IMD0030
LINGUAGEM DE PROGRAMAÇÃO I

Aula 17 – Classes Abstratas

IMD INSTITUTO METRÓPOLE DIGITAL
UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Objetivos desta aula

- Introduzir o conceito de classe abstrata na linguagem de programação C++
- Para isso, estudaremos:
  - O conceito de método virtual
  - O conceito de classe abstrata
  - O conceito de interface
  - Como implementar métodos virtuais e classes abstratas
- Ao final da aula, espera-se que o aluno seja capaz de:
  - Compreender os conceitos de método virtual e classe abstrata
  - Implementar classes abstratas na linguagem de programação C++

Métodos virtuais

- Um **método virtual** é um método de uma classe base que pode ser **redefinido** pelas suas classes derivadas, porém **ainda pode ser acessado** por um objeto da classe base
  - Este é um **verdadeiro polimorfismo**
  - Isso é possível devido a um mecanismo de **conversão de ponteiros**, possibilitando um objeto de uma classe derivada ser convertido em um objeto da classe base
    - Isso parte do princípio de que, se uma classe *B* é derivada de uma classe *A*, um objeto de *B* **também é um** objeto de *A* devido à herança

Métodos virtuais

- **Métodos não virtuais** de uma classe base também podem ser redefinidos pelas classes derivadas, porém **não podem ser acessados** a partir de objetos da classe base
  - A última definição (na classe derivada) sobrescreve a anterior (na classe base)
  - Se um método virtual não for redefinido, o método correspondente da classe base será utilizado
- A definição de métodos virtuais é feita utilizando da palavra-chave `virtual` **antes** do tipo de retorno do método
  - **Não é necessário adicionar a palavra-chave à assinatura dos métodos das classes derivadas**

Métodos virtuais

```
class Poligono {
   protected:
      double largura;
      double altura;
   public:
      Poligono(double a, double h) : largura(a), altura(h) {}
      virtual double area() { return 0; }
};

class Retangulo : public Poligono {
   public:
      Retangulo(double a, double h) : Poligono(a, h) {}
      double area() { return largura * altura; }
};

class Triangulo : public Poligono {
   public:
      Triangulo(double a, double h) : Poligono(a, h) {}
      double area() { return (largura * altura / 2); }
};
```

Redefinição do método `area` definido na classe base `Poligono` pelas classes derivadas `Retangulo` e `Triangulo`

Métodos virtuais

```
#include <iostream>
using std::cout;
using std::endl;

int main() {
    Poligono* r = new Retangulo(1, 2);
    Poligono* t = new Triangulo(3, 4);
    Poligono* p = new Poligono(2, 1);

    cout << "Area do retangulo: " << r->area() << endl;
    cout << "Area do triangulo: " << t->area() << endl;
    cout << "Area do poligono: "  << p->area() << endl;

    return 0;
}
```

Resultado da execução:

```
$ ./poligono
Area do retangulo: 2
Area do triangulo: 6
Area do polígono: 0
```

A invocação a `area` de `Poligono` é possível pois esse método foi definido como virtual nessa classe base

Métodos virtuais

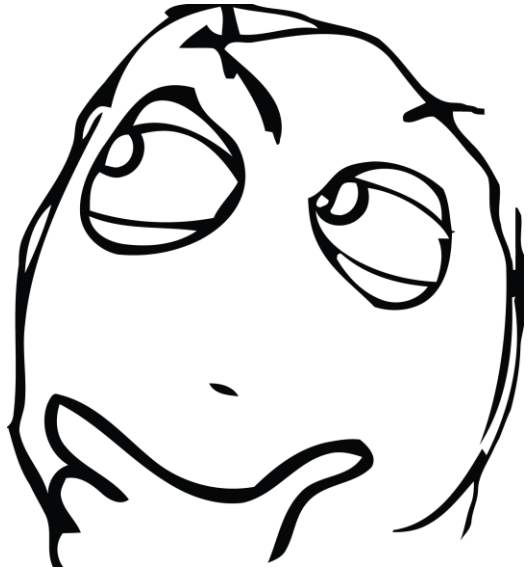
- Internamente, o compilador cria uma **tabela de métodos virtuais** (TMV), que é um **vetor de ponteiros de função** para armazenar referências aos métodos da classe
  - É criada uma TMV para cada classe contendo métodos virtuais, compartilhada por todos os objetos dessa classe
  - **A TVM não é acessível diretamente via código**
- O número de entradas na TMV é igual ao número de métodos virtuais da classe e cada posição armazena um ponteiro de função para um desses métodos
- Quando um objeto da classe é instanciado, um **ponteiro para a TMV** (*vpointer* ou VPTR) é adicionado como um membro oculto dessa classe

Mas e se...

... quiséssemos **impedir** a instanciação de objetos da classe base `Poligono`, para forçar um polígono a ter uma forma predefinida?

- Com o que temos até agora, é plenamente possível instanciar um objeto da classe `Poligono` e invocar seus métodos, uma vez que ela é uma classe convencional, como qualquer outra

Resposta:
Definindo toda a classe `Poligono` como uma
classe abstrata



Classes abstratas

- Uma **classe abstrata** é uma classe que apresenta **pelo menos um método virtual puro**, um método que possui apenas a definição de sua assinatura e não possui implementação
  - **Não é possível instanciar objetos de uma classe abstrata**, que serve simplesmente de base para a definição de outras classes
  - É necessária a definição de uma classe derivada a partir da classe base implementando **todos** os métodos que foram definidos na classe abstrata como virtuais puros
  - Os métodos virtuais puros ainda podem ser utilizados por outros métodos da classe abstrata, mesmo sem possuir implementação
- Define-se um método virtual puro através da palavra-chave `virtual` e atribuindo-lhe o valor zero
  - **Não é necessário adicionar a palavra-chave à assinatura dos métodos das classes derivadas**

Classes abstratas

```
class Poligono {
   protected:
      double largura;
      double altura;
   public:
      Poligono(double a, double h) : largura(a), altura(h) {}
      virtual double area() = 0;
      void printArea() { cout << this->area() << endl; }
};

class Retangulo : public Poligono {
   public:
      Retangulo(double a, double h) : Poligono(a, h) {}
      double area() { return largura * altura; }
};

class Triangulo : public Poligono {
   public:
      Triangulo(double a, double h) : Poligono(a, h) {}
      double area() { return (largura * altura / 2); }
};
```

O método `area` definido na classe base `Poligono` é virtual puro e, portanto, não possui implementação. Contudo, pode ser utilizado pelo método `printArea`

Implementação do método `area` definido na classe base `Poligono` pelas classes derivadas `Retangulo` e `Triangulo`

Classes abstratas

```
#include <iostream>
using std::cout;

int main() {
    Poligono* r = new Retangulo(1, 2);
    Poligono* t = new Triangulo(3, 4);
    Poligono* p = new Poligono(2, 1);

    cout << "Area do retangulo: ";
    r->printArea();

    cout << "Area do triangulo: ";
    t->printArea();

    cout << "Area do poligono: ";
    p->printArea();

    return 0;
}
```

A compilação deste código resulta em erro porque a classe `Poligono` agora é abstrata

```
In function 'int main()':
error: invalid new-expression of abstract class
type 'Poligono'
note: because the following virtual functions are
pure within 'Poligono':
note: virtual double Poligono::area()
```

Classes abstratas

```
#include <iostream>
using std::cout;

int main() {
    Poligono* r = new Retangulo(1, 2);
    Poligono* t = new Triangulo(3, 4);

    cout << "Area do retangulo: ";
    r->printArea();

    cout << "Area do triangulo: ";
    t->printArea();

    return 0;
}
```

Resultado da execução:

```
$ ./poligono
Area do retangulo: 2
Area do triangulo: 6
```

Interfaces

- Objetos de uma classe definem sua interação com o mundo exterior através do conjunto de métodos que eles expõem, a sua **interface**
- Uma interface é um conjunto de **métodos com corpo vazio**
  - A interface define apenas **o que** o objeto expõe, sem se ater a detalhes de **como** realizar a implementação
- Diz-se que uma classe **implementa** uma interface quando todos os métodos declarados nessa interface são definidos na classe
  - A interface obriga a classe a cumprir o que nela foi declarado, como uma espécie de contrato
  - Caso a classe não implemente corretamente a interface e seus métodos, o compilador atesta um erro

Interfaces

- Interfaces
  - **não podem ser instanciadas**, ou seja, não é possível criar objetos com elas
  - **definem apenas assinaturas de métodos**, todos com visibilidade pública e sem corpo
  - **não podem possuir métodos concretos** (com corpo) nem métodos estáticos
  - **não podem conter variáveis como atributos ou membros estáticos**
  - podem conter declarações de constantes
  - podem ser criadas a partir de uma interface já existente, via herança
- **A linguagem C++ não possui um conceito explícito de interface**, como ocorre na linguagem Java
  - Java também faz uso do conceito de interface porque a linguagem não dispõe de herança múltipla
    - A interface torna-se necessária uma vez que uma classe pode implementar múltiplas interfaces
  - Interfaces em C++ são essencialmente **classes abstratas em que todos os métodos são virtuais puros**

Interfaces

```
#include <iostream>
using std::cout;
using std::endl;

class Poligono {
   public:
      virtual void print() = 0;
      virtual double area() = 0;
};

class Retangulo : public Poligono {
   protected:
      double largura;
      double altura;
   public:
      Retangulo(double l, double h) : largura(l), altura(a) {}
      void print() { cout << "Retangulo: l = " << largura << ", a = " << altura << endl; }
      double area() { return largura * altura; }
};
```

A interface `Poligono` declara todos os métodos que deverão obrigatoriamente ser definidos pelas classes que a implementarem

Interfaces

```
#include <iostream>
using std::cout;
using std::endl;

int main() {
    Poligono* r = new Retangulo(1, 2);
    r->print();

    cout << "Area do retangulo: " << r->area() << endl;
    return 0;
}
```

Resultado da execução:

```
$ ./poligono
Retangulo: l = 1, a = 2
Area do retangulo: 2
```

Alguma Questão?