GRASP: PADRÕES PARA ATRIBUIÇÃO DE RESPONSABILIDADES

SSC 124 – Análise e Projeto Orientado a Objeto

Profa. Dra. Elisa Yumi Nakagawa

2º semestre de 2016

RESPONSABILIDADE

Responsabilidade

- um contrato ou obrigação de um tipo ou classe
- serviços fornecidos por um elemento (classe ou subsistema)

Dois tipos de responsabilidades básicas:

- Fazer
  - fazer algo (criar um objeto, executar uma operação,…)
  - iniciar ações em outros objetos
  - coordenar e controlar atividades em outros objetos
- Saber
  - conhecer dados privados encapsulados
  - conhecer objetos relacionados
  - conhecer coisas que podem ser derivadas ou calculadas

RESPONSABILIDADE

Exemplos:

- "uma *Venda* é responsável por criar *ItemLinhaVenda* " – FAZER
- "uma *Venda* é responsável por conhecer o seu total"- SABER

OBS: responsabilidades do tipo saber frequentemente podem ser deduzidas do modelo conceitual (atributos e associações)

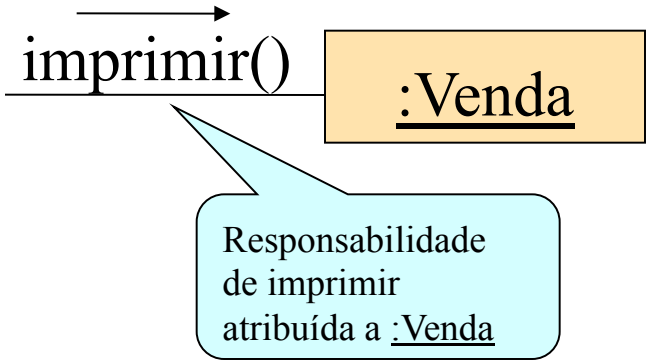
RESPONSABILIDADE E DIAGRAMA DE COMUNICAÇÃO

Diagramas de comunicação mostram escolhas de atribuição de responsabilidade a objetos

Métodos são implementados para satisfazer uma responsabilidade

Objetos podem colaborar com outros objetos para atender a uma responsabilidade

Exemplo:



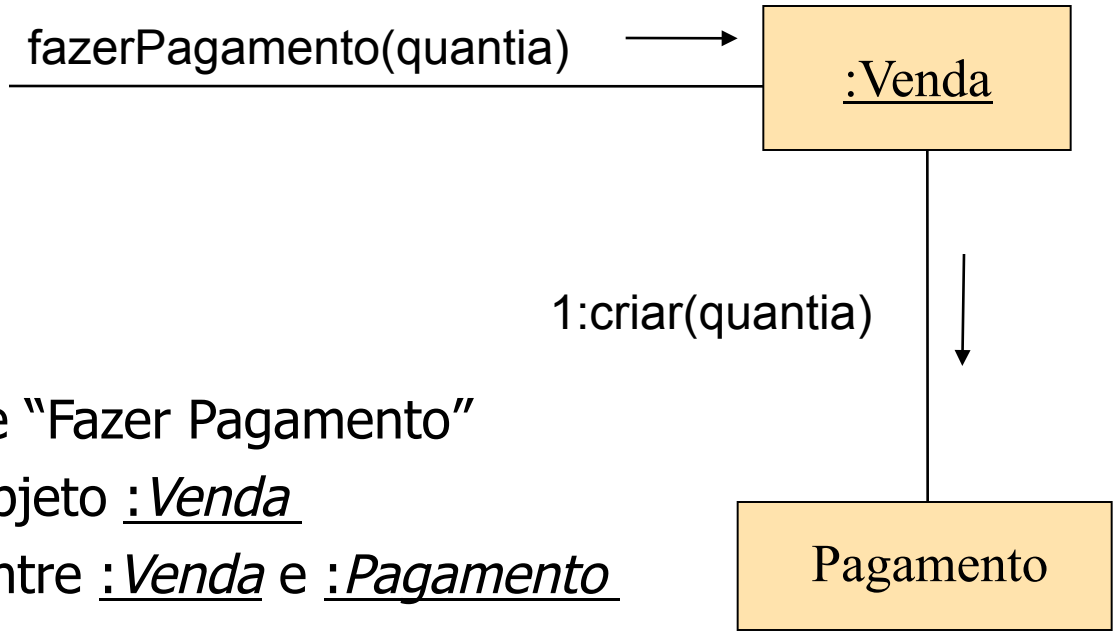
RESPONSABILIDADE

Existem diferentes granularidades da responsabilidade

Exemplo:

- “Fornecer acesso a bancos de dados relacionais” – pode envolver muitos objetos e métodos
- “Imprimir uma venda” – pode envolver apenas um objeto ou alguns poucos métodos

EXEMPLO



Responsabilidade "Fazer Pagamento"

- atribuída ao objeto *:Venda*
- colaboração entre *:Venda* e *:Pagamento*

PADRÕES

Repertório de princípios gerais e boas soluções para guiar a construção de software

Descritas em um formato padronizado (nome, problema e solução) e podem ser usadas em outros contextos

Usualmente não contêm novas ideias

- organizam conhecimentos e princípios existentes, testados e consagrados

Padrão é uma descrição nomeada de um problema e uma solução, que pode ser aplicado em novos contextos

PADRÕES GRASP

GRASP = *General Responsibility Assignment Software Patterns*

Descrevem princípios fundamentais de atribuição de responsabilidade a objetos

Principais padrões GRASP:

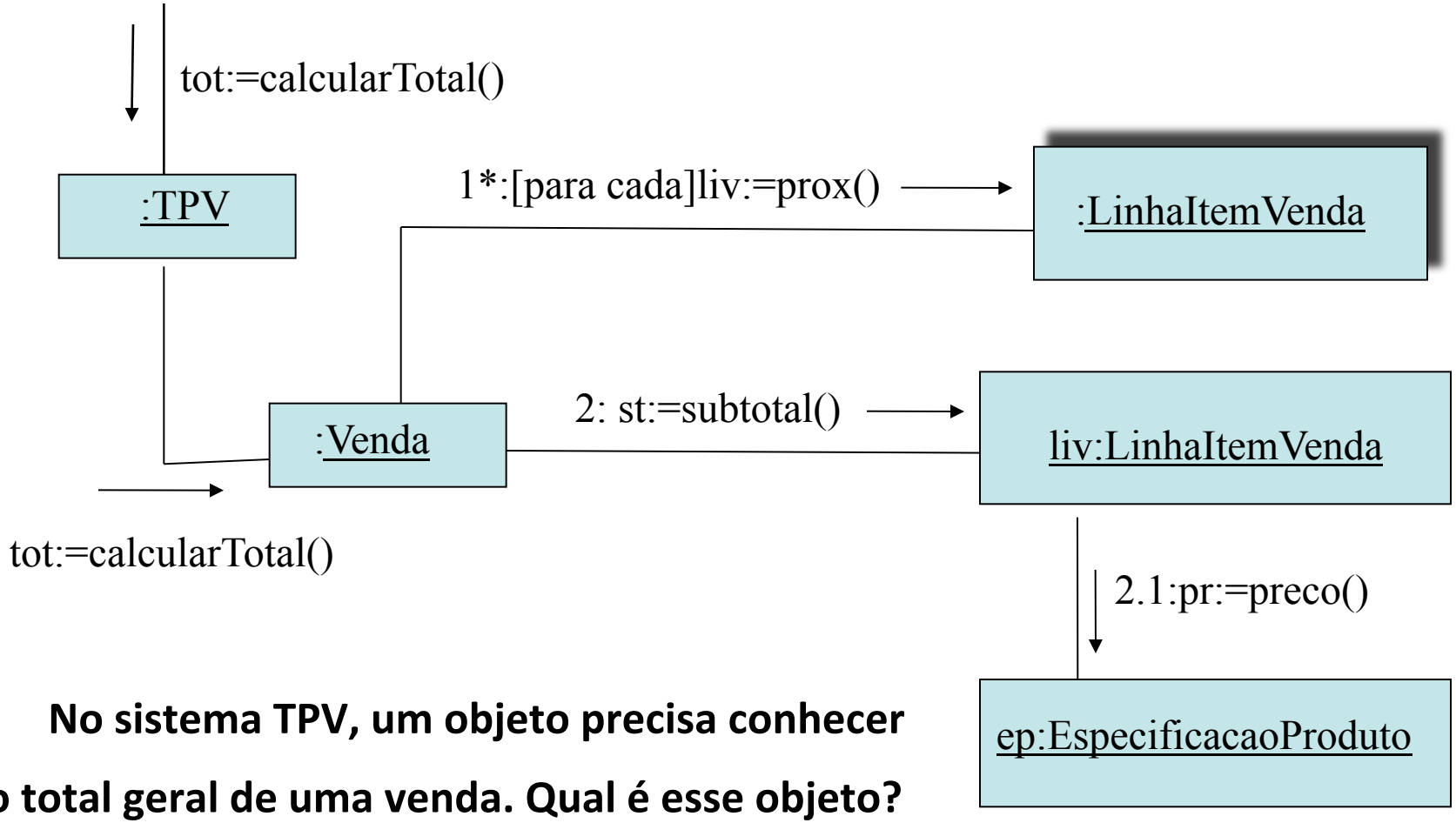
- Especialista (*Expert*)
- Criador (*Creator*)
- Coesão alta (*High Cohesion*)
- Acoplamento fraco (*Low Coupling*)
- Controlador (*Controller*)

ESPECIALISTA

Problema: qual é o princípio mais básico de atribuição de responsabilidades a objetos?

Solução: Atribuir responsabilidade ao especialista da informação.

EXEMPLO



- **No sistema TPV, um objeto precisa conhecer o total geral de uma venda. Qual é esse objeto?**
  - *:Venda* → conhece o total da venda
  - *:LinhaItemVenda* → conhece o subtotal da linha
  - *:EspecificacaoProduto* → conhece o preço do produto

ESPECIALISTA

Discussão

- Padrão mais utilizado
- "Fazê-lo eu mesmo"
  - objetos fazem coisas relacionadas à informação que têm
- Existem especialistas parciais que colaboram em uma tarefa
  - informação espalhada → comunicação via mensagens
- Há uma analogia no mundo real

ESPECIALISTA

Benefícios

- Mantém encapsulamento → favorece o acoplamento fraco
- Comportamento fica distribuído entre as classes que têm a informação necessária (classes "leves") → favorece alta coesão

Contra-indicações

- contra indicado quando aumenta acoplamento e reduz coesão

CRIADOR

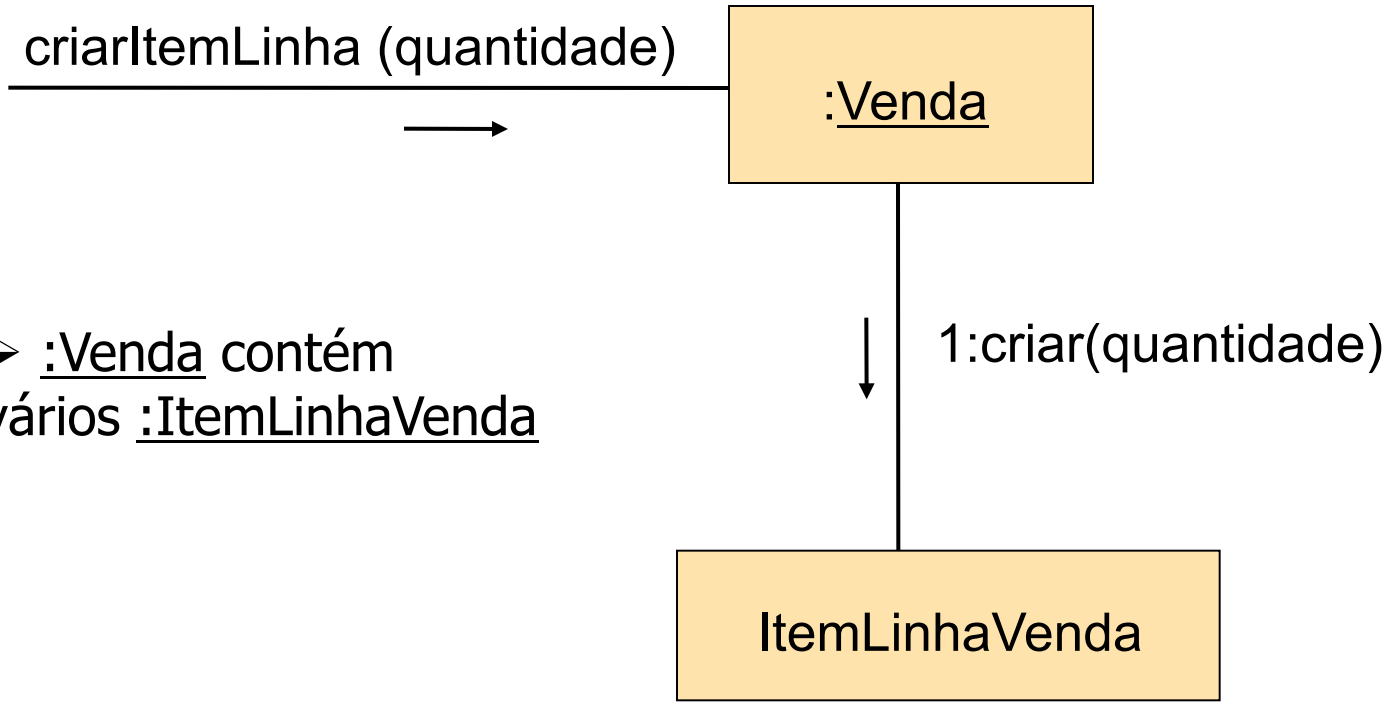
Problema: Quem deveria ser responsável pela criação de uma nova instância de alguma classe?

Solução: atribua à classe B a responsabilidade de criar uma nova instância da classe A se uma das seguintes condições for verdadeira:

- B agrega/contém/registra objetos de A
- B usa objetos de A
- B tem os valores iniciais que serão passados para objetos de A, quando de sua criação

CRIADOR

Exemplo: No sistema TPV, quem é responsável pela criação de uma instância de *ItemLinhaVenda* ?



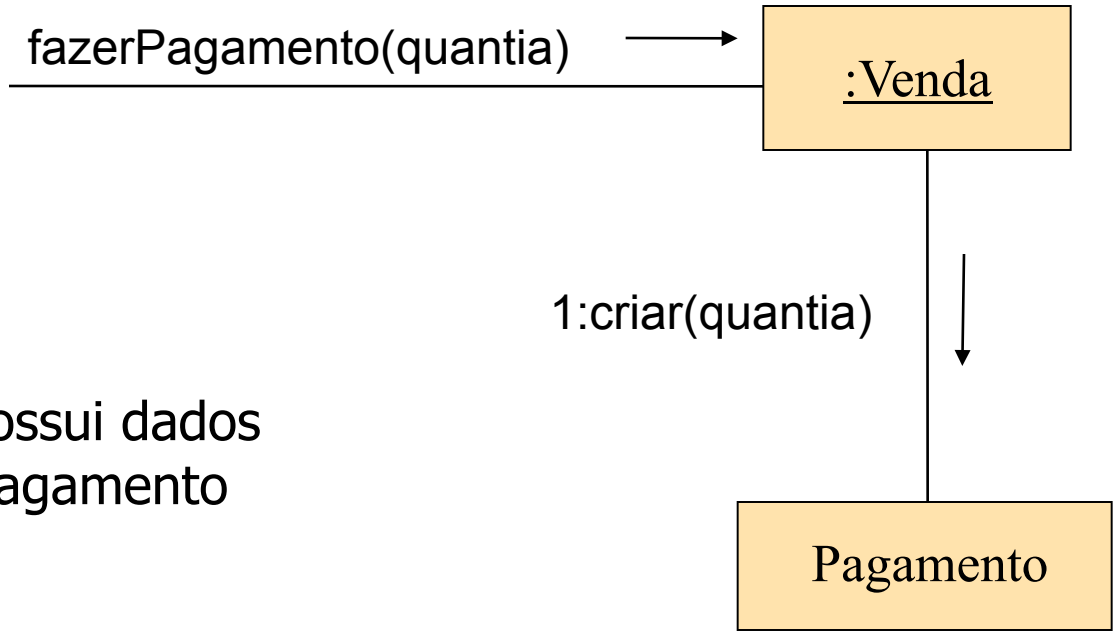
- :Venda contém vários :ItemLinhaVenda

CRIADOR

Discussão

- objetivo do padrão: definir como criador o objeto que precise ser conectado ao objeto criado em algum evento
  - escolha adequada favorece acoplamento fraco
- objetos agregados, contêineres e registradores são bons candidatos à responsabilidade de criar outros objetos
- algumas vezes o candidato a criador é o objeto que conhece os dados iniciais do objeto a ser criado
  - Ex: *Venda* e *Pagamento*

EXEMPLO



- :Venda possui dados iniciais do pagamento

CRIADOR

Benefícios

- favorece o acoplamento fraco
  - provavelmente o acoplamento não é aumentado porque o objeto criado provavelmente já é visível para o objeto criador, devido às associações existentes que motivaram sua escolha como criador

ACOMPLAMENTO

Dependência entre elementos (classes, subsistemas,…), normalmente resultante de comunicação para atender a uma responsabilidade

Mede o quanto um objeto está conectado a, tem conhecimento de ou depende de outros objetos

- acoplamento fraco (ou baixo) – um objeto não depende de muitos outros
- acoplamento forte (ou alto) – um objeto depende de muitos outros

ACOPLAMENTO

Problemas do acoplamento alto:

- Mudanças em classes interdependentes forçam mudanças locais
- Dificulta a compreensão do objetivo de cada classe
- Dificulta reutilização

FORMAS DE ACOPLAMENTO

- **Objeto tem um atributo que referencia um objeto de outra classe**
- **Objeto tem um método que referencia um objeto de outra classe**
  - parâmetro, variável local ou retorno
- **Classe é subclasse de outra, direta ou indiretamente**

ACOPLAMENTO FRACO

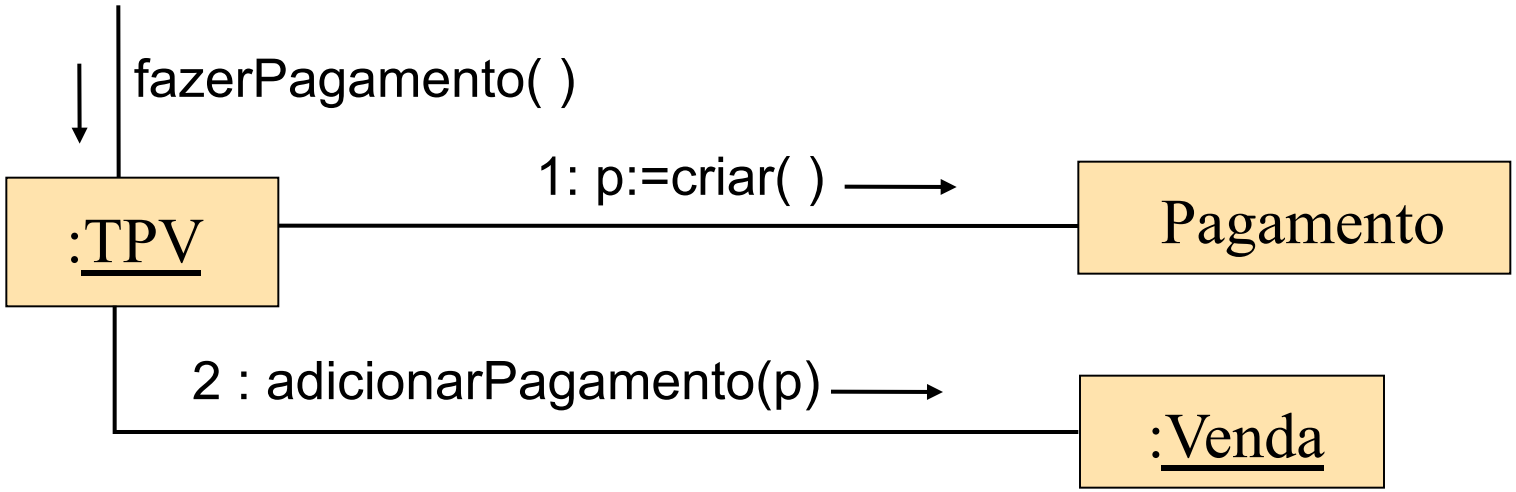
Problema: como favorecer a baixa dependência e aumentar a reutilização ?

Solução: Atribuir responsabilidade de maneira que o acoplamento permaneça baixo.

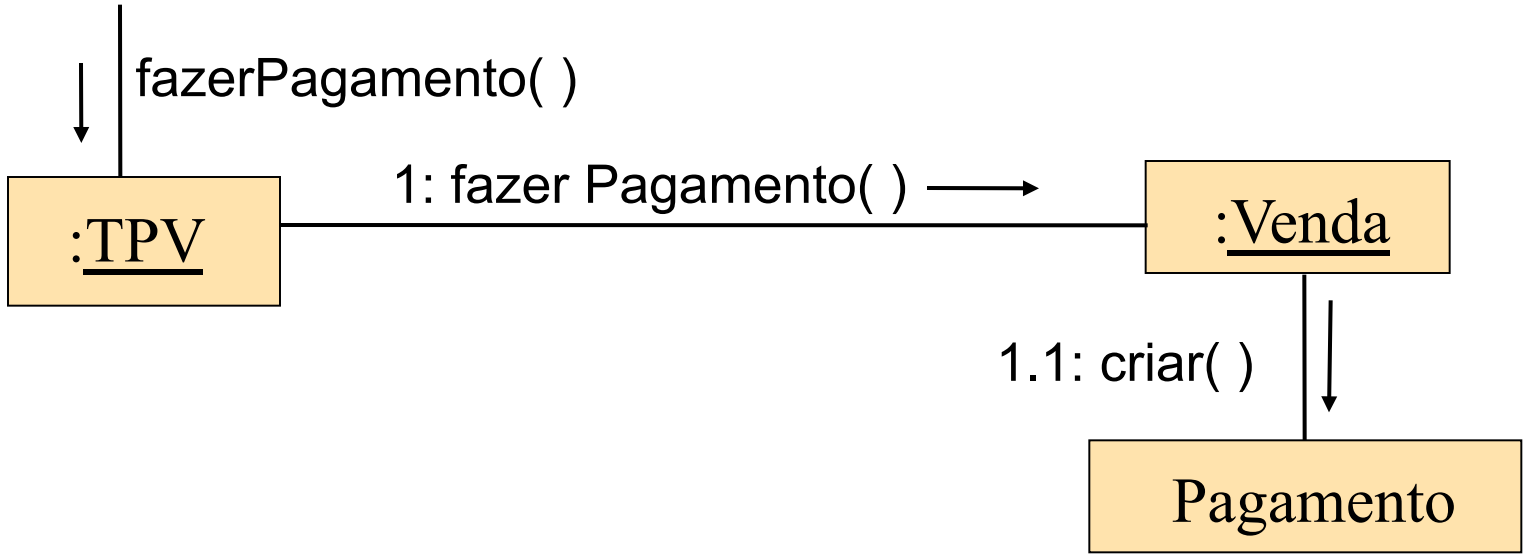
Exemplo:

- **No sistema TPV, suponha que queremos criar uma instância de Pagamento e associá-la à Venda. Qual classe deve ser responsável por essa tarefa?**

Solução 1: segundo padrão Criador, responsabilidade atribuída ao *TPV*



Solução 2: segundo padrão Acoplamento Fraco – responsabilidade atribuída à *Venda*



QUAL É MELHOR?

Qual das soluções anteriores favorece o acoplamento fraco?

- em ambos os casos, *Venda* será acoplada a (terá conhecimento de) *Pagamento*
- Solução 1 – acoplamento entre *Pagamento* e *TPV*
- Solução 2 – não aumenta acoplamento

ACOPLAMENTO FRACO

Discussão:

- Acoplamento fraco → classes mais independentes
  - reduz impacto de mudanças
  - favorece reúso de classes
- Considerado em conjunto com outros padrões
- Extremo de acoplamento fraco não é desejável
  - fere princípios da orientação a objetos que é a comunicação por mensagens
  - projeto pobre, ou seja, objetos inchados e complexos, responsáveis por muito trabalho → baixa coesão

ACOPLAMENTO FRACO

Discussão:

- Concentre-se em reduzir o acoplamento em pontos de evolução ou de alta instabilidade do sistema
  - exemplo: cálculo de impostos no sistema TPV

Benefícios:

- Classe são pouco afetadas por mudanças em outras partes
- Classes são simples de entender isoladamente
- Conveniente para reutilização

COESÃO

Coesão mede o quanto as responsabilidades de um elemento (classe, objeto, subsistema,…) são fortemente relacionadas

Objeto com Coesão Alta → objeto cujas responsabilidades são altamente relacionadas e que não executa um volume muito grande de trabalho

Objeto com Coesão Baixa → objeto que faz muitas coisas não relacionadas ou executa muitas tarefas

- difícil de compreender, reutilizar e manter
- constantemente afetadas por mudanças

COESÃO ALTA

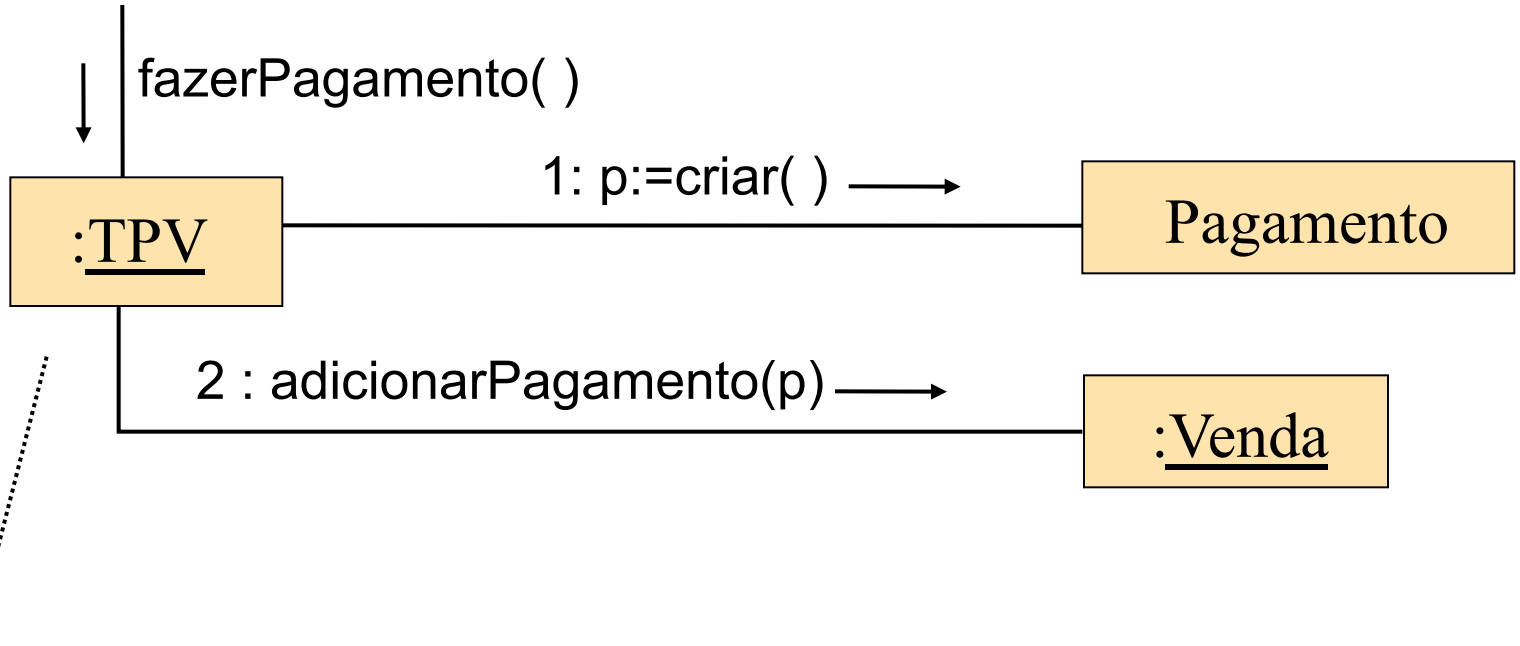
Problema: Como manter a complexidade sob controle?

Solução: Atribuir responsabilidade de tal forma que a coesão permaneça alta.

Exemplo (o mesmo para o acoplamento fraco):

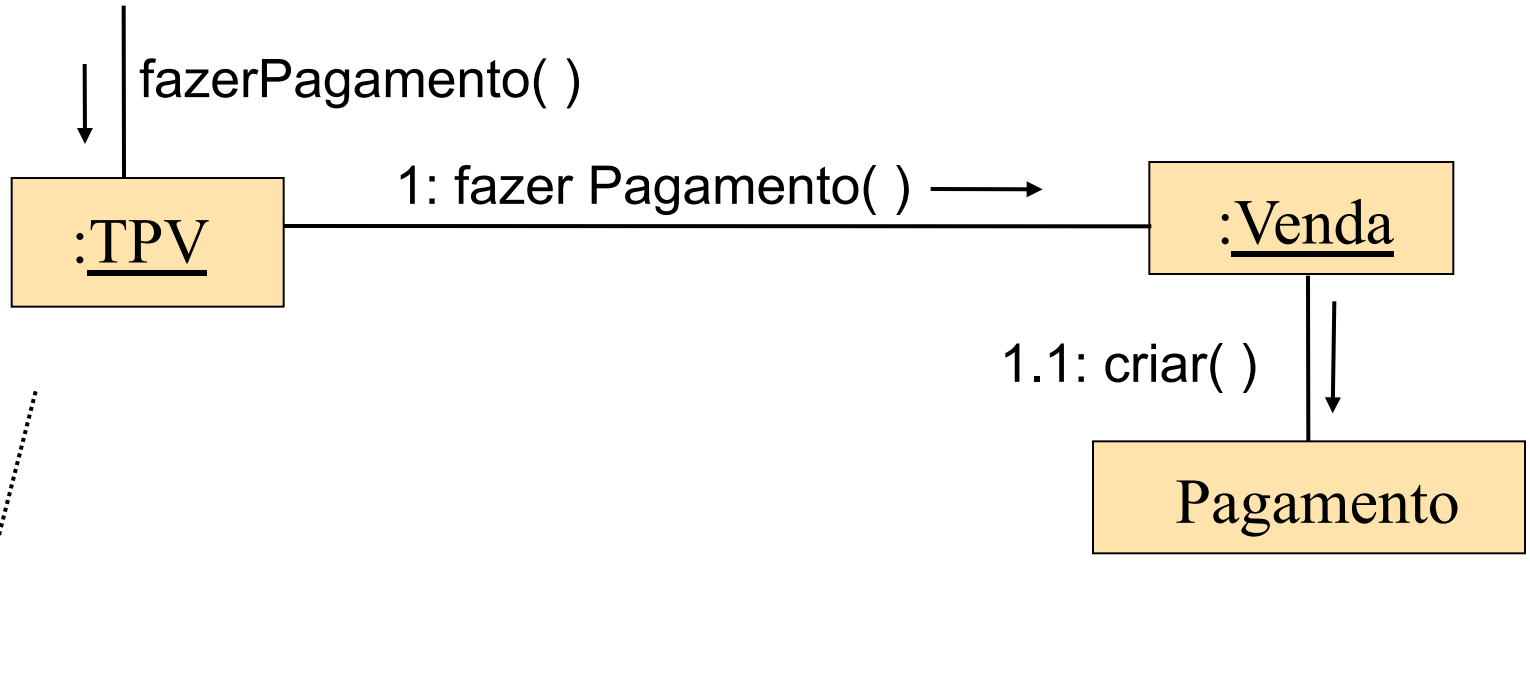
- **No sistema TPV, suponha que queremos criar uma instância de pagamento e associá-la à venda. Qual classe deve ser responsável por essa tarefa?**

Solução 1: segundo padrão Criador, responsabilidade atribuída ao *TPV*



O TPV toma parte na responsabilidade de fazer pagamento. Neste exemplo, isso seria aceitável, mas o que aconteceria se houvessem 50 mensagens recebidas por TPV?

Solução 2: segundo padrão Coesão Alta – responsabilidade atribuída à *Venda*



Esta solução favorece uma coesão mais alta em TPV e também um acoplamento mais fraco. Portanto, projeto 2 é preferível.

COESÃO ALTA

Discussão:

- Coesão alta, assim como Acoplamento Fraco, são princípios que devem ser considerados no projeto de objetos
  - má coesão traz acoplamento ruim e vice-versa
- Regra prática: classe com coesão alta tem um número relativamente pequeno de métodos, com funcionalidades relacionadas, e não executa muito trabalho
- Analogia com mundo real
  - Pessoas que assume muitas responsabilidades não associadas podem tornar-se (e normalmente tornam-se) ineficientes

COESÃO ALTA

Benefícios:

- Mais clareza e facilidade de compreensão no projeto
- Simplificação de manutenção e acréscimo de funcionalidade/melhorias
- Favorecimento do acoplamento fraco
- Aumento no potencial de reutilização
    - classe altamente coesa pode ser usada para uma finalidade bastante específica

CONTROLADOR

Problema: Quem deve ser responsável por tratar um evento do sistema ?

Solução: A responsabilidade de receber ou tratar os eventos (operações) do sistema pode ser atribuída a uma classe que:

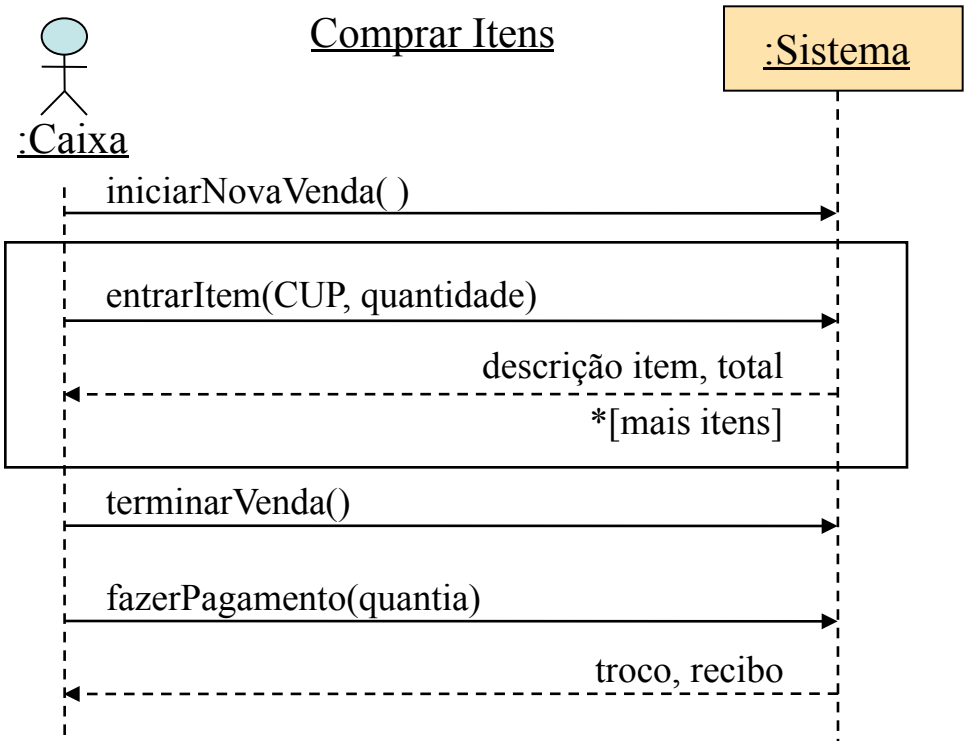
- Represente todo o sistema, um dispositivo ou um subsistema – chamado de controlador fachada

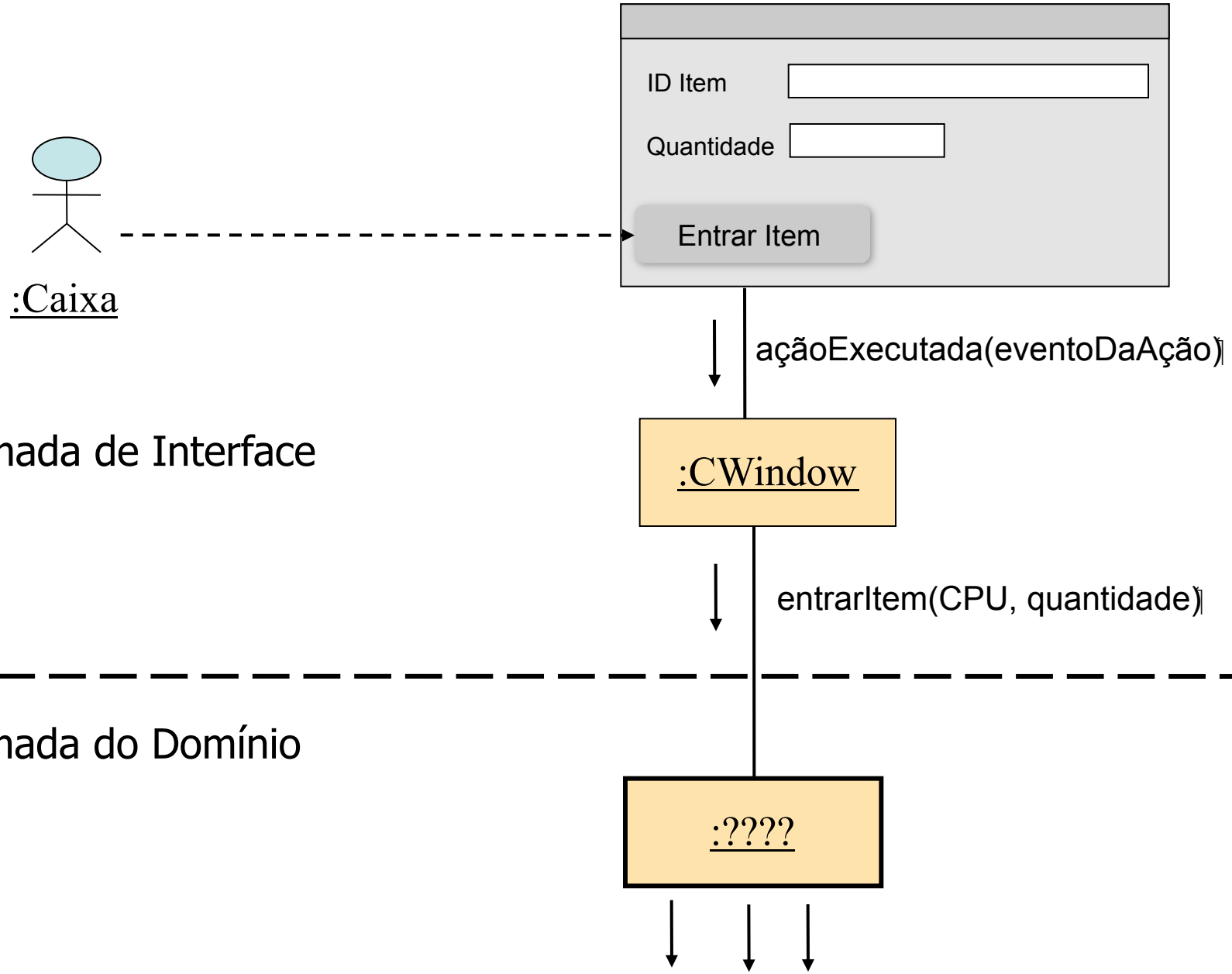
  OU

- Represente um cenário de um caso de uso dentro do qual ocorra o evento
  - TratadorDe<NomeDoCasoDeUso>, ControladorDe<NomeDoCasoDeUso>

CONTROLADOR

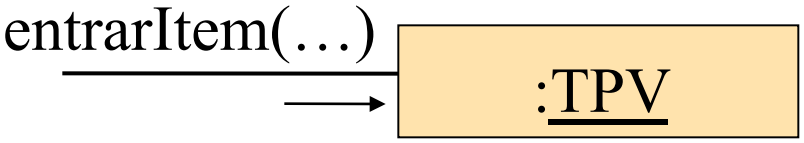
Exemplo: quem vai tratar os eventos do sistema TPV?



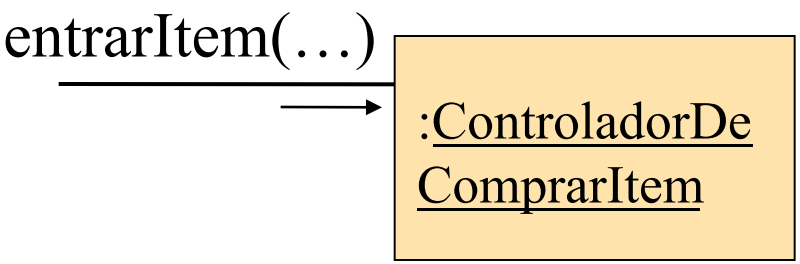

ID Item
Quantidade
Entrar Item
:Caixa
açãoExecutada(eventoDaAção)
Camada de Interface
:CWindow
entrarItem(CPU, quantidade)
Camada do Domínio
:????

EXEMPLO: OPÇÕES DE CONTROLADOR

Todo o sistema (controlador fachada): TPV



Tratador artificial do caso de uso: *ControladorDeComprarItem*



DISCUSSÃO: CONTROLADORES FACHADA

Deve ser um objeto (do domínio) e que seja o ponto principal para as mensagens provenientes da interface com o usuário ou de outros sistemas

- Pode ser uma abstração de uma entidade física
  - Exemplo: *TPV*
- Pode ser um conceito que represente o sistema
  - Exemplo: *SistemaTPV*

Adequados quando não há uma quantidade muito grande de eventos de sistema

DISCUSSÃO: CONTROLADORES DE CASOS DE USO

Não é um objeto do domínio

Controlador diferente para cada caso de uso

Uma alternativa se a escolha de controladores fachada deixar a classe controladora com alto acoplamento e/ou baixa coesão (controlador inchado por excesso de responsabilidade)

Boa alternativa quando existem muitos eventos envolvendo diferentes e muitos casos de uso

CONTROLADORES INCHADOS

Classe controladora mal projetada – inchada

- Coesão baixa – falta de foco e tratamentos de muitas responsabilidades

Sinais de inchaço:

- Única classe controladora tratando todos os eventos, que são muitos. Comum com controladores fachada
- Próprio controlador executa as tarefas necessárias para atender o evento, sem delegar para outras classes (coesão alta, não especialista)
- Controlador tem muitos atributos e mantém informação significativa sobre o domínio, ou duplica informações existentes em outros lugares

CONTROLADOR

Curas para controladores inchados

- Acrescentar mais controladores
- Misturar controladores fachada e de casos de uso

<u>Observação</u>:

- **Objetos de interface (como objetos “janela”) e da camada de apresentação não devem ter a responsabilidade de tratar eventos do sistema**

CONTROLADOR

Benefícios:

- Aumento das possibilidades de reutilização de classes
- Aumento das possibilidades de interfaces “plugáveis”
- Conhecimento do estado do caso de uso – controlador pode armazenar estado do caso de uso, garantindo a sequência correta de execução de operações

GRASP: PADRÕES PARA ATRIBUIÇÃO DE RESPONSABILIDADES

SSC 124 – Análise e Projeto Orientado a Objeto

Profa. Dra. Elisa Yumi Nakagawa

2º semestre de 2016