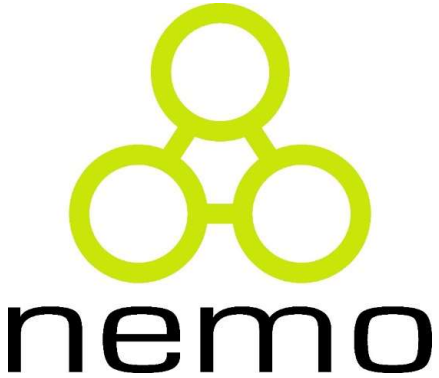


Programação III

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

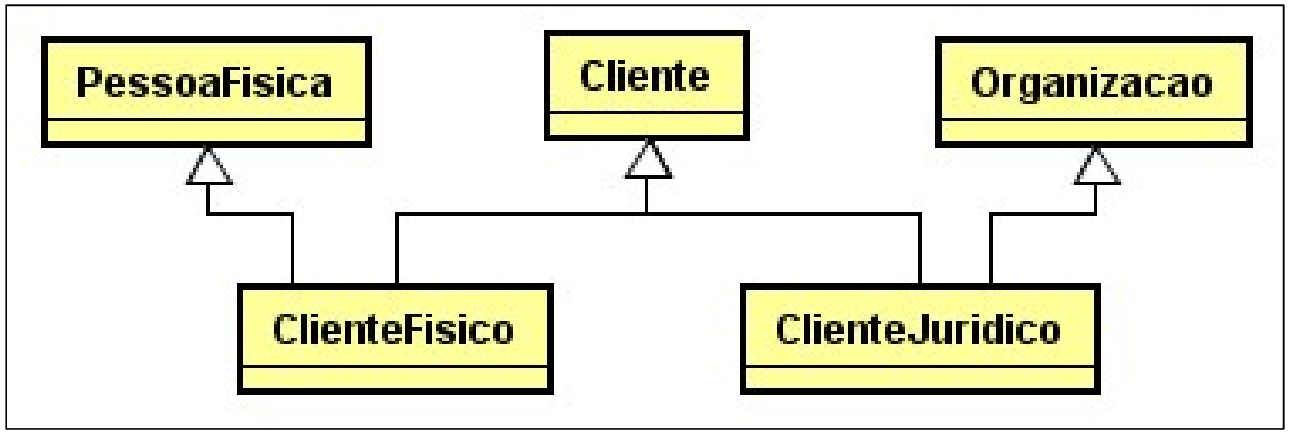
CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Herança Múltipla

Herança Múltipla

- A linguagem C++ permite que classe possam herdar membros de mais de uma classe.



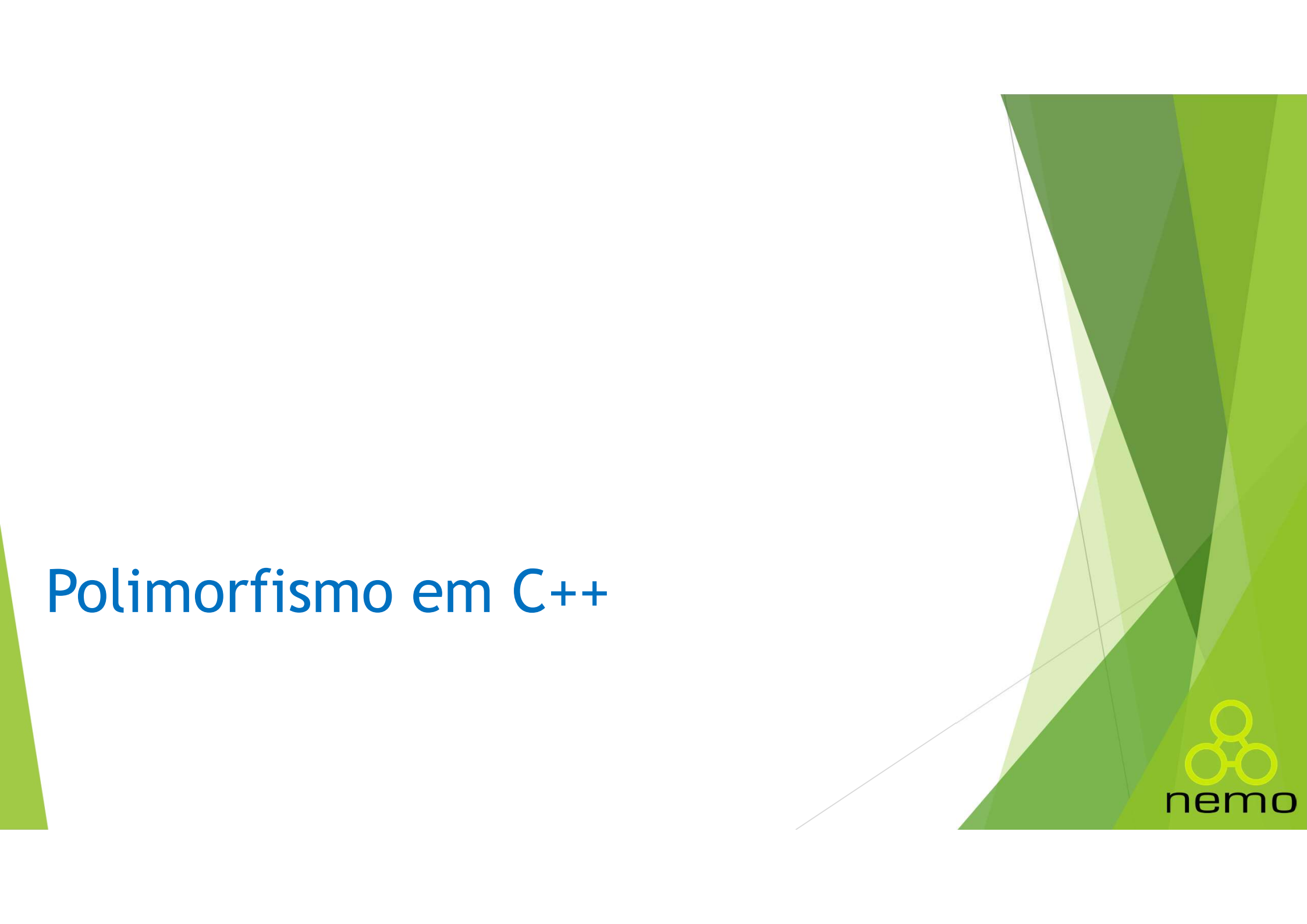
- A sintaxe é a seguinte:

```
class derived-class: access baseA, access baseB....
```

- O tipo de acesso neste caso é único para todas relações de herança.

nemo

Polimorfismo em C++



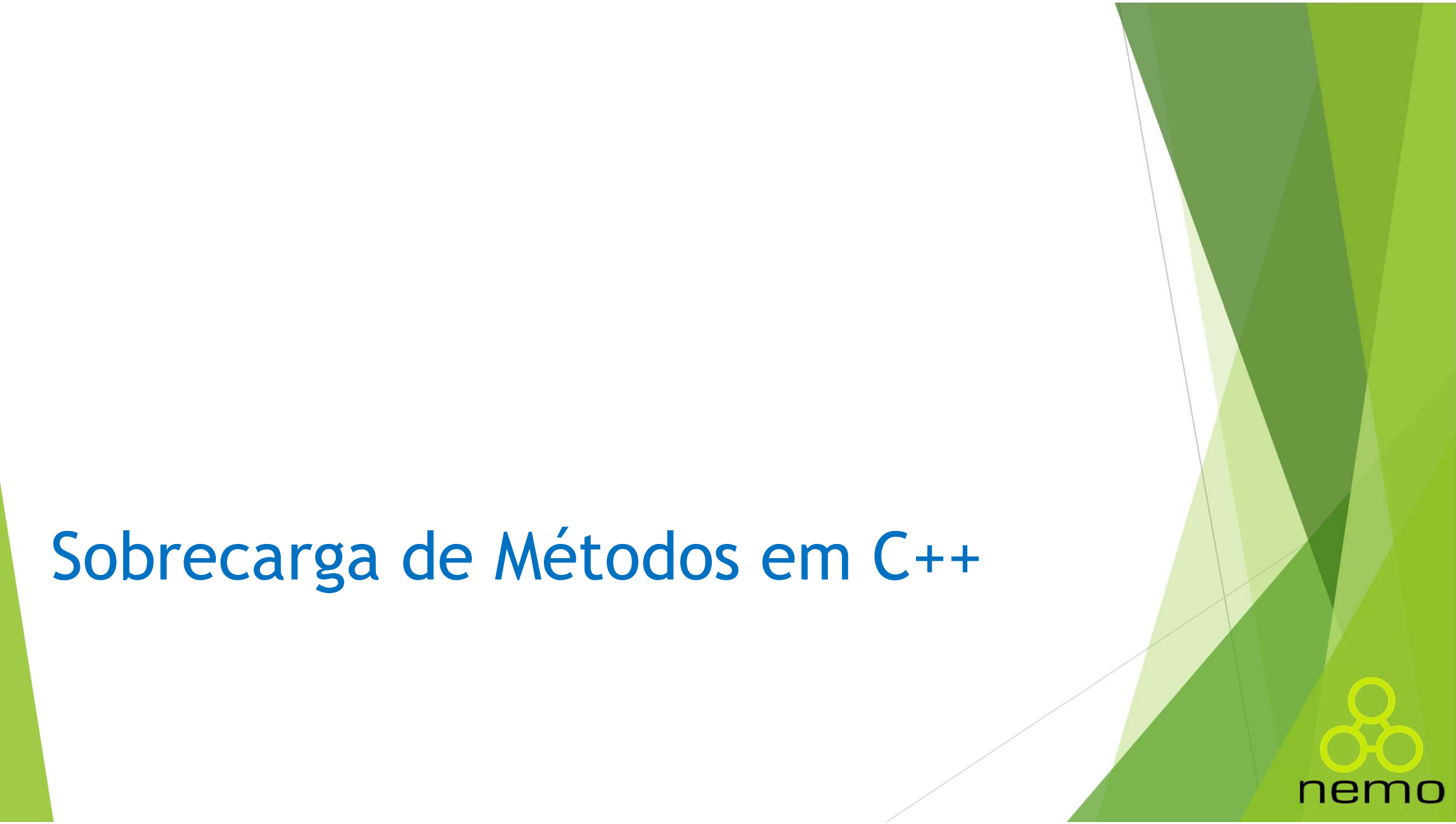
Introdução - Polimorfismo

- A palavra polimorfismo significa possuir várias formas.

- O polimorfismo permite que chamemos métodos de classes derivadas e as funções chamadas tenham um comportamento diferente dependendo do objeto que as chama.

- Ou seja, as mesmas funções são compartilhadas por objetos distintos com implementação lógicas diferentes.

Sobrecarga de Métodos em C++



Polimorfismo – Sobrecarga de Métodos

- Nos permite declarar métodos de mesmo nome mas com parâmetros e implementação diferentes.
- Ex:

```
class Impressora{
    public:
        void imprime(int i){
            cout << "Imprimindo um inteiro: " << i << endl;
        }
        void imprime(double f){
            cout << "Imprimindo um double: " << f << endl;
        }
        void imprime(string s){
            cout << "Imprimindo uma string: " << s << endl;
        }
};
```

```
class Calculadora{
    public:
        int soma(int a, int b){
            return a + b;
        }
        double soma(double a, double b){
            return a + b;
        }
        int subtracao(int a, int b){
            return a - b;
        }
        double subtracao(double a, double b){
            return a - b;
        }
};
```

Polimorfismo – Sobrecarga de Métodos

- Podemos utilizar a sobrecarga para múltiplos construtores.

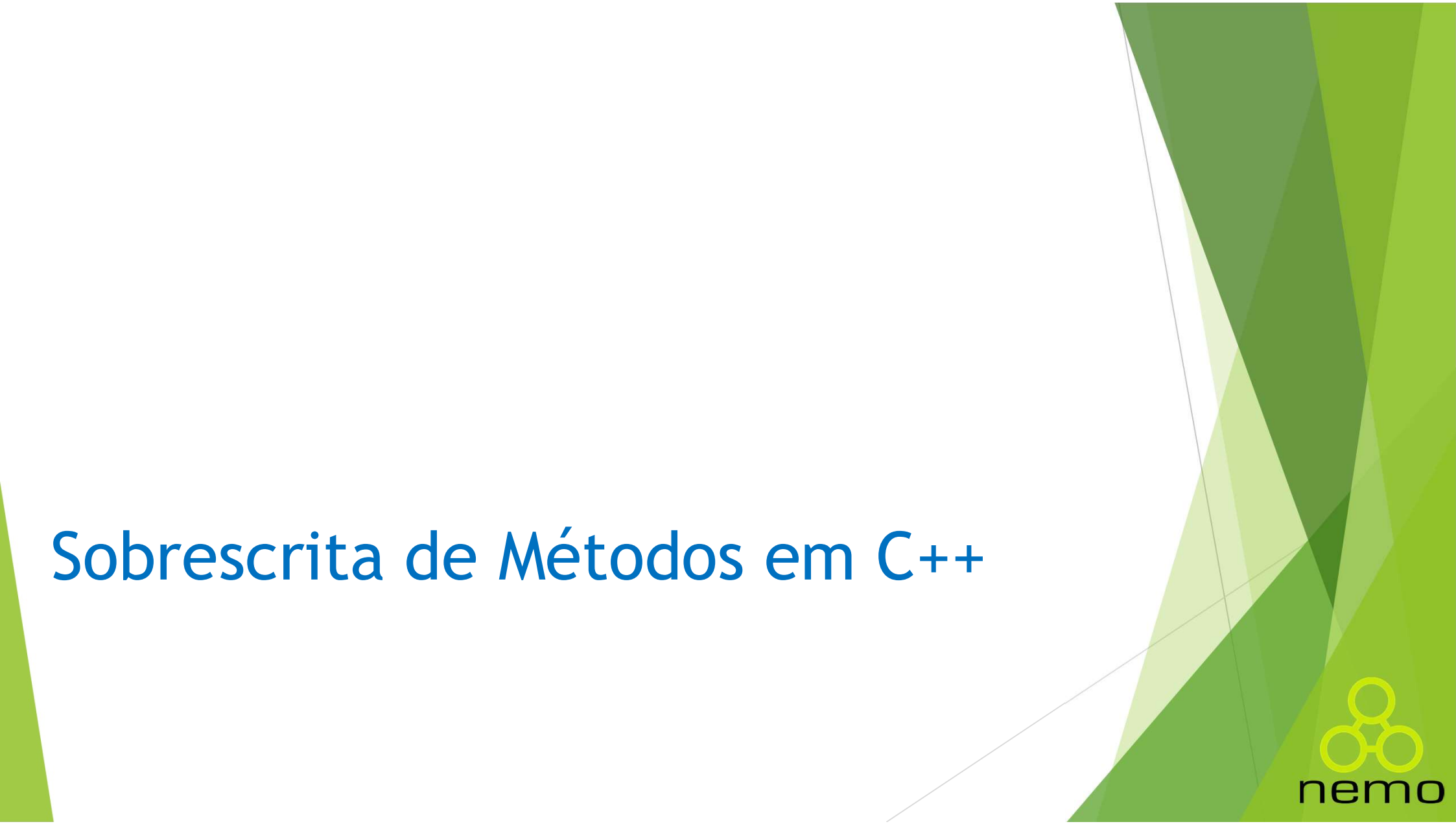
```
class Salario{
    protected:
        double valorBruto;
        double valorLiquido;
        double descontos;
    public:
        Salario(){ //construtor 1
        }

        Salario(double _valorBruto, double _valorLiquido, double _descontos){ //construtor 2
            this->valorBruto = _valorBruto;
            this->valorLiquido = _valorLiquido;
            this->descontos = _descontos;
        }

        Salario(double _valorBruto, double _valorLiquido){ // construtor 3
            this->valorBruto = _valorBruto;
            this->valorLiquido = _valorLiquido;
            this->descontos = _valorBruto - _valorLiquido;
        }
};

int main(){
    Salario *s1, *s2, *s3;
    s1 = new Salario(); //construtor 1
    s2 = new Salario(3000.5, 2500.5, 500.0); //construtor 2
    s3 = new Salario(3100.54, 2665.28); // construtor 3

}
```

nemo

Informações – Sobrecarga de Métodos

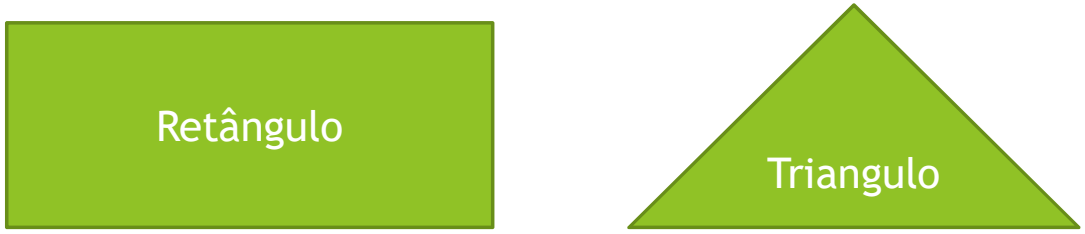
- É possível criar múltiplas definições de uma mesma função no escopo da classe.
- A definição da função precisa ser única nos tipos de parâmetros e/ou no numero de parâmetros.
- **NÃO É POSSÍVEL SOBRECARREGAR UMA FUNÇÃO APENAS MUDANDO O RETORNO.**
- Quando uma função sobrecarregada é chamada, o compilador determina a definição mais apropriada a ser utilizado.
- Ele faz isso através da comparação dos tipos de parâmetros que você utilizou para chamar a função.

Sobrescrita de Métodos em C++



Polimorfismo – Sobrescrita de Métodos

- A sobrescrita de métodos ocorre quando existem hierarquia entre classes e as classes derivadas herdam os métodos da classe, porém desempenham um comportamento diferente.
- Suponhamos as seguintes formas geométricas:



- Em ambas formas geométricas seria possível calcular a área. Então pela generalização, é certo dizer que se triângulos e retângulos possuem a funcionalidade de calcular área, eles herdam essa funcionalidade da forma geométrica.

nemo

Polimorfismo – Sobrescrita de Métodos

```
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;
        }
};
```

```
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;
        }
};
```

Polimorfismo – Sobrescrita de Métodos

- Qual o problema desse tipo de implementação?
- Não conseguimos utilizar o real poder do polimorfismo, que é chamar o método independente do tipo que a classe derivada pertence e realmente as classes derivadas redefinirem o conteúdo do método.
- Veja como esses métodos são chamada na main()

```
int main(){
    Retangulo *r;
    Triangulo *t;
    float area;

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


    t = new Triangulo(10,5);
    area = t->area();
    cout << "Valor area: " << area << endl;

    return 0;
}
```

nemo

Funções Virtuais

- Para resolver o problema anterior e permitir chamar o método independente do tipo da classe derivada, podemos transformar o método da classe base em **VIRTUAL**.
- Para isso basta adicionar a palavra reservada **virtual**, antes da funções.
- Deste modo o método da classe base jamais será instanciado.

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

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

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


    fg = new Triangulo(10,5);
    area = fg->area();
    cout << "Valor area: " << area << endl;

    return 0;
}
```

nemo

Funções Virtuais Puras

- Uma vez que a função virtual jamais será instanciada, ela não precisa possuir um corpo, apenas o retorno, que é obrigatório.

```
virtual float area(){
    return 0;
}
```

- Mas a linguagem permite que especifiquemos para o compilador que a função não possui corpo e nunca será utilizada. Chamamos essas funções de **funções virtuais puras**.

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

- O código "= 0" indica ao compilador que essa função não possui um corpo e não será utilizada, ou seja, é uma **função virtual pura**.

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

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

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

That’s all Folks!