GRASP: Designing Objetos com Responsabilidades

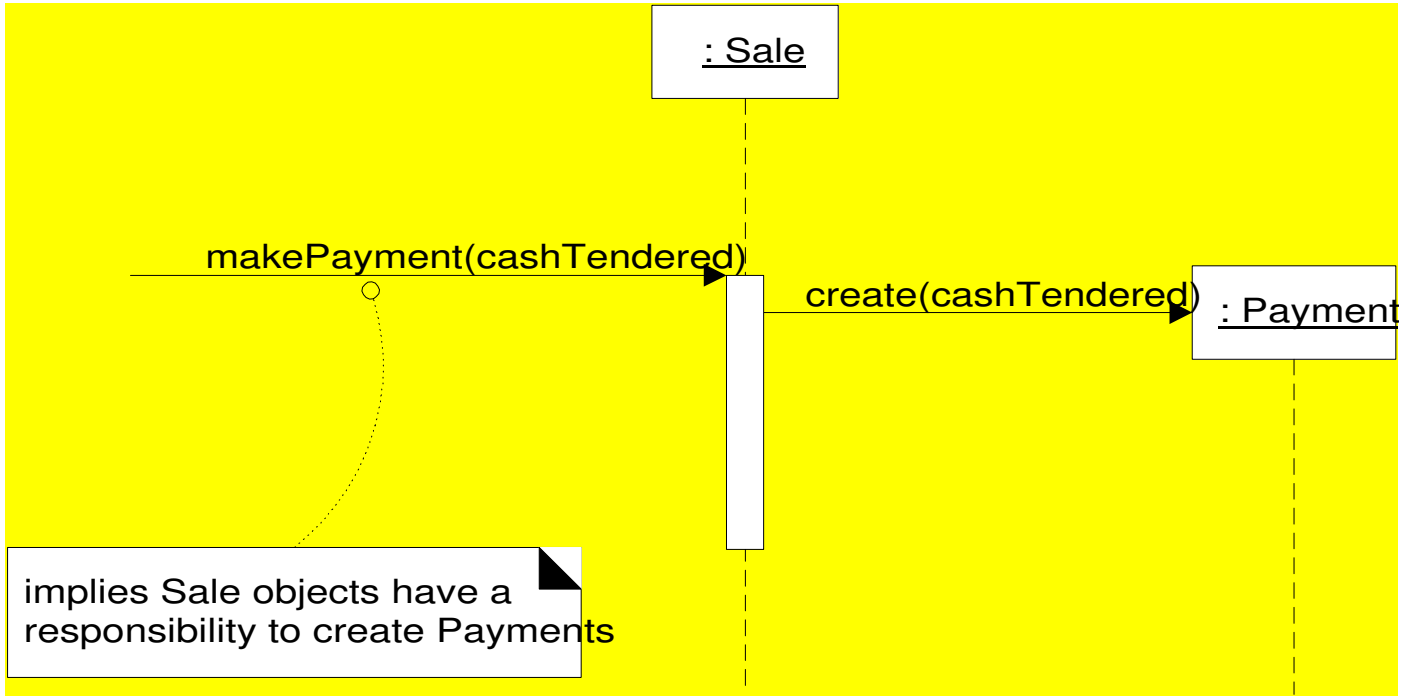
Método para Design de Objetos

- Decidir que métodos pertencem a quem, e como os objetos devem interagir não é trivial ...
- Os padrões **GRASP** são uma ajuda para entender a essência de OD e fazer o design de forma racional
  - Baseado em padrões de atribuição de responsabilidades

Responsabilidades

- São relacionadas com obrigações de um objeto em termos de seu comportamento.
- Responsabilidades do **fazer** de um objeto incluem:
  - Fazer alguma coisa por si [um cálculo, criar um outro objeto]
  - Iniciar ações em outros objetos
  - Controlar e coordenar atividades em outros objetos
- Responsabilidades do **conhecer** de um objeto incluem:
  - Conhecer objetos relacionados
  - Conhecer dados privados e encapsulados

Responsabilidades e métodos são relacionados



Uma *Sale* é responsável por criar *Payments* (um fazer) e conhecer seu total (um conhecer)

Um Padrão

- É um par problema/solução que pode ser aplicado em novas situações, com recomendações sobre como aplicá-lo e discussão de trade-offs. EX.

Problema: Qual é o princípio básico para se atribuir responsabilidades a objetos?

Solução: Atribua responsabilidade à classe que tem a informação necessária

Nome do Padrão: *Information Expert*

O que são padrões GRASP?

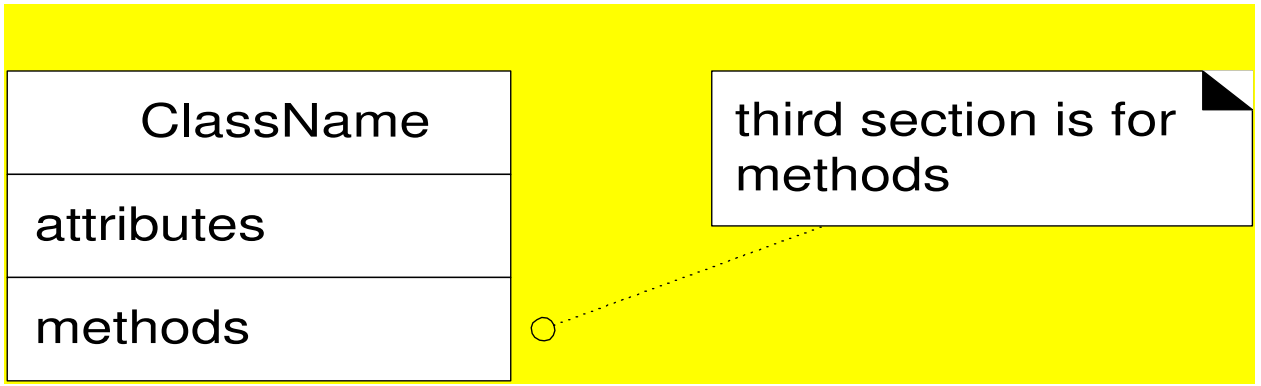
- **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns
- Eles descrevem princípios fundamentais de OD e atribuição de responsabilidades, expressos como padrões
- A atribuição habilidosa de responsabilidades é extremamente importante em OD
- Formam as bases de como o sistema será projetado
- A atribuição de responsabilidades ocorre durante a criação de diagramas de interação

Os 5 primeiros Padrões

- 1. Information Expert
- 2. Creator
- 3. High Cohesion
- 4. Low Coupling
- 5. Controller

Aplicá-los durante a criação de novos diagramas de interação

A Notação UML para Diagrama de Classes



P1: Information Expert

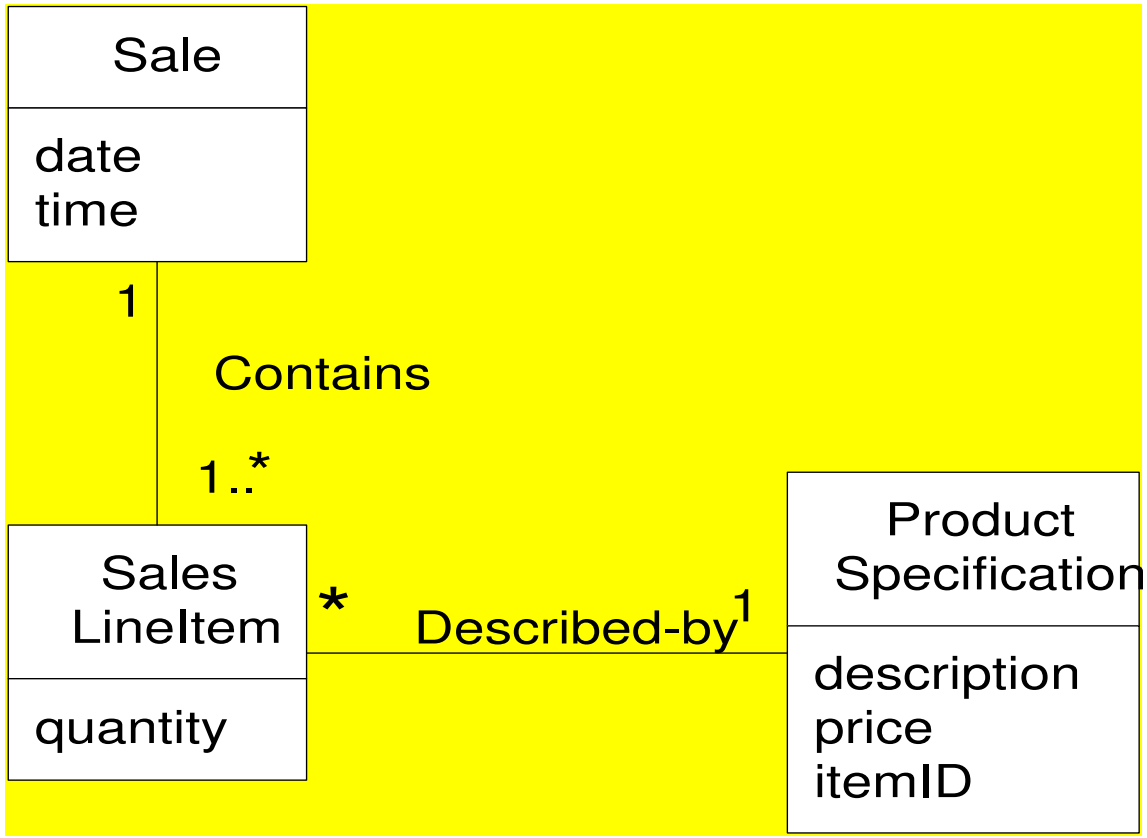
Problema que resolve: Qual é o princípio geral para atribuição de responsabilidades a objetos?

Solução: Atribua uma responsabilidade à classe que tem a informação necessária

Exemplo: Na aplicação POS, algumas classes necessitam conhecer o total geral de *Sale*

Quem deveria ser responsável por conhecer o total geral de sale?

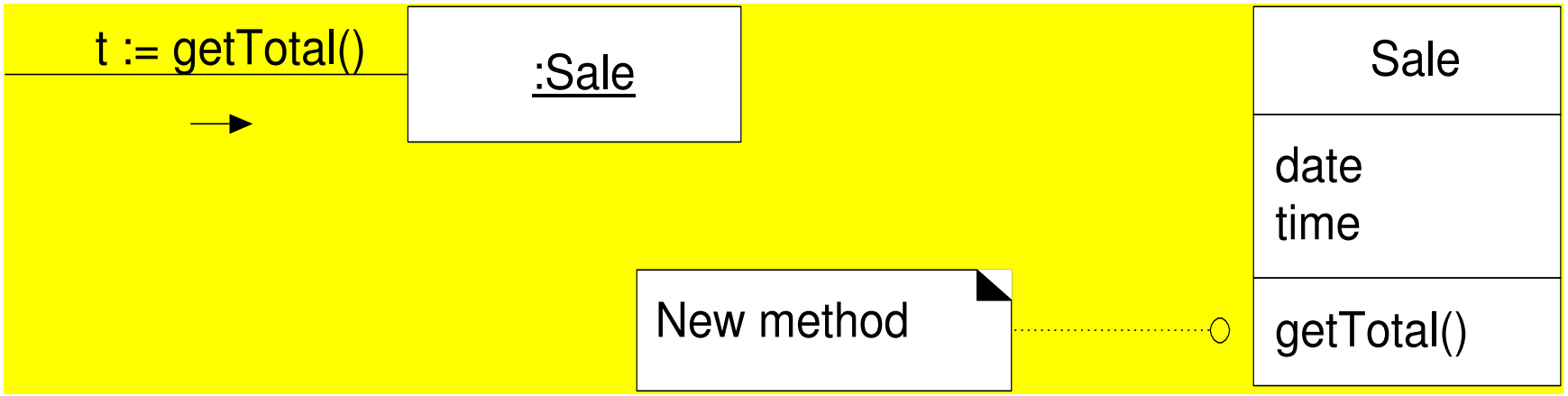
Associações de Sale (DM)



Que informação é necessária para determinar o total geral?

- É necessário conhecer todas as instâncias de *SalesLineItem* de uma *Sale* e a soma de seus sub-totais.
- Uma instância de *Sale* contem esses elementos.
- *Sale* é uma classe de objetos adequada para esta responsabilidade.
- Ela é uma **Information Expert** para esse trabalho

Diagramas parciais de Interação e de Classe



Que informação é necessária para determinar o sub-total de line item?

- *SaleLineItem.quantity* e *ProductSpecification.price*

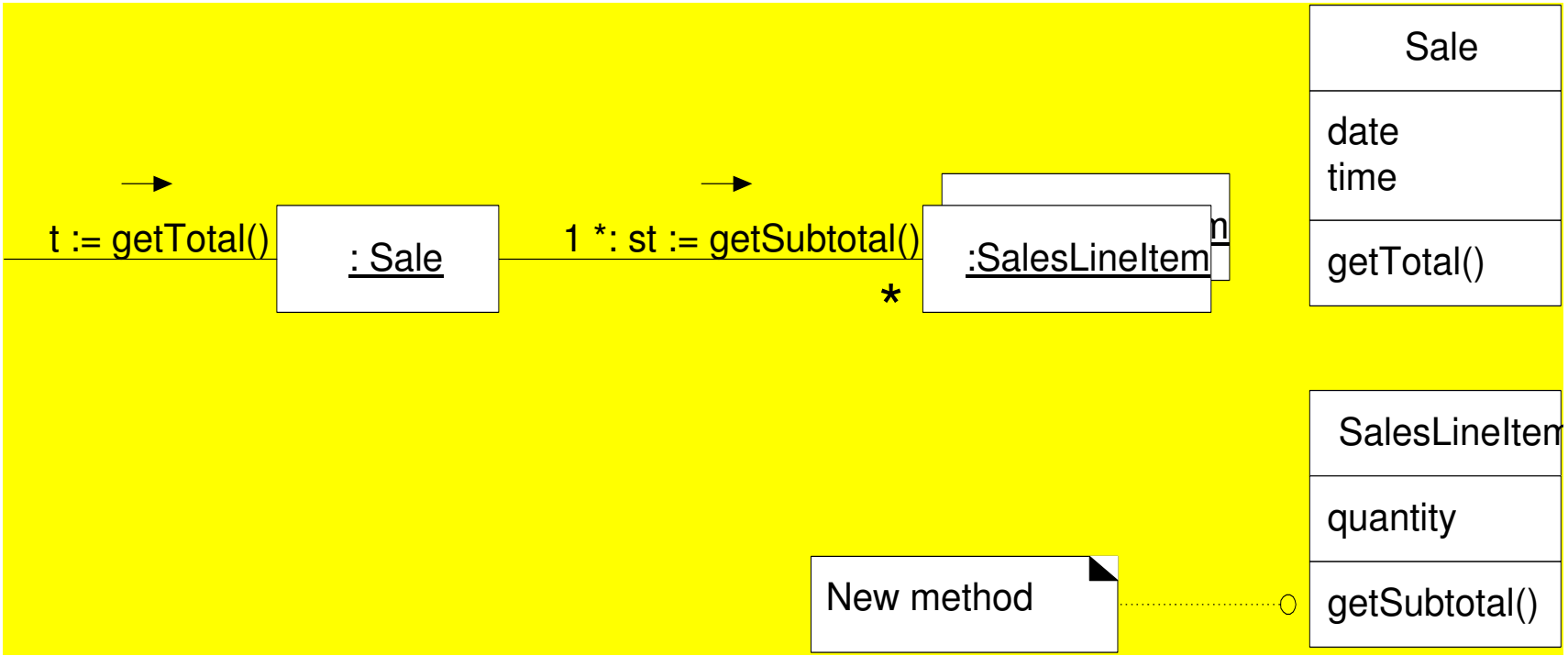
- O *SalesLineItem* conhece sua *quantity* e seu *ProductSpecification a*ssociado

=> by Expert

SalesLineItem deveria determinar o sub-total [ela é a Information Expert]

- Em termos de um Diagrama de Interação, isto significa que o *Sale* necessita enviar mensagem *getSubtotal para* cada um dos *SaleLineItems* e somar os resultados

Calculando o total de Sale



- Para responder à responsabilidade de conhecer e responder por seu subtotal, um *SalesLineItem* necessita conhecer o preço do produto
- O *ProductSpecification* é um *information expert* em responder seu preço.

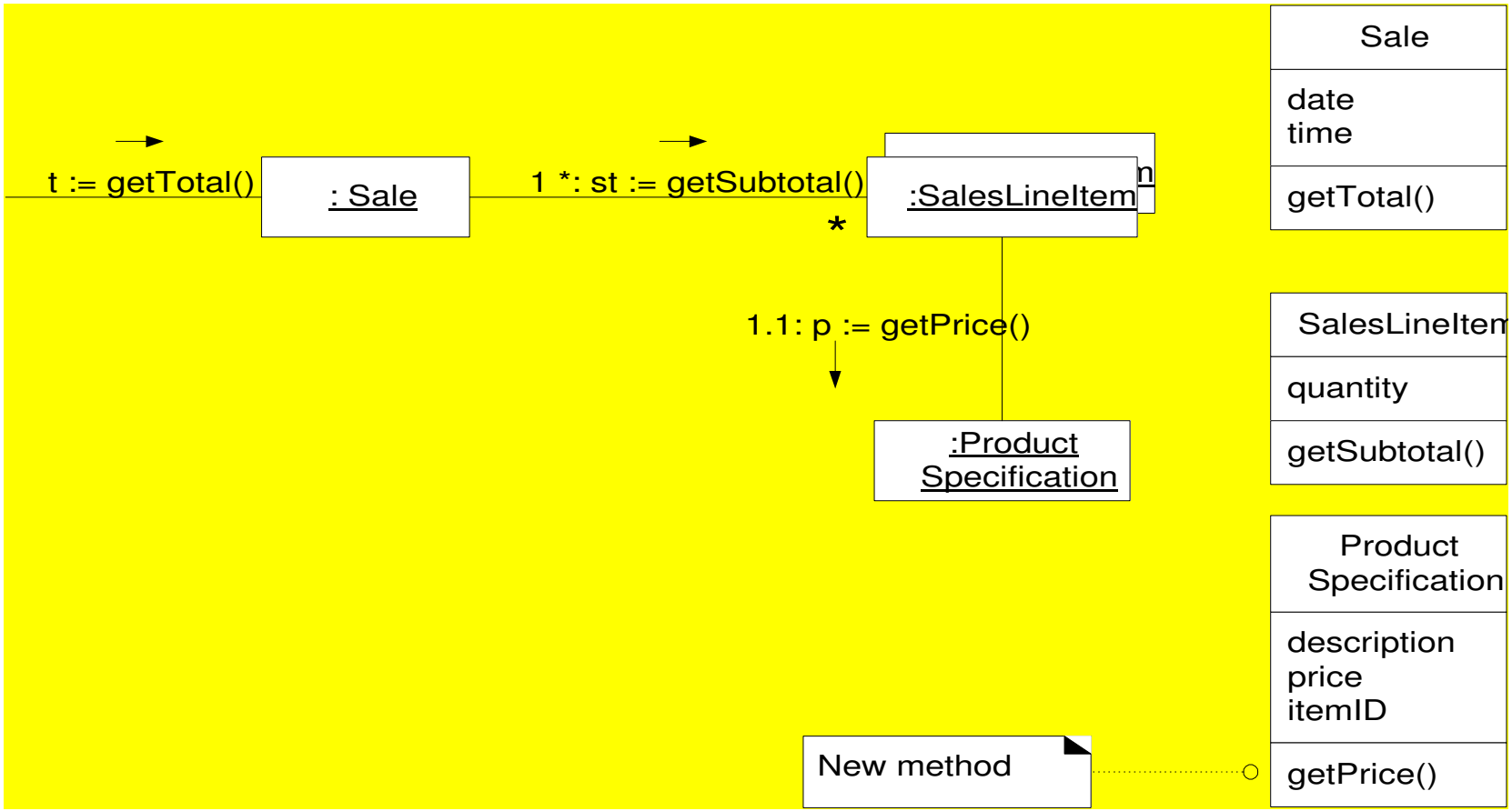
=> uma messagem deveria ser enviada para ele perguntando por seu preço

Concluindo

- Para responder à responsabilidade de conhecer e responder o total de *sale*, 3 responsabilidades foram atribuídas a 3 classes de design de objetos

Design Class	Responsibility
Sale	Knows sale total
SalesLineItem	Knows line item sub total
ProductSpecification	Knows product price

Calculando o total de Sale



Discussão sobre P1 – Information Expert

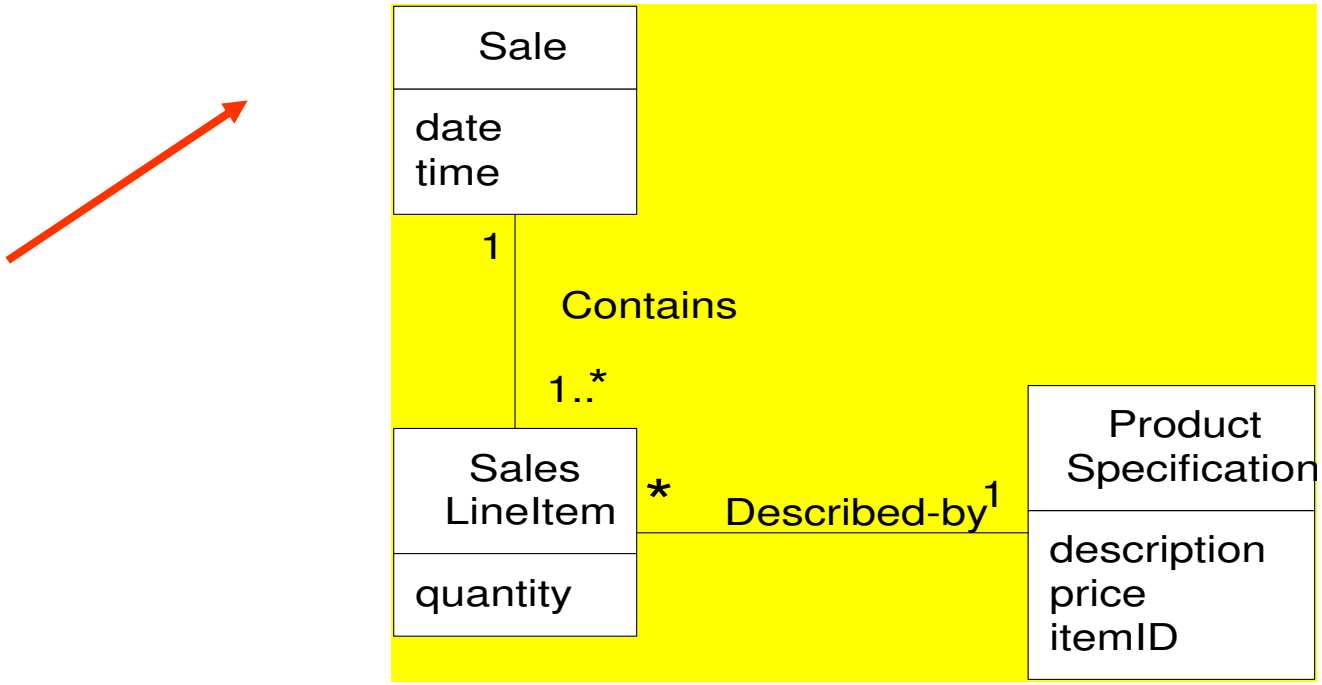
- ***Information Expert*** é um princípio básico usado para guiar continuamente o design orientado a objetos
- A resposta a uma responsabilidade geralmente requer informação que está espalhada em diferentes classes de objetos
  - Isso significa que há muitos information experts "parciais" que colaborarão
  - Eles precisarão interagir via mensages para compartilhar o trabalho

P2: Creator

- **Problema:** Quem deveria ser responsável por criar uma nova instância de alguma classe?

- **Solução:** Atribua à classe B a responsabilidade de criar uma instância da classe A se uma ou mais das seguintes afirmações forem Verdadeiras:
  - B agrega objetos de A. B contém objetos de A. B registra instâncias de objetos de A. B usa objetos de A. B tem os dados de inicialização que serão passados a A qdo ela for criada
- **Exemplo:** *Na aplicação POS, quem deve ser responsável por criar uma instância de SalesLineItem?*

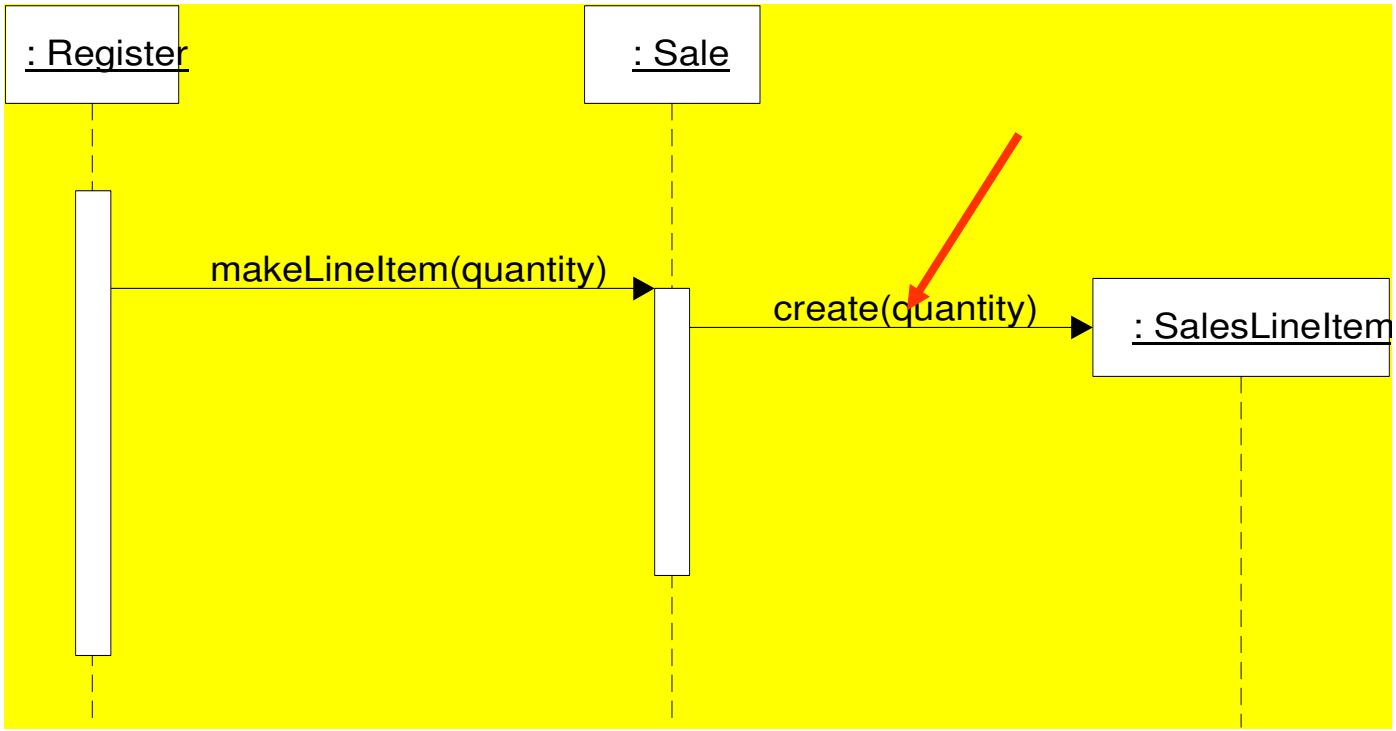
Modelo de Domínio Parcial



Pelo *Creator* deveríamos olhar para a classe que agrega, contém, etc instâncias de *SalesLineItem*

Criando uma SalesLineItem

Isto leva ao design de interações entre objetos como :



Isto requer que um método *makeLineItem* seja definido em *Sale*

Discussão sobre P2 - Creator

- **Creator** guia a atribuição de responsabilidades relacionadas à criação de objetos
- Agregados *agregam* Partes, Containers *contém* Conteúdo, Recorder *records* Recorded são relações muito comuns entre classes em um diagrama de classes
- **agregação** envolve coisas que estão em relação Parte-Todo
  - Corpo *agrega* Pernas,
  - Parágrafo *agrega* Sentenças

P3: Baixo Acoplamento (Low Coupling)

- **Problema:** Como apoiar baixa dependência, baixo impacto em mudança, e aumento de reuso?
- **Solução:** Atribua responsabilidade de modo que acoplamento permaneça baixo
- **Coupling** é uma medida de quão fortemente um elemento é conectado a outro, tem conhecimento de, ou depende de outros.

  Um elemento com acoplamento baixo (ou fraco) não é dependente de muitos outros elementos

Acoplamento

- Uma classe com alto (ou forte) acoplamento sofre dos problemas:
  - Mudanças nas classes relacionadas forçam mudanças locais
  - Mais difíceis de entender isoladamente
  - Mais difíceis de reutilizar pq seu reuso requer a presença adicional das classes das quais ela depende

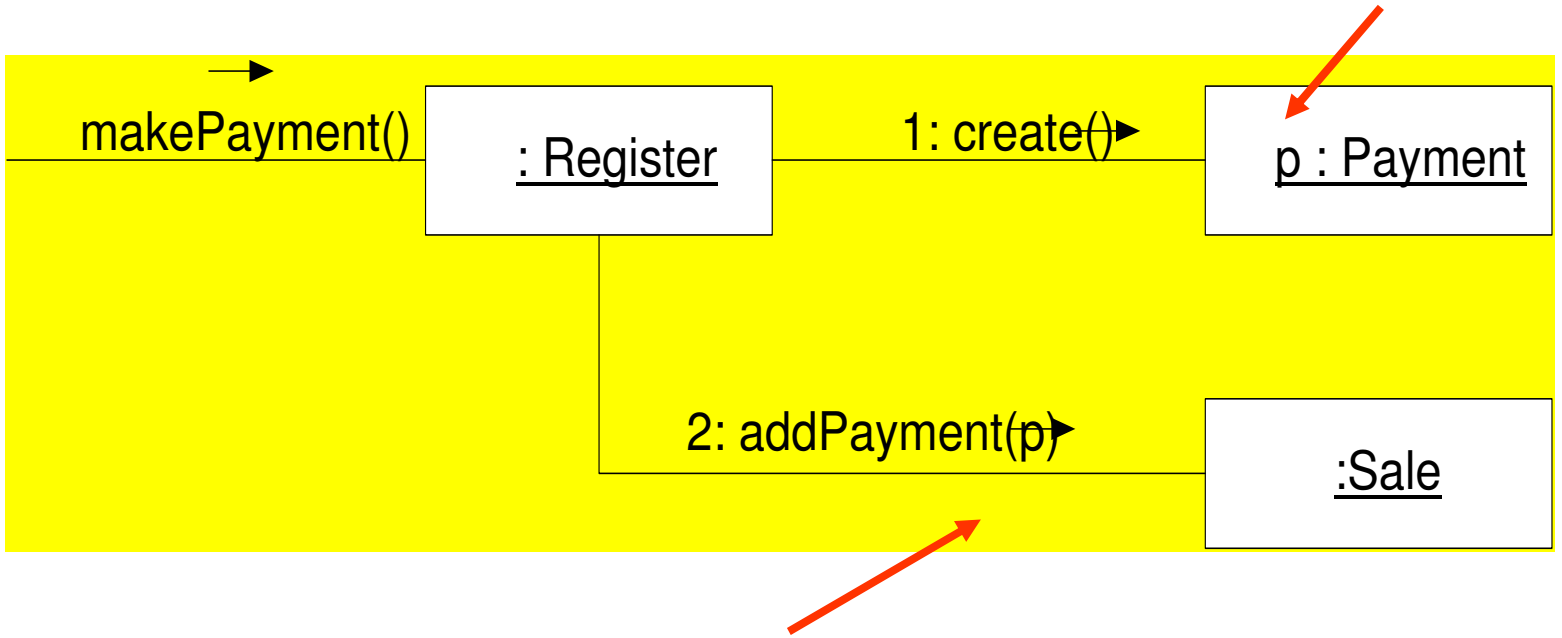
Exemplo de P3 – Baixo Acoplamento

Assuma que temos que criar uma instância de *Payment* e associá-la a *Sale*. Que classe deveria ser responsável por isto?

A *Register* "registra" um *Payment* no domínio.

O Padrão Creator sugere Register como candidata para criar Payment

Register cria Payment



Esta atribuição de responsabilidade **acopla** a classe Register a conhecer a classe Payment [ver mensagem 2]

Sale cria Payment [bom]

Solução alternativa:

Sale deve ser naturalmente acoplada ao conhecimento de Payment. Esta solução **não** aumenta acoplamento

Discussão sobre P3 – Baixo Acoplamento

- **Baixo acoplamento** é um princípio para ter em mente durante as decisões de design
- É um princípio avaliativo que o designer aplica avaliando decisões
- Apoia o design de classes mais independentes, reduzindo o impacto de mudanças
- Uma subclasse é fortemente acoplada à sua superclasse [precisa ser considerado cuidadosamente]
- O caso extremo de baixo acoplamento é qdo não há acoplamento entre classes [isto **não** é desejável]

P4 – Alta Coesão

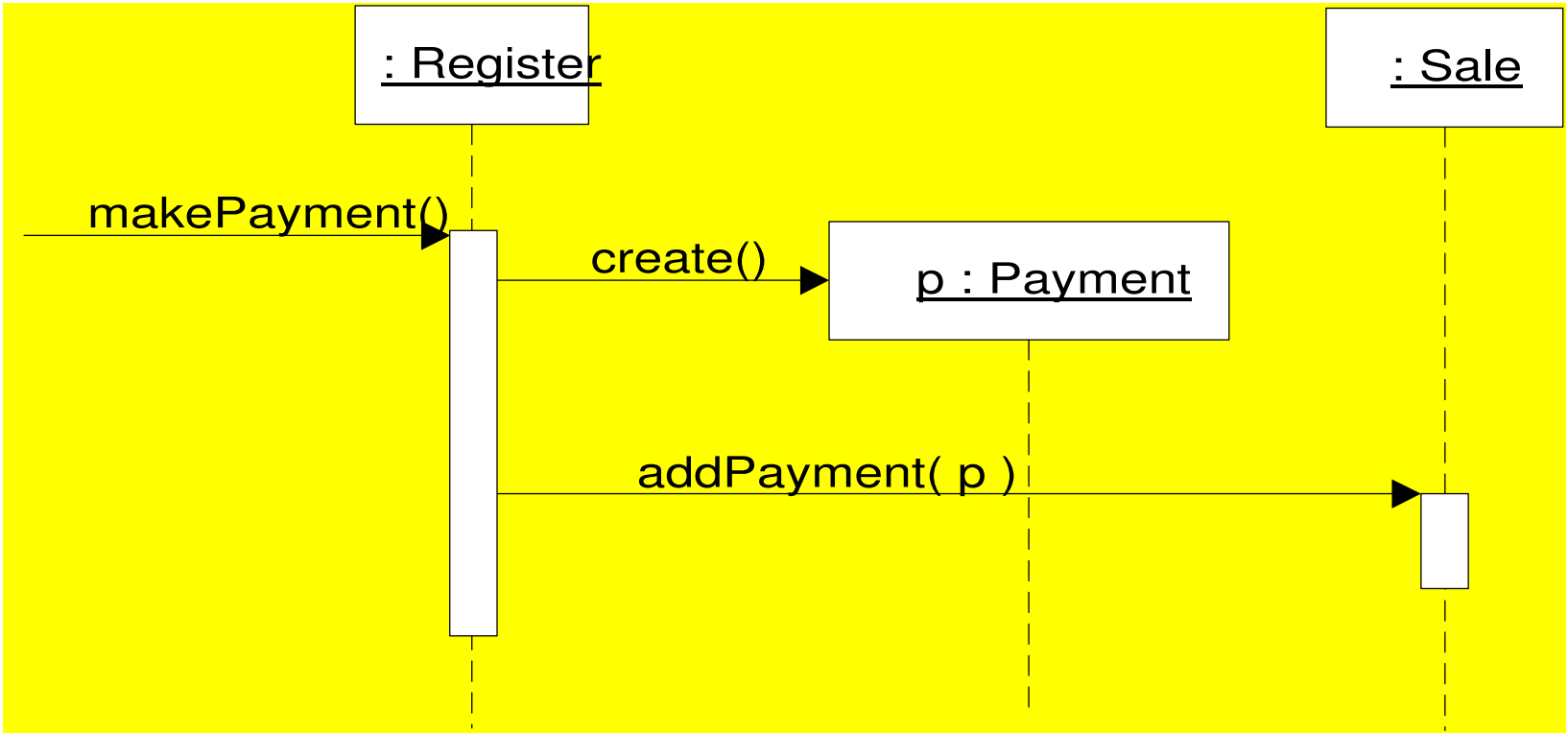
- **Problema:** Como manter a complexidade gerenciável?
- **Solução:** Atribua responsabilidades de modo que coesão permaneça alta
- **Coesão** [funcional] é uma medida de quão fortemente relacionadas e focadas são as responsabilidades de um elemento
- Um elemento com responsabilidades relacionadas e que não fazem muito trabalho têm alta coesão [classes, subsistemas, etc.]

Coesão

- Uma classe com baixa coesão faz muitas coisas não relacionadas, ou faz muito trabalho
- Tais classes com baixa coesão sofrem dos seguintes problemas:
  - Difíceis de compreender; difíceis de reutilizar; difíceis de manter; delicadas; constantemente afetadas por mudanças
- **Exemplo:** Assuma que temos que criar uma instância de Payment e associá-la com Sale. Que classe deveria ser responsável por isto?

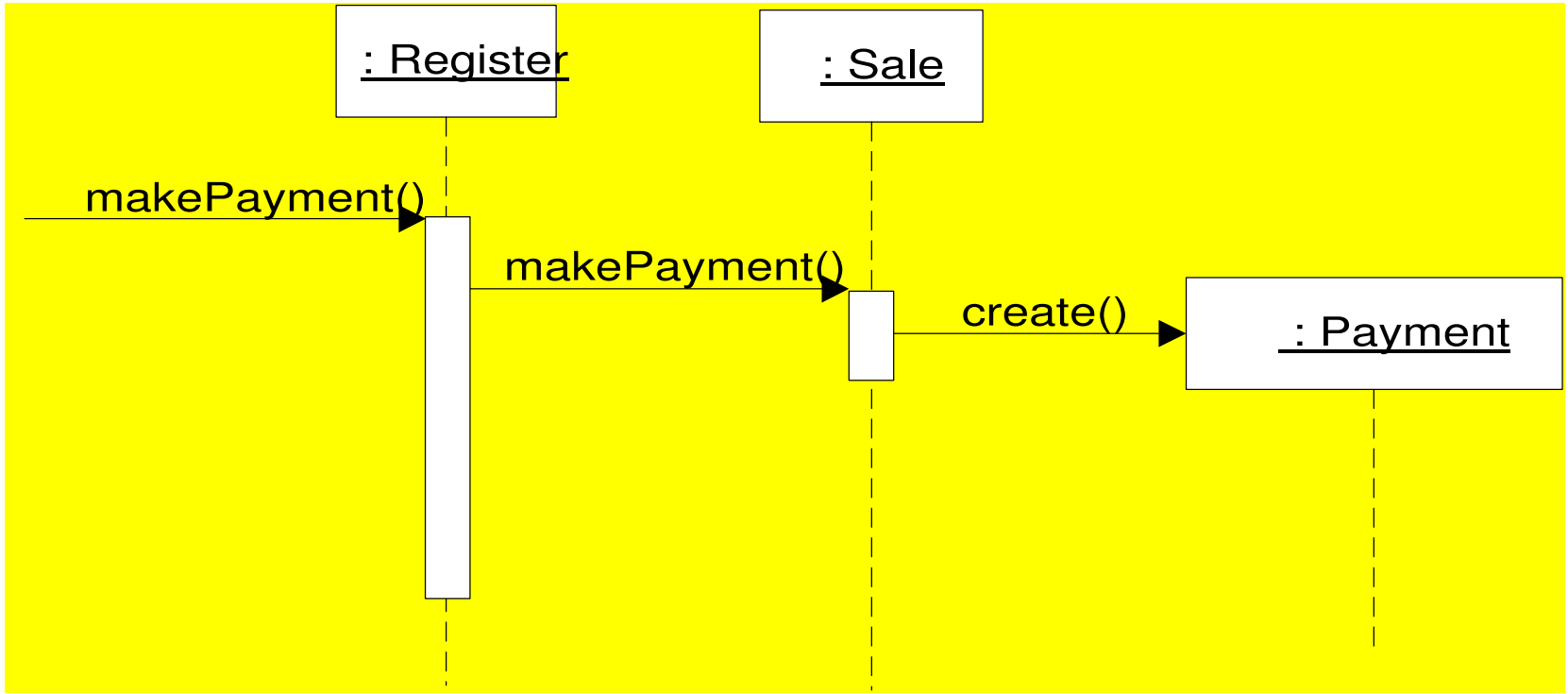
Register cria Payment

Como antes…



- Se continuamos a fazer a classe *Register* responsável por fazer trabalho relacionado a mais que uma operação do sistema, ela se tornará não coesa.
- Por outro lado, delegando a responsabilidade de criação de Payment a *Sale*, apoiará coesão em *Register*
  - E tb apoiará baixo acoplamento

Sale cria Payment

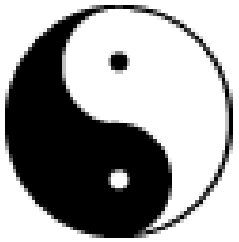


O nível de coesão não deve ser considerado de forma isolada de outros princípios

Discussão sobre P4 – Alta Coesão

- Como Baixo Acoplamento, Alta Coesão é um princípio para ter em mente durante as decisões de design
- É um princípio avaliativo
- Uma classe com alta coesão tem um número relativamente pequeno de métodos, com funcionalidade altamente relacionada, e não faz muito trabalho
- Acoplamento e Coesão são velhos princípios do design de software
- Promove-se **design modular** criando-se métodos e classes com alta coesão

Yin e Yang de SE



Coesão ruim usualmente leva a Acoplamento ruim e vice-versa

Cohesion and coupling *are the yin and yang of software engineering*

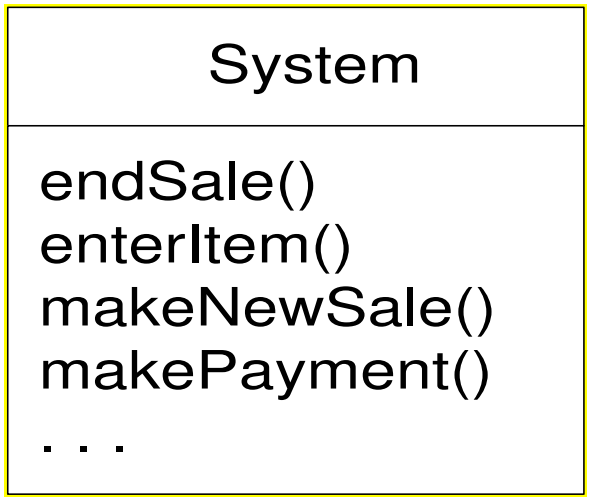
P5 - Controlador

- **Problema:** Quem deveria ser responsável por lidar com um evento de entrada do sistema?
- Um evento de entrada do sistema é um evento gerado por um ator externo
- **Solução:** Atribuir a responsabilidade por receber ou lidar com uma mensagem-evento de entrada do sistema à classe representando:
  - O sistema geral, periférico ou subsistema
  - Um cenário de caso de uso no qual o evento do sistema ocorra

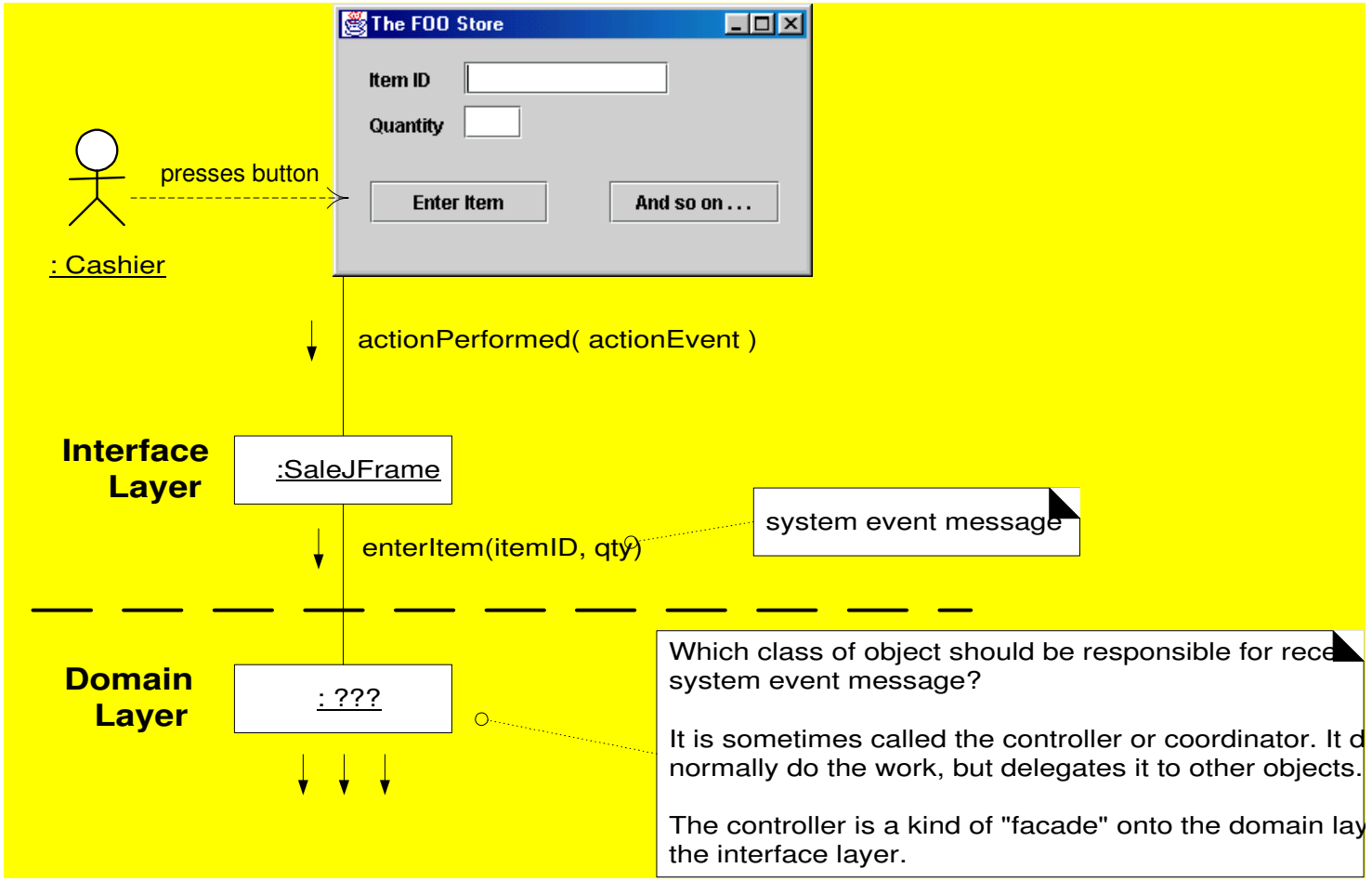
Operações de sistema associadas com os eventos do sistema

Um **Controlador** é um objeto (não-user interface) responsável por receber e lidar com um evento do sistema.

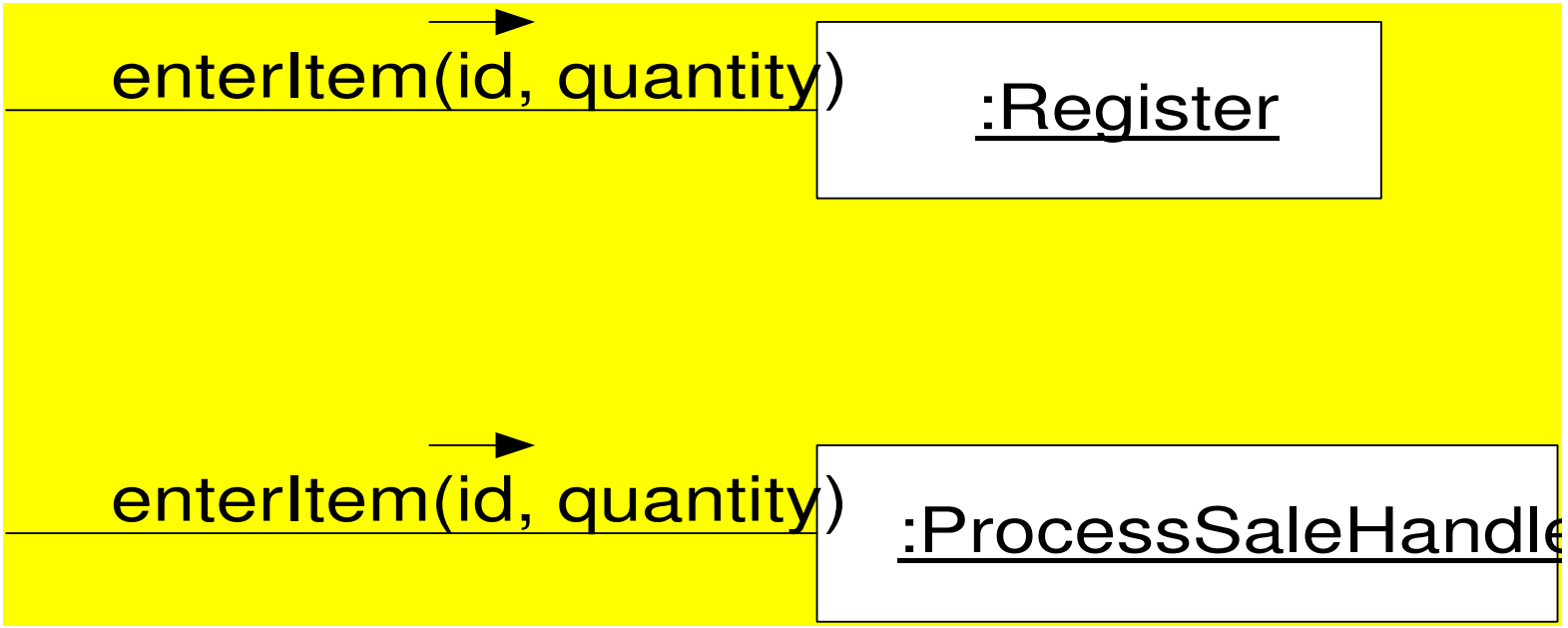
Define o método para operação do sistema



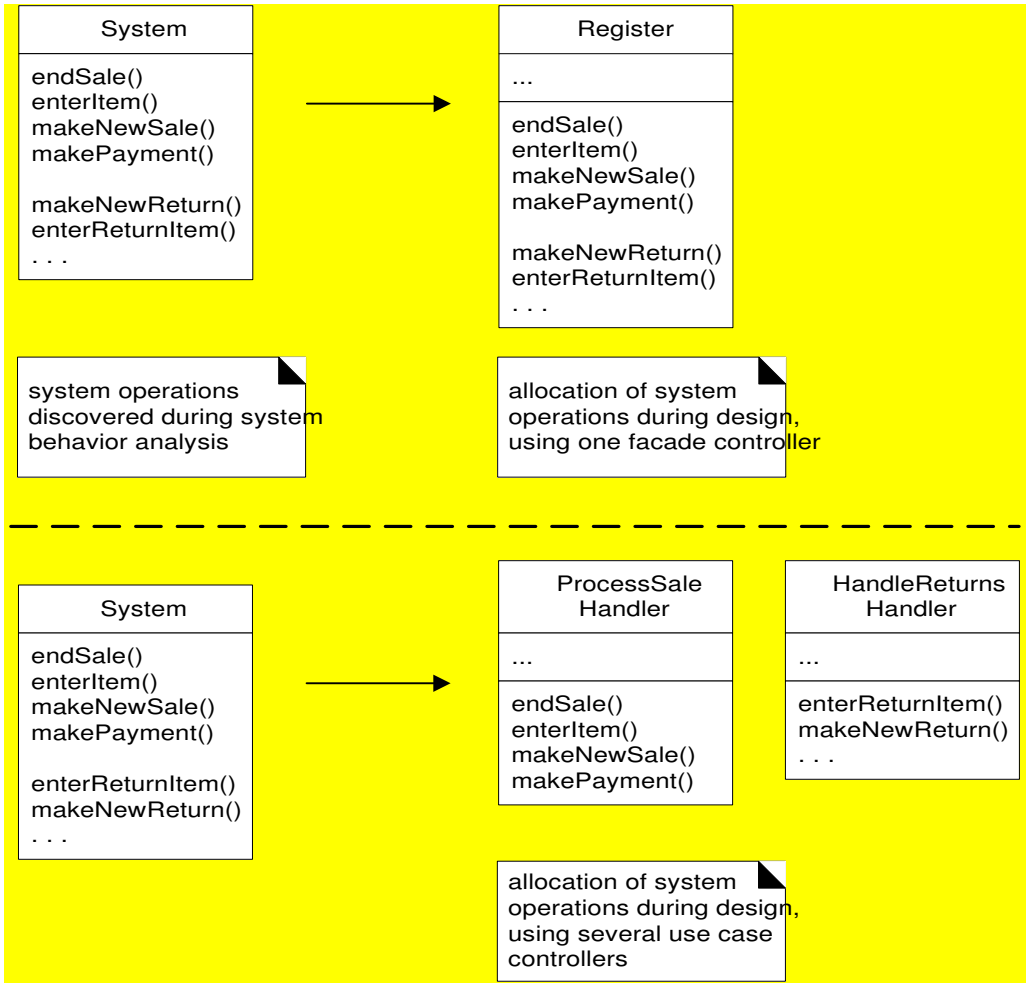
Controlador para enterItem?



Escolhas do Controlador



Alocação de operações do sistema

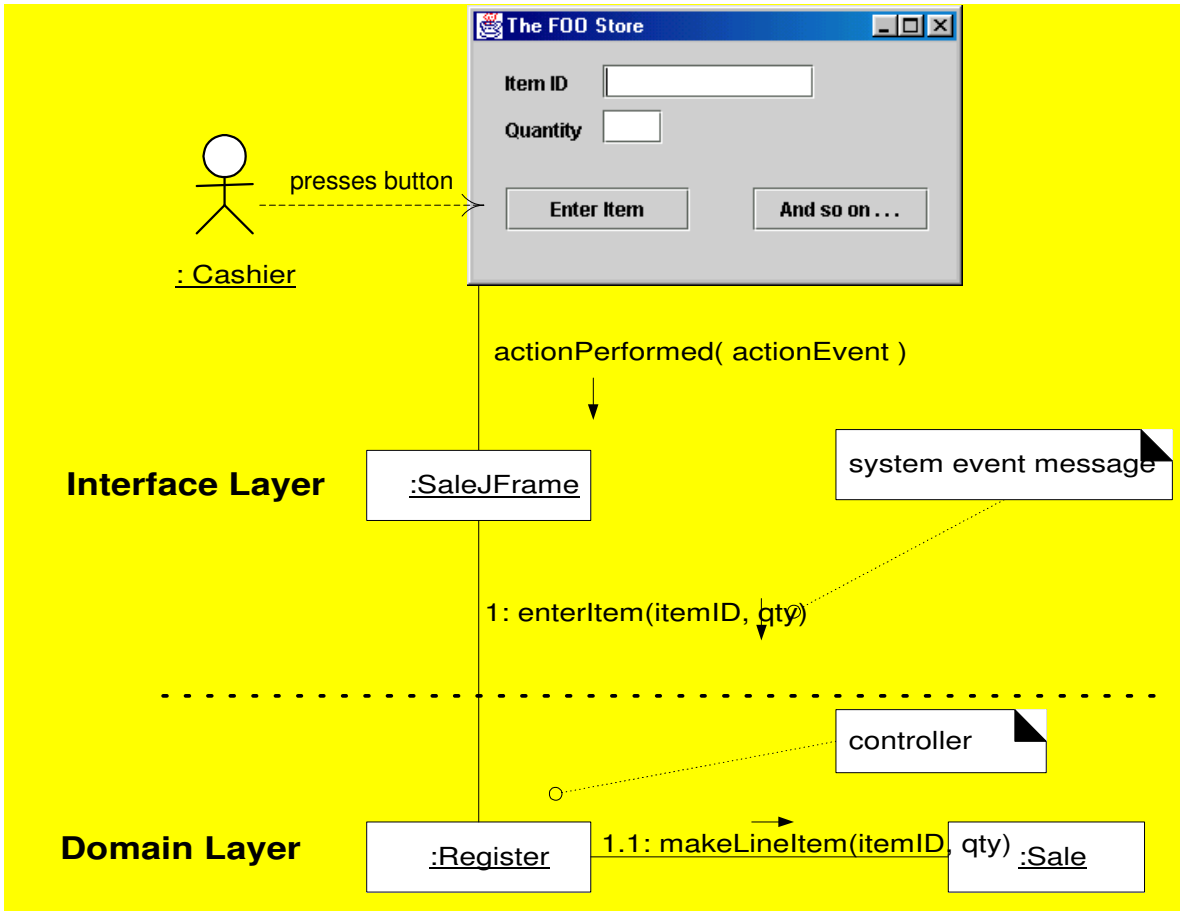


Durante design as operações identificadas do sistema são atribuídas a uma ou mais classes controladoras

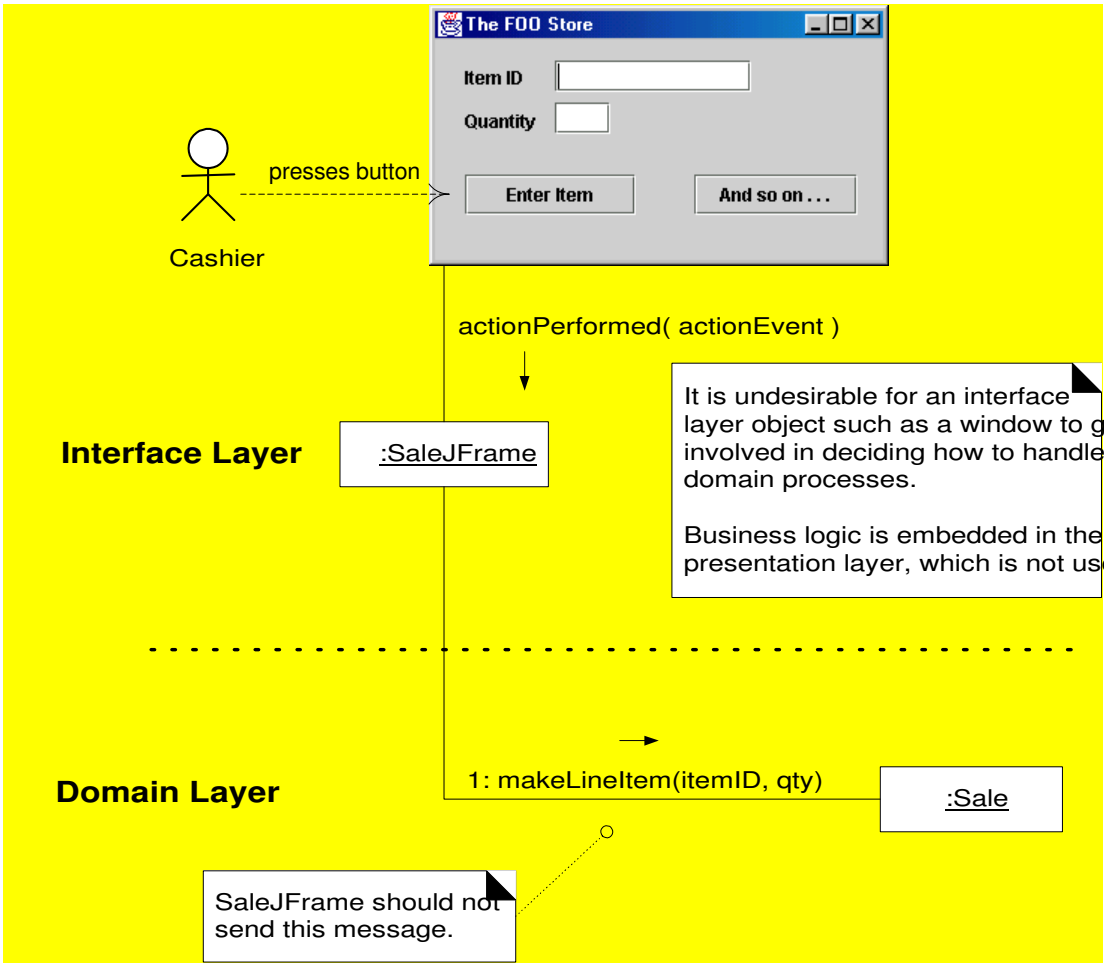
Corolário do Padrão Controller

- Objetos de Interface [ex. objetos janela ou widgets] e a camada de apresentação não deveriam ter responsabilidade por eventos do sistema.
  - Operações do sistema devem ser lidadas na lógica da aplicação ou camada de objetos do domínio e não na camada de interface do sistema
- O Controlador recebe solicitações de serviço da camada de UI e coordena delegando a outros objetos

Acoplamento desejável de camada de interface e camada do domínio



Não desejável...



Ex. uma aplicação com muitos eventos de sistema

Um sistema de reserva aérea pode conter:

Use-case controllers
MakeReservationHandler
ManageSchedulesHandler
ManageFaresHandler

Referências

- Larman, C. (2002) *Applying UML and Patterns – An Introduction to Object Oriented Analysis and Design and the Unified Process*, Prentice-Hall Inc.
- Muller, P.A. (1997) *Instant UML*,Wrox Press Ltd.