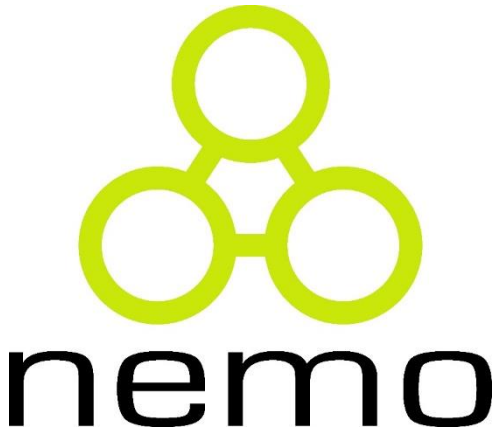


Programação III

Jordana S. Salamon

jssalamon@inf.ufes.br

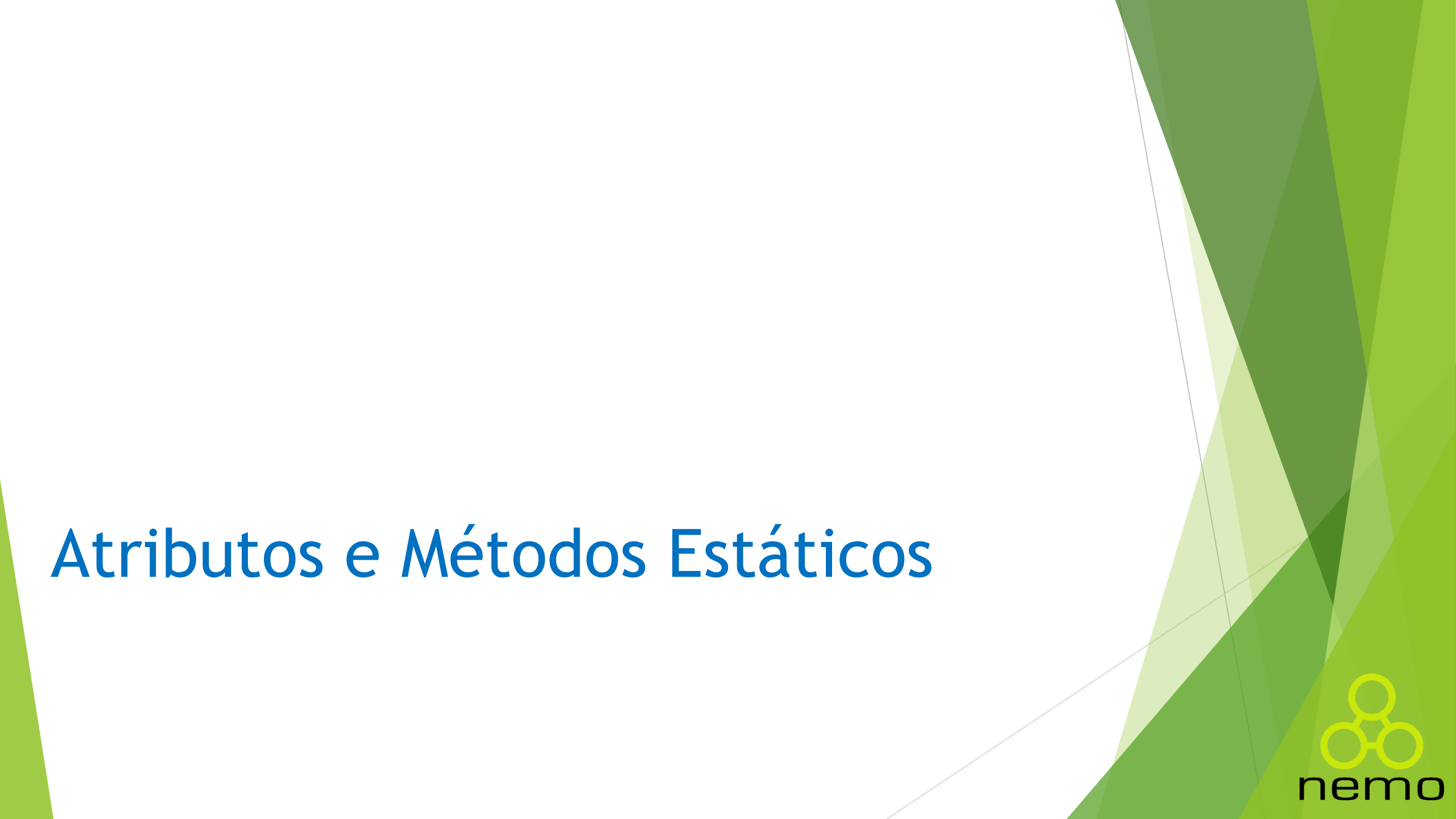
jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Atributos e Métodos Estáticos



Atributos Estáticos

- Atributos estáticos são atributos que armazenam informações independente da instanciação da classe.
- O compilador reserva um espaço de memória separado para esse atributo.
- Quando definimos um atributo estático (utilizando a palavra reservada **static**) isso significa que esse atributo é um atributo da classe e não dos objetos.
- Em outras palavras, independente do número de objetos que são instanciados, todos compartilham o mesmo atributo estático.
- O atributo estático deve ser inicializado com um valor inicial (no caso de inteiro, por exemplo, o valor 0), caso não exista outra inicialização presente, e sempre que é alterado, a alteração é refletida para todos os outros objetos que instanciam aquela classe.

nemo

Atributos Estáticos

- Não podemos inicializar esse atributo dentro da definição da classe, mas podemos inicializá-la fora da classe, usando o operador de escopo :: para identificar a classe acima.

```
class Veiculo{
    private:
        string nome;

    public:
        static int objectCout;

        Veiculo(string _nome){
            this->nome = _nome;
            objectCout++;
        }

        ~Veiculo(){
            objectCout--;
        }
};
```

```
int Veiculo::objectCout = 0;

int main()
{
    Veiculo *v1, *v2, *v3;
    cout << "Total objetos: " << Veiculo::objectCout << endl;
    v1 = new Veiculo("Corsa");
    v2 = new Veiculo("Fusca");
    v3 = new Veiculo("Brasilia");
    Veiculo::objectCout = 5;
    cout << "Total objetos: " << Veiculo::objectCout << endl;
    delete (v1);
    cout << "Total objetos: " << Veiculo::objectCout << endl;

    return 0;
}
```

nemo

Métodos Estáticos

- Métodos estáticos são funções que possuem um comportamento comum a uma classe e independente de um objeto em particular.
- Um método estático pode ser chamado mesmo se nenhum objeto for instanciado.
- Quando definimos um método estático (utilizando a palavra reservada **static**) isso significa que esse método é um método da classe e não dos objetos.
- Eles são acessado apenas utilizando o nome da classe e o operador de escopo (**::**).
- Métodos estáticas podem acessar apenas atributos e métodos estáticos de classes.
- Uma vez que métodos estáticos são independente de objetos, eles não tem acesso ao ponteiro **this.**

Métodos Estáticos

```
class Calculadora{
    public:
        static int ultimoResultado;

        static int soma (int n1, int n2){
            ultimoResultado = n1 + n2;
            return ultimoResultado;
        }
        static int subtracao (int n1, int n2){
            ultimoResultado = n1 - n2;
            return ultimoResultado;
        }
};
```

```
int Calculadora::ultimoResultado = 0;

int main(){
    cout << Calculadora::soma (2, 2) << endl;
    cout << Calculadora::ultimoResultado << endl;
    cout << Calculadora::subtracao (2, 2) << endl;
    cout << Calculadora::ultimoResultado << endl;

    return 0;
}
```

nemo

Relacionamentos entre classes

Tipos de Relacionamento

Associação


```
class Aluno{
    public:
        string nome;
        string matricula;
        vector<Professor *> professores;
};

class Professor{
    public:
        string nome;
        string siape;
        vector<Aluno *> alunos;
};
```

Um aluno pode ter vários professores e um professor pode ter vários alunos. **Um não depende do outro para existir**. Professores podem existir sem alunos e alunos podem existir sem professores.

nemo

Tipos de Relacionamento

Composição


```
class Arvore{
    int idade;
};

class Floresta{
    vector <Arvore *> arvores;
};
```

Toda vez que dizemos que a relação entre duas classe é de **composição** estamos dizendo que uma dessas classe (a Parte) está contida na outra (o Todo) e a parte não vive/não existe sem o todo.

Sendo assim, toda vez que destruirmos o todo, a parte que é única e exclusiva do todo se vai junto. Por esse motivo que algum dizem que: a parte **está contida** no todo. Quando se jogo o todo fora, a parte estava dentro e se vai junto.

Tipos de Relacionamento

Agregação

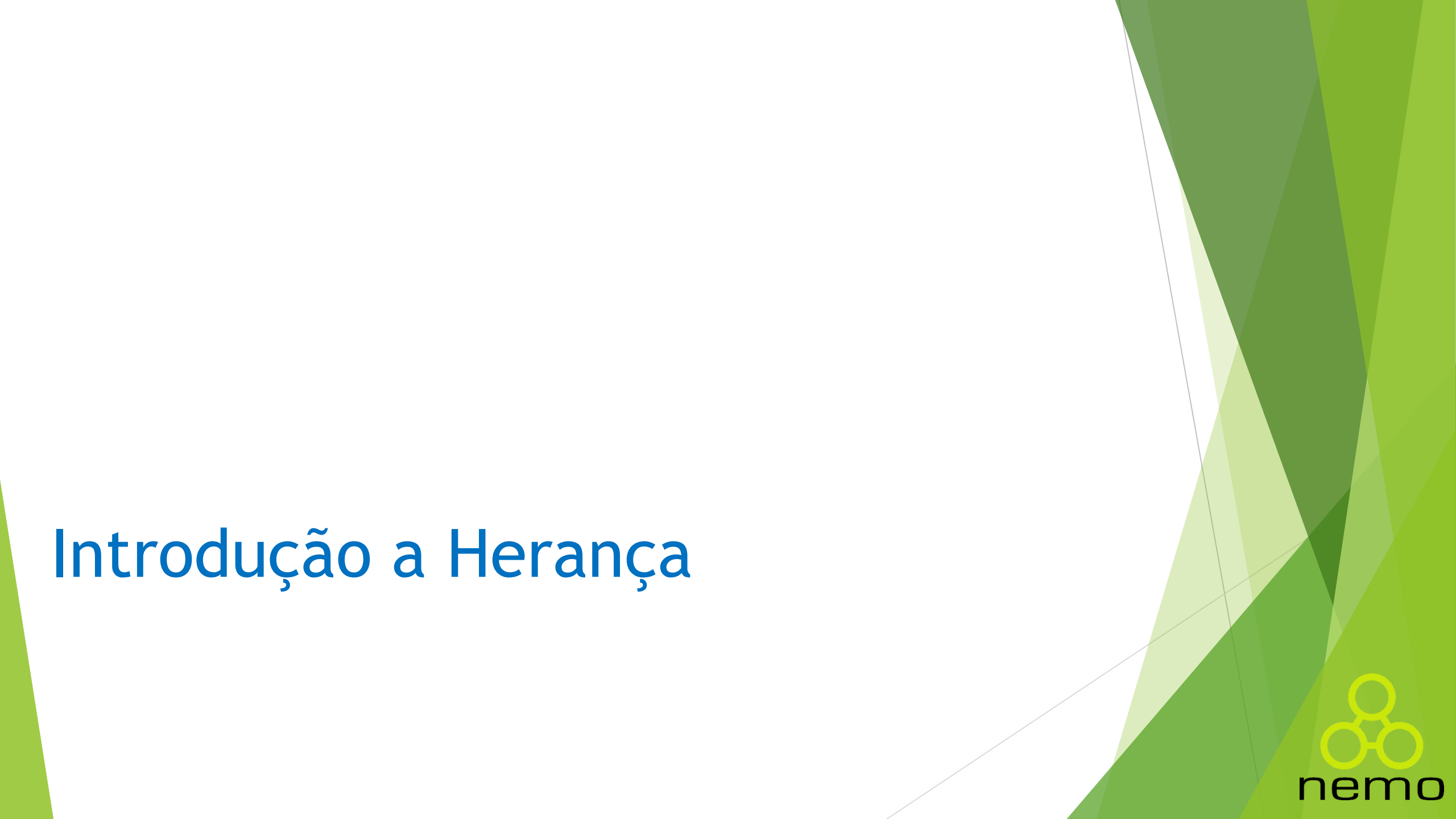

```
class Musico{
    string nome;
};

class Banda{
    vector <Musico *> musicos;
};
```

Essa também é uma relação todo/parte, dizemos que o todo precisa de uma parte para existir, porém, essa parte não necessariamente precisa ser única, essa parte pode ser substituída por outra e até mesmo compor outro todo.

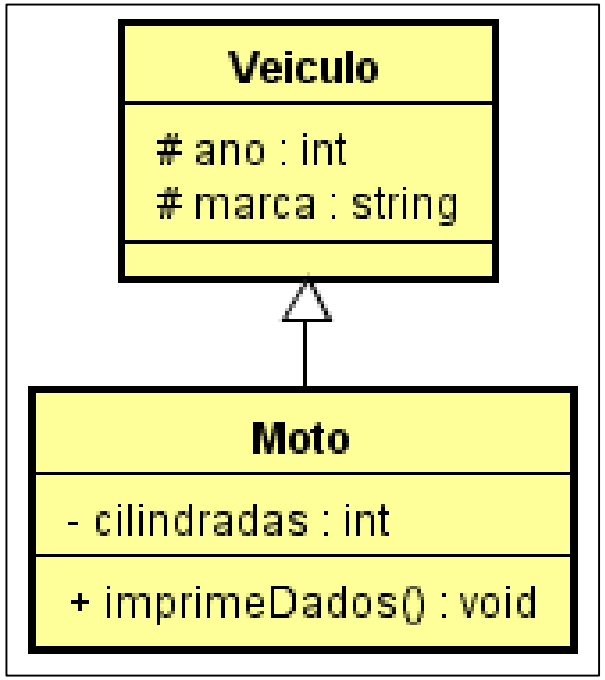
Isso significa que a parte de um tipo **A** está contido em um tipo **B**, quando esse tem relação de agregação entre eles, porém, essa mesma parte **A** pode ser substituída por **A'** e **A** pode compor outro **B'**.

Introdução a Herança



Herança - Introdução

- A classe mais geral em uma relação de herança é denominada classe **BASE** e a classe que estende a classe base é denominada **DERIVADA**.
- Uma herança implementa o relacionamento **É UM**. Exemplo: Cachorro é um Animal, Cliente é uma Pessoa, Moto é um Veiculo.




```
class Veiculo{
    protected:
        int ano;
        string marca;
};

class Moto:Veiculo{
    private:
        int cilindradas;

    public:
        void imprimeDados(){
            cout << ano << marca << this->cilindradas;
        }
};
```

Definindo Heranças

- A estrutura para criação de uma herança segue o seguinte padrão:

```
class derived-class: access-specifier base-class
```

- **Derived-class**: nome da classe derivada;
- **Base-class**: nome da classe base;
- **Access-specifier**: modificador de acesso (public, protected ou private) que define o tipo de herança.

- Obs: A classe base precisa ser definida antes da classe derivada, senão o compilador não consegue encontra-la.

Herdando Membros da Classe Base

- A **classe derivada** pode acessar todos os membros (atributos e funções) não privados da classe base.

- Desta forma caso existam membros que a **classe derivada** não deve ter conhecimento basta coloca-los como privado na **classe base**.

- A **classe derivada** herda todos os métodos da classe base, com exceção dos seguintes:
  - Construtores e Destruidores
  - Funções Amigas

Modificadores de Acesso Protected

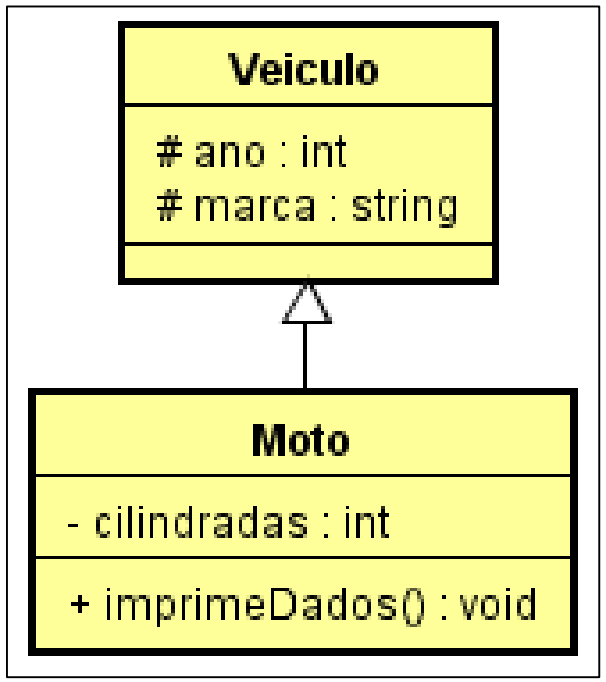
Modificador de Acesso a Classes

- Nos permite evitar que funções de um programa acesse diretamente a representação interna de uma classe.
- A restrição de acesso para atributos e métodos de uma classe é especificada pela palavras reservadas:
  - **Public:** um atributo ou método é acessível em qualquer lugar fora da classe, dentro do programa.
  - **Private:** um atributo ou método não pode ser acessado ou sequer visto de fora da classe. Apenas a própria classe e funções amigas (vamos explicar isso mais pra frente) podem acessar atributos ou métodos privados.
  - **Protected:** é bem similar a restrição de acesso **private** mas ele provê um beneficio adicional de permitir que classes derivadas possam acessar a classe base (conceito de herança).

nemo

Modificador de Acesso - Protected

- Neste exemplo os atributos ano e marca são acessíveis pela classe derivada Moto (classe derivada), uma vez que estão com modificador de acesso protegido na classe Veiculo (classe base).



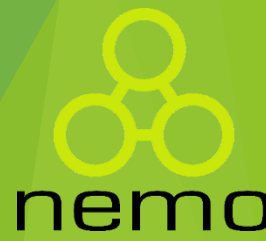

```
class Veiculo{
    protected:
        int ano;
        string marca;
};

class Moto:Veiculo{
    private:
        int cilindradas;

    public:
        void imprimeDados(){
            cout << ano << marca << this->cilindradas;
        }
};
```

nemo

Especificadores de Acesso em Heranças



Especificadores de Acesso em Heranças

- Quando uma classe é derivada de uma classe base, essa herança pode ser dos tipos **public, protected** ou **private**.
  - **Herança Pública**
    - Membros Públicos da **classe base** se tornam membros públicos na **classe derivada**.
    - Membros Protegidos da **classe base** se tornam membros protegidos na **classe derivada.**
    - Membros Privados da **classe base** nunca serão acessados diretamente pela **classe derivada**, mas podem ser acessados por membros públicos e protegidos da **classe base**.

Especificadores de Acesso em Heranças

- Quando uma classe é derivada de uma classe base, essa herança pode ser dos tipos **public, protected** ou **private**.
  - **Herança Protegida**
    - Membros Públicos e Protegidos da **classe base** tornam-se membros protegidos na **classe derivada**.
  - **Herança Privada**
    - Membros Públicos e Protegidos da **classe base** tornam-se membros privados na **classe derivada**.

Exercício

- Sobre o cenário abaixo, indique como ficariam os membros da **classes bases** nas **classes derivadas** (indicando o tipo de acesso) de acordo com o tipo de herança que está sendo realizada.

```
class MeioTransporte{
    public:
        int anoFabricacao;
    private:
        void imprimeAno(){}
};

class MeioTransporteTerrestre: private MeioTransporte{
    protected:
        int numEixos;
    public:
        void imprimeNumEixos(){}
        ~MeioTransporteTerrestre(){}
};

class Veiculo: public MeioTransporteTerrestre{
    protected:
        string placa;
        void imprimePlaca(){}
    public:
        Veiculo(){}
};
```

```
class Carro: protected Veiculo{
    private:
        string marca;
        string modelo;
    protected:
        void imprimeMarcaModelo(){}
};

class CarroSport : private Carro {
    private:
        int potencia;
    public:
        void imprimePotencia(){}
};
```

nemo

That’s all Folks!