Programação Orientada a Objetos

Aula V – Herança II

Polimorfismo, funções virtuais e classes abstratas

Prof.: Bruno E. G. Gomes
IFRN

Objetivos

- Criação dinâmica de objetos com o operador **new**
- Conceitos:
  - métodos virtuais
  - classes abstratas
- Criar objetos utilizando-se do recurso de polimorfismo

Operador *new*

- Permite a criação dinâmica de memória para armazenar objetos.
  - Após o uso, pode-se explicitamente retirar o objeto da memória (**delete** *nome_objeto*)
- Irá permitir utilizar o recurso de polimorfismo
- Criando objetos com **new**:
  - *<Tipo> *<nome> =* ***new*** *<Tipo>(<parâmetros_construtor>);*
- *Exemplos*

```
Carro *cr = new Carro();
Triangulo *triangulo = new Triangulo(2, 3);
```

Operador new

- **new** :
  1) reserva uma área na memória para o objeto;
  2) cria o objeto;
  3) Retorna a referência ao objeto (endereço na memória) para a variável

- Por exemplo:

  *Carro *cr* = ***new*** *Carro*();
  - **cr** é um apontador para um objeto do tipo *Carro*.

- Para acessar um membro do objeto, usa-se “->” (*menos* seguido de *maior*) ao invés de “.” (ponto)
  - cr->acelerar(50.0)
  - cr->parar(10.3)

Objetos criados com new

- Torna possível que um apontador do tipo de uma classe base (mãe) seja usada para apontar para objetos das classes filhas
  - É a base do que chamamos de polimorfismo

- Exemplo:
  - Um “Funcionário” pode ser:
    - Diretor acadêmico, diretor geral, técnico administrativo ou professor.
  - Então, podemos fazer:

```
Funcionario *f = new Professor (“João”, “383292”);
Funcionario *f2 =
    new TecnicoAdministrativo (“Francisco”, “177299”);
```

Objetos criados com new (continuação)

- Permite uma maior flexibilidade na criação de objetos
  - No exemplo, é possível substituir a referência "Funcionário" por qualquer subtipo de funcionário

- Exemplo:

```
//Criou-se um novo funcionário, que é um diretor geral
Funcionario *fn = new DiretorGeral("Rady", "383292");

//"fn" agora aponta para um objeto diretor acadêmico
fn = new DiretorAcademico("Ramon", "242424");

//"fn" agora aponta para um objeto do tipo professor
fn = new Professor("Elionardo Rochelly", "1712828");
```

Chamada de métodos com apontador

- Se um mesmo *método* estiver definido na superclasse e em suas subclasses, o método chamado através do apontador será sempre o da superclasse
  - Mesmo que esteja apontando para uma classe base.

- Exemplo:
  - Se em *Funcionário* e nas suas subclasses, estiver declarado um método *calcularSalario*, que retorne o salário líquido do funcionário, o código:

```
Funcionario prof = new Professor("Maria", "1111");
//irá chamar calcular salário de Funcionario, e não de professor.
prof->calcularSalario();
```

- Para resolver isso, declare métodos que devem ser *redefinidos* na subclasse como **virtuais**

Método virtual

- Um método **virtual** é aquele que deve ser *redefinido* nas subclasses
  - Deve ser feito quando as subclasses tem diferentes implementações do mesmo método
  - Assim, ao apontar para uma subclasse, através de uma referência da classe base, o método correto da subclasse será chamado
- Deve-se colocar a palavra **virtual** antes do nome método na classe mãe
  - de preferência, mas não obrigatório, o nome virtual deve ser colocado também nos métodos das classes filhas

Exemplo: classe Retângulo

→ O método área deve ser implementado de forma diferente nas subclasses

→ Anteriormente, o que fizemos foi declarar cada versão do método na subclasse. No entanto, o método não aparecia na classe mãe (Polígono)

```
class Poligono {
protected:
    int base, altura;
public:
    Poligono(int base, int alt);
};
```

```
class Retangulo : public Poligono {
public:
    Retangulo(int base, int alt);
    int area();
};

class Triangulo : public Poligono{
public:
    Triangulo(int base, int alt);
    int area();
};
```

Classe Retangulo: método área como virtual

→ Declarando o método como virtual força que as subclasses o sobrescrevam

→ Permite também utilizar o polimorfismo (um apontador do tipo *Poligono* ser utilizado para apontar para classes *Retangulo* e Triangulo)

```
class Poligono {
protected:
    int base, altura;
    virtual int area();
public:
    Poligono(int base, int alt);
};
```

```
class Retangulo : public Poligono {
public:
    Retangulo(int base, int alt);
    virtual int area();
};

class Triangulo : public Poligono{
public:
    Triangulo(int base, int alt);
    virtual int area();
};
```

Classe Abstrata

- Uma classe é abstrata quanto possui ao menos 1 método virtual puro
  - aquele que não tem implementação na classe, apenas em subclasses;
  - Para declarar um método como virtual puro coloque “= 0” depois do parênteses que fecha os parâmetros

- Um Polígono é por definição uma classe abstrata
  - não criamos e trabalhamos com o conceito “polígono”, mas com triângulos, quadrados, hexágonos, etc.
  - Ou seja, polígono é um conceito abstrato, e não concreto

```
class Poligono {
protected:
    int base, altura;
    virtual int area() = 0;
public:
    Poligono(int base, int alt);
};
```

Exercício

- Crie as classes correspondentes para o exemplo Funcionario. Um funcionário deve ter um nome, uma *matrícula* e um *salário*
  1) Crie um construtor que receba nome e matrícula
  2) Crie métodos para inserir (set) e obter (get) o nome, a matrícula e o salário de um funcionário
  3) Crie um método *calcularSalario*, que retorne um salário diferente a depender do funcionário (deve ser redefinido nas subclases)
     a) O método salário da superclasse deve ser virtual puro. Ou seja, a superclasse (Funcionario) é abstrata
  4) No método main, crie objetos do tipo Funcionário (com o operador new), apontanto para diversos tipos de funcionário