Linguagem de Programação II

Prof.Antonio Carlos Sobieranski

DEC7532 | ENC | DEC | CTS

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Orientação à Objetos

Herança em POO

- Nesta Seção: Herança
  - Como a Herança se encaixa na OO ?
  - Definição de uma nova classe a partir de uma já existente
  - Uso da keyword *protected* na OO
  - Classes friend's
  - Funções Virtuais e como usá-las
  - Classes abstratas
  - Destrutores virtuais e onde utilizar
  - Herança múltipla

Orientação à Objetos

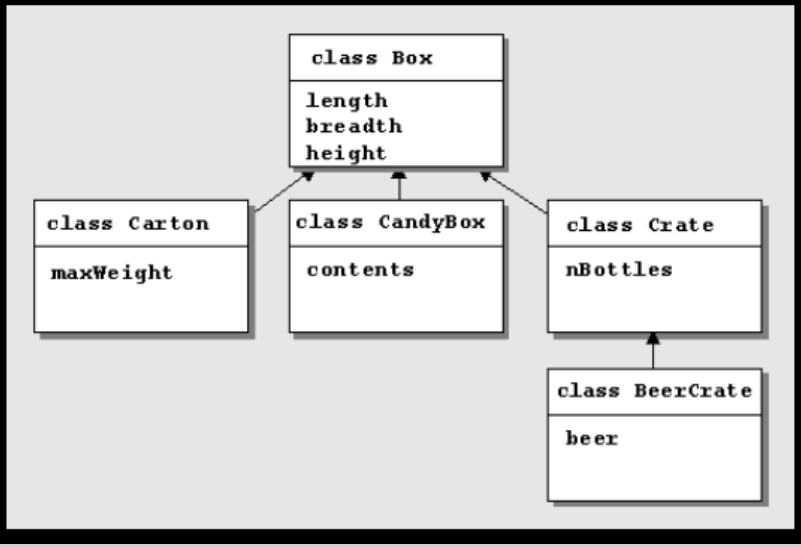
Herança em POO

- Herança
  - Considerado um dos principais mecanismos da orientação a objetos
  - Permite estender níveis de classes hierarquicamente, curva inversa em generalidade x especificidade
  - Reúso de código também se dá por herança

Orientação à Objetos

Herança em POO

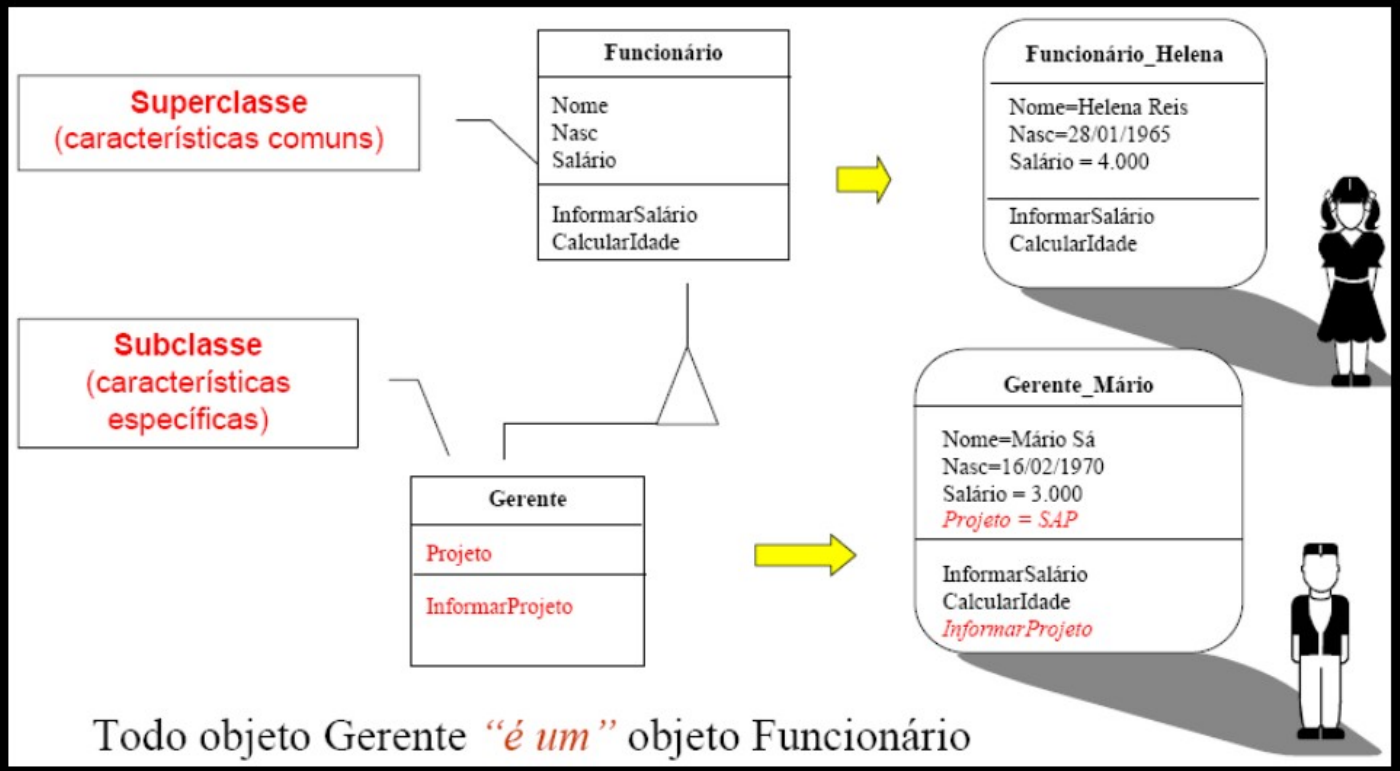
- Classes x Hierarquia
  - Classes: tipos de dados que podem representar objetos abstrados → objetos do mundo real
  - Hierarquia: classes de uma mesma família, que compartilham hierarquicamente propriedades



Orientação à Objetos

Herança em POO

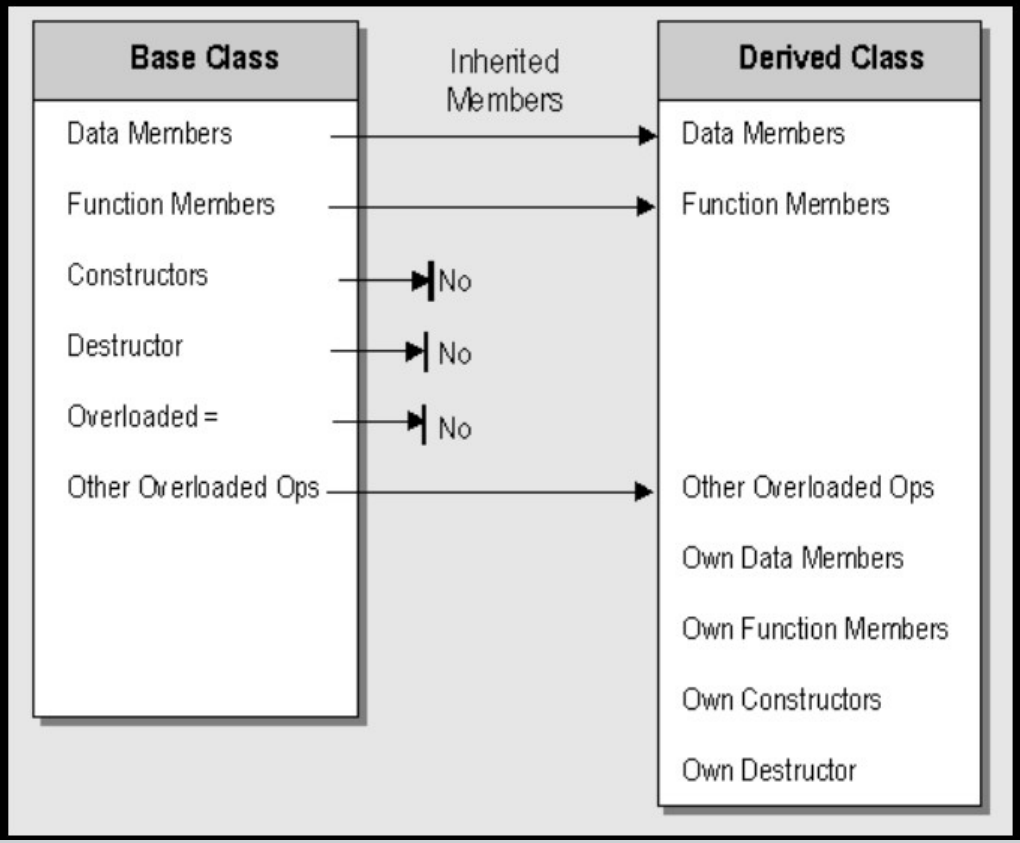
Superclasse (classe base) e Subclasse (classe derivada)



Orientação à Objetos

Herança em POO

Relação entre classe base e derivada:



Orientação à Objetos

Exemplo #01

```
class Box
{
private:
    float m_width, m_height, m_depth;

public:
    Box( ) : m_width(1), m_height(1), m_depth(1) {};
    Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
    Box(const Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) {};

    ~Box( ) {};

    void SetDimensions(float width, float height, float depth){
        m_width=width;
        m_height=height;
        m_depth=depth;
    }
};
```

```
int main( )
{
    Box a;
    CandyBox b;

    a.SetDimensions(10,10,10);
    b.SetDimensions(10,10,10);
    b.SetName("doces deliciosos da Vovó");

    return 0;
}
```

```
class CandyBox : public Box
{
private:
    string m_contents;

public:
    CandyBox( ) : m_contents("") {};
    CandyBox(string boxname) : m_contents(boxname) {};
    ~CandyBox( ) {};

    void SetName(string boxname){
        m_contents=boxname;
    }
};
```

Orientação à Objetos

Uso de qualificadores nos atributos das classes

Acessar atributos da classe base

Protected

```
class Box
{
private:
    float m_width, m_height, m_depth;

public:
    Box( ) : m_width(1), m_height(1), m_depth(1) {};
    Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
    Box(const Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) {};

    ~Box( ) {};

    void SetDimensions(float width, float height, float depth){
        m_width=width;
        m_height=height;
        m_depth=depth;
    }
};
```

```
class CandyBox : public Box
{
private:
    string m_contents;

public:
    CandyBox( ) : m_contents("") {};
    CandyBox(string boxname) : m_contents(boxname) {};
    ~CandyBox( ) {};

    void SetName(string boxname){
        m_contents=boxname;
        m_width=100;
    }
};
```

Private: somente para a classe que implementa

Orientação à Objetos

Uso de qualificadores nos atributos das classes

Acessar atributos da classe base

Logo, na herança, *protected* **deve** ser utilizando quando há a necessidade das sub-classes terem acesso direto p/ membros

```
class Box
{
protected:
    float m_width, m_height, m_depth;

public:
    Box() : m_width(1), m_height(1), m_depth(1) {};
    Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
    Box(const Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) {};

    ~Box() {};

    void SetDimensions(float width, float height, float depth){
        m_width=width;
        m_height=height;
        m_depth=depth;
    }
};
```

```
class CandyBox : public Box
{
private:
    string m_contents;

public:
    CandyBox() : m_contents("") {};
    CandyBox(string boxname) : m_contents(boxname) {};
    ~CandyBox() {};

    void SetName(string boxname){
        m_contents=boxname;
        m_width=100;
    }
};
```

Orientação à Objetos

Uso de qualificadores nos atributos das classes

Acessar atributos da classe base

- A classe derivada tem acesso a todos os membros da sua classe base que não são do tipo **private.**
- Por isso os membros da casse base que não devem ser acessados por objetos da classe derivada devem ser declarados com **private**.

Acesso	public	protected	private
Mesma classe	sim	sim	sim
Classe derivada	sim	sim	não
Outras classes	sim	não	não

Orientação à Objetos

Sobrecarga na Herança

Interessante que ambas as classes (Box e CandyBox) tenham um método ShowMe(), cada um implementado especificamente

```
a.ShowMe();
b.ShowMe();
```

```
void Box::ShowMe()
{
    std::cout << "Dimensions are " << m_width << "-" << m_height << "-" << m_depth << std::endl;
}
```

```
void CandyBox::ShowMe()
{
    std::cout << "Dimensions are " << m_width << "-" << m_height << "-" << m_depth << " and the
    content is " << m_content << std::endl;
}
```

Orientação à Objetos

Sobrecarga na Herança

Chamando o ShowMe() da super classe (explicitamente):

```
a.ShowMe();
b.ShowMe();
```

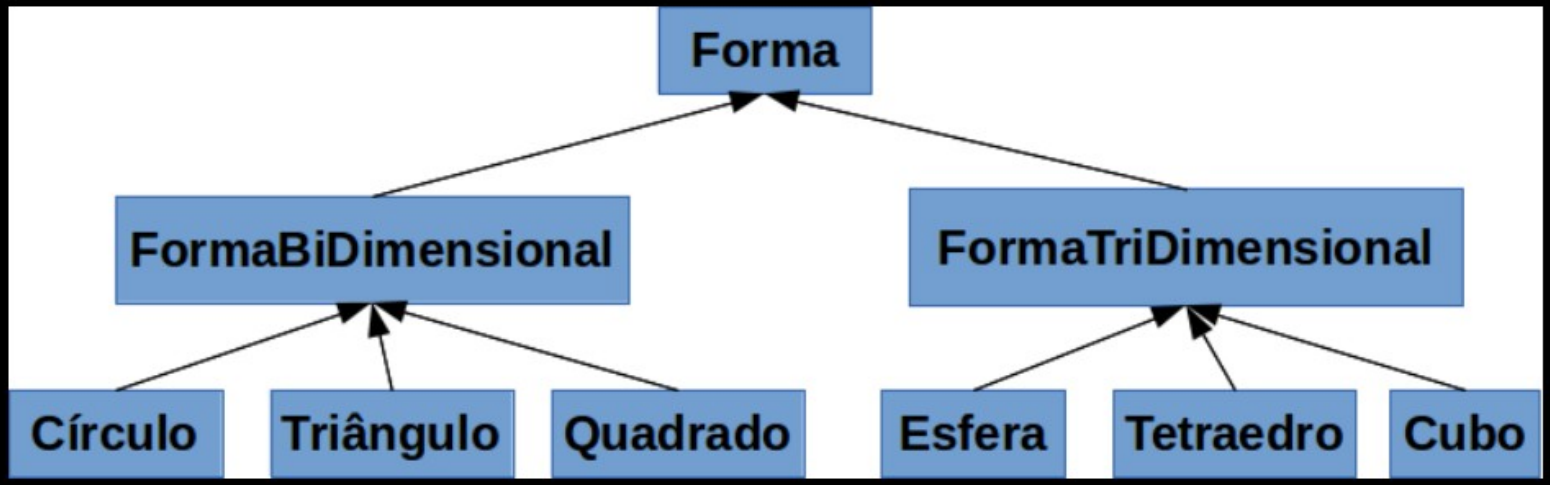
```
void CandyBox::ShowMe()
{
    std::cout << "Dimensions are " << m_width << "-" << m_height << "-" << m_depth << " and the
    content is " << m_content << std::endl;

    Box::ShowMe();
}
```

Orientação à Objetos

Exercício #01

Exercícios – Implementação 1 – Herança

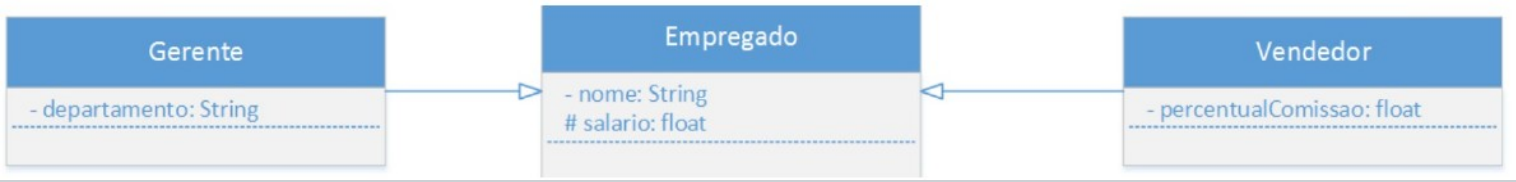
- O que são os atributos e métodos que devem ser implementados nas super-classes ?
- E quais são pertinentes às sub-classes ?

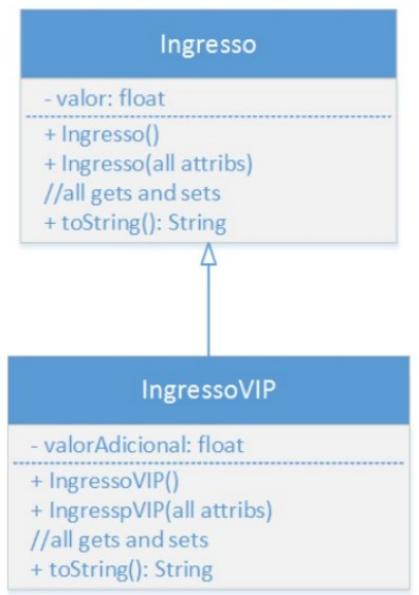


Orientação à Objetos

Exercício #02

Ver lista no Moodle, com 4 enunciados





Orientação à Objetos

Uso de qualificadores na herança da classe

Controle de Acesso na herança da sub-classe (níveis inferiores)

O C++ oferece três tipos de herança:

- **public** (a mais utilizada)
- **protected** (é raramente utilizada)
- **private** (pode ser utilizada como uma alternativa à composição)

```
class CandyBox : public Box
{
private:
    string m_contents;

public:
    CandyBox() : m_contents("") {};
    CandyBox(string boxname) : m_contents(boxname) {};
    ~CandyBox() {};

    void SetName(string boxname){
         m_contents=boxname;
    }
};
```

Orientação à Objetos

Uso de qualificadores na herança da classe

Controle de Acesso na herança da sub-classe (níveis inferiores)

- Ao derivar de uma classe base usando herança **public,** os membros **public** e **protected** da classe base não se alteram.
- Ao derivar de uma classe base usando herança **protected,** os membros **public** da classe base tornam-se os membros **protected** da classe derivada.
- Ao derivar de uma classe base usando herança **private** os membros **public** e **protected** da classe base tornam-se os membros **private** da classe derivada.

Orientação à Objetos

Uso de qualificadores na herança da classe

Controle de Acesso na herança da sub-classe (níveis inferiores)

Especificador de acesso de membro de classe básica	Tipo de herança		
	public	**protected**	**private**
public	**public** na classe derivada. Pode ser acessada diretamente por funções-membro, funções **friend** e funções não-membro.	**protected** na classe derivada. Pode ser acessada diretamente por funções-membro e funções **friend.**	**private** na classe derivada. Pode ser acessada diretamente por funções-membro e funções **friend.**
protected	**protected** na classe derivada. Pode ser acessada diretamente por funções-membro e funções **friend.**	**protected** na classe derivada. Pode ser acessada diretamente por funções-membro e funções **friend.**	**private** na classe derivada. Pode ser acessada diretamente por funções-membro e funções **friend.**
private	**private (oculta)** na classe derivada. Pode ser acessada por funções-membro e funções **friend** por meio das funções-membro **public** ou **protected** da classe básica.	**private (oculta)** na classe derivada. Pode ser acessada por funções-membro e funções **friend** por meio das funções-membro **public** ou **protected** da classe básica.	**private (oculta)** na classe derivada. Pode ser acessada por funções-membro e funções **friend** por meio das funções-membro **public** ou **protected** da classe básica.

Orientação à Objetos

Uso de qualificadores na herança da classe

Uso do qualificador *public*, *protected*, *private* na classe derivada

```
class Box
{
private:
    float m_width, m_height, m_depth;

public:
    Box() : m_width(1), m_height(1), m_depth(1) {};
    Box(float width, float height, float depth) : m_width(width), m_height(height), m_depth(depth) {};
    Box(const Box& b) : m_width(b.m_width), m_height(b.m_height), m_depth(b.m_depth) {};

    ~Box() {};

    void SetDimensions(float width, float height, float depth){
        m_width=width;
        m_height=height;
        m_depth=depth;
    }
};
```

```
int main()
{
    Box a;
    CandyBox b;

    a.SetDimensions(10,10,10);
    b.SetDimensions(10,10,10);
    b.SetName("doces deliciosos da Vovó");

    return 0;
}
```

```
class CandyBox : public Box
{
private:
    string m_contents;

public:
    CandyBox() : m_contents("") {};
    CandyBox(string boxname) : m_contents(boxname) {};
    ~CandyBox() {};

    void SetName(string boxname){
        m_contents=boxname;
    }
};
```

Remover novamente e testar

Contato

Prof.Antonio Carlos Sobieranski – DEC | A316

E-mail: a.sobieranski@ufsc.br

Inst: @antonio.sob

UNIVERSIDADE FEDERAL
DE SANTA CATARINA