Residência em Software

Introdução a Orientação a Objetos

Professores:

Álvaro Coelho, Edgar Alexander, Esbel Valero e Hélder Almeida

INSTITUIÇÃO EXECUTORA cepedi UESC

COORDENADORA MCTI FUTURO Softex

APOIO MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Residência em Software

Introdução

Objetivos:

- Compreender os conceitos fundamentais da programação orientada a objetos (POO) em C++.
- Aprender a criar classes, objetos, métodos e atributos em C++.
- Explorar os princípios do encapsulamento, herança e polimorfismo.

Residência em Software

Relembrando…

Até agora estudamos a sintaxe básica, as expressões, as estruturas de controle, o uso de funções, tipos de dados abstratos e outros tópicos… mas tudo do ponto de vista da programação estruturada (PE).

Na última aula vimos como o uso de structs junto com funções nos leva, intuitivamente, ao paradigma de Programação Orientada a Objetos (POO).

Residência em Software

PE x POO

Programação Estruturada:

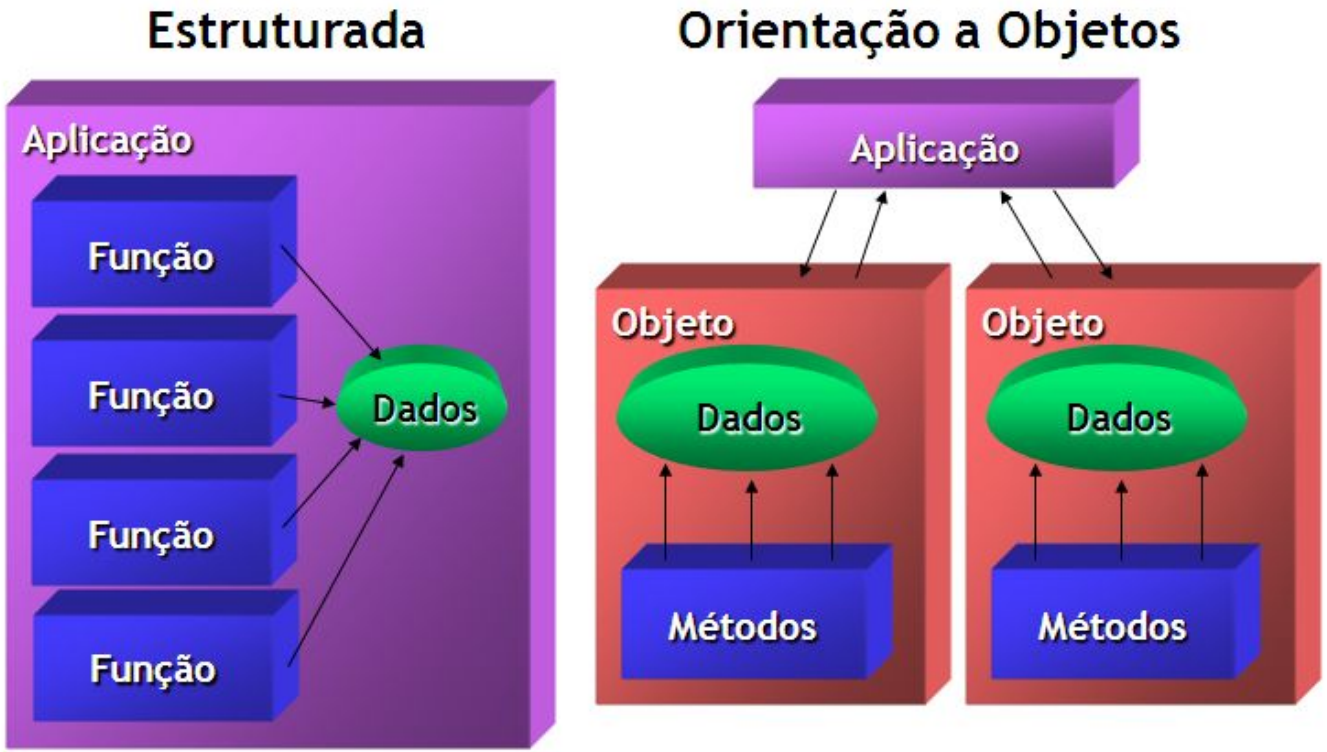
- **Sequência**: Uma tarefa é executada após a outra,linearmente.
- **Decisão**: A partir de um teste lógico, determinado trecho de código é executado, ou não.
- **Iteração**: A partir de um teste lógico, determinado trecho de código é repetido por um número finito de vezes.

Programação Orientada a Objetos:

- **Classes** e **Objetos**
- **Métodos** e **Atributos**

Residência em Software

PE x POO



Residência em Software

PE x POO

Programação Estruturada:

- **Vantagens:**
  - Fácil de entender. Execução mais rápida.
- **Desvantagens:**
  - Baixa reutilização de código. Dados misturados com comportamento.

Programação Orientada a Objetos:

- **Vantagens:**
  - Melhor organização e reaproveitamento do código.
- **Desvantagens:**
  - Desempenho inferior. Entendimento mais complexo e demorado.

História da POO

- Raízes do Conceito de Objetos:
  - Platão, 375 a.C., teoria das formas: uma forma ideal de uma cama é a base para todas as camas do mundo (A República).
- Anos 60 - SIMULA I e SIMULA 67:
  - Os conceitos de objetos, classes e herança apareceram sem essa nomenclatura.
- Anos 70 - Smalltalk:
  - Alan Kay propôs essa linguagem com o propósito de ser ser tão fácil quanto bater um papo furado (small talk) em inglês;

História da POO

AlanKay pensando na programação OO

História da POO

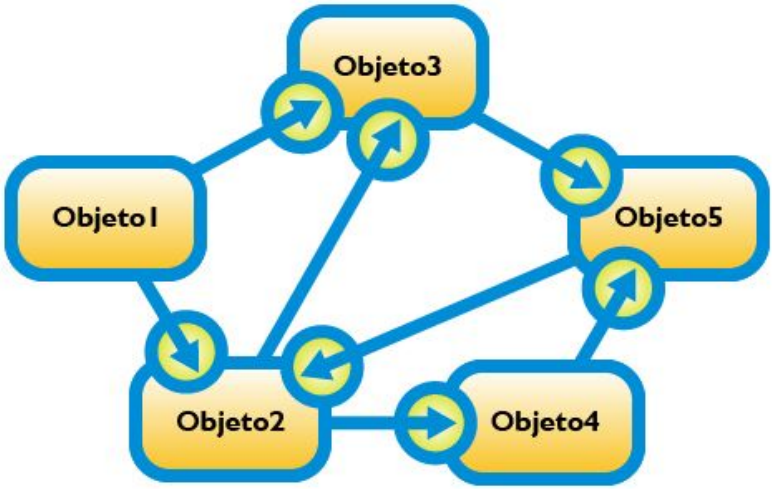
"I invented the term Object-Oriented, and I can tell you I did not have C++ in mind." - Alan Kay

- Anos 80 - C++:
  - Bjarne Stroustrup estendeu a linguagem C com alguns conceitos de objetos e criou C++;
  - **Em termos de orientação a objetos, pode-se dizer que C++ está no extremo oposto de Smalltalk**
- Anos 90 - Java e C#:
  - Java altamente influenciada por Smalltalk mas com sintaxe parecida com C++;
  - C# mais complexa que Java. Menos elegante e com melhor desempenho.
- Anos 2000 - Ruby:
  - Influenciada por Perl, Smalltalk, Eiffel e Lisp;
  - O criador,Yukihiro Matsumoto, queria produzir uma linguagem mais poderosa que Perl e mais orientada a objetos do que Python;
- A partir de 2015:
  - Explosão de novas linguagens: JavaScript , Python, Go, Rust, Kotlin, e outras.

Residência em Software

Conceitos de POO

Na POO, um programa é visto como um conjunto de objetos que se comunicam através de mensagens. Cada objeto mantém dados internos, chamados de atributos. Dessa forma, um sistema (programa de computador) desenvolvido usando a POO possui objetos que colaboram entre si, executando tarefas específicas em busca de um objetivo comum.



Residência em Software

Pilares da POO

A Programação Orientada a Objetos está sedimentada sobre quatro pilares derivados do princípio da abstração, são eles:

- **Encapsulamento**,
- **Herança**,
- **Composição** e
- **Polimorfismo**.

Residência em Software

Abstração

O Princípio da abstração é a nossa capacidade de abstrair a complexidade de um sistema e se concentrar em apenas partes desse sistema.

Exemplo: quando um médico torna-se um especialista em algum órgão do nosso corpo (exemplo, o coração), ele abstrai sem desconsiderar as influências dos outros órgãos e foca apenas sua atenção nesse órgão.

Residência em Software

Objetos

Na POO objetos são usados para representar entidades do mundo real ou computacional. Ou seja, os objetos são usados para representar aqueles elementos e abstrações que fazem parte da solução do programa que estamos desenvolvendo.

Os objetos são a base da POO, encapsulam dados e comportamentos, onde comportamento é o conjunto de funções (métodos) que operam nesses dados.

Objetos

Objetos podem representar entidades **concretas** ou **abstratas**:

Concretas: cachorro, carro, casa, etc.
Abstratas: música, transação bancária, sentimento, etc.

Os objetos possuem **características** (atributos) e **comportamentos** (métodos)**.**

Características (Atributos)	**Comportamentos (Métodos)**
Raça: Beagle Nome: Sherlock Peso: 8 kg	Latir Comer Brincar

Fabricante: Harley-Davidson Modelo: Breakout 117 Potência: 1.923 cm³	Acelerar Frear Abastecer

Residência em Software

Objetos - atividade

1. Cite 4 atributos de um aluno e 4 atributos de um professor.

2. Cite 4 métodos de um aluno e 4 métodos de um professor.

Residência em Software

Classes

Organizamos o mundo segundo a nossa visão das características e finalidades das coisas, logo pensamos o mundo Orientado a Objetos.

Normalmente agrupamos os objetos de acordo com suas características e comportamentos.

Em POO, **Classe** é o conceito usado para descrever (modelar e programar) um conjunto de objetos que possuem características e comportamentos comuns.

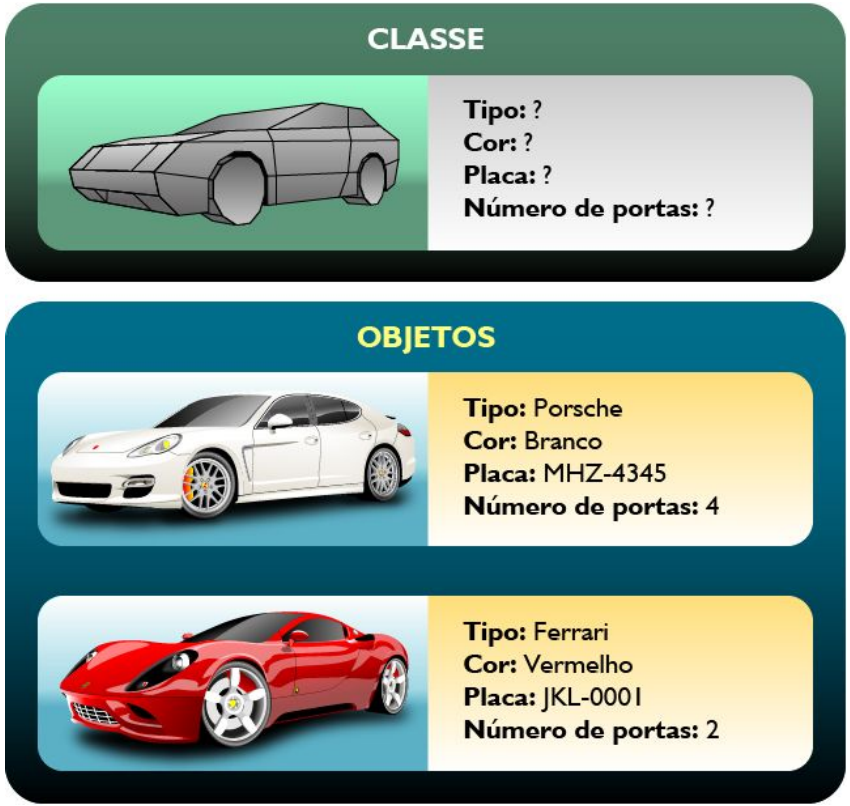
Residência em Software

Classes

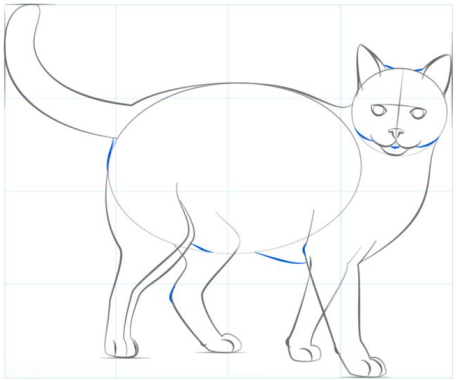
- A classe é o modelo ou molde de construção de objetos.
- O modelo define as características e comportamentos que os objetos irão determinar seus valores e desempenhar suas ações, respectivamente.
- A classe é abstrata (não existe concretamente).

Classes

- Conjunto de objetos:
  - Características semelhantes
  - Comportamento comum
  - Interação com outros objetos
- Uma **classe** é a forma para **criação** de objetos
- Objetos são representações concretas (**instâncias**) de uma classe



Classes e Objetos



Gato
Raça: Savannah
Nome: Gatuno
Peso: 2,5 quilos
Idade: 2 anos

Gato
Raça: Maine Moon
Nome: Listrado
Peso: 3 quilos
Idade: 5 anos

Gato
Raça: Siamês
Nome: Bichano
Peso: 4 quilos

Residência em Software

Classes - atividade

Imagine classes para: Livro, Celular, Mamífero, Ave e Flor.

1. Como seriam seus objetos?

2. Escreva as características (ou atributos) das classes e objetos!

Residência em Software

Classes em C++

Uma definição de classe em C++ é muito similar a uma definição de estrutura.

```
class arquivo
{
    public:
        char nome_arquivo[32];    // variável-membro publica
        long tamanho;
    private:
        unsigned proteção;    // variável-membro privada
        long data;
        int copiar(char *nome_destino);    //função-membro privada
        int renomear(char *nome_destino);
        int deletar(void);
}
```

Residência em Software

Objetos em C++

Assim como variáveis em de tipos predefinidos (int, double, char, etc.), podemos declarar tantas variáveis quanto quisermos de uma classe já definida. Uma classe não cria um objeto. Para criar um objeto, você simplesmente especifica o tipo da classe, seguido pelo nome da variável objeto:

```
main() {
    nome_da_classe nome_do_objeto;
     // cria um objeto nome_do_objeto do tipo nome_da_classe
     :
     :
    int var; // definição de variável simples do tipo int
}
```

Residência em Software

Primeiros exemplos


```
class Carro{
    public:
        string tipo;
        string cor;
        string placa;
        int numPortas;
};
```

Residência em Software

Mensagem

Objetos só são úteis quando associados com outros objetos para alcançar um resultado.

Ex. *Para modelar o uso de objetos do tipo Carro, que vimos a pouco, poderíamos criar um objeto do tipo motorista e usá-lo associado ao objeto carro. Por usa vez o motorista aumentaria a velocidade do carro através de mensagens como “acelerar” e “frear”.*

Uma mensagem possui, basicamente, três componentes:

- objeto para receber a mensagem (carro);
- um nome da operação (método/função-membro) a ser executada pelo objeto (acelerar/frear);
- um parâmetro (argumento) do qual a velocidade pode depender (velocidade).

Comportamentos (métodos)


```cpp
#include <iostream>
#include <string>

using namespace std;
```

```cpp
class Carro{
    public:
        string tipo;
        string cor;
        string placa;
        int numPortas;

        void acelerar(float _velocidade){
            cout << "acelerando para " << _velocidade
            << " km/h" << endl;
        }

        void frear (){
            cout << "freando" << endl;
        }
};
```

```cpp
int main(void){
    Carro civic;
    civic.tipo = "Sedan";
    cout << civic.tipo << endl;

    civic.acelerar(30);
    civic.frear();
}
```

Residência em Software

Praticar

- Implemente as classes que você imaginou no slide anterior:
  - Livro
  - Celular
  - Mamífero
  - Ave
  - Flor
- Crie objetos a partir destas classes para testar seu funcionamento.

Residência em Software

Encapsulamento

A definição de uma classe em C++ basicamente cria um novo tipo de dado, semelhante à uma struct vista anteriormente. A diferença básica é que funções que representam o comportamento do tipo também podem ser incluídas.

Mas, adicionando o conceito de encapsulamento, as classes ganham super poderes!

Encapsulamento é o processo de organizar os elementos de uma abstração de modo que os clientes de uma determinada classe saibam apenas o necessário sobre o comportamento dos objetos que a ela pertencem!

Residência em Software

Encapsulamento

O processo de encapsulamento permite que o usuário tenha controle sobre o que pode e o que não pode ser observado ou modificado nas entranhas dos objetos de uma classe. Isto é feito através do uso de especificadores de acesso que, em C++, são três:

- **Public**
  - Acessíveis por todos os clientes.
- **Private**
  - Acessíveis, exclusivamente, pela própria classe e suas “amigas” (friends).
- **Protected**
  - Acessíveis pela própria classe, suas “amigas” e suas subclasses

Os conceitos de classes “amigas” e subclasses serão vistos mais à frente!

Residência em Software

Encapsulamento

Em C++, por padrão, todos os itens definidos numa classe são privados.

Embora se possa ter variáveis públicas, filosoficamente deve-se tentar limitar ou eliminar seu uso. O ideal é tornar todos os dados privados e controlar o acesso a eles com funções públicas, principalmente para aplicar as devidas validações e evitar erros nas atribuições

Vamos refazer o exemplo anterior, utilizando o conceito de encapsulamento.

```
class Carro{
    private:
        string tipo;
        string cor;
        string placa;
        int numPortas;

    public:
        string getTipo() {
            return tipo;
        }

        void setTipo(string _tipo) {
            tipo = _tipo;
        }

        string getCor() {
            return cor;
        }

        void setCor(string _cor) {
            cor = _cor;
        }

        string getPlaca() {
            return placa;
        }

        void setPlaca(string _placa) {
            placa = _placa;
        }

        int getNumPortas() {
            return numPortas;
        }

        void setNumPortas(int _numPortas) {
            numPortas = _numPortas;
        }

        void acelerar(float _velocidade){
            cout << "acelerando para " << _velocidade
            << " km/h" << endl;
        }

        void frear (){
            cout << "freando" << endl;
        }
};
```

Residência em Software

Construtores

Qual o valor inicial, assim que criamos os objetos, dos 4 atributos do exemplo anterior?

A priori, pode-se afirmar que são os conteúdos presentes nas regiões de memória reservadas para abrigar estas variáveis. Geralmente, é praticamente impossível prever que valores seriam estes. Portanto, pode ser bastante conveniente a existência de mecanismos que inicializem as variáveis internas do objeto para que seu estado inicial seja previamente conhecido.

Para isso (e outras coisas legais!) existem os **construtores!** Que são métodos chamadas automaticamente logo após o objeto ser criado, permitindo que o programador defina todos os procedimentos necessários para inicializar o estado do respectivo objeto após a sua criação. Em outras palavras, eles permitem construir o estado inicial de um objeto.

Construtores

Em C++, os construtores são métodos que recebem o mesmo nome da classe, podem existir mais de um por classe, e são diferenciados pelo número e tipos de argumentos.

Para o nosso exemplo de classe carros, podemos criar um construtor da seguinte forma:

```
public:
    Carro(){
        tipo="indefinido";
        cor="indefinida";
        placa="indefinido";
        numPortas=2;
    }
```

```
int main(void){
    Carro civic;
```

E se quiséssemos garantir que o objeto tivesse, ao menos, o valor da placa informado pelo cliente?! O que mudar?

```
public:
    Carro(string _placa){
        tipo="indefinido";
        cor="indefinida";
        placa=_placa;
        numPortas=2;
    }
```

```
int main(void){
    Carro civic("OUP-4062");
```

Construtores

Em C++, podemos utilizar o artifício dos argumentos *default*, que permite atribuir valores automaticamente caso seja omitido o argumento correspondente para a função chamada. Argumentos *default* devem ser fornecidos da direita para a esquerda de modo que não se pode prover valores para um argumento default a menos que todos os argumentos à sua direita tenham valores default também atribuídos.

```
Carro(
string _placa,
string _tipo="indefinido",
string _cor="indefinido",
int _numPortas=2
){
    tipo=_tipo;
    cor=_cor;
    placa=_placa;
    numPortas=_numPortas;
}
```

Qual(is) argumento(s) pode(m) ser omitido(s) ao instanciar um objeto do tipo Carro?

```
Carro civic("OUP-4062");
```

```
Carro civic("OUP-4062",4);
```

```
Carro civic("OUP-4062",4,"SEDAN MEDIO");
```

```
Carro civic("OUP-4062",4,"SEDAN MEDIO","CINZA");
```

Residência em Software

Destrutores

Destrutores são **chamados automaticamente quando o objeto está para ser excluído da memória da máquina**. Podem ser importantes para **liberar recursos** alocados ou acionar outros mecanismos necessários durante a fase de destruição de um objeto criado.

Somente um destrutor pode existir, de modo que é vedado ao programador escolher a função de destruição, que é única e não recebe parâmetros. Ela **recebe o mesmo nome da classe precedida pelo símbolo ~.**

```
~Carro(){
    cout << "Carro destruido!" << endl;
}
```

Residência em Software

Atividade em sala

- Divididos em 5 grupos, cada grupo deve aplicar os conceitos vistos na aula para as classes:
  - Livro
  - Celular
  - Mamífero
  - Ave
  - Flor
- Abusem da criatividade e explorem pontos não vistos na aula.
- O passo a passo das demonstrações devem ser versionados no Git e enviados para o Github.

Residência em Software

Referências

DEITEL, H. M.; DEITEL, P. J. Java como programar. Porto Alegre: Bookman, 2003.

Instituto Metrópole Digital. Curso de Automação Industrial. Disponível em: https://materialpublic.imd.ufrn.br/curso/disciplina/1/8/1/13. Acessado em 02 de Agosto de 2023.

PEREIRA, Silvio do Lago. Linguagem C++. São Paulo: FATEC, 1999. Disponível em: https://www.ime.usp.br/~slago/slago-C++.pdf. Acessado em 03 de agosto de 2023.