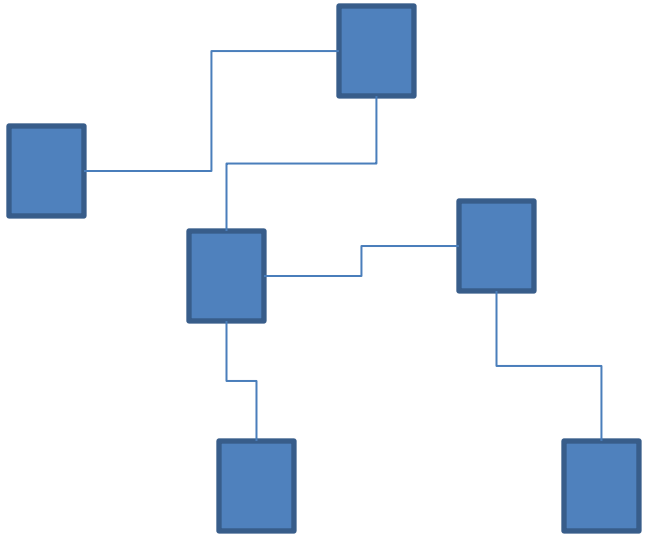
Diagrama de Classes

O que é?

- Diagrama mais utilizado da UML
- Representa os tipos (classes) de objetos de um sistema
  - Propriedades desses tipos
  - Funcionalidades providas por esses tipos
  - Relacionamentos entre esses tipos
- Pode ser mapeado diretamente para uma linguagem de programação orientada a objetos
  - Ajuda no processo transitório dos requisitos para o código
  - Pode representar visualmente o código do sistema

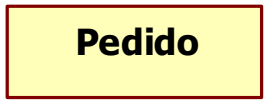
A 1 km de distância...

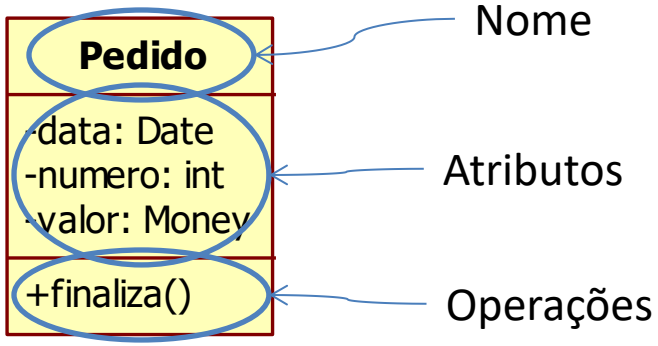
- Caixas representando as classes
- Linhas representando os relacionamentos



A 1 metro de distância...

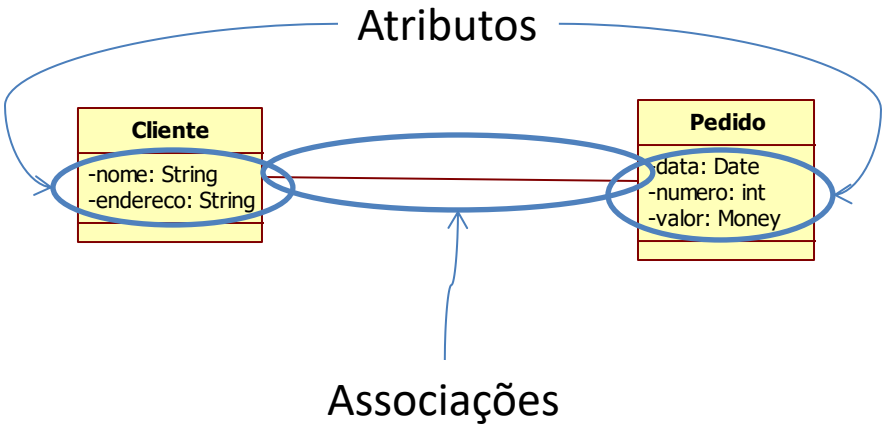
- As classes são representadas por caixas contendo
  - Nome (obrigatório)
  - Lista de atributos
  - Lista de operações





Propriedades

- Classes são descritas via suas propriedades
  - Primitivas: representadas como atributos
  - Compostas: representadas como associações para outras classes
- Quando transformadas para código, as propriedades se tornam sempre campos (atributos) da classe



A 1 centímetro de distância... dos atributos

- Atributos são descritos via
  - Visibilidade
  - Nome
  - Tipo
  - Multiplicidade
  - Valor padrão

- endereco : String[1] = “Sem Endereço”

Atributos (visibilidade)

- Privado (-)
  - Somente a própria classe pode manipular o atributo
  - Indicado na maioria dos casos
- Pacote (~)
  - Qualquer classe do mesmo pacote pode manipular o atributo
- Protegido (#)
  - Qualquer subclasse pode manipular o atributo
- Publico (+)
  - Qualquer classe do sistema pode manipular o atributo

- endereco : String

Atributos (nome e tipo)

- O nome do atributo corresponde ao nome que será utilizado no código fonte
  - É aceitável utilizar nomes com espaço e acentos na fase de análise
- O tipo do atributo corresponde ao tipo que será utilizado no código fonte
  - Tipos primitivos da linguagem
  - Classes de apoio da linguagem (String, Date, Money, etc.)

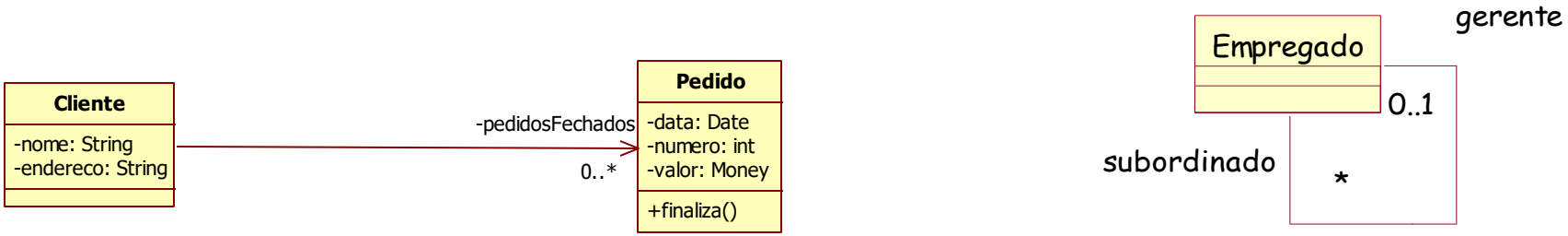
Atributos (multiplicidade)

- Representa o número de elementos de uma propriedade
- Estrutura X..Y onde
  - Opcional: X = 0
  - Mandatório: X = 1
  - Somente um valor: Y = 1
  - Multivalorado: Y > 1
- Valores clássicos
  - 0..1
  - 1 (equivalente a 1..1 → default)
  - * (equivalente a 0..*)
  - 1..*

- endereco : String[0..3]

A 1 centímetro de distância... das associações

- Associações
  - Guarda as mesmas informações dos atributos
  - Utiliza uma notação gráfica
  - Deve ser utilizado para propriedades que são relevantes ao diagrama
  - Determina o papel das classes na associação
  - Determina o sentido de navegação



A 1 centímetro de distância... das operações

- Operações são descritas via
  - Visibilidade
  - Nome
  - Lista de parâmetros
  - Tipo de retorno

+ finaliza(data : Date) : Money

Operações (visibilidade)

- Valem as mesmas regras de visibilidade de atributos
- Privado (-)
  - Funcionalidades de apoio à própria classe
- Pacote (~)
  - Funcionalidades de apoio a outras classes do pacote (ex. construção de um componente)
- Protegido (#)
  - Funcionalidades que precisam ser estendidas por outras classes (ex. construção de um *framework*)
- Publico (+)
  - Funcionalidades visíveis por todas as classes do sistema

Operações (nome e tipo de retorno)

- Valem as mesmas regras já vistas para atributos...
  - Normalmente o nome de uma operação é formado por um verbo (opcionalmente seguido de substantivo)
  - A ausência de um tipo de retorno indica que a operação não retorna nada (i.e., *void*)

+ finaliza(data : Date) : Money

Operações (lista de parâmetros)

- A lista de parâmetros pode ser composta por zero ou mais parâmetros separados por vírgula
  - Parâmetro: [direção] nome : tipo [= valor padrão]
  - Direção (opcional)
    - in (default)
    - out
    - inout
  - Nome
  - Tipo
    - Primitivo
    - Classe
  - Valor padrão (opcional)

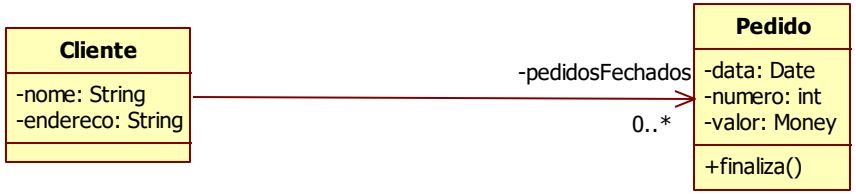
+ finaliza(data : Date) : Money

Em análise...

- Não se atenha aos detalhes
  - Visibilidade
  - Navegabilidade
  - Tipo
- Visibilidade pública em propriedades
  - Assume campo privado e métodos de acesso (get e set)
- Operações
  - Somente as responsabilidades obvias das classes

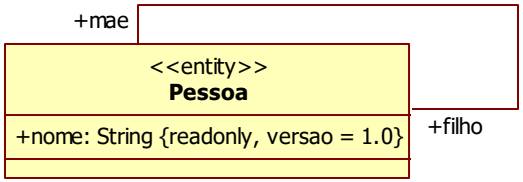
Exercício

- Traduza o seguinte diagrama em código Java
- Crie métodos de acesso para as propriedades da classe Cliente



Palavras-chave, propriedades e restrições

- Apóiam a linguagem gráfica com informações textuais
- Permitem dar mais semântica aos elementos do modelo
- Notação de palavra-chave
  - Textual: <<palavra>> (ex.: <<interface>>)
  - Icônica: imagem representando a palavra-chave
- Notação de propriedades e restrições
  - {propriedade} (ex.: {readonly})
  - {nome = valor} (ex.: {versão = 1.0}
  - {restrição} (ex.: {Mãe deve ser do sexo feminino})



Propriedades de atributos e associações

- Alguns exemplos...
- {readonly}
  - Somente oferece operações de leitura
- {ordered}, {unordered}
  - Indica se o atributo ou associação multivalorado mantém a sequência dos itens inseridos
- {unique}, {nonunique}
  - Indica se o atributo ou associação multivalorado permite repetição

- endereco : String = “Sem Endereço” {readonly}

Propriedades de operações

- {query}
  - Não modifica o estado do sistema após a execução
- {sequential}
  - A instância foi projetada para tratar uma thread por vez, mas não é responsabilidade da classe assegurar que isso ocorra
- {guarded}
  - A instância foi projetada para tratar uma thread por vez, e é responsabilidade da classe assegurar que isso ocorra (ex.: métodos synchronized em Java)
- {concurrent}
  - A instância é capaz de tratar múltiplas threads concorrentemente

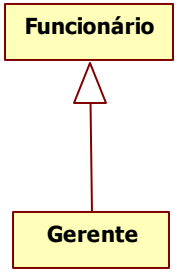
+ finaliza(data : Date) : Money {sequential}

Outros relacionamentos entre classes

- Além das associações, alguns outros tipos de relacionamentos são importantes
  - Generalização
  - Composição
  - Agregação
  - Dependência
  - Classes de associação

Generalização

- Visa estabelecer relações entre tipos
- Leitura: “é um”
- Se Gerente “é um” Funcionário
  - Todas as operações e propriedades (não privadas) de Funcionário vão estar disponíveis em Gerente
  - Toda instância de Gerente pode ser utilizada onde se espera instâncias de Funcionário
  - Gera o efeito de herança e polimorfismo quando mapeado para código



Agregação

- É uma associação com a semântica de “contém”
- Serve como uma relação todo-parte fraca
- O todo existe sem as partes
- As partes existem sem o todo
- A parte pode ser agregada por vários todos

Composição

- É uma associação com a semântica de “é composto de”
- Serve como uma relação todo-parte forte
- O todo não existe sem as partes
  - As partes pertencem a somente um todo
  - A remoção do todo implica na remoção das partes

Dependência

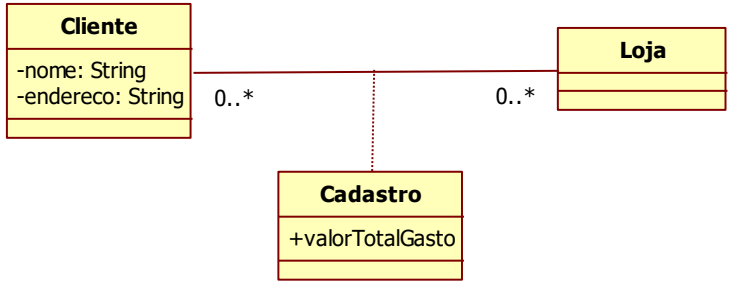
- Deixa explícito que mudanças em uma classe podem gerar consequências em outra classe
- Exemplos:
  - Uma classe chama métodos de outra
  - Uma classe tem operações que retornam outra classe
  - Uma classe tem operações que esperam como parâmetro outra classe
- Outros relacionamento (ex.: associação com navegação) implicitamente determinam dependência
- Não tente mostrar todas as dependências no seu diagrama!

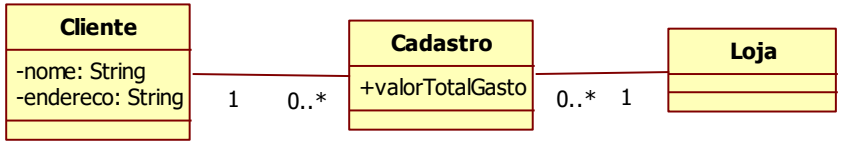
Leitura: classe A depende da classe B

Classes de associação

- Permitem a adição de informações em uma associação
- Devem ser transformadas em classes comuns posteriormente para viabilizar implementação

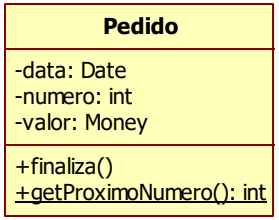
Qual o valor total gasto em cada loja?





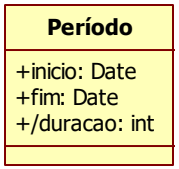
A 1 milímetro de distância... propriedades e operações estáticas

- Propriedades que não são instanciadas nos objetos
- Operações que atuam somente sobre propriedades estáticas
- Ambos são acessados diretamente na classe
  - Ex.: Pedido.getProximoNumero()
- São sublinhadas no diagrama



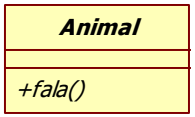
A 1 milímetro de distância... propriedades derivadas

- São propriedades que na verdade não existem como atributos ou associações
- Podem ser inferidas por outras propriedades da classe
- É interessante explicitar através de nota ou restrição a fórmula de derivação
- São marcadas com o símbolo “/”



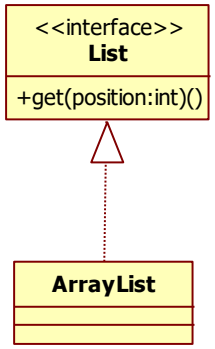
A 1 milímetro de distância... Classes e operações abstratas

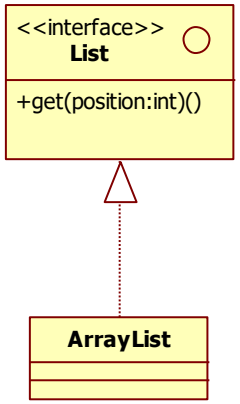
- Classes que não podem ter instâncias
  - Usualmente têm operações abstratas, ou seja, sem implementação
- Suas subclasses usualmente são concretas
  - Implementam métodos com comportamentos específicos para as operações abstratas
- Utilizam nome em itálico

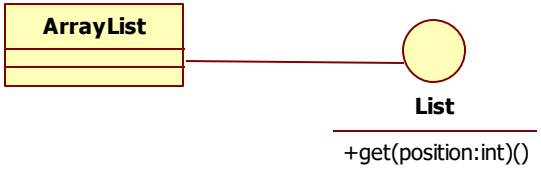


A 1 milímetro de distância... Interfaces

- Uma classe sem nenhuma implementação
  - Todas operações são abstratas
- Faz uso da palavra-chave <<interface>>
  - Pode ser representado também como um ícone
- O relacionamento de realização indicas as classes que implementam a interface
  - Equivalente a generalização



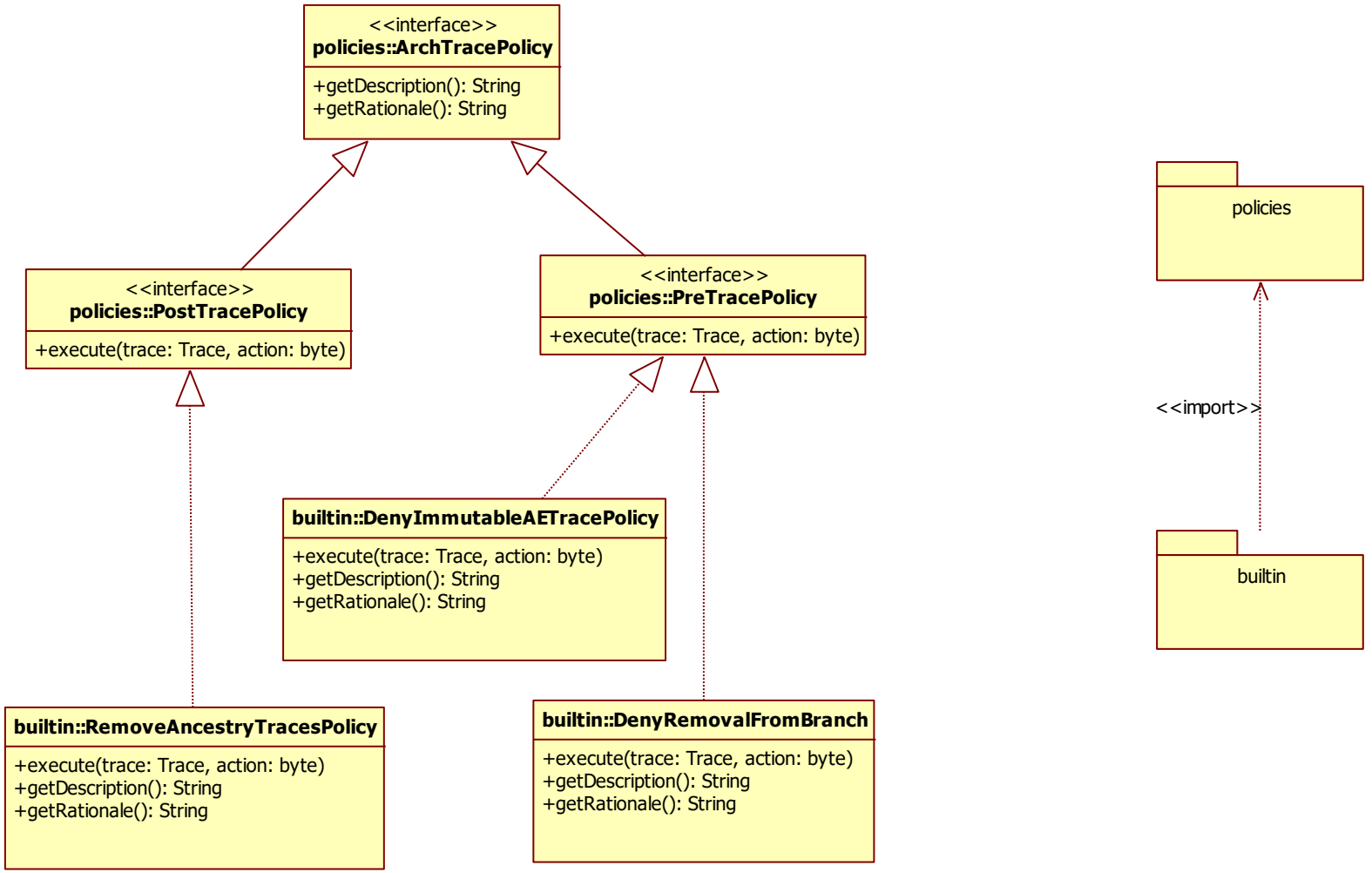




A 10 km de distância... pacotes

- Em algumas situações se deseja ter uma visão geral das partes do sistema
- Para isso, o **diagrama de pacotes** é a ferramenta indicada
- Pacotes agregam classes e outros pacotes
  - Dependências podem ser inferidas indiretamente
- Exemplo
  - Classe C1 pertence ao pacote P1
  - Classe C2 pertence ao pacote P2
  - Classe C1 depende da classe C2
  - Logo, pacote P1 depende do pacote P2

Pacotes


<<interface>>
policies::ArchTracePolicy
+getDescription(): String
+getRationale(): String
<<interface>>
policies::PostTracePolicy
+execute(trace: Trace, action: byte)
<<interface>>
policies::PreTracePolicy
+execute(trace: Trace, action: byte)
builtin::DenyImmutableAETracePolicy
+execute(trace: Trace, action: byte)
+getDescription(): String
+getRationale(): String
builtin::RemoveAncestryTracesPolicy
+execute(trace: Trace, action: byte)
+getDescription(): String
+getRationale(): String
builtin::DenyRemovalFromBranch
+execute(trace: Trace, action: byte)
+getDescription(): String
+getRationale(): String
policies
<<import>>
builtin

Dicas

- Inicie com um diagrama simples
- O que normalmente tem em todo diagrama
  - Classes
  - Atributos
  - Operações
  - Associações
- Use os demais recursos da linguagem somente quando for realmente necessário

Dicas
(possíveis candidatos)

- Classes
  - Entidades externas que produzem ou consomem informações (ex.: sistema de validação do cartão de crédito)
  - Coisas que são parte do problema e que são informações compostas (ex.: Produto)
  - Eventos que ocorrem durante a operação do sistema (ex.: Pedido)
  - Papeis que interagem com o sistema (ex.: Cliente)
  - Unidades organizacionais relevantes (ex.: Rede de lojas)
  - Lugares que fornecem o contexto do problema ou do sistema (ex.: Loja)
  - Estruturas definidas no problema (ex.: Estoque)

Dicas
(possíveis candidatos)

- Atributos
  - Informação primitiva que precisa ser memorizada (ex.: Preço)
- Associações
  - A classe A precisa se relacionar com a classe B para atender a operações específicas (ex.: Cliente – Pedido)
- Operações
  - Funcionalidades que devem ser providas por uma classe para viabilizar o uso do sistema (ex.: calculaTotal em Pedido)

Exercício

- Elabore um diagrama de classes para um sistema de ponto de vendas
  - R01. O gerente deve poder fazer login com um ID e senha para iniciar e finalizar o sistema;
  - R02. O caixa (operador) deve poder fazer login com um ID e senha para poder utilizar o sistema;
  - R03. O sistema deve registrar a venda em andamento – os itens comprados;
  - R04. O sistema deve exibir a descrição e preço e do item registrado;
  - R05. O sistema deve calcular o total da venda corrente;
  - R06. O sistema deve tratar pagamento com dinheiro – capturar a quantidade recebida e calcular o troco;
  - R07. O sistema deve tratar pagamento com cartão de crédito – capturar a informação do cartão através de um leitor de cartões ou entrada manual e autorizar o pagamento utilizando o serviço de autorização de crédito (externo) via conexão por modem;
  - R08. O sistema deve tratar pagamento com cheque – capturar o número da carteira de identidade por entrada manual e autorizar o pagamento utilizando o serviço de autorização de cheque (externo) via conexão por modem;
  - R09. O sistema deve reduzir as quantidades em estoque quando a venda é confirmada;
  - R10. O sistema deve registrar as vendas completadas;
  - R11. O sistema deve controlar diversas lojas, com catálogo de produtos e preços unificado, porém estoques separados;

Bibliografia

- Fowler, Martin. 2003. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. Addison-Wesley Professional.
- Pressman, Roger. 2004. *Software Engineering: A Practitioner's Approach*. 6th ed. McGraw-Hill.