Residência em Software

Interfaces

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

Interfaces

- Em C++, uma interface é frequentemente implementada usando uma classe abstrata que contém apenas métodos puramente virtuais.
- As interfaces definem contratos que as classes derivadas devem seguir.

Residência em Software

Interfaces

```
class Imprimivel {
public:
    virtual void imprimir() const = 0;
    // ... outros métodos relacionados à impressão
};

class Documento : public Imprimivel {
public:
    void imprimir() const override {
        // Implementação específica para imprimir um documento
    }
    // ... outros métodos da classe Documento
};
```

Residência em Software

void imprimir() const override { }

1. void imprimir() const:

- **void**: Indica que o método não retorna nenhum valor.
- **imprimir()**: É o nome do método.
- **const:** Declara que o método não modificará nenhum membro de dados da classe (é um método constante).

Este método, em termos gerais, seria responsável por imprimir ou exibir informações associadas a um objeto da classe.

2. override:

- **override** é uma palavra-chave introduzida no C++11 para **indicar explicitamente** que a função está substituindo uma função virtual da classe base.
- Isso ajuda a evitar erros sutis de digitação ou alterações inadvertidas na assinatura do método.

Residência em Software

Vantagens

- **Reutilização de Código**: Classes abstratas e interfaces promovem a reutilização de código ao definir contratos claros para as classes derivadas.
- **Polimorfismo**: Permite a utilização de polimorfismo, onde objetos de classes derivadas podem ser tratados de maneira uniforme através de ponteiros ou referências à classe base.
- **Manutenção e Extensibilidade**: Facilita a manutenção e extensibilidade do código, pois novas classes podem ser adicionadas sem modificar as existentes.

Residência em Software

Conclusão

- Classes abstratas e interfaces são elementos cruciais na construção de sistemas orientados a objetos em C++.
- Elas fornecem estruturas poderosas para criar hierarquias de classes flexíveis e bem definidas.
- Ao adotar esses conceitos, os desenvolvedores podem criar sistemas mais eficientes, reutilizáveis e fáceis de manter.

Residência em Software

Exemplo

```cpp
#include <iostream>

// Classe abstrata para representar uma forma geométrica
class Forma {
public:
    // Método virtual puro para calcular a área da forma
    virtual double calcularArea() const = 0;

    // Método virtual para exibir informações sobre a forma
    virtual void mostrarInfo() const {
        std::cout << "Forma genérica" << std::endl;
    }

    // Destrutor virtual
    virtual ~Forma() {
    }
};
```

Residência em Software

Exemplo

```
// Classe derivada para representar um círculo
class Circulo : public Forma {
private:
    double raio;

public:
    Circulo(double r) : raio(r) {}

    // Implementação do método para calcular a área do círculo
    double calcularArea() const override {
        return 3.141592653589793 * raio * raio;
    }
```

Residência em Software

Exemplo

```
    // Implementação do método para mostrar informações específicas do círculo
    void mostrarInfo() const override {
        std::cout << "Círculo com raio: " << raio << std::endl;
    }
};
```

Residência em Software

Exemplo

```
// Classe derivada para representar um retângulo
class Retangulo : public Forma {
private:
    double largura;
    double altura;

public:
    Retangulo(double w, double h) : largura(w), altura(h) {}

    // Implementação do método para calcular a área do retângulo
    double calcularArea() const override {
        return largura * altura;
    }
```

Residência em Software

Exemplo

```
    // Implementação do método para mostrar informações específicas do retângulo
    void mostrarInfo() const override {
        std::cout << "Retângulo com largura: " << largura << " e altura: "
                  << altura << std::endl;
    }
};
```

Residência em Software

Exemplo

```
int main() {
    // Exemplo de uso de classes abstratas
    Forma* forma1 = new Circulo(5.0);
    Forma* forma2 = new Retangulo(4.0, 6.0);
    // Chamada de métodos polimórficos
    forma1->mostrarInfo();
    std::cout << "Área: " << forma1->calcularArea() << std::endl;
    forma2->mostrarInfo();
    std::cout << "Área: " << forma2->calcularArea() << std::endl;
    // Libera a memória
    delete forma1;
    delete forma2;
    return 0;
}
```

Residência em Software

Exercício

1. Um banco trabalha com três tipos de contas:
   - ★ conta corrente comum;
   - ★ conta corrente com limite;
   - ★ conta poupança.

- Em todos os casos é necessário guardar o **número da conta**, o n**ome do correntista** e o **saldo**.
- Para a conta poupança é necessário guardar o **dia do aniversário** da conta (quando são creditados os juros).
- Já para a conta com limite é necessário guardar o **valor do limite**.

Residência em Software

Exercício

- As contas também armazenam uma **lista de transações**.
- Uma transação é definida por uma **data**, **valor da transação** e **descrição**. Se o valor for negativo, a transação é considerada um **débito** (**crédito** caso contrário).
- As operações possíveis são: **depósito**, **retirada** e **impressão de extrato**. Essas operações devem ser definidas numa **classe abstrata pura** (interface) denominada **Conta**.
- A **operação de depósito** é igual nos três tipos de conta.
- A **retirada** só é diferente na conta com limite, pois esta admite que o saldo fique negativo até o limite estabelecido.

Residência em Software

Exercício

- Finalmente o **extrato** é diferente para as três:
  - **na conta comum** exibe o número da conta, nome do cliente, transações e o saldo;
  - **na conta limite** imprime também o valor do limite;
  - **na conta poupança** imprime também o dia do aniversário.

- Implemente a hierarquia de classes das contas explorando polimorfismo.

Residência em Software

Exercício

2. Faça um programa em C++ que permita ao usuário fazer **depósitos**, **retiradas** e **verificação de extrato** nas suas contas a partir do número da conta.

Utilize uma única coleção (vector) para armazenar todos os tipos de contas.