UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Fundamentos de Programação

ET43I

Orientada a Objetos

Herança. Polimorfismo. Classes abstratas e interfaces.

Aula 6
Prof. Daniel Cavalcanti Jeronymo

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia Eletrônica – 3º Período
2016.1

Plano de Aula

- Herança

- Polimorfismo

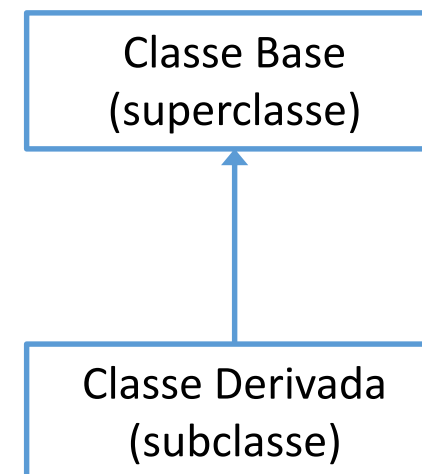
- Classes abstratas e Interfaces

Herança

- Permite que novas classes extendam classes já existentes
- A classe “filha” herda membros (atributos e métodos) da classe “pai”
- Descreve relacionamentos do tipo **é-um**

- Uso da sintaxe:

```
class classeDerivada : [modificador de acesso] classeBase
{
            //corpo da classe derivada
}
```

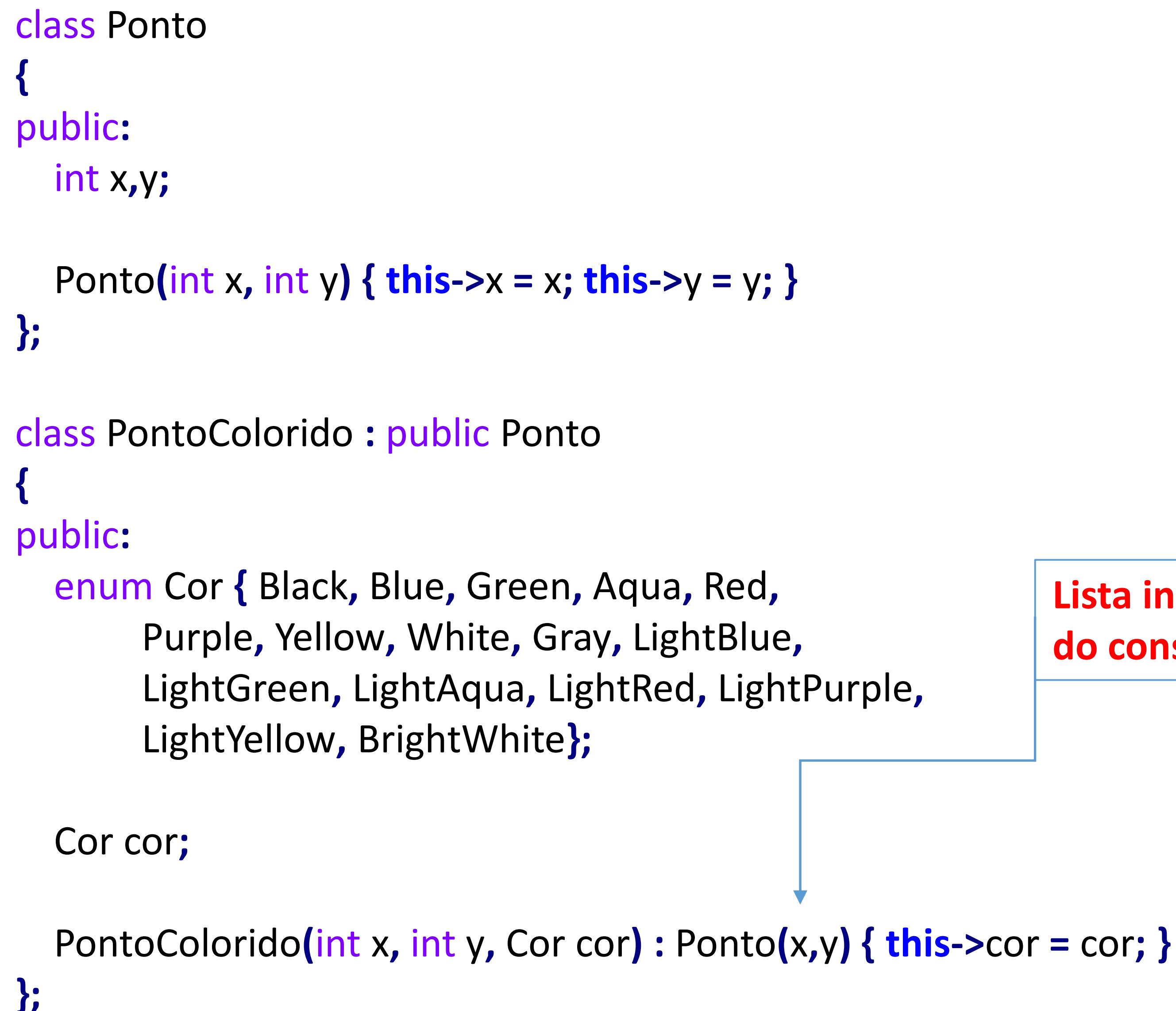


- Exemplo **Ponto**

- Um **PontoColorido** é um **Ponto** (*especializado*)

- Exemplo **Ponto**

- A inicialização da classe base deve ser realizada na **lista inicializadora** da classe derivada



- Herança: **O que não é herdado**

- Construtores, destrutores e construtores de cópia da classe base.

- Operadores sobrecarregados da classe base.

- Funções friend da classe base.

Herança

- Herança: **Construtores**

- Primeiro são executados os construtores da classe base e seus membros

- Ponto::Ponto() -> PontoColorido::PontoColorido()

- Herança: **Destrutores**

- Primeiro são executados os destrutores da classe derivada e seus membros

- PontoColorido::PontoColorido() -> Ponto::Ponto()

- As premissas anteriores quanto a ordem dos construtores e destrutores são válidas?

- Adicione *tracers* aos construtores e destrutores de **Ponto** e **PontoColorido**

Como consertar?
Próxima aula!

- Explique:

```
PontoColorido *p = new
PontoColorido(10,1,PontoColorido::Green);

cout << "P: " << p->x << " cor: " <<
PontoColorido::Green << endl;

delete p;
```

```
PontoColorido *p = new
PontoColorido(10,1,PontoColorido::Green);

cout << "P: " << p->x << " cor: " <<
PontoColorido::Green << endl;

delete static_cast<Ponto*>(p);
```

Herança

- Funções da classe base são **ocultadas** na classe derivada quando redefinidas

- As funções da classe base podem ser acessadas pelo operador de escopo ou por **cast**

Classe Ponto

```
void imprime()
{
   cout << "." << endl;
}
```

Classe PontoColorido

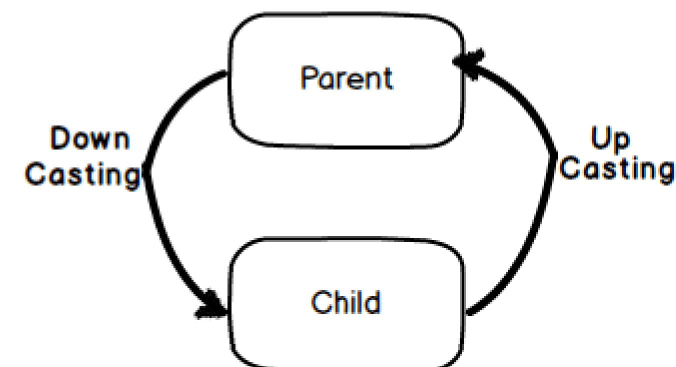
```
#include <windows.h>
void imprime()
{
   HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
   SetConsoleTextAttribute(hConsole, cor);
   cout << "." << endl;
   SetConsoleTextAttribute(hConsole, (FOREGROUND_RED |
FOREGROUND_GREEN | FOREGROUND_BLUE));
}
```

- **Upcasting:** quando o objeto é tratado de acordo com seu tipo base

- **Downcasting:** quando o objeto é tratado de acordo com seu tipo derivado

```
Ponto p1(10,1);
PontoColorido p2(10,1,PontoColorido::Green);
Ponto &rp2 = p2;

p1.imprime();
p2.imprime();
rp2.imprime();
```



Herança

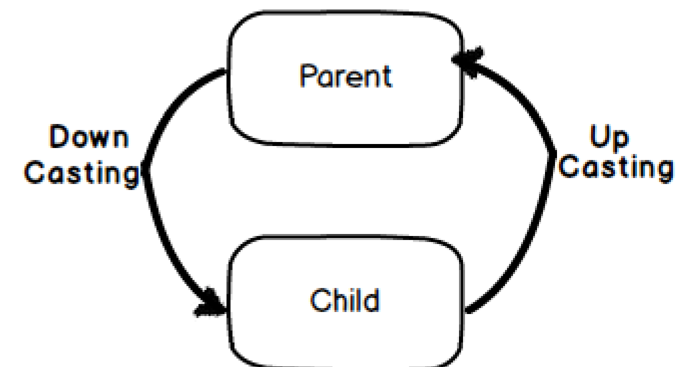
- **Downcasting:** cuidado, pode incorrer em comportamento indefinido

```
class PontoAlien : public Ponto
{
public:
   int prog;

   PontoAlien(int x, int y) : Ponto(x,y) { prog = 0xDEADC0D3; }
   void imprime()
   {
      cout << (char)178 << endl;
   }
};

   Ponto p1(10,1);
   PontoAlien p3(-1,-1);
   PontoAlien *pp3 = static_cast<PontoAlien*>(&p1);

   p1.imprime();
   pp3->imprime();
```



dynamic_cast avisa o erro

- **Controle de acesso:** redefine os acessos da classe base em função da classe derivada

```
class classeDerivada : [modificador de acesso] classeBase
{
        //corpo da classe derivada
}
```

- Pode ser definido como **public**, **private** (padrão) ou **protected**

- **Controle de acesso:** public
  - membros públicos da classe base permanecem membros públicos na classe derivada
  - membros protegidos da classe base permanecem membros protegidos na classe derivada
  - membros privados da classe base não são acessíveis na classe derivada

- **Controle de acesso:** protected
  - membros públicos da classe base tornam-se membros protegidos na classe derivada
  - membros protegidos da classe base permanecem membros protegidos na classe derivada
  - membros privados da classe base não são acessíveis na classe derivada

- **Controle de acesso:** private
  - membros públicos da classe base tornam-se membros privados na classe derivada
  - membros protegidos da classe base tornam-se membros privados na classe derivada
  - membros privados da classe base não são acessíveis na classe derivada

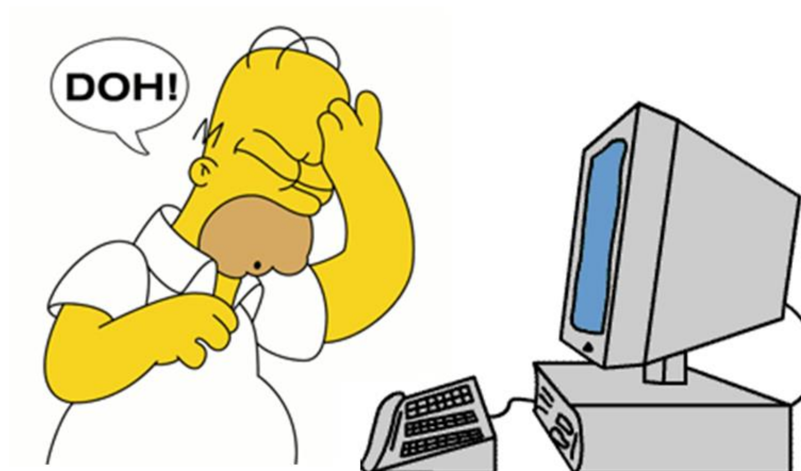
- **Controle de acesso:** resumo

Acesso na Classe Base	Especificador na Herança	Acesso Herdado
public	public	public
protected	public	protected
private	public	inacessível
public	protected	protected
protected	protected	protected
private	protected	inacessível
public	private	private
protected	private	private
private	private	inacessível

- **Controle de acesso:** upcasting é afetado pelo controle de acesso

```
class PontoColorido : private Ponto
{ ... }


PontoColorido p2(10,1,PontoColorido::Green);
Ponto &rp2 = p2; // erro!
```



Herança

- **Leitura para casa:**

- Heranças modelam relacionamentos do tipo **é-um** (*is a*)
- Composições modelam relacionamentos do tipo **tem-um** (*has a*)

- Quando usar um? Quando usar outro? Vantagens e Desvantagens?

https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose

http://www.javaworld.com/article/2076814/core-java/inheritance-versus-composition--which-one-should-you-choose-.html