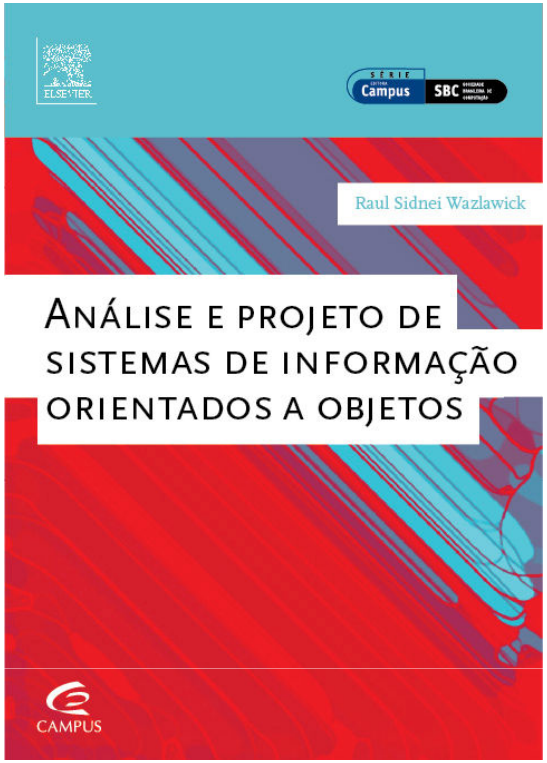


GERAÇÃO AUTOMÁTICA DE CÓDIGO EXECUTÁVEL A PARTIR DE CONTRATOS OCL

Prof. Raul Sidnei Wazlawick

Universidade Federal de Santa Catarina – UFSC

Pós-Graduação em Ciência da Computação - PPGCC

Resumo

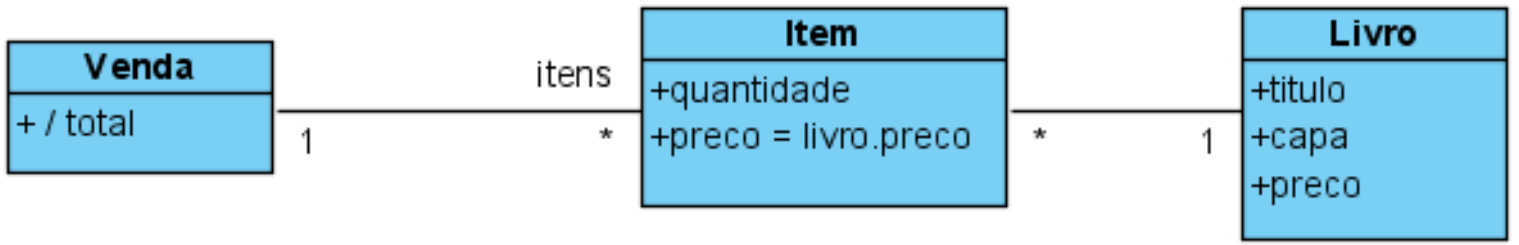
- A atividade de programação de computadores ainda é, em muitas empresas, altamente onerosa em termos de tempo e esforço despendido, e em termos de defeitos inseridos por puro descuido ou por não se seguir padrões estabelecidos. Isso baixa a produtividade e a qualidade do produto final, dificultando sua manutenção e evolução.
- Uma solução para este problema é a geração automática de código a partir de modelos de mais alto nível. A geração de "esqueletos" de programa já é realidade há muitos anos, mas a geração de código efetivamente executável com qualidade ainda é um objetivo alcançado por relativamente poucos.
- É que para que um modelo tenha efetivamente potencial para geração de código, ele deve ser de alta cerimônia, o que nem sempre é conseguido por equipes de modelagem.
- Este tutorial ensina como produzir modelos orientados a objetos de alta cerimônia, usando a linguagem OCL (Object Constraint Language), parte do padrão UML (Unified Modeling Language).
- Será mostrado também como a especificação de diagramas de classe UML complementada pela especificação das operações de sistema na forma de contratos OCL permite a geração automatizada de código executável de alta qualidade, dispensando assim a necessidade de realização de testes de unidade.

TUTORIAL

- **1. Geração de código a partir do modelo conceitual**
  - **Diagrama de Classes**
- 2. Geração de código a partir do modelo funcional
  - Contratos de Operação de Sistema

1. Elementos do Modelo Conceitual

- **Atributos**:
  - informações alfanuméricas simples, como números, textos, datas, etc.
- **Classes** ou **conceitos**:
  - representação da informação complexa que agrega atributos.
- **Associações**:
  - um tipo de informação que liga diferentes conceitos entre si.



1.1 CLASSES E ATRIBUTOS

```
CLASSE Livro
        VAR PRIVADA
                isbn : String
                titulo : String
                autor : String
                nrPaginas : Inteiro
```

```
MÉTODO getIsbn():String
        RETORNA isbn
FIM METODO

MÉTODO setIsbn(umIsbn:String)
        isbn := umIsbn
FIM Método
```

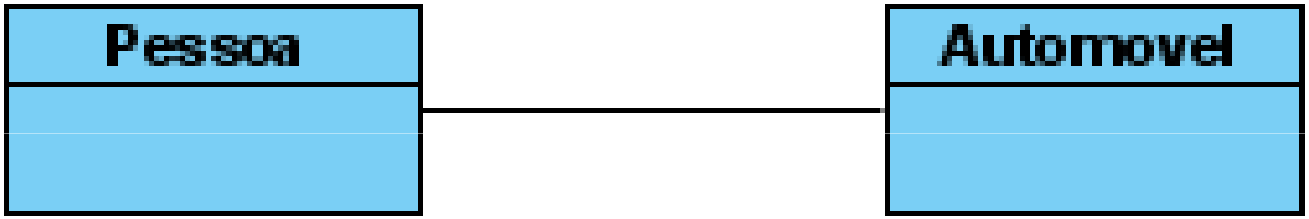
*... – **getter** e **setter** similares para titulo, autor e nrPaginas*

Classes e atributos no Banco de Dados

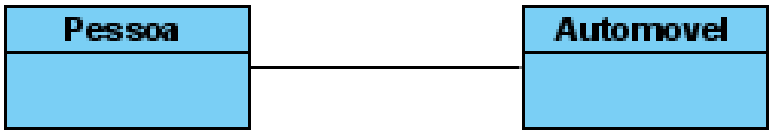
Tabela: Livro					
pkLivro <<pk>>	**isbn <<unique>>**	**titulo**	**editora**	**autor**	**nrPaginas**
10001	12345	análise e projeto	campus	raul	302
10002	54321	metologia de pesquisa	campus	raul	156
10003	11111	a república	acrópole	platão	205

1.2 Associações

- Relacionam duas ou mais classes entre si.

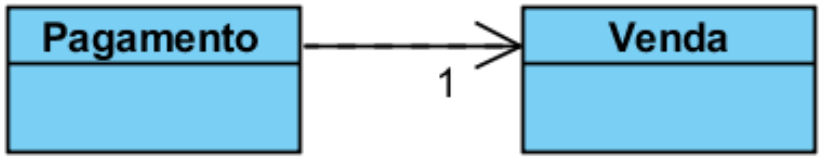


Métodos de uma associação



- Add
  - tendo como parâmetro o objeto a ser associado.
    - `umaPessoa.addAutomovel(umAutomovel)`
- Remove
  - tendo como parâmetro o objeto a ser desassociado.
    - `umaPessoa.removeAutomovel(umAutomovel)`
- Get
  - retornando uma cópia da coleção de objetos associados, sobre a qual é possível realizar iterações.
    - `umaPessoa.getAutomovel()`

1.2.1 Associações unidirecionais

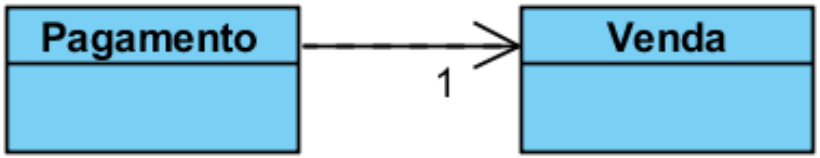


- Associações unidirecionais => variáveis de instância.

```
Classe Pagamento
    var venda : Venda
```

- Diferenças com atributos:
  - atributos são sempre implementados por variáveis cujos tipos são primitivos (alfanuméricos).
  - associações são implementadas por variáveis que são classes (no caso de associações para um) ou estruturas de dados (no caso de associações para muitos).

ASSOCIAÇÃO PARA 1




```
Classe Pagamento
VAR PRIVADA
        venda : Venda

MÉTODO addVenda(umaVenda)
   venda := umaVenda
FIM MÉTODO

MÉTODO removeVenda ()
        venda := NULL
FIM MÉTODO

MÉTODO getVenda():Venda
        RETORNA venda
FIM MÉTODO
FIM CLASSE
```

Associação de um para um no Banco de Dados (BD)

- As associações persistentes entre as classes corresponderão a *tabelas associativas* no modelo relacional:
  - tabelas com uma chave primária composta pelas chaves primárias de duas outras tabelas.
- Conforme o tipo de multiplicidade dos papéis da associação, algumas regras devem ser observadas.
- No caso de associações de 1 para 1, a tabela associativa terá *unique* nas duas colunas da chave primária, impedindo, com isso, que qualquer dos elementos associe-se com mais de um elemento da outra classe

EXEMPLO

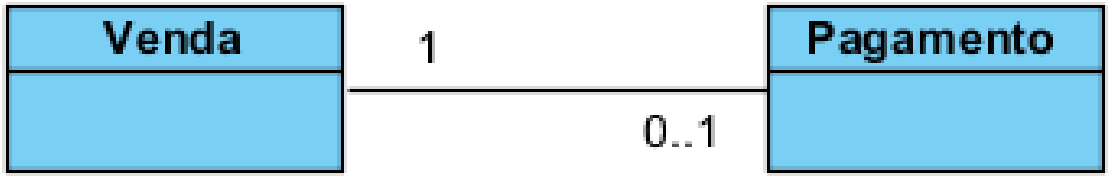


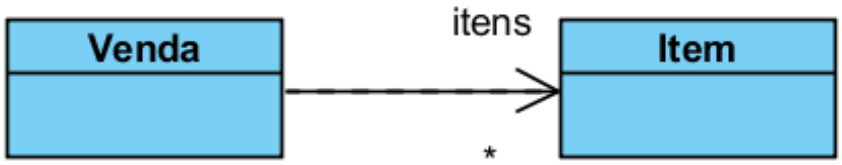
Tabela: venda_pagamento	
pkVenda <<pk>> <<unique>>	**pkPagamento <<pk>> <<unique>>**
50001	60001
50003	60002
50005	60003
50011	60004

Observações:

• Todo pagamento existente aparece na tabela associativa, pois tem associação para 1 com venda.

• Nem toda venda existente aparece na tabela associativa, pois tem associação para 0..1 com pagamento.

1.2.1.2 Associação para muitos




```
CLASSE Venda
VAR PRIVADA
        itens : SET[Item]

MÉTODO addItens(umItem:Item)
        itens.add(umItem)
FIM MÉTODO

MÉTODO removeItens(umItem:Item)
        itens.remove(umItem)
FIM MÉTODO

MÉTODO getItens():Set[Item]
        RETORNA itens.proxy()
FIM MÉTODO
```

Associação de muitos para muitos no BD

- Se nenhum dos lados da associação tiver multiplicidade 1, então nenhuma das colunas que formam a chave primária será marcada com *unique*.

EXEMPLO

Tabela: Livro					
pkLivro <<pk>>	**isbn <<unique>>**	**titulo**	**editora**	**autor**	**nrPaginas**
10001	12345	análise e projeto	campus	raul	302
10002	54321	metologia de pesquisa	campus	raul	156
10003	11111	a república	acrópole	platão	205

Tabela> Pessoa			
pkPessoa <<pk>>	**cpf <<unique>>**	**nome**	**endereco**
20001	3637283	joão	rua não
20002	3729109	miguel	av. das dores
20003	3938204	maria	rua talvez

Tabela: listaDesejos_quemDeseja	
pkLivro<<pk>>	**pkPessoa <<pk>>**
10001	20001
10001	20003
10003	20001

Associação de um para muitos

- No caso de associações de 1 para muitos ou de muitos para 1, a tabela associativa terá a condição *unique* na coluna correspondente à classe do lado “muitos”.
- Isso significa que a coluna correspondente ao lado “muitos” da associação não pode ter seus valores individuais repetidos.

EXEMPLO

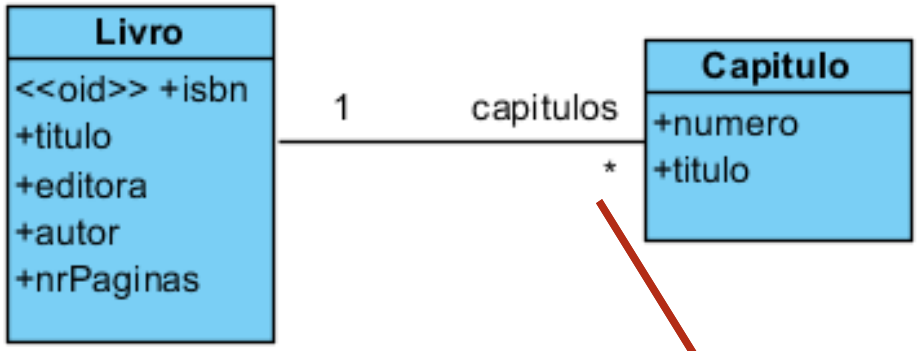


Tabela: livro_capitulos	
pkLivro<<pk>>	**pkCapitulo <<pk>> <<unique>>**
10001	30001
10001	30002
10001	30003
10002	30004
10002	30005
10003	30006
10003	30007

LIMITES MÁXIMO E MÍNIMO

- Genericamente, considerado que *A* tem uma associação para *B* e que o limite mínimo do papel de *A* para *B* é *n* enquanto o limite máximo é *m* (onde *m* pode ser * ou infinito), o número de vezes que cada instância de *A* aparece na tabela associativa é limitado inferiormente por *n* e superiormente por *m*.
- Por exemplo, se a multiplicidade de papel de *A* para *B* for 2..5, cada instância de *A* deve aparecer na tabela associativa no mínimo 2 e no máximo 5 vezes.

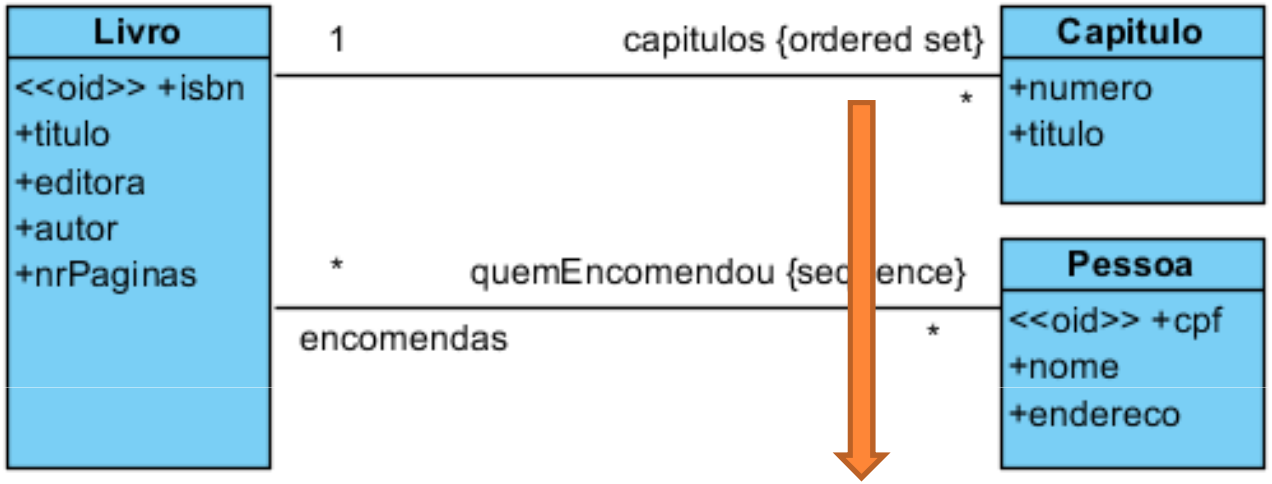
1.2.1.3 Outros tipos de associações para muitos

- Se a associação for rotulada com {sequence}, {ordered set} ou {bag}, deve-se substituir o tipo de dados da variável de instância Set, pelo tipo apropriado de acordo com a linguagem.
- Podem ser usados também, conforme o caso, tipos concretos de dados como *array* ou árvore binária, por exemplo.
- No caso de associações para muitos com limite inferior e superior idênticos, inclusive, recomenda-se a implementação como *array*.
  - Por exemplo, uma associação com multiplicidade de papel 5 (ou 5..5) deve ser implementada como um array de cinco posições.

Associação ordenada no BD

- Uma associação que tenha um papel ordenado (*sequence* ou *ordered set*) em um dos lados deverá implementar na tabela relacional uma coluna adicional que representa a ordem de ocorrência dos elementos.

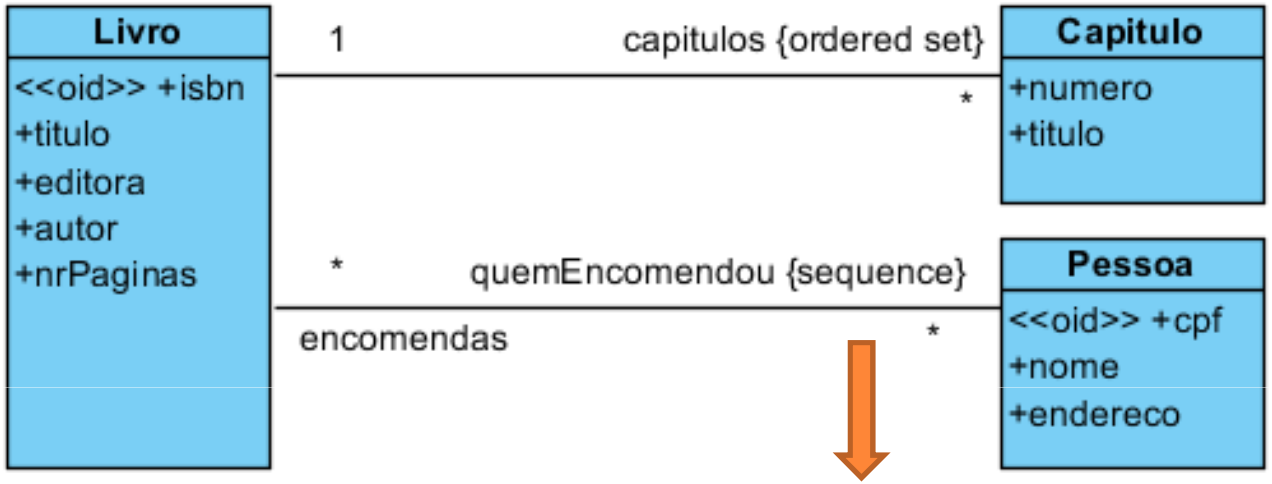
EXEMPLO: ORDERED SET



Caso se trate de um conjunto ordenado (portanto sem repetição de elementos), a coluna de ordem não deve fazer parte da chave primária.

Tabela: livro_capitulos		
pkLivro <<pk>>	**pkCapitulo <<pk>> <<unique>>**	**ordem**
10001	30001	1
10001	30002	2
10001	30003	3
10002	30004	1
10002	30005	2
10003	30006	1
10003	30007	2

EXEMPLO: SEQUENCE



Caso se trate de uma *sequence*, ou seja, com repetição de elementos na lista, então a coluna de ordem deve fazer parte da chave primária da tabela.

Tabela: encomendas_quemEncomendou		
pkLivro <<pk>>	**pkPessoa <<pk>>**	**ordem <<pk>>**
10001	20001	1
10001	20003	2
10001	20002	3
10001	20001	4
10002	20003	1
10003	20001	1
10003	20002	2

MULTICONJUNTO (BAG) NO BD

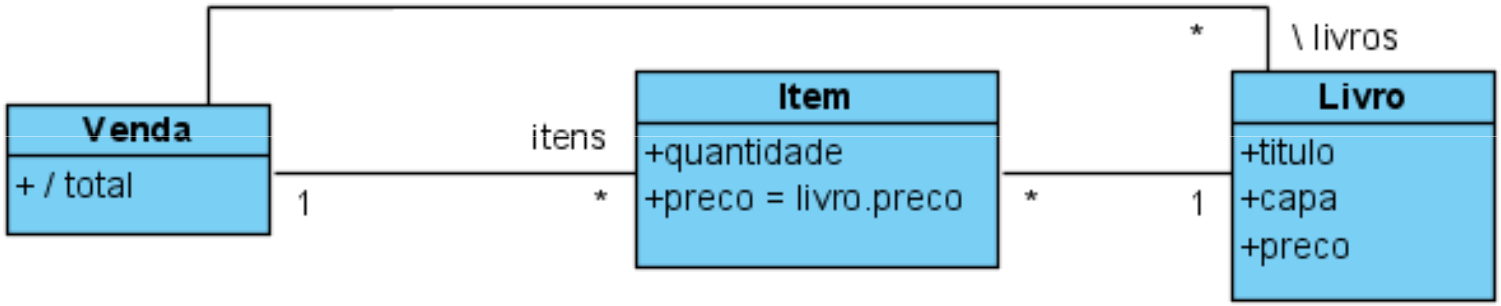
- No caso de multiconjuntos, ou *bags*, que são conjuntos que admitem repetição de elementos, mas que não estabelecem ordem entre eles, a solução usual é adicionar uma coluna extra com um contador do número de vezes que uma instância aparece na associação.

EXEMPLO

Tabela: livro_visualizadores		
pkLivro <<pk>>	**pkPessoa <<pk>>**	**quantidade**
10001	20001	1
10001	20002	1
10002	20001	2
10002	20003	6
10003	20001	1

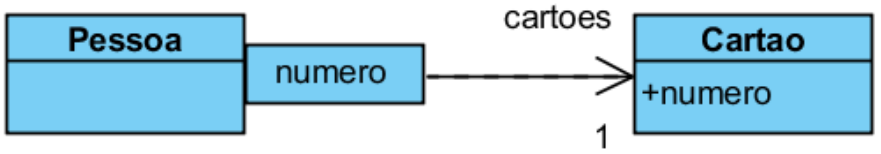
1.2.1.4 Associação derivada

- Implementa apenas o método get




```
Classe Venda
METODO getLivros()
  RETORNA self.getItens().getLivro()
FIM METODO
```

1.2.1.5 Associação qualificada (qualificador interno)

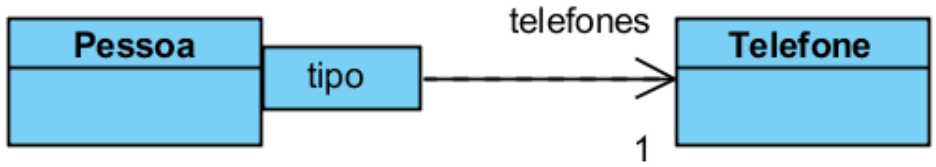



```
Classe Pessoa
VAR PRIVADA
        cartoes : MAP[String->Cartao]

MÉTODO addCartoes(umCartao)
        cartoes.put(umCartao.getNumero(), umCartao)
FIM MÉTODO

MÉTODO removeCartoes (umNumero:String)
        cartoes.removeKey(umNumero)
FIM MÉTODO

MÉTODO getCartoes(umNumero:String):Cartao
        RETORNA cartoes.at(umNumero)
FIM MÉTODO
```

Associação qualificada com qualificador interno no BD

- No caso de associação qualificada para 1 com *qualificador interno* (o qualificador é atributo da classe), basta implementar a associação como uma mera associação para muitos, tomando o cuidado de fazer com que a coluna que contém o atributo qualificador seja marcada com *unique* na tabela que contém o conceito original.

QUALIFICADOR EXTERNO




```
Classe Pessoa
VAR PRIVADA
        telefones : MAP[String->Telefone]

MÉTODO addTelefones(umTipo:String, umTelefone:Telefone)
        telefones.put(umTipo, umTelefone)
FIM MÉTODO

MÉTODO removeTelefones(umTipo:String)
        telefones.removeKey (umTipo)
FIM MÉTODO

MÉTODO getTelefones(umTipo:String):Telefone
        RETORNA cartoes.at(umTipo)
FIM MÉTODO
```

EXEMPLO: QUALIFICADOR EXTERNO NO BD

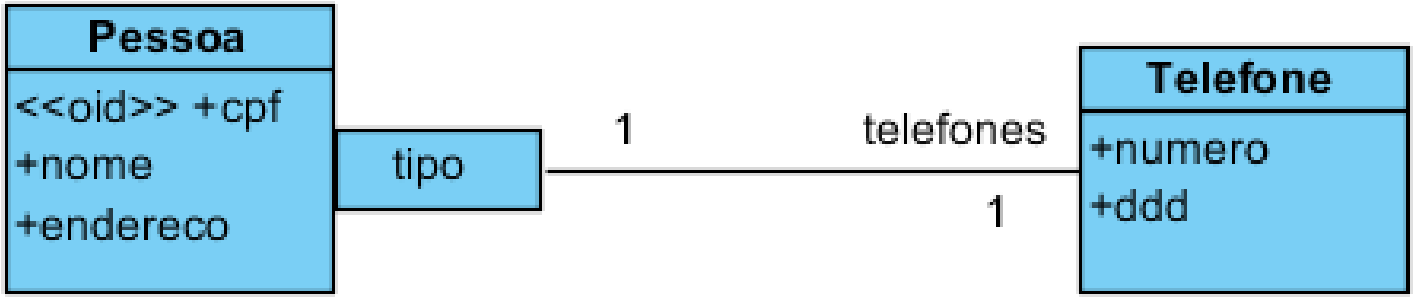


Tabela: pessoa_telefones		
pkPessoa <<pk>>	**tipo <<pk>>**	**pkTelefone <<unique>>**
20001	casa	70001
20001	celular	70002
20002	casa	70003

1.2.1.6 Associação ordenada

- Pode-se ter um método *get* que retorna um elemento conforme sua posição:
  - getReserva(5);
- Da mesma forma o *add* poderá adicionar elementos diretamente em uma posição indicada como parâmetro:
  - addReserva(5,umaReserva)
- *O método remove* poderá remover da posição indicada:
  - removeReserva(5)

Pilhas e filas

- Terão métodos específicos como *push* e *pop*, que seguem as regras específicas destas estruturas:
  - pushReserva(umaReserva)
  - popReserva():Reserva

1.2.2 Associações bidirecionais

- Opções de implementação:
  - Implementar a associação como **duas associações unidirecionais** nas duas classes participantes.
  - Implementar a associação como unidirecional **apenas em uma das classes**. A outra classe pode acessar os elementos da associação fazendo uma pesquisa.
  - Implementar um **objeto intermediário** que representa a associação e pode ser identificado através de métodos de localização rápida como *hash*.

Os métodos get, add e remove nas bidirecionais

- O método *get*, em todos os casos de associação bidirecional, deve ser implementado nas duas classes para permitir a navegação nas duas direções.

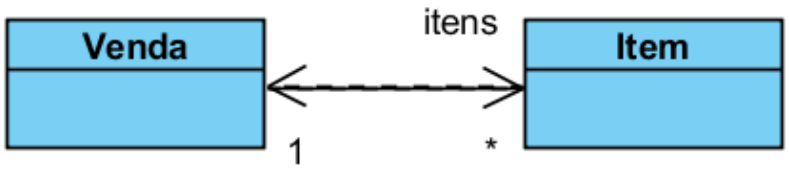
- Os métodos *add* e *remove* podem ser implementados em apenas uma das duas classes, pois se existissem em ambas as classes seriam operações perfeitamente simétricas e, portanto, desnecessárias.

Implementação nas duas direções

- A opção de implementação das associações bidirecionais nas duas direções é a mais eficiente em termos de tempo de processamento, mas produz código mais complexo e gasta mais espaço de armazenamento, pois a informação sobre a associação é representada de forma duplicada, ou seja, nas duas classes que participam dela.

IMPLEMENTAÇÃO NAS DUAS DIREÇÕES

```
CLASSE Venda
VAR PRIVADA
          itens : SET[Item]

MÉTODO addItensPrivado(umItem:Item)
          itens.add(umItem)
FIM MÉTODO

MÉTODO removeItensPrivado(umItem:Item)
          itens.remove(umItem)
FIM MÉTODO

MÉTODO addItens(umItem:Item)
          self.addItensPrivado(umItem)
          umItem.addVendaPrivado(self)
FIM MÉTODO

MÉTODO removeItens(umItem:Item)
          self.removeItensPrivado(umItem)
          umItem.removeVendaPrivado()
FIM MÉTODO

MÉTODO getItens():Set[Item]
          RETORNA itens.proxy()
FIM MÉTODO
```



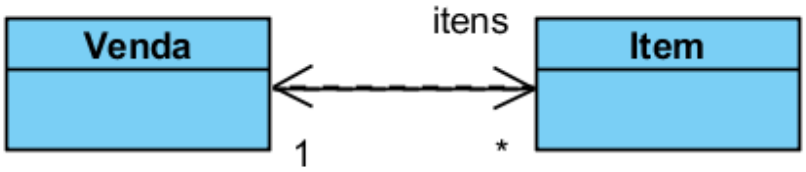

```
Classe Item
VAR PRIVADA
          venda : Venda

MÉTODO addVendaPrivado(umaVenda)
          venda := umaVenda
FIM MÉTODO

MÉTODO removeVendaPrivado ()
          venda := NULL
FIM MÉTODO

MÉTODO getVenda():Venda
          RETORNA venda
FIM MÉTODO
FIM CLASSE
```

Implementação unidirecional

- Mesmo que a associação seja bidirecional pode acontecer que a navegação seja muito mais freqüente ou mais crítica em uma direção do que em outra.
- Se isso acontecer, pode ser uma opção realizar a implementação apenas numa direção.
- A vantagem é o código mais simples e a economia de espaço.
- A desvantagem é que a navegação na direção oposta será uma operação bem mais lenta do que na direção implementada.

IMPLEMENTAÇÃO UNIDIRECIONAL

```
CLASSE Venda
VAR PRIVADA
          itens : SET[Item]

MÉTODO addItens(umItem:Item)
          itens.add(umItem)
FIM MÉTODO

MÉTODO removeItens(umItem:Item)
          itens.remove(umItem)
FIM MÉTODO

MÉTODO getItens():Set[Item]
          RETORNA itens.proxy()
FIM MÉTODO
```




```
CLASSE Item
-- não se declara aqui a variável como nos casos anteriores
MÉTODO getVenda():Venda
          PARA TODA venda EM Venda.allInstances() FAÇA
                    SE venda.itens().includes(self) ENTÃO
                              RETORNA venda
                    FIM SE
          FIM PARA
FIM MÉTODO
FIM CLASSE
```

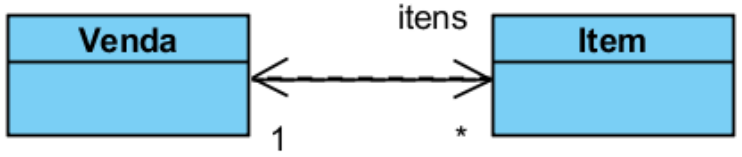
SE A ASSOCIAÇÃO REVERSA FOSSE PARA MUITOS:

```
MÉTODO getVendas():SET[Venda]
        VAR venda : SET[Venda]
        PARA TODA venda EM Venda.allInstances() FAÇA
                SE venda.itens().includes(self) ENTÃO
                        vendas.add(venda)
                FIM SE
        FIM PARA
        RETORNA vendas
FIM MÉTODO
FIM CLASSE
```

Implementação com objeto de associação

- Uma associação bidirecional também pode ser implementada através e um objeto intermediário representando a associação.
- O objeto intermediário consistirá de uma tabela com os pares de instância associadas e cada uma das classes participantes terá acesso a este objeto.

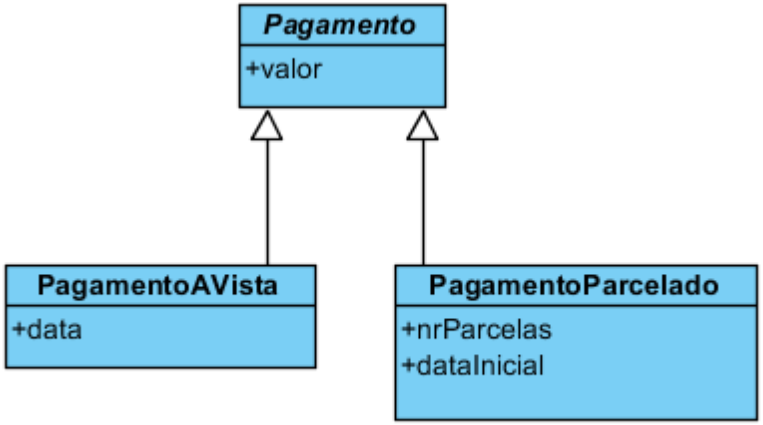
OBJETO DE ASSOCIAÇÃO



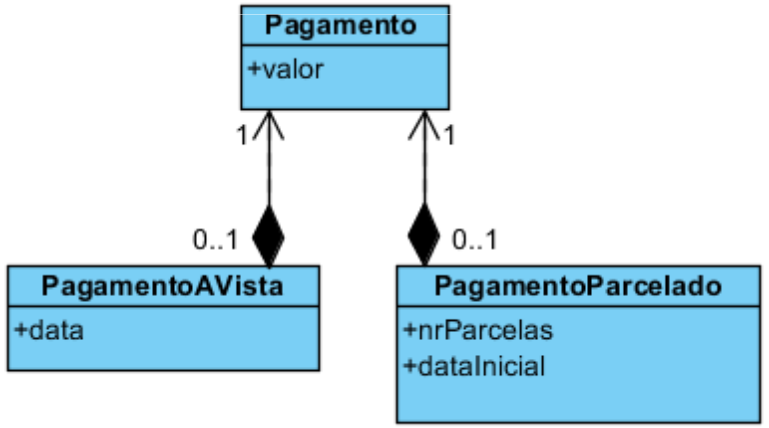

```
VAR GLOBAL venda_itens : RELAÇÃO[Venda, Item]

CLASSE Venda
MÉTODO addItens(umItem:Item)
        venda_itens.add(self,umItem)
FIM MÉTODO

MÉTODO removeItens(umItem:Item)
        venda_itens.remove(self,umItem)
FIM MÉTODO

MÉTODO getItens():Set[Item]
        RETORNA venda_itens.getKey(self)
FIM MÉTODO
```

```
CLASSE Item
MÉTODO getVenda():Venda
        RETORNA venda_itens.getValue(self)
FIM MÉTODO
FIM CLASSE
```

HERANÇA NO DB



Herança traduzida como *composição*




```
Context Pagamento inv:
        pagamentoAVista→size() + pagamentoParcelado→size() = 1
```

Representação no BD

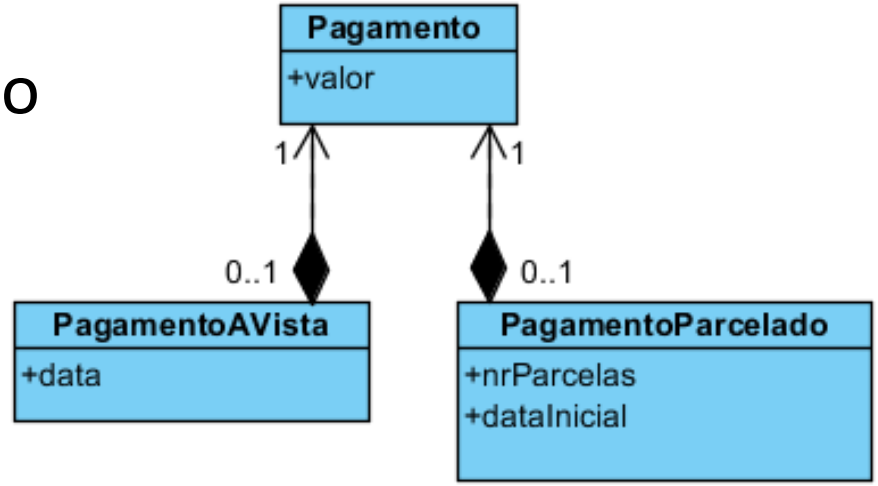


Tabela: Pagamento	
pkPagamento <<pk>>	valor
60001	105,00
60002	430,20
60003	28,51
60004	23,22
60005	24,42
60006	345,32
60007	32,85
60008	893,89
60009	326,22

Tabela: PagamentoAVista		
pkPagamentoAVista <<pk>>	data	pkPagamento
61001	20/10/2010	60001
61002	21/10/2010	60004
61003	25/10/2010	60007
61004	25/10/2010	60008

Tabela: PagamentoParcelado			
pkPagamentoParcelado <<pk>>	nrParcelas	dataInicial	pkPagamento
62001	12	20/10/2010	60002
62002	12	21/10/2010	60003
62003	12	22/10/2010	60005
62004	18	24/10/2010	60006
62005	6	30/10/2010	60009

Fim da primeira parte