

# Análise e Projeto Orientado a Objetos

**Modelagem de Domínio**

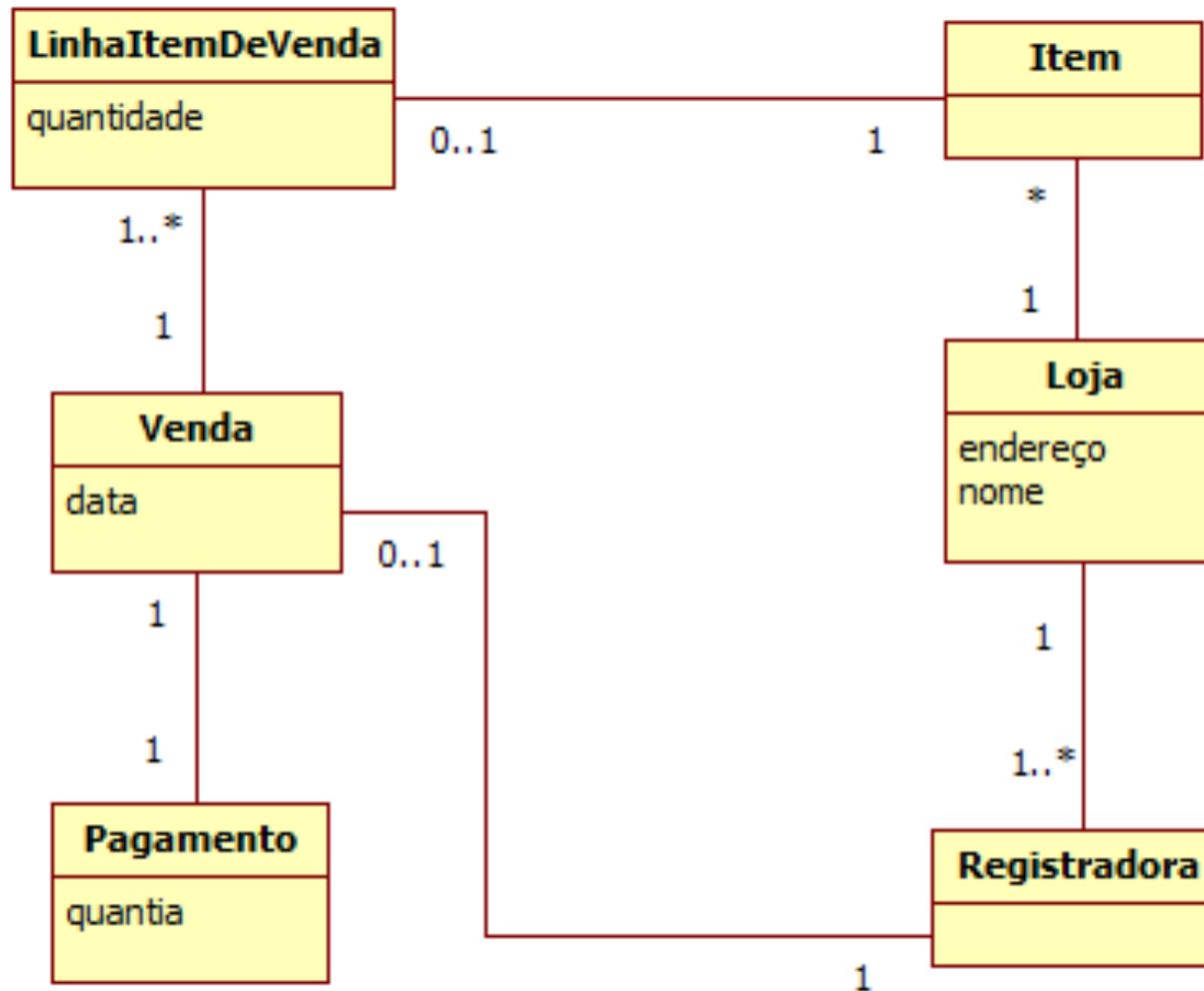
# Introdução

- A modelagem do domínio está relacionada à descoberta das informações que são gerenciadas pelo sistema. O resultado dessa investigação é expressa no **modelo de domínio**.
- Um **modelo** pode ser visto como uma representação idealizada de um sistema a ser construído (e.g, maquetes, plantas de circuitos, plantas arquitetônicas).
- Diagramas UML integram os modelos utilizados em desenvolvimento de software.
- Benefícios do uso de modelos: *melhor gerenciamento da complexidade* (abstração), melhor comunicação, redução dos custos, predição do comportamento do sistema.

# Introdução

- Denomina-se **domínio do negócio**, ou apenas domínio, a área de conhecimento específica na qual um determinado sistema de software será desenvolvido.
- Modelo de domínio é uma representação visual de conceitos do mundo real. É um dos artefatos mais importantes e mais utilizados em desenvolvimento de software.
- Diagramas de classes da UML são amplamente utilizados para expressar os conceitos do domínio. No entanto, diagramas do tipo entidade-relacionamento também possuem essa mesma aplicabilidade.

# Introdução



# Benefícios

- Melhor compreensão do domínio
- Estabelecimento de uma comunicação menos ambígua
  - Desenvolvedores e pessoas do negócio passam a falar a mesma língua (menor hiato representacional)
- Abstração: foco nos dados e conceitos que devem ser manipulados pelo sistema e não na solução tecnológica
  - Modelos de domínio são utilizados principalmente na identificação dos dados persistentes, mas podem ser aplicados em outras camadas ou módulos do sistema

# Perspectiva conceitual

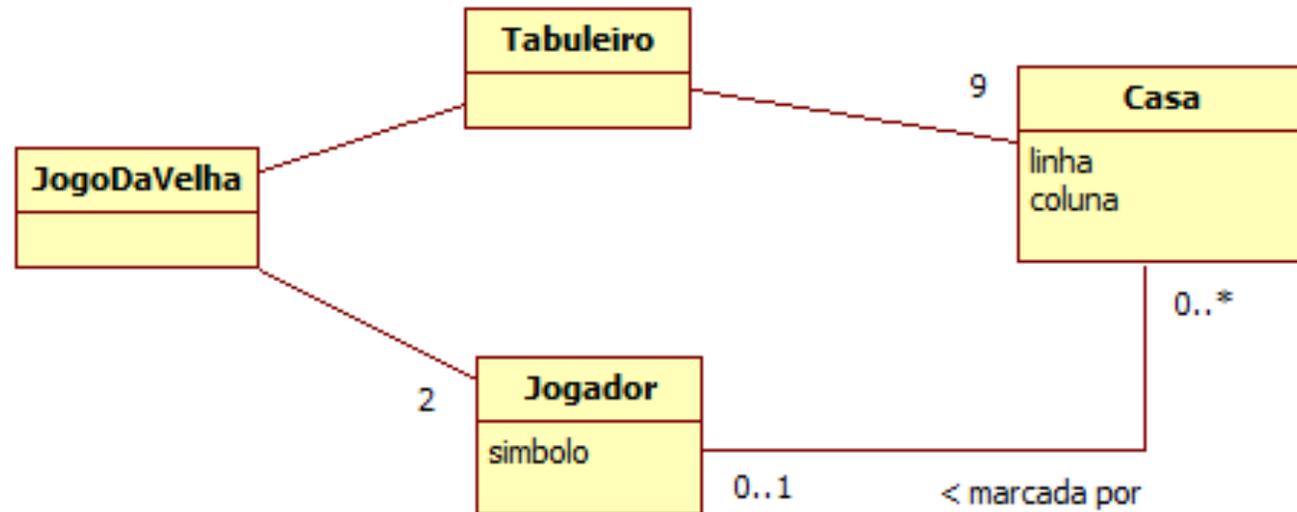
- Um modelo de domínio é uma descrição de coisas em uma situação real do domínio de interesse, **e não de objetos de software**
- O modelo de domínio faz parte da análise, ou seja da investigação do problema
  - Modelo de domínio não é projeto
- As classes em um diagrama de domínio representam conceitos e não classes de software a serem implementadas em um linguagem de programação
- Um modelo de domínio também não é um modelo de dados. Portanto, não exclua classes que não serão persistidas mas que possuem significado comportamental no domínio

# Perspectiva conceitual

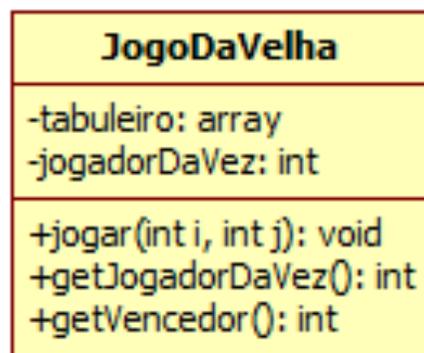
- O modelo de domínio descreve a informação que sistema deve gerenciar
  - Mas não indica como o sistema a deve gerenciar
- O modelo de domínio é estático. Então, não devem existir referências a operações ou aspectos dinâmicos dos sistemas
  - Embora o modelo conceitual seja representando pelo diagrama de classes da UML, o analista não deve ainda adicionar métodos a essas classes.

# Perspectiva conceitual

## ■ Análise:



## ■ Projeto:



# Elementos do modelo de domínio

- **Atributos:** informações alfanuméricas simples, como números, textos, datas, etc.
- **Classes ou conceitos:** que são a representação da informação complexa que agrega atributos e que não pode ser descrita meramente por tipos alfanuméricos.
- **Associações:** que consistem em um tipo de informação que liga diferentes conceitos entre si.

# Encontrando classes conceituais

- Algumas técnicas:
  - Revisar modelos existentes.
  - Lista de categorias. Larman apresenta um exemplo de lista com 16 categorias que são úteis na identificação inicial.
  - Modelagem CRC (classes, responsabilidades e colaboradores). Identifica as classes candidatas a partir de sessões envolvendo especialistas do negócio e desenvolvedores. As classes são registradas em cartões. Bezerra apresenta informações detalhadas sobre este método.
  - Análise linguística. Técnica simples e muito difundida. Realizada a partir de descrições textuais do domínio.

# Análise linguística

- A partir do texto dos casos de uso:
  - Passo 1: isole os substantivos e frases nominais
    - Também inclua os verbos relacionados a eventos que possuem informações importantes que devem ser guardadas pelo sistema
  - Passo 2: mantenha aqueles que são importantes para o sistema
  - Passo 3: agrupe os sinônimos
  - Passo 4: diferencie conceitos e atributos

# Atributos

- São os tipos escalares
- NÃO são estruturas de dados como listas, tabelas e arrays
- São sempre representados no contexto de uma classe:



# Tipagem

- Atributos podem ter tipos clássicos como string, inteiro, data, etc., ou tipos primitivos definidos pelo analista:



# Valores Iniciais

- Atributos podem ser definidos com valores iniciais.
- Valores iniciais são produzidos no atributo no momento que as instâncias da classe correspondente forem criadas

Venda
+data : Data
+valorTotal : Moeda = 0,00
+número : Natural

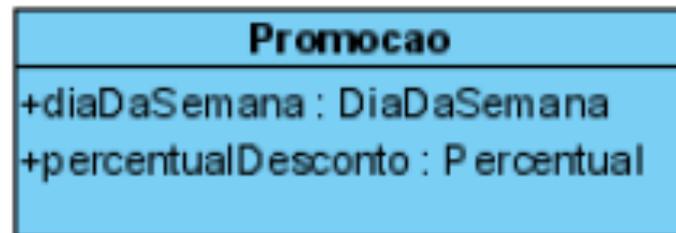
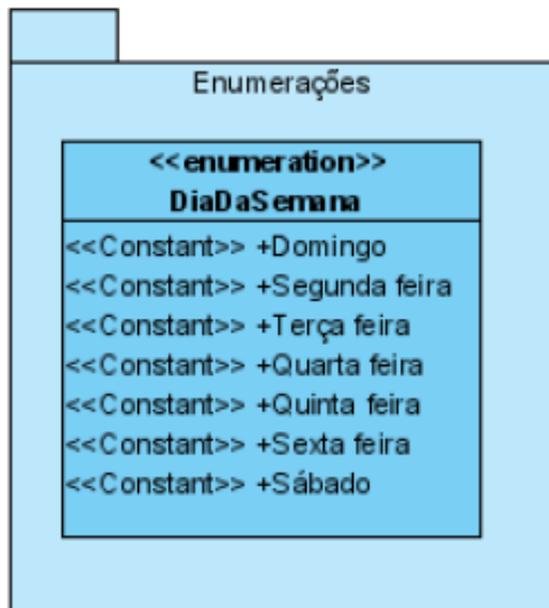
# Atributos Derivados

- Não são definidos diretamente, mas calculados

Produto
+precoCompra : Moeda
+precoVenda : Moeda
+ / lucroBruto : Moeda = precoVenda-precoCompra

# Enumerações

- São um meio termo entre o conceito e o atributo.
- São basicamente strings e se comportam como tal, mas há um conjunto predefinido de strings válidas que constitui a enumeração.

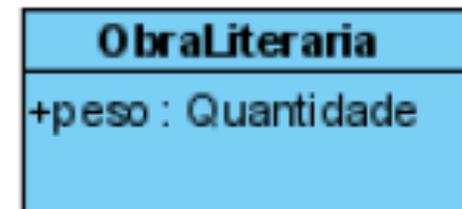
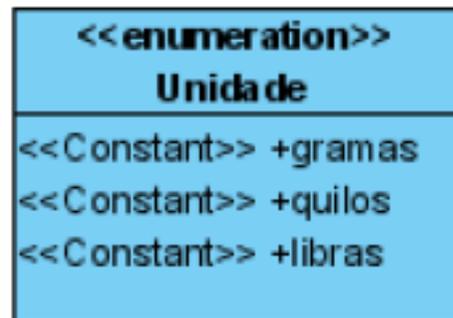
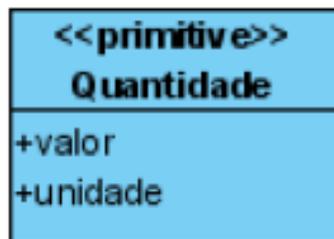


# Características de Enumerações

- NÃO podem ter associações com outros elementos.
- NÃO podem ter atributos.
- Se isso acontecer, então não se trata mais de uma enumeração, mas de um conceito complexo.

# Tipos Primitivos

- O analista pode e deve definir tipos primitivos sempre que se deparar com atributos que tenham regras de formação, como no caso do Quantidade em que sua formação é composta de valor e unidade..
- Tipos primitivos podem ser classes estereotipadas com <<primitive>>.



# Exemplo: CDU Comprar Livros

## ■ Fluxo Principal

1. [IN] Comprador informa sua identificação.
2. [OUT] Sistema informa os livros disponíveis para venda (título, capa e preço) e o conteúdo atual do carrinho de compras.
3. [IN] Comprador seleciona os livros que deseja comprar.
4. Comprador decide finalizar a compra.
5. [OUT] Sistema informa o valor total dos livros e apresenta as opções de endereço cadastradas.
6. [IN] Comprador seleciona um endereço para entrega.
7. [OUT] Sistema informa o valor do frete e total geral, bem como a lista de cartões de crédito já cadastrados para pagamento.
8. [IN] Comprador seleciona um cartão de crédito.
9. [OUT] Sistema envia os dados do cartão e valor da venda para a operadora.
10. [IN] Operadora informa o código de autorização.
11. [OUT] Sistema informa o prazo de entrega.

### • Fluxo alternativo (4): Comprador decide guardar carrinho

4a.1 [OUT] Sistema informa o prazo em dias em que o carrinho será mantido.

### • Fluxo de exceção 6a: Endereço consta como inválido

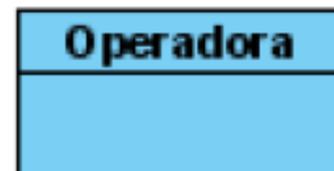
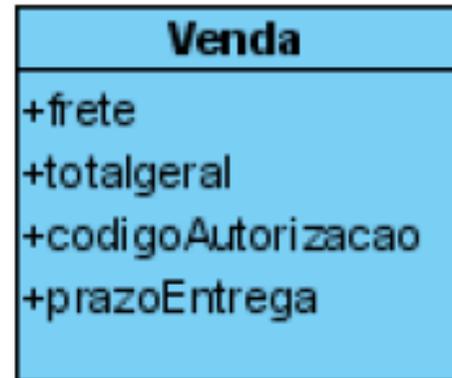
6a.1 [IN] Comprador atualiza o endereço e caso de uso segue para o passo 6.

### • Fluxo de exceção 10a: Operadora não autoriza a venda

10a.1 [OUT] Sistema apresenta outras opções de cartão ao comprador.

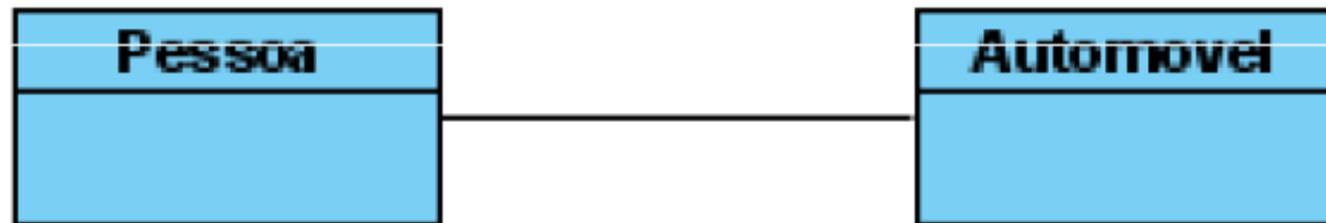
•10a.2 [IN] Comprador seleciona outro cartão e caso de uso segue para o passo 9.

# Exemplo: resultado



# Associações

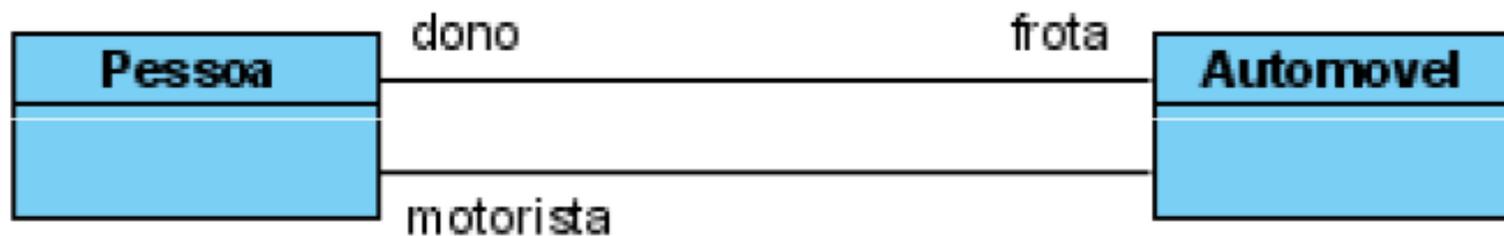
- Relacionam dois ou mais conceitos entre si.



# Papeis

- Correspondem à função que um lado da associação representa em relação aos objetos do lado oposto.

- Múltiplo:



# Como Encontrar Associações?

- Procure observar cada conceito complexo e se pergunte se a informação representada por ele é completa
- se não for, deve-se criar uma associação entre este conceito e outro(s) conceito(s) de forma a complementar a informação necessária para que o conceito faça sentido

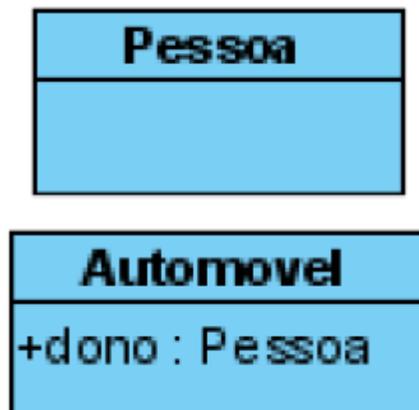
# Dependência entre Conceitos

- Conceitos dependentes (como Compra) precisam necessariamente estar ligados aos conceitos que os complementam (como Comprador e Item).
- Informações associativas só podem ser representadas através de associações

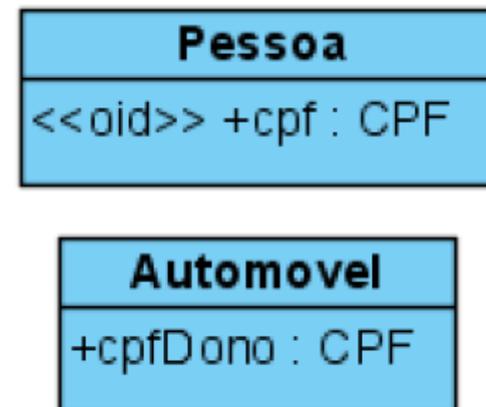
# Atributos Disfarçando Associações

- Não se deve colocar no modelo conceitual os atributos que representam “chaves estrangeiras”
  - como se fosse uma tabela de banco de dados relacional

**Errado**



**Errado**



# Multiplicidade de Papel

- Indica quantos objetos podem se associar.
- Sempre há um limite inferior.
- Pode haver um limite superior.
- Considerações de Multiplicidade
  - O papel é obrigatório ou não?
    - Uma pessoa é obrigada a ter pelo menos um automóvel?
    - Um automóvel deve obrigatoriamente ter um dono?
  - A quantidade de instâncias que podem ser associadas através do papel tem um limite conceitual definido?
    - Existe um número máximo ou mínimo de automóveis que uma pessoa pode possuir?

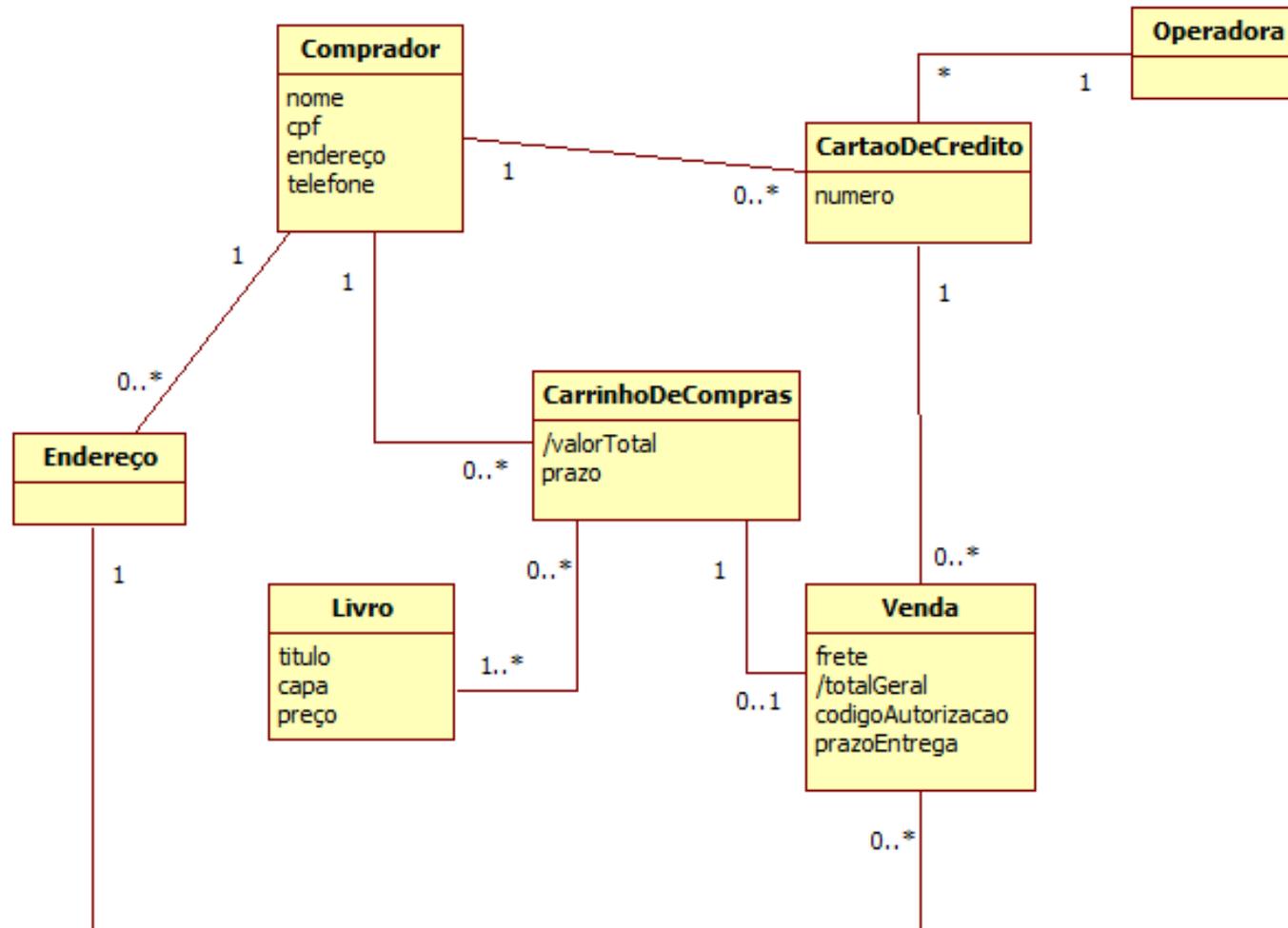
# Armadilha da Obrigatoriedade

- A toda venda corresponde um pagamento.
- Mas isso não torna a associação obrigatória, pois a venda pode existir sem um pagamento.
- Um dia ela possivelmente será paga, mas ela pode existir sem o pagamento por algum tempo.
- Então esse papel não é obrigatório para a venda.

# Armadilha do Limite Máximo

- O número máximo de automóveis que uma pessoa pode possuir é o número de automóveis que existe no planeta. Mas à medida que outros automóveis venham a ser construídos, esse magnata poderá possuí-los também.
- Embora exista um limite físico, não há um limite lógico para a posse.
- Então o papel deve ser considerado virtualmente sem limite superior.

# Exemplo: CDU Comprar Livros



# Exercício

- As informações a seguir se referem a uma aplicação de controle de comanda eletrônica da padaria Doce Sabor de Seu Joaquim.
  - O cliente utiliza uma comanda eletrônica durante suas compras na padaria. A cada produto consumido, o atendente registra em sua comanda (que possui uma numeração) o produto e a quantidade. Ao passar no caixa na saída da padaria, a caixa lê os gastos da comanda, finalizando a compra. Na leitura da comanda, verifica-se o valor unitário de cada produto a fim de calcular o valor total da compra.
- Faça um diagrama de classes conceituais desse cenário.

# Referências

- LARMAN, Craig. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objeto e ao desenvolvimento iterativo**. Porto Alegre: Bookman, 2007, 3<sup>a</sup> ed.
- WAZLAWICK, Raul Sidnei. **Análise e Projeto de Sistemas de Informação Orientados a Objetos**. Rio de Janeiro: Elsevier, 2011, 2<sup>a</sup> ed.
- BEZERRA, Eduardo. **Princípios de Análise Projeto de Sistemas com UML**. Rio de Janeiro: Campus, 2002.
- MELO, Ana Cristina. **Exercitando modelagem em UML**. Rio de Janeiro: Brasport, 2006.