Introdução à Programação Orientada a Objetos em C++

1. Objetivos da Aula

- Introduzir os conceitos básicos de Programação Orientada a Objetos (POO).
- Demonstrar diferenças entre C e C++ por meio de exemplos práticos.
- Evidenciar as vantagens da POO em comparação ao estilo procedural.

---

2. Introdução à Programação Orientada a Objetos

2.1. O que é POO?

A Programação Orientada a Objetos é um paradigma que organiza o código em torno de **objetos**, que são instâncias de **classes**. Cada classe encapsula **atributos** (dados) e **métodos** (funções) para operar sobre esses dados.

Principais conceitos:

- **Encapsulamento:** Oculta detalhes internos e expõe apenas o necessário.
- **Herança:** Reutilização de código entre classes.
- **Polimorfismo:** Comportamentos diferentes para a mesma interface.
- **Composição:** Objetos dentro de objetos para construir funcionalidades complexas.

---

3. Comparação entre C e C++

Aspecto	C	C++
Encapsulamento	Não há	Oferecido por meio de classes.
Herança	Inexistente	Permite a criação de hierarquias.
Polimorfismo	Não suportado	Implementado com classes virtuais.
Gestão de memória	Manual (malloc, free)	Automática com `new` e `delete`.
Estruturas	Apenas armazenamento de dados	Classes com dados e métodos.

---

4. Exemplo Prático: Estruturas em C vs Classes em C++

4.1. Código em C: Estruturas

```
#include <stdio.h>
#include <string.h>

typedef struct {
```

```
    char nome[50];
    int idade;
} Pessoa;

void exibir_pessoa(Pessoa* p) {
    printf("Nome: %s, Idade: %d\n", p->nome, p->idade);
}

int main() {
    Pessoa pessoa1;
    strcpy(pessoa1.nome, "João");
    pessoa1.idade = 30;
    exibir_pessoa(&pessoa1);
    return 0;
}
```

4.2. Código em C++: Classes

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

using namespace std;

class Pessoa {
public:
    string nome;
    int idade;

    void exibir_detalhes() {
        cout << "Nome: " << nome << ", Idade: " << idade << endl;
    }
};

int main() {
    Pessoa pessoa1;
    pessoa1.nome = "João";
    pessoa1.idade = 30;
    pessoa1.exibir_detalhes();
    return 0;
}
```

---

5. Classes e Objetos em C++

5.1. Estrutura de uma Classe

```
class MinhaClasse {
public:
    int atributo;
```

```
    void metodo() {
        cout << "Método chamado!" << endl;
    }
};
```

5.2. Declaração e Instância de Objetos

```
int main() {
    MinhaClasse obj;
    obj.atributo = 10;
    obj.metodo();
    return 0;
}
```

---

6. Encapsulamento

Controle de acesso com modificadores:

- **public:** Acessível de qualquer lugar.
- **private:** Acessível apenas dentro da classe.
- **protected:** Acessível dentro da classe e classes derivadas.

Exemplo:

```
class Exemplo {
private:
    int valor_privado;
public:
    void set_valor(int v) { valor_privado = v; }
    int get_valor() { return valor_privado; }
};
```

---

7. Construtores e Destrutores

7.1. Construtor

Método especial chamado ao criar um objeto.

```
class Livro {
public:
    string titulo;
    Livro(string t) { titulo = t; }
};
```

7.2. Destrutor

Método chamado ao destruir um objeto.

```
~Livro() { cout << "Livro destruído!" << endl; }
```

---

8. Alocação Dinâmica de Memória

C++ facilita a alocação dinâmica com `new` e `delete`.

Exemplo:

```
int* ptr = new int;
*ptr = 20;
delete ptr;
```

---

9. Ponteiros e Referências

Característica	Ponteiro	Referência
Nulidade	Pode ser nulo	Não pode ser nula.
Reatribuição	Pode mudar de endereço	Fixa após inicialização.
Sintaxe	Usado com `*` e `&`	Usado com `&`.

Exemplo de Referência:

```
int valor = 10;
int& ref = valor;
ref = 20; // Altera valor diretamente
```

---

10. Herança e Polimorfismo (Introdução)

Herança permite que uma classe herde atributos e métodos de outra.

```
class Animal {
public:
    void som() { cout << "Emitindo som" << endl; }
};

class Cachorro : public Animal {
public:
```

```
    void som() { cout << "Latido!" << endl; }
};
```
