
Que padrão GoF a este padrão GRASP ?

- Que classe deve ser responsável por receber eventos do sistema?



Normalmente, o controlador não realiza o trabalho, mas delega para outras subpartes do sistema.

- Possíveis escolhas



  - A primeira solução representa o sistema inteiro
  - A segunda solução representa o destinatário ou handler de todos os eventos de um caso de uso
- A escolha dependerá de outros fatores, que veremos a seguir.

Padrões GRASP avançados

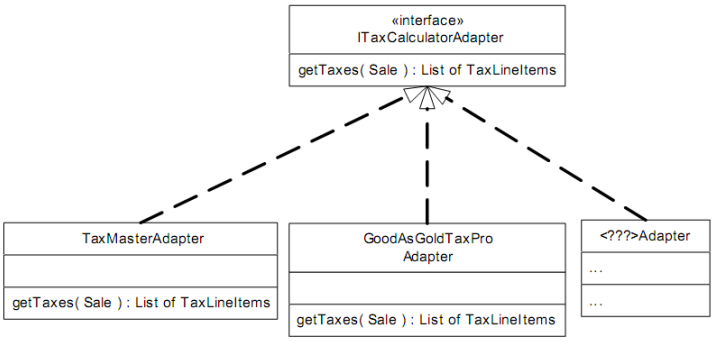
- Domine primeiro os cinco básicos antes de explorar estes quatro:
  - Polymorphism
  - Indirection
  - Pure Fabrication
  - Protected Variations

Polymorphism

- **Problema**
  - Como lidar com alternativas baseadas no tipo? Como criar componentes de software plugáveis?
  - Deseja-se evitar variação condicional (if-then-else): pouco extensível.
  - Deseja-se substituir um componente por outro sem afetar o cliente.
- **Solução**
  - Não use lógica condicional para realizar alternativas diferentes baseadas em tipo.
  - Atribua responsabilidades ao comportamento usando operações polimórficas
  - Refatore!

Exemplo

- No nosso sistema, possuímos várias calculadores de taxas, cujo comportamento varia de acordo seu tipo.
- Através de polimorfismos, podemos criar vários adaptadores semelhantes (mesma interface), que recebem uma venda e adaptam para diferentes calculadoras externas.

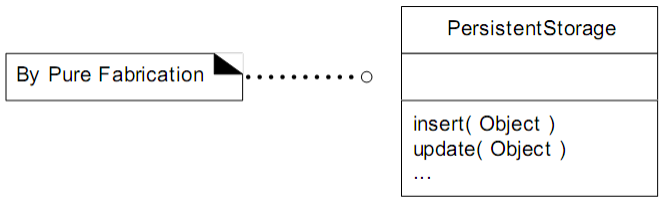


Pure Fabrication

- **Problema**
  - Que objeto deve ter a responsabilidade, quando você não quer violar High Cohesion e Low Coupling, mas as soluções oferecidas por Expert não são adequadas?
  - Atribuir responsabilidades apenas para classes do domínio conceitual pode levar a situações de maior acoplamento e menos coesão.
- **Solução**
  - Atribuir um conjunto altamente coesivo de responsabilidades a uma classe artificial que não representa um conceito do domínio do problema

Exemplo

- Apesar de Sale ser a candidata lógica para ser a Expert para salvar a si mesma em um banco de dados, isto levaria o projeto a ter baixo acoplamento, alta coesão e baixo reuso.
- Uma solução seria criar uma classe responsável somente por isto.

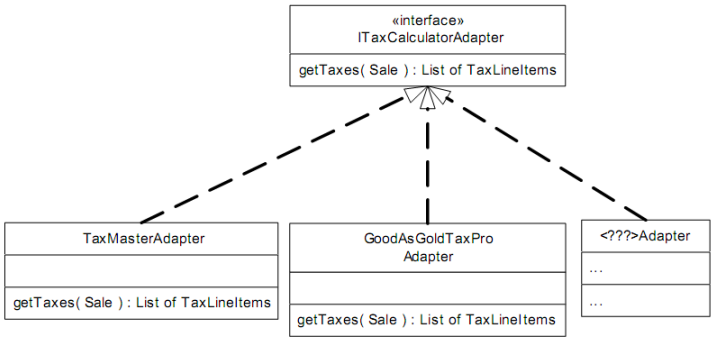


Protected Variations

- **Problema**
  - Como projetar objetos, subsistema e sistemas para que as variações ou instabilidades nesses elementos não tenha um impacto indesejável nos outros elementos?
- **Solução**
  - Identificar pontos de variação ou instabilidade potenciais.
  - Atribuir responsabilidades para criar uma interface estável em volta desses pontos.
  - Encapsulamento, interfaces, polimorfismo, indireção e padrões; máquinas virtuais e brokers são motivados por este princípio
  - Evite enviar mensagens a objetos muito distantes.

Exemplo

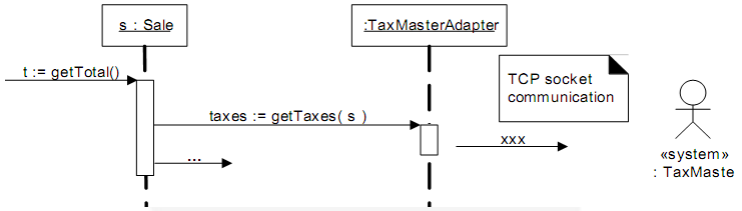
- O exemplo anterior das calculadores de taxa são um exemplo de proteção à variações.
- O sistema deve estar protegido à variações externas dos sistemas de calculadora.



Indirection

- **Problema**
  - Onde atribuir uma responsabilidade para evitar acoplamento direto entre duas ou mais coisas?
  - Como desacoplar objetos para que seja possível suportar baixo acoplamento e manter elevado o potencial de reuso?
- **Solução**
  - Atribua a responsabilidade a um objeto intermediário para mediar as mensagens entre outros componentes ou serviços para que não sejam diretamente acoplados.
  - O objeto intermediário cria uma camada de indireção entre os dois componentes que não mais dependem um do outro:
    - agora ambos dependem da indireção.

- No exemplo da calculadora de taxas, o sistema e a calculadora são intermediados pelo adaptador.



- O exemplo de persistência (PersistentStorage) também suporta indireção.

"A maioria dos problemas em ciência da computação podem ser resolvidos por um outro nível de indireção." Dijkstra