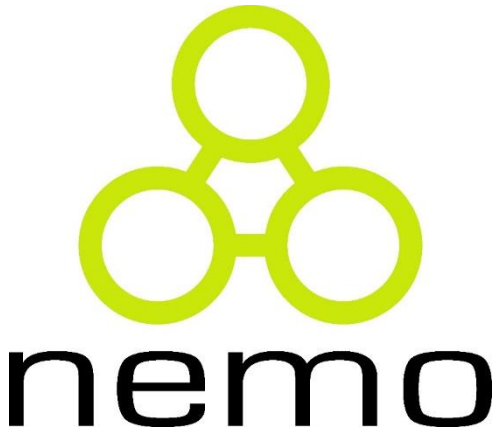


Programação III

Jordana S. Salamon

jssalamon@inf.ufes.br

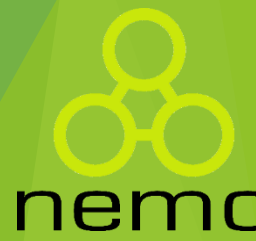
jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Classes Abstratas



Classes Abstratas

- Uma interface descreve o comportamento ou capacidade de um classe sem o comprometimento de uma implementação específica.
- As interfaces são implementadas usando **classes abstratas**.
- Uma classe se torna abstrata quando declaramos ao menos uma de suas funções como **função virtual pura**.

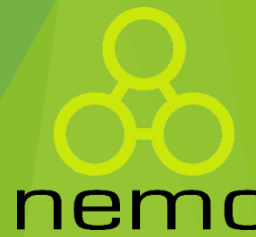
```
virtual float area() = 0;
```

- O proposito de uma **classe abstrata** é prover uma classe base apropriada para que outras classes possam herdar.

nemo

Classes Abstratas

- Classes abstratas não podem ser utilizadas para instanciar objetos e servem apenas como um interface.
- Tentar instanciar um objeto de uma classe abstrata causa um erro de compilação.
- Dessa forma, se uma classe derivada de uma classe precisa ser instanciada, ela precisa implementar cada uma das funções virtuais.
- Caso as funções virtuais puras não sejam sobrescritas na classe derivada, então no momento que os objetos da classe derivada forem instanciados, gera um erro de compilação.
- Classes derivadas que podem ser utilizadas para instanciar objetos são chamados de **classes concretas**.



Classes Abstratas x Classes Concretas

Classes Abstratas →

Método Abstrato →

```
class FormaGeometrica{
    protected:
        float altura;
        float largura;

    public:
        virtual float area() = 0;
};
```

Classes Concretas

```
class Retangulo: public FormaGeometrica{
    public:
        Retangulo(int _altura, int _largura){
            this->altura = _altura;
            this->largura = _largura;
        }

        float area (){
            cout << "Calculando a area do Retangulo" << endl;
            return this->altura * this->largura;
        }
};
```

```
class Triangulo: public FormaGeometrica{
    public:
        Triangulo(int _altura, int _largura){
            this->altura = _altura;
            this->largura = _largura;
        }

        float area (){
            cout << "Calculando a area do Triangulo" << endl;
            return (this->altura * this->largura)/2;
        }
};
```

Métodos Concretos

Diferença entre Tipos de Sobrescritas – Sobrescrita Simples

```
class FormaGeometrica{
    protected:
        float altura;
        float largura;

    public:
        float area(){
            cout << "Calculando a area da Forma Geometrica" << endl;
            return 0;
        }
};

class Retangulo: public FormaGeometrica{
    public:
        Retangulo(int _altura, int _largura){
            this->altura = _altura;
            this->largura = _largura;
        }

        float area (){
            cout << "Calculando a area do Retangulo" << endl;
            return this->altura * this->largura;
        }
};
```

```
int main(){
    FormaGeometrica *fg;
    Retangulo *r;
    float area;

    fg = new FormaGeometrica();
    area = fg->area(); //Valor area: 0
    cout << "Valor area: " << area << endl;

    fg = new Retangulo(10,7);
    area = fg->area(); //Valor area: 0
    cout << "Valor area: " << area << endl;

    r = new Retangulo(10,7);
    area = r->area(); //Valor area: 70
    cout << "Valor area: " << area << endl;

    return 0;
}
```

nemo

Diferença entre Tipos de Sobrescritas – Sobrescrita Virtual

```
class FormaGeometrica{
    protected:
        float altura;
        float largura;

    public:
        virtual float area(){
            cout << "Calculando a area da Forma Geometrica" << endl;
            return 0;
        }
};

class Retangulo: public FormaGeometrica{
    public:
        Retangulo(int _altura, int _largura){
            this->altura = _altura;
            this->largura = _largura;
        }

        float area (){
            cout << "Calculando a area do Retangulo" << endl;
            return this->altura * this->largura;
        }
};
```

```
int main(){
    FormaGeometrica *fg;
    Retangulo *r;
    float area;

    fg = new FormaGeometrica();
    area = fg->area(); //Valor area: 0
    cout << "Valor area: " << area << endl;

    fg = new Retangulo(10,7);
    area = fg->area(); //Valor area: 70
    cout << "Valor area: " << area << endl;

    r = new Retangulo(10,7);
    area = r->area(); //Valor area: 70
    cout << "Valor area: " << area << endl;

    return 0;
}
```

nemo

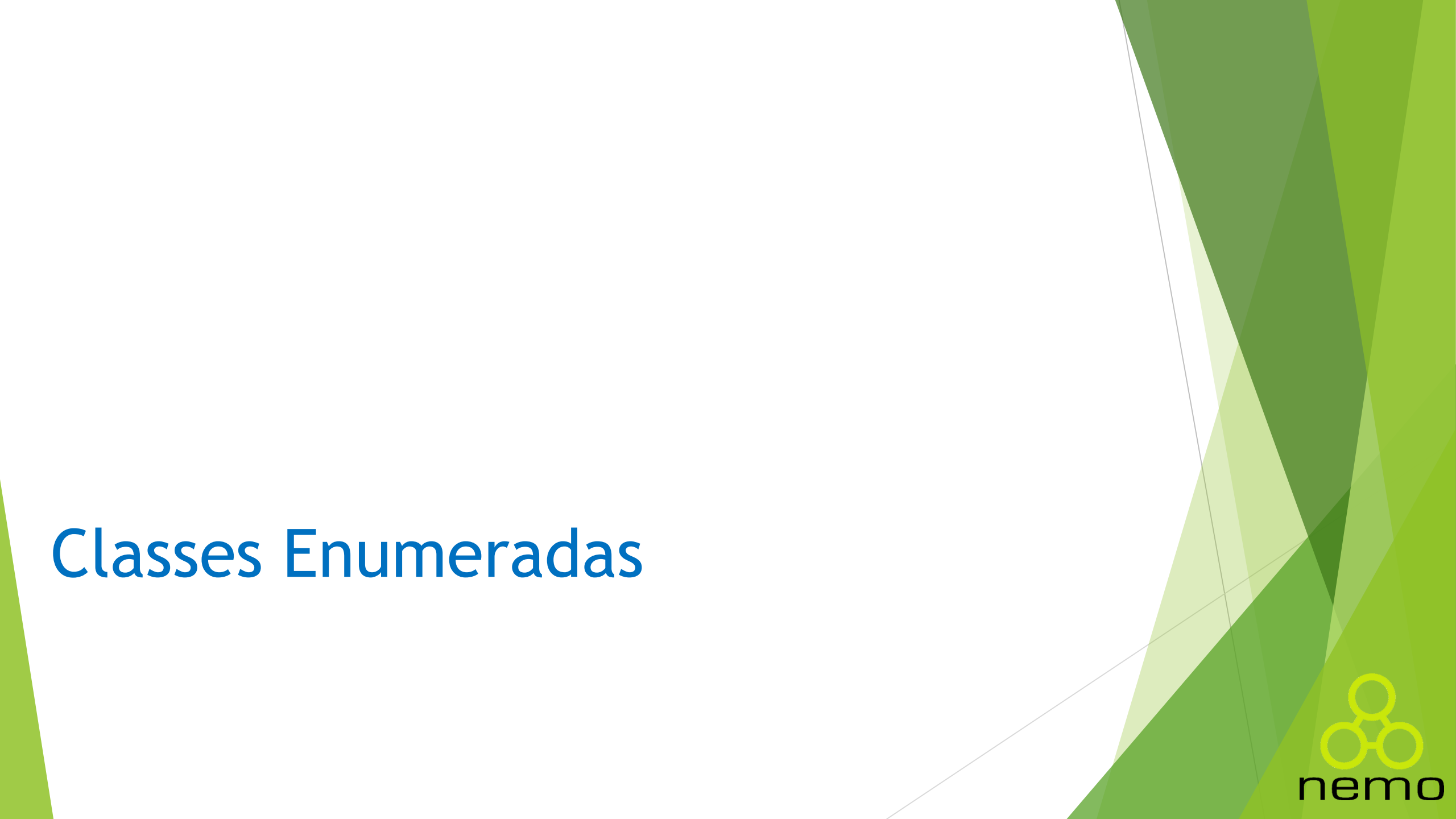
Diferença entre Tipos de Sobrescritas – Sobrescrita Virtual Pura

```
class FormaGeometrica{
    protected:
        float altura;
        float largura;

    public:
        virtual float area() = 0;
};

class Retangulo: public FormaGeometrica{
    public:
        Retangulo(int _altura, int _largura){
            this->altura = _altura;
            this->largura = _largura;
        }

        float area (){
            cout << "Calculando a area do Retangulo" << endl;
            return this->altura * this->largura;
        }
};
```

```
int main(){
    FormaGeometrica *fg;
    Retangulo *r;
    float area;

    fg = new FormaGeometrica(); //Erro de Compilação
    area = fg->area();
    cout << "Valor area: " << area << endl;

    fg = new Retangulo(10,7);
    area = fg->area(); //Valor area: 70
    cout << "Valor area: " << area << endl;

    r = new Retangulo(10,7);
    area = r->area(); //Valor area: 70
    cout << "Valor area: " << area << endl;

    return 0;
}
```

nemo

Classes Enumeradas



Introdução

- Em programação, uma enumeração é um tipo de dado abstrato, cujos valores são atribuídos a exatamente um elemento de um conjunto finito de identificadores escolhidos pelo programador.

- Esse tipo é geralmente usado para variáveis categóricas (como os naipes de um baralho, estado de um país), que não possuem uma ordem numérica definida.

- Em tempo de execução, um tipo de dado enumerado é geralmente implementado usando-se inteiros.

Definido um Enum

- Entretanto, comparando-se a usar somente inteiros, os tipos enumerados tornam o código fonte mais bem documentado que através do uso explícito de "números mágicos".

```
enum Paises
{
        BRASIL,
        ITALIA,
        PORTUGAL,
        ALEMANHA
};
```

```
enum Dias{
     DOMINGO = 1,
     SEGUNDA = 2,
     TERCA = 3,
     QUARTA = 4,
     QUINTA = 5,
     SEXTA = 6,
     SABADO = 7
}
```

- Por default a sequencias dos valores enumerado começa de 0, mas é possível definir um valor especifico para cada um.

nemo

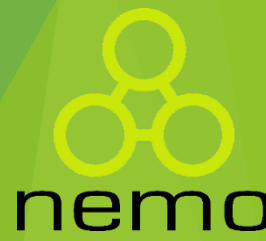
Criando um objeto do tipo enumerado

- Existem duas formas de darmos valor para um objeto de um tipo enumerado:
  - Podemos utilizar o nome de um valor enumerado (ex: PRETO, PRATA, etc.)
  - Podemos utilizar o número dado ao valor enumerado (ex: Cor(4) = PRETO, Cor(2) = PRATA).

```
enum Cor{VERMELHO=1, PRATA=2, BRANCO=3, PRETO=4, AMARELO=5};

int main(){
    Cor c;
    c = PRETO;
    c = Cor(4);
}
```

nemo

Classes Internas em C++



Estrutura Classe Interna

- A linguagem C++ permite que construamos uma classe dentro de outra classe.
- Esse tipo de estrutura permite uma melhor organização do código. Ex:

```
class Veiculo
{
    private:
        int ano;
        string placa;

    class InfoFabricacao
    {
        private:
            string marca;
            string modelo;
    };
};
```

nemo

Instanciação Classe Interna

- Uma vez criado o protótipo da **classe interna** dentro da **classe externa**, dizemos que a **classe interna** está no escopo da **classe externa.**
- Dessa forma para criar um objeto da classe interna é necessário utilizar o operador de escopo (::) para acessar a classe externa, e assim, a classe interna.

```
int main()
{
    Veiculo::InfoFabricacao *info;

    info = new Veiculo::InfoFabricacao("Ford", "Focus");

    return 0;
}
```

nemo

Classes Internas x Classes Externas

- É importante notar que apesar de classes internas estarem dentre da classe externa, a classe interna não tem acesso direto as informações da externa.

```
class Veiculo
{
    private:
        int ano;
        string placa;

    class InfoFabricacao
    {
        private:
            string marca;
            string modelo;
        public:
            InfoFabricacao(string _marca, string _modelo)
            {
                cout << "InfoFabricacao sendo criada" << endl;
                this->marca = _marca; //Classe Interna
                this->modelo = _modelo; //Classe Interna
                this->ano = 2016; //Classe Externa
            }
    };
};
```

```
error: 'class Veiculo::InfoFabricacao' has no member named 'ano'
```

nemo

Definindo Objeto da Classe Interna

- A classe interna deve ser definida antes de criar um atributo na classe externa do tipo da classe interna.

```
class Veiculo
{
    public:
        class InfoFabricacao
        {
            private:
                string marca;
                string modelo;

            public:
                InfoFabricacao(string _marca, string _modelo);
                ~InfoFabricacao();
                void imprimeInfoFabricacao();
        };

    private:
        int ano;
        string placa;
        InfoFabricacao * fabricacao;
};
```

Definindo a classe interna

Criando um atributo

Implementação Externa Método Classe Interna

```
class Veiculo
{
    public:
        class InfoFabricacao
        {
            private:
                string marca;
                string modelo;

            public:
                InfoFabricacao(string _marca, string _modelo);
                ~InfoFabricacao();
                void imprimeInfoFabricacao();
        };

    private:
        int ano;
        string placa;
        InfoFabricacao * fabricacao;
};
```

nemo

Implementação Externa Método Classe Interna

```
Veiculo::InfoFabricacao::InfoFabricacao(string _marca, string _modelo)
{
    cout << "InfoFabricacao sendo criada" << endl;
    this->marca = _marca;
    this->modelo = _modelo;
}

Veiculo::InfoFabricacao::~InfoFabricacao()
{
    cout << "InfoFabricacao sendo destruida" << endl;
}

void Veiculo::InfoFabricacao::imprimeInfoFabricacao()
{
    cout << "Marca = " << this->marca << endl;
    cout << "Modelo = " << this->modelo << endl;
}
```

Escopo

Nome do Método

nemo

Múltiplas Classes Internas

```
class Veiculo
{
    private:
        int ano;
        string placa;
    public:

        class Marca
        {
            private:
                string nome;
            public:
                class Modelo
                {
                    private:
                        string nome;
                    public:
                        Modelo(){
                            cout << "Criando um Modelo" << endl;
                        }
                };
                Modelo *modelo;

                Marca(){
                    cout << "Criando um Marca" << endl;
                    this->modelo = new Veiculo::Marca::Modelo();
                }
        };

        Marca *marca;

        Veiculo()
        {
            cout << "Criando um Veiculo" << endl;
            this->marca = new Veiculo::Marca();
        }
};
```

nemo

Exercício

That’s all Folks!