

Classes em C++

Classes Derivadas em C++

Introdução

Classes derivadas em C++ permitem criar novas classes baseadas em classes existentes. Elas herdam os membros (atributos e métodos) da classe base e podem adicionar novos membros ou redefinir os existentes. Este conceito é fundamental para a programação orientada a objetos, permitindo reutilização de código e polimorfismo.

1. Definição e Sintaxe

- Definição: Uma classe derivada é uma classe que herda de outra classe, chamada de classe base.

- Sintaxe:

```
class ClasseBase {  
  
    public:  
  
        // Membros da classe base  
  
};  
  
class ClasseDerivada : public ClasseBase {  
  
    public:  
  
        // Membros da classe derivada  
  
};
```

Classes em C++

2. Herança e Acesso aos Membros

- Acesso: Membros públicos e protegidos da classe base são acessíveis na classe derivada.

Membros privados não são acessíveis diretamente.

- Exemplo:

```
class Base {  
  
    public:  
  
        int publico;  
  
    protected:  
  
        int protegido;  
  
    private:  
  
        int privado;  
  
};  
  
class Derivada : public Base {  
  
    public:  
  
        void mostrar() {  
  
            publico = 10;    // Acessível  
  
            protegido = 20;  // Acessível  
  
            // privado = 30; // Não acessível  
  
        }  
  
};
```

3. Construtores e Destrutores

Classes em C++

- Definição: Construtores da classe derivada devem chamar explicitamente o construtor da classe base.

- Exemplo:

```
class Base {  
  
public:  
  
    Base(int v) : valor(v) {}  
  
private:  
  
    int valor;  
  
};  
  
class Derivada : public Base {  
  
public:  
  
    Derivada(int v) : Base(v) {}  
  
};
```

4. Sobrecarga de Funções

- Definição: Classes derivadas podem redefinir funções membros da classe base.

- Exemplo:

```
class Base {  
  
public:  
  
    virtual void mostrar() {  
  
        std::cout << "Base" << std::endl;  
  
    }  
  
};
```

Classes em C++

```
class Derivada : public Base {  
  
public:  
  
    void mostrar() override {  
  
        std::cout << "Derivada" << std::endl;  
  
    }  
  
};
```

```
int main() {  
  
    Base* b = new Derivada();  
  
    b->mostrar(); // Chamará Derivada::mostrar  
  
    delete b;  
  
    return 0;  
  
}
```

5. Polimorfismo

- Definição: O polimorfismo permite que objetos de classes derivadas sejam tratados como objetos de classes base.

- Exemplo:

```
class Base {  
  
public:  
  
    virtual void desenhar() {  
  
        std::cout << "Desenhar base" << std::endl;  
  
    }  
  
}
```

Classes em C++

```
};
```

```
class Circulo : public Base {
```

```
public:
```

```
    void desenhar() override {
```

```
        std::cout << "Desenhar círculo" << std::endl;
```

```
    }
```

```
};
```

```
class Quadrado : public Base {
```

```
public:
```

```
    void desenhar() override {
```

```
        std::cout << "Desenhar quadrado" << std::endl;
```

```
    }
```

```
};
```

```
int main() {
```

```
    Base* formas[] = {new Circulo(), new Quadrado()};
```

```
    for (Base* forma : formas) {
```

```
        forma->desenhar();
```

```
    }
```

```
    for (Base* forma : formas) {
```

```
        delete forma;
```

```
    }
```

```
    return 0;
```

Classes em C++

```
}
```

6. Acesso Específico da Classe Base

- Definição: A classe derivada pode acessar explicitamente membros da classe base usando o operador de resolução de escopo `::`.

- Exemplo:

```
class Base {  
  
public:  
  
    void mostrar() {  
  
        std::cout << "Base" << std::endl;  
  
    }  
  
};  
  
class Derivada : public Base {  
  
public:  
  
    void mostrar() {  
  
        Base::mostrar(); // Chama Base::mostrar  
  
        std::cout << "Derivada" << std::endl;  
  
    }  
  
};  
  
int main() {  
  
    Derivada d;  
  
    d.mostrar();  
  
}
```

Classes em C++

```
    return 0;  
}
```

7. Classes Derivadas e Acesso Protegido

- Definição: Membros protegidos da classe base são acessíveis na classe derivada.

- Exemplo:

```
class Base {  
  
protected:  
  
    int protegido;  
  
};  
  
class Derivada : public Base {  
  
public:  
  
    void funcao() {  
  
        protegido = 10; // Acessível  
  
    }  
  
};
```

Dicas de Boas Práticas

- Encapsulamento: Mantenha a visibilidade adequada dos membros para proteger o estado da classe.

- Reutilização: Use herança para reutilizar código, mas evite herança múltipla para minimizar a complexidade.

Classes em C++

- Polimorfismo: Aproveite o polimorfismo para escrever código mais flexível e reutilizável.

Esta seção abrange os conceitos sobre classes derivadas em C++. Para mais detalhes, consulte a documentação oficial: https://en.cppreference.com/w/cpp/language/derived_classes