Análise e Projeto Orientado a Objetos

Tecnologia de Software

Aula 2: Diagrama de classes

Bruno Sofiato
bruno.sofiato@usp.br
Material criado originalmente pelo Prof. Dr. Fábio Levy Siqueira

Representação

- *Código* é o principal produto do desenvolvimento de software

- É fácil conversar sobre o projeto usando o código?
  - E pensar sobre a solução?

```
public class Agencia {
  private int numero;
  private ContaCorrente contaDaAgencia;
  private HashMap<Integer, ContaCorrente> contas;
  ...
}
```

```
public class ContaCorrente {
  private double saldo = 0;
  private int numero;
  ...
}
```

```
public class Cliente {
  private ArrayList<ContaCorrente> contas;
  private String nome;
  ...
}
```

Representação

- Problemas
  - Nível de abstração: detalhes de implementação
  - Visão do todo: foco em *uma* classe
  - Visão estática
    - E o funcionamento em um cenário específico?
  - Dificuldade de entendimento
    - Complexidade e conhecimento da linguagem

- ...um software OO de médio / grande porte terá *muitas* classes...

- Como lidar com a dificuldade de comunicação?

Modelo

- Modelos permitem simplificar a comunicação e facilitam pensar sobre a solução

- O que é um modelo?
  - Representação simplificada da realidade
  - Estrutura do modelo se assemelha com a do que é modelado
  - Algumas operações aplicáveis no que é modelado também são aplicáveis ao modelo

- Permite representar diversas visões de um mesmo sistema

Modelo

- Problemas
  - Diferença entre o modelo e o código
  - Manutenção (do modelo)
  - É uma abstração
  - Dificuldade de juntar diversas visões
  - Linguagem específica (apesar de mais simples)

- Devemos criar modelos? Sem sim, quais?

Depende do contexto e do processo!

Modelo

- Usos de modelos na Engenharia de Software
  - Especificar um sistema
    - Definir restrições para a construção do sistema
      - Permite compreender e manipular melhor o sistema a ser construído
  - Descrever um sistema
    - Analisar o sistema existente
      - Permite analisá-lo em um determinado nível de abstração ou ponto de vista

Modelo

- Formas de uso (Fowler, 2003)
  - Rascunho
    - Comunicação (discussão) entre os desenvolvedores
      - (Pode ser jogado fora após a conversa)
    - Seletivo: só alguns aspectos relevantes
  - Planta (*blueprint*)
    - Projeto detalhado (completo)
    - Aspectos relevantes para o programador
  - Linguagem de programação
    - Descrição do sistema completamente
    - Modelo como código fonte ("compilado")
    - MDA – Model Driven Architecture

UML

UML

- *Unified Modeling Language*
  - *Método Booch*
  - *OMT (Rumbaugh)*
  - *OOSE (Jacobson)*
- Linguagem para elaborar a estrutura de sistemas
  - Modelagem gráfica
  - *Principalmente* softwares Ori
- Mantida pela OMG
  - Padrão
  - http://www.uml.org
- Versão 2.5.1 (2017)

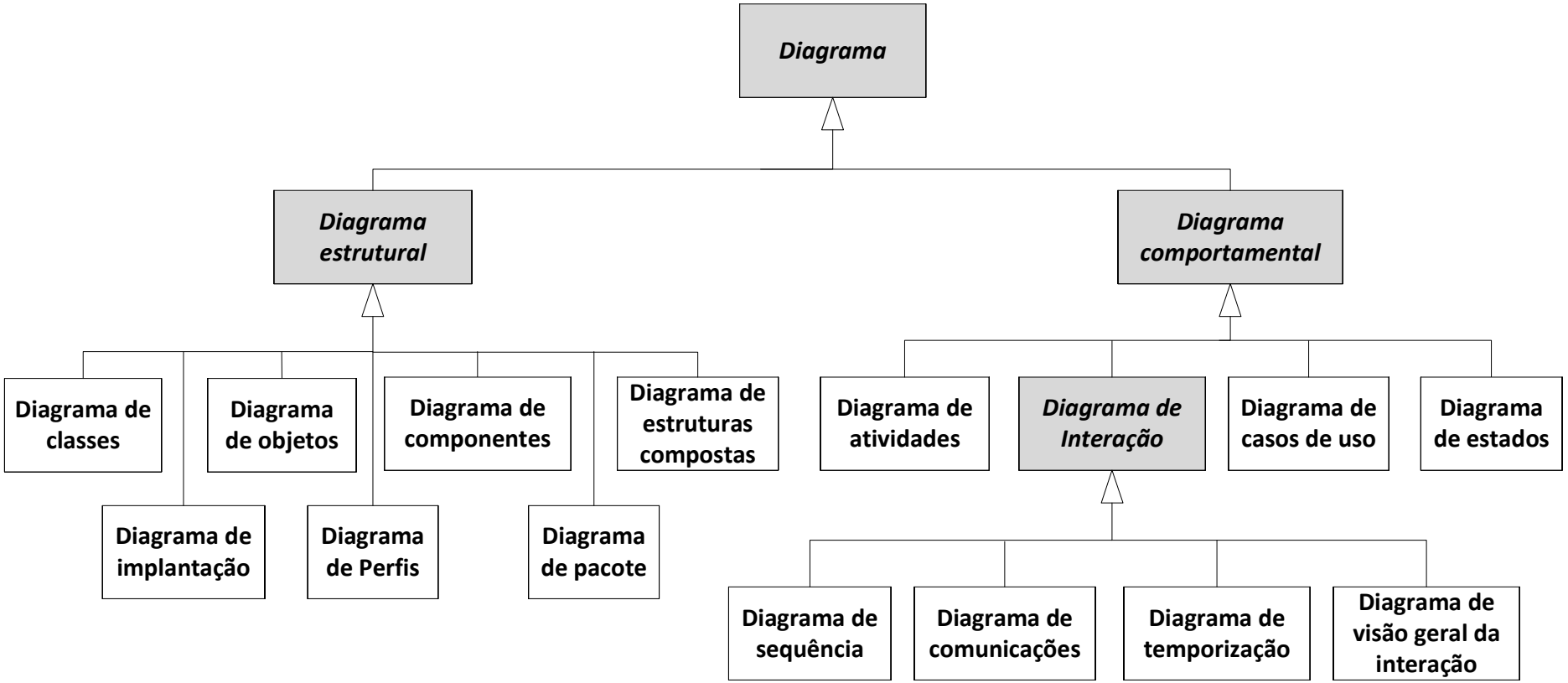
UML

- Define um padrão para **comunicação**

- Permite representar diferentes visões de um mesmo sistema
  - Diversos diagramas
  - *Exemplo*
    - Requisitos: diagrama de casos de uso
    - Classes: diagrama de classes
    - Execução de um cenário: diagramas de interação
    - Organização das classes: diagrama de pacotes

Diagramas da UML

- Apresentação gráfica de um conjunto de elementos
- Representa uma visão de um modelo



Uso da UML

- UML é uma linguagem
  - **UML não é um processo / método!**
  - A UML tem um "modelo conceitual" (*metamodelo*)
    - Sintaxe e semântica
    - Elementos comuns a vários diagramas

- Cada processo define **se** serão produzidos e como serão usados os diagramas da UML
  - *Exemplo*: ICONIX e eXtreme Programming
  - Nada impede o uso de diagramas UML como rascunho

O modelo deve ser o **suficiente** para facilitar a implementação.

Diagrama de classes da UML

Diagrama de classes

- Diagrama UML mais usado
- Visão estática do sistema
  - Conceitos do problema e da solução

- Representa as classes, relacionamentos e outros elementos
  - Segue o paradigma Orientado a Objetos...
    - (Por mais que a UML possa ser usada para modelar sistemas não necessariamente OO)

Ferramentas

- Gratuitas
  - Modelio: https://www.modelio.org/
  - Papyrus (Eclipse): https://eclipse.org/papyrus/
  - PlantUML (Visual Studio Code): https://github.com/qjebbs/vscode-plantuml
- Pagas com versão para a "comunidade"
  - Visual Paradigm: http://www.visual-paradigm.com/
  - StarUML: http://staruml.io/
- Pagas
  - RSAD: https://www.ibm.com/us-en/marketplace/rational-software-architect-designer
  - Enterprise Architect: http://www.sparxsystems.com.au



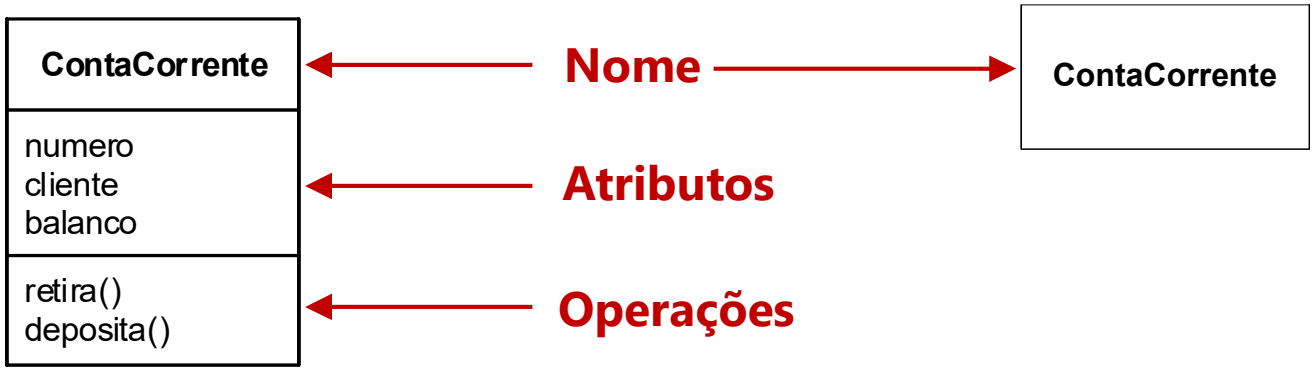




Uma lista mais completa:
http://en.wikipedia.org/wiki/List_of_UML_tools

Classes

- Descrição de um conjunto de objetos com características comuns
  - (Pode ter outros *compartimentos*)



- **Nome**: substantivo e singular
  - Representa um conceito

Atributos

- Abstração dos estados: propriedade
- Intervalo de valores que uma propriedade pode apresentar
  - Nome
  - Tipo (*opcional*)
  - Valor padrão (*opcional*)

Cliente
nome cpf endereço telefone dataDeNascimento estaBloqueado

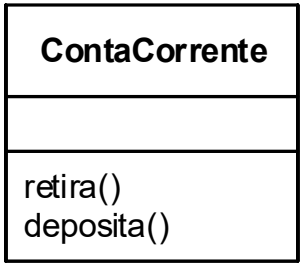
Cliente
nome: String cpf: String endereço: Endereco telefone: String dataDeNascimento: Date estaBloqueado: boolean = false

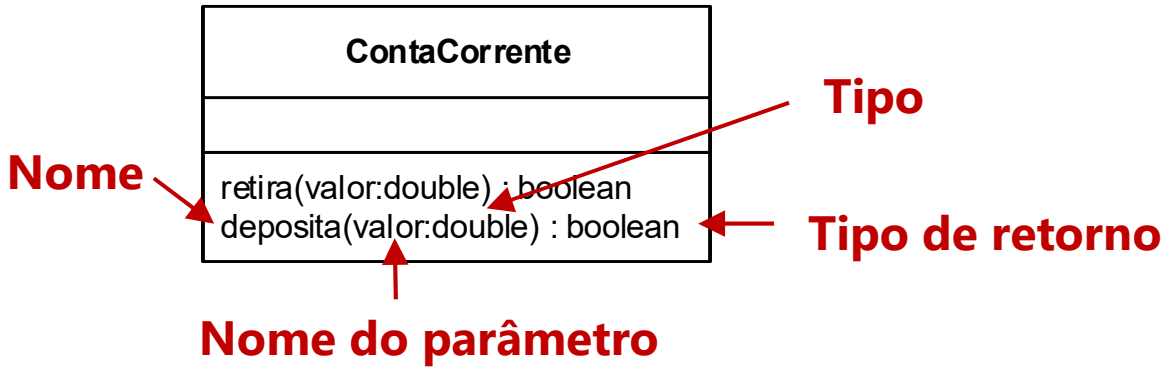
Atributos

- Nome de um atributo
  - Substantivo
    - *Exemplo*: nome, CEP, idade, quantidade
  - Verbos representado estado
    - *Exemplo*: estáAtivo, executando (um processamento), cancelado

Operações

- Implementação de um serviço
  - Pode mudar o estado do objeto
- Assinatura
  - Nome da operação
  - Parâmetros (*opcional*)
    - Nome, Tipo (*opcional*) e Valor padrão (*opcional*)
  - Tipo do valor de resposta (*opcional*)





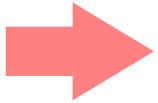
Operações

- Nome de uma operação
  - Verbo (no imperativo afirmativo ou no infinitivo)
    - Representa uma ação que pode ser feita com o objeto
    - *Exemplo*
      - Retira, adiciona, aluga, embaralha
      - Reitrar, adicionar, alugar, embaralhar

Linguagem de programação

- O mapeamento é direto (menos o em vermelho!)

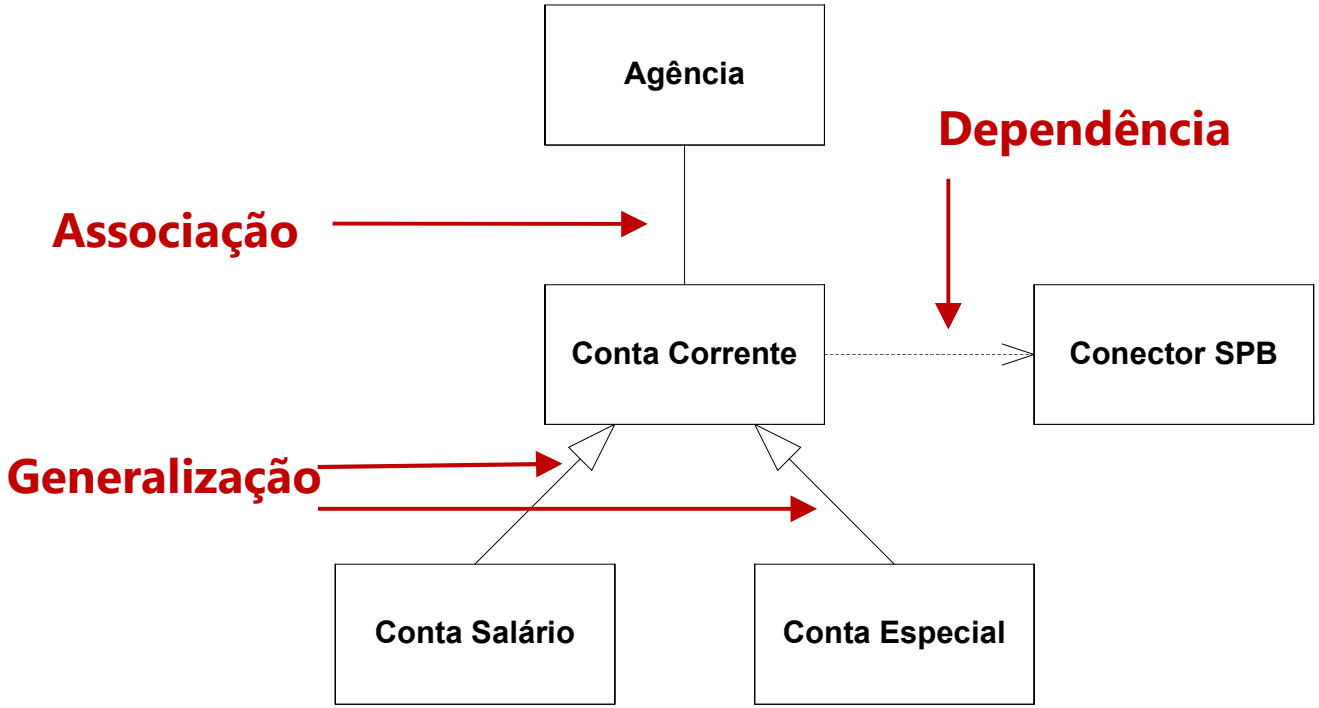
ContaCorrente
saldo: double numero: int
retira(valor: double): boolean deposita(valor: double) imprimeExtrato()



```
public class ContaCorrente {
   double saldo = 0;
   int numero;

   public boolean retira(double valor) {
      if (valor > saldo) return false;
      saldo -= valor;
      return true;
   }

   public void deposita(double valor) {
      saldo += valor;
   }

   public void imprimeExtrato() {
      System.out.println("Conta " + numero);
      System.out.println(saldo);
   }
}
```

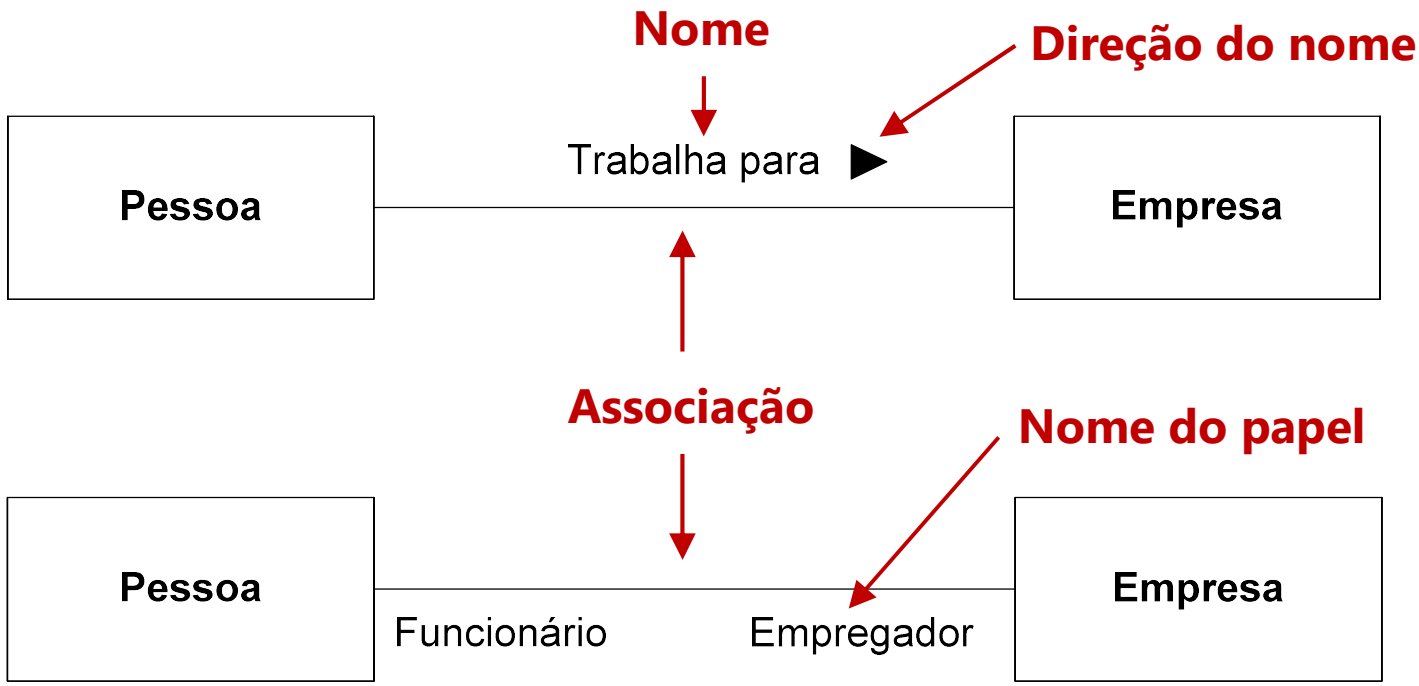
Relacionamento

- Forma de representar a colaboração entre classes
  - Associação
  - Dependência
  - Generalização



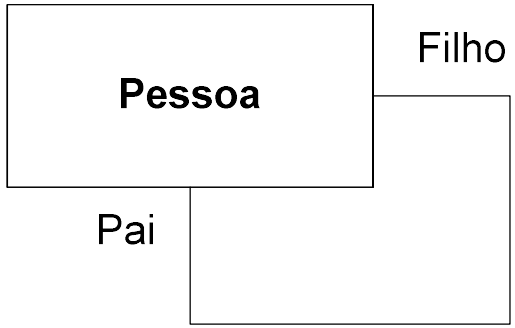
Associação

- Objetos de uma classe estão conectados a objetos de outra classe
  - Representam requisitos de informação

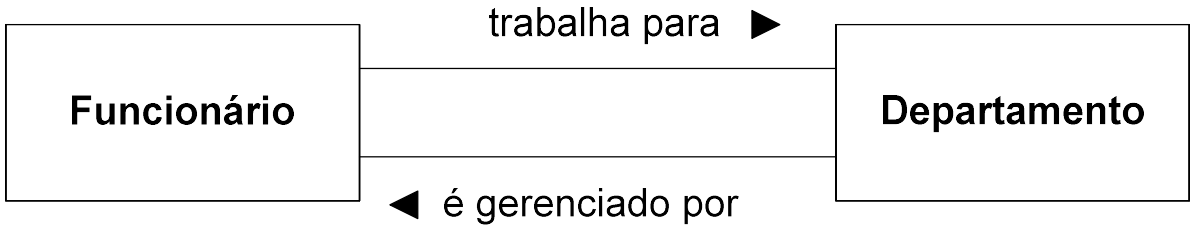


Associação

- Uma classe pode se associar a ela mesma



- Uma classe pode ter mais de uma associação com uma outra classe



Associação

- Multiplicidade
  - Número de **objetos** que podem estar envolvidos na relação em um determinado ponto do tempo
    - É uma **faixa de valores**

  - Não há uma multiplicidade padrão

Associação

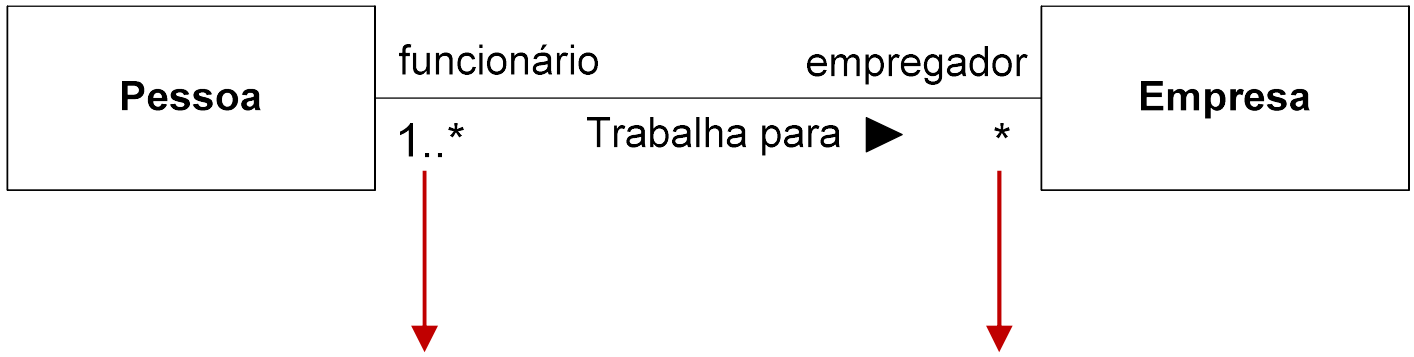
- Multiplicidade: valores

Valor	Representação	Exemplo
Valor fixo	número	1 (apenas 1 objeto) 3 (obrigatoriamente 3 objetos)
Faixa de valor	mínimo..máximo	1..5 (de 1 a 5 objetos) 1..* (1 ou mais objetos) 0..* (0 ou mais objetos)

  - Observação: `*` e `0..*` são equivalentes

Associação

- O que significa uma associação?
  - *Exemplo*



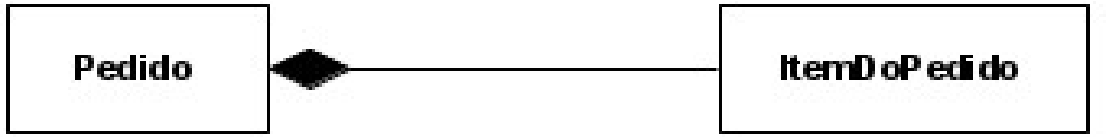

```
class Empresa {
  Pessoa[] funcionários;
  // ...
}
```

```
class Pessoa {
  Collection<Empresa> empregadores;
  // ...
}
```

(essa associação é bidirecional)

Composição

- Associação onde o filho **não existe** independentemente do pai (relação têm um)
  - *Exemplo*



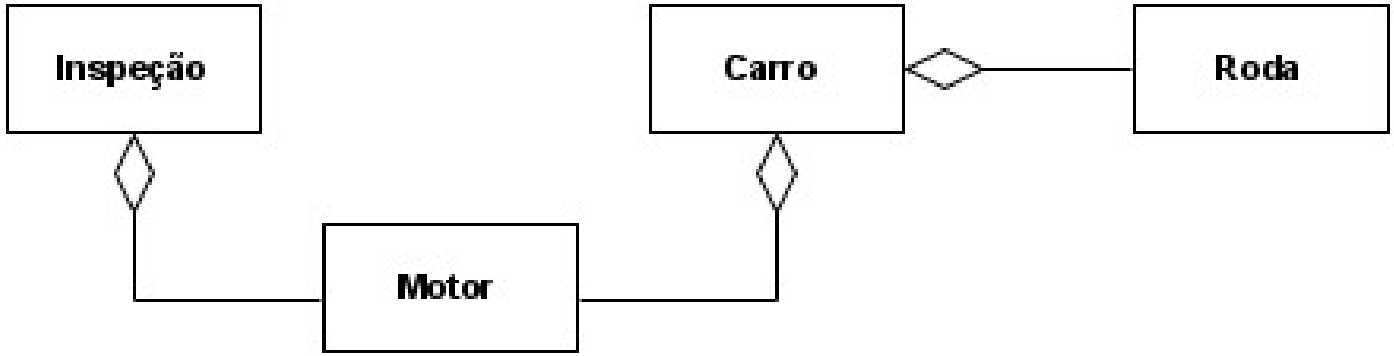

```
class Pedido {
  Collection<ItemDoPedido> items;
  // ...
}
```

```
class ItemDoPedido {
  Pedido pedido;
  // ...
}
```

(se o pedido for deletado, os itens também o são).

Agregação

- Associação onde **não existe** uma relação de posse entre os elementos
- *Exemplo*

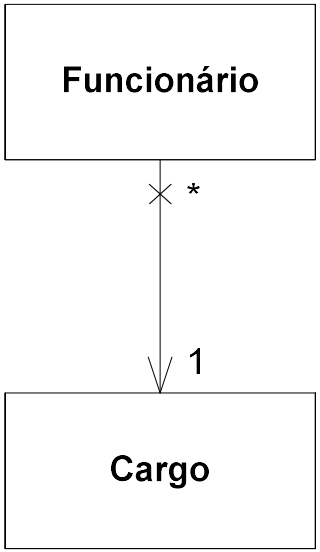


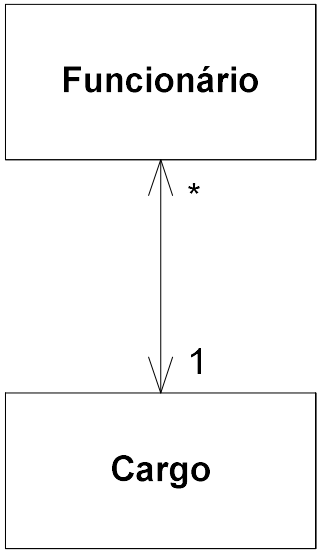
(Fonte: Visual Paradigm)

Navegação

- Representa a possibilidade de ir de um objeto à outro (acesso)
  - *Na prática*: a classe tem ou não um atributo relativo à associação
- É possível especificar se bidirecional ou não

O funcionário consegue observar o seu cargo, mas o cargo não consegue saber quais são os funcionários

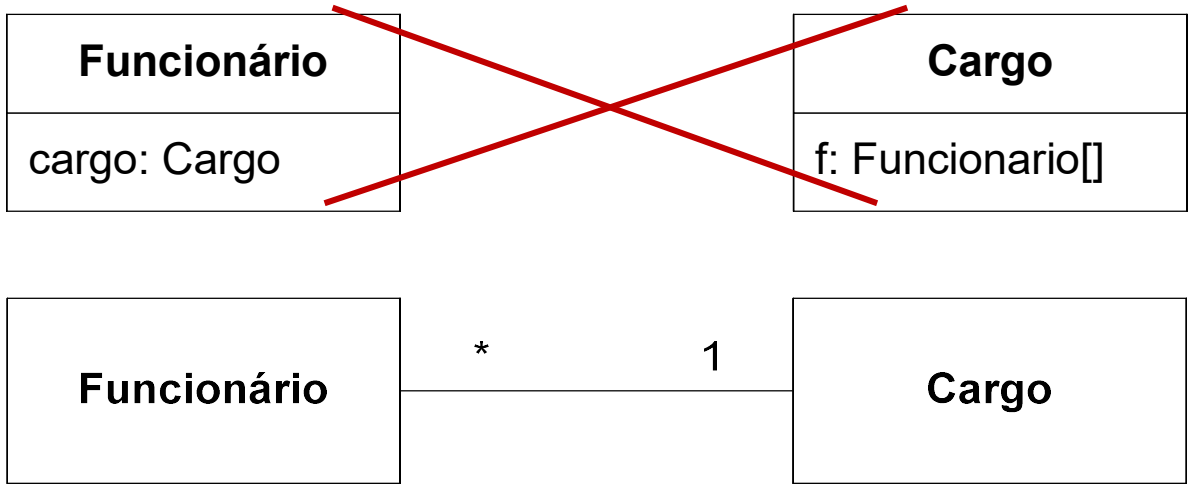




O funcionário consegue observar o seu cargo e vice-versa

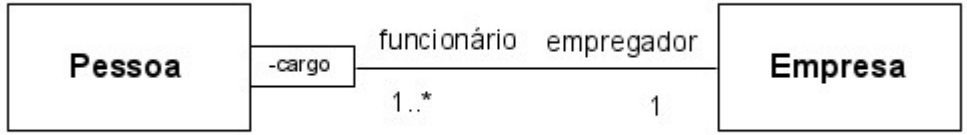
Navegação

- Em modelos *mais abstratos* é comum pensar na navegação como **bidirecional**
  - Em modelos mais concretos é preciso **analisar**
    - A navegação gera acoplamento
    - Navegação precisa ser *gerenciada*
- Represente **associações** ao invés de atributos



Associação Qualificada

- Associações qualificada é usada quando uma associação têm uma chave de qualificação
- *Exemplo*



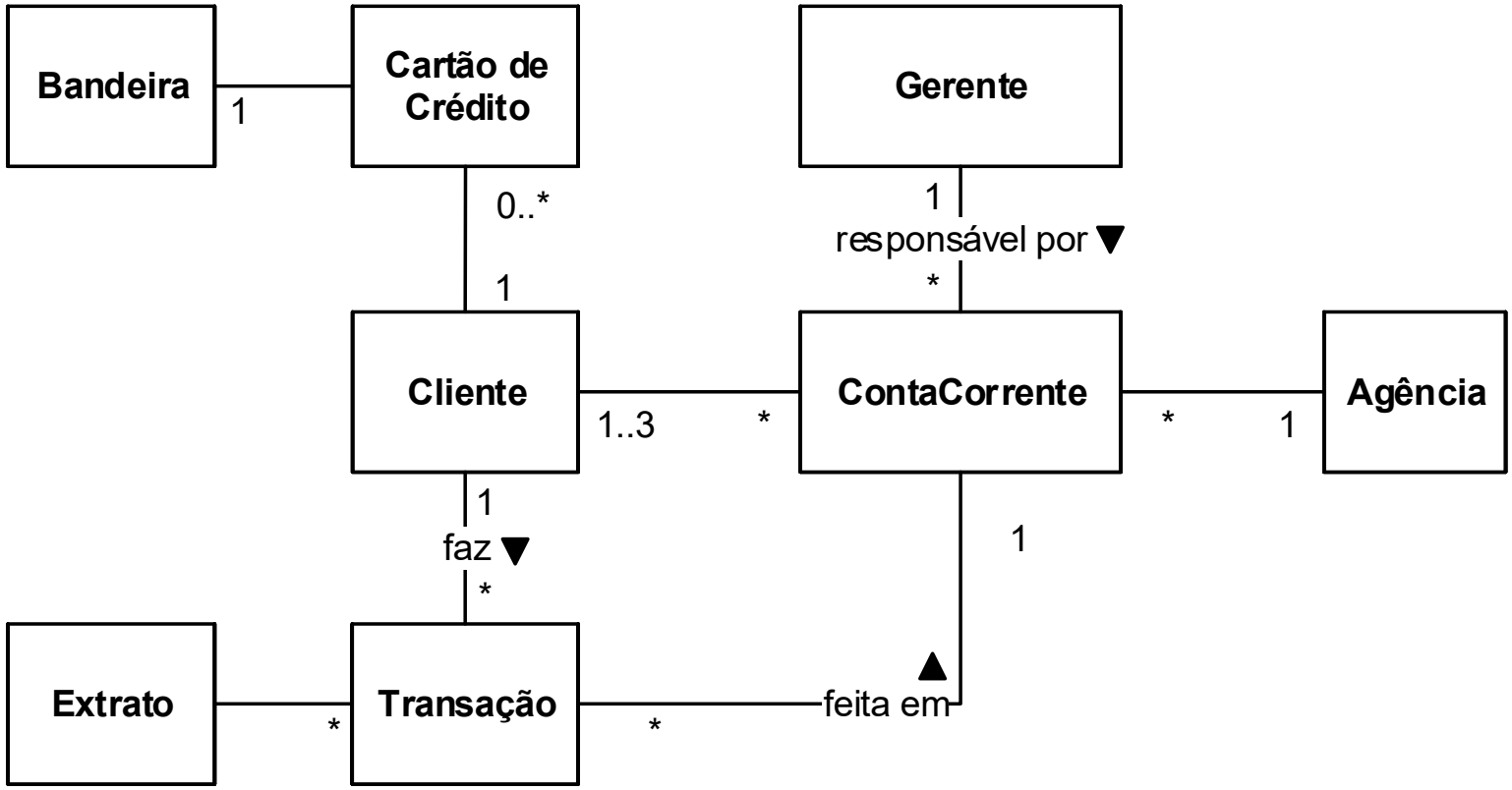

```
class Empresa {
  Map<String, Pessoa> funcionários;
  // ...
}
```

```
class Pessoa {
  Empresa empregador;
  // ...
}
```

(a multiplicidade é será um nesse caso !!!)

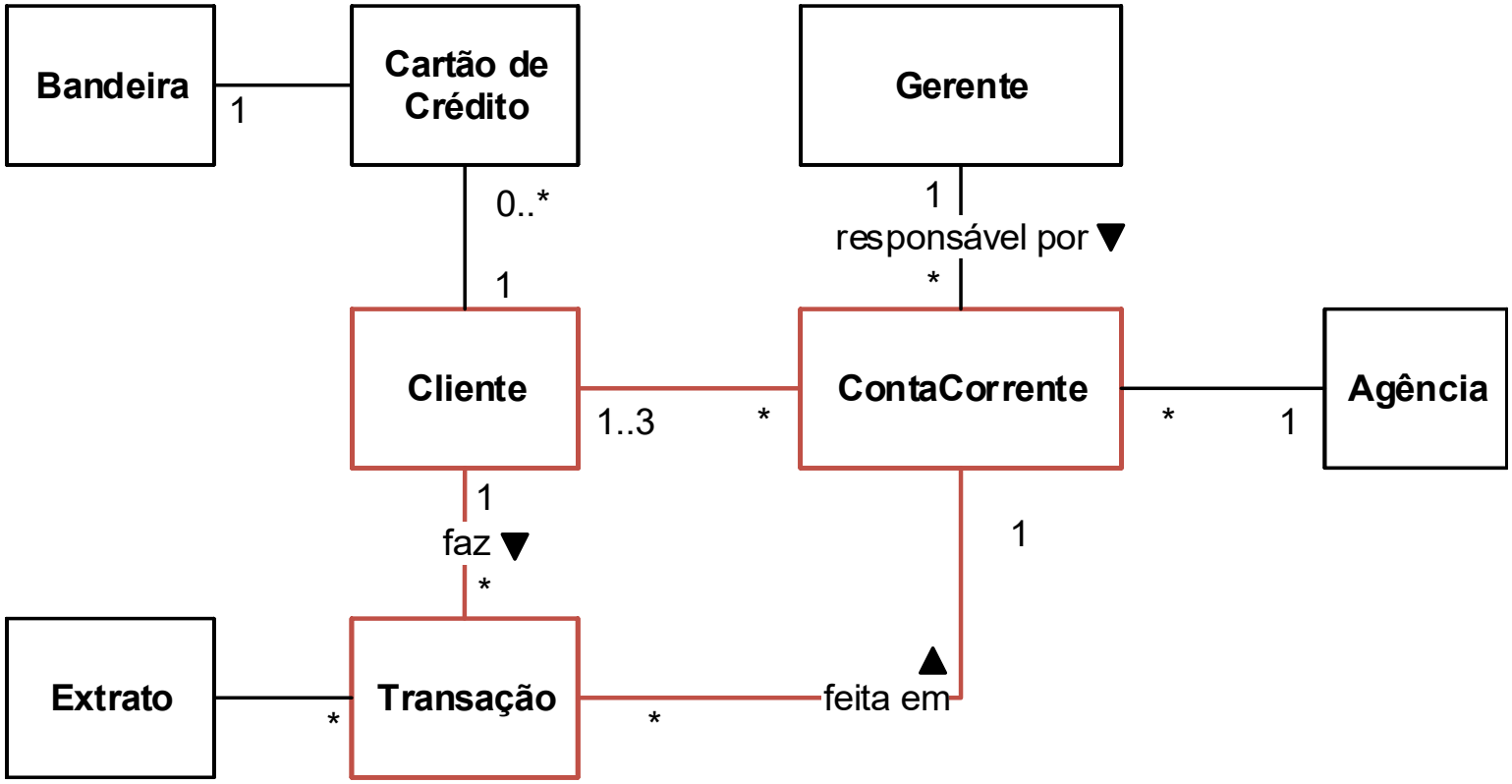
Exemplo

- Classes de um sistema bancário
  - Defina multiplicidade e nomes (sempre que possível)



Exemplo

- Como representar que o cliente só faz transações em uma conta corrente dele?



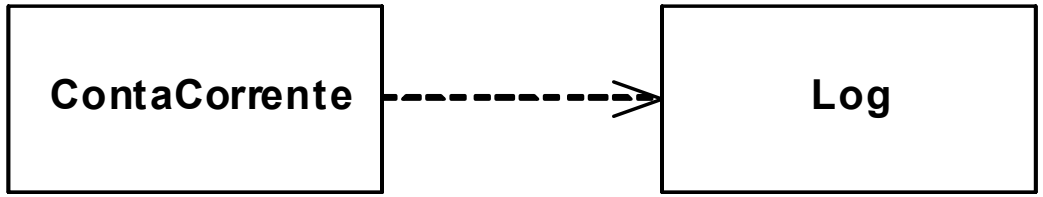
Problemas

- A multiplicidade não consegue representar <u>todas</u> as restrições entre os objetos...
  - Em geral outras restrições são representadas como comentários em **linguagem natural**
  - Para especificar *formalmente* usa-se a OCL
    - *Object Constraint Language* – mantida pela OMG
    - *Exemplo*

```
context Transacao

inv TransacaoEhDeUmaContaDoCliente:
  self.cliente.contas->includes(self.conta)
```

Dependência

- Relacionamento de uso de informações e serviços



A ContaCorrente depende do log

  - A mudança na *interface* do fornecedor afeta o cliente

- Use quando há uma relação mas não é associação*
  - *Exemplo*
    - Objeto usado como parâmetro para uma operação
    - Chamada de operação em um objeto de outra classe

por enquanto

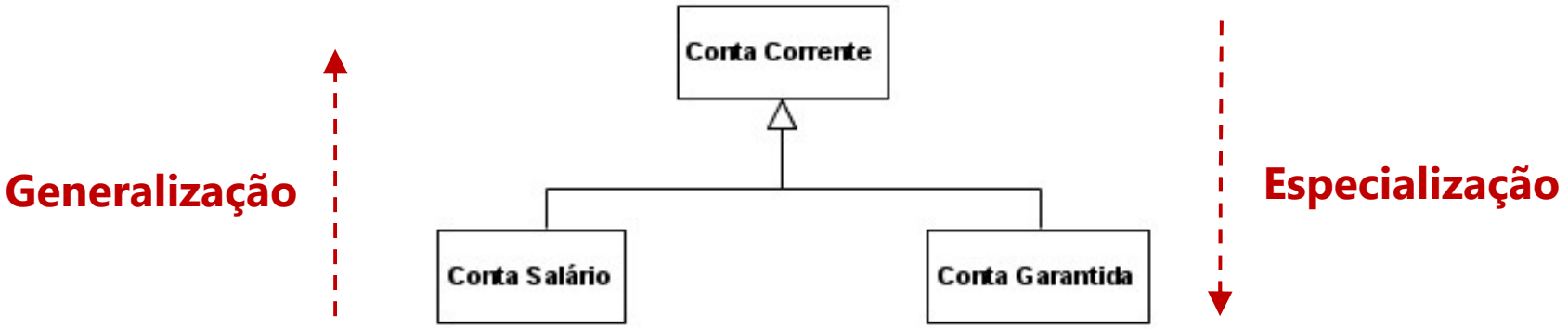
Dependência

- Existem diferentes tipos de dependência, mas na prática eles são pouco usados
  - *Use*, *Call*, *Instantiate* etc.

- Use apenas quando acrescentar algo ao modelo
  - Atrapalha legibilidade

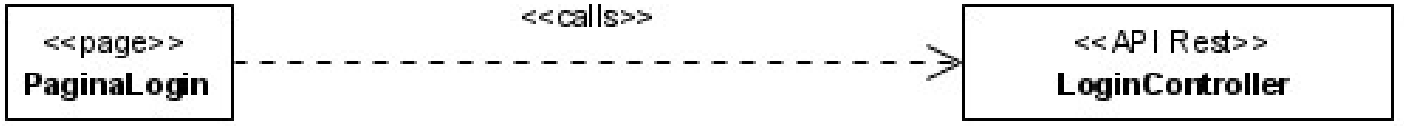
Generalização

- Combinação de classes similares de objetos em uma classe mais genéricas.
- Implementação depende do contexto (usualmente é herança)



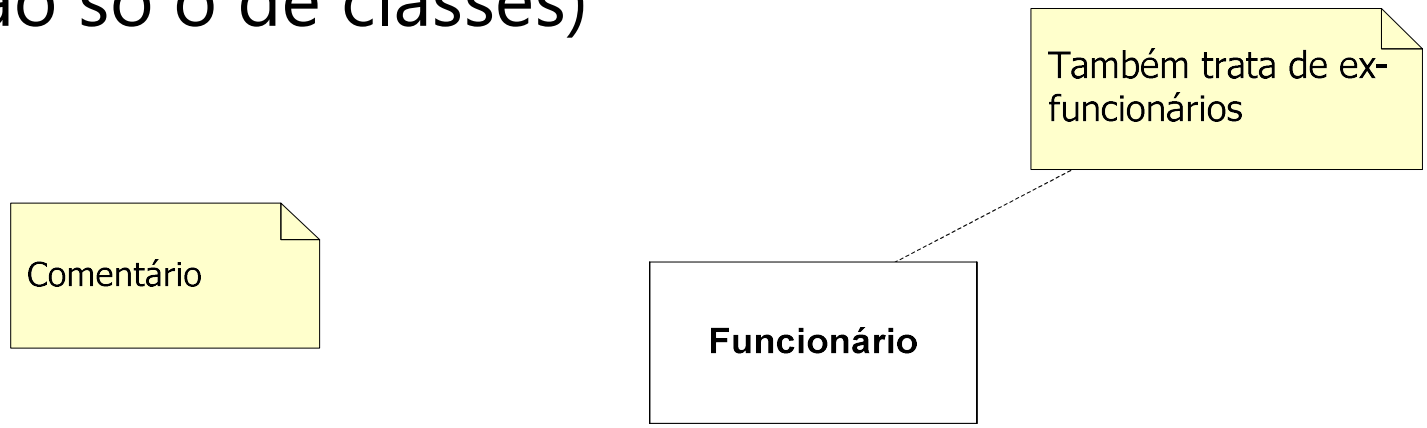
Estereótipos

- Permite estender o vocabulário da linguagem UML de modo a criar novos elementos.
- *Exemplo*



Comentários

- É possível criar comentários nos diagramas UML (não só o de classes)

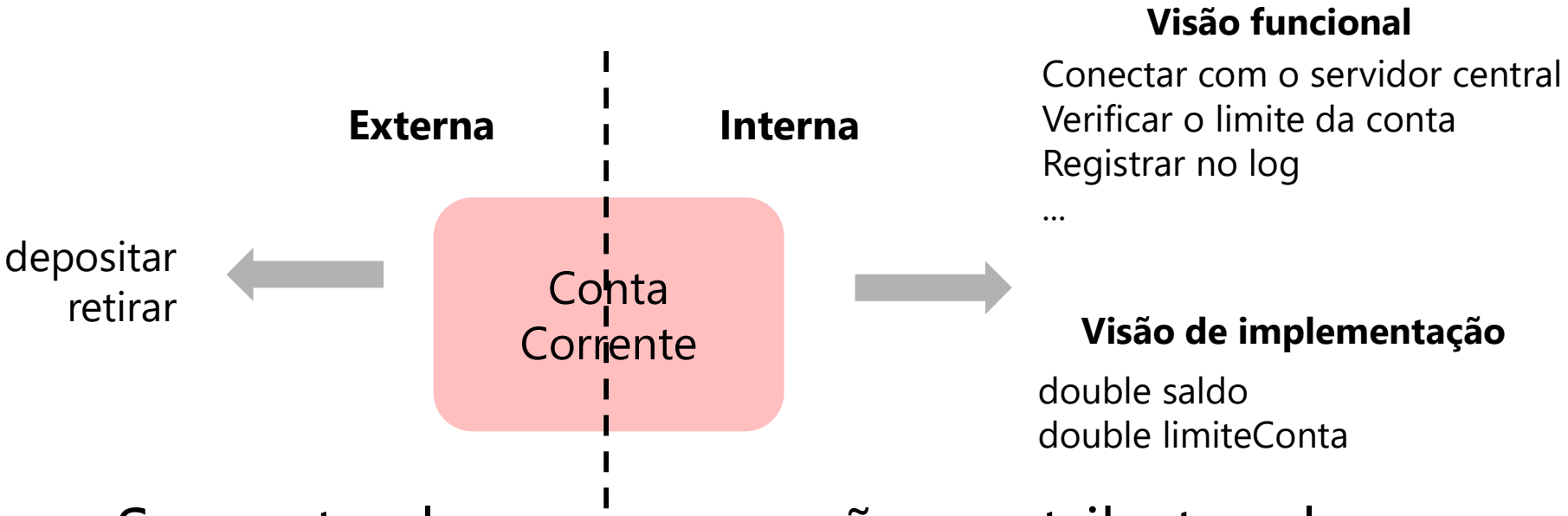


- Além de textos, eles também podem expressar restrições em OCL

Encapsulamento

Encapsulamento

- Separação da visão interna de uma classe da visão externa



- Somente algumas operações e atributos devem ser vistos externamente

Encapsulamento

- Uma classe não deve depender dos detalhes internos de outras classes
  - Acoplamento

- Como aplicar o encapsulamento?
  - Visibilidade
  - Pacotes
  - Interfaces (Aula 5)

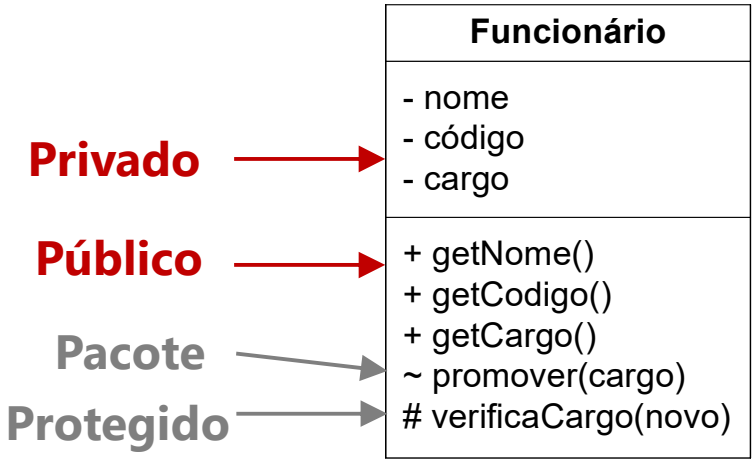
Visibilidade

- Forma de *restrição de acesso*
  - Evita o acesso aos elementos internos da classe
  - Classes, operações e atributos
- Modos de visibilidade na UML

Modo de Visibilidade	Explicação	Representação
Público	Todas as classes acessam	+
Protegido	Apenas as classes filhas acessam	#
Pacote	Apenas as classes no mesmo pacote acessam	~
Privado	Somente a classe acessa	-

Visibilidade

- Exemplo




```java
public class Funcionario {
  private String nome;
  private int codigo;
  private Cargo cargo;

  public String getNome() {...}
  public int getCodigo() {...}
  public Cargo getCargo() {...}
  boolean promover(Cargo novo) {...}
  protected boolean verificaCargo(Cargo novo) {...}
}
```

Erro de compilação!

```java
public class Main {
  public static void main(String[] args) {
    Funcionario f = new Funcionario();
    f.nome = "Fabio L. S.";
  }
}
```

Visibilidade

- Normalmente os atributos são privados ou *protegidos*
  - Restringir acesso (*ex.*: somente leitura)
  - Aplicar lógica de negócio (*ex.*: valor maior que 0)
  - Esconder representação usada

Modos de visibilidade

- Os modos de visibilidade e a semântica deles depende da linguagem de programação
  - *Exemplo*
    - **Java**: O modo protegido também é acessível para classes do mesmo pacote
    - **C#**: O modo "protected internal" não tem representação
    - **Python**: não há modo de visibilidade
      - Membros com “_” ou “__” ainda são acessíveis
- Na prática use as regras de visibilidade da linguagem de programação / convenções

Uso do diagrama de classes

Diagrama de classes

- Ajuda na comunicação
  - Entre desenvolvedores
  - Desenvolvedores e o cliente (*Domain Driven Design*)

- O detalhamento da classe varia de acordo com o público alvo e o uso desejado
  - É possível omitir atributos, operações, relacionamentos, compartimentos etc. se necessário
  - Necessidade de apresentar uma relação depende do objetivo do diagrama!

Uso do diagrama de classes

- Diagrama de domínio
  - Entendimento do contexto do sistema
  - Linguagem comum
- Diagrama de análise
  - Conceitos do domínio do problema
- Diagrama de projeto
  - Detalhes de implementação
  - Considerações arquiteturais e de tecnologia
- (Diagrama de implementação)
  - Resultado da implementação
  - Engenharia reversa

Outras referências

- ARLOW, J.; NEUSTADT, I. **UML 2 and The Unified Process: Practical Object-Oriented Analysis and Design**. 2a edição, Addison-Wesley, 2005.

- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Software Development Process**. Addison-Wesley, 1999.

- BOOCH, G. et al. **Object-Oriented Analysis and Design with Applications**. 3ª edição, Addison-Wesley, 2007.