Residência em Software

Prática sobre Conceitos Básicos de Estatística

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

Objetivo

Importância do Polimorfismo:

- O polimorfismo é um conceito essencial na programação orientada a objetos.
- Ele permite a flexibilidade, reutilização de código e facilita a manutenção do software.

Tópicos Abordados na Aula:

- Definição de Polimorfismo
- Tipos de Polimorfismo
- Exemplos Práticos em C++
- Vantagens do Polimorfismo

Residência em Software

O que é polimorfismo?

O polimorfismo é um conceito fundamental na programação orientada a objetos (POO) que se baseia na capacidade de diferentes objetos responderem de maneira única a chamadas de funções comuns. Em termos simples, o polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme, tornando o código mais flexível e extensível.

Distinguir o polimorfismo da sobrecarga de funções é essencial. A sobrecarga de funções envolve a criação de várias funções com o mesmo nome, mas com argumentos diferentes. O polimorfismo, por outro lado, lida com objetos de diferentes classes que compartilham uma interface comum.

Residência em Software

Tipos de polimorfismo

Existem diferentes tipos de polimorfismo em programação, cada um com suas próprias aplicações e implementações.

- Polimorfismo de Subtipos
- Polimorfismo Paramétrico (Templates em C++)
- Polimorfismo de Inclusão (Interfaces em C++)

Residência em Software

Polimorfismo de Subtipos

Este tipo de polimorfismo é baseado na herança e na capacidade das classes derivadas de substituir métodos de uma classe base. A palavra-chave **virtual** é usada em C++ para marcar os métodos que podem ser substituídos por classes derivadas.

Quando um método marcado como virtual é chamado em um objeto de uma classe derivada, o compilador seleciona a implementação correta com base no tipo real do objeto em tempo de execução. Isso permite que objetos de classes diferentes respondam de maneira única às chamadas de função.

O polimorfismo de subtipos é uma das características mais poderosas da programação orientada a objetos, permitindo que o código seja flexível e extensível.

Residência em Software

Exemplo

```cpp
#include <iostream>

class Animal {
public:
    virtual void som() {
        std::cout << "Som genérico de um animal" << std::endl;
    }
};

class Cachorro : public Animal {
public:
    void som() override {
        std::cout << "Latido de cachorro" << std::endl;
    }
};

class Gato : public Animal {
public:
    void som() override {
        std::cout << "Miado de gato" << std::endl;
    }
};

int main() {
    Animal* animal;

    Cachorro cachorro;
    Gato gato;

    animal = &cachorro;
    animal->som();  // Chama o método som() da classe Cachorro

    animal = &gato;
    animal->som();  // Chama o método som() da classe Gato

    return 0;
}
```

Residência em Software

Polimorfismo Paramétrico (Templates em C++)

Nesse tipo, o código é parametrizado pelo tipo de dado, permitindo a reutilização de código para diferentes tipos. Isso é alcançado por meio de templates, uma característica poderosa em C++.

O polimorfismo paramétrico, ou o uso de templates em C++, permite que funções e classes sejam genéricas em relação aos tipos de dados que manipulam. Isso é uma parte essencial do C++ moderno, permitindo a criação de estruturas de dados e algoritmos que funcionam com diferentes tipos de dados de maneira eficiente. É uma técnica valiosa para escrever código flexível e reutilizável em situações em que os tipos de dados podem variar. Durante a apresentação, exploraremos exemplos práticos que destacam a utilidade e a versatilidade dos templates em C++.

Residência em Software

Exemplos

```cpp
#include <iostream>

template <typename T>
T maior(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
    int num1 = 5, num2 = 10;
    double x = 3.5, y = 7.2;

    std::cout << "Maior entre " << num1 << " e " << num2 << " é: " << maior(num1, num2) << std::endl;
    std::cout << "Maior entre " << x << " e " << y << " é: " << maior(x, y) << std::endl;

    return 0;
}
```

Residência em Software

Polimorfismo de Inclusão (Interfaces em C++)

Interfaces em C++ são semelhantes às interfaces em outras linguagens de programação e permitem que várias classes compartilhem uma interface comum. Isso é especialmente útil quando diferentes classes têm comportamentos semelhantes, mas não estão relacionadas por herança.

Quando uma classe implementa uma interface, ela deve fornecer uma implementação para todos os métodos definidos na interface. Isso garante que objetos de diferentes classes que implementam a mesma interface possam ser tratados de maneira uniforme.

O polimorfismo de inclusão é valioso quando se trabalha com várias classes que devem se comportar de maneira consistente em certos aspectos, independentemente de sua hierarquia de herança.

Residência em Software

Exemplos

```cpp
#include <iostream>

class Imprimivel {
public:
    virtual void imprimir() = 0;
};

class Livro : public Imprimivel {
public:
    void imprimir() override {
        std::cout << "Imprimindo um livro..." << std::endl;
    }
};

class Documento : public Imprimivel {
public:
    void imprimir() override {
        std::cout << "Imprimindo um documento..." << std::endl;
    }
};

int main() {
    Livro livro;
    Documento documento;

    Imprimivel* item1 = &livro;
    Imprimivel* item2 = &documento;

    item1->imprimir();  // Chama o método imprimir() da classe Livro
    item2->imprimir();  // Chama o método imprimir() da classe Documento

    return 0;
}
```

Residência em Software

Exercício 1: Polimorfismo de Subtipos

Crie uma hierarquia de classes que representam diferentes formas geométricas, como **círculos** e **retângulos**. Implemente um método **area()** em cada classe que calcule a área da forma. Use o polimorfismo para chamar o método **area()** de diferentes objetos e calcular suas áreas.

Residência em Software

Exercício 2: Polimorfismo de Inclusão (Interfaces)

Crie uma **interface** chamada **Desenhavel** que contenha um método **desenhar()**. Em seguida, crie várias classes que implementem essa **interface**, como **Circulo**, **Retangulo** e **Triangulo**. Cada classe deve ter uma implementação diferente do método **desenhar()**. Use o polimorfismo para chamar o método **desenhar()** em diferentes objetos e exibir suas representações gráficas.

Residência em Software

Exercício 3: Polimorfismo Paramétrico (Templates)

Crie uma classe de lista genérica que use templates para armazenar elementos de qualquer tipo. Implemente métodos para adicionar, remover e listar elementos da lista. Use a classe de lista para armazenar diferentes tipos de dados, como inteiros, strings e números de ponto flutuante.