
INSTITUTO
VIANNA JUNIOR

INSTITUTO VIANNA JÚNIOR

Análise OO

GRASP

Professor: Camillo Falcão

Introdução

- O projeto orientado a objetos é comumente descrito como alguma variação do seguinte:
  - Após identificar os requisitos e criar o modelo de domínio, adicione métodos às classes e defina as mensagens entre objetos para atender aos requisitos.

- Cuidado: decidir quais métodos criar e como os objetos podem interagir é muito importante, mas não é uma atividade trivial.

GRASP

- GRASP são um conjunto de padrões que nos auxiliam a projetar sistemas orientados a objetos.
- Os padrões GRASP auxiliam a identificação das responsabilidades de cada classe.
- Essa tarefa é crítica para o desenvolvimento de um sistema orientado a objetos.

Responsabilidades

- Responsabilidades estão relacionadas às obrigações de um objeto em termos de seu comportamento.

- Basicamente, as obrigações são dos seguintes tipos:
  - Fazer;
  - Conhecer.

Responsabilidades - Fazer

- As responsabilidades do tipo fazer de um objeto incluem:
  - Fazer algo por ele mesmo. Exemplos:
    - Criar um objeto
    - Calcular algo
  - Iniciar ações em outros objetos
  - Controlar e coordenar atividades em outros objetos.

Responsabilidades - Conhecer

- As responsabilidades do tipo conhecer de um objeto incluem:
  - Conhecer sobre dados privados encapsulados
  - Conhecer objetos relacionados
  - Conhecer coisas em que esse objeto pode derivar ou calcular

Responsabilidades

- As responsabilidades são atribuídas para uma classe de um objeto durante o seu projeto. Exemplo:
  - Uma Venda é responsável por criar os ItensDeVenda (responsabilidade do tipo fazer).
  - Uma Venda é responsável por conhecer o seu total (responsabilidade do tipo conhecer).
- Responsabilidades relevantes do tipo conhecer são frequentemente inferidas à partir do modelo do domínio, por causa dos atributos e associações que esse modelo ilustra.

Responsabilidades

- A tradução de responsabilidades em classes e métodos é influenciada pela granularidade da responsabilidade.
  - “Prover acesso à bases de dados relacionais” pode envolver dezenas de classes e centenas de métodos empacotados em um subsistema.
  - “Criar uma Venda” pode envolver somente um ou poucos métodos.

Responsabilidades

- Uma responsabilidade não é o mesmo que um método, mas métodos são implementados para cumprir responsabilidades.
- Responsabilidades são implementadas usando métodos que agem sozinhos ou em colaboração com outros métodos e objetos. Exemplo:
  - A classe Venda pode definir um ou mais métodos para conhecer o seu total. Vamos chamar esse método de getTotal. Para cumprir essa responsabilidade, a Venda pode colaborar com outros objetos, como enviando a mensagem getSubtotal para cada item.

GRASP

- GRASP: **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns

- São padrões gerais em atribuição de responsabilidades em software.

GRASP

- A habilidade de atribuir responsabilidades é extremamente importante no projeto orientado a objetos.
- A atribuição das responsabilidades frequentemente ocorre durante a criação dos diagramas de interação e, certamente, durante a programação.
- Padrões são pares de problemas/soluções nomeados que sistematizam bons conselhos e princípios frequentemente relacionados à atribuição de responsabilidades.

Padrão Information Expert (ou Expert)

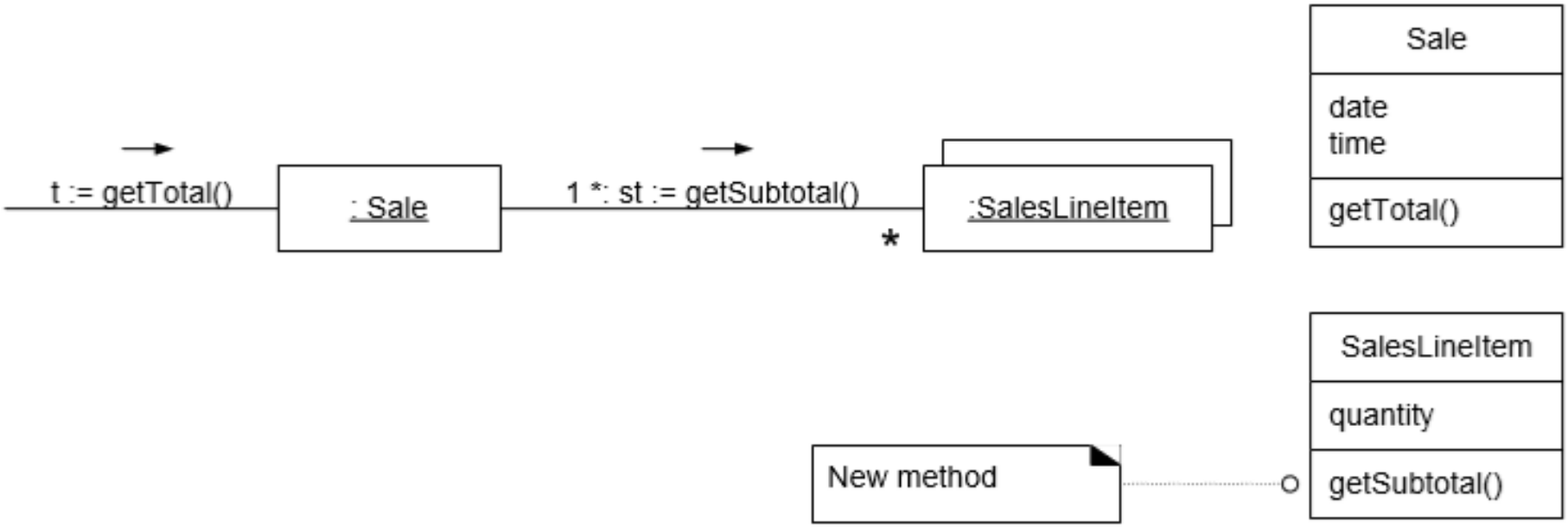
- **Solução**: atribua a responsabilidade à classe que possui a informação necessária para cumprir a responsabilidade.

- **Problema**: qual o princípio geral de atribuição de responsabilidades a objetos?

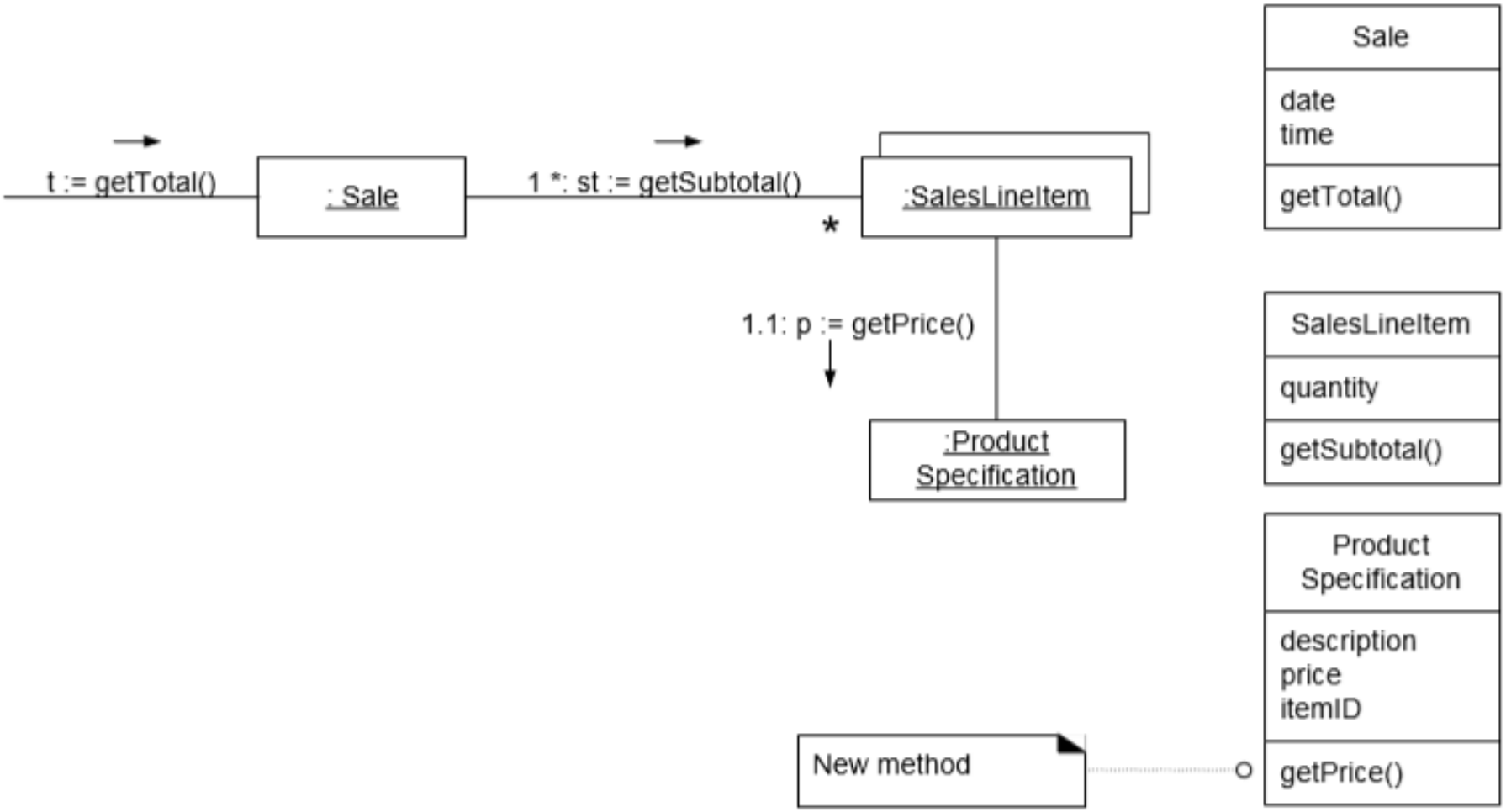
  Um modelo de projeto pode definir centenas ou milhares de classes e uma aplicação pode requerer centenas ou milhares de responsabilidades para serem atribuídas. Durante o projeto do objeto, quando as interações entre os objetos são definidas, nós fazemos escolhas sobre a atribuição de responsabilidades entre as classes. Se isso for bem feito, o sistema tenderá a ser mais fácil de entender, manter e estender, além se ser mais fácil reutilizar componentes em aplicações futuras.

Padrão Information Expert (ou Expert)

Padrão Information Expert (ou Expert)



Padrão Information Expert (ou Expert)



Padrão Information Expert (ou Expert)

- **Contraindicações**: existem situações onde uma solução indicada pelo padrão *Information Expert* é indesejada, usualmente devido a problemas relacionados a acoplamento e coesão.
  - Exemplo: quem deve ser o responsável por salvar uma Venda em uma base de dados?

  Ao contrário do que concluiríamos utilizando o *Information Expert*, essa classe não deve ser a classe Venda, pois isso nos levaria a problemas de coesão, acoplamento e duplicação.

  Considerar o *Information Expert* para esse caso seria uma violação de um princípio arquitetural básico: projete para uma separação dos principais interesses do sistema. De acordo com esse princípio, **devemos manter a lógica da aplicação em um local, a lógica de acesso à base de dados em outro local, e assim sucessivamente.**

Padrão Information Expert (ou Expert)

- **Benefícios**: o encapsulamento das informações é mantido, uma vez que os objetos usam suas próprias informações para cumprir tarefas.
  - Usualmente esse padrão apoia o padrão *Low Coupling* que conduz a um sistema mais robusto e manutenível.

Creator

- **Solução**: atribua à classe B a responsabilidade de criar uma instância da classe A se um ou mais das seguintes situações for(em) verdadeira(s):
  - B agrega objetos do tipo A;
  - B contém objetos do tipo A;
  - B grava objetos do tipo A;
  - B usa de forma próxima e frequente objetos do tipo A;
  - B possui os dados de inicialização que serão passados para o construtor de A no momento de sua instanciação.

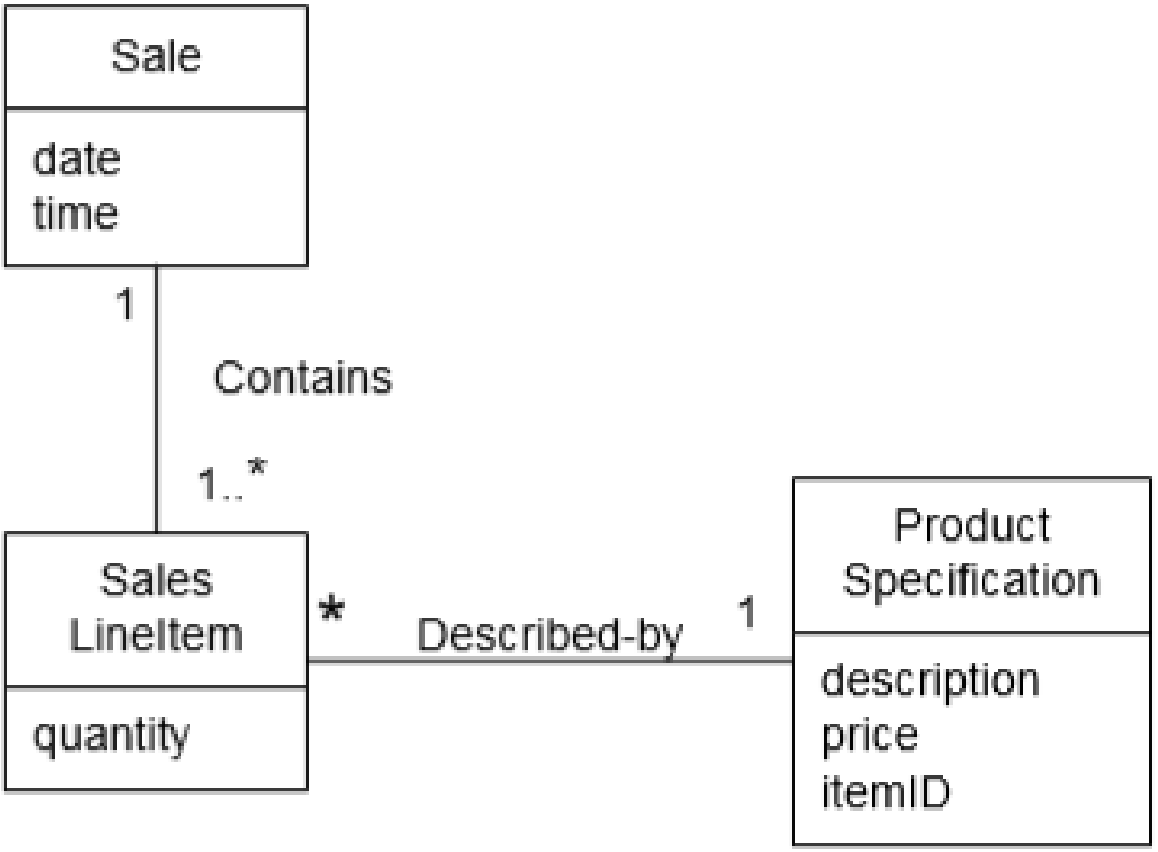
Se mais de uma opção for aplicável, prefira uma classe B que agrega ou contém a classe A.

Creator

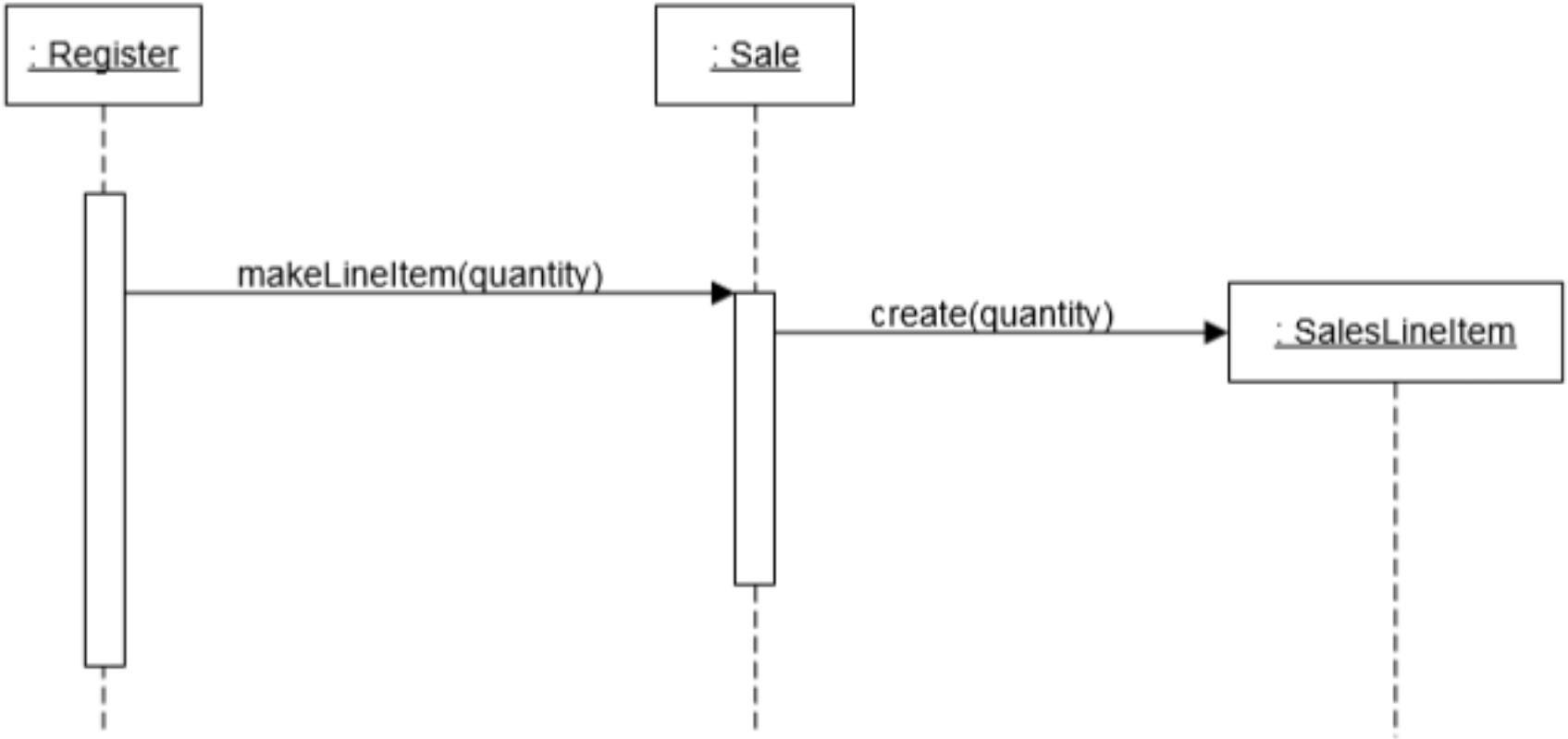
- **Problema**: quem deve ser capaz de criar uma nova instância de uma determinada classe?

  – A criação de objetos é uma das atividades mais comuns em sistemas orientados a objetos. Consequentemente, é útil um princípio geral para atribuir as responsabilidades de instanciação. Se essas responsabilidades forem bem atribuídas, o projeto poderá suportar o baixo acoplamento e aumentará a sua clareza, encapsulamento e reusabilidade.

Creator


Sale
date
time
1
Contains
1..*
Sales
LineItem
quantity
*
Described-by
1
Product
Specification
description
price
itemID

Creator


: Register
: Sale
makeLineItem(quantity)
create(quantity)
: SalesLineItem

Creator

- **Contraindicações**: frequentemente a instanciação de classes exige uma complexidade significante, como quando a criação de uma instância exige a escolha, baseada em alguma propriedade externa, de uma classe entre uma família de classes similares. Para esse caso, pode-se utilizar o padrão de projeto conhecido como *Factory*.

Creator

- **Benefícios**: baixa dependência de manutenção e grandes oportunidades de reuso são suportadas pelo baixo acoplamento. O acoplamento não é aumentado por causa do fato da classe criada já ser visível para a classe criadora, devido às associações existentes que motivaram a escolha da classe criadora.

Low Coupling

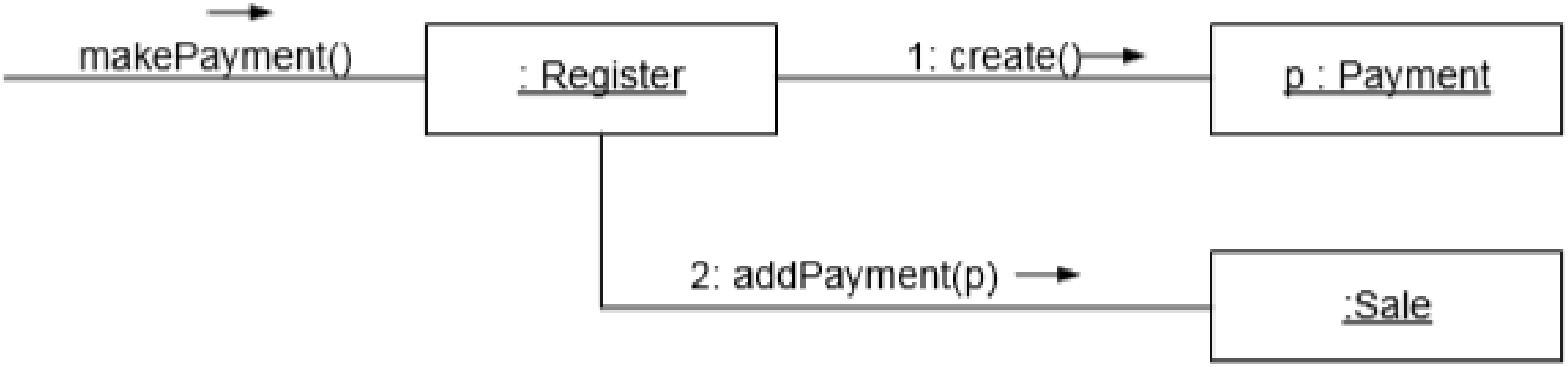
- **Solução**: atribua a responsabilidade de forma que o acoplamento permaneça baixo.

- Acoplamento é uma medida do quão fortemente um elemento é conectado a, conhece, ou depende de outros elementos.
  - Um elemento com baixo acoplamento não depende de muitos outros elementos.
  - O termo elementos inclui classes, subsistemas, sistemas, etc.

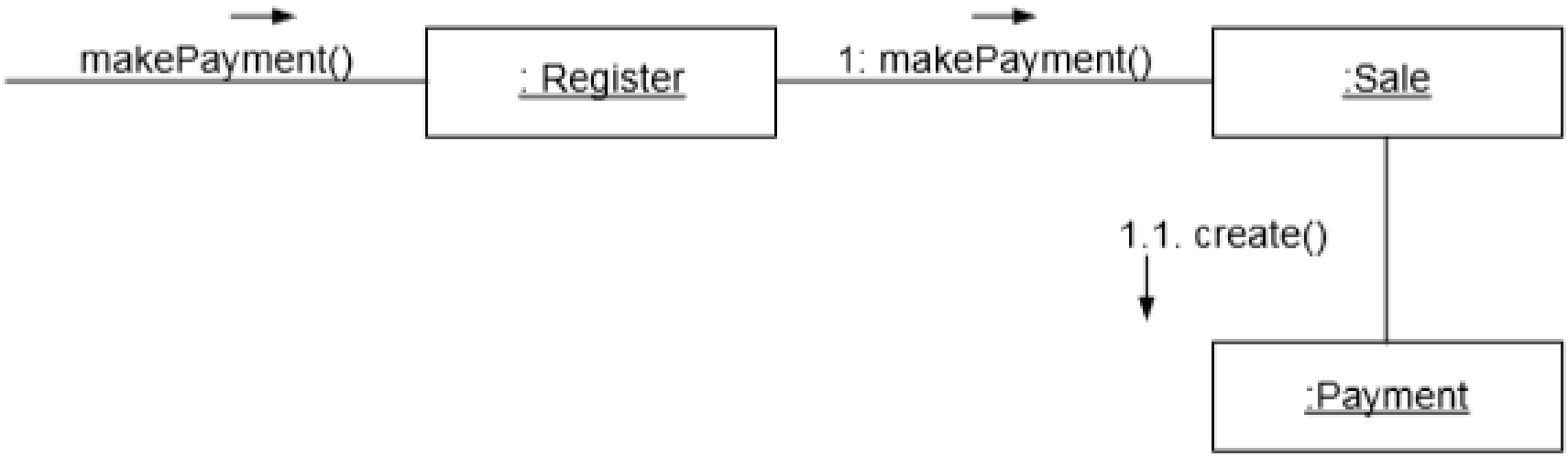
Low Coupling

- **Problema**: como suportar baixa dependência, baixo impacto de mudança e aumentar o reuso?
- Uma classe com alto acoplamento depende de muitas outras classes e isso pode ser indesejado devido a:
  - Mudanças nas classes relacionadas forçarem mudanças locais;
  - Ser difícil entender isoladamente uma classe altamente acoplada;
  - A reutilização ser dificultada por requerer a presença de classes dependentes.

Low Coupling



Low Coupling



Low Coupling

- **Contraindicações**: alto acoplamento com elementos estáveis e altamente difundidos dificilmente é um problema.

- **Benefícios**:
  - Aumenta a chance de elementos não serem afetados por mudanças em outros elementos;
  - Simplifica o entendimento de partes isoladas;
  - É conveniente quando pensa-se em reuso.

High Cohesion

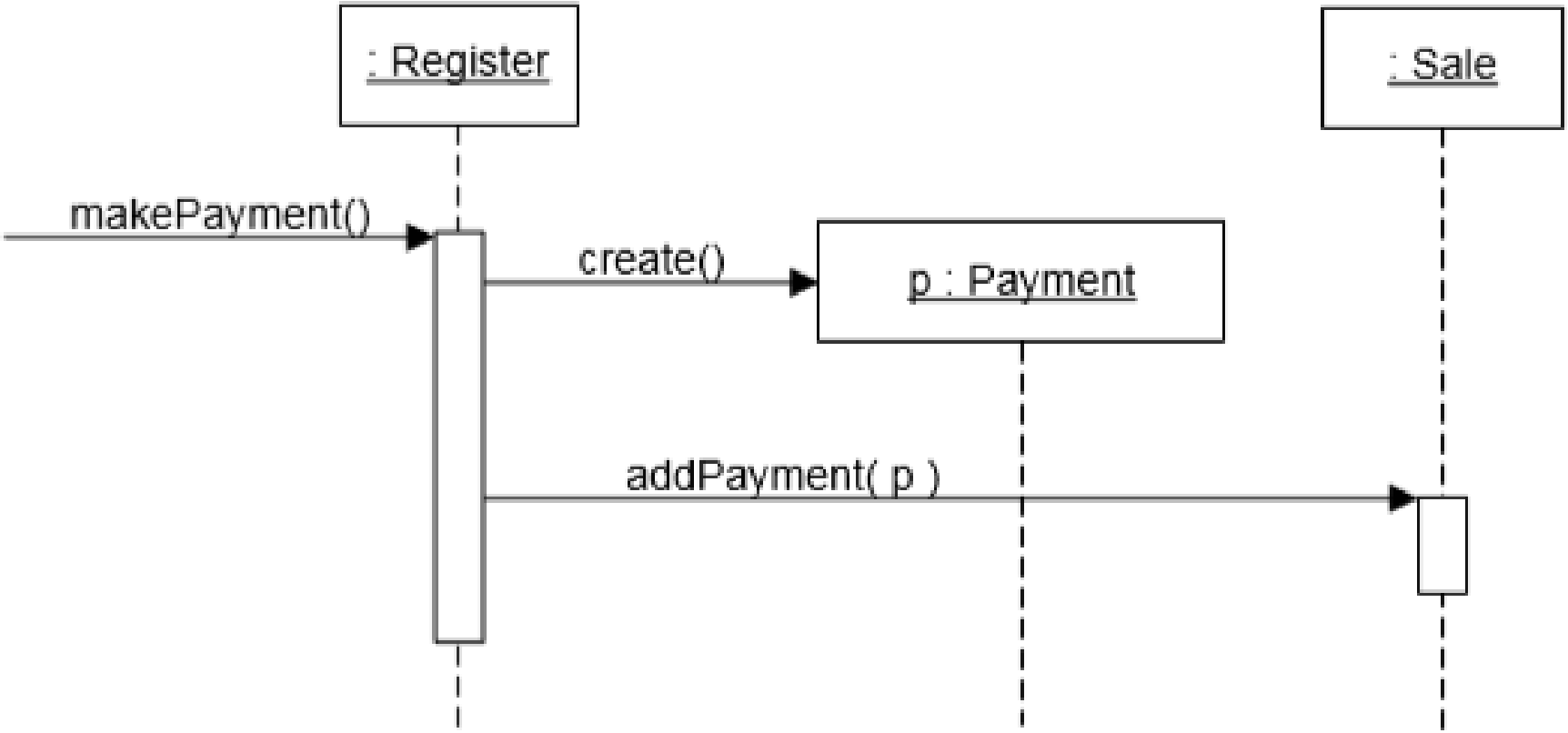
- **Solução**: atribua a responsabilidade de forma que a coesão permaneça alta.

- Coesão é uma medida do quão fortemente relacionadas e focadas as responsabilidades de um objeto são.

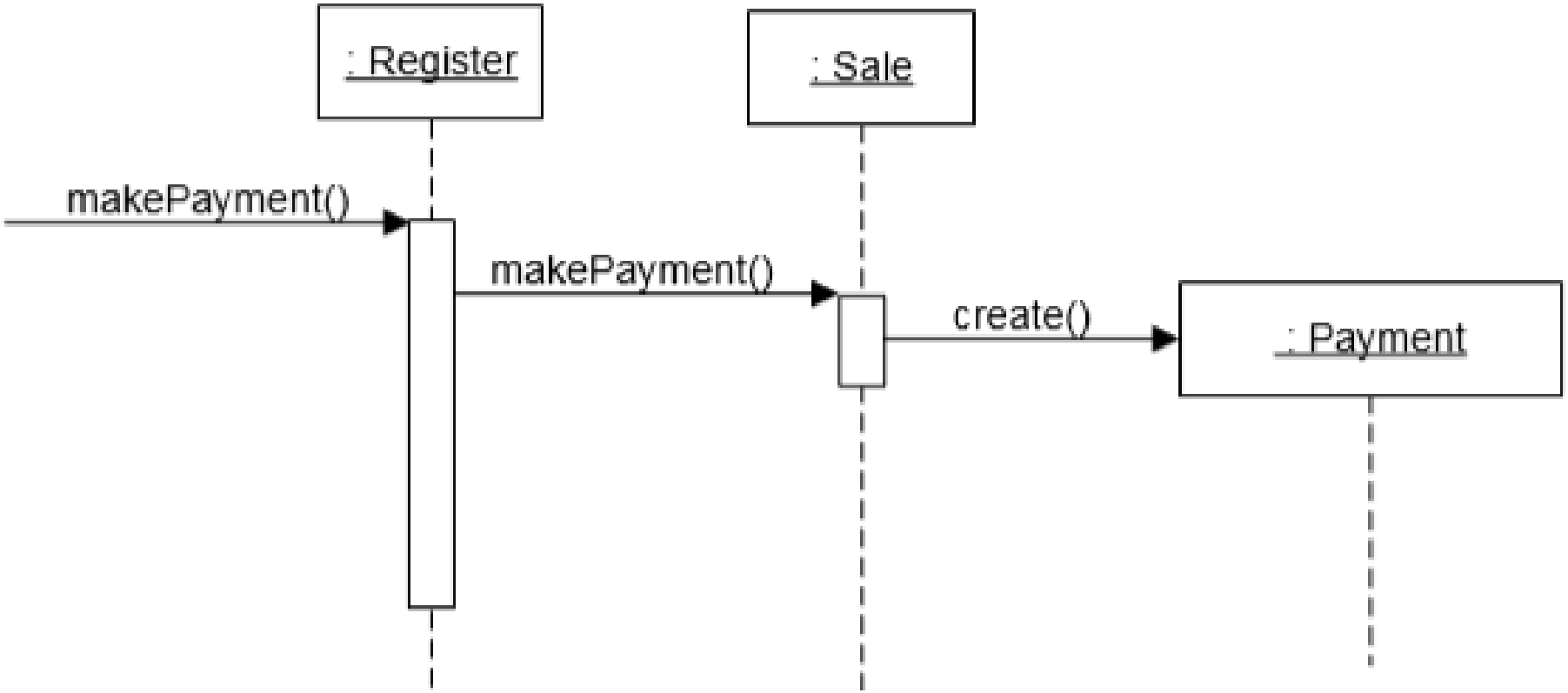
High Cohesion

- **Problema**: como manter a complexidade gerenciável?
- Uma classe com baixa coesão faz muitas coisas não relacionadas com a própria classe ou realiza uma quantidade excessiva de tarefas.
  - Essas classes sofrem com os seguintes problemas:
    - É difícil compreendê-las;
    - É difícil reutilizá-las;
    - É difícil mantê-las;
    - São constantemente afetadas por mudanças.

High Cohesion



High Cohesion



High Cohesion

- **Benefícios**:
  - A clareza e a facilidade de compreensão do projeto é aumentada;
  - A manutenção e as melhorias são simplificadas;
  - Apoia o baixo acoplamento;
  - Aumenta o reuso, pois uma classe coesa pode ser utilizada para um propósito bastante específico.