



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Centro Tecnológico
Departamento de Informática

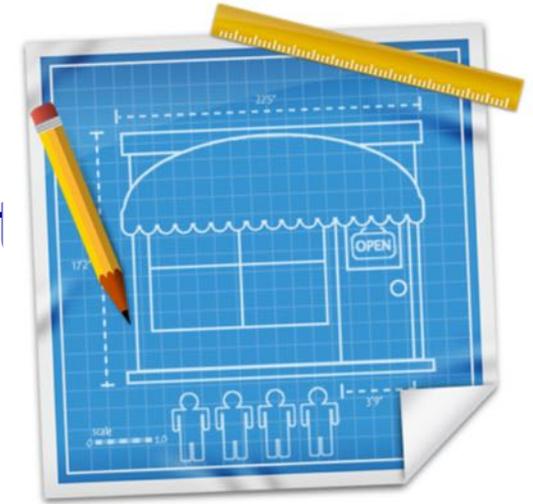
Prof. Vítor E. Silva Souza

<http://www.inf.ufes.br/~vitorsouza>

Modelagem OO
com UML

Modelos

- Maneira de **projetar, comunicar, documentar** soluções computacionais;
- Diversos **níveis**, por exemplo:
 - *Ontologias (modelos **genéricos**, de domínio);*
 - *Requisitos (foco em um **problema**);*
 - *Projeto / arquitetura (foco em uma **solução**).*
- Essenciais para o desenvolvimento de software;
- Também seguem os paradigmas (estruturado, OO, etc.).



Unified Modeling Language

- Padrão “**de facto**” para especificar, visualizar, documentar e construir artefatos de um sistema desenvolvido sob o **paradigma Orientado a Objetos**;
- Nasceu na **Rational** Software, desde 1997 é um padrão da Object Management Group (**OMG**);
- Reconhecido pela **ISO** em 2000;
- Teve origem em três outros métodos:
 - *OMT (Rumbaugh et al., 1994)*;
 - *Método de Booch (Booch, 1994)*;
 - *Método OOSE (Jacobson, 1992)*.

Versão atual:
2.5.1 (2017).

Diagramas da UML

- de Casos de Uso;
- de Classes;
- de Objetos;
- de Estrutura Composta;
- de Sequência;
- de Comunicação;
- de Estados;
- de Atividades;
- de Componentes;
- de Implantação;
- de Pacotes;
- de Interface Geral;
- de Tempo.

UNIFIED
MODELING
LANGUAGE

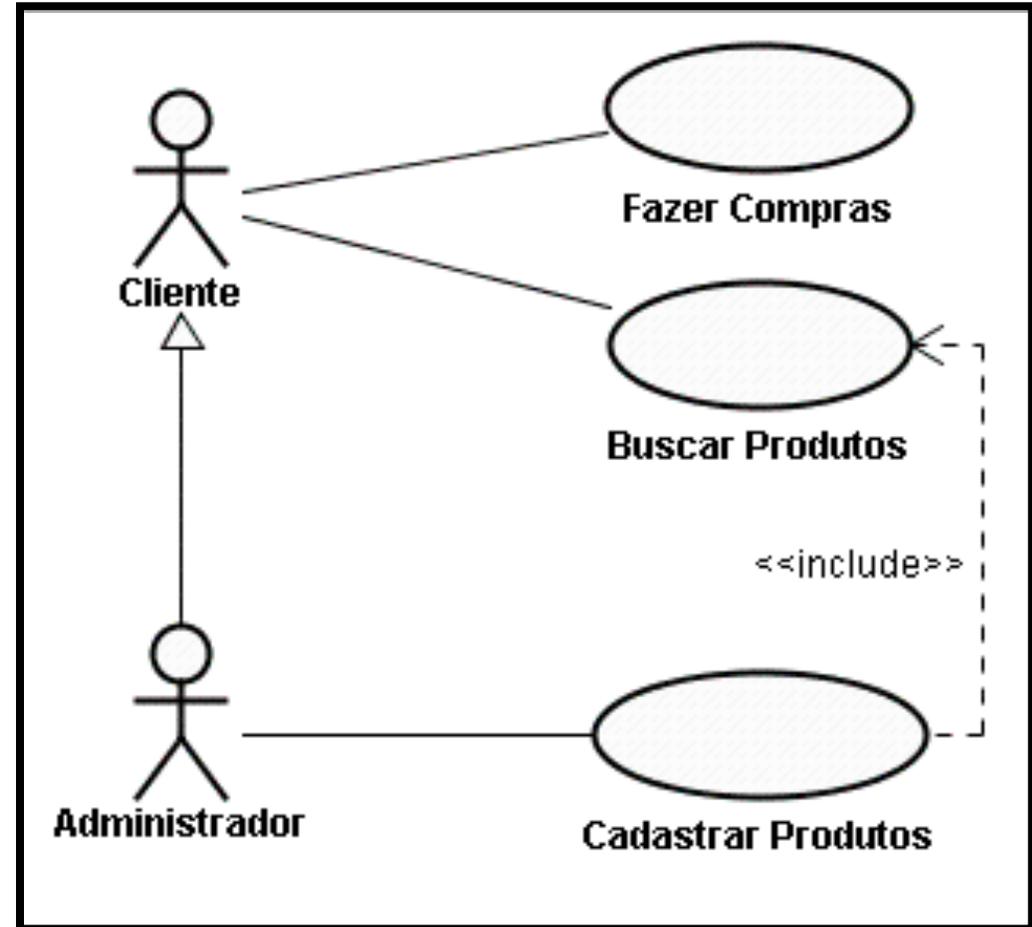


Modelagem Unificada?

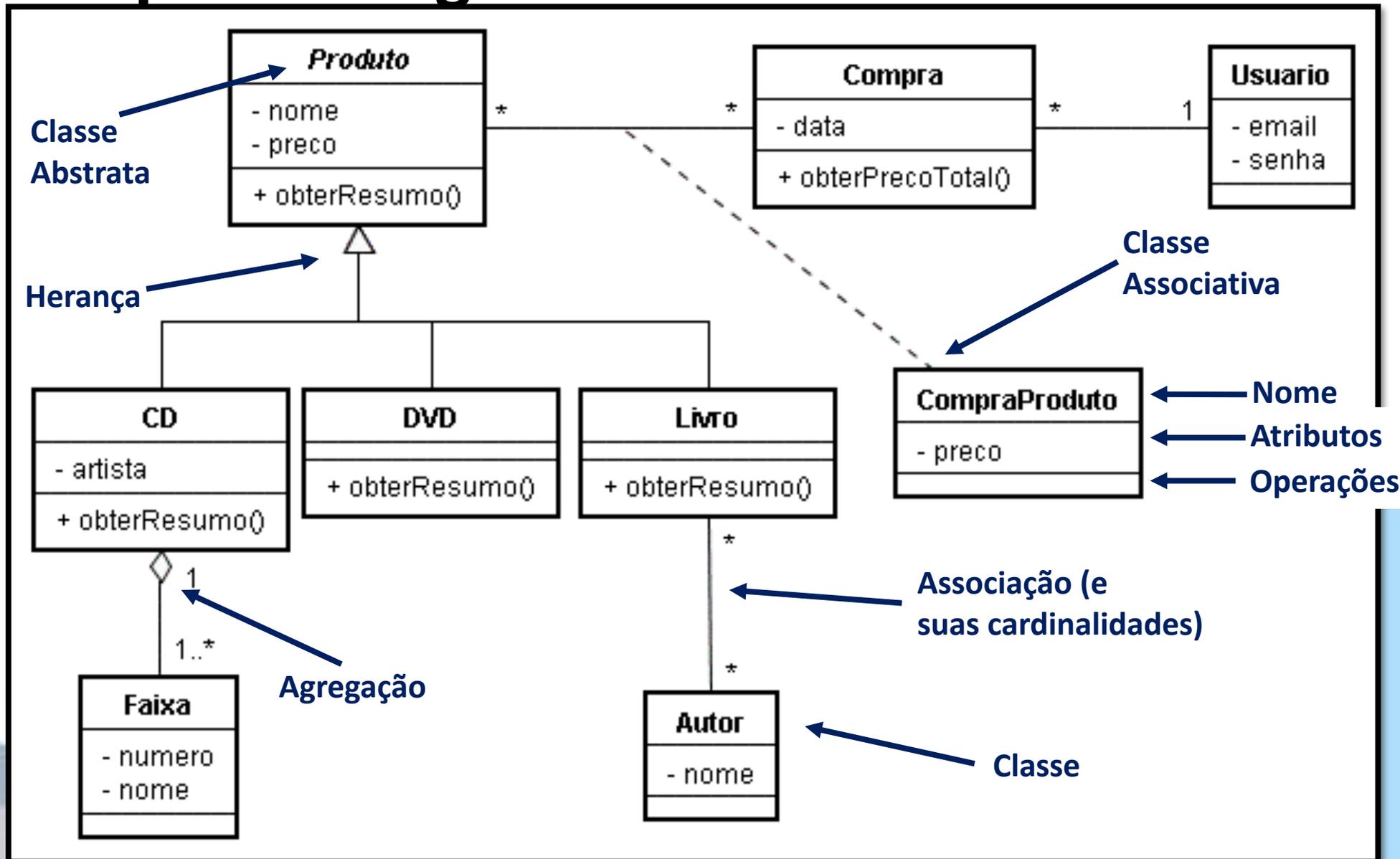
- A **notação** é unificada: quase todo desenvolvedor de software **conhece** ao menos parte da UML;
- A decisão de **qual artefato** (diagrama) produzir, porém, depende do **processo** definido para o projeto;
 - *Projetos diferentes, **necessidades** diferentes.*
- Pode ser utilizada em **diferentes processos** de desenvolvimento orientados a objetos, em **todas as etapas** do ciclo de desenvolvimento.
 - *Exemplo: domínio, requisitos, arquitetura, etc.*

Exemplo: Diagrama de Casos de Uso

- Modela as **funcionalidades** do sistema;
- Captura típicas **interações** usuário – sistema;
- Usuários são **atores**;
- Atores e **casos de uso** são associados;
- Cada caso é descrito em **detalhes** separadamente.



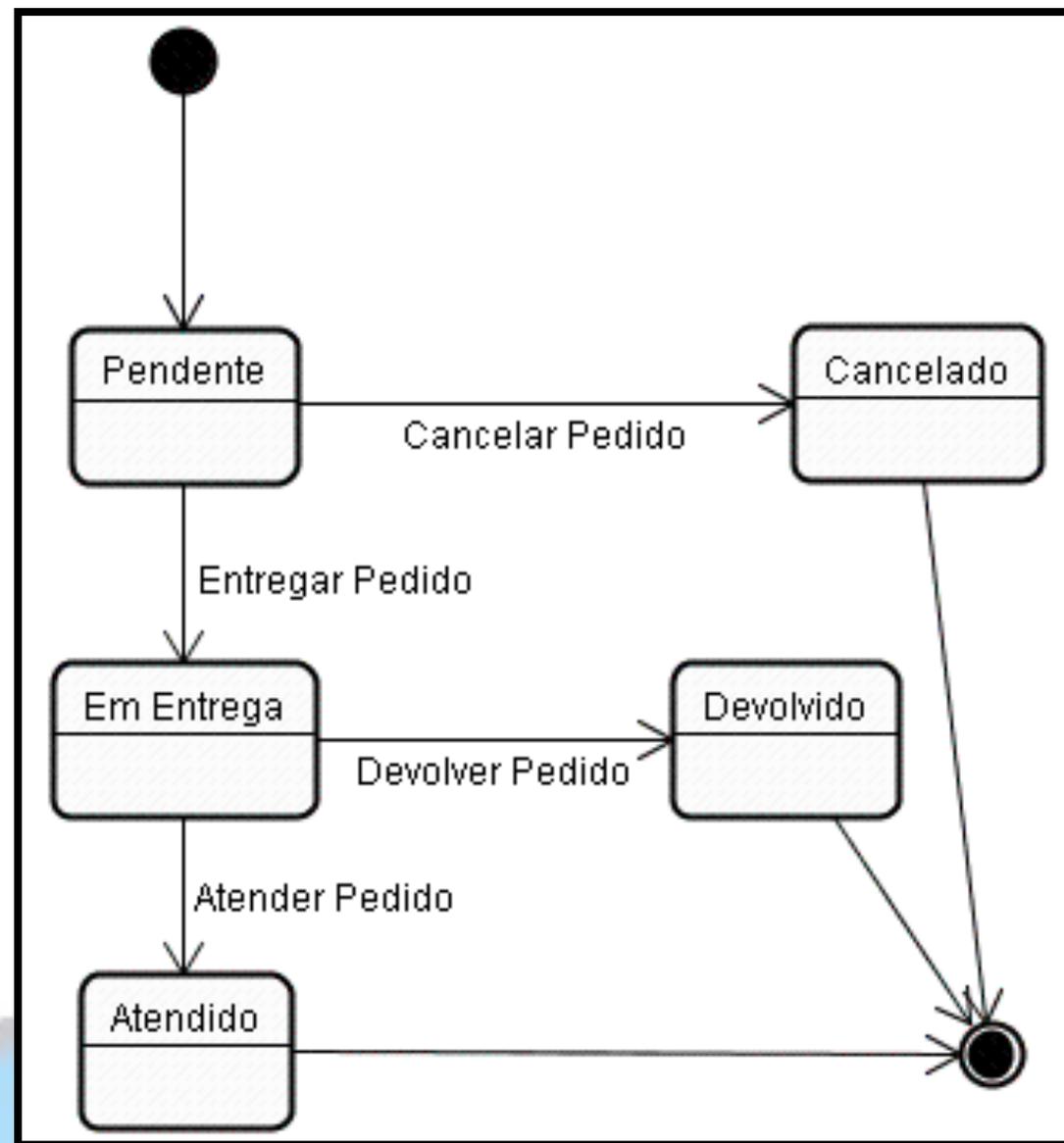
Exemplo: o Diagrama de Classes



Representa as classes relevantes (abstração!) para o domínio, problema ou solução.

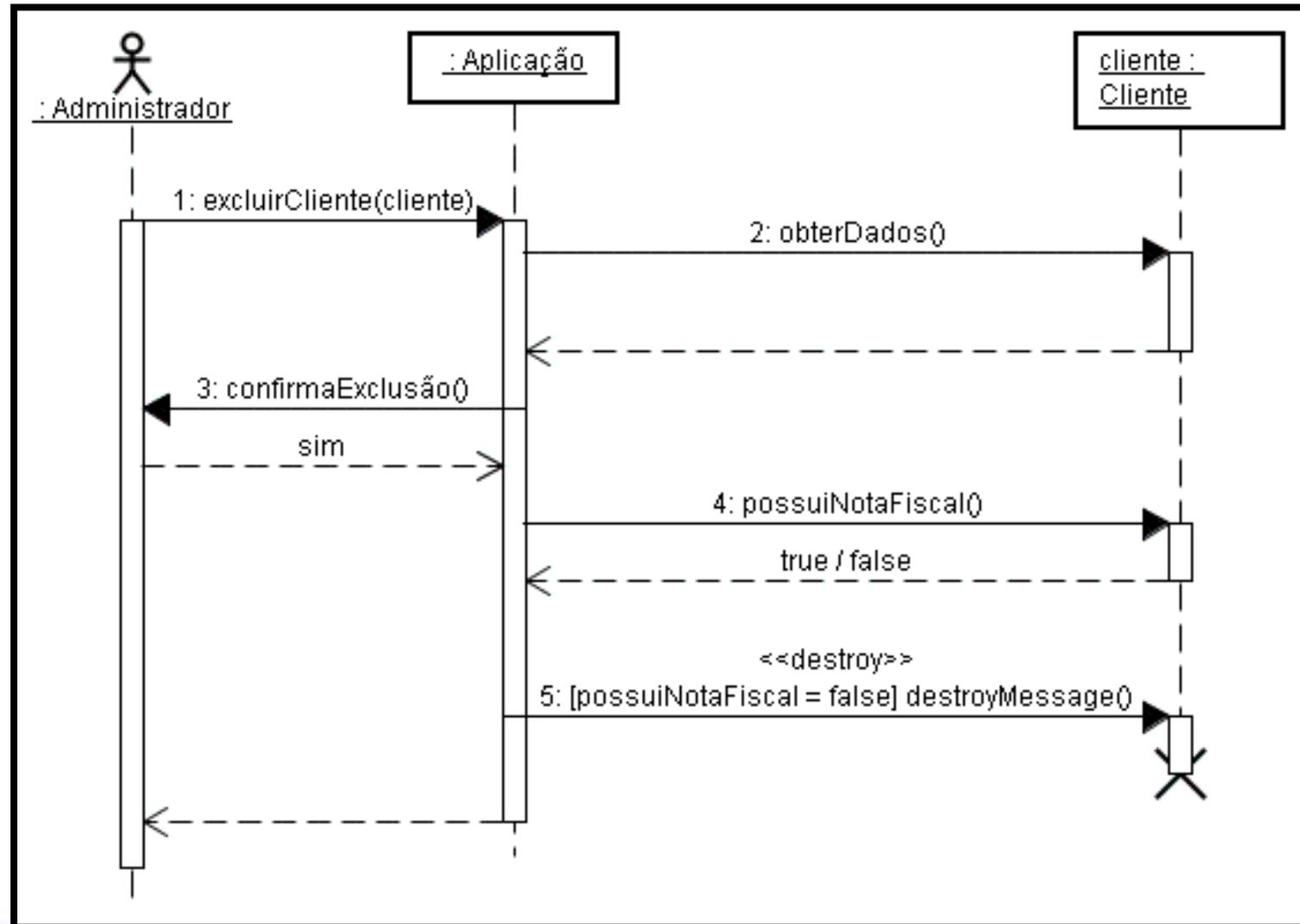
Exemplo: Diagrama de Estados

- Representa diferentes **estados** em que um **objeto** pode estar;
- Foco em uma **classe** de objetos específica (no exemplo, Pedido);
- Captura a **dinâmica** de um sistema, com foco numa **classe**.



Exemplo: Diagrama de Sequência

- Também captura a **dinâmica** de um sistema;
- Porém, o **foco** é em uma **função** específica.



Modelagem Estática

Modelagem estática

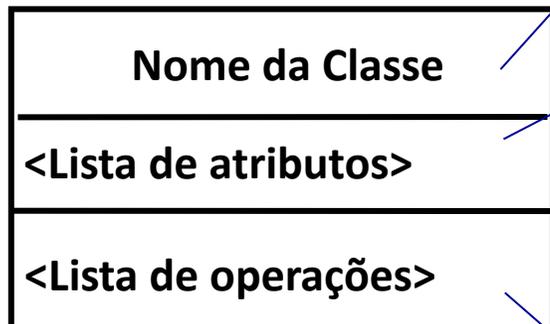
- Centrada no diagrama de **classes**:
 - *Identificação de classes;*
 - *Especificação de hierarquias de generalização / especialização;*
 - *Identificação de subsistemas;*
 - *Identificação de associações e atributos.*

Lembre-se: na orientação a objetos modelamos classes, portanto este diagrama é central!

Classes: níveis de abstração

- **Ontologias:**
 - *Conceitos de um **domínio**, relacionando-os com conceitos fundamentais;*
- **Análise** (de requisitos) de sistemas:
 - *Conceitos específicos do **problema** analisado;*
- **Projeto** (arquitetural) de sistemas:
 - *Modelos de **aplicação** (serviços), **interface** gráfica com o usuário, **persistência** de dados, etc.*
 - *Modelos de domínio mais **detalhados**.*

Representação de classes



Se estiver em itálico, a classe é abstrata.

Sintaxe: <escopo> <nome> : <tipo> =
<valor default>

Escopo:

- privado
- + público
- # protegido

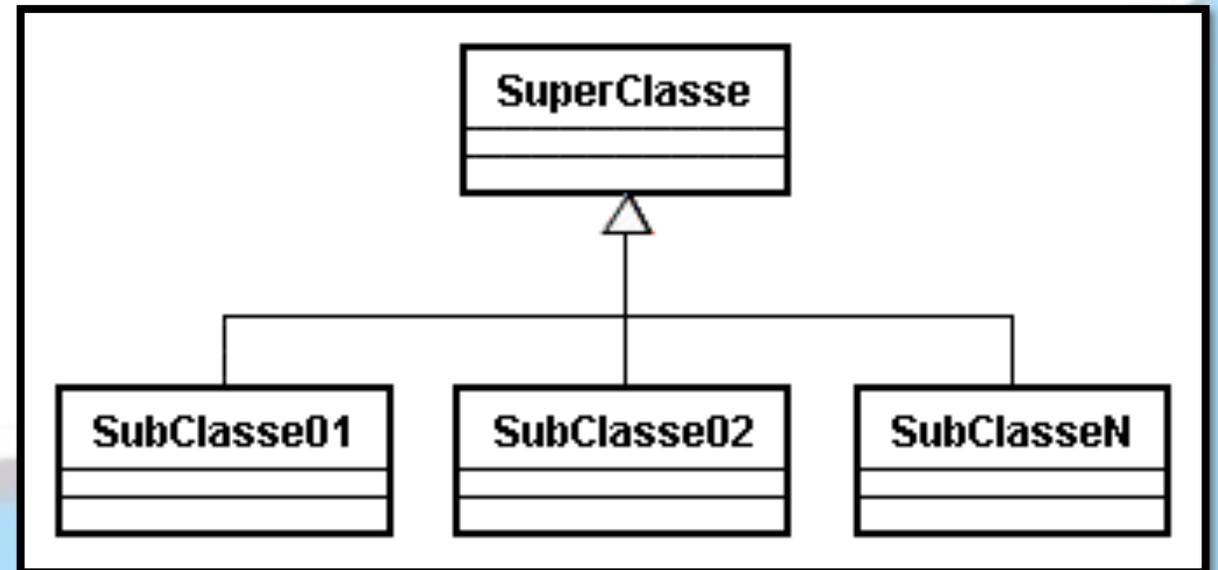
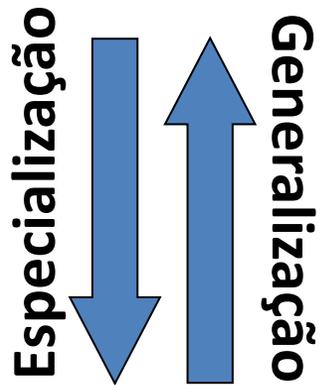
Sintaxe: <escopo> <nome> (<parâmetros>) :
<tipo>

<parâmetros> = lista de pares “<nome> :
<tipo>”, separada por vírgula.

Dependendo do nível de abstração, alguns detalhes podem ser omitidos
(ex.: tipo e escopo na fase de análise).

Herança (inheritance)

- Devem modelar relações “é-um-tipo-de”;
- Subclasses devem **suportar** toda a **funcionalidade** das superclasses e possivelmente **mais**;
- Funcionalidade **comum** a **diversas** classes deve estar o mais **alto** possível na hierarquia;
- Classes abstratas **não podem** herdar de classes concretas.

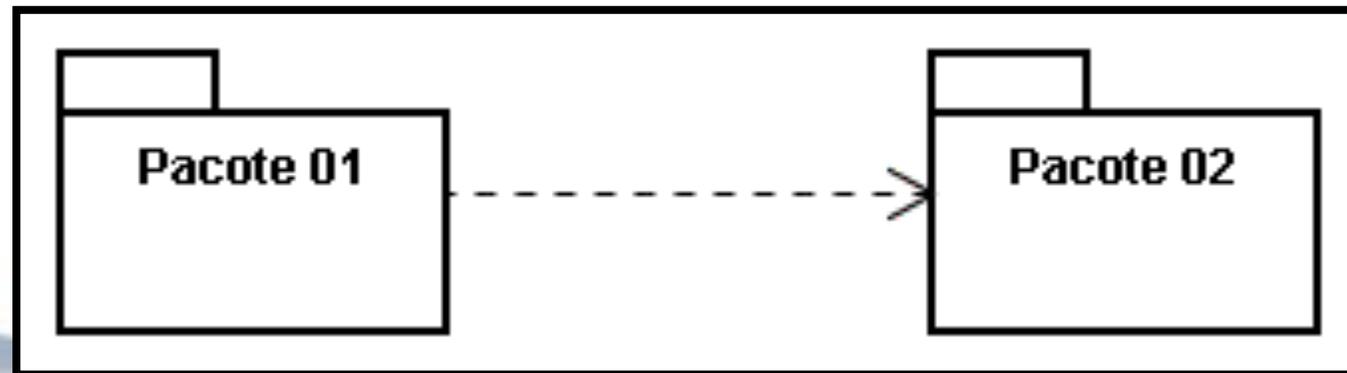


Separação em subsistemas / módulos

- Projetos **grandes** podem conter **centenas** de classes e estruturas diversas;
- **Divisão** das classes em **pacotes**:
 - *Coleção de classes que **colaboram** entre si;*
 - *Conjunto **coeso** de responsabilidades;*
 - *“**Caixa preta**”.*
- **Vantagens**:
 - *Facilita o **entendimento** para leitores;*
 - *Auxilia na **organização** de grupos de trabalho;*
 - *Organiza a **documentação**;*
 - *Em suma, facilita a **manutenção**.*

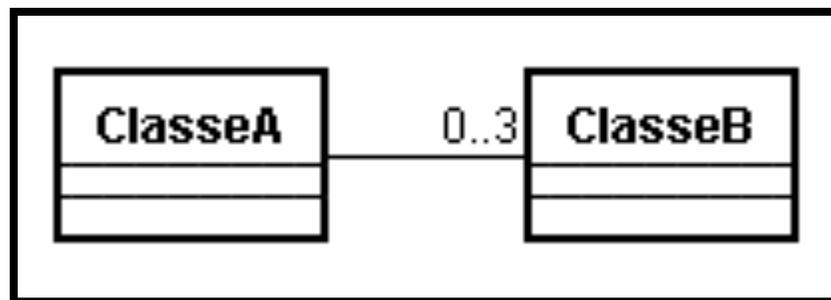
Pacotes (packages)

- Podem ser usados para **organizar** diversos tipos de **elementos de modelos**, inclusive **diagramas** inteiros;
- Muito **utilizados** para organizar **classes** em módulos, da mesma forma que será feito em **Java/C++**;
- É possível representar relação de **dependência** entre pacotes:

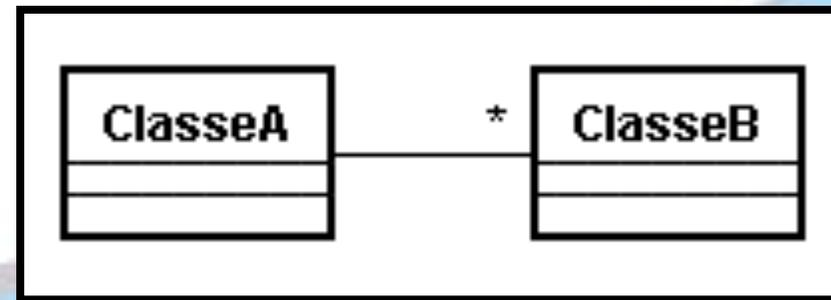
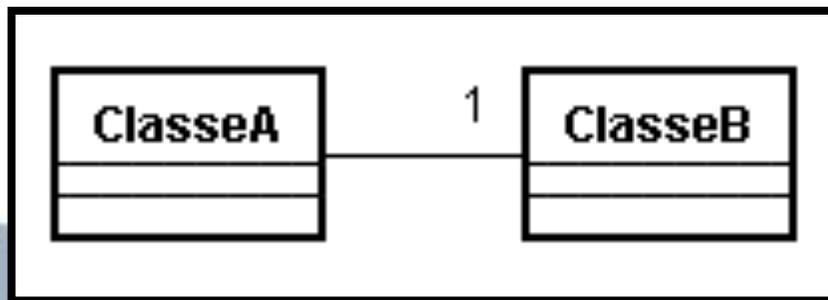


Associações (associations)

- Relacionamento entre classes é representado por associações, agregações e composições;
- Associações podem indicar **cardinalidade** (cardinality):

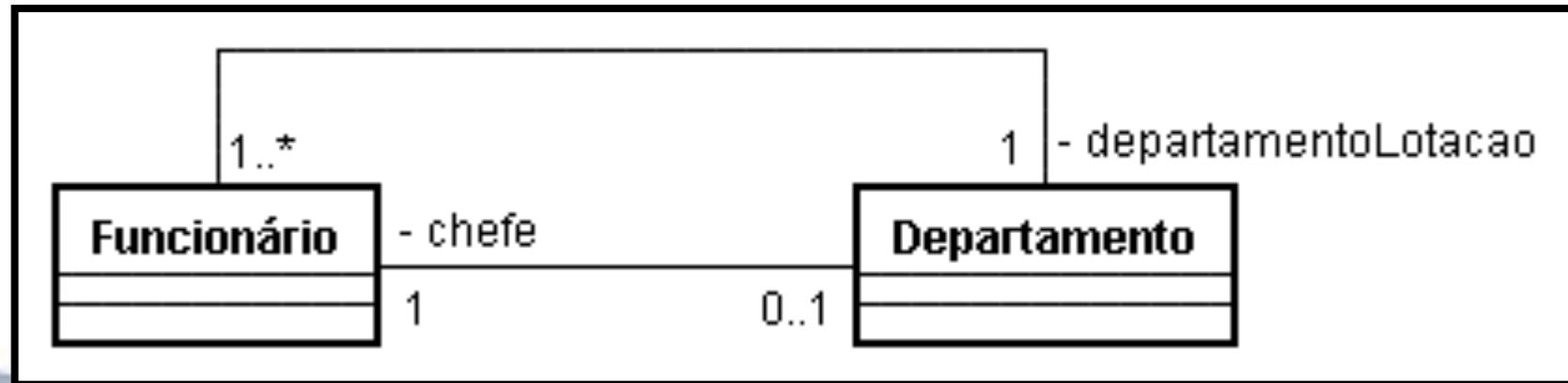


← Objetos da ClasseA podem se relacionar com no mínimo zero e no máximo três objetos da ClasseB.



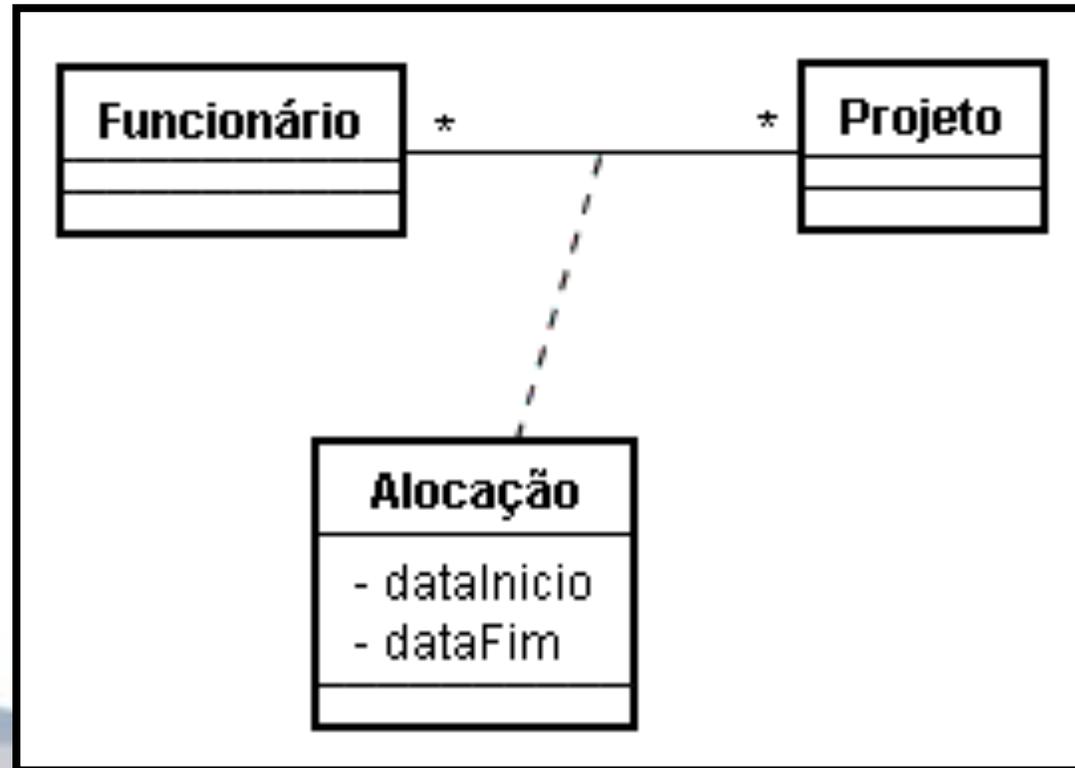
Papéis (roles)

- Indicam o papel que a classe **desempenha** na **associação** (são usados substantivos);
- É **opcional**, usado quando melhora o **entendimento** do modelo;
- **Sintaxe**: <escopo> <nome>.



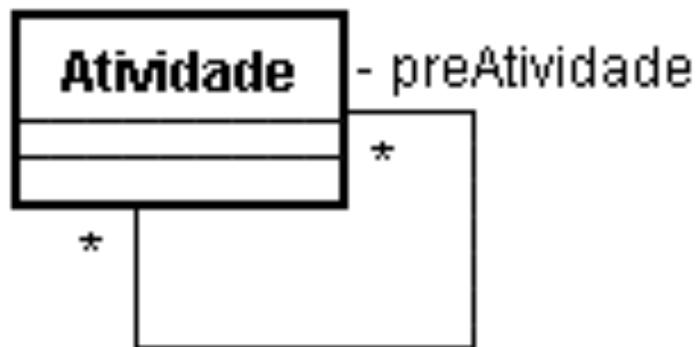
Classes associativas (association class)

- Utilizadas quando a **associação** possui **atributos**;
- Comuns em relações **n-para-n**.



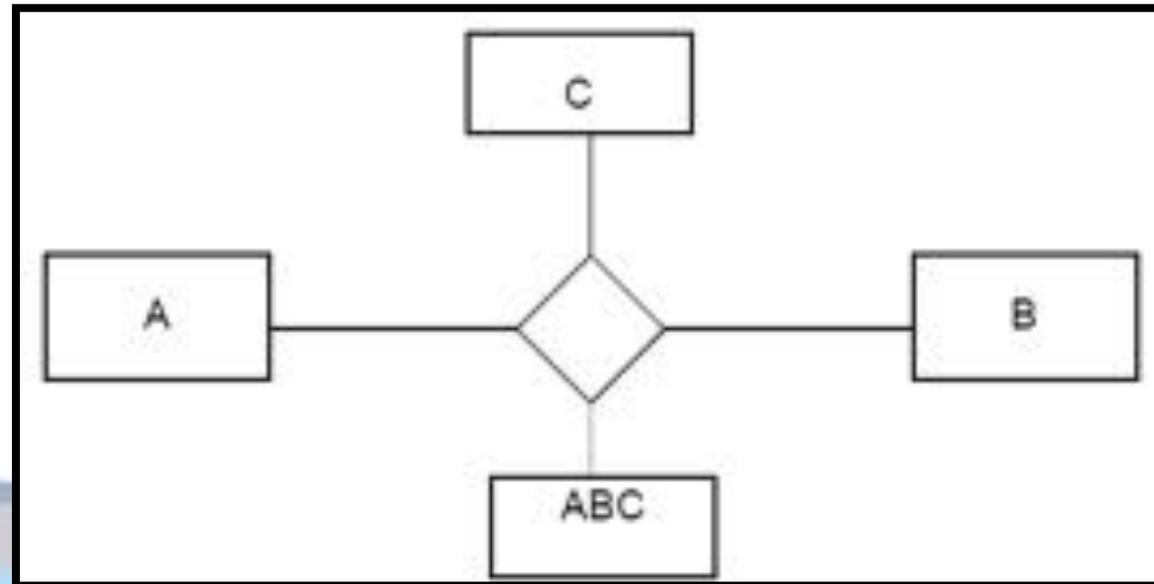
Relacionamentos recursivos

- Perfeitamente **legais**;
- Geralmente **pedem** definição de **papéis**.



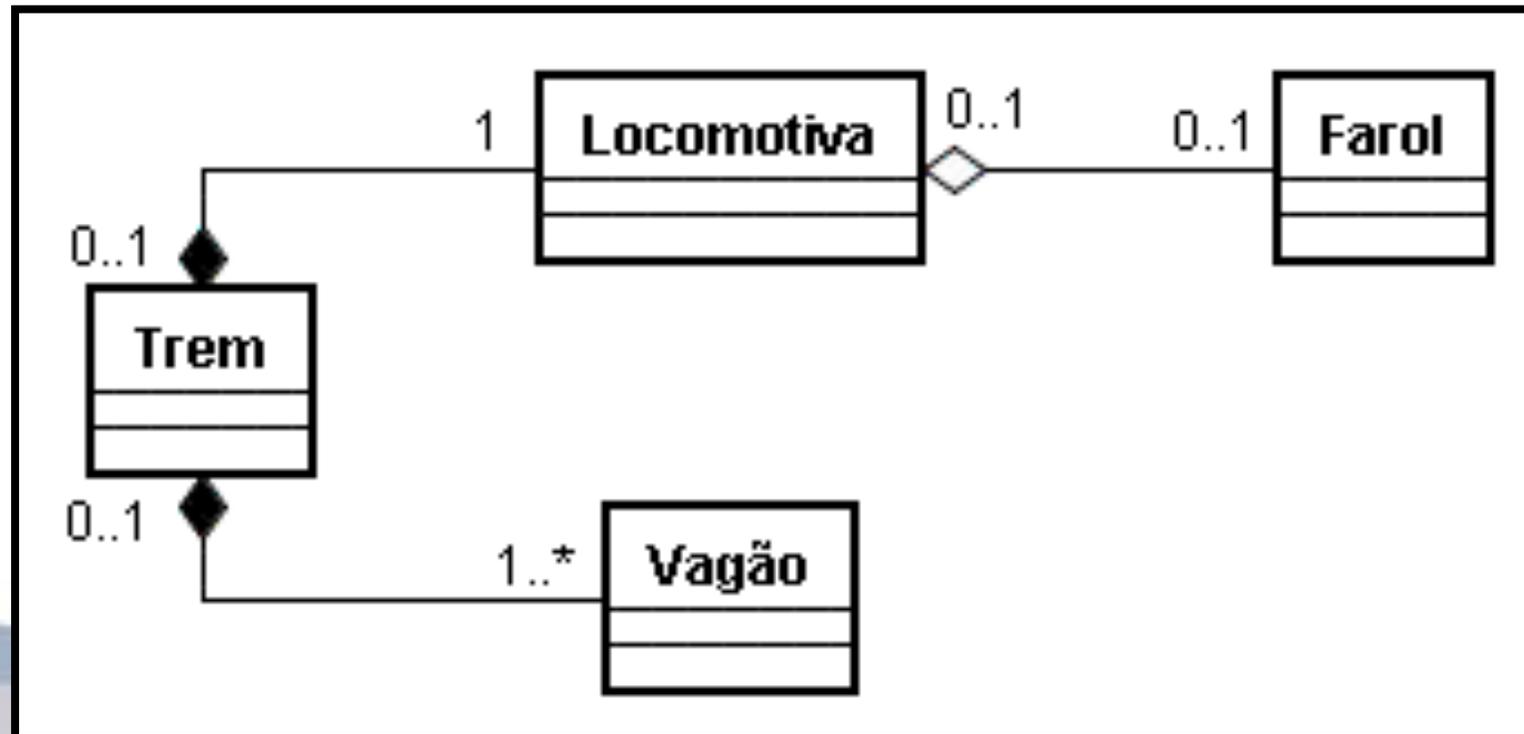
Associações n-árias

- Associações entre **três ou mais** classes;
- Extremamente **raras**, muitas vezes as ferramentas CASE nem dão **suporte**;
- Podem ser **substituídas** por uma nova **classe e N associações**.



Agregação e composição

- Relações todo-parte;
- Adicionam um **losango** à sintaxe, na extremidade da classe que representa o todo:

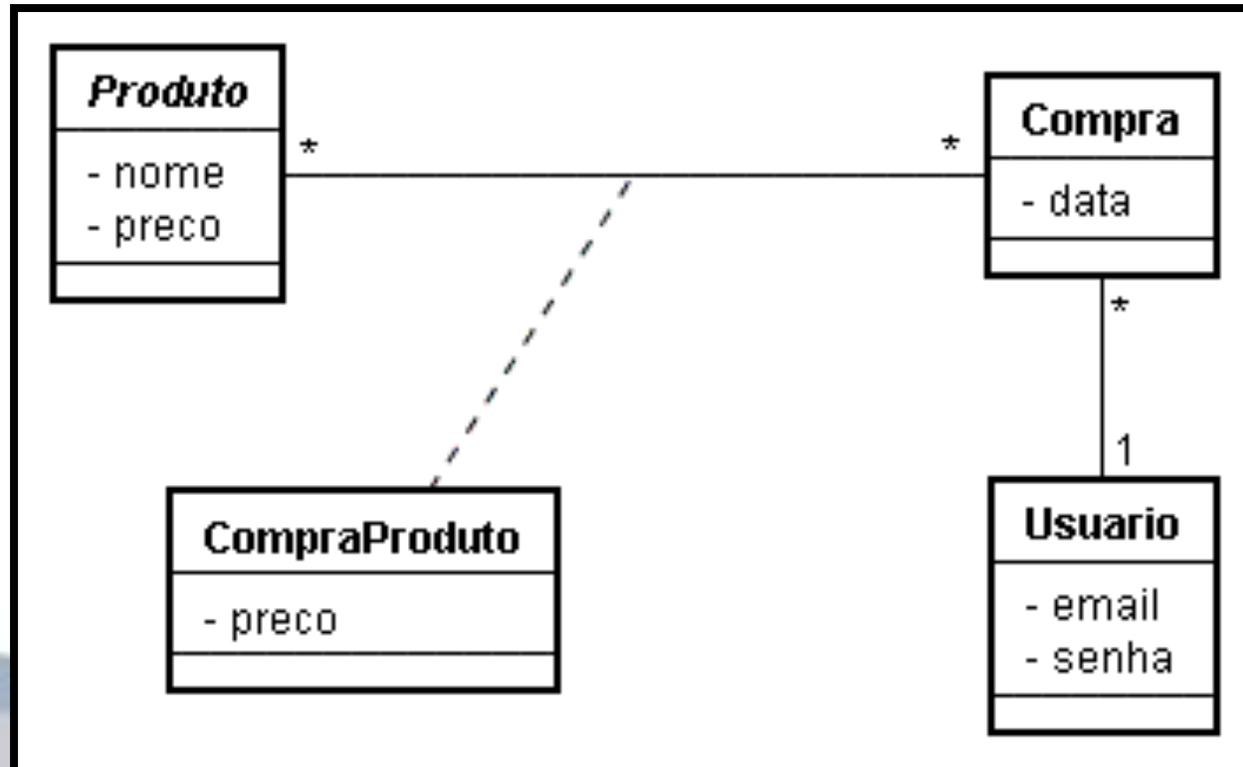


Atributos (attributes)

- Atributos são informações de **estado** (propriedades) para o qual cada **objeto** em uma classe tem seu **valor**;
- Muito **similares** às associações:
 - *Como atributos têm um tipo, podemos considerar que são associações com um tipo;*
 - *Para tipos primitivos definimos atributos, do contrário modelamos uma associação;*
 - *Em última instância, associações e atributos são implementados da mesma forma;*
 - *Atributos e associações definem uma classe.*

Especificação de atributos

- Escolha um nome com **significado**;
- Siga um **padrão** de nomenclatura;
- Inclua-o na **modelagem** de classes:



Atributos e hierarquias de classe

- Atenção à **hierarquias** de classes:
 - Atributos **genéricos** ficam mais **acima** na hierarquia;
 - Por outro lado, se ele **não se aplica** a algumas subclasses, deve ser trazido “para baixo”, somente para as **classes apropriadas**.
- **Revisão** da hierarquia:
 - Descoberta de **atributos** nos leva a um melhor **entendimento**, o que possivelmente implicará **revisão** de hierarquias.

Exemplo

A empresa de entrega de refeições à domicílio Disque-Rango deseja um sistema de informação para melhor atender seus clientes. Clientes fazem pedidos, discriminando um ou mais itens de cardápio e suas respectivas quantidades (por exemplo, João faz um pedido para receber em casa 2 lasanhas, 1 filé com fritas e 3 latas de cerveja). De um cliente deseja-se saber: nome, endereço, telefone e ponto de referência.

Itens de cardápio podem ser de três tipos: refeições, sobremesas e bebidas. É necessário saber o nome do item de cardápio e seu tipo e valor, sendo que das bebidas é necessário saber também a quantidade em estoque. Não são aceitos pedidos com quantidades de bebidas superiores às quantidades em estoque. Toda vez que bebidas forem compradas, deve-se atualizar a quantidade em estoque.

Uma vez que um pedido é feito, ele é considerado pendente até que seja passado para um entregador. Quando o entregador retorna com o pagamento, o pedido é considerado atendido. Apenas pedidos ainda pendentes podem ser alterados ou cancelados pelo cliente. No último caso, o pedido é excluído do sistema. Caso o entregador não encontre o cliente em seu endereço e retorne com os produtos, o pedido deve ser considerado devolvido e não deverá ser excluído do sistema. Os clientes com três devoluções de pedidos são desativados e só poderão fazer novos pedidos se forem reativados pelo funcionário.

De um pedido deseja-se saber os itens pedidos, a data, o entregador, o cliente e, caso tenha sido pago em cheque, o número do cheque, conta, agência e banco. De um entregador deseja-se saber nome e placa de seu veículo.

Exemplo

