Algoritmos e Estruturas de Dados I
Aula 13.1 - Destrutor, Composição e Herança

Prof. Felipe Lara

Construtor e Destrutor

A criação de objetos é feita por meio de **construtores**. São funções com o mesmo nome da classe e sem retorno, que inicializam o objeto e seus atributos.

- Construtor default:
  - é um construtor que pode ser chamado sem argumentos;

```
class  Produto  {

private :
    string descricao;
    float  preco ;
    int  quant ;

    public Produto ( )  {
        descricao = " Novo Produto " ;
        preco = 0.01F ;
        quant = 0 ;
    }
};
```

Construtor e Destrutor

Destrutor é função complementar às funções construtoras de uma classe. Sempre que o escopo de um objeto encerra-se, esta função é chamada.

Cada classe pode ter somente um destrutor que jamais recebe parâmetros. O destrutor também não tem nenhum tipo de retorno.

```
class Y {
  public:
    ~Y();
};
```

PUC Minas

Lista de Inicialização

Lista de Inicialização

Em C++, a lista de inicialização é uma forma especial de inicializar atributos de um objeto antes do corpo do construtor ser executado.

Ela é escrita após os dois pontos (:) e antes das chaves { } do construtor.

```
class Exemplo {
private:
    int x;
    int y;

public:
    Exemplo(int a, int b) : x(a), y(b) {
        // corpo do construtor
    }
};
```

Exemplo de uso de Lista de Inicialização

```
class Motor {
    int potencia;
public:
    Motor(int p){
       potencia =  p;
    }
};
```

Exemplo de uso de Lista de Inicialização

```
class Motor {
    int potencia;
public:
    Motor(int p){
       potencia =  p;
    }
};
```

```
class Motor {
    int potencia;
public:
    Motor(int p) : potencia(p) {}
};
```

PUC Minas

Composição

Composição

- Na composição utilizam-se objetos das classes existentes dentro da nova classe: a nova classe é composta de objetos de classes existentes.
- Trata-se de uma forma de reutilização do paradigma OO.
- É um tipo de relacionamento entre classes.

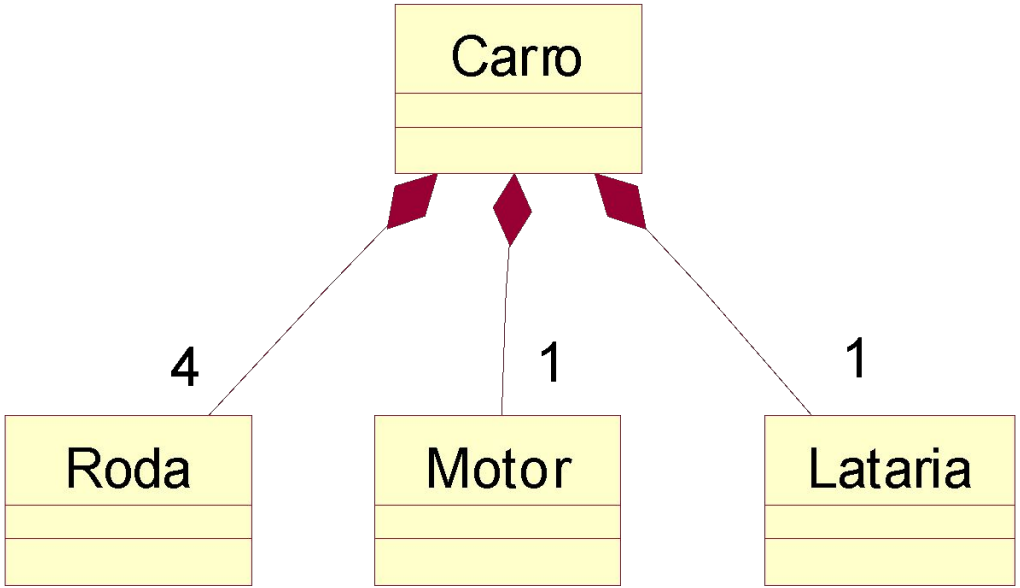
Composição

- Na composição utilizam-se objetos das classes existentes dentro da nova classe: a nova classe é composta de objetos de classes existentes.
- Trata-se de uma forma de reutilização do paradigma OO.
- É um tipo de relacionamento entre classes.
- Como se identifica a composição?
  - Identifica-se a possibilidade de composição através dos seguintes verbos típicos: conter, possuir. Ex: Um carro contém 4 rodas, 1 motor, 1 lataria, ...

Composição

- Na composição utilizam-se objetos das classes existentes dentro da nova classe: a nova classe é composta de objetos de classes existentes.
- Trata-se de uma forma de reutilização do paradigma OO.
- É um tipo de relacionamento entre classes.
- Como se identifica a composição?
  - Identifica-se a possibilidade de composição através dos seguintes verbos típicos: conter, possuir. Ex: Um carro contém 4 rodas, 1 motor, 1 lataria, ...
- A composição também é chamada de relacionamento do tipo parte-todo.
- O lado todo do relacionamento possui as partes.
- Trata-se de um relacionamento forte, pois no caso do objeto todo deixar de existir, as partes também deixam.

Composição



Exemplo de código de Composição

```
class Roda {
        public:
            Roda() {
                cout << "Roda criada.\n";
            }
            ~Roda() {
                cout << "Roda destruída.\n";
            }
};


class Motor {
        public:
            Motor() {
                cout << "Motor criado.\n";
            }
            ~Motor() {
                cout << "Motor destruído.\n";
            }
};
```

```
class Carro {
        private:
            Roda rodas[4];   // O carro *possui* 4 rodas → COMPOSIÇÃO
            Motor motor;     // O carro *possui* 1 motor → COMPOSIÇÃO
            string modelo;

        public:
            Carro(string m) : modelo(m) {
                cout << "Carro \"" << modelo << "\" criado.\n";
            }

            ~Carro() {
                cout << "Carro \"" << modelo << "\" destruído.\n";
            }
};

int main() {
            Carro c("Sedan");
            return 0;
}
```

PUC Minas

Herança

Herança

- Muitas vezes, classes diferentes têm características comuns
- Por exemplo:
  - Imagine uma classe **Pessoa** com as seguintes características:
    - nome, endereço, telefone
  - Se imaginarmos uma pessoa física, esta possui as seguintes características:
    - nome, endereço, telefone, CPF, RG
  - Se imaginarmos uma pessoa jurídica, esta possui as seguintes características:
    - nome, endereço, telefone, CNPJ, IE

Herança

- Muitas vezes, classes diferentes têm características comuns
- Por exemplo:
  - Imagine uma classe **Pessoa** com as seguintes características:
    - nome, endereço, telefone
  - Se imaginarmos uma pessoa física, esta possui as seguintes características:
    - nome, endereço, telefone, CPF, RG
  - Se imaginarmos uma pessoa jurídica, esta possui as seguintes características:
    - nome, endereço, telefone, CNPJ, IE
- Então, ao invés de criarmos uma nova classe para pessoa física e uma nova para pessoa jurídica, com todas essas características, **podemos usar as características da classe Pessoa já existente**.

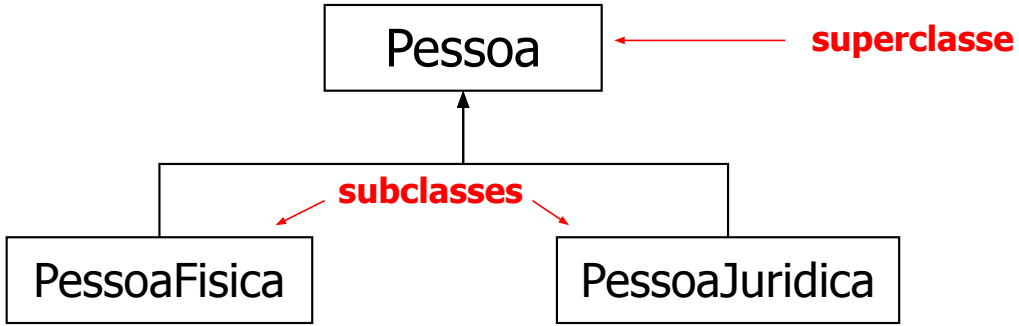
Herança

Em C++, definimos uma classe derivada de uma classe base da seguinte forma:

```
class Subclasse : public Superclasse {
        //definições da subclasse
};
```

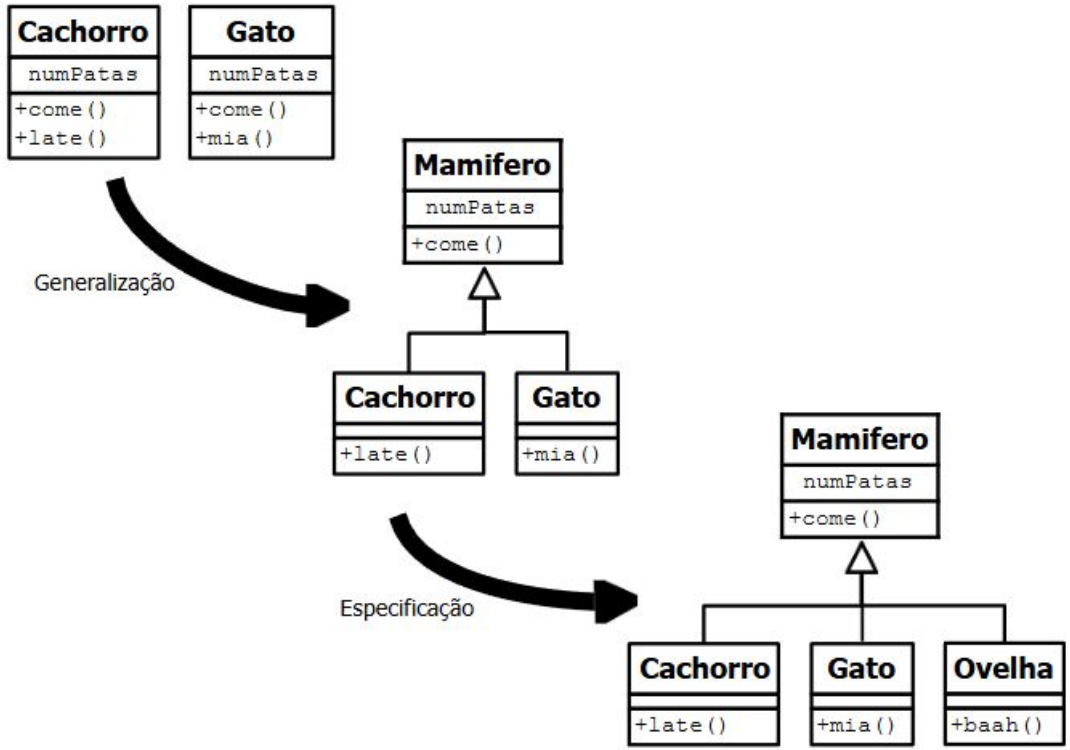
Herança

- **Superclasses** (ou ascendente): são as ascendentes de um classe
  - Também chamada de classe pai (mãe) ou classe base
- **Subclasses** (ou descendente): são as descendentes de um classe
  - Também chamada de classe filho (filha) ou classe derivada



PessoaFisica **é um tipo** de Pessoa
PessoaJuridica **é um tipo** de Pessoa

Herança


Cachorro
numPatas
+come()
+late()
Gato
numPatas
+come()
+mia()
Generalização
Mamifero
numPatas
+come()
Cachorro
+late()
Gato
+mia()
Especificação
Mamifero
numPatas
+come()
Cachorro
+late()
Gato
+mia()
Ovelha
+baah()

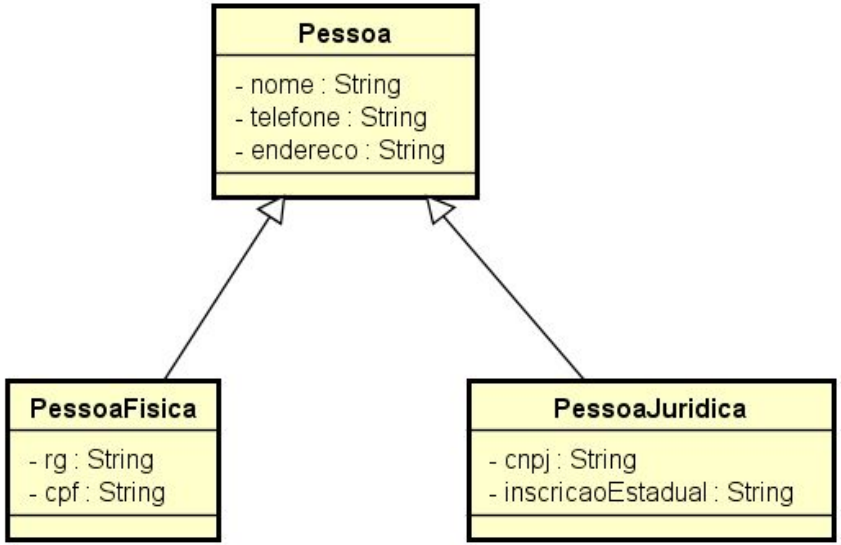
```cpp
#include <iostream>
#include <string>
using namespace std;

// Classe base
class Animal {
protected:
    string nome;

public:
    // Construtor
    Animal(string n) : nome(n) {

    }

    // Método para emitir som (polimórfico)
    virtual void som() {
        cout << nome << " está fazendo barulho" << endl;
    }

    // Método para exibir o nome
    void mostraNome() {
        cout << "Este é o " << nome << "." << endl;
    }
};
```

```
27  // Classe derivada
28  class Cachorro : public Animal {
29  public:
30      // Construtor para a classe derivada
31      Cachorro(string n) : Animal(n) {
32
33      }
34
35      // Método sobrescrito para emitir som específico
36      void som() override {
37          cout << nome << " diz: Au! Au!" << endl;
38      }
39
40      // Método exclusivo para a classe Dog
41      void buscar() {
42          cout << nome << " está pegando a bola!" << endl;
43      }
44  };
```

```
46  int main() {
47      // Criando um objeto da classe base
48      Animal animalGenerico("Animal Genérico");
49      animalGenerico.mostraNome();
50      animalGenerico.som();
51
52      cout << endl;
53
54      // Criando um objeto da classe derivada
55      Cachorro meuCachorro("Buddy");
56      meuCachorro.mostraNome();
57      meuCachorro.som();
58      meuCachorro.buscar();
59
60      return 0;
61  }
```

Outro exemplo

Implemente as classes em C++ conforme diagrama de classes a seguir. Por simplificação, na classe base considere apenas o atributo nome e, nas classes derivadas, apenas cpf e cnpj.



Atenção:
- Se um atributo na classe Base é privado (private), ele não é acessível nas subclasses.

- Se um atributo na classe Base é protegido (protected), ele é acessível nas subclasses. Vamos implementar, neste exemplo, os atributos em Pessoa como protected!

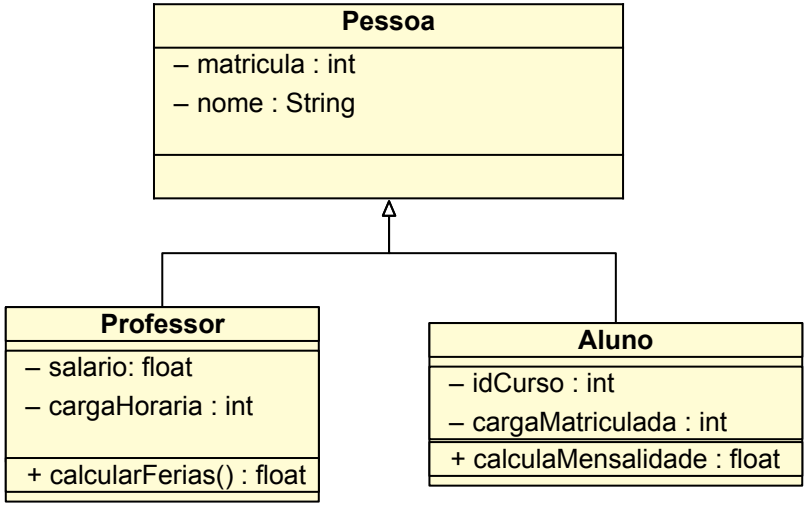
```
#include <iostream>
#include <string>
using namespace std;

class Pessoa {
protected:
   string nome;

public:
   Pessoa(string nome){
      this->nome = nome;
   }
};
```

```
class PessoaFisica : public Pessoa {
   string cpf;

public:
   PessoaFisica(string nome, string cpf)  :
                     Pessoa(nome), cpf(cpf) {
   }

   void imprimir() {
      cout << "PessoaFisica, nome=" << nome
               << ", cpf=" << cpf << endl;
   }
};
```

```
class PessoaJuridica : public Pessoa {
   string cnpj;

public:
   PessoaJuridica(string nome, string cnpj): Pessoa(nome) {
      this->cnpj = cnpj;
   }

   void imprimir() {
      cout << "PessoaJuridica, nome=" << nome << ",      cnpj=" << cnpj << endl;
   }
};


int main()
{
   PessoaFisica pessoaFisica("Joao da Silva",  "123456789-00");
   pessoaFisica.imprimir();
   PessoaJuridica pessoaJuridica("Empresa do Joao SA", "12.299.535/0001-94");
   pessoaJuridica.imprimir();
   return 0;
}
```

Exercício

Crie um código em C++ para representar o contexto apresentado no diagrama:



PUC Minas

Dúvidas?