Relacionamento entre Classes

Roland Teodorowitsch

Programação Orientada a Objetos - ECo - Curso de Engenharia de Computação - PUCRS

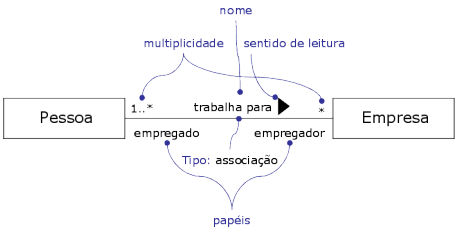
13 de setembro de 2023

Relacionamentos entre Classes

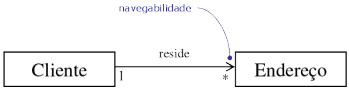
Relacionamentos entre Classes

- O que são?
  - Os relacionamentos descrevem como as classes interagem umas com as outras
  - Os relacionamentos entre as classes também definem responsabilidades
- Os relacionamentos possuem:
  - **Nome**: descrição dada ao relacionamento (faz, tem, possui, etc.)
  - **Sentido de leitura**
  - **Navegabilidade**: indicada por uma seta no fim do relacionamento
  - **Multiplicidade**: 0..1, 0..*, 1, 1..*, 2, 3..7
  - **Tipo**: dependência, associação (agregação, composição) e generalização
  - **Papéis**: desempenhados por classes em um relacionamento

Relacionamentos entre Classes



- O cliente sabe quais são seus endereços, mas o endereço não sabe a quais cliente pertence:



- Os relacionamentos possíveis entre as classes são: Dependência, Associação, Agregação, Composição e Herança (Generalização/Especialização)

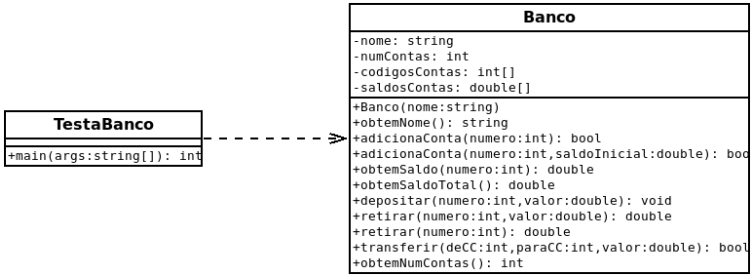
Dependência

Dependência

- Dependência é o relacionamento mais simples entre classes/objetos
- A dependência indica que um objeto depende da especificação de outro objeto
  - Por especificação, podemos entender a interface pública do objeto (seu conjunto de métodos públicos)
- Em um relacionamento de dependência, um objeto é dependente da especificação de outro objeto
- Se a especificação mudar, você precisará atualizar o objeto dependente

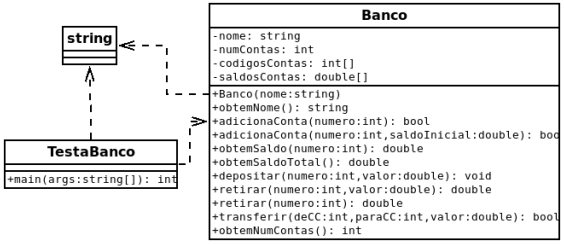
Dependência

- A classe A **depende** da classe B quando:
  - A classe A requer a classe B na sua **especificação** ou **implementação**
- Em UML, a dependência é representada por ---------->
- Por exemplo:



Dependência

- Quando se deve modelar as dependências?
  - Para ressaltar grupos de classes relacionadas
  - Geralmente entre classes importantes
  - Evitar representação de dependência óbvias
- Normalmente, se modela dependências quando quer mostrar que um objeto usa outro
  - Um lugar comum onde um objeto usa outro é através de um argumento de método ou construtor
  - Por exemplo, o construtor de um objeto da classe Banco recebe um objeto da classe `string` como argumento.



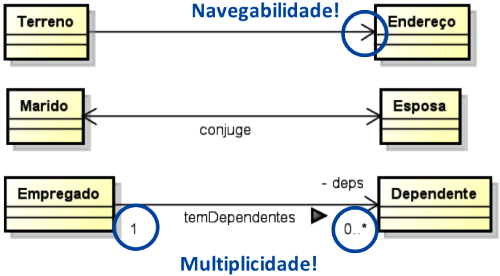
Associação

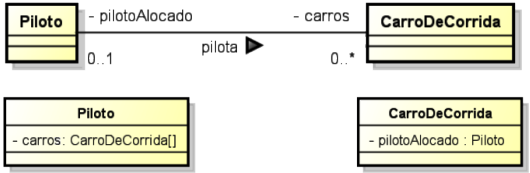
Associação

- Os relacionamentos de associação vão um pouco mais fundo do que os relacionamentos de dependência
- As associações representam relação entre objetos
- As associações são relacionamentos estruturais
- Uma associação indica que um objeto **contém (ou que está conectado a)** outro objeto

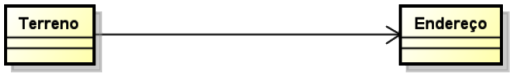
Associação

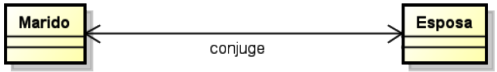
- A classe A **tem associação X** com a classe B
  - Objetos da classe A tem relacionamento X com objetos da classe B
- Em UML, a associação é representada por ⟶
- Por exemplo:





Relações 1:1

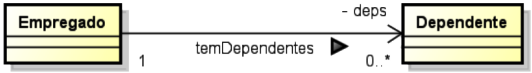





```
class Terreno {
private:
  Endereco endereco;
  ...
};



class Endereco {
  ...
};
```

```
class Marido {
private:
  Esposa *conjuge;
  ...
};

class Esposa {
private:
  Marido *conjuge;
  ...
```

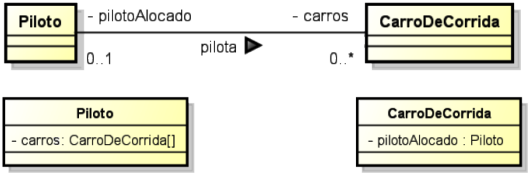
Relações 1:n




```
class Empregado {
private:
  Dependente deps[N];
  ...
};

class Dependente {
  ...
};
```

Associação

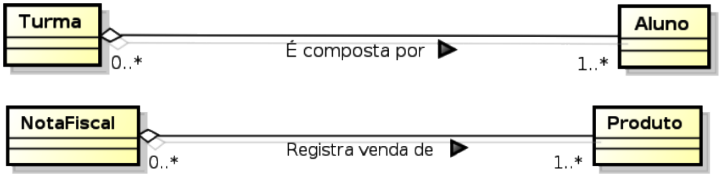
- Quando você deve modelar associações?
  - Você deve modelar associações quando um objeto contiver outro objeto (o relacionamento "tem um")
  - Você também pode modelar uma associação quando um objeto usa outro
  - Uma associação permite que você modele quem faz o que em um relacionamento
- Para refinar ainda mais os modelos, UML define dois tipos de associação: agregação e composição



Agregação

Agregação

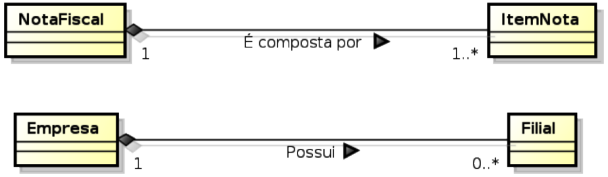
- Uma agregação é um tipo especial de associação
- Uma agregação modela um relacionamento **tem um** (ou **parte de**) entre pares
- Esse relacionamento significa que **um objeto NÃO é mais importante do que o outro**
- Importância, no contexto de agregação, significa que os objetos podem existir independentemente uns dos outros
- Isto é, se o objeto deixa de existir, os outros objetos que fazem parte do modelo podem continuar existindo



Composição

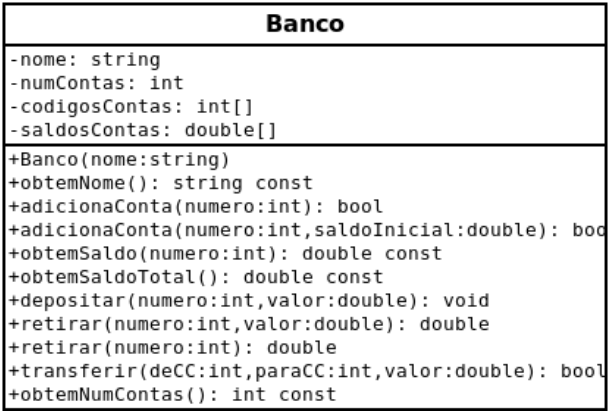
Composição

- A composição é um pouco mais rigorosa do que a agregação
- A composição não é um relacionamento entre pares. Os objetos não são independentes uns dos outros
- Isso significa, em termos de programação, que quando o objeto todo é destruído, todos os objetos parte são automaticamente destruídos também

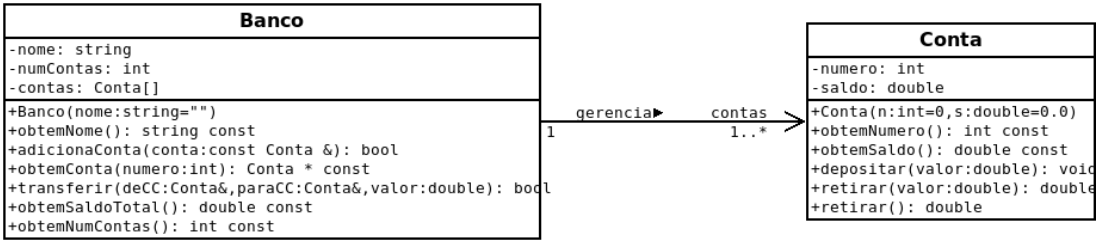


Aplicando Relacionamento entre Classes na Classe Banco

Classe Banco



Classe Banco com Classe Conta



Classe Banco em C++

```
class Banco {
private:
  string nome;
  Conta contas[N];
  int numContas;
public:

  Banco(string nome){
    this->nome = nome;
    numContas = 0;
  }

  ...

  bool adicionaConta(const Conta &c){
    if (numContas < N){
        contas[numContas] = c;
        numContas++;
        return true;
    }
    return false;
  }

  ...
};
```

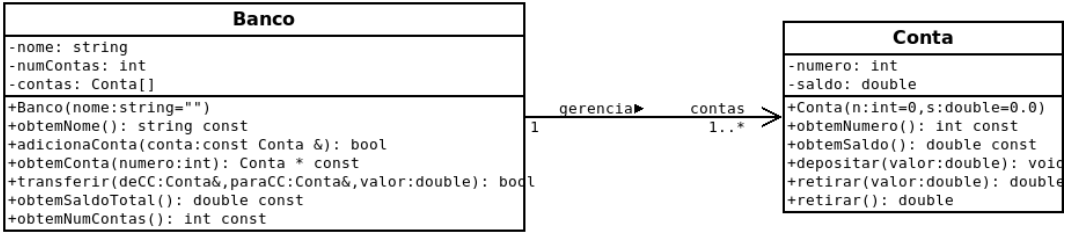
Associações e Dependências

- Associações:
  - Normalmente mapeiam para atributos
- Atributo x Associação:
  - Convenção:
    - Atributos para classes são associações
    - Atributos para tipos primitivos não
  - Quando usar no caso de classes:
    - Modelo compacto: atributos
    - Ressaltar relacionamento: associação
- Dependência x Associação:
  - Associação comumente implica em dependência

Exercício

Exercício

1. Reimplemente Banco segundo o diagrama abaixo:



2. Faça um programa principal para testar os métodos das classes.

Créditos

Créditos

- Estas lâminas contêm trechos de materiais disponibilizados pelos professores Rafael Garibotti, Daniel Callegari, Sandro Fiorini e Bernardo Copstein.

Soluções

Exercício 1: Conta.hpp

```
#ifndef _CONTA_HPP
#define _CONTA_HPP

using namespace std;

class Conta {
private:
  int numero;
  double saldo;
public:
  Conta(int n=0, double s=0.0);
  ~Conta();
  int obtemNumero() const;
  double obtemSaldo() const;
  void depositar(double valor);
  double retirar(double valor);
  double retirar();
};
#endif
```

Exercício 1: Conta.cpp

```cpp
#ifdef DEBUG
#include <iostream>
#endif
#include "Conta.hpp"

Conta::Conta(int n, double s){
  numero = n;
  saldo = s;
#ifdef DEBUG
  cout << "+ Conta " << numero << " criada..." << endl;
#endif
}

Conta::~Conta(){
#ifdef DEBUG
  cout << "- Conta " << numero << " destruida..." << endl;
#endif
}

int Conta::obtemNumero() const { return numero; }
double Conta::obtemSaldo() const { return saldo; }
void Conta::depositar(double valor){ saldo += valor; }

double Conta::retirar(double valor){
  double res;
  if (valor > saldo) {  res = saldo;  saldo = 0.0;  }
  else               {  res = valor;  saldo -= valor;  }
  return res;
}

double Conta::retirar() {
  double res = saldo;
  saldo = 0.0;
  return res;
}
```