




## Projeto Orientado a Objetos

### Diagrama de Colaboração


---

Engenharia de Software II




## Projeto OO

- Nesta fase é desenvolvida uma solução lógica baseada no paradigma de orientação a objetos – objetos, mensagens, classes, métodos, ....
  - "Fazer Certo a Coisa" – projetar de maneira competente uma solução que satisfaça os requisitos
- Os dois artefatos principais a serem desenvolvidos são:
  - Diagramas de Interação
  - Diagramas de Classe de Projeto

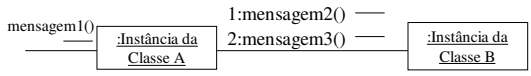


## Diagramas de Interação


- Apresentam como os objetos interagem, por meio de mensagens, para responder a um determinado evento
  - são importantes para o desenvolvimento de um bom projeto
  - exigem criatividade
- Dois tipos de diagramas de interação que permitem representar interação (colaboração) entre classes (ou objetos):
  - diagramas de colaboração – formato de grafo
  - diagramas de seqüência – formato de cerca



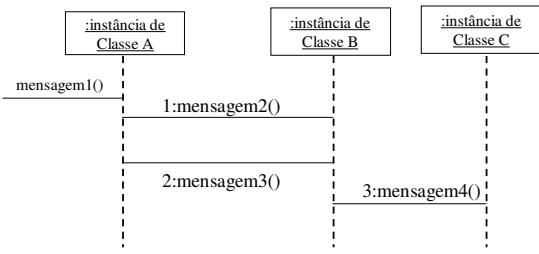

## Diagrama de Colaboração



- Os diagramas de colaboração têm melhor capacidade de expressar informações contextuais e podem ser mais econômicos em termos de espaço



## Diagrama de Seqüência

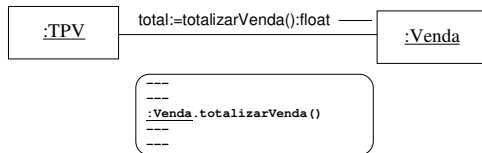
## Mensagens

- No paradigma de orientação a objetos:
  - Mensagem é o mecanismo de comunicação entre objetos
    - invoca as operações desejadas
  - "O processo de invocar um método é chamado de envio de uma mensagem ao objeto"
- Ex: quando uma mensagem *façaAlgo()* é enviada a uma objeto *obj*, o método *façaAlgo()* definido na classe de *obj* é executado:
 

```
obj.fazAlgo()
```

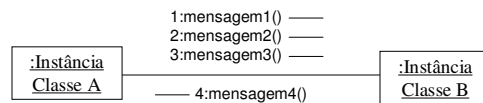
## Ligações

- **Ligação:** conexão em dois objetos que indica a possibilidade de alguma forma de navegação ou de visibilidade entre eles
  - permite que mensagens fluam de um objeto para outro



## Mensagens Múltiplas

- Várias mensagens, em ambos os sentidos, podem fluir ao longo de uma mesma ligação
  - usar seta para indicar direção da mensagem
  - usar números de sequência para indicar ordem de execução das mensagens



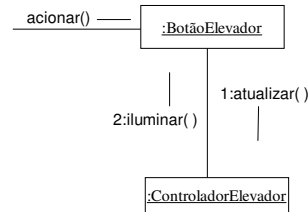
## Mensagens

- **Sintaxe UML:**

**retorno := mensagem ( parâmetro : tipoParâmetro ) : tipoRetorno**

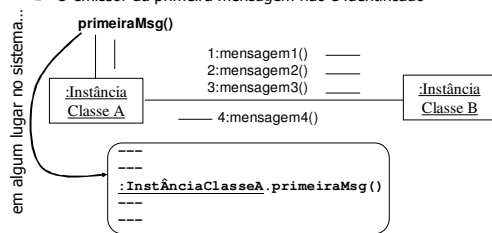
- O retorno pode não existir
- Os tipos podem ser omitidos se forem óbvios ou não forem importantes
- Exemplo:  
 especificacao := obterEspecificacaoProduto(id)  
 especificacao := obterEspecificacaoProduto(id:IdItem)  
 especificacao := obterEspecificacaoProduto(id:IdItem) : EspecificacaoProduto

## Exemplo

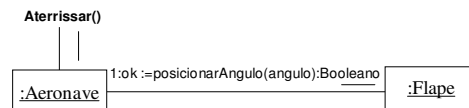


## Primeira Mensagem

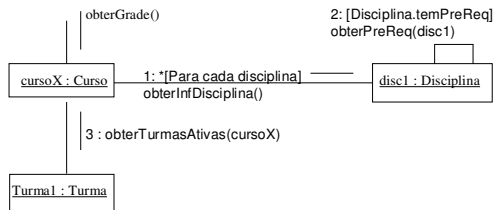
- A **primeira mensagem** é aquela enviada ao objeto que inicia o tratamento a um determinado evento
- O emissor da primeira mensagem não é identificado



## Exemplo



## Exemplo



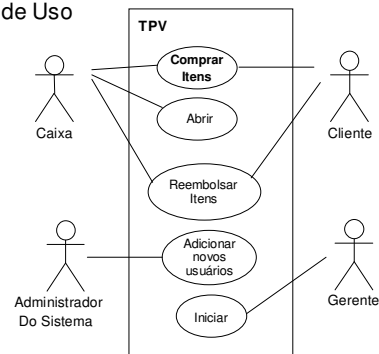
## Criando Diagramas de Colaboração

- O caso de uso sugere eventos do sistema → diagramas de seqüência do sistema
  - Os eventos de sistema representam mensagens que iniciam diagramas de colaboração
- Os diagramas de colaboração ilustram como os objetos interagem para realizar tarefas
  - interação por mensagens de objetos de software, cujos nomes podem ser inspirados pelos nomes dos conceitos (objetos) do Modelo Conceitual

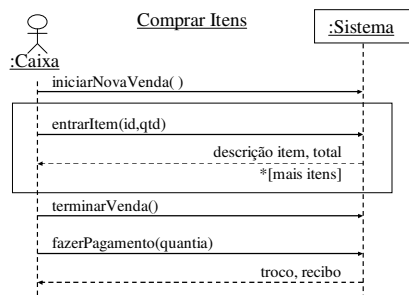
## Criando Diagramas de Colaboração

- Crie um diagrama separado para cada operação do sistema
  - para cada evento do sistema, crie um diagrama com o evento como a primeira mensagem
- Se o diagrama se tornar complexo, separe-o em diagramas menores
- Use o modelo conceitual como apoio
  - conceitos podem inspirar criação de classes de software com organização similar
- Comece escolhendo a classe controladora do evento
- Exemplo sistema TPV...

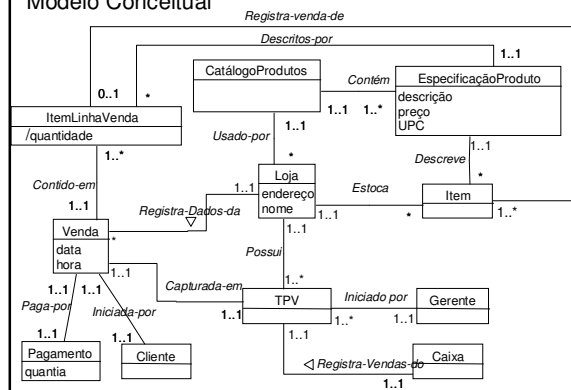
## Diagrama de Casos de Uso



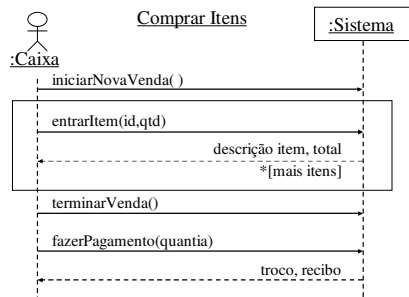
## Diagrama de Seqüência do Sistema para o Caso de Uso "Comprar Itens"



## Modelo Conceitual



### Diagrama de Sequência do Sistema para o Caso de Uso "Comprar Itens"



### Diagrama de Colaboração – iniciarNovaVenda

- Quem é a classe controladora?
- Opções:
  - representando todo o sistema, um dispositivo ou um subsistema  
*TPV, SistemaTPV*
  - representando um tratador de todos os eventos de um cenário de caso de uso  
*ControladorDeComprarItens*  
*TratadorDeComprarItens*

### Diagrama de Colaboração – iniciarNovaVenda

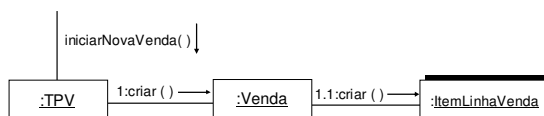
- Classe controladora (de acordo com padrão Controlador): **TPV**
- ```

sequenceDiagram
    participant TPV as :TPV
    TPV->>TPV: iniciarNovaVenda()
  
```
- Observação:
    - na fase de projeto, o TPV é um objeto no "mundo do software", e não o registrador físico (como no Modelo Conceitual)
    - mesmo nome foi escolhido para diminuir *gap* semântico

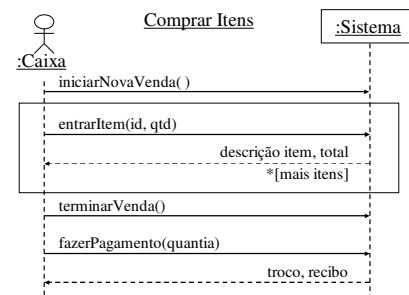
### Diagrama de Colaboração – iniciarNovaVenda

- Criar objeto *Venda*
  - atribuir a responsabilidade à classe que agrega, registra ou contém o objeto a ser criado
    - pelo Modelo Conceitual: *TPV* registra *Venda*
    - TPV* é bom candidato para criar *Venda*
- Pelo Modelo Conceitual: *Venda* possui vários *ItemLinhaVenda*. Então, quando uma *Venda* é criada, uma coleção vazia de *ItemLinhaVenda* deve ser criada para armazenar futuras instâncias
  - Venda* é um bom candidato para criar a coleção

### Diagrama de Colaboração – iniciarNovaVenda



### Diagrama de Sequência do Sistema para o Caso de Uso "Comprar Itens"



## Diagrama de Colaboração – entrarItem

- Classe controladora: **TPV**

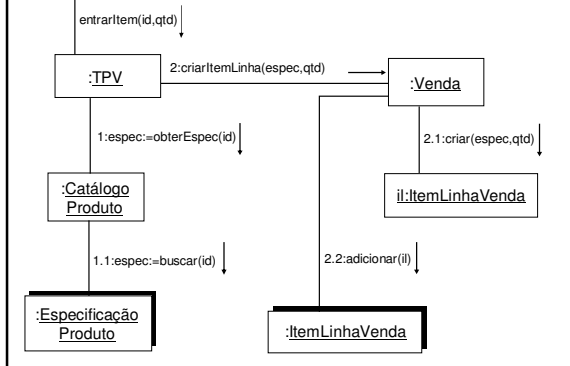
`entrarItem(id, qtd) :TPV`

- A cada tipo de item vendido está associado um *ItemLinhaVenda* na *Venda*
  - cada linha de venda criada será inserida na coleção de *ItemLinhaVenda* associada à *Venda*

## Diagrama de Colaboração – entrarItem

- Pelo Modelo Conceitual: *ItemLinhaVenda* tem quantidade e está associado a *EspecificaçãoProduto*
  - como obter a informação de especificação de produto por meio do ID do item?
  - quem é o especialista nesta informação??
  - Modelo Conceitual: o *CatálogoProdutos* contém todas as especificações → pelo padrão Especialista, o *CatálogoProduto* é bom candidato para a responsabilidade de obter a especificação
  - quem vai enviar a mensagem *obterEspecificação* para o objeto *CatálogoProduto*??
    - é razoável assumir conexão permanente entre *CatálogoProduto* e *TPV*?

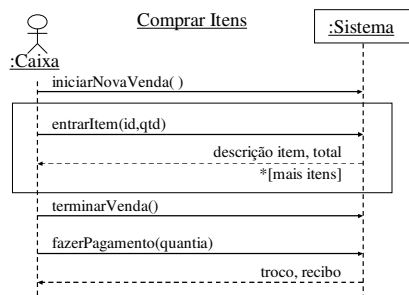
## Diagrama de Colaboração – entrarItem



## Diagrama de Colaboração – entrarItem

- E quanto às informações *descrição* e *preço* do item que, segundo o diagrama de sequência do sistema, devem ser exibidas?
  - não são responsabilidade dos objetos do domínio as tarefas relacionadas a operações de saída do sistema
  - isso deverá ser feito pela camada de apresentação
  - tudo o que é necessário com relação às responsabilidades de exibição de informação é que esta seja conhecida e esteja disponível nos objetos do domínio

## Diagrama de Sequência do Sistema para o Caso de Uso "Comprar Itens"



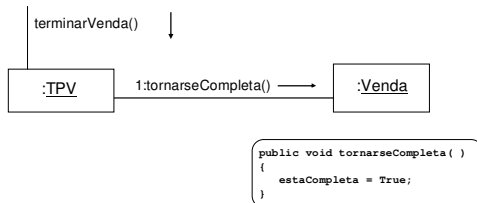
## Diagrama de Colaboração – terminarVenda

- Classe controladora: **TPV**

`terminarVenda() :TPV`

- Quem deve ser responsável por indicar que a venda está terminada?
  - razoável considerar que *Venda* tem um atributo booleano (*estáCompleta*) que é verdadeiro quando a venda é finalizada
  - Com isso, a própria *Venda* é uma candidata apropriada para atribuir o valor Verdadeiro ao atributo *estáCompleta*, já que é dona da informação (mantém o atributo)

## Diagrama de Colaboração – *terminarVenda*



## Diagrama de Colaboração – *obterTotal*

- Terminada a venda, o sistema deve apresentar o total (de acordo com a seqüência típica de eventos do caso de uso *Comprar Itens*)
- Análise Lógica:
  - definir a responsabilidade: quem deve ser responsável por saber o total da venda?
    - total da venda é a soma dos subtotais de todos os itens de linha de venda
    - o subtotal de cada item de linha de venda é dado por: quantidade de itens X preço item
  - resumir informações requeridas

## Diagrama de Colaboração – *obterTotal*

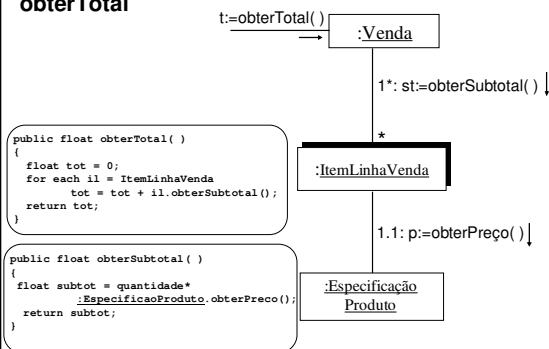
- Análise Lógica (cont.):
  - listar classes que conhecem as informações necessárias

| Informação requerida                             | Especialista                |
|--------------------------------------------------|-----------------------------|
| preço do produto                                 | <i>EspecificaçãoProduto</i> |
| quantidade de itens                              | <i>ItemLinhaVenda</i>       |
| todos os <i>ItemLinhasVendas</i> da <i>Venda</i> | <i>Venda</i>                |

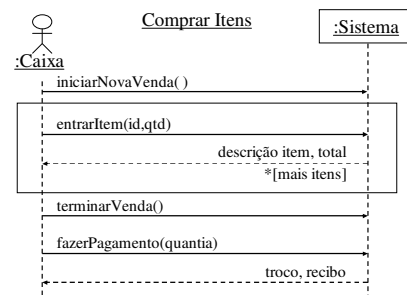
## Diagrama de Colaboração – *obterTotal*

- Quem deve ser responsável por calcular o total da *Venda*?
  - Venda* (conhece todos os *ItemLinhaVenda*)
- O cálculo do total requer cálculo do subtotal de cada *ItemLinhaVenda*. Quem deve ser responsável por calcular esse subtotal?
  - ItemLinhaVenda* (conhece quantidade e *EspecificaçãoProduto*)
- O cálculo do subtotal requer o preço do produto. Quem deve ser responsável por fornecer o preço?
  - EspecificaçãoProduto* (conhece preço)

## Diagrama de Colaboração – *obterTotal*

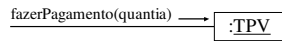


## Diagrama de Sequência do Sistema para o Caso de Uso "Comprar Itens"



## Diagrama de Colaboração – fazerPagamento

- Classe controladora: **TPV**

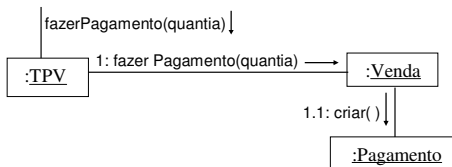


- Quem deve ser responsável pela criação de *Pagamento*? Opções:
  - TPV* – registra *Pagamento* e tem o valor inicial (quantia) para criação da instância
  - Venda* – usa, de maneira bem próxima, *Pagamento*

## Diagrama de Colaboração – fazerPagamento

- Avaliando as opções – considerar Coesão e Acoplamento
  - escolha de *Venda* :
    - pelo Modelo Conceitual - já existe uma associação entre *Venda* e *Pagamento*
    - trabalho de *TPV* fica mais leve → aumenta coesão
    - TPV* não toma conhecimento da existência de *Pagamento* → favorece baixo acoplamento

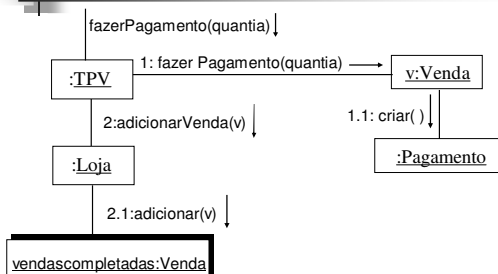
## Diagrama de Colaboração – fazerPagamento



## Diagrama de Colaboração – fazerPagamento

- A venda completada deve ser registrada, por exemplo, num arquivo histórico da loja (ver Modelo Conceitual)
- Quem deve ser responsável por conhecer todas as vendas registradas e por registrá-las?
  - Loja* conhece e registra todas as vendas

## Diagrama de Colaboração – fazerPagamento



## Diagrama de Colaboração – calcularTroco

- Caso de Uso *Comprar Itens* requer que seja devolvido o troco e apresentado o recibo
  - a forma de exibição do troco e impressão do recibo não cabe aqui, mas a informação tem que ser conhecida por objetos do domínio
- Quem deve ser responsável por saber/calcular o troco?
  - Venda* (conhece total da compra) e *Pagamento* (conhece quantia paga) são especialistas parciais

## Diagrama de Colaboração – calcularTroco

- Avaliando as opções:
  - *Pagamento* : possui quantia paga, mas precisa pedir total da compra para *Venda*
    - *Pagamento* precisará ter visibilidade de *Venda* (o que ainda não tem) → aumento no acoplamento
  - *Venda* : possui total da compra, mas precisa solicitar quantia paga a *Pagamento*
    - *Venda* já tem visibilidade de *Pagamento* (pois é seu criador) → não causa aumento no acoplamento, e portanto é a solução mais desejável

## Diagrama de Colaboração – calcularTroco

