Uma introdução à Orientação por Objetos

Abstração de Dados + Abstração de Operações

Prof. Lúcio Mauro Pereira

18/11/2024

Objetos

Elementos do mundo real

Identificação, descrição e representação controladas por diferentes graus de abstração

Abstração de Dados + Abstração de Operações

Classes de Objetos

Uma classe descreve um objeto por ambos:

Dados (atributos)

Operações (funcionalidades)

Ciclo:

Objeto (real)	Classe	Objeto (sistema)
Abstração	Modelagem	Instância

Princípios da Orientação por Objetos (OO)

1. Abstração
2. Encapsulamento
3. Generalização (herança)
4. Polimorfismo

Classe: modelando dados e operações

Uma possível estrutura Data e suas operações em .h

```
// Tipo Abstrato de Dados
// data.h

#include<stdbool.h>
typedef struct
      {
        int dia;
        int mes,
        int ano;
      } Data;
```

```
void escrevaData(Data D)
{
    printf("\n%i/%i/%i", D.dia, D.mes, D.ano);
}

void leiaData(Data D)
{
        printf("\ndd/mm/aaaa: ");
        scanf("%i/%i/%i", &D.dia, &D.mes, &D.ano);
}
```

Classe: modelando dados e operações

Descrevendo o tipo
Data como uma classe

```
class Data
{
   private :
      int dia;
      int mes;
      int ano;

   public :
      void escreveData()
      {
         printf("\n%i/%i/%i", dia, mes, ano);
      }
      void leiaData()
      {
            printf("\ndd/mm/aaaa: ");
            scanf("%i/%i/%i", &dia, &mes, &ano);
      }
};
```

Encapsulamento

```
class Data
{
   private :
      int dia;
      int mes;
      int ano;

   public :
      void escreveData()
      {
         printf("\n%i/%i/%i", dia, mes, ano);
      }
     void leiaData()
      {
            printf("\ndd/mm/aaaa: ");
            scanf("%i/%i/%i", &dia, &mes, &ano);
      }
};
```

Observe ao tentar:

```
Data D;

D.dia = 5;
D.mes = 6;
D.ano = 2024;
```

Encapsulamento

Encapsulamento

Encapsulamento

Implementado através do controle do escopo de visibilidade das propriedades de uma classe.

Classes e objetos proveem o encapsulamento.

Mensagem:

Evocar uma operação sobre um objeto.

Público *versus* Privado.

Métodos *sets* e *gets*: *Permitem implementar domínio dos dados e regras de negócio*

Encapsulamento

```
class Data
{
   private :
      int dia;
      int mes;
      int ano;

   public :
      void escreveData()
      {
          printf("\n%i/%i/%i", dia, mes, ano);
      }
     void leiaData()
      {
            printf("\ndd/mm/aaaa: ");
            scanf("%i/%i/%i", &dia, &mes, &ano);
      }
};
```

Alterar escopo de visibilidade

.

Alternativas de como solucionar (1):

```
Data D;

D.dia = 5;
D.mes = 6;
D.ano = 2024;
```

```
class Data
{
   private :
      int dia;
      int mes;
      int ano;

   public :
      bool setDia( int dia )
      {
          bool sucesso=false;
          if(dia >= 0 && dia <= 31)
          {
              this->dia = dia;
              sucesso=true;
          }
          return sucesso;
      }
      ...
};
```

Encapsulamento

Alternativas de como solucionar (2):

Prover método público
para manipular atributo privado.

Isto traz uma alternativa para
implementar regras de negócio.

```
class Data
{
   private :
      int dia;
      int mes;
      int ano;

   public :
      bool setDia( int dia )
      {
          bool sucesso=false;
          if(dia >= 0 && dia <= 31)
          {
              this->dia = dia;
              sucesso=true;
          }
          return sucesso;
      }
      ...
};
```

Encapsulamento

Em discussão:

- Implementação de regra de negócio
- Domínio
- Manutenibilidade
- Dado-Membro *versus* Função-Membro
- Atributos e Métodos
- Mensagem
- Objeto corrente
- Escopo de visibilidade
- Variável paramétrica *versus* atributo
- Operador ***this***

Alguns aspectos da codificação em C++

Bibliotecas

iostream : operadores de entrada e saída

```
#include <iostream>

using namespace std;
```

cin e *cout*

```
std::cout << "Hello world!" << std::endl;

int idade;
std::cin >> idade;
```

cin e *cout*

```
#include <iostream>
using namespace std;

cout << "Hello world!" << endl;

int idade;
cin >> idade;
```

Strings em C++

string é um tipo primitivo em C++.

Exemplo:

```
string  nome;
```

Permite codificar algo como:

```
cin >> nome;
fflush(stdin); // Linux: __fpurge(stdin)
cin.ignore();

getline( cin, nome );

cout << "\nNome= " << nome;
cout << nome[0] << nome[2];
cout << "\nTamanho da string: " << nome.length();
```

O tipo Quadrado

Considere o tipo Quadrado descrito pelo seguinte atributo *(dado-membro)*:

Lado, um atributo do tipo real

E as seguintes operações:

Atribuir um valor parametrizado ao atributo Lado - rejeitar valor negativo;

Recuperar o valor do atributo Lado;

Gerar o perímetro do quadrado;

Gerar a área do quadrado;

Ler um valor do teclado e guardá-lo no atributo Lado;

Escreva na tela do monitor de vídeo o valor atribuído ao atributo Lado.

•

Na função principal:

Criar uma coleção de instâncias de quadrado (de tamanho MAX) e:

- para cada instância, guardar o valor do lado fornecido pelo usuário;
- para cada instância, escrever na tela:

- a sequência do quadrado (1, 2, 3 *etc.);*
- o **valor** do lado daquele quadrado;
- o **perímetro** daquele quadrado;
- a **área** daquele quadrado.

•

No Trabalho Prático Final: uma alternativa de codificação da classe Data em C++

Classe Data

Privado :
- dia
- mes
- ano

Público:
- setDia(int)
- setMes(int)
- setAno(int)
- setData(int, int, int)
- getDia()
- getMes()
- getAno()
- dataValida()
- mesExtenso()
- diasMes()
- escrevaData()
- leiaData()

```
class Data
{
   private :
      int dia;
      int mes;
      int ano;

   public :
      bool setDia( int dia )
      {
          bool sucesso=false;
          if(dia >= 0 && dia <= 31)
          {
              this->dia = dia;
              sucesso=true;
          }
          return sucesso;
      }
```

```
};
```

Apoio para modelagem: um padrão

Notação UML

Unified Modeling Language

* Linguagem de Modelagem Unificada

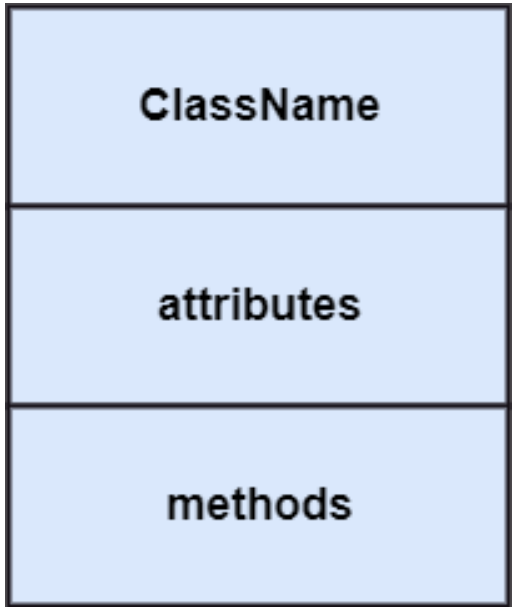
Representação de todos os elementos da OO, como classes, objetos e suas relações

Os três amigos: Rumbaugh, Jacobson e Booch

Relação com Processo Unificado

Classes em UML

Três partes: identificação, atributos e métodos



Classe Data

Data
- dia : Integer - mes : Integer - ano : Integer
+ setDia(Integer) : Boolean + setMes(Integer) : Boolean + setAno(Integer) + setData(Integer, Integer, Integer) : Boolean + getDia() : Integer + getMes() : Integer + getAno() : Integer + dataValida() : Boolean + mesExtenso() : String + diasMes() : Integer + escrevaData() + leiaData()

Classes Data e Pessoa

Pessoa
- nome : String - nascimento : Data
+ setNome(String) + getNome() : String + setNascimento(Integer, Integer, Integer):Boolean + getNascimento() : Data + leiaNome() + escrevaNome() + leiaPessoa() + escrevaPessoa()

Data
- dia : Integer - mes : Integer - ano : Integer
+ setDia(Integer) : Boolean + setMes(Integer) : Boolean + setAno(Integer) + setData(Integer, Integer, Integer) : Boolean + getDia() : Integer + getMes() : Integer + getAno() : Integer + dataValida() : Boolean + mesExtenso() : String + diasMes() : Integer + escrevaData() + leiaData()

Questões relativas ao trabalho

A classe Pessoa deverá ser descrita por dois dados:

```
string  nome;
Data    nascimento;
```

onde Data é também uma classe.

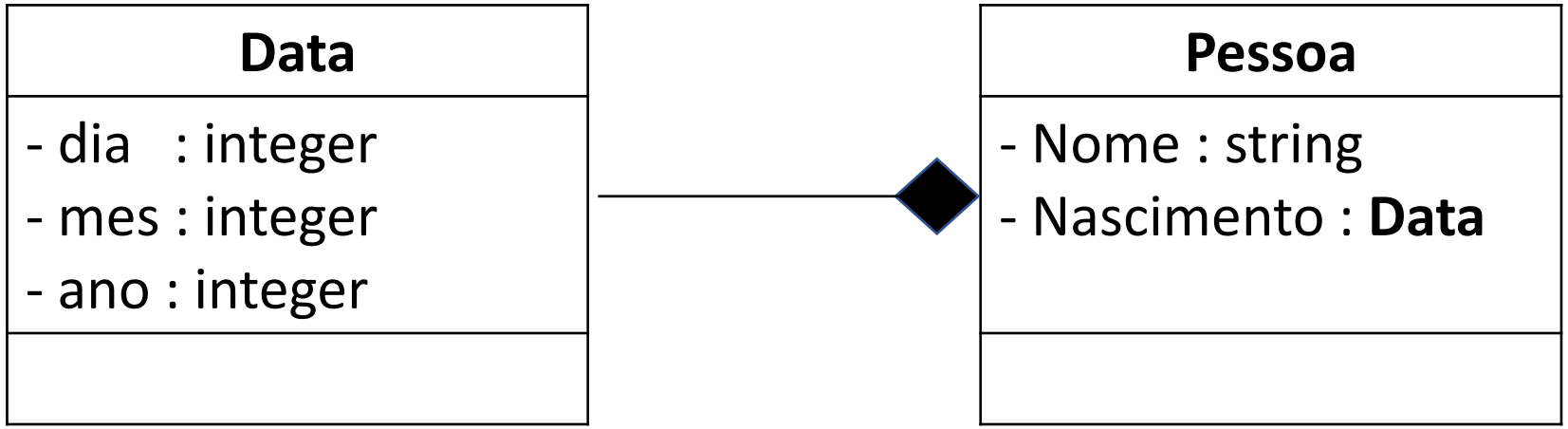
Aguarde: Um terceiro atributo será acrescentado à Pessoa

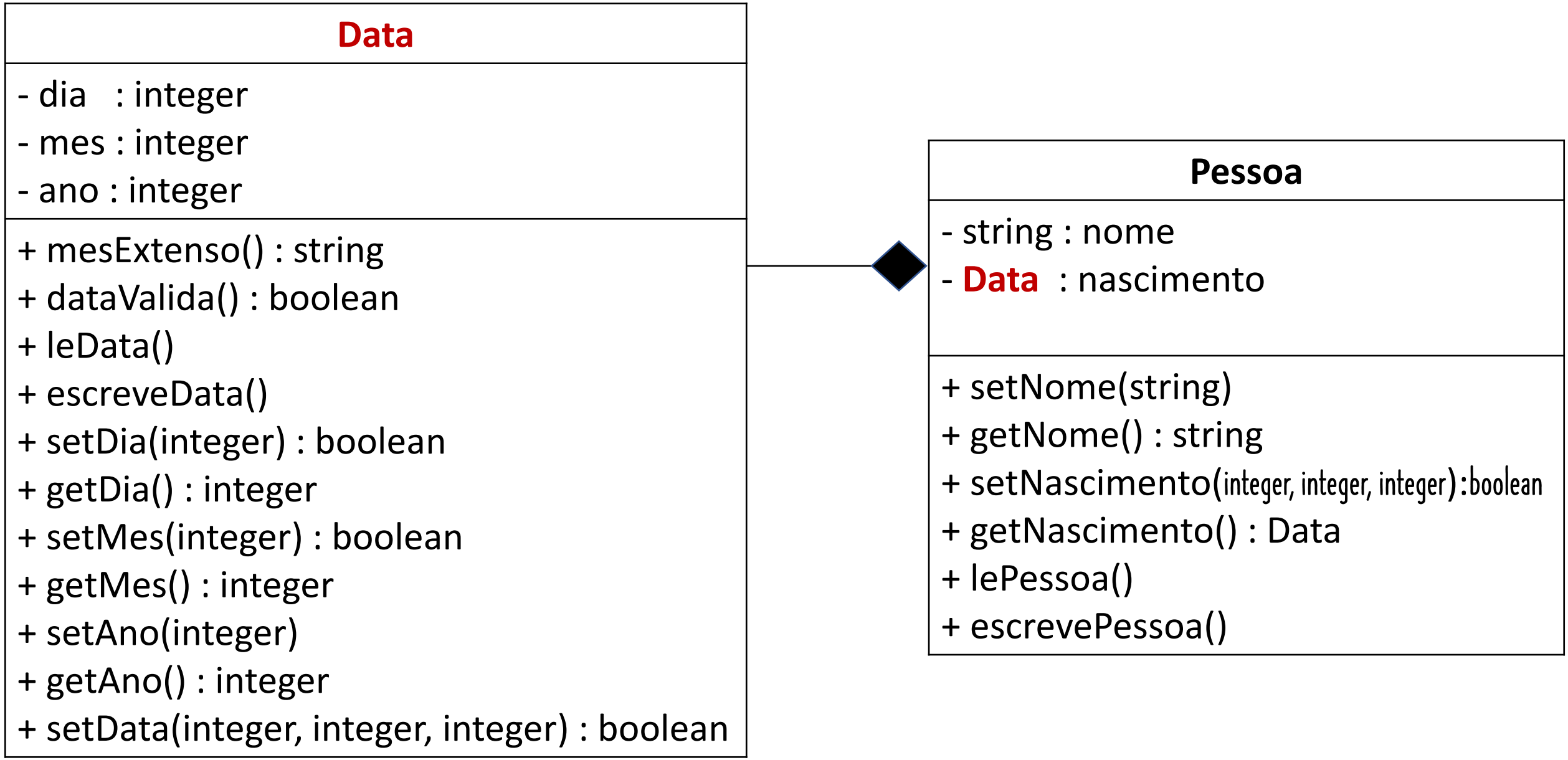
Composição

No mundo real, é comum um objeto ser composto por outros.

Ao modelar, dizer que uma pessoa tem uma *data de nascimento* é dizer que a *data de nascimento* faz parte da classe Pessoa.

Relacionamento do tipo: ***has a***




Data
- dia : integer
- mes : integer
- ano : integer
+ mesExtenso() : string
+ dataValida() : boolean
+ leData()
+ escreveData()
+ setDia(integer) : boolean
+ getDia() : integer
+ setMes(integer) : boolean
+ getMes() : integer
+ setAno(integer)
+ getAno() : integer
+ setData(integer, integer, integer) : boolean
Pessoa
- string : nome
- Data : nascimento
+ setNome(string)
+ getNome() : string
+ setNascimento(integer, integer, integer):boolean
+ getNascimento() : Data
+ lePessoa()
+ escrevePessoa()