>>> Programação Orientada a Objetos (POO)

... Polimorfismo

Prof: André de Freitas Smaira

[~]$ _

>>> Herança e Ponteiros

* **Relembrando**: objeto da classe derivada **é-um** objeto da classe base
* **Ponteiro** para **classe base pode apontar** para objeto de **classe derivada**
* **Chamada de método através de ponteiro** para classe base faz sentido para **objeto de classe derivada**
* Problema: método é alterado na classe derivada e **ponteiro da classe base chama método da classe base**

>>> Exemplo

```
class Contador {
    int _n;
public:
    Contador() : n(0) {}
    void anda() { _n++; }
    int valor() { return _n; }
};
class Pulador : public Contador {
public:
    void anda() {
        Contador::anda();
        Contador::anda();
    }
};
int main() {
    Contador *vc[3];
    vc[0] = new Contador();
    vc[1] = new Pulador();
    vc[2] = new Contador();
    for (int i = 0; i < 3; i++)
        vc[i]->anda();
    return 0;
}
```

~]$ _

>>> Polimorfismo

* Queremos que o **método chamado seja do objeto apontado**, não do ponteiro
* **Método virtual** na classe base.
* Esse método é então denominado **polimórfico**

>>> Exemplo

```
class Contador {
    int _n;
public:
    Contador() : n(0) {}
    virtual void anda() { _n++; }
    int valor() { return _n; }
};
class Pulador : public Contador {
public:
    void anda() {
        Contador::anda();
        Contador::anda();
    }
};
int main() {
    Contador *vc[3];
    vc[0] = new Contador();
    vc[1] = new Pulador();
    vc[2] = new Contador();
    for (int i = 0; i < 3; i++)
        vc[i]->anda();
    return 0;
}
```

~]$ _

>>> Método Virtual

* O atributo **virtual** se **propaga para todas as derivadas** na hierarquia
* Se classe derivada **não redefine** método, usa da **classe base**
* Métodos virtuais devem ter a **mesma assinatura em toda a hierarquia**
* Polimorfismo funciona para **ponteiros e referências**
* Em algumas situações, um método **deve existir** nas classes derivadas, mas **não sabemos implementar** para a classe base
* Método **puramente virtual** com assinatura e **= 0** ao invés de implementação
* => **Classe base abstrata**

[~]$ _

>>> Exemplo

```
class Figura {
public:
    ...
    virtual void desenha() = 0;
    ...
};
```

[~]$ _

>>> Destruidores Virtuais

* Ao liberar o objeto (delete) é importante que o destruidor correto seja chamado
* Se a classe tiver um método virtual, o destruidor será virtual (se existir)
* O atributo virtual se propaga entre os destruidores, mesmo eles tendo nomes diferentes

[~]$ _

>>> Frame Title

* Podemos executar funções através de **ponteiros para funções**
* Podemos fazer o **mesmo para métodos**
* A **sintaxe é distinta**, pois as situações são diferentes:
  * Escopo do método é na classe
  * Método precisa de um objeto
  * Método pode ser polimórfico
* O método precisa estar **atrelado a uma classe**

>>> Exemplo

```
int main() {
    Exemplo obj;

    void (Exemplo::*ptrMetodo1)() = &Exemplo::metodo1;
    void (Exemplo::*ptrMetodo2)(int) = &Exemplo::metodo2;

    (obj.*ptrMetodo1)();  // Chamando metodo1
    (obj.*ptrMetodo2)(42);  // Chamando metodo2 com o argumento

    return 0;
}
```

[~]$ _