ORIENTAÇÃO A OBJETOS

SISTEMAS DE INFORMAÇÃO

DR. EDNALDO B. PIZZOLATO

HERANÇA

A sintaxe de herança para C++ e para Java são diferentes.

Além disso, em C++ é possível construir herança múltipla... Em Java isso não é possível...

Vamos à sintaxe...

Em C++

```
class pessoa
{
      ...
};
class funcionario: public pessoa
{
      ...
};
```

O uso de herança private ou protected é muito raro. Consulte um manual quando for o caso. Na grande maioria das vezes, a herança será public.

Em C++

```
class pessoa
{
      ...
};
class funcionario: public pessoa
{
      ...
};
```

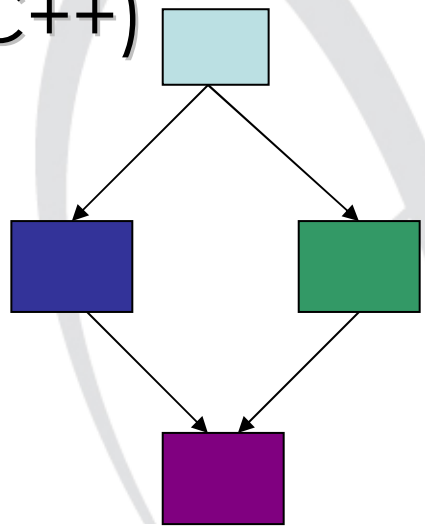
Dados **private** da classe base (pessoa) não podem ser acessados diretamente pela classe derivada (funcionario).
Isso é bom, porque não permitirá uma mudança não supervisionada dos dados.

Em C++

```
class pessoa
{
      ...
};
class funcionario: public pessoa
{
      ...
};
```

Dados **protected** da classe base (pessoa), por outro lado, podem ser acessados diretamente pela classe derivada (funcionario). Isso não é bom, porque será possível alterar os dados e comprometer a integridade dos mesmos.

Herança múltipla (somente em C++)



```
class veiculo
{   ... };
class passeio : public veiculo
{   ... };
class carga : public veiculo
{   ... };
class utilitario : public passeio, public carga
{   ... };
```

Construtores da classe base, destrutores e operadores de atribuição sobrecarregados não são herdados, mas podem ser chamados pela classe derivada.

É importante notar que o construtor da classe base é o primeiro a ser finalizado (na construção) e o último a ser destruído (na destruição).

Em Java

```
class pessoa
{
      ...
};
class funcionario extends pessoa
{
      ...
};
```

Para chamar o construtor da classe base (ou superclasse) use “super()”.
Para chamar o construtor da classe atual utilize “this()”

Como Java só faz herança simples, cada classe só pode herdar membros de uma única classe base ou superclasse. Isso cria uma hierarquia simples.

Todas as classes em Java herdam características de alguma outra classe. Se nada for especificado, a herança implícita da classe Object (do pacote java.lang) ocorre.

ESTUDO DE CASO
CLASSE PONTO E CÍRCULO
(C++)

Exemplo: Ponto/circulo

- Ponto
  - Coordenadas <x, y>
- Circulo
  - Coordenadas <x, y>
  - Raio

Programação Orientada a Objetos

```
class Ponto {

   public:
      Ponto( int = 0, int = 0 ); // construtor padrão

     void setX( int );
     int getX();

     void setY( int );
     int getY() ;

     void print();

   private:
     int x;
     int y;

   };
```

```
// Construtor padrão
  Ponto::Ponto( int Valor_x, int Valor_y )
  {
     x = Valor_x;
     y = Valor_y;

  }
```

Programação Orientada a Objetos

```
// Set x
   void Ponto::setX( int Valor_x )
   {
     x = Valor_x;
   }
// Get x
   int Ponto::getX()
   {
     return x;
    }
```

Programação Orientada a Objetos

```
// impressão
   void Ponto::print()
   {
      cout << '[' << x << ', ' << y << ']';
   }
```

Programação Orientada a Objetos

```
int main()
{
      Ponto p( 72, 115 );

      // imprime coordenadas x y
      cout << "Coord. X é " << p.getX()  << "\nCoord. Y é " << p.getY();

      p.setX( 10 ); // set x
      p.setY( 10 ); // set y

      // imprime novas coordenadas
      cout << "\n\nA nova posição é ";
      p.print();
      cout << endl;
    return 0;
 }
```

Programação Orientada a Objetos

```
class Circulo {
public:
      // construtor padrão
      Circulo( int = 0, int = 0, double = 0.0 );

      void setX( int );          // repetir para Y (setY)
      int getX();                // repetir para Y (getY)
      void setRaio( double );
      double getRaio();
      double getDiametro();
      double getCircunferencia();
      double getArea();
    void print();
private:
      int x;
      int y;
      double raio;
  };
```

Sem o uso de herança, é preciso repetir a construção de vários métodos. Além disso, precisamos repetir os atributos.

Programação Orientada a Objetos

```
class NovoCirculo : public ponto {
public:
      // construtor padrão
      NovoCirculo( int = 0, int = 0, double = 0.0 );

      void setRaio( double );
      double getRaio();
      double getDiametro();
      double getCircunferencia();
      double getArea();
      void print();

private:
      double raio;
   };
```

As características de ponto são herdadas...
NovoCirculo pode utilizar os métodos de ponto, mas não pode acessar os elementos private.

```
NovoCirculo::NovoCirculo( int Vx, int Vy, double Vraio )
  {
    x = Vx;      // Erro! Tem que usar setX(Vx)
    y = Vy;      // Erro! Tem que usar setY(Vy)
    setRaio( Vraio );
  }
```

NovoCirculo pode utilizar os métodos de ponto, mas não pode acessar os elementos private.

```
// imprime objeto
  void NovoCirculo::print()
  {
     cout << "Centro = [" << x << ", " << y << ']'
          << "; Raio = " << raio;
  }
```

NovoCirculo pode utilizar os métodos de ponto, mas não pode acessar os elementos private.

```
// imprime objeto
  void NovoCirculo::print()
  {
    cout << "Centro = [" << getX() << ", " << getY() << ']'
         << "; Raio = " << raio;
  }
```

ou

```
 void NovoCirculo::print()
 {
    cout << "Centro = ";
    Ponto::print(); // chamando função de impressão do ponto
    cout << "; Raio = " << getRaio();
 }
```

Para não confundir com chamada recursiva, a chamada do método print de ponto tem que ser identificada (com o escopo).

Conclusões:

É fácil construir classes à partir de outras tanto em C++ como em Java.

Java, porém, tem uma grande biblioteca com classes prontas para você usar. Uma rápida busca na internet permitirá que você compreenda as vantagens de se trabalhar com uma linguagem onde já existe muita coisa pronta.

FIM